HZ BOOKS

电子与电气工程丛书

WILEY

VLSI数字信号处理系统设计与实现

VLSI DIGITAL SIGNAL PROCESSING SYSTEMS

DESIGN AND IMPLEMENTATION

(美) Keshab K. Parhi 著
明尼苏达大学

陈弘毅 白国强 吴行军 等译
清华大学

Keshab K. Parhi

机械工业出版社
China Machine Press

电子与电气工程丛书

VLSI数字信号处理系统设计与实现

VLSI Digital Signal Processing Systems Design and Implementation

机械工业出版社
China Machine Press

本书详细而全面地论述VLSI信号处理中性能优化技术，汇集了VLSI的架构理论与算法，描述了硬件实现层中的各种架构，给出了若干种分析、估计与降低功耗的方法，重点讲解设计应用时所需要的定制或半定制VLSI电路。

本书适合作为高年级本科生和低年级研究生的教科书或参考书，对于专业技术人员，也是一本很好的参考书。

Keshab K. Parhi: VLSI Digital Signal Processing Systems: Design and Implementation (ISBN: 0-471-24186-5)

Copyright © 1999 by John Wiley & Sons, Inc.

All rights reserved.

AUTHORIZED TRANSLATION OF THE EDITION PUBLISHED BY JOHE WILEY & SONS, Hoboken, Chichester, Brisbane, Singapore AND Toronto. No part of this book may be reproduced in any form without the writer permission of John Wiley & Sons Inc.

本书中文简体字版由约翰-威利父子公司授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书版权登记号：图字：01-2003-7223

图书在版编目（CIP）数据

VLSI数字信号处理系统：设计与实现 /（美）帕里（Parhi, K. K.）著；陈弘毅等译. -北京：机械工业出版社，2004.6

（电子与电气工程丛书）

书名原文：VLSI Digital Signal Processing Systems: Design and Implementation

ISBN 7-111-14108-3

Ⅰ.V… Ⅱ.①帕…②陈… Ⅲ.超大规模集成电路－电路设计 Ⅳ.TN470.2

中国版本图书馆CIP数据核字（2004）第017990号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：蒋 祎

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2004年6月第1版第1次印刷

787mm × 1092mm 1/16 · 37.25印张

印数：0 001-4 000册

定价：65.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

当今信息技术领域，数字音频、语音识别、有线调制解调器、雷达、高清晰度电视等只是依赖数字信号处理（DSP）及其相应的专用集成电路（ASIC）实现的现代计算机与通信应用中很小的一部分例子。随着信息化时代的进步，工业界不断地在推出功耗更低、效率更高的ASIC芯片，而从事集成电路设计的人们对DSP超大规模集成电路（VLSI）设计方法学的需求也日益增长。本书是一部论述VLSI信号处理中性能优化技术的全面的、指导性的著作。它是凯夏博·帕里教授基于多年从事研究生课程教学的内容与经验编著的。本书出版后的短短几年，就被美、欧多所大学DSP的课程教学所采用，现已成为这一领域中的一本标准教科书或参考资料。

本书汇集了VLSI的架构理论与算法，描述了硬件实现层中的各种架构，给出了若干种分析、估计与降低功耗的方法，重点讲述设计以上应用所需要的定制或半定制VLSI电路。书中所提供的许多技术对于利用市场上可买到的可编程数字信号处理器快速实现也是适用的。

本书适于用作一年级研究生或在讲授VCSI、DSP架构，或VLSI、高性能VLSI系统设计的DSP结构等方面内容的高级课程时的教科书。本书对于那些牵涉到DSP应用算法、DSP架构或电路设计的人们也是一本很好的参考书。

本书译稿由清华大学微电子学研究所陈弘毅教授审核。参加翻译工作的人员有：陈弘毅、白国强、吴行军、王海永、林敏、刘雷波、涂春江、陈刚、孙志锐等。另外，感谢机械工业出版社华章分社在组织出版和编辑工作中所给予的支持。原书作者凯夏博·帕里对本书的翻译给予了支持与帮助，并专门为中译本写了序言。在此对凯夏博·帕里教授也表示衷心的感谢。

限于译者水平，中译本中难免有错误与不妥之处，恳请读者批评指正。

译 者

2003年12月于清华园

中文版序

得知清华大学陈弘毅教授的小组经过努力工作，已经完成了教科书“VLSI Digital Signal Processing Systems: Design and Implementation”(John Wiley and Sons, Inc., 1999)的中文翻译，我很高兴。尤其使我感到欣慰的是，从现在起，很多在中国的学生将能够阅读到这本书了。

信号处理系统的VLSI实现如今已经是一个非常基础性的课题了。按照这本书里讲述的技术设计的各种通信系统与每个人日常生活密不可分，例如千兆以太网、ADSL系统、电缆调制解调器以及其他形式的宽带系统。采用数据流变换的方法使这些系统的面积和功耗减至最小对于改善通信集成电路的性能是非常关键的。

在世界各地的中国工程师们对集成电路芯片设计已经做出了很大的贡献，这本书将帮助他们进一步提高设计水平。我非常感谢陈弘毅教授的小组在翻译这本厚书的困难任务中所做的非同寻常的努力。他的引导一定会鼓励很多代对这一领域感兴趣的学生!

Keshab K. Parhi

2004年1月

It gives me great pleasure to learn that Prof. Hongyi Chen of Tsinghua University and his group have worked hard to bring out a Chinese translation of the text book "VLSI Digital Signal Processing Systems: Design and Implementation" (John Wiley and Sons, Inc., 1999). I am particularly pleased that many students in China will now have access to this book.

VLSI implementation of signal processing systems is now a very fundamental topic. The communications systems designed by the techniques described in this book are integral parts of every one's daily life. Examples of a few of these systems include gigabit ethernet systems, ADSL systems, cable modems, and other forms of broadband systems. Minimizing area and power consumption of these systems by data-flow transformations is critical to improving performance of the communications integrated circuits.

Chinese engineers have made significant contributions to the integrated circuit chip design all over the world. This book will go one step further in helping them improve their designs. I can only praise Prof. Hongyi Chen and his group for their extraordinary efforts in undertaking an extremely difficult task in translating a book of this size. His leadership will inspire many generations of students interested in working in this field!

作者简介

凯夏博·帕里（Keshab K. Parhi）教授分别于1982年、1984年和1988年在印度理工学院、美国宾夕法尼亚大学和加州大学伯克利分校取得电机工程专业学士、硕士和博士学位。从1988年起，Parhi 教授在明尼苏达大学从事教学与研究工作，并被评为电机与计算机工程系的McKinght荣誉教授。

他的研究领域重点在于宽带通信系统物理层方面的VLSI体系结构设计。他当前的研究包括纠错码的编/解码器与加密/解密算法的体系结构、高速发送/接收器、超宽带系统、量子纠错编/解码器与量子加密/解密。他已经发表了350多篇学术论文，撰写了教科书“VLSI Digital Signal Processing Systems: Design and Implementation”（Wiley, 1999）。这套教材被美国、荷兰、瑞典、芬兰等国的多所大学采用。此外，他还参与了“多媒体系统的数字信号处理”（Marcel Dekker, 1999）一书的编写工作。

Parhi教授曾获得多次殊荣，包括2003年度“IEEE Kiyo Tomiyasu”技术领域奖、2001年IEEE W.R.G. Baker最佳论文奖和1999年度IEEE电路与系统协会Golden Jubilee奖。1996年，他被评选为电气电子工程师学会会士（IEEE Fellow）。历年来，他曾担任“IEEE Transactions on Circuits and Systems, Circuits and Systems-II”、“VLSI Systems”、“Signal Processing”、“Signal Processing Letters”等学术杂志的编委会成员。此外，他还担任“IEEE Trans. On Circuits and Systems-I”（2004—2005年度）的主编和“IEEE Signal Processing Magazine”的编委。Parhi教授是IEEE 1995年“VLSI信号处理研讨会”和1996年“专用系统、体系结构与处理器国际会议”(ASAP)的技术委员会联合主席，以及2002年“IEEE信号处理系统研讨会”的主席。他曾被IEEE电路与系统协会授予1996—1998年度杰出讲师的称号。

译者简介

陈弘毅，男，江苏省徐州市人，1942年出生于重庆市。教授，博士生导师，清华大学信息学院学术委员会副主任、微电子学研究所学术委员会主任，中国电子学会高级会员。多年来从事集成电路（IC）设计研究，包括：基于门阵列（GA）、标准单元（SC）和功能块（BB）的专用集成电路（ASIC）设计方法；库开发技术；系统的芯片集成（System on-a-Chip）；算法的硬件架构与实现；超大规模集成电路-数字信号处理（VLSI-DSP）及其在多媒体信号（语言、音频、图像、视频)处理和信息安全领域的应用等等。

发表学术论文170余篇；参与出版专著两部（《集成电路工业全书》电子工业出版社，1993年；《ULSI器件、电路与系统》科学出版社，2000年）；获得国家科技进步二等奖一项、北京市科技进步一等奖和二等奖各一项；持有美国发明专利13项；已培养出硕士10名、博士13名。

白国强，男，陕西省清涧县人。1963年11月生。清华大学微电子学研究所副教授。2000年12月获西安电子科技大学军事指挥学密码学专业博士学位。从2001年5起在清华大学电子科学与技术博士后科研流动站做博士后研究，2003年7月出站后留校工作。主要研究领域包括应用数学、编码密码学和密码算法的芯片集成技术等。在国内外重要刊物已发表论文20多篇，主编《概率论与数理统计》(陕西人民教育出版社，2000年2月)、参编《信息工程概论》(科学出版社，2002年8月）教材各一本。曾参加国家自然科学基金项目、“97-3”国家重点基础科学研究项目等国家级和省部级科研项目多项。获国防科学技术成果二等奖一次，陕西省高等学校科技进步一等奖一次。2002年9月至2003年12月主持完成国家“十五”期间高技术研究发展计划（863计划）课题一项，组织完成了我国第一块椭圆曲线密码芯片的研制。目前承担国家自然科学基金项目一项。

吴行军，男，副教授。湖南省平江县人。1969年2月出生，分别于1992年6月和1994年12月获清华大学工学学士学位和硕士学位。从1994年起至今在清华大学微电子学研究所工作，期间曾任集成电路与系统集成设计研究室主任。1997年至2003年期间做为主要人员曾参与我国第二代身份证芯片的研发工作。发表论文10余篇。目前主要研究方向为信息安全类集成电路和智能IC卡集成电路研制。

前　言

数字信号处理（Digital Signal Processing, DSP）具有极其广泛的应用，例如视频压缩（Video Compression）、数字机顶盒（Digital Set-Top Box, STB）、有线调制解调器（Cable Modem）、数字多用盘（Digital Versatile Disk）、便携视频系统/计算机、数字音频（Digital Audio）、多媒体和无线通信、数字收音机（Digital Radio）、数字静止图像照相机（Digital Still Camera, DSC）和网络照相机、语音处理、传输系统、雷达成像（Radar Imaging）、声学波束成形器（Acoustic Beamformer）、全球定位系统（Global Positioning System, GPS）和生物医学信号处理，等等。DSP领域始终是由DSP应用和超大规模集成（Very-Large-Scale-Integrated, VLSI）技术的进展驱动的。因此，在任何特定的DSP应用出现时，总是随之带来对于相应DSP系统实现的挑战，这些实现必须满足实时的DSP应用所强制的采样速率约束，此外，还必须减小所占的空间与消耗的功率。

本书讲述设计上面提及的各种应用所需要的定制或半定制VLSI电路的方法学，书中所提供的许多技术也适用于利用非定制的可编程数字信号处理器来快速地实现。本书适于用作一年级研究生或高年级本科生课程的教科书，讲授VLSI DSP架构，或VLSI、高性能VLSI系统设计的DSP结构。本书对于那些牵涉到DSP应用算法、架构或电路设计的专业人员也是一本很好的参考材料。

本书把计算机架构（Computer Architecture）理论与DSP这两个不同领域联系在一起。DSP计算与通用计算有如下含义方面的区别：DSP程序是一种无休止的（Non-terminating）程序，即在DSP计算中，相同的程序在一个无限时间序列上重复地执行。通过发掘在同一迭代过程内及多重迭代过程间的任务依赖性，可以利用这种无休止的性质来设计更高效的DSP系统。而且，在DSP算法中，长的关键路径（Critical Path）限制了DSP系统的性能，需要对这样的算法进行变换，以便得到高速度、小面积或低功耗实现的设计。本书的重点放在高效的架构、算法和电路的设计方面，以获得具有较小面积或较小功耗、或者具有较高速度或较低舍入噪声的效果。但是，实际的VLSI电路设计则不在本书讨论范围之内。

DSP算法用在各种具有不同采样速率需求的实时应用中，从语音应用中的20 kHz到雷达和高清晰度电视应用中的超过500 MHz。一个高清晰度电视（High Definity Television, HDTV）用的视频压缩系统的计算需求在10GOPS至100 GOPS（每秒千兆操作）范围。对于DSP算法的实现来说，采样速率和计算需求千差万别，因此所需架构也不同。例如，在语音应用中，倾向于使用时分多路（Time-Division Multiples）的架构，许多算法操作都映射到同一硬件上来完成。然而，在视频应用中，则必须通过算法操作与处理器之间一对一的映射，才能满足高速的需求。因此，研究不仅可以设计某一种架构、而是可以设计一组架构的技术是重要的，这样能够针对特定的应用从中选择适当的架构。

本书第一部分（第2章至第7章）讨论若干高层次的架构变换，它们可以用来设计给定算法的一系列架构。这些变换包括流水线、重定时、展开、折叠和脉动阵列设计方法。第二部分（第8章至第12章）涉及高层次的算法变换，如强度缩减（Strength Reduction）、超前（Look-

ahead）或弛豫超前（Relaxed Look-ahead）。强度缩减变换在卷积、并行有限冲激响应（Finite Impulse Response, FIR）数字滤波器、离散余弦变换（Descrete Cosine Transform, DCT）和并行秩-阶滤波器（Rank-Order Filter）中用来减少乘法的次数。而超前或弛豫超前变换则用来设计流水线直接形式或格型递归数字滤波器（Lattice Recursive Digital Filter）、自适应数字滤波器（Adaptive Digital Filter）和并行递归数字滤波器。该部分探索算法设计与集成电路实现之间的相互作用。第三部分（第13章到第18章）讨论VLSI加法、乘法和数字滤波器的架构，以及与高性能VLSI系统设计有关的问题，如流水线风格、低功耗设计和可编程数字信号处理器的架构。

本书第1章综述各种DSP算法，并讨论如何利用方框图、信号流图和数据流图表示这些算法。第2章讨论迭代边界（Iterative Bound），它是任何一种递归信号处理算法迭代周期的基本下限。该章描述了两种确定迭代边界的算法。其后的五章讨论旨在改善数字信号处理实现性能的各种变换。在第3章中论述了流水线（Pipelining）和并行处理器（Parallel Processing）的基本概念，以及说明在高速应用或低功耗应用设计中如何使用这些技术。第4章讨论重定时（Retiming）变换，它是流水线方法的一种通用化描述。第5章讨论展开（Unfolding）技术，它可以用来设计并行架构。第6章和第7章讨论折叠（Folding）技术，用于在缩小面积十分重要的场合下设计时分多路架构。第6章讨论任意数据流图（Data Flow Graph, DFG）的折叠，第7章讨论基于脉动（Systolic）设计方法的规则数据流图的折叠。

第8章至第12章讨论基于算法变换的各种DSP算法的算法结构设计，如强度缩减、超前与弛豫超前、数字滤波器中的缩扩（Scaled）噪声与舍入（Roud-off）噪声。第8章讨论基于Cook-Toom（库克-图姆）和Winograd（维诺格拉德）卷积算法的快速卷积。在第9章中，算法强度缩减用在并行FIR滤波器、离散余弦变换和并行的秩-阶滤波器中来减少乘法操作次数。虽然快速傅立叶变换（FFT）结构设计也是基于强度缩减变换的，但是不包含在本书内容中，因为在很多导论性的DSP教科书中都包含了FFT的内容。在非递归计算中很容易实现流水线与并行处理，但是在递归与自适应数字滤波器中，由于存在反馈环路，不容易实现流水线和并行处理。第10章中讨论超前技术，并用于一阶无限冲激响应（Infinite Impulse Response, IIR）数字滤波器的流水线实现。对于较高阶的滤波器，讨论了两类超前技术：簇集（Cluster）超前技术和发散（Scattering）超前技术。可以看出，发散超前技术能够保证流水线IIR滤波器的稳定性。IIR数字滤波器的并行实现以及如何在这些数字滤波器中将流水线与并行处理结合起来的问题也在该章进行讨论。自适应滤波器是基于了弛豫超前技术来实现流水线的，而弛豫超前技术则是基于超前技术的某些近似或弛豫。第11章讨论缩扩噪声与舍入噪声，它们对于定点运算DSP系统的VLSI实现来说是十分重要的。舍入噪声计算技术在很多数字滤波器中无法应用，这些滤波器利用降低速度、流水线和（或）重定时等方法进行预处理，以便使每个舍入噪声节点能够表达为一个状态变量。直接形式的IIR数字滤波器在某些应用中不能满足滤波器的需求，而格型滤波器却因为其优良的舍入噪声特性能够更好地适合于这些应用。第12章介绍Schur（肖尔）多项式、Schur多项式的正交归一性，及其在设计基本的（双乘法器和单乘法器）、归一化及缩放归一化的格型数字滤波器中的应用。对如何实现这些格型数字滤波器的流水线也进行了讨论。

第13章至第18章讨论运算操作（如加法、乘法、数字滤波器）、高性能VLSI系统设计问题（如流水线风格、低功耗设计）和可编程数字信号处理器的VLSI实现。在第13章中讨论了加法器和乘法器的设计，采用了各种实现方式（如位并行、位串行和字串行）以及各种数制

（如二进制补码、正则符号数和进位保留）。该章还讨论了分布式运算。第14章讨论了基于冗余数或符号数实现的运算架构。冗余数运算的主要优点在于它的免进位特性，因此计算既能从最低位开始，也能从最高位开始。对冗余数与非冗余数之间的相互转换也进行了讨论。在这几章中，位串乘法器可以利用脉动设计方法从位并乘法器设计导出。余数运算可用于FIR数字滤波器和变换的实现，但是不在本书中研究。第15章介绍在数字级的强度缩减，用于减小基于二进制补码和正则符号数的数字滤波器的面积和功耗。第16章讨论各种流水线风格，如同步流水线、波流水线和异步流水线。该章也讨论了在同步系统中减少时钟歪斜的方法和在异步系统中接口电路的综合。关于低功耗设计的第17章介绍了用于在架构级和工艺级降低功耗和估计功耗的各种方法。第18章讨论用于可编程数字信号处理器中的各种架构。

在本书中的七个附录涵盖了用于确定迭代边界和用于重定时的最短路径算法；用于设计折叠架构、确定折叠集合的调度和分配技术；Euclidian（欧几里德）求最大公因子（Greatest Common Divisor, GCD）算法用于Winograd卷积；用于设计格型数字滤波器的Schur多项式的正交归一性；快速位并行加法与乘法；用于位串行系统的调度技术；以及FIR滤波器系统的量化。

本书中的概念均采用与工艺无关的方式来描述。本书中的例子都是基于数字滤波器和变换的。很多实时DSP系统使用了控制流的结构，如条件转移、中断和跳转等。控制优先的DSP系统设计超出了本书的范畴。练习可以采用任意的编程语言来完成，如MATLAB或C。许多与应用有关的习题都包括在各章的末尾，例如在算法强度缩减一章中末尾的习题，就涉及到设计通信系统均衡器中的快速滤波器、小波、二维FIR数字滤波器和运动估值等的使用。这些习题给读者介绍了该章涵盖的概念所能涉及到的各种应用。

本书的内容是建立在作者当前在明尼苏达大学执教的两门课基础之上的，这两门课程是EE5329：VLSI数字信号处理系统以及EE5549：数字信号处理的VLSI结构。EE5329（以VLSI设计的基础课程为先修课程）覆盖了第2章至第7章的全部内容和第13章至第18章（按此次序）的部分内容。EE5549（以数字信号处理的基础课程为先修课程）包含了第2章、第3章和第4章的部分内容、第8章至第12章的全部内容，以及基于杂志和会议论文的一些关于视频压缩架构方面的内容。这两门课在过去是作为三学季课程讲授的。对于一门关于VLSI数字信号处理的单学期课程来说，推荐采用第2章至第7章、第9、10、13和第15章的部分内容，以及第17、18章中对所论及题目的综述。当然，教师也可以选择适合他们需要的章节。

不一定要按照书中所排的章节顺序来讲授，很多章节可以独立地教授。图0-1中所给的优先图表示出各章之间的依赖关系。虚线代表当前章节对以前章节间依赖性较弱。

作者有幸得到许多同事、学生和朋友的可贵帮助、支持与建议。作者感谢LeiLei Song在我写此书期间自始至终所给予的热心帮助。作者也对Jin-Gyun Chung、Tracy Denk、David Parker、Janardhan Satyanarayana和Ching-YWang在撰写本书前面部分时给予的帮助表示感谢。作者衷心感谢Wayne Burleson、Francky Catthoor、Ed F. Deprettere、Graham Jallien和Naresh R. Shanbhag对本书初稿全面的、建设性的审阅。Ed F. Deprettere和Scott Douglas还以本书作为预备版本，分别在Delft工业大学和犹他大学进行了授课，并且提出了大量的建议。

作者感谢来自Dovid G Messerschmitt 和Mos Kaveh的一贯支持与鼓励。作者写进本书中的研究成果得到了国家科学基金会、陆军研究办公室、海军研究办公室、国防先进研究项目机构、德州仪器公司、朗迅技术公司和日本NEC公司的支持。作者衷心感谢John Cozzens、Wanda Gass、Arup Gupta、Clifford Lau、Jose Munoz、Takao Nishitani和Bill Saner给予的鼓励。

目 录

译者序
中文版序
作者简介
译者简介
前言
第1章 数字信号处理系统导论 ……1
1.1 引言 ……1
1.2 典型的DSP算法 ……1
1.2.1 卷积 ……2
1.2.2 相关 ……3
1.2.3 数字滤波器 ……3
1.2.4 自适应滤波器 ……5
1.2.5 运动估值 ……8
1.2.6 离散余弦变换 ……9
1.2.7 矢量量化 ……12
1.2.8 Viterbi算法和动态规划 ……14
1.2.9 抽取器和扩展器 ……16
1.2.10 小波和滤波器组 ……18
1.3 DSP应用需求和按比例缩小的CMOS工艺 ……20
1.4 DSP算法的表示 ……23
1.4.1 框图 ……24
1.4.2 信号流图 ……26
1.4.3 数据流图 ……27
1.4.4 依赖图 ……29
1.5 本书概要 ……29
参考文献 ……30
第2章 迭代边界 ……33
2.1 引言 ……33
2.2 数据流图表示 ……33
2.3 环路边界和迭代边界 ……34
2.4 计算迭代边界的算法 ……35
2.4.1 最长路径矩阵算法 ……35
2.4.2 最小环均值算法 ……38
2.5 多速率数据流图的迭代边界 ……41
2.6 结论 ……42
2.7 习题 ……42
参考文献 ……45
第3章 流水线与并行处理 ……47
3.1 引言 ……47
3.2 FIR数字滤波器的流水线 ……48
3.2.1 数据广播结构 ……50
3.2.2 细粒度流水线 ……51
3.3 并行处理 ……51
3.4 流水线与并行处理的功耗减低 ……54
3.4.1 用流水线降低功耗 ……55
3.4.2 用并行处理降低功耗 ……57
3.4.3 流水线和并行处理的结合 ……61
3.5 结论 ……61
3.6 习题 ……61
参考文献 ……65
第4章 重定时 ……67
4.1 引言 ……67
4.2 定义与性质 ……68
4.2.1 重定时的定量描述 ……68
4.2.2 重定时的性质 ……68
4.3 不等式求解系统 ……70
4.4 重定时技术 ……71
4.4.1 割集重定时和流水线 ……71
4.4.2 时钟周期最小化的重定时 ……73
4.4.3 使寄存器数最小化的重定时 ……78
4.5 结论 ……82
4.6 习题 ……82
参考文献 ……86
第5章 展开 ……89
5.1 引言 ……89

5.2 一种展开算法 ……90
5.3 展开的属性 ……93
5.4 关键路径、展开和重定时 ……94
5.5 展开的应用 ……95
5.5.1 采样周期的缩短……95
5.5.2 并行处理……98
5.6 结论……103
5.7 习题……104
参考文献……108
第6章 折叠 ……111
6.1 引言……111
6.2 折叠变换……112
6.3 寄存器最小化技术……116
6.3.1 寿命分析 ……116
6.3.2 采用前向-后向寄存器分配的数据分配技术 ……118
6.4 折叠架构的寄存器最小化……121
6.4.1 双2次节滤波器例子……121
6.4.2 IIR滤波器例子……122
6.5 多速率系统的折叠……124
6.6 结论……128
6.7 习题……128
参考文献……137
第7章 脉动结构设计 ……139
7.1 引言……139
7.2 脉动阵列设计方法原理……139
7.3 FIR脉动阵列 ……141
7.3.1 设计 B_1（输入广播，结果移动，权重保持）……141
7.3.2 设计 B_2（输入广播，权重移动，结果保持）……143
7.3.3 设计 F（结果扇入，输入移动，权重保持）……144
7.3.4 设计 R_1（结果保持，输入和权重反向移动）……145
7.3.5 设计 R_2 和双 R_2（结果保持，输入和权重同方向但不同速度移动）……146
7.3.6 设计 W_1（权重保持，输入和结果反向移动）……146
7.3.7 设计 W_2 和双 W_2（权重保持，输入和结果同方向但不同速度移动）……147
7.3.8 应用变换的关联脉动设计 ……147
7.4 调度矢量的选择……147
7.4.1 基于调度不等式选择 s^T ……147
7.4.2 RIA描述 ……149
7.4.3 应用RDG的调度矢量和脉动阵列设计 ……149
7.5 矩阵乘法与二维脉动阵列设计……151
7.6 包含延迟的空间表示脉动设计……154
7.7 结论……155
7.8 习题……155
参考文献……164
第8章 快速卷积……165
8.1 引言……165
8.2 Cook-Toom算法……165
8.3 Winograd算法……172
8.4 迭代卷积……178
8.5 循环卷积……179
8.6 通过观察设计快速卷积算法……182
8.7 结论……183
8.8 习题……183
参考文献……184
第9章 滤波器和变换中的算法强度缩减…185
9.1 引言……185
9.2 并行FIR滤波器 ……185
9.2.1 并行FIR滤波器的多相式分解表示 …185
9.2.2 快速FIR算法 ……188
9.3 离散余弦变换和反离散余弦变换……200
9.3.1 算法结构变换 ……202
9.3.2 对 2^m 点DCT的频率抽取快速DCT ……205
9.4 秩-阶滤波器的并行结构 ……209
9.4.1 奇偶合并分类结构 ……209
9.4.2 秩-阶滤波器结构 ……211
9.4.3 并行秩-阶滤波器 ……212
9.4.4 运行次序合并分类器-时间映射技术 ……214
9.4.5 低功耗秩-阶滤波器 ……216

9.5 结论……217
9.6 习题……217
参考文献……226
第10章 流水线结构的并行自适应递归滤波器……229
10.1 引言……229
10.2 数字滤波器中的流水线交织操作……229
10.2.1 低效率的单通道/多通道交织……230
10.2.2 高效的单通道交织……231
10.2.3 高效率的多通道交织……233
10.3 一阶IIR数字滤波器中的流水线实现…233
10.3.1 一阶IIR滤波器的超前流水线结构…234
10.3.2 在2的幂次分解中实现超前流水线…234
10.3.3 在通用分解中实现超前流水线……236
10.4 高阶IIR数字滤波器中的流水线实现…237
10.4.1 聚类超前流水线……239
10.4.2 稳定的聚类超前滤波器设计……240
10.4.3 离散超前流水线……241
10.4.4 基于2的幂次分解的离散超前流水线……243
10.4.5 在通用分解中实现离散超前流水线……244
10.4.6 受限的滤波器设计技术……244
10.5 IIR滤波器的并行处理……247
10.6 组合了流水线和并行处理的IIR滤波器……252
10.7 利用流水线和并行处理的低功耗IIR滤波器设计……254
10.8 流水线自适应数字滤波器……257
10.8.1 弛豫超前变换……257
10.8.2 流水线LMS自适应滤波器……260
10.8.3 用流水线实现随机梯度格型结构…261
10.9 结论……269
10.10 习题……269
参考文献……273
第11章 缩放噪声与舍入噪声……277
11.1 引言……277
11.2 缩放噪声和舍入噪声……277
11.2.1 缩放操作……277
11.2.2 舍入噪声……278
11.3 数字滤波器的状态变量描述……280
11.4 缩放噪声和舍入噪声的计算……282
11.4.1 缩放操作……282
11.4.2 舍入噪声……284
11.5 流水线IIR滤波器中的舍入噪声……287
11.5.1 一阶IIR滤波器……287
11.5.2 二阶IIR滤波器……293
11.6 舍入噪声的状态变量描述计算……296
11.7 降速、重定时和流水线……298
11.8 结论……302
11.9 习题……302
参考文献……308
第12章 格型数字滤波器结构……311
12.1 引言……311
12.2 Schur算法……311
12.2.1 Schur多项式的计算……311
12.2.2 Schur多项式的正规直交性……312
12.2.3 多项式展开算法……315
12.2.4 应用Schur算法的功率计算……316
12.3 基本数字格型滤波器……317
12.3.1 基本格型滤波器的导出……317
12.3.2 基本格型滤波器的逆Schur多项式导出……320
12.3.3 FIR格型滤波器的导出……322
12.4 单乘法器格型滤波器的导出……324
12.5 归一化格型滤波器的导出……328
12.6 可缩放归一化格型滤波器的导出……331
12.7 格型滤波器中的舍入噪声计算……335
12.7.1 应用转置定理的舍入噪声计算……336
12.7.2 舍入噪声比较……337
12.8 格型IIR数字滤波器的流水线……338
12.8.1 Schur算法的流水线性质……339
12.8.2 基本格型滤波器的流水线……340
12.8.3 单乘法器格型滤波器的流水线……341
12.8.4 归一化格型滤波器的流水线……342
12.8.5 可缩放归一化格型滤波器的流水线……342
12.8.6 重定时流水线的格型滤波器……343

12.9 流水线格型滤波器的设计例子 ……344
12.10 低功耗CMOS格型IIR滤波器 ……346
12.11 结论 ……347
12.12 习题 ……347
参考文献 ……350
第13章 位级运算架构 ……353
13.1 引言 ……353
13.2 并行乘法器 ……354
13.2.1 具有符号扩展的并行乘法 ……354
13.2.2 Baugh-Wooley乘法器 ……358
13.2.3 改进的Booth重编码并行乘法器 …359
13.3 交织布局规则与基于位平面的数字滤波器 ……361
13.4 位串行乘法器 ……362
13.4.1 利用Horner法则的Lyon位串行乘法器的设计 ……362
13.4.2 利用脉动映射的位串行乘法器的设计 ……365
13.5 位串行滤波器的设计与实现 ……368
13.5.1 位串行FIR滤波器 ……369
13.5.2 位串行IIR滤波器 ……369
13.6 正则符号数运算 ……372
13.6.1 CSD表示法 ……372
13.6.2 CSD乘法 ……374
13.7 分布式运算 ……377
13.7.1 传统的分布式运算 ……377
13.7.2 使用偏移二进制编码的分布式运算 ……379
13.7.3 分布式运算的ROM分解 ……381
13.8 结论 ……382
13.9 习题 ……382
参考文献 ……388
第14章 冗余运算 ……391
14.1 引言 ……391
14.2 冗余数表示 ……391
14.3 无进位基2加法与减法 ……392
14.3.1 混合基2加法 ……393
14.3.2 混合基2减法 ……394
14.3.3 混合基2加法减法器 ……395
14.3.4 有符号二进制数位加法减法器 ……395
14.4 混合基4加法 ……397
14.4.1 最大冗余混合基4加法 ……397
14.4.2 最小冗余混合基4加法 ……398
14.5 基2混合冗余乘法架构 ……399
14.6 数据格式转换 ……403
14.6.1 非冗余到冗余转换 ……403
14.7 冗余到非冗余转换器 ……404
14.7.1 最低数位优先转换 ……405
14.7.2 最高数位优先转换 ……405
14.8 结论 ……408
14.9 习题 ……408
参考文献 ……410
第15章 数字强度缩减 ……413
15.1 引言 ……413
15.2 子表达式消除 ……413
15.3 多常数乘法 ……413
15.3.1 线性变换 ……415
15.3.2 多项式求值 ……416
15.4 数字滤波器中的子表达式共享 ……417
15.5 加性和乘性数字拆分 ……423
15.5.1 基于行的加性数字拆分 ……424
15.5.2 基于列的加性数字拆分 ……425
15.5.3 乘性数字拆分 ……426
15.6 结论 ……429
15.7 习题 ……429
参考文献 ……434
第16章 同步流水线、波流水线和异步流水线 ……435
16.1 引言 ……435
16.2 同步流水线与时钟风格 ……436
16.3 位级流水线VLSI设计中的时钟歪斜与时钟分布 ……442
16.3.1 时钟歪斜 ……442
16.3.2 时钟分布 ……444
16.4 波流水线 ……445
16.5 约束空间图与波流水线深度 ……449
16.5.1 $\Delta < T_{clk}$ ……450
16.5.2 $\Delta > 0$ ……451

16.6 波流水线系统的实现 ……451
16.6.1 NPCPL ……452
16.6.2 给定电路的波流水线算法……452
16.7 异步流水线 ……455
16.7.1 捆绑数据协议和双轨协议……456
16.7.2 两相和四相协议……457
16.8 信号转换图 ……457
16.9 信号转换图应用于设计互连电路 ……459
16.10 运算单元的实现……463
16.10.1 全定制VLSI实现 ……463
16.10.2 FPGA实现 ……465
16.11 结论……469
16.12 习题……469
参考文献……470
第17章 低功耗设计 ……473
17.1 引言 ……473
17.2 理论背景 ……475
17.3 按比例缩小与功耗 ……476
17.4 功耗分析 ……478
17.4.1 开关活动率……478
17.4.2 物理电容……484
17.5 功耗降低技术 ……485
17.5.1 路径平衡……485
17.5.2 晶体管和逻辑门尺寸优化……486
17.5.3 晶体管重排序……486
17.5.4 用于低功耗的重定时……487
17.5.5 电压按比例缩小和多电源电压……487
17.5.6 双/多-V_{th} ……489
17.5.7 时钟……490
17.5.8 电路风格 ……490
17.6 功耗估计方法 ……492
17.6.1 基于仿真的方法……492
17.6.2 非仿真性方法……493
17.7 结论 ……502
17.8 习题 ……503
参考文献……504
第18章 可编程数字信号处理器 ……509
18.1 引言 ……509
18.2 可编程数字信号处理器的进化 ……509
18.3 DSP处理器的重要特性 ……510
18.3.1 数据路径……510
18.3.2 存储器架构……511
18.3.3 流水线作业DSP处理器中的时间固定编码和数据固定编码……513
18.4 移动通信与无线通信用的DSP处理器 …514
18.5 多媒体信号处理用的处理器 ……515
18.5.1 增强对多媒体信号处理的能力……515
18.5.2 媒体处理器……518
18.5.3 具有多媒体辅助部件的通用目的微处理器……521
18.6 结论 ……522
参考文献……522
附录A 最短路径算法 ……525
附录B 调度技术和分配技术 ……529
附录C 欧几里德最大公因子算法 ……545
附录D Schur多项式的正规直交性 ……549
附录E 快速二进制加法器和乘法器……553
附录F 位串行系统中的调度 ……561
附录G FIR滤波器的系数量化 ……565
索引 ……569

第1章 数字信号处理系统导论

1.1 引言

数字信号处理（Digital Signal Processing, DSP）与模拟信号处理相比具有很多优点。例如，数字信号对于温度漂移和工艺变化更具鲁棒性，数字表示的精度通过改变信号的字长可以得到更好的控制。不仅如此，DSP技术可以在放大信号的同时消除噪声和干扰，而在模拟信号处理中，信号和噪声却都被放大了。数字信号本质上可以无误差地进行存储与恢复、发送与接收以及处理与操作。当然，对于如无线通信中超高频射频收发机这样的系统，模拟信号处理是不可替代的，在芯片面积要求极小和芯片功耗要求极低，像应用于探测破裂以及与其他应力相关的材料缺陷的微机械传感器系统中，也是如此。但是，大量的复杂系统都采用了数字实现，它们具有高精确度、高信噪比（Signal to Noise Ratio, SNR）的特点，同时具有可重复性和灵活性。

DSP系统可以用可编程的处理器实现，也可以用超大规模集成电路（Very-Large-Scale-Integrated, VLSI）技术生产的全定制硬件电路实现。数字设计的目标是在尽可能提高性能的同时保持低廉的价格。在通用数字设计领域，性能按照以下指标来衡量：所需的硬件电路和资源（即占用的空间或面积）；执行的速度，这取决于吞吐率和时钟速率；功耗，或者说是完成某个给定任务需要的总能量。对于定点DSP系统，尤其是数字滤波器，有限字长的性能（即量化噪声和舍入噪声）是其第四个性能指标，因为一个舍入噪声大的数字滤波器即使在面积、速度以及功耗上有很好的性能也是无法使用的。

DSP区别于其他通用计算的两个重要特性是：*实时吞吐率的需要和数据驱动的性质*。设计硬件时需满足实时处理中对吞吐率的严格约束。在这里，新的输入样点从信号源被周期性接收进来时就需要及时处理，这与一般的首先将输入的数据存储起来然后再批处理的方式相反。如果系统吞吐率低于需要的采样率，新的输入数据就得停顿（或缓存）下来，从而需要无限大的缓存器。然而，一旦硬件满足了采样率要求，则再加快计算速度也并无优势了。信号处理系统中的第二个重要属性是它的*数据驱动性质*。一旦所有的输入数据都准备好时，任何一个DSP系统中的子任务或计算皆可执行。从这种意义上说，系统是由数据流而非系统时钟进行同步的。这一点使得DSP系统可以由不需要全局时钟的异步电路实现。

本章作为引论将给出关于典型的DSP算法、目前在多媒体信号处理中的设计难点以及DSP算法的表达等方面的综述。第1.2节评述了几种典型的DSP算法，第1.3节介绍一些多媒体DSP功能的计算需要以及按比例缩小的VLSI技术对DSP系统的实现所带来的机会。第1.4节阐述了四种不同DSP算法的表示方法：框图(block diagram)、信号流图(signal flow graph)、数据流图(data flow graph)以及依赖图(dependence graph)表示。第1.5节将简介本书中各章节的内容。

1.2 典型的DSP算法

表1-1中列出了一些DSP算法及其一般应用[1]。本节将介绍以下几种重要的DSP运算，

其中包括：相关、卷积和数字滤波器；随机梯度和最小均方差（Least-mean-Square, LMS）自适应滤波器；应用于运动估值的块匹配算法(Motion Estimation, ME)、离散余弦变换（Discrete Cosine Transform, DCT）和应用于图像处理与压缩的矢量量化（Vector quatization, VQ）；Viterbi（维特比）算法和动态规划；用于多速率信号处理的抽取与插入、小波与滤波器组。

2

表1-1 一般DSP算法及其应用举例

DSP算法	系统应用
语音编码与解码	数字移动电话、个人通信系统、数字无绳电话、多媒体计算机、安全通信
语音加密与解密	数字蜂窝电话、个人通信系统、数字无绳电话、安全通信
语音识别	超级用户接口、多媒体工作站、机器人以及自动化应用、数字蜂窝电话、个人通信系统、数字无绳电话
语音合成	多媒体PC机、超级用户接口、机器人
调制解调算法	数字蜂窝电话、个人通信系统、数字无绳电话、数字音频广播、多媒体计算机、无线计算、导航、数据/传真调制解调器、安全通信
噪声消除	专业音响、高级汽车音响
音频均衡	消费级音响、专业音响、高级汽车音响
图像压缩和解压缩	数码相机、数码摄影机、多媒体计算机、消费级摄像机
波束成形	导航、雷达/声纳、信号智能
回声消除	免提电话机、调制解调器、电话交换机

1.2.1 卷积

两个离散序列 $h(n)$ 和 $x(n)$ 的卷积定义为:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{1-1}$$

在时刻 n 的输出 $y(n)$ 可以看作是 $x(k)$ 和 $h(-k+n)$ 的内积（在$-\infty < k < \infty$范围内求和）。

卷积是用来描述和分析线性时不变（Linear Time-Invariant, LTI）系统的。这种系统可以完全由它们的单位样值（或冲激）响应 $h(n)$ 来描述[2]。一个LTI系统的输出序列可以用输入序列 $x(n)$ 以及它的单位样值响应 $h(n)$ 的卷积来计算。如果一个系统的单位样值响应由有限个非零值组成，即 $h(n)$ 有限长，那么该系统称为有限冲激响应（Finite Impulse Response, FIR）。如果 $h(n)$ 具有无限长度，该系统则称为无限冲激响应（Infinite Impulese Response, IIR）。例如，移动平均系统就是FIR系统，它的冲激响应为

3

$$h(n) = \frac{1}{M_1 + M_2 + 1} \sum_{-M_1}^{M_2} \delta(n-k)$$

累加器的单位样值响应为:

$$h(n) = \sum_{k=-\infty}^{n} \delta(k)$$

它是一个IIR系统。该响应是一个阶跃函数，当 $n > 0$ 时等于1，当 $n < 0$时等于0。

如果 $y(n_0)$ 的计算只与过去的采样值 $x(k)$ $(k < n_0)$ 相关，那么称该系统是因果（Causal）

的。对于一个因果的LTI系统，单位样值响应应满足当 $n<0$时 $h(n)=0$。我们只对因果的数字滤波器有兴趣，因为非因果的系统无法用硬件或者软件实现。

1.2.2 相关

相关运算在数字通信以及其他随机信号处理中有大量应用。两个序列 $a(n)$ 和 $x(n)$ 的相关定义为

$$y(n)=\sum_{k=-\infty}^{\infty}a(k)x(n+k) \tag{1-2}$$

公式（1-2）中的操作可以看成如下的卷积运算:

$$y(n)=\sum_{k=-\infty}^{\infty}a(-k)x(n-k)=a(-n)*x(n) \tag{1-3}$$

如果 $a(n)$ 和 $x(n)$ 都具有有限长度 N，即它们在 $n=0,1,\cdots,N-1$的样点中有非零值，则数字相关运算可以写为:

$$y(n)=\sum_{k=0}^{N-1}a(k)x(n+k) \tag{1-4}$$

其中 $n=-N+1,-N+2,\cdots,N-2,N-1$。公式（1-4）中的数字相关运算还可以写成矩阵-矢量相乘形式。例如 $N=4$ 时可写成:

$$\begin{bmatrix} y(-3)\\ y(-2)\\ y(-1)\\ y(0)\\ y(1)\\ y(2)\\ y(3)\end{bmatrix}=\begin{bmatrix} 0 & 0 & 0 & x(0)\\ 0 & 0 & x(0) & x(1)\\ 0 & x(0) & x(1) & x(2)\\ x(0) & x(1) & x(2) & x(3)\\ x(1) & x(2) & x(3) & 0\\ x(2) & x(3) & 0 & 0\\ x(3) & 0 & 0 & 0\end{bmatrix}\begin{bmatrix} a(0)\\ a(1)\\ a(2)\\ a(3)\end{bmatrix} \tag{1-5}$$

1.2.3 数字滤波器

数字滤波器是LTI系统中重要的一类，它用于改变输入信号 $x(n)$ 的频谱特性以满足某种特定的设计要求。一个因果的数字滤波器可以用它的单位样值响应 $h(n)$、频率响应 $H(e^{j\omega})$（或者传输函数 $H(z)$）或者差分方程来完整地表达。其中单位样值响应和频率响应描述了系统的时域和频域性质，差分方程表示则清楚地反映实现该滤波器所需的计算。

一个线性时不变因果滤波器可以用如下差分方程描述:

$$y(n)=-\sum_{k=1}^{N}a_k y(n-k)+\sum_{k=0}^{M-1}b_k x(n-k) \tag{1-6}$$

如果 $1<k<N$ 时 $a_k=0$，公式（1-6）可化简为:

$$y(n)=\sum_{k=0}^{M-1}b_k x(n-k) \tag{1-7}$$

它是一个 M 阶有限冲激响应（FIR）滤波器，当 $0<k<M-1$ 时其单位样值响应 $h(k)=b_k$，其他时间 $h(k)=0$。这是一个非递归的运算。如果在 $0<k<N$ 中至少有一个 $a_k\neq 0$，那么公式（1-6）表示一个递归运算，即 $y(n)$ 的运算需要过去的输出值，因此称之为递归滤波器。与之对应的单位样值响应的长度是无限的，因此，它也被叫做无限冲激响应（IIR）滤波器。具体选择FIR滤波器还是IIR滤波器取决于实际应用的需求。

在设计频率选择性数字滤波器时，通常希望能有近似恒定的频响幅度，并尽量减少通带内的相位失真[2]。斜率为整数的线性相位对应于时域中简单的延时，它在频域中可将相位失真降低到最小的程度。因此，我们更倾向于将滤波器设计成具有准确的或者是近似的线性相位。线性相位FIR滤波器之所以特别具有吸引力是因为它的单位样值响应是对称的，这样只需要一半的乘法运算。比如，一个 M 阶线性相位的FIR滤波器的单位样值响应满足：

5

$$h(n)=h(M-n)$$

于是，一个7阶线性相位FIR滤波器的冲激响应为：

$$h(0)=h(6)=b_0,\ h(1)=h(5)=b_1,\ h(2)=h(4)=b_2,\ h(3)=b_3$$

该滤波器可写为：

$$\begin{aligned} y(n) &= b_0x(n)+b_1x(n-1)+b_2x(n-2)+b_3x(n-3) \\ &\quad +b_0x(n-6)+b_1x(n-5)+b_2x(n-4) \end{aligned} \tag{1-8}$$

由此可以用图1-1a或图1-1b的结构实现，其中只用到了4个乘法器、6个加法器以及6个存储单元。图1-1b采用了数据广播结构，输入的数据在同一时刻广播到了所有的乘法器进行运算。

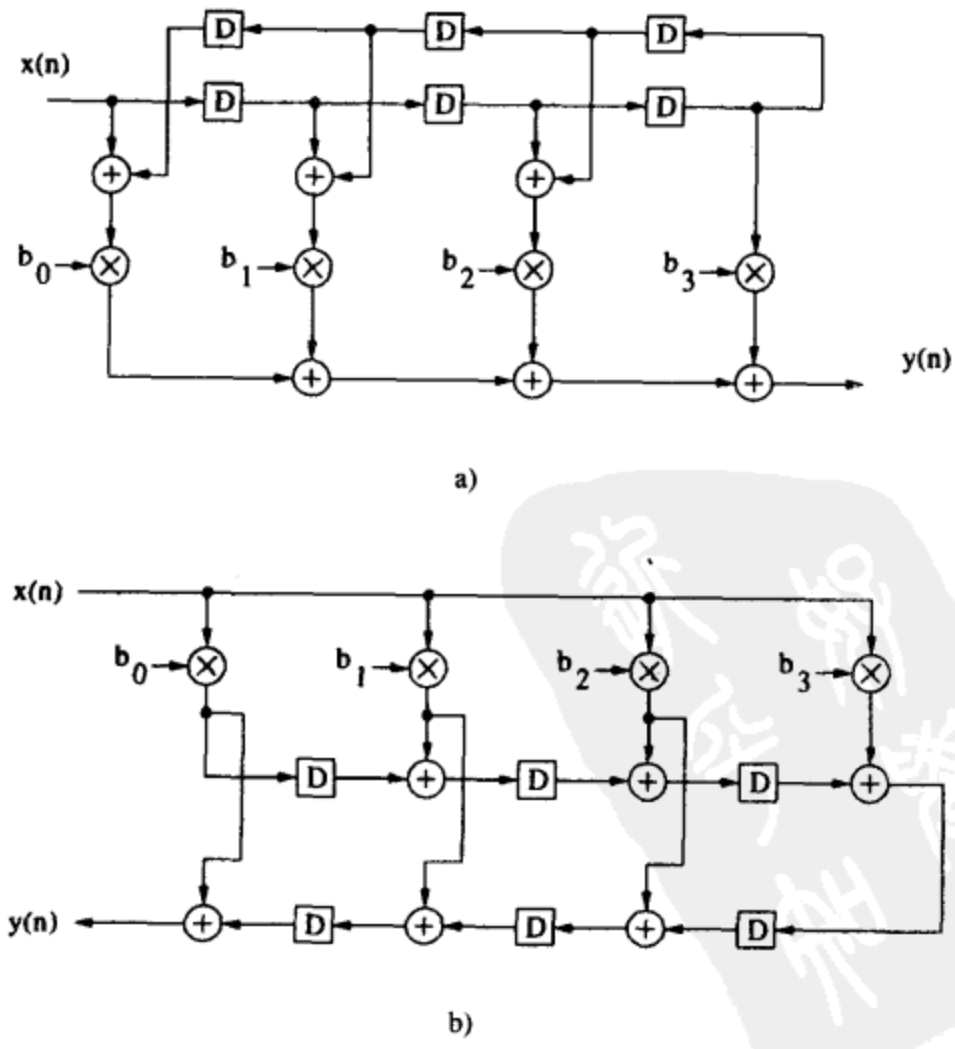


图1-1 7阶线性相位FIR滤波器框图

1.2.4 自适应滤波器

在确定性环境下，固定系数的数字滤波器用做频率整形是非常理想的。自适应滤波器可应用于回声消除、信道均衡、语音频带的调制解调、数字移动无线设备、系统辨识、控制、声学波束成形以及语音和图像处理[3]。自适应滤波器采用了线性模型，比如数字滤波器，它通过观察一个随机过程 $\{x(n)\}$ 来预测另外一个随机过程 $\{y(n)\}$。与固定系数的滤波器（比如FIR和IIR）不同，自适应滤波器的系数在每次迭代的时候会更新，其目的是为了减小滤波器的输出与理想信号之间的差别（difference）。当系数收敛后，更新的过程才会结束。所以自适应数字滤波器通常包含有一个通用的滤波器模块和一个系数更新模块。根据不同的差别最小化目标可以推导出各种自适应处理过程。下面的小节将阐述LMS自适应滤波器（具有FIR滤波器模块）和随机梯度自适应格型滤波器的推导。

1.2.4.1 LMS自适应滤波器

在LMS自适应算法中，所有观察的权重和

$$\hat{d}(n) = \mathbf{W}^T(n-1)\mathbf{U}(n) \tag{1-9}$$

表示信号 $d(n)$ 的估计，其中

$$\mathbf{W}^T(n) = [w_1(n), w_2(n), \cdots, w_N(n)]$$

是权重矢量，而

$$\mathbf{U}^T(n) = [u(n), u(n-1), \cdots, u(n-N+1)]$$

包含当前和过去的输入信号。估计误差用 $e(n)$ 表示，是期望信号与估计信号之差，即

6~7

$$e(n) = d(n) - \hat{d}(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{U}(n) \tag{1-10}$$

在第 n 次迭代时，LMS算法选择这样的 $W^T(n)$，使得平方误差 $e^2(n)$最小。因此，LMS自适应滤波器包含有两部分，一部分是由系数矢量 $W^T(n)$ 和输入序列 $u(n)$ 组成的FIR滤波器(F-Block)，另一部分是权重（系数）更新（Weight Update, WUD）模块。

为了推导出权重更新算法，求 $e^2(n)$ 对 $W^T(n-1)$ 的导数（为简单起见，时间索引 n 略掉）:

$$\begin{aligned}\Delta_{\mathbf{W}^T}(e^2) &= \frac{\partial e^2}{\partial \mathbf{W}^T} = -2d\mathbf{U} + 2\mathbf{W}^T\mathbf{U}\cdot\mathbf{U} \\ &= -2(d - \mathbf{W}^T\mathbf{U})\mathbf{U} = -2e\mathbf{U}\end{aligned} \tag{1-11}$$

权重矢量的更新过程可以描述为:

$$\mathbf{W}(n) = \mathbf{W}(n-1) - \frac{1}{2}\mu\nabla_{\mathbf{W}^T}(e^2(n)) \tag{1-12}$$

最后可得:

$$\mathbf{W}(n) = \mathbf{W}(n-1) + \mu e(n)\mathbf{U}(n) \tag{1-13}$$

这个LMS滤波器的系统级框图如图1-2所示。在每次迭代中，误差 $e(n)$（公式1-10）通过滤波器模块（F-block）来计算，而权重矢量（公式1-13）则通过WUD模块来更新。图1-3表示LMS滤波器的详细框图。

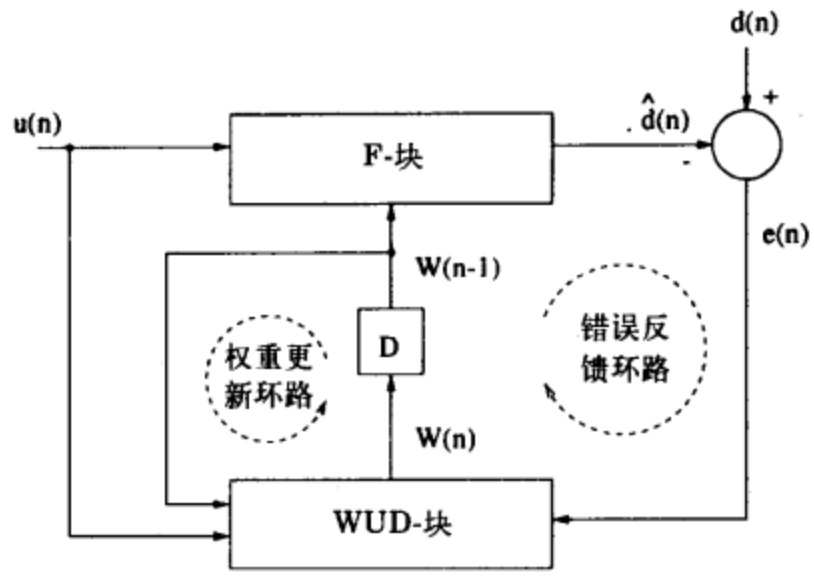


图1-2 LMS自适应滤波器系统框图

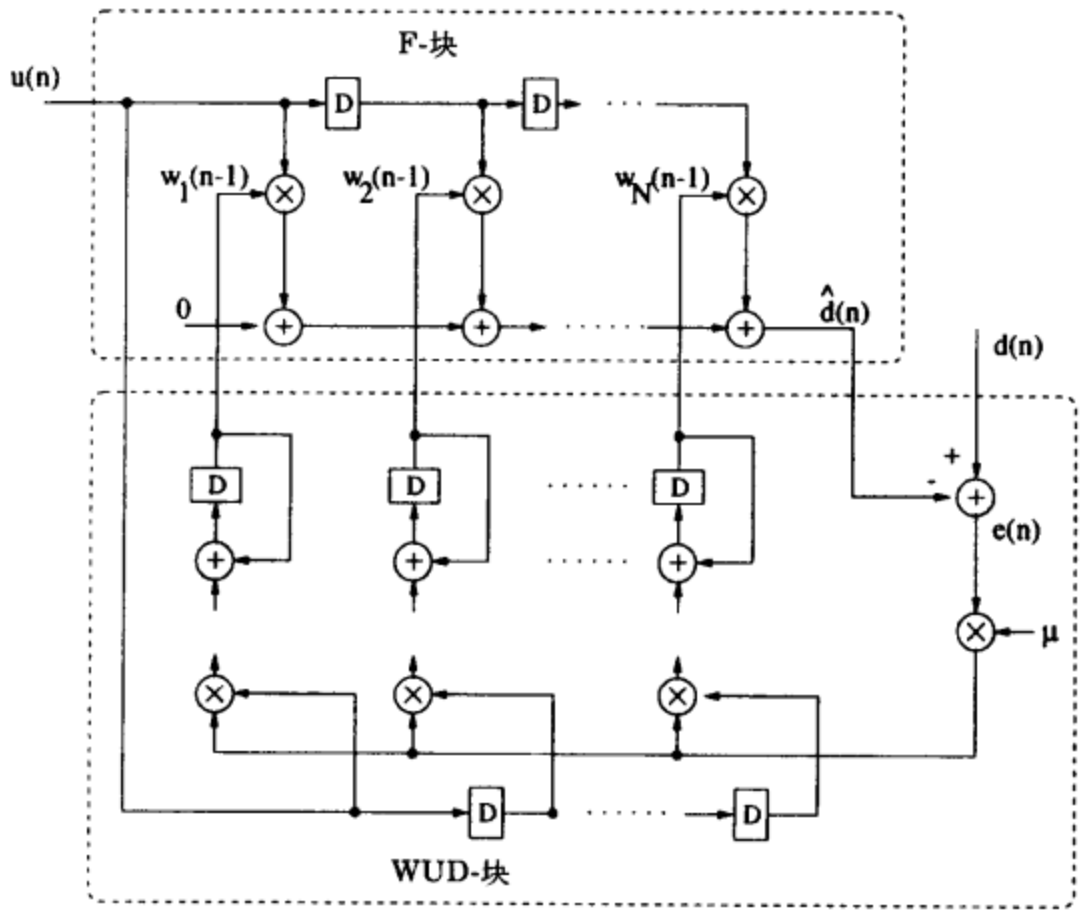


图1-3 LMS自适应数字滤波器

1.2.4.2 随机梯度自适应格型滤波器

关于格型滤波器结构的基本概念和推导在[4]，[2]中已有详细介绍（也可参考第12章），因此这里假定成立。

用 n 表示时刻，$m = 1, 2, \cdots, N$ 表示格子的级数（其中N表示总级数或者滤波器的阶数）。那么对于前向和后向预测误差各级的更新方程可以分别写为：

$$\begin{aligned} e_f(n|m) &= e_f(n|m-1) - k_m(n)e_b(n-1|m-1) \\ e_b(n|m) &= e_b(n-1|m-1) - k_m(n)e_f(n|m-1) \end{aligned} \tag{1-14}$$

其中 k_m 是偏相关系数（也叫做PARCOR系数或者反射系数）[5]（参考第12.3.3节）。在图1-4中，阴影部分表示一个前向格子级，并且进行公式（1-14）的运算。

随机梯度自适应算法选配 k_m，使得

$$J(n) = e_f^2(n|m) + e_b^2(n|m) \tag{1-15}$$

最小化，该式表示前向和后向预测误差的平方和。将 $J(n)$ 用 $e_f(n \mid m-1)$, $e_b(n-1 \mid m-1)$ 和 $k_m(n)$ 表示，取 $J(n)$ 对 k_m 的导数，可得:

$$\begin{aligned}\frac{\partial J(n)}{\partial k_m(n)} = &-2[e_f(n|m-1) - k_m(n)e_b(n-1|m-1)]e_b(n-1|m-1)\\ &-2[e_b(n-1|m-1) - k_m(n)e_f(n|m-1)]e_f(n|m-1)\end{aligned}$$

8~9

该导数利用当前对 $k_m(n)$ 的估计值进行计算得出，加权之后从当前的估计值中减去，得到新的估计值 $k_m(n)$。更新方程写为:

$$\begin{aligned}k_m(n+1) &= k_m(n) - \frac{\beta_m(n)}{2}\frac{\partial J(n)}{\partial k_m}\\ &= k_m(n)[1 - \beta_m(n)(e_f^2(n|m-1) + e_b^2(n-1|m-1))]\\ &\quad +2\beta_m(n)e_f(n|m-1)e_b(n-1|m-1)\end{aligned} \tag{1-16}$$

其中 $\beta_m(n)$ 是自适应常量。为了使自适应速度与输入信号电平相对无关，需要用第 $(m-1)$ 阶预测误差的方差之和的估计值对步长 $\beta_m(n)$ 进行归一化，可写为:

$$\beta_m(n) = \frac{1}{S(n|m-1)} \tag{1-17}$$

其中 $S(n \mid m-1)$ 通过递归方法估计为:

$$\begin{aligned}S(n+1|m-1) &= (1-\beta)S(n|m-1)\\ &\quad +e_f^2(n|m-1) + e_b^2(n-1|m-1)\end{aligned} \tag{1-18}$$

这里 β 是个常量，与 $S(0 \mid m-1)$ 的初始值一起用来控制自适应的速度。

随机梯度自适应算法可用下列的方程完整地描述:

$$\begin{aligned}k_m(n+1) &= [1 - \beta_m(e_f^2(n|m-1) + e_b^2(n-1|m-1))]k_m(n)\\ &\quad +2\beta_m e_f(n|m-1)e_b(n-1|m-1)\end{aligned} \tag{1-19}$$

$$\begin{aligned}S(n+1|m-1) &= (1-\beta)S(n|m-1)\\ &\quad +e_f^2(n|m-1) + e_b^2(n-1|m-1)\end{aligned} \tag{1-20}$$

$$\beta_m = \frac{1}{S(n|m-1)} \tag{1-21}$$

$$e_f(n|m) = e_f(n|m-1) - k_m(n)e_b(n-1|m-1) \tag{1-22}$$

$$e_b(n|m) = e_b(n-1|m-1) - k_m(n)e_f(n|m-1) \tag{1-23}$$

公式（1-19）~（1-21）和公式（1-22）~（1-23）可以分别称为自适应方程与阶数更新方程。随机梯度格型滤波器的框图如图1-4所示，其中前向和后向预估误差 e_f 和 e_b 可用前向格子级计算（阴影部分）。

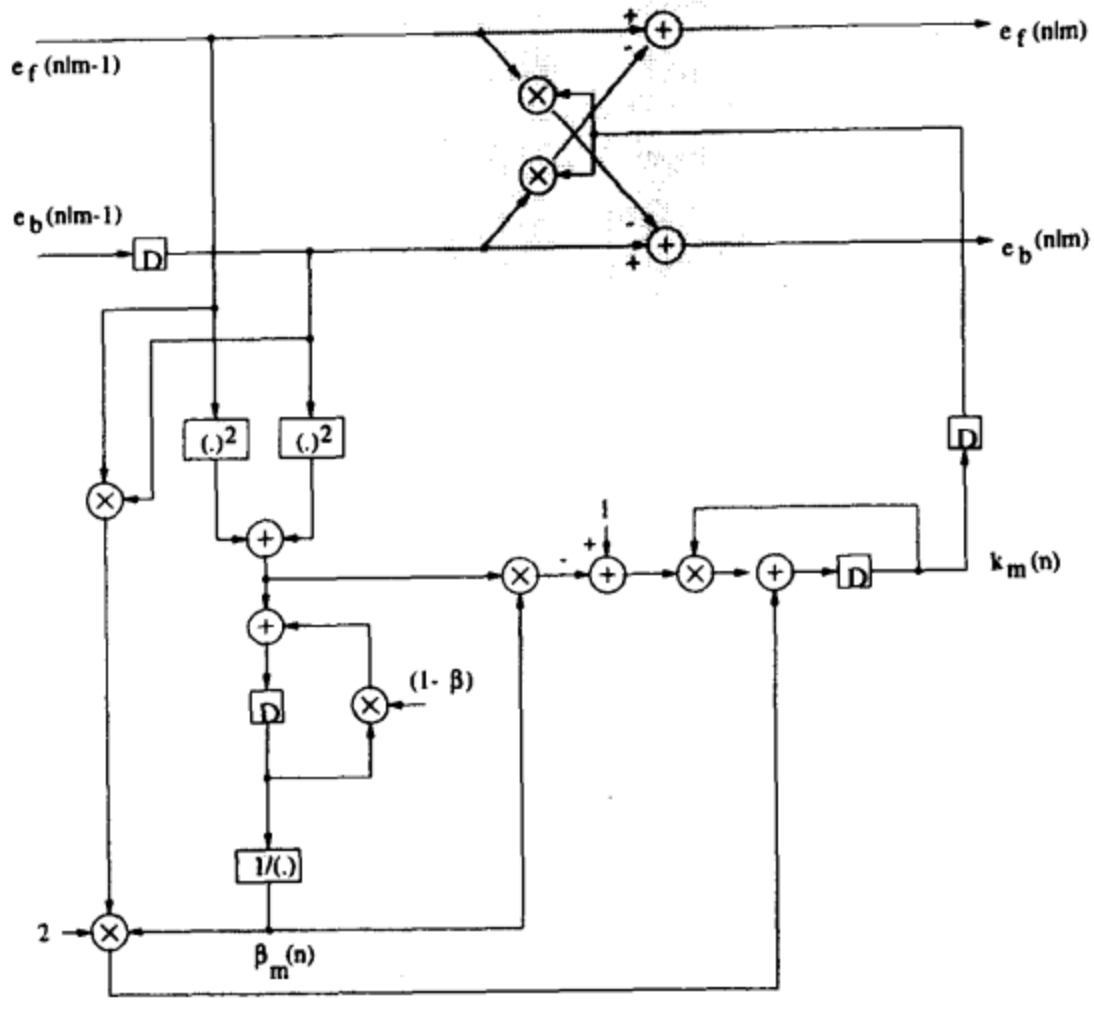


图1-4 随机梯度自适应格型滤波器结构

1.2.5 运动估值

运动估值主要用于帧间预测编码，它在视频编码中消耗运算资源最多。在运动估值中，视频序列按帧相继经过分析以便预测像素或像素块的运动（或者是位移）矢量。进行传输的是运动矢量而不是当前帧的相应像素块。对前一编码帧的像素块进行运动-补偿预测可获得最佳的压缩效果。因为块匹配算法（Block-Matching Algorithm, BMA）相对比较简单，所以是最优先应用的方法。在BMA中，每帧被分为了 $N \times N$ 的宏参考像素块，并假定在一个块中所有像素的运动相同。每个当前帧的参考块与前一帧中位移的候选块相比较，将其中最佳匹配的候选块与参考块之间的偏移定义为运动矢量。前一帧的搜寻范围定义为搜索窗（search window），它表示在水平和垂直方向上，搜索范围相对于参考块的位移都限定在 $+/-p$ 个像素中。因此搜索窗包含有 $(N+2p)^2$ 个像素。块匹配算法描述见图1-5。

10 ~ 11

定义最佳匹配的搜索判据有好几种，包括互相关函数(Cross-Correlation Function, CCF)，均方误差（Mean-Square Error, MSE）和平均绝对差（Mean Absolute Difference, MAD）。在实际应用中，因为MAD函数简单，而且具有令人满意的性能，因此在实际实现中应用最为广泛。在MAD算法中，位移为 (m, n) 的位移块与参考块的MAD差值 $s(m, n)$ 定义为

$$s(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |x(i,j) - y(i+m, j+n)| \quad for \ -p \leqslant m,\ n \leqslant p \tag{1-24}$$

其中 $x(i, j)$ 表示当前帧参考块的像素值，$y(i+m, j+n)$ 表示前一帧搜索窗中候选块的像素值。

公式1-24需要 $3N^2$ 次运算（每次差的绝对值计算中需要进行一次减法、一次求绝对值和一次加法）。可以采用很多搜索策略来确定最佳匹配块，其中全搜索（full search）是最为直接的方法。这种方法搜索窗内的所有 $(2p+1)^2$ 个像素，用下面的式子计算运动矢量 **v**:

$$\begin{aligned} u &= min_{(m,n)}\{s(m,n)\},\ 对于 -p \leqslant m,\ n \leqslant p \\ \mathbf{v} &= (m,n)|_u \end{aligned}$$

因此，对于一个 $N_h \times N_v$ 的帧（在每帧中每行有 N_h 像素，每帧有 N_v 行），全搜索BMA每帧都有

$$\frac{N_h \times N_v}{N^2} \cdot (2p+1)^2 \cdot 3N^2 = 3(2p+1)^2 N_h N_v$$

次运算。假定帧速率为 F 帧每秒，那么全搜索BMA的运算负载为 $3(2p+1)^2 N_h N_v F$ 次操作每秒。

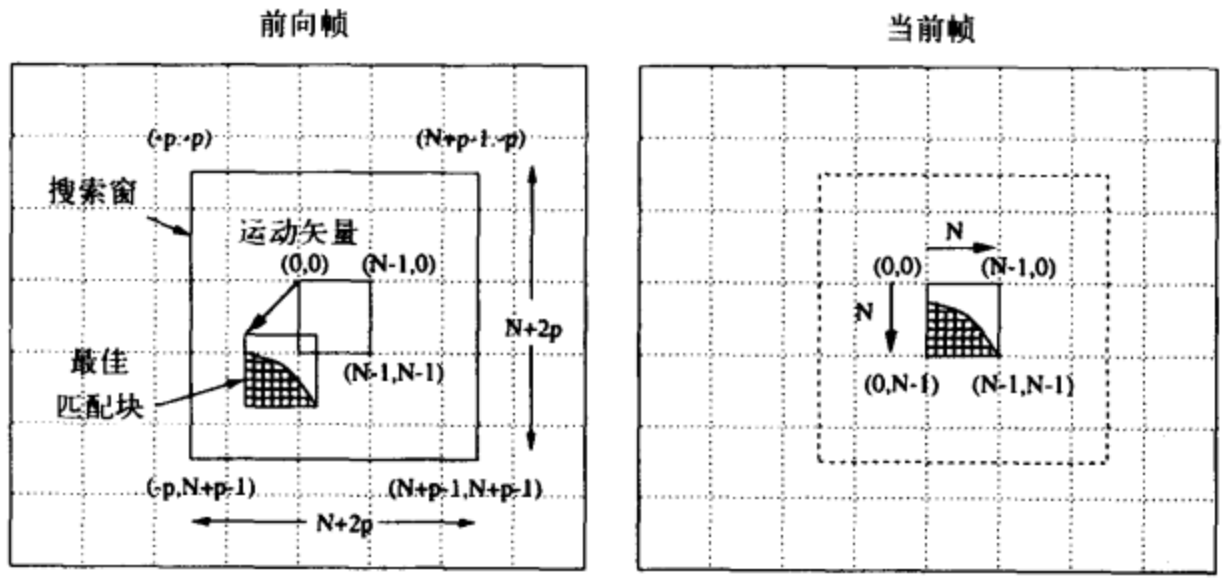


图1-5 用于运动估值的块匹配算法

1.2.6 离散余弦变换

离散余弦变换（Discrete Cosine Transform, DCT）是一种频域变换，它是为了在图像处理中进行图案识别和Wiener(维纳)滤波而首次引入的[6]。当前作为变换编码器，广泛地用于静态与运动图像和视频压缩中。本节介绍偶对称一维DCT。

考虑一个 N 个点的序列 $x(n)$，即当 $n<0$ 或者 $n>N-1$ 时，$x(n)=0$。对于该序列的 N 点DCT和离散余弦反变换（IDCT）定义如下:

$$X(k) = e(k)\sum_{n=0}^{N-1} x(n)\cos[\frac{(2n+1)k\pi}{2N}],\ k=0,1,\cdots,N-1 \tag{1-25}$$

$$x(n) = \frac{2}{N}\sum_{k=0}^{N-1} e(k)X(k)\cos[\frac{(2n+1)k\pi}{2N}],\ n=0,1,\cdots,N-1 \tag{1-26}$$

其中

$$e(k) = \begin{cases} \frac{1}{\sqrt{2}}, & 如果\ k=0 \\ 1, & 否则 \end{cases} \tag{1-27}$$

N 点DCT和IDCT可以用$2N$点离散傅立叶变换（Discrete Fourrier Transform, DFT）及其反变换推导出来。利用 $x(n)$ 及其镜像建立一个$2N$点序列 $y(n)$ 如下:

$$y(n) = x(n) + x(2N-n-1) = \begin{cases} x(n), & 0 \leqslant n \leqslant N-1 \\ x(2N-n-1), & N \leqslant n \leqslant 2N-1 \end{cases} \tag{1-28}$$

因此 $y(n)$ 对于中点 $n=(N-1)/2$ 是对称的。图1-6示出 $N=5$的例子。

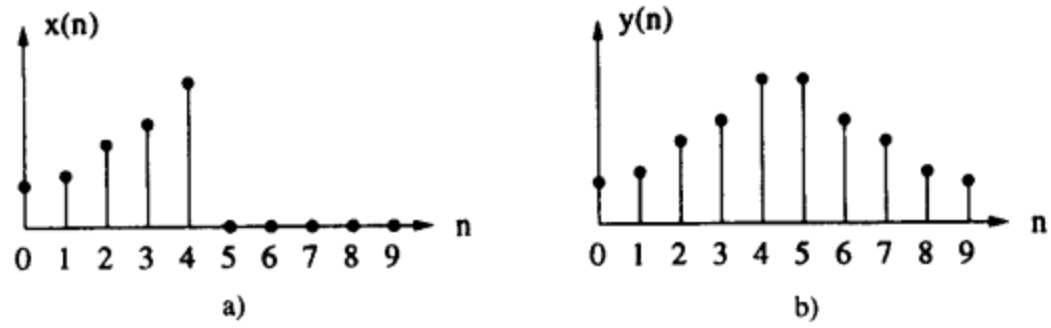


图1-6 N点序列 $x(n)$ 与$2N$点序列 $y(n)=x(n)+x(2N-n-1)$ 之间关系

$2N$点 $y(n)$ 的DFT可以表示为:

$$\begin{aligned} Y_D(k) &= \sum_{n=0}^{2N-1} y(n)e^{-j\frac{2\pi}{2N}kn} \\ &= \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{2N}kn} + \sum_{n=N}^{2N-1} x(2N-n-1)e^{-j\frac{2\pi}{2N}kn} \end{aligned} \tag{1-29}$$

其中 $0 \leqslant k \leqslant 2N-1$。用 $n=2N-n'-1$ 代入公式(1-29)中的第二个求和式子中，可得:

$$\begin{aligned} \sum_{n=N}^{2N-1} x(2N-n-1)e^{-j\frac{2\pi}{2N}kn} &= \sum_{n'=N-1}^{0} x(n')e^{-j\frac{2\pi}{2N}k(2N-n'-1)} \\ &= \sum_{n'=0}^{N-1} x(n')e^{j\frac{2\pi}{2N}kn'}e^{j\frac{2\pi}{2N}k} \end{aligned} \tag{1-30}$$

根据（1-30)，(1-29）可重写为:

$$\begin{aligned} Y_D(k) &= \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{2N}kn} + \sum_{n=0}^{N-1} x(n)e^{j\frac{2\pi}{2N}kn}e^{j\frac{2\pi}{2N}k} \\ &= e^{j\frac{k\pi}{2N}}\left(\sum_{n=0}^{N-1} x(n)e^{-j\frac{(2n+1)k\pi}{2N}} + \sum_{n=0}^{N-1} x(n)e^{j\frac{(2n+1)k\pi}{2N}}\right) \\ &= e^{j\frac{k\pi}{2N}} \sum_{n=0}^{N-1} 2x(n)\cos\left(\frac{(2n+1)k\pi}{2N}\right) \end{aligned} \tag{1-31}$$

定义

$$\hat{X}(k) = \begin{cases} Y_D(k)e^{-j(\frac{k\pi}{2N})}, & 0 \leqslant k \leqslant N-1 \\ 0, & \text{其他} \end{cases} \tag{1-32}$$

那么N点DCT可表示为 $X(k)=e(k)\,\hat{X}(k)/2$。

12 ~ 14

IDCT可以通过求 $Y_D(k)$ 与 $X(k)$ 的关系来推导。通过IDFT将 $Y_D(k)$ 变换到 $y(n)$，然后由 $y(n)$ 重构 $x(n)$ 即可。尽管 $Y_D(k)$ 序列长度为$2N$，$X(k)$ 序列长度为N，利用 $y(n)$ 中的冗

余性（对称性）可以使用 $X(k)$ 来表示 $Y_D(k)$，即当 $0 \leqslant k \leqslant N-1$时，$Y_D(k) = e^{j\frac{k\pi}{2N}}\hat{X}(k)$；$Y_D(N) = 0$。对于 $N+1 \leqslant k \leqslant 2N-1$ 有 $1 \leqslant 2N-k \leqslant N-1$，因此，

$$Y_D(2N-k) = e^{j\frac{(2N-k)\pi}{2N}}\hat{X}(2N-k) = -e^{-j\frac{k\pi}{2N}}\hat{X}(2N-k) \tag{1-33}$$

但是，根据公式（1-31），

$$\begin{aligned} Y_D(2N-k) &= e^{j\frac{(2N-k)\pi}{2N}}\sum_{n=0}^{N-1} 2x(n)\cos(\frac{(2n+1)(2N-k)\pi}{2N}) \\ &= -e^{j\frac{2N\pi}{2N}}e^{-j\frac{k\pi}{2N}}\sum_{n=0}^{N-1} 2x(n)\cos(\frac{(2n+1)k\pi}{2N}) \\ &= e^{-j\frac{2k\pi}{2N}}e^{j\frac{k\pi}{2N}}\sum_{n=0}^{N-1} 2x(n)\cos(\frac{(2n+1)k\pi}{2N}) \\ &= e^{-j\frac{2k\pi}{2N}}Y_D(k) \end{aligned} \tag{1-34}$$

则对于 $N+1 \leqslant k \leqslant 2N-1$ 有:

$$\begin{aligned} Y_D(k) &= e^{j\frac{2k\pi}{2N}}Y_D(2N-k) \\ &= -e^{j\frac{2k\pi}{2N}}e^{-j\frac{k\pi}{2N}}\hat{X}(2N-k) \\ &= -e^{j\frac{k\pi}{2N}}\hat{X}(2N-k) \end{aligned} \tag{1-35}$$

因此，我们有

$$Y_D(k) = \begin{cases} e^{j\frac{k\pi}{2N}}\hat{X}(k), & 0 \leqslant k \leqslant N-1 \\ 0, & k = N \\ -e^{j\frac{k\pi}{2N}}\hat{X}(2N-k), & N+1 \leqslant k \leqslant 2N-1 \end{cases} \tag{1-36}$$

对 $Y_D(k)$ 进行IDFT，可得:

$$\begin{aligned} y(n) &= \frac{1}{2N}\sum_{k=0}^{2N-1} Y_D(k)e^{j\frac{2\pi}{2N}kn} \\ &= \frac{1}{2N}(\sum_{k=0}^{N-1}\hat{X}(k)e^{j\frac{(2n+1)k\pi}{2N}} + \sum_{k=N+1}^{2N-1}(-e^{j\frac{k\pi}{2N}}\hat{X}(2N-k))e^{j\frac{2\pi}{2N}kn}) \end{aligned} \tag{1-37}$$

对上式中第二部分进行变量代换，然后进行某些代数推导之后，利用 $1/e(0) = 2e(0)$ 及当 $k \neq 0$ 时 $1/e(k) = e(k)$，公式（1-37）可以重写为:

$$\begin{aligned} y(n) &= \frac{1}{2N}(\sum_{k=0}^{N-1}\hat{X}(k)e^{j\frac{(2n+1)k\pi}{2N}} + \sum_{k=1}^{N-1}\hat{X}(k)e^{-j\frac{(2n+1)k\pi}{2N}}) \\ &= \frac{1}{2N}(\hat{X}(0) + 2\sum_{k=1}^{N-1}\hat{X}(k)\cos(\frac{(2n+1)k\pi}{2N})) \\ &= \frac{2}{N}(\ e(0)X(0) + \sum_{k=1}^{N-1}X(k)e(k)\cos(\frac{(2n+1)k\pi}{2N})\ ) \end{aligned} \tag{1-38}$$

其中 $0 < n < 2N-1$。将 $y(n)$ 中前N个点取出，可得到IDCT:

$$x(n) = y(n) = \frac{2}{N}\sum_{k=0}^{N-1} e(k)X(k)\cos(\frac{(2n+1)k\pi}{2N}) \tag{1-39}$$

其中 $0 < n < 2N-1$。

将N点序列 $x(n)$ 和 $X(k)$ 按矢量形式表示如下:

$$\mathbf{x} = \begin{bmatrix} x(0) \\ x(1) \\ \cdots \\ x(N-1) \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} X(0) \\ X(1) \\ \cdots \\ X(N-1) \end{bmatrix} \tag{1-40}$$

公式（1-25）中的DCT变换可以按矩阵形式表示为:

$$\mathbf{\Lambda} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & \cdots & 1/\sqrt{2} \\ \cos(\frac{\pi}{2N}) & \cos(\frac{3\pi}{2N}) & \cdots & \cos(\frac{(2N-1)\pi}{2N}) \\ & \cdots & \cdots & \\ \cos(\frac{(N-1)\pi}{2N}) & \cos(\frac{3(N-1)\pi}{2N}) & \cdots & \cos(\frac{(2N-1)(N-1)\pi}{2N}) \end{bmatrix} \tag{1-41}$$

DCT和IDCT的系数可用下式计算:

$$\mathbf{X} = \mathbf{\Lambda x}, \quad \mathbf{x} = \tfrac{2}{N}\mathbf{\Lambda}^T\mathbf{X} \tag{1-42}$$

从而 $\Lambda\Lambda^T = \frac{N}{2}\mathbf{I}_{N\times N}$，其中 $\mathbf{I}_{N\times N}$ 表示大小为 $N \times N$ 的单位矩阵，因此DCT是一个正交变换。

公式（1-25）中的N点一维DCT需要 N^2 次乘法和加法运算。对于图像压缩，一个图像帧分成了 $N \times N$ 个块，对于每个块需要进行 $N \times N$ 点二维DCT变换。长度为 N 的二维DCT直接计算需要 N^4 次乘法和加法。另一方面，利用二维DCT的可分离性，即可先对图像块的行进行N点一维DCT，然后对得到的结果计算列的N点一维DCT[7]。这样简化之后，$N \times N$ 点二维DCT只需要 $2N^3$ 次乘加操作，或者 $4N^3$ 次算术操作。

16

1.2.7 矢量量化

矢量量化(Vector Quantization, VQ)起源于模式匹配(pattern matching)方法，目前通常应用于语音数据压缩、图像和视频编码以及语音识别。它是一种有损压缩技术，它探究空间上毗邻的信号采样值之间的相关信息。在VQ中，一组信号采样值并非单个进行量化，而是一起量化。一个矢量量化器能够直接对图像块操作，从而可以在得到空间信息压缩的同时进行量化。在VQ系统中，发送端和接收端都拥有相同的包含码字矢量的码本。矢量量化器并不传送码字本身，而是传送码字的索引。

图1-7显示了编码和解码的过程。在编码器端，矢量量化器收集到一组输入采样值（在图1-7的图像压缩情况中即为像素值）之后，将该输入矢量与码本中的码字进行比较，然后选择具有最小失真（distortion）的码字。假定矢量是 k 维的，码本大小为 N。如果矢量元素的字长为 W 并且 $N = 2^m$，那么所需传送的只是 m 比特的地址，而不是 kW 比特，从而压缩比为 m/kW。解码器只需接收 m 比特的索引作为码本的地址，然后取出最佳码字从而重建输入矢量。在图1-7中，每个矢量有 $k = 16$ 个像素，每个像素的字长为 $W = 8$。码本中有 $N =$

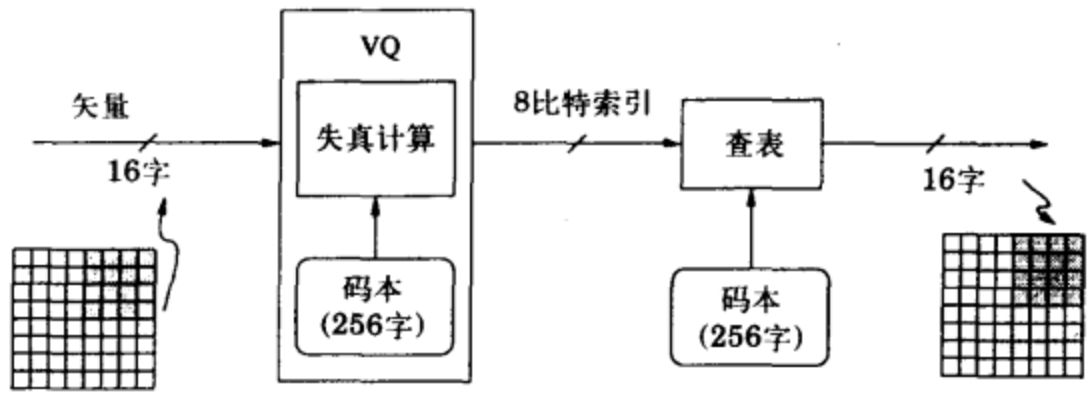


图1-7 基于矢量量化的矢量压缩和解压缩

256 个码字，那么 $m = 8$。因此图1-7所示的矢量量化器的压缩比为1/16。矢量量化器中的编码算法可以看作是一种穷尽搜索的算法。即依次计算码本中的每个码字矢量所对应的失真，并且保持跟踪与更新最小失真，直到测试完所有的码字矢量。通常两个矢量之间的欧几里德距离（也称作平方误差）可用来判定失真: 17

$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^2 = \sum_{i=0}^{k-1}(x_i - y_i)^2 \tag{1-43}$$

在实际的实现中，输入矢量 $\mathbf{x}$ 和第 j 个码字矢量 $\mathbf{c_j}$ $(0 \leqslant j \leqslant N-1)$ 之间的失真是通过计算它们的内积而不是进行平方运算得到的。展开公式（1-43），可得:

$$d(\mathbf{x},\mathbf{c_j}) = \|\mathbf{x}\|^2 - 2(\mathbf{x}\cdot\mathbf{c_j} + e_j) \tag{1-44}$$

其中

$$e_j = -\frac{1}{2}\|\mathbf{c_j}\|^2 = -\frac{1}{2}\sum_{i=0}^{k-1} c_{ji}^2 \tag{1-45}$$

内积由下式表示

$$\mathbf{x}\cdot\mathbf{c_j} = \sum_{i=0}^{k-1} x_i c_{ji} \tag{1-46}$$

因为 e_j 只与码字矢量 $\mathbf{c_j}$ 相关，是个常量，因此可以事先计算出来，看作是矢量 $\mathbf{c_j}$ 的附加成分。因此，对于一个固定的输入矢量 $\mathbf{x}$，在N个码字矢量中最小化公式（1-44）的失真等价于最大化 $\mathbf{x}\cdot\mathbf{c_j} + e_j$，其中 $0 \leqslant j \leqslant N-1$。所以VQ中的搜索过程可用下式描述:

$$ind_n = \Big(\min_{0\leqslant j\leqslant N-1} d_j\Big)^{-1} = \Big(\max_{0\leqslant j\leqslant N-1}\sum_{i=0}^{k-1}(x_i^n c_{ji} + e_j)\Big)^{-1} \tag{1-47}$$

其中求逆表示“把达到最大值或者最小值的索引 ind_n 输出”，n 表示时刻。搜索过程也可以等价地描述为下式，即先矩阵-矢量相乘，然后比较:

$$\begin{aligned} \mathbf{D} &= [d_0\ d_1\ \cdots\ d_{N-1}]^T = \mathbf{C}\mathbf{x} + \mathbf{e} \\ ind_n &= (\mathbf{max}\{d_i\})^{-1} \end{aligned} \tag{1-48}$$

其中 $\mathbf{C} = \{c_{ji}\}$ 是 $N\times k$ 的矩阵，其第 j 行即为第 j 个码字矢量 $\mathbf{c_j^T}$，$\mathbf{x}$ 是 k 维的输入矢量，

而 $\mathbf{e}=[e_0\ e_1\ \cdots\ e_{N-1}]^T$。

上述办法是一种“畜力劳动”(bruteforce)方法，每个输入矢量与每个码字之间的失真都要进行计算，因此称之为全搜索矢量量化（full-search vector quantitization）。每次全搜索操作需要进行 N 次失真运算，每次失真运算包含有 k 次乘加操作。因此，除去存储器读写操作之外，计算一个 k 维的输入矢量需要 Nk 次乘加操作和 $N-1$ 次比较。对于性能要求较高的情况，如果N值较大，该算法将成为瓶颈。对于这种情况，可采用树形矢量量化（tree-structured vector quantization）方法，它的复杂度正比于$\log_2 N$。如图1-8所示，该算法的基本思想是采用二进制序列搜索的办法，以代替全搜索。在树的每一级，输入矢量与两个码字矢量相比较，计算两个失真结果。重复该过程，直到达到树的底层叶子。对于图1-7中的例子，全搜索需要256次失真运算，而树搜索只需要16次。树搜索VQ是一种次优量化器，因此带来一定性能上的退化。但是对于精心设计的码本，可以尽量减少这种退化。

18

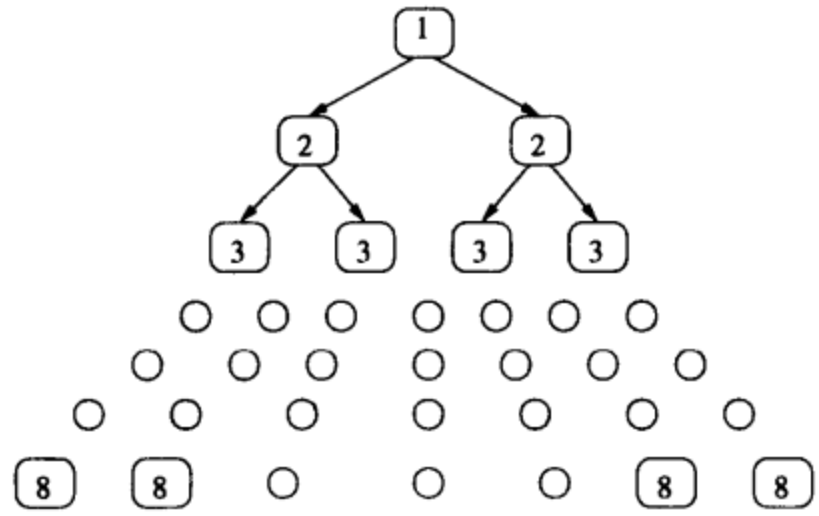


图1-8 树形矢量量化器

1.2.8 Viterbi算法和动态规划

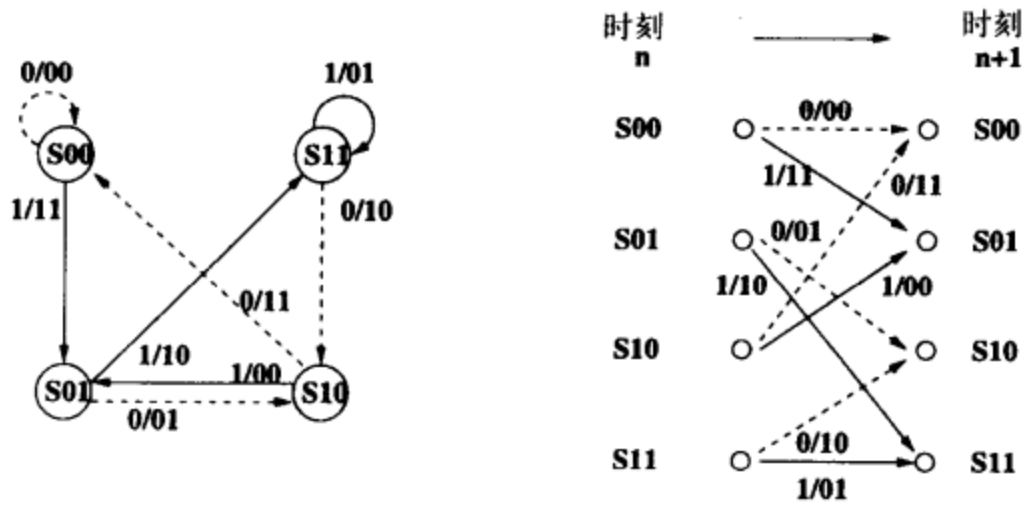
本节评述了基于动态规划的Viterbi算法[10]。Viterbi算法在序列误差控制码（如卷积码）和记忆性信道的符号检测上有广泛应用。给定一个被噪声干扰的有限状态序列，从最大似然角度来看，要寻找最似然的无噪声序列，采用该算法是最佳的。所谓有限状态信号是指从有限状态转换图中产生的信号。例如，图1-9a所示的即是一个状态转换图，它有四种状态，S00、S01、S10和S11。假定输入是0或者1，那么每种状态有两种可能输出边。每个边上标明了目前输入比特和对应的输出符号。比如从状态S00输出的边线上的标注是1/11，这表示目前的输入为1，输出的符号为11。这种状态转换图可以用来描述如下卷积编码算法：码率为1/2，存储器长度为2，生成多项式为 $g_1(z)=1+z^{-2}$ 和 $g_2(z)=1+z^{-1}+z^{-2}$。

19

图1-9a中的状态转换图也可以用与之等价的时间索引的格状图（Trellis Diagram）表示，如图1-9b所示。这种表示方法描述了从时刻 n 到 $n+1$ 时所有的状态转换。如果一个从网格中产生的符号序列被加性白噪声干扰，Viterbi算法采用欧几里德距离（最优）或者海明距离（次优）作为距离测度，能够在给定的网格中找到最近的符号序列。所得的序列被称为全局最大似然序列（global most-likely sequence）。

对于一个接收到的N个状态、包含有L个符号的序列 $\mathbf{v}$

$$\mathbf{v}=\{v(0),v(1),\cdots,v(L-1)\}$$



a) 4状态的状态机　　b) 网格表达

图1-9　有限状态机的网格描述

其中第一个符号 $v(0)$ 在时刻0接收到，最后一个 $v(L-1)$ 是在时刻 $L-1$ 接收到。Viterbi算法迭代地计算在时刻1到 $L-1$ 中进入每个状态的幸存路径（survivor path）。对于在时刻 n 的给定状态的幸存路径是指与时刻 n 所接收到的序列之间距离最小的符号序列；每个状态都赋予一个路径度量（path metric）$x_i(n)$，它用来衡量状态 i 上的幸存路径与时刻 n 所接收序列的距离。从时刻 n 到 $n+1$ 时（$0 \leqslant n \leqslant L$），Viterbi算法根据时刻 n 的幸存路径度量和在给定网格中的支路度量，更新幸存路径和路径度量 $x_i(n+1)$，它采用的方法是动态规划（dynamic programming, DP）:

$$x_j(n+1) = \min_i [x_i(n) + a_{ij}(n)],\ i, j = 1, 2, \cdots, N \tag{1-49}$$

其中 $a_{ij}(n)$ 表示在译码网格中 n 时刻 i 状态转换到 $n+1$ 时刻的 j 状态支路的度量，即当前接收到的符号 $v(n)$ 与编码网格中输出的符号之间的距离差。考虑图1-9b所示的一个4状态编码网格产生的序列。假定在时刻 n 时，四个状态的路径度量是: 20

$$x_1(n) = 2,\ x_2(n) = 0,\ x_3(n) = 1,\ x_4(n) = 2 \tag{1-50}$$

当前接收到的符号为 $v(n) = 11$。采用海明距离，网格中所有转换的支路度量可以计算为:

$$\begin{aligned}
a_{11}(n) &= weight(|11-00|) = 2 \quad & a_{12} &= weight(|11-11|) = 0 \\
a_{23}(n) &= weight(|11-01|) = 1 \quad & a_{24} &= weight(|11-10|) = 1 \\
a_{31}(n) &= weight(|11-11|) = 0 \quad & a_{32} &= weight(|11-00|) = 2 \\
a_{43}(n) &= weight(|11-10|) = 1 \quad & a_{44} &= weight(|11-01|) = 1
\end{aligned}$$

每个状态从时刻 n 到 $n+1$ 幸存路径和路径度量的更新采用图1-10所示的办法。每个状态都有两种可能进入的路径。其中具有较大度量的被抛弃掉，另外一个具有较小度量的被保留下来并被记入幸存路径中，如图1-10中较黑的线所示。这种更新过程迭代地重复进行，从 $n = 1$ 一直持续到 $n = L$。$n = L$ 时刻具有最小路径度量的状态对应的幸存路径就是全局最似然序列，即下面状态对应的幸存路径:

$$i = ind^{-1}(\ min_j\ \{x_j(L)\}\ ) \tag{1-51}$$

其中 ind^{-1}表示“取对应状态的索引”。动态规划算法的特点是：开始时的迭代到第 $n+m$ 次的迭代所得到的最优解必然包含从初始迭代到第 n 次迭代的最优解和从第 n 次迭代到第 $n+m$次迭代所得到的最优解。因此，从这种观点来看，该迭代算法是最优的。

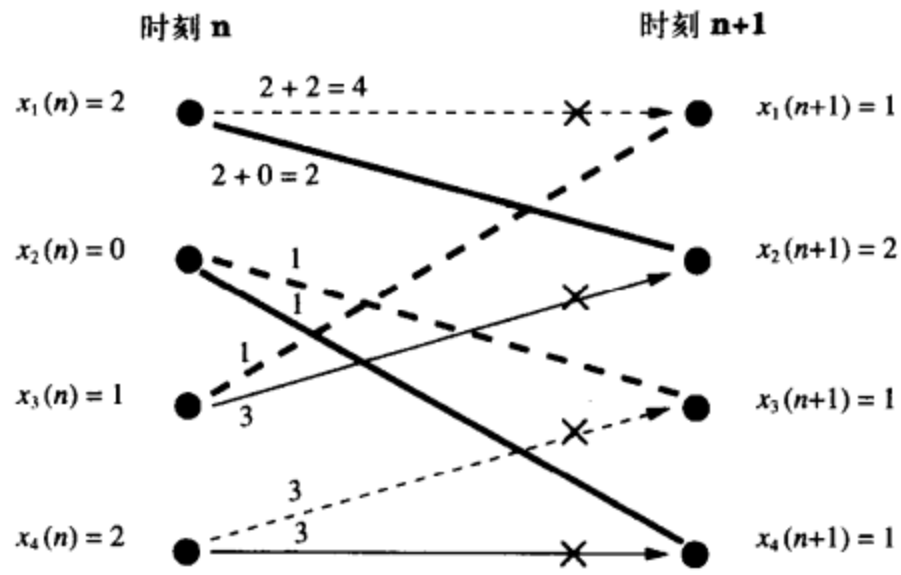


图1-10 从时刻 n 到 $n+1$的各个状态更新的幸存路径

图1-11是运行Viterbi解码算法的一个例子。输入序列为1 1 0 0 1 0 0 0，它采用图1-9b中所示的4状态网格进行编码，发送的序列为11 10 10 11 11 01 11 00。接收的序列被噪声干扰从而与发送的序列有3个符号不同。图1-11显示了从时刻0到8的解码网格，其中标有“x”的支路是被抛弃的支路。全局最似序列是在时刻 $n=L=8$时第一个状态的幸存路径。全局最近似序列在解码网格中已用较黑的线标出，可以看出它与发送的序列完全一致。

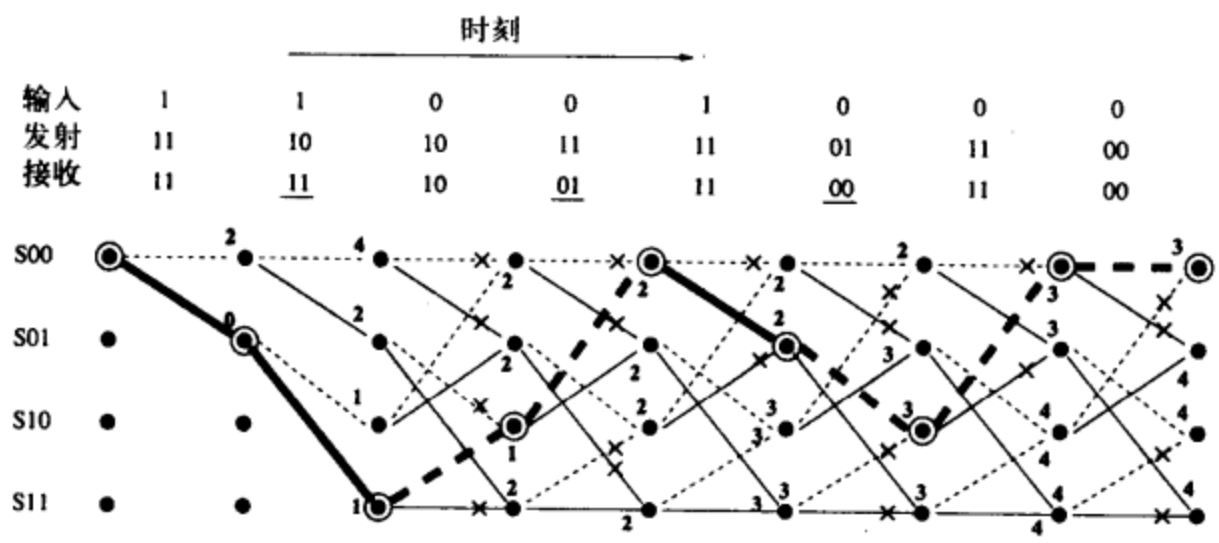


图1-11 图1-9中所示的状态图产生的有限状态序列的Viterbi解码例子。

Viterbi算法包括计算支路度量 $a_{ij}(n)$、更新路径度量、选择最后状态以及回溯幸存路径，其中更新路径度量需要在每个时刻对每个状态进行加法、比较以及选择（addition, comparison and selection, ACS）。Viterbi解码器的速度主要受到递归的ACS操作的限制。

1.2.9 抽取器和扩展器

一个DSP系统的采样率可以通过内部的减采样（downsampling）和增采样（upsampling）操作来改变，这样的系统称为多速率系统。多速率信号处理的应用包括语音和图像压缩、数字音频系统、统计和自适应信号处理等等。抽取器和扩展器是多速率系统中最基本的两种功能块。

抽取器处理输入 $x(n)$，产生输出序列:

$$y_D(n)=x(Mn) \tag{1-52}$$

其中 M 为正整数。抽取器也可称为压缩器（compressor）和减采样器（downsampler）。抽取器的符号及其功能描述在图1-12中。抽取器对于每 M 个输入样值产生一个输出样值，因此输出速率是输入序列的 $1/M$。

21～22

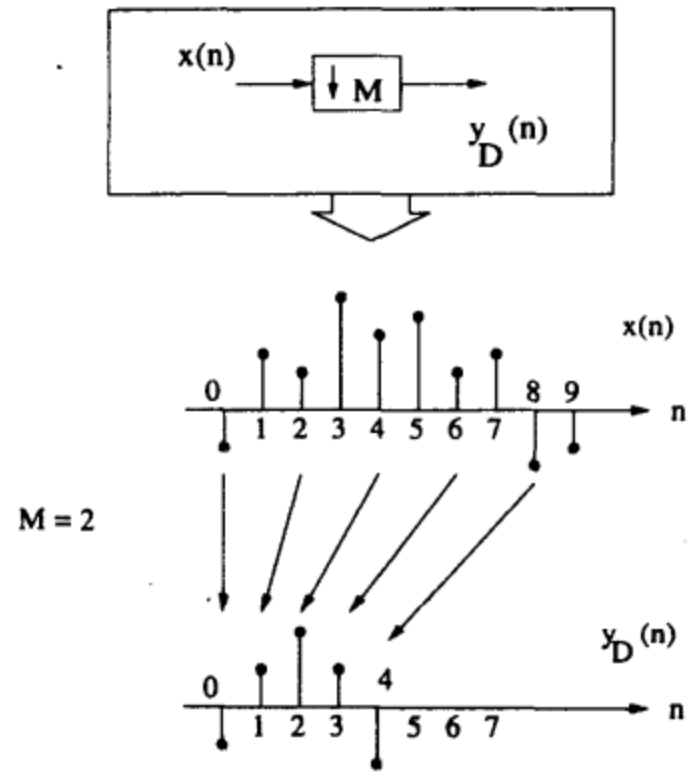


图1-12 抽取器及其功能

扩展是一增采样过程。它取一个输入样值 $x(n)$，产生一个输出序列

$$y_E(n) = \begin{cases} x(n/L), & \text{如果}n\text{是}L\text{的整数倍} \\ 0, & \text{其它} \end{cases} \tag{1-53}$$

其中 L 是整数。扩展器的符号及其功能描述在图1-13中。扩展器也可称作插入器（interpolator）或增采样器（upsampler）。扩展器对于每个输入的样值，通过添加 $L-1$ 个0，输出 L 个样值。因此输出序列的采样率是输入序列的 L 倍。

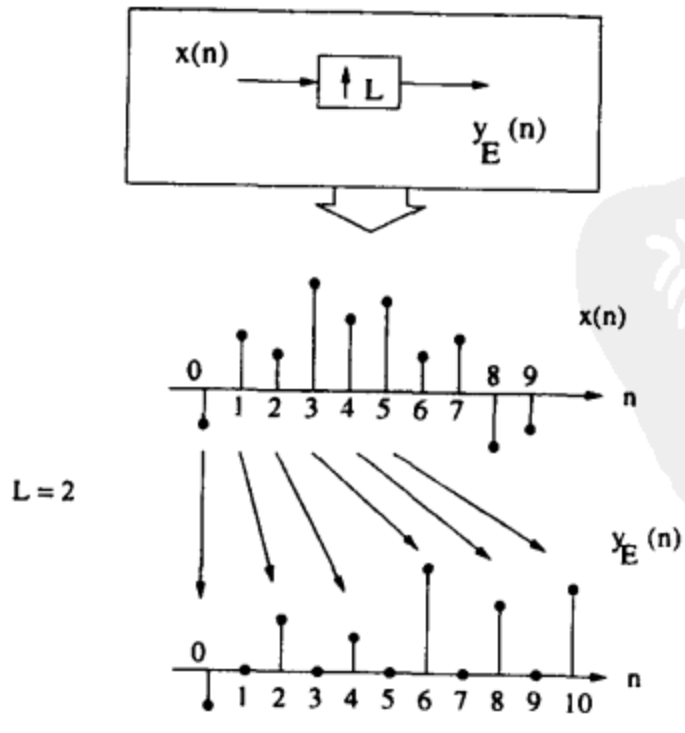


图1-13 扩展器及其功能

抽取和插入是非线性操作。一个 M 层(M-fold)的抽取器后级连一个 L 层的插入器与一个 L 层的插入器后级连一个 M 层的抽取器通常是不等价的。为了分析多速率系统，我们采用了几种等价方式：惰性恒等变换（Noble Identity）用来将抽取器/扩展器输入端的延时单元转移到输出端，如图1-14所示。注意 n 个延时单元经过减采样后等价于 n/M 个延时单元，而经过增采样后等价于 nL 个延时单元。

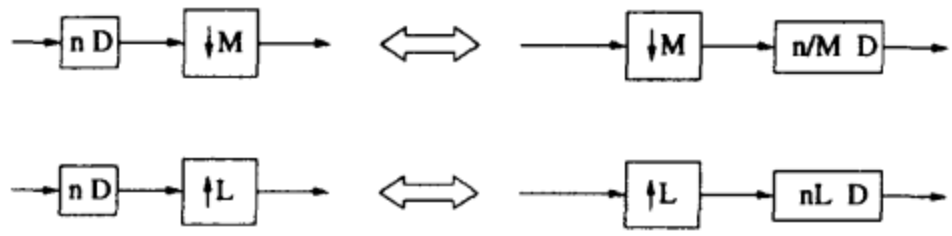


图1-14 用于多采样系统的惰性恒等变换

1.2.10 小波和滤波器组

小波变换用于语音和图像压缩[12][13]。在小波分析中，信号用一组基函数（小波）表示，这组基函数是通过对一个称之为“母函数”的原型函数进行时间上的平移和缩放而导出来的。小波变换可以看成是信号在时间-尺度（频率）平面上的分解。

$x(n)$ 的一维离散小波变换（DWT）定义为：

$$\begin{aligned} y_i(n) &= \sum_{k=-\infty}^{\infty} x(k)h_i(2^{i+1}n-k),\ 0 \leqslant i \leqslant m-2 \\ y_{m-1}(n) &= \sum_{k=-\infty}^{\infty} x(k)h_{m-1}(2^{m-1}n-k),\ i=m-1 \end{aligned} \tag{1-54}$$

23 ~ 24 其中母小波函数 $h(n)$ 的平移和缩放版本 $\{h_i(2^{i+1}n-k), 0 \leqslant i \leqslant m-1, -\infty < k < \infty\}$ 即为基函数，$y_i(n)$ 是小波系数（wavelet coefficient）。反变换如下计算：

$$\begin{aligned} x(n) &= \sum_{i=0}^{m-2} \sum_{k=-\infty}^{\infty} y_i(k)f_i(n-2^{i+1}k) \\ &\quad + \sum_{k=-\infty}^{\infty} y_{m-1}(k)f_{m-1}(n-2^{m-1}k) \end{aligned} \tag{1-55}$$

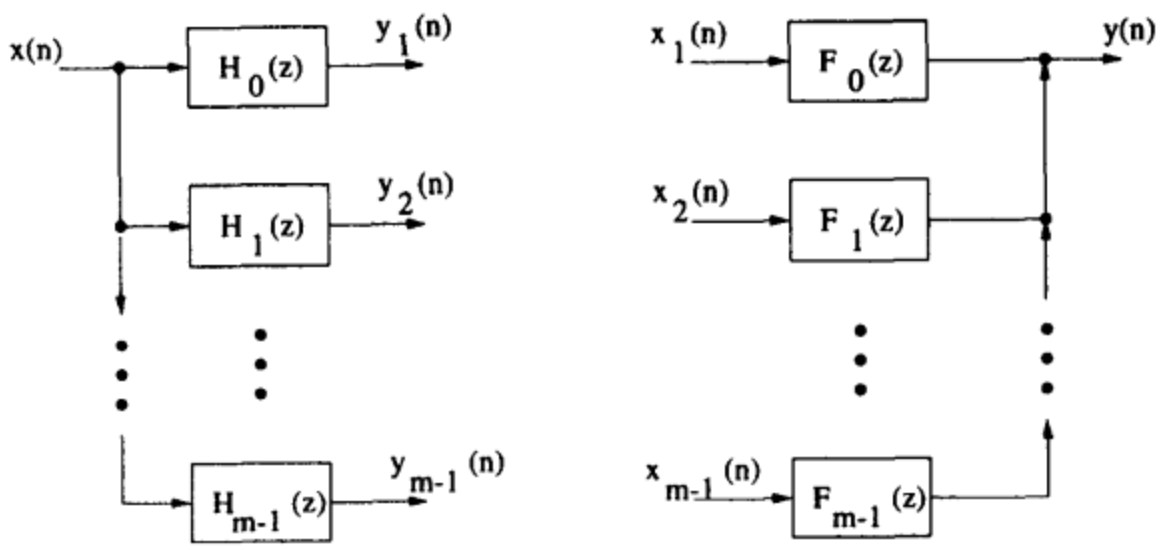
其中 $\{f_i(n-2^{i+1}k)\}$ 构造的原则是使公式（1-55）能够完全重构原有信号 $x(n)$。注意DWT和DWT反变换（IDWT）的计算类似于卷积操作。事实上，DWT和IDWT可以作为一系列的卷积和抽取进行递归运算，所以可以用滤波器组实现。

数字滤波器组（filter bank）是指具有共同输入（称为分析滤波器组，analysis filter bank）或者共同输出（称为合成滤波器组，synthesis filter bank）的一组滤波器。图1-15表示出了分析滤波器组和合成滤波器组的框图。滤波器组通常用作子带（subband）编码。单独的信号 $x(n)$ 在分析滤波器组中分成了 m 个子带信号；在合成滤波器组中，m 个输入子带信号合并起来重构信号 $y(n)$。

考虑采用滤波器组计算 $m=4$ 时的离散小波变换。小波系数

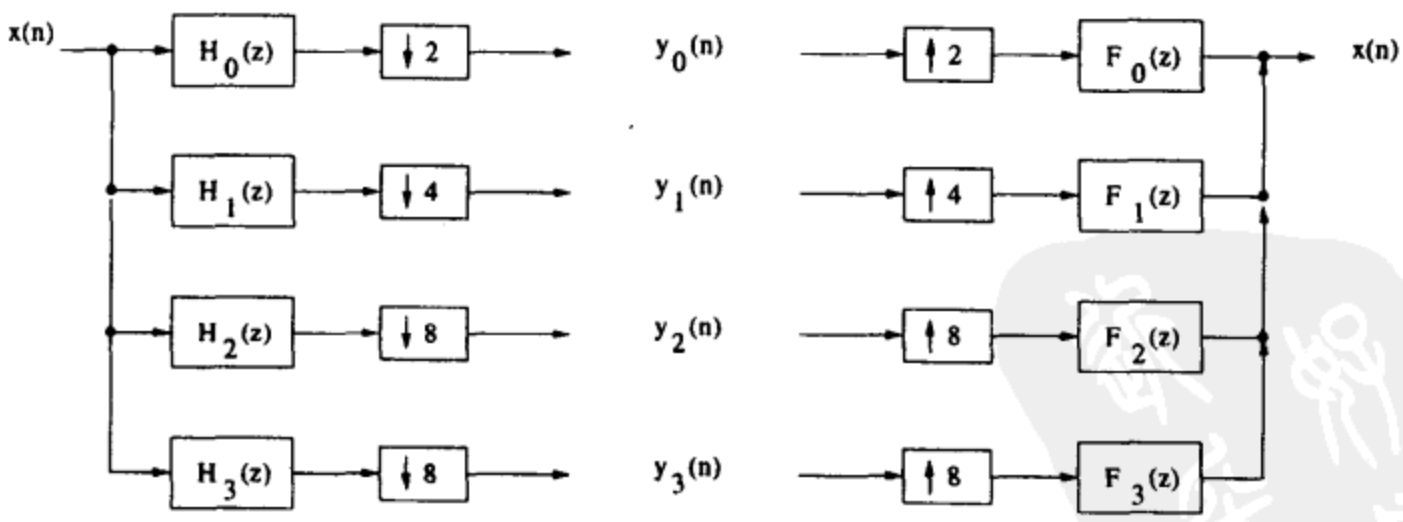
$$
\begin{aligned}
y_0(n) &= \sum_{k=-\infty}^{\infty} x(k)h_0(2n-k) \\
y_1(n) &= \sum_{k=-\infty}^{\infty} x(k)h_1(4n-k) \\
y_2(n) &= \sum_{k=-\infty}^{\infty} x(k)h_2(8n-k) \\
y_3(n) &= \sum_{k=-\infty}^{\infty} x(k)h_3(8n-k)
\end{aligned}
\tag{1-56}
$$

可用带有图1-16a所示抽取器的分析滤波器组进行计算。然后，通过图1-16b中的插入器和合成滤波器组，可以进行小波反变换来重构信号 $x(n)$。



a) 分析滤波器组　　b) 合成滤波器组

图 1-15



a) 用于DWT和IDWT的分析滤波器组　　b) 用于DWT和IDWT的滤波器组

图 1-16

实际上，离散小波变换每次周期性地处理 M 个输入样值，并在各个频带上输出 M 个样值，其中 $M = 2^m$，m 表示频带数目或者小波的级数。通常采用*树形结构*（tree-structured）的滤波器组来实现，其中 M 个小波系数可用$\log_2 M$个倍频级计算，同时每个倍频级完成一次

低通滤波和一次高通滤波操作。在某个倍频级 j，输入序列 $s_{j-1}(n)$ 分别输入到低通滤波器 $g(n)$ 和高通滤波器 $h(n)$ 中。来自高通滤波器 $h(n)$ 的输出表示原有信号在给定的级 j 中的细节信息，用 $w_j(n)$表示。来自低通滤波器 $g(n)$ 的输出表示原有信号的剩余（粗，coarse）信息，用 $s_j(n)$ 表示。倍频级 j 中的计算可以表示为:

25 ~ 26

$$\begin{aligned} s_j(n) &= \sum_k s_{j-1}(k)g(2n-k) = \sum_k g(k)s_{j-1}(2n-k) \\ w_j(n) &= \sum_k s_{j-1}(k)h(2n-k) = \sum_k h(k)s_{j-1}(2n-k) \end{aligned} \tag{1-57}$$

其中 n 表示样值索引，j 表示倍频级索引。开始时，$s_0(n) = x(n)$。图1-17所示的是3级树形DWT的框图。值得指出的是，因为每个滤波后的输出都是隔2抽1产生的，所以只有那些没有被抛弃的信号需要进行计算。比如考虑 $m = \log_2 M$ 级小波，其输入序列有 M 个样值（假定 M 是2的幂）。第一个倍频级输出的样值总数为 M 个（高通滤波器产生 $M/2$ 个输出，低通滤波器产生 $M/2$ 个输出）；第二个倍频级产生的样值总数为 $M/2$；第三个倍频级产生 $M/4$ 个样值，等等。因此对于这个 m 级的小波，每 M 个输入，需要计算的样值总数为:

$$M + M/2 + M/4 + \cdots + 2 = 2(M-1) \tag{1-58}$$

在对 M 个输入样值进行周期小波计算时，每个采样周期中计算的样值数目为 $2(M-1)/M$ 或者是 $2(1-1/M)$ [14]。

图1-16a和图1-17a中的两个分析滤波器组的功能是相同的。采用惰性恒等变换，图1-17a中的分析滤波器的传输函数可以用 $h(n)$ 和 $g(n)$ 如下表示:

$$\begin{aligned} H_0(z) &= H(z) & H_1(z) &= G(z)H(z^2) \\ H_2(z) &= G(z)G(z^2)H(z^4) & H_3(z) &= G(z)G(z^2)G(z^4) \end{aligned} \tag{1-59}$$

其中 $H_0(z)$ 是高通滤波器，$H_1(z)$ 和 $H_2(z)$ 是带通滤波器，$H_3(z)$ 是低通滤波器。这些滤波器的通带宽度从 $H_0(z)$ 到 $H_3(z)$ 逐渐下降，因此小波变换也可看作是非均匀子带编码。

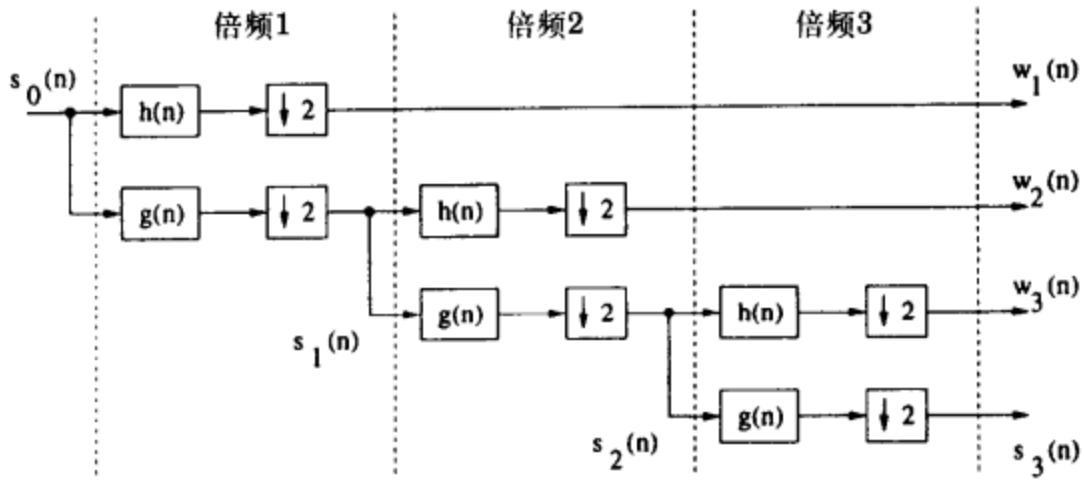
1.3 DSP应用需求和按比例缩小的CMOS工艺

对DSP应用（尤其是在多媒体和无线通信方面）需求的增加以及按比例缩小的CMOS工艺的进展，持续地推动了VLSI DSP领域的发展。本节介绍在数字视频处理应用以及按比例缩小的CMOS VLSI工艺中对计算提出的需求。视频处理系统需要10~100千兆操作数/秒（Gigaoperations per second，GOP / s）的运算速度，因此，设计这种系统将极富挑战性!

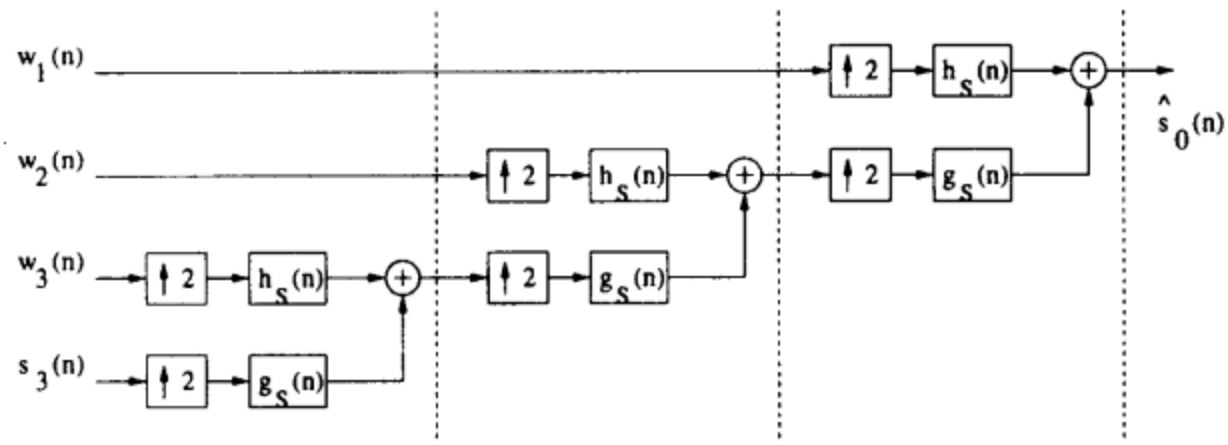
27

彩色数字图像由图形元素（像素）组成，像素是用三原色即红（R）、绿（G）、蓝（B）来表示的。根据人类视觉系统，RGB表示可以转换为 *YUV* 表示。其中 Y 表示亮度，U 和 V 分别表示 Y 与蓝色、Y 与红色之间的色差，称之为色度。*YUV* 的全采样表示为4：4：4的采样，采样所得的像素用24比特表示，其中每个变量8比特。在4：4：4的采样情况下，对于一个大小为288 × 352个像素、帧速率为30帧/秒的通用媒介格式（common intermediate format，CIF）的帧，每一个帧需要2.433Mb的存储容量以及72.99Mb/s的视频源数据速率。对于高清晰电视（High-Definition TV， HDTV）视频，每个帧大小为1920 × 1250个像素，帧速率为50帧/s，每帧需要57.6Mb的存储容量以及2.88Gb/s的视频源数据率。从目前的技术

来看，实时存储并传输这样含有成百上千帧的视频序列是不可能的。实际中，在存储或传输之前，视频像素要经过减采样，同时视频帧也要进行压缩。



a) 树形分析滤波器组的DWT



b) 合成滤波器组的DWT

图 1-17

人眼对色彩的空间分辨率比亮度低，感知度（receptor）也比亮度差。因此色度可以通过减采样来减少源数据率和帧存储空间。通常进行4：2：2或者4：2：0这样的采样。4：2：2采样时，亮度 Y 在每个像素上都采样，而色度 U 和 V 在水平方向上每隔一个像素采样一次，从而节省了33%。4：2：0采样时，U 和 V 在水平和垂直方向上都只采样一半，从而节省50%。 28

除了减采样外，视频压缩是降低源数据流带宽要求的关键。一般说来，视频序列的单个帧包含有相当多的空间冗余性，而在相继帧之间包含有相当多的时间冗余性。MPEG是一个由运动图形专家组（Moving Picture Experts Group）开发的视频通信标准。这个标准通过帧内和帧间的编码技术，发掘空间和时间上的冗余性，将比特率降低。MPEG标准的实际目的是根据指定的比特率，优化图像和视频的质量，满足“客观”或者“主观”优化的指标[15]。图1-18示出了MPEG-2编码过程的框图。在时间上进行运动-补偿预测之后，对剩余的空间信息进行变换编码，然后对传送的序列进行变换编码，以获得高的数据压缩率[16]。

采用运动－补偿视频压缩技术，数字视频序列能够实时地进行存储和传送。但是，这些复杂的压缩技术带来了大量的高速运算。比如全搜索块匹配算法的复杂度正比于$3(2p+1)^2N_hN_vF$ GOP/s，其中 $N_h \times N_v$ 是帧大小，$+/-p$ 是搜索范围，F 表示帧速率，单位是帧/秒。对于一个大小为288 × 352像素、帧速率为30帧/秒、搜索范围为+/-7像素的CIF帧，全搜索BMA需要大约

2 GOP/s。更高的分辨率、更高的帧速率以及更大的搜索面积需要更多的操作。对于帧大小为1920 × 1250像素的HDTV视频，帧速率为50帧每秒，搜索面积为+16/−15像素，全搜索BMA需要368.64 GOP/s的计算速度。视频通信中的DCT对计算要求也很高。$N \times N$ 二维DCT需要 $2N^3$ 次乘加运算或者 $4N^3$ 次算术操作。对于图像块大小为8 × 8的CIF帧，二维DCT所需的计算是97.32 MOP/s（megaoperations per second）。对于图像块大小为8 × 8的HDTV视频，二维DCT所需的计算是3.84 GOP/s。只有仔细地设计硬件和软件，用并行处理技术才能满足这么高的处理要求。主要的挑战来自于视频压缩以及多媒体信号处理系统的设计和实现。

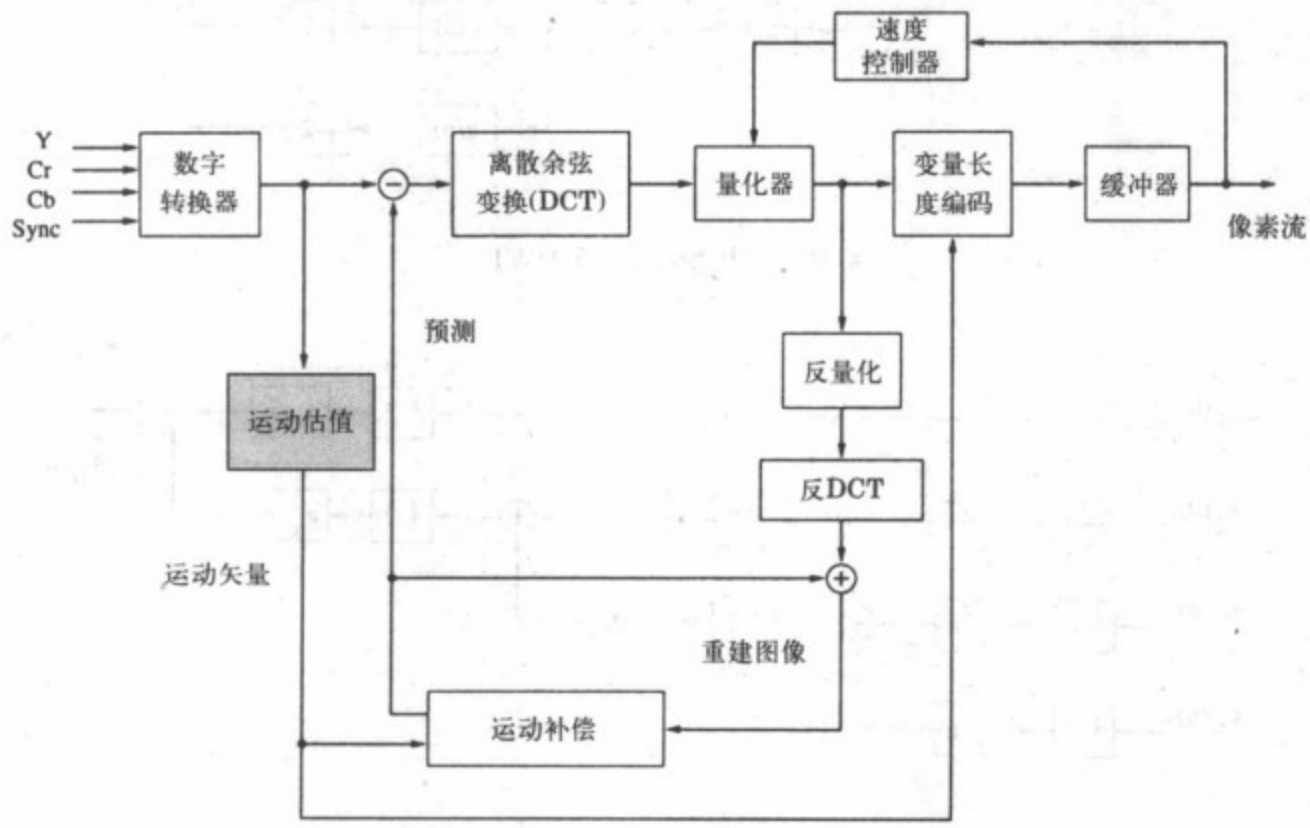


图1-18 MPEG-2编码器的框图

不仅仅是理论上的进展，尺寸按比例缩小的CMOS VLSI工艺的迅速进展也推动了数字信号处理复杂度的增加。近年来集成度的增加引人注目，越来越多的DSP应用能够集成于单片之上。前些年人们对提高集成度的兴趣主要集中在为高性能DSP应用设计专用（或者面向具体应用）的结构。如今随着更为复杂的DSP和图像/视频处理算法需要实现在单个芯片上，这一趋势还要持续下去。例如，目前的亚微米工艺能在单片上完成实时的MPEG-2运动估值（大约需要10到30GOP/s）。一个例子是NEC的芯片[17]，它包含3.7M个晶体管，峰值性能可达12GOP/s。随后另外一种趋势开始获得越来越多的动力，即DSP系统采用可编程DSP和专用于多媒体处理的微处理器来实现。这种发展趋势主要源自于对缩短进入市场时间以及更多的便携性和灵活性这两方面的要求。

VLSI工艺尺寸的不断缩小，使得设计复杂的多媒体系统成为可能。预计CMOS工艺将会缩小到0.07μm的器件，表1-2列出了相应的在密度、速度以及价格上的改善。注意单片上晶体管数量预计从目前的6M到2010年将增加到90M。大视场的光刻技术是按比例不断缩小的主要难点。如果这一点无法实现，很难做到高成品率和低成本。光学光刻技术可以用到0.25μm以下的工艺。但是在0.18μm工艺以下，光刻系统需要做较大的变化。由于接触电阻和统计上越来越显著的掺杂分布起伏，实际器件的可用性限制从0.05μm尺度开始面临严峻的挑战。

29 ~ 30

表1-2 按比例缩小的CMOS工艺的特性

第一个DRAM出货的年份	1998	2001	2004	2010
最小特征尺寸（μm）	0.25	0.18	0.13	0.07
存储器				
每片上的比特数（DRAM/Flash）	256M	1G	4G	64G
量产时每比特的成本（毫美分）	0.007	0.003	0.001	0.0002
逻辑（高产量：微处理器）				
每平方厘米上的逻辑晶体管数目（布满的）	7M	13M	25M	90M
每平方厘米上的比特数（缓存SRAM）	6M	20M	50M	300M
量产时每个晶体管的成本（毫美分）	0.5	0.2	0.1	0.02
逻辑（低产量ASIC）				
每平方厘米上的晶体管数目（自动布图）	4M	7M	12M	40M
一次性工程费用的每个晶体管的成本（毫美分）	0.1	0.05	0.03	0.01

观察图1-19所示的应用发展也是很有趣的。在这张图中示出了原来主要用专用集成电路（Application-Specific Integrated Circuits，ASIC）实现的一些应用，比如桌面视频等，现在正在采用可编程DSP（PDSP）来实现。因此从应用的性质上看存在着一种从ASIC到PDSP，到微处理器，再到现场可编程逻辑阵列（FPGA）的普遍趋势。从另一个角度来看，正是因为工艺的提高，对于给定的某个功能，“最好”的实现方式将随着时间由ASIC变到了DSP，到微处理器，再到FPGA。

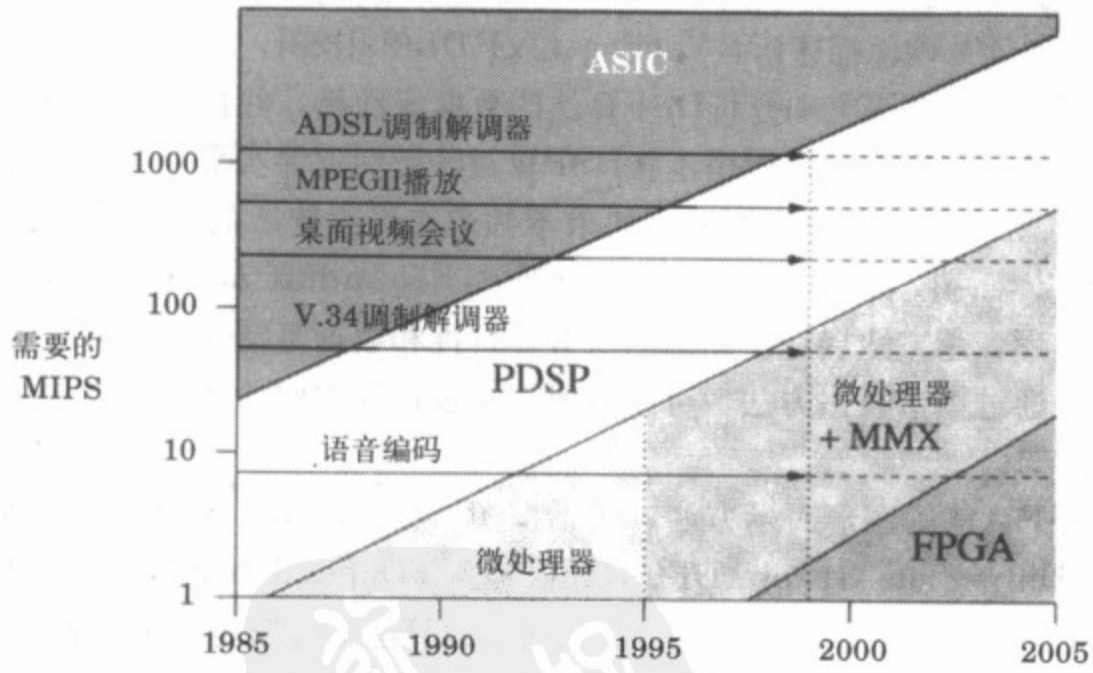


图1-19 应用的演化

1.4 DSP算法的表示

DSP算法可以用无休止程序（nonterminating program）来描述，这种方法重复执行同样的代码。例如，一个3阶FIR数字滤波器用无休止程序描述为：

$$y(n) = ax(n) + bx(n-1) + cx(n-2) \qquad n = 1 \text{至} n = \infty \tag{1-60}$$

将算法中的所有计算运行一次称为一次迭代（iteration）。迭代周期（iteration period）是

指算法中进行一次迭代所需的执行时间。*迭代率*（iteration rate）是迭代周期的倒数。每次迭代期间，公式（1-60）中的3阶FIR滤波器对一个输入信号进行处理，完成3次乘法和2次加法操作（串行或者并行），然后产生一个输出样值。DSP系统也可以用*采样率*（也称作*吞吐率*）来表征，它是指每秒处理的样点数目。组合逻辑的*关键路径*（critical path）定义为输入与输出之间的最长路径，在这里路径的长度正比于它的运算时间。DSP系统通常用时序电路实现，则此处关键路径定义为任何两个存储单元（或者延时单元）之间的最长路径。关键路径的运算时间确定了DSP系统最小可行的*时钟周期*（clock period）。*迟滞*（latency）定义为产生输出的时间与系统接收对应输入的时间之差。对于只有组合逻辑的系统，迟滞通常按绝对时间单位或者门延时数来表示；对于时序系统，迟滞通常按时钟周期数量表示。一般情况下，DSP系统的时钟速率与其采样率并不相等。

在描述系统的功能比描述内部操作的顺序和结构更为重要的地方，DSP算法可以用更高层次的数学化的公式表达来描述。对于架构设计，这种数学化的公式表达需要转换为行为级描述语言或者是用图形表示。

行为级描述语言包括应用性语言、说明性语言以及叙述语言。*应用性语言*（applicative language）表示满足算法中变量之间关系的一组方程，而不是一系列的动作（要执行的任务，即任务描述的顺序并不重要）。应用性语言在DSP系统的描述中是很普遍的。比如用于Cathedral综合工具[21]中的Silage语言[20]就是应用性语言。*说明性语言*（prescriptive language）明确地指定任务描述的顺序。这类语言的例子包括高层次编程语言比如Pascal、C或Fortran。更近的一段时间，很多应用采用了*叙述语言*（descriptive language），它描述了DSP系统的结构。比如硬件描述语言Verilog 和VHDL[22][23]。

31 ~ 32

图形表示方法可以有效地研究和分析DSP算法的数据流性质，并揭示不同子任务中内在的并行性。更为重要的是，图形表示能用于将DSP算法映射到硬件实现上。因此，这种表示方法可以把算法描述与结构实现连接起来。如果不知道实现系统所需的支持工艺，无法对DSP系统的性能指标：面积、速度、功耗以及舍入噪声（roundoff noise）等进行绝对意义上的衡量。但是，图形表示方法展示了系统的所有并行性和数据驱动性能，同时还能看出空间和时间上的折中。通过图形表达方法，可以看出架构设计的空间，并且可以采用与工艺无关的方式选出合适的架构。

本节介绍4种DSP算法的图形表示方法，包括框图、信号流图（Signal-Flow Graph, SFG）、数据流图（Data-Flow Graph, DFG）和依赖图（Dependence Graph, DG），上述这些都是直接用图形表示的。它们在不同的抽象层次上对算法进行描述。通常情况下，DG将算法中的内在并行性和数据流约束尽可能地展现出来，而不去考虑结构。DG中的并行性可以通过将其映射到SFG或者DFG上，以各种方式展示出来。实际上，DG用于脉动阵列的设计。SFG和DFG用于分析结构性能，并通过高层次的变换来探索架构上的各种选择。

1.4.1 框图

框图经常用于图形化地描述DSP系统。第1.2节中的所有DSP算法都是用框图描述的。框图由功能模块以及将其连接起来的带有方向的边组成的，这些边代表数据从输入模块到输出模块的流动。这些边可以包含延时单元,也可以不包含延时单元。对于所有的系统，都可以在不同的抽象层次上来构建框图。

比如3阶FIR滤波器

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) \tag{1-61}$$

可以用图1-20中的框图描述。图中包含有两种功能模块：加法器和乘法器。单位延时单元也可以处理为一种功能模块，因为它们在实际中是用寄存器实现的。在图1-20中，单位延时单元表示为$\boxed{z^{-1}}$（或$\boxed{D}$）。 33

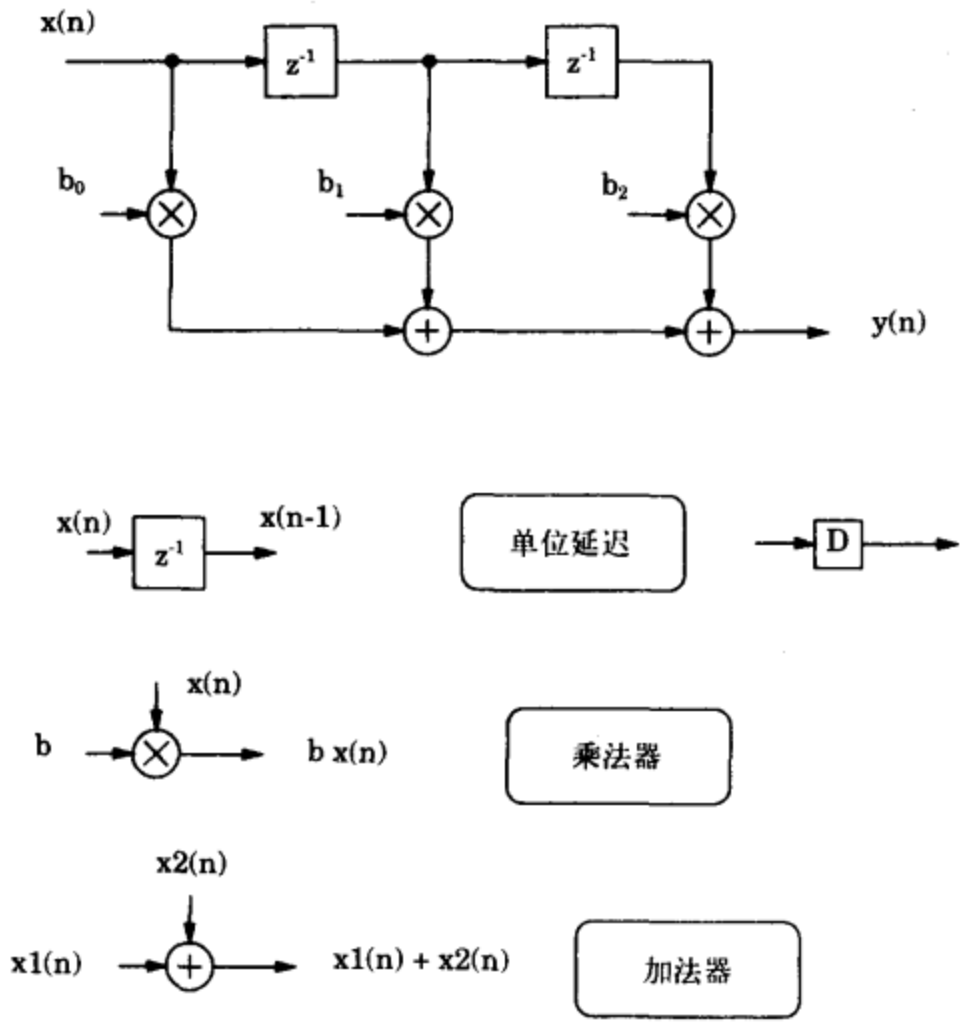


图1-20 3阶FIR滤波器的框图

一个系统可以用不同的框图表示，每种表示意味着相同功能的不同实现方法。比如公式（1-61）中的3阶FIR滤波器也可以用图1-21中的框图表示，这种结构称为数据广播（data-broadcast）结构。

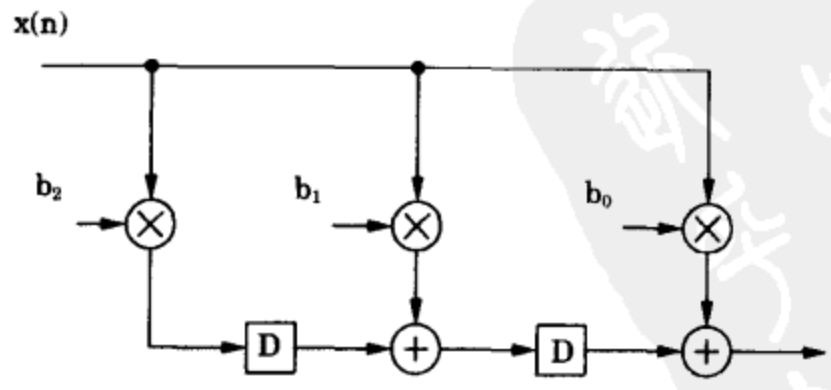


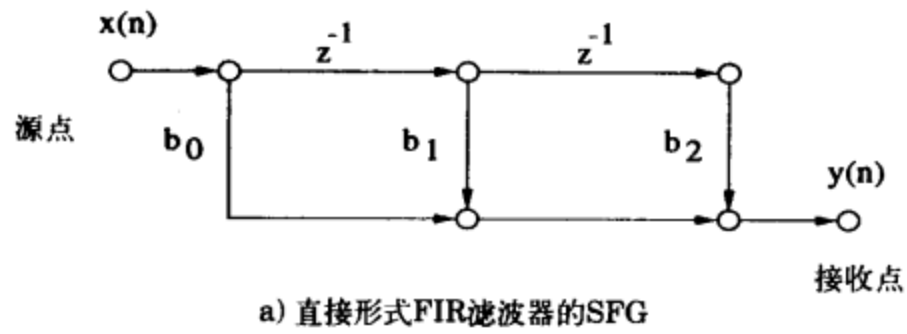
图1-21 3阶数据广播FIR滤波器的框图

框图是描述系统确切功能的一种象形模型，能清楚地表达出每个信号和功能单元。同一个系统可以具有不同的框图，它们具有不同的排列，从而有不同的实现方法。

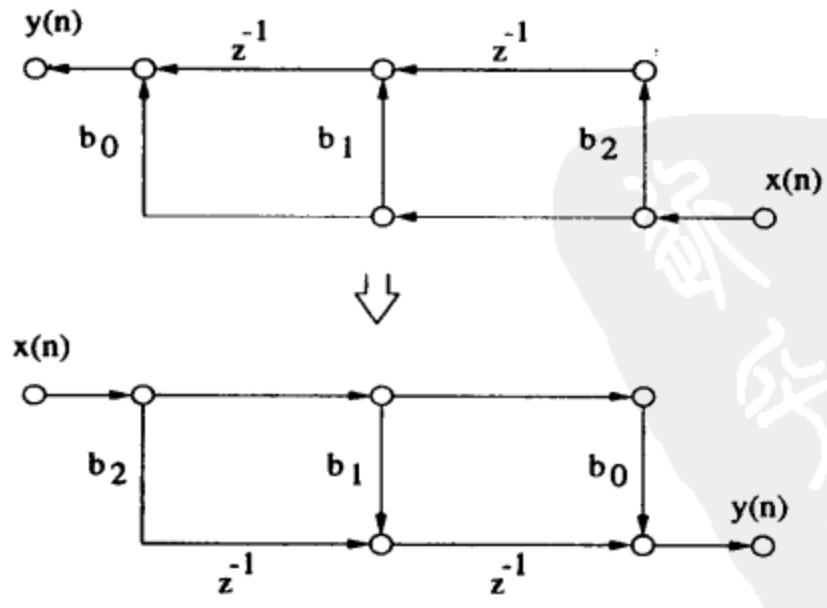
1.4.2 信号流图

信号流图（SFG）是节点和有向边的集合[25]。节点表示运算或者任务。一条有向边 (j, k) 表示从节点 j 出发到节点 k 结束的支路。对于节点 j 处的输入信号以及节点 k 处的输出信号，边线 (j, k) 表示节点 j 处的信号到节点 k 处信号的线性变换。SFG已经广泛用来分析、表示以及评价线性数字网络，尤其是数字滤波器结构。在数字网络中，边通常限制在常数增益乘法器（边的输出是输入信号的常数倍），或者延时单元（边的输出是输入信号的延时之后的版本）。加法器和乘法器用带有多个输入边和单个输出边线的节点表示。在SFG中有两种特殊节点，源节点和汇节点。*源节点*（source node）指没有输入边的节点，它用来表示向信号流图注入外部输入信号。*汇节点*（sink node）指只有输入边的节点，用来表示从信号流图提取输出。一个3阶FIR滤波器（与图1-20中的框图对应）的直接形式的SFG示于1-22a，其中没有标注的边表示单位增益操作（恒等变换）。注意这个SFG中的源节点和汇节点与信号流图其余部分通过单位增益边相连，以便清楚地表示系统的输入和输出。

线性SFG在不改变系统功能的情况下可以变换为不同的形式。*流图反转*（flow graph reversal）或*转置*（transposition）是其中的一种。它可用于单输入单输出系统，获得等价的转置结构。按照如下方式对SFG进行转置：将所有边反向，然后交换输入和输出节点并保持边的增益和延时不变，产生的SFG保持同样的系统功能。图1-22b展示了从直接形式的SFG推出转置FIR滤波器的SFG（对应于图1-21中的数据广播框图）的过程。读者可以验证图1-1b中的数据广播对称FIR滤波器可以从图1-1a经过转置导出。对于用对称变换矩阵（symmetric transformation matrix）描述的多输入多输出（multi-input multi-output, MIMO）系统，转置操作也是可用的（见第9章）。



a) 直接形式FIR滤波器的SFG



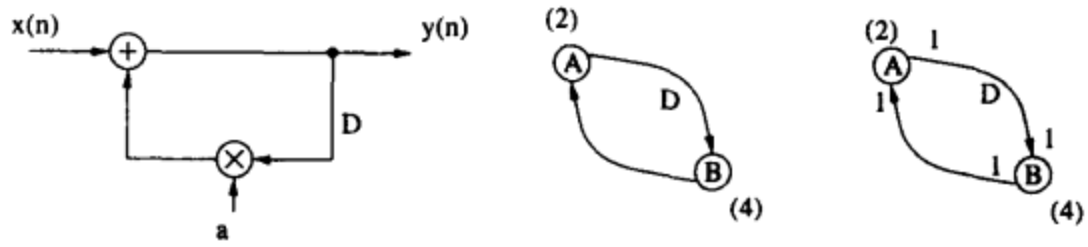
b) 转置后分FIR滤波器的SFG

图1-22 3阶FIR滤波器的SFG

SFG提供了线性网络的一种抽象流图表示方法，已经广泛地应用在数字滤波器结构设计以及有限字长效应的分析中。通常SFG只能用于线性网络，无法用来描述多采样率DSP系统。

1.4.3 数据流图

在数据流图（DFG）表示方法中，节点表示运算（或功能、或子任务），有向边表示数据通路（节点之间的通信），每条边有一个非负的延时数与之相关。比如图1-23b中是运算 $y(n) = ay(n-1) + x(n)$ 的数据流图，其中节点 A 表示加法，节点 B 表示乘法，从 A 到 B（用 $A \rightarrow B$ 表示）的边带一个延时，从 B 到 A 的边（用 $B \rightarrow A$ 表示）不带延时。与节点相关的还有它的执行时间，执行时间的单位采用归一化的时间单位（u.t., units of time）。比如节点 A 的执行时间为2 u.t.，节点 B 的执行时间为4 u.t.，如图1-23b所示。这个DFG的迭代周期等于6 u.t.。



a) 运算 $y(n) = ay(n-1) + x(n)$ 的框图表示　b) 常规的DFG表示　c) 同步DFG表示

图 1-23

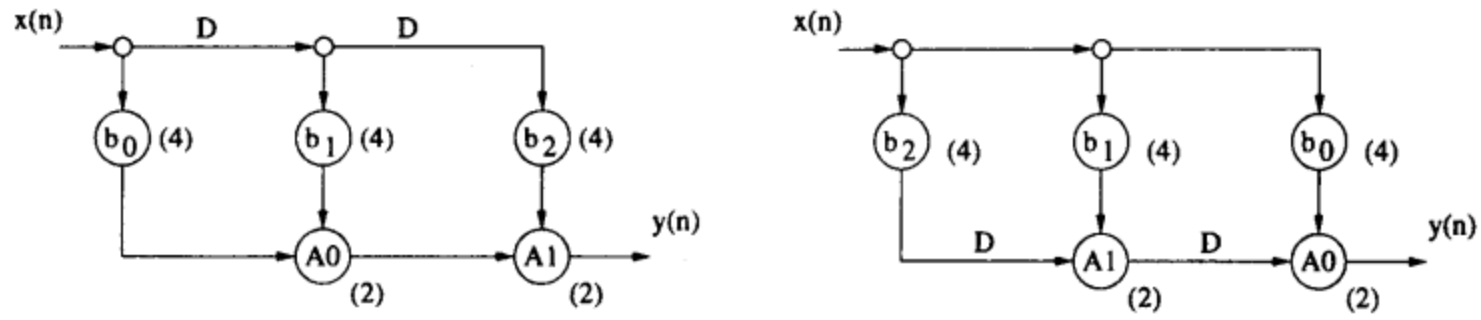
数据流图捕捉DSP算法的数据驱动（data-driven）性质，数据流图中的任意节点一旦它的所有输入数据准备好时就可以启动（fire, i.e. perform its computation，进行运算）。这暗示一个没有输入边的节点可以在任何时候启动；因此很多节点可以同时启动，而具有并发性。反过来说，一个带有多条输入边的节点只有所有的前节点都启动之后才能启动。后一情况给DFG施加了优先级约束的含义，即每条边还描述了两个节点间的优先级约束。如果边的延时为零，那么这种优先级约束是一种迭代内优先级约束（intra-iteration precedence constraint），如果边线有一个延时或者多个延时，那么这种优先级约束是一种迭代间优先级约束（inter-iteration precedence constraint）。迭代内和迭代间优先级约束一起定义了DFG中的节点执行的顺序。比如图1-23b中从节点 A 到节点 B 的边指定了迭代间的优先级约束，它规定 A 的第 k 次迭代的执行必须在 B 的第 $(k+1)$ 次迭代前完成。从 B 到 A 的边线指定了迭代内的优先级约束，它规定 B 的第 k 次迭代必须在 A 的第 k 次迭代之前执行。

DFG以及框图既能用来描述线性单速率DSP系统，也能用来描述非线性多速率DSP系统。框图描述与实际硬件架构更为接近，其中的信号处理功能用功能单元（block，功能块）描述。另一方面，DFG还描述信号处理算法中的子任务（或者是用节点表示的基本运算）之间的数据流。来自于同一个算法的不同的DFG之间可以用高层次变换互相得到[24]。图1-24示出公式（1-61）中的3阶FIR滤波器的DFG。图1-24a从图1-20中的直接形式框图获得，图1-24b从图1-21中的转置形式的框图获得。DFG主要用在高层次综合中，它将DSP应用的并发实现方法推导到并行硬件上，其中子任务的调度以及资源分配是主要关心的目标（调度（schedule）是指确定节点何时并且在哪个硬件单元中执行）。

DFG中的节点可以是简单到不可分割的基本运算单元，如加法或者是基本逻辑运算。这样的DFG称为是原子的（atomic）。如果粒度是在信号处理子任务层次上，比如滤波，那么

34 ~ 37

这种DFG称为粗粒度（coarse-grain）的数据流图。

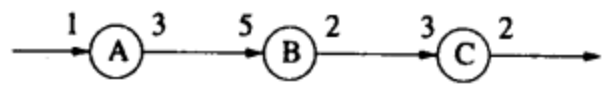


a) 直接形式3阶FIR滤波器DFG　　b) 转置3阶FIR滤波器DFG

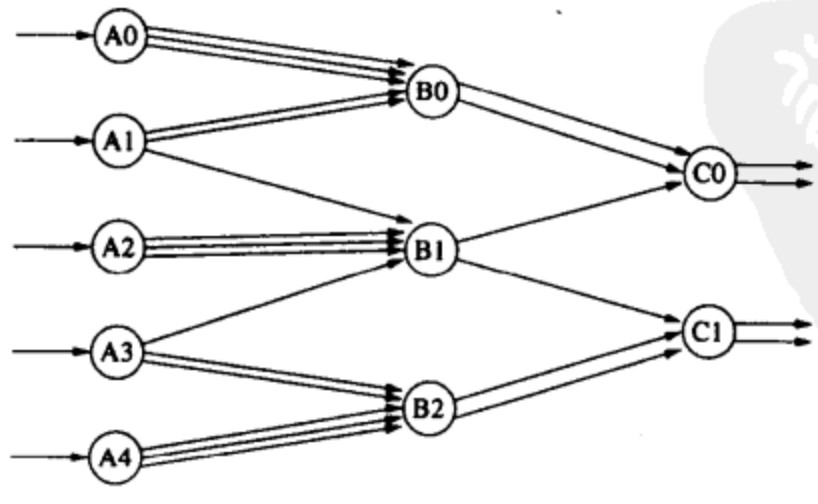
图 1-24

同步（synchronous）数据流图（SDFG）是数据流图（原子的或者是粗粒度）的一种特殊情况，每个节点在每次执行中产生或者消耗的数据样值数目是事先（priori）指定的。比如图1-23c就是一个描述运算 $y(n) = ay(n-1) + x(n)$ 的同步数据流图，其中明确指定了节点 A 和节点 B 的每次执行都消耗一个数据样值并产生一个输出。它描述了一个单速率系统。SDFG还可以用简单的方式描述多速率系统。比如，图1-25a所示的多速率系统，其中节点 A、B 和 C 分别运行在不同的频率 f_A、f_B 和 f_C。注意，A 每个时间单位处理 f_A 个输入样值，而每个时间单位产生 $3f_A$ 个输出样值。节点 B 每次执行期间消耗5个输入样值，因此每个时间单位消耗 $5f_B$个输入样值。利用等式 $3f_A = 5f_B$，可得 $f_B = 3f_A/5$。类似地，节点 C 的工作速率可计算为 $f_C = 2f_B/3 = 2f_A/5$。这样，对于给定的输入采样率，节点 A、B 和 C 的工作频率都可以计算出来。但是，正如第2.5节所说的那样，利用将多速率系统展开（unrolling）成单速率系统的方法，单采样率DFG（SRDFG）也可以用来描述多速率系统，与图1-25a的多速率DFG（MRDFG）等价的单速率DFG示于图1-25b。但是，这个单速率DFG包含有10个节点和30条边，而MDFG只需3个节点和4条边。

38



a) 多采样率DFG



b) 从(a) 多采样率DFG转换为单采样率DFG，后者可用线性SFG表示

图 1-25

39

1.4.4 依赖图

依赖图（DG）是一种有向图，它表明算法中计算之间的依赖关系。DG中的节点表示运算（指定为节点），边表示节点之间的优先级约束。在DG中，只要算法（程序）中调用一个新的运算，就会产生一个新的节点，从而基于单一计算的节点不会出现复用的情况[27]。因为DG明确表明了图中节点与其他节点之间的依赖关系，所以它与DFG表达方法是类似的。它们的区别如下：DFG中的节点只涵盖了对应算法在一次迭代中的计算，而这些计算每次迭代都是重复执行的，而DG包含算法中所有迭代的运算。DFG包含延时单元，这些延时单元将当前迭代中的数据存储并传给下次迭代，而DG不包含延时单元。公式（1-61）中的3阶FIR滤波器采用图1-26所示的二维DG描述。

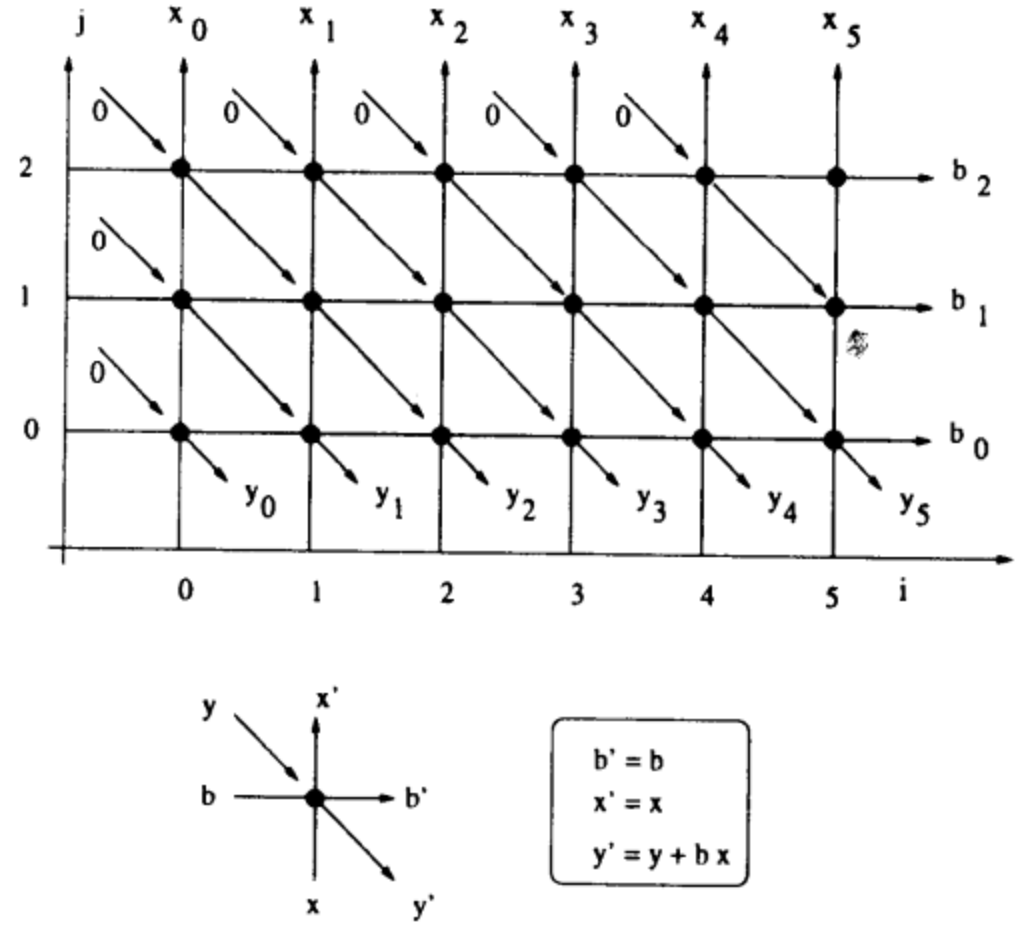


图1-26 3阶FIR滤波器 $y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2)$ 的依赖图

依赖图用于脉动阵列设计。在脉动阵列设计中，通过用不同的方式探究DG中的并行性（见第7章），可以从单个DG中推导出各种不同的实现方法。

1.5 本书概要

本书中各章讲述了用于改善DSP系统性能特点的各种不同方法，包括面积、速度、功耗和舍入误差。第1章评述了各种DSP算法，介绍了当前DSP系统对运算的要求，描述了框图、信号流图、数据流图的各种DSP算法的表示方法。其余的章节可分成三个部分。

本书的第一部分（第2章到第7章）介绍了几种高层次架构变换，它可针对某种给定的算法设计一系列的架构。这种变换包括流水线、重定时、展开、折叠以及脉动阵列设计方法。本书第二部分（第8章到第12章）研究高层次算法变换，比如强度缩减（strength reduction）、超前（look-ahead）和弛豫超前（relaxed look-ahead）。强度缩减变换用于减少卷积、并行FIR

数字滤波器、DCT和并行秩-阶滤波器（parallel rank-order filter）中乘法的数目，超前和
40 弛豫超前变换用于设计流水线直接形式和格型递归的数字滤波器、自适应数字滤波器和并行递归数字滤波器。本书的这一部分揭示了算法设计与集成电路实现之间的相互作用关系。本书的第三部分（第13章到第18章）介绍了VLSI中加法、乘法和数字滤波器的架构，以及有关高性能VLSI设计的问题，比如流水线风格、低功耗设计以及用于可编程数字信号处理器的架构。

本书有如下七个附录：1. 用于确定迭代边界和重定时的最短路径算法；2. 用于设计折叠架构所需的确定折叠集合的调度和配置技术；3. 欧几里德最大公约数（Euclid's Common Divisor，GCD）算法，它用于Winograd卷积；4. Schur多项式的正交性，用于设计格型数字滤波器；5. 快速位并行加法和乘法；6. 位串行系统的调度；7. FIR滤波器系数的量化。

本书中采用的技术可以应用于DSP系统的高速度、小面积、低功耗实现，以满足比如多媒体[28]、有线和无线通信等领域的应用。

参考文献

1. P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features.* Elsevier, 1993.
2. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Prentice Hall, 1989.
3. S. Haykin, *Adaptive Filter Theory.* Prentice Hall, 1991.
4. J. G. Chung and K. K. Parhi, *Pipelined Lattice and Wave Digital Recursive Filters.* Kluwer, 1996.
5. M. L. Honig and D. G. Messerschmitt, *Adaptive Filters: Structures, Algorithms and Applications.* Kluwer, 1984.
6. N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. on Computers*, pp. 90–93, Jan. 1974.
7. P. Pirsch, N. Demassieux, and W. Cehrke, "VLSI architectures for video compression – a survey," *Proc. IEEE*, no. 2, pp. 220–245, Feb. 1995.
8. G. A. Davidson, P. R. Cappello, and A. Gersho, "Systolic architectures for vector quantization," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1651–1664, Oct. 1988.
9. S. Y. Kung, *VLSI Array Processors.* Prentice Hall, 1988.
10. G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, pp. 268–278, March 1973.
11. R. Bellman and S. Dreyfus, *Applied Dynamic Programming.* Princeton University Press, 1962.
12. P. P. Vaidyanathan, *Multirate Digital Signal Processing.* Prentice Hall, 1993.
13. R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing.* Prentice Hall, 1983.
14. O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, pp. 14–38, Oct. 1991.
15. T. Sikora, "MPEG digital video-coding standards," *IEEE Signal Processing Magazine*, pp. 82–100, Sept. 1997.

41

16. B. Bhatt, D. Birks, and D. Hermreck, "Digital television: making it work," *IEEE Spectrum*, pp. 19–28, Oct. 1997.

17. M. Mizuno et al., "A 1.5W single-chip MPEG2 MP@ML encoder with low-power motion estimation and clocking," in *Proc. of IEEE International Solid State Circuits Conference (ISSCC97)*, pp. 256–257, Feb. 1997.

18. W. F. Brinkman, "The transistor: 50 glorious years and where we are going," in *Proc. of IEEE International Solid State Circuits Conference (ISSCC)*, pp. 22–26, Feb. 1997.

19. B. Ackland, "Programmable multimedia signal processors," in *Circuits and Systems in the Information Age* (Y.-F. Huang and C.-H. Wei, eds.), pp. 23–30, IEEE Press, 1997.

20. P. Hilfinger, "A high-level language and silicon compiler for digital signal processing," in *Proc. of IEEE Custom Integrated Circuits Conference*, pp. 213–216, 1985.

21. H. De Man, J. Rabaey, P. Six, and L. J. Claesen, "Cathedral-II: a silicon compiler for digital signal processing," *IEEE Design and Test*, vol. 3, no. 6, pp. 13–25, Dec. 1986.

22. A. Stoll and P. Duzy, "High-level synthesis from VHDL with exact timing constraints," in *Proc. of 29th ACM/IEEE Design Automation Conference*, June 1992.

23. S. Amellal and B. Kaminska, "Functional synthesis of digital systems with tass," *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 5, pp. 537–552, May 1994.

24. K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE*, pp. 1879–1895, Dec. 1989.

25. R. E. Crochiere and A. V. Oppenheim, "Analysis of linear digital networks," *Proc. IEEE*, no. 4, pp. 581–595, Apr. 1975.

26. E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE, special issue on hardware and software for digital signal processing*, pp. 1235–1245, Sept. 1987.

27. J. Allen, "Computer architecture for digital signal processing," *Proc. IEEE*, no. 5, pp. 852–873, May 1985.

28. K. K. Parhi and T. Nishitani, eds., *Digital Signal Processing for Multimedia Systems*. Marcel Dekker, 1999.

第2章 迭代边界

2.1 引言

很多DSP算法比如递归数字滤波器和自适应数字滤波器都包含有反馈环路，因此对于可实现系统，迭代以及采样周期存在其内在基本的最低边界。这种边界称为迭代周期边界，或者简称为迭代边界。迭代边界是用数据流图（DFG）所表示的算法的一种特征。同样算法的不同表示方法有可能会产生不同的迭代边界。即使是用无限多处理器（infinite processor）也无法实现少于这个迭代边界的迭代周期。本章定义了迭代边界，并提出了两种计算迭代边界的方法，其中包括最长路径矩阵（longest path matrix，LPM）和最小环均值（minimum cycle mean，MCM）方法。本章还介绍了多速率DFG的迭代边界。

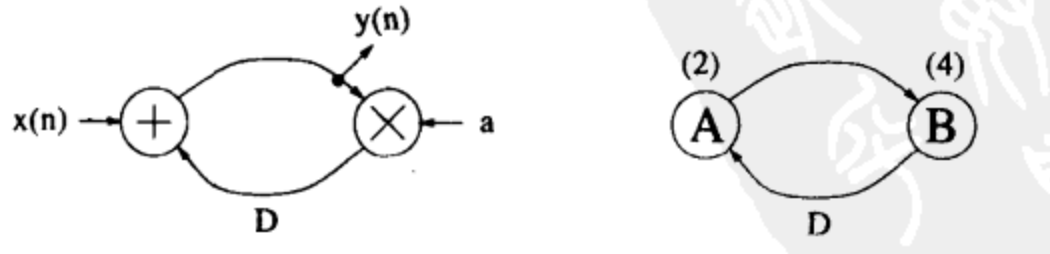
2.2 数据流图表示

DSP程序是一种无休止的程序，从时刻 $n = 0$ 运行到 $n = \infty$。比如计算 $y(n) = ay(n-1) + x(n)$ 的DSP程序表示如下：

```
for n = 0 to ∞
    y(n) = ay(n − 1) + x(n)
```

该DSP程序的输入是序列 $x(n)$，其中 $n = 0, 1, 2, \cdots$，以及初始状态 $y(-1)$。输出为序列 $y(n)$，$n = 0, 1, 2, \cdots$。

DSP程序通常用DFG表示，它是一种可以用来描述程序的有向图（即每条边有明显的方向）（参考第1.4.3节）。例如，图2-1a即表示程序 $y(n) = ay(n-1) + x(n)$，图2-1b是其简化后的表示。图2-1b的结构是个DFG，它包含有节点和边的集合。节点表示任务或者运算（节点 A 表示加法，节点 B 表示乘法），每个节点都有与之相关的执行时间。边表示节点之间相互通信，每条边都带有一个非负的延时。在本例中，边 $A \to B$ 的延时为0，边 $B \to A$ 的延时为1。一个节点的一次迭代（iteration）表示该节点正好执行一次，一个DFG的一次迭代表示该DFG中每个节点都正好执行一次。



a) $y(n) = ay(n-1) + x(n)$ 的图形表示　　b) 该程序的DFG，括号中的数字表示该节点的执行时间

图 2-1

DFG中的每条边描述了两个节点之间的优先级约束。如果一条边没有延时，那么这种优

先级约束是一种迭代内优先级约束（intra-iteration precedence constraint），如果一条边有一个延时或者多个延时，那么它是一种迭代间优先级约束（inter-iteration precedence constraint）。迭代内和迭代间优先级约束共同指定了DFG中的节点执行顺序。例如图2-1b中从 A 到 B 的边指定了迭代内的优先级约束，即 A 的第 k 次迭代的执行必须先于 B 的第 k 次迭代。从 B 到 A的边指定了迭代间的优先级约束，即 B 的第 k 次迭代必须先于 A 的第 $(k+1)$ 次迭代。用 X_k表示节点 X 的第 k 次迭代。如果要着重区别迭代内优先级约束和迭代间优先级约束，用单箭头线表示迭代内优先级约束，比如 $A_k \rightarrow B_k$，用双箭头表示迭代间优先级约束，比如 $B_k \Rightarrow A_{k+1}$，否则所有优先级约束都用单箭头表示。

44

在DFG中，具有零延时的所有路径中，具有最长运算时间的路径定义为关键路径（critical path）。图2-1b所示的DFG中的关键路径是 $A \rightarrow B$，它需要的运算时间为6 u.t.。图2-2中的DFG含有几条零延时路径。这些路径中最长的运算时间是5 u.t.（两条路径：6 → 3 → 2 → 1以及5 → 3 → 2 → 1都是关键路径），所以关键路径的运算时间为5 u.t.。在DFG中，关键路径是组合逻辑在每拍的执行（rippling）中所需最长的路径。所以关键路径的运算时间就是DFG的一次迭代所需的最小运算时间。

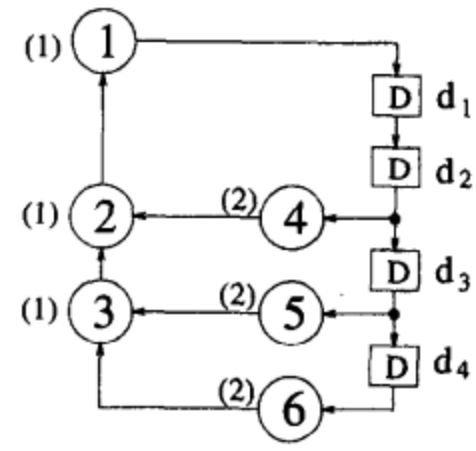


图2-2 具有三个环路的DFG，其环路边界为 4/2 u.t.，5/3 u.t.以及5/4 u.t. 该DFG的迭代边界为$T\infty = 2$ u.t.

DFG可以分为非递归的和递归的。非递归DFG不包含环路，而递归DFG至少包含有一个环路。例如FIR滤波器是非递归的，而图2-1b中的DFG是递归的，因为它含有环路 $A \rightarrow B \rightarrow A$。递归DFG存在着一个基本极限，该极限表示了关于用硬件实现基本的DSP程序能够有多快。这个极限称为迭代边界（iteration bound）T_∞[1][2]，它与DSP程序实现中可用的计算能力无关。

2.3 环路边界和迭代边界

环路是指起点和终点都是同一个节点的有向路径，例如图2-1b中的 $A \rightarrow B \rightarrow A$。“环路（loop）”和“环（cycle）”都可以用来描述有向图中的带方向的环。执行一个环路所需的时间可以由DFG中的边所描述的优先级关系来确定。图2-1b中的DFG中的边描述了如下优先级约束：

45

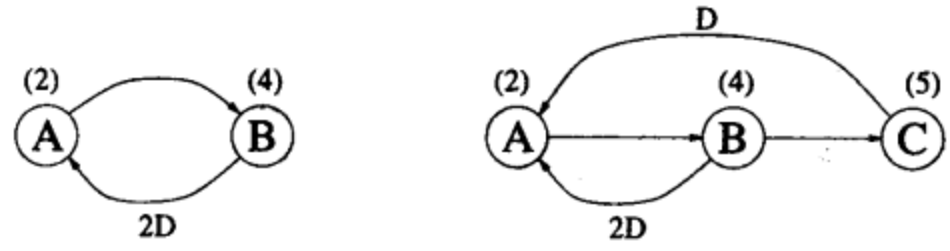
$$A_0 \rightarrow B_0 \Rightarrow A_1 \rightarrow B_1 \Rightarrow A_2 \rightarrow B_2 \Rightarrow A_3 \rightarrow \cdots$$

其中单箭头（→）表示迭代内优先级约束，双箭头（=>）表示迭代间优先级约束。根据这些优先级约束，环路第 k 次迭代包含依次执行 A_k 和 B_k。如果节点A和B的执行时间分别为2 u.t.和4 u.t.，环路每次迭代需要6 u.t.。这就是环路边界（loop bound），它表示环路运行时间的最低限制。正式表达如下：第 l 个环路的环路边界是指 t_l / w_l，其中 t_l 是环路运行时间，w_l 是指环路中延时数目。可以验证图2-1b中环路的环路边界是6 /1 = 6 u.t.。

图2-3a中的环路有两个延时，需要6 u.t.进行计算，因此环路边界为6/2 = 3 u.t.。为了了解到底该环路如何用3 u.t. 完成一次迭代，我们需要观察优先级约束。

$$A_0 \rightarrow B_0 \Rightarrow A_2 \rightarrow B_2 \Rightarrow A_4 \rightarrow B_4 \Rightarrow A_6 \rightarrow \cdots$$
$$A_1 \rightarrow B_1 \Rightarrow A_3 \rightarrow B_3 \Rightarrow A_5 \rightarrow B_5 \Rightarrow A_7 \rightarrow \cdots$$

这表明两组独立的优先级约束，其中一组偶数迭代，另一组奇数迭代。在本例中，两次迭代可以在6 u.t.内完成运算，因此环路边界为6/2 = 3。



a) 有一个环路边界为6/2 = 3 u.t的DFG，该DFG的迭代边界为3 u.t.

b) 迭代边界 T_∞ = max{6/2, 11/1} = 11 u.t. 的DFG

图 2-3

图2-3b中的DFG有两个环路，即环路 $l_1 = A \to B \to A$ 和环路 $l_2 = A \to B \to C \to A$。$l_1$ 和 l_2 的环路边界分别是6/2 = 3 u.t. 和11/1 = 11 u.t.。*关键环路*（critical loop）是指具有最大环路边界的环路，所以在本例中 l_2 是关键环路。关键环路的环路边界就是DSP程序的*迭代边界*（iteration bound），这个边界是DSP程序中迭代或者采样周期的最低限制，与可用的运算资源的多少无关。正式定义迭代周期为 46

$$T_\infty = \max_{l \in L} \left\{ \frac{t_l}{w_l} \right\} \tag{2-1}$$

其中 L 指DFG中的环路集合，t_l 是环路 l 的运算时间，w_l 是环路 l 延时数目。对于图2-3b中的DFG，有

$$T_\infty = \max \left\{ \frac{6}{2}, \frac{11}{1} \right\} = 11 \text{ u.t.}$$

寻找DFG中的迭代边界的直接办法就是找出所有的环路，然后利用公式(2-1)直接计算 T_∞。但是随着DFG中节点数目的增加，环路数目会以指数形式增加，因此这种方法需要很长的运行时间。有三种具有多项式时间的方法可用来计算 T_∞，即最长路径矩阵（longest path matrix）算法[3]；最小环均值（minimum cycle mean）方法以及负周期检测算法（negative cycle detection）[5]。第2.4节介绍了前两种方法。

2.4 计算迭代边界的算法

本节中描述的两种迭代边界算法可由图2-2中所示的DFG来说明。该DFG有三个环路：环路 $l_1 = 1 \to 4 \to 2 \to 1$，其环路边界为 4/2 u.t.；环路 $l_2 = 1 \to 5 \to 3 \to 2 \to 1$，其环路边界为5/3 u.t.；以及环路 $l_3 = 1 \to 6 \to 3 \to 2 \to 1$，其环路边界为5/4 u.t.。因此该DFG的迭代边界是

$$T_\infty = \max \left\{ \frac{4}{2}, \frac{5}{3}, \frac{5}{4} \right\} = 2 \text{ u.t.}$$

2.4.1 最长路径矩阵算法

在最长路径矩阵（LPM）算法[3]中，首先构建一系列的矩阵，然后通过观察矩阵斜对角线上的数据，就可以得到迭代边界。用 d 表示DFG中的延时数目。矩阵 $\mathbf{L}^{(m)}$，$m = 1, 2, \cdots$，

d，按照下面方法构建：从延时单元 d_i 到延时单元 d_j 中经过正好 $m-1$ 个延时（不包含 d_i 和 d_j）的所有路径中，用 $l_{i,j}^{(m)}$ 表示最长的运算时间。如果这样的路径不存在，那么 $l_{i,j}^{(m)} = -1$。值得注意的是，任何两个节点之间的最长路径可以采用附录A中的任何一种路径算法进行计算。例如为确定图2-2中的DFG的 $l_{3,1}^{(1)}$，必须考虑从延时单元 d_3 到延时单元 d_1 中所有正好经过零个延时单元的路径。存在一条这样的路径，即 $d_3 \to n_5 \to n_3 \to n_2 \to n_1 \to d_1$，该路径需要的运算时间为5，因此 $l_{3,1}^{(1)} = 5$。为了确定 $l_{4,3}^{(1)}$，注意到从延时单元 d_4 到 d_3 的所有路径中没有延时单元数为零的路径，因此 $l_{4,3}^{(1)} = -1$。在确定 $\mathbf{L}^{(1)}$ 其余的元素后，我们有：

47

$$\mathbf{L}^{(1)} = \begin{bmatrix} -1 & 0 & -1 & -1 \\ 4 & -1 & 0 & -1 \\ 5 & -1 & -1 & 0 \\ 5 & -1 & -1 & -1 \end{bmatrix}$$

更高阶的矩阵 $\mathbf{L}^{(m)}$，$m = 2, 3, \cdots, d$，并不需要直接从DFG确定。它们可以通过下面的规则进行递归计算：

$$l_{i,j}^{(m+1)} = \max_{k \in K}(-1, l_{i,k}^{(1)} + l_{k,j}^{(m)})$$

其中 K 是区间 $[1, d]$ 中使得 $l_{i,k}^{(1)} = -1$ 或 $l_{k,j}^{(m)} = -1$ 都不满足的整数 k 的集合。例如为计算 $l_{2,1}^{(2)}$ 第一步是在可能的集合{1, 2, 3 ,4}中寻找集合 K。因为 $l_{2,3}^{(1)} = 0$，$l_{3,1}^{(1)} = 5$，所以3属于 K。对于$k = 1, 2, 4$ 中的每个数，$l_{2,k}^{(1)}$和 $l_{k,1}^{(1)}$两者至少有一个等于-1，所以 $k = 1, 2, 4$都不属于 K。这样 $K = \{3\}$，$l_{2,1}^{(2)}$ 的值为

$$\begin{aligned} l_{2,1}^{(2)} &= \max_{k \in \{3\}}(-1, l_{2,k}^{(1)} + l_{k,1}^{(1)}) \\ &= \max(-1, 0+5) = 5 \end{aligned}$$

计算 $l_{i,j}^{(2)}$ 中的其余部分，可得

$$\mathbf{L}^{(2)} = \begin{bmatrix} 4 & -1 & 0 & -1 \\ 5 & 4 & -1 & 0 \\ 5 & 5 & -1 & -1 \\ -1 & 5 & -1 & -1 \end{bmatrix}$$

$\mathbf{L}^{(2)}$ 的计算只需要 $\mathbf{L}^{(1)}$，而 $\mathbf{L}^{(3)}$ 的计算同时需要 $\mathbf{L}^{(1)}$ 和 $\mathbf{L}^{(2)}$。计算 $l_{3,3}^{(3)}$ 时，因为 $l_{3,1}^{(1)} = 5$，$l_{1,3}^{(2)} = 0$，并且对于 $k = 2, 3, 4$，$l_{3,k}^{(1)}$ 和 $l_{k,3}^{(3)}$ 中至少有一个等于-1，因此 $K = \{1\}$。$l_{3,3}^{(3)}$ 的值为

$$\begin{aligned} l_{3,3}^{(3)} &= \max_{k \in \{1\}}(-1, l_{3,k}^{(1)} + l_{k,3}^{(2)}) \\ &= \max(-1, 5+0) = 5 \end{aligned}$$

48

计算 $\mathbf{L}^{(3)}$ 和 $\mathbf{L}^{(4)}$ 中剩余部分，可得

$$\mathbf{L}^{(3)} = \begin{bmatrix} 5 & 4 & -1 & 0 \\ 8 & 5 & 4 & -1 \\ 9 & 5 & 5 & -1 \\ 9 & -1 & 5 & -1 \end{bmatrix}$$

和

$$\mathbf{L}^{(4)} = \begin{bmatrix} 8 & 5 & 4 & -1 \\ 9 & 8 & 5 & 4 \\ 10 & 9 & 5 & 5 \\ 10 & 9 & -1 & 5 \end{bmatrix}$$

一旦矩阵 $\mathbf{L}^{(m)}$ 计算出之后，迭代边界可以用下式确定

$$T_\infty = \max_{i,m\in\{1,2,\cdots d\}} \left\{ \frac{l_{i,i}^{(m)}}{m} \right\}$$

在本例中，

$$T_\infty = \max\left\{\frac{4}{2},\frac{4}{2},\frac{5}{3},\frac{5}{3},\frac{5}{3},\frac{8}{4},\frac{8}{4},\frac{5}{4},\frac{5}{4}\right\} = 2$$

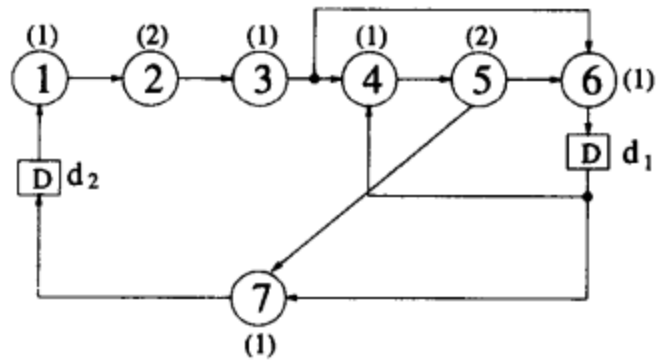


图2-4 迭代边界$T\infty$ = 8的滤波器

作为另外一个例子，考虑图2-4中的DFG。矩阵 $\mathbf{L}^{(m)}$ $(m = 1, 2)$ 为

$$\mathbf{L}^{(1)} = \begin{bmatrix} 4 & 4 \\ 8 & 8 \end{bmatrix}$$

49

和

$$\mathbf{L}^{(2)} = \begin{bmatrix} 12 & 12 \\ 16 & 16 \end{bmatrix}$$

迭代边界为

$$T_\infty = \max\left\{\frac{4}{1},\frac{8}{1},\frac{12}{2},\frac{16}{2}\right\} = 8$$

因为 $l_{ij}^{(m)}$ 表示包含有延时单元 d_i 并且有 m 个延时的所有路径中的最长运算时间，所以LPM算法是正确的。对所有的 i 和 m，选择 $l_{ij}^{(m)}/m$ 的最大值，可以找到DFG中所有环路的最大环路边界，这就是迭代边界。

因为在 $\mathbf{L}^{(k+1)}$ 中有 d^2 个元素，并且每次运算的时间复杂度为 $O(d)$，所以利用 $\mathbf{L}^{(1)}$ 和 $\mathbf{L}^{(k)}$ 计算 $\mathbf{L}^{(k+1)}$ 所需的时间复杂度为 $O(d^3)$。因此从 $\mathbf{L}^{(1)}$ 计算 $\mathbf{L}^{(d)}$ 的时间复杂度为 $O(d^4)$。尽管我们可以通过观察来确定 $\mathbf{L}^{(1)}$，但是[3]中给出了一种时间复杂度为 $O(de)$ 的算法用来寻找 $\mathbf{L}^{(1)}$，其中 d 和 e 分别是DFG中延时和边的数目。因此计算迭代边界的LPM算法的时间复杂

度为$O(d^4 + de)$。有兴趣的读者，可以参考[3]中的内容，该时间复杂度可以降低到 $O(d^3\log d + de)$。值得注意的是，$\mathbf{L}^{(d)}$ 可以用复杂度为 $O(d^3\log d)$的方法计算，即从 $\mathbf{L}^{(1)}$ 计算 $\mathbf{L}^{(2)}$，从 $\mathbf{L}^{(2)}$ 计算 $\mathbf{L}^{(4)}$，从 $\mathbf{L}^{(4)}$ 计算 $\mathbf{L}^{(8)}$ 等。

2.4.2 最小环均值算法

最小环均值（MCM）算法[4]将确定迭代边界简化为寻找图形的MCM。[6]中的方法能够有效地计算MCM。注意“环（cycle）”和“环路（loop）”能够互换。

本节描述的算法用到了环均值（cycle mean）、最大环均值（maximum cycle mean）以及MCM的概念。环 c 的环均值 $m(c)$ 表示 c 的边的平均长度，即只需将环中边的长度求和然后除以环中边线的数目即可。MCM $\lambda_{\min}$ 表示所有环均值的最小值，即

$$\lambda_{\min} = \min_c m(c)$$

类似地，最大环均值 $\lambda_{\max}$ 为

$$\lambda_{\max} = \max_c m(c)$$

一个新的图形 G_d 的环均值可以用来计算迭代边界，其中 G_d 可以从相应的DFG中得到（称该DFG为 G）。如果 d 为 G 中的延时单元的数目，那么图形 G_d 有 d 个节点，其中每个节点对应 G 中的一个延时单元。图 G 中从延时 d_i 到延时 d_j 的所有不经过任何延时的路径中的最长路径的长度就是 G_d 中从节点 i 到节点 j 的边的权重，用 $w(i, j)$ 表示。如果从延时 d_i 到延时 d_j 没有零延时路径，那么 G_d 中边 $i \to j$ 不存在。图2-2中DFG对应的图形 G_d 如图2-5a所示。值得注意的是构建 G_d 在本质上与构建第2.4.1节描述的LPM算法中的 $\mathbf{L}^{(1)}$ 是一致的。

50

G_d 中的一个环 c 的边的权重之和就是 G 中所有包含环 c 节点所代表的延时单元的环的最大运算时间。这是因为 G_d 中的边权重是 G 中延时中的最大运算时间。例如，图2-6a中所示的图形 G，存在着两个包含延时 D_α 和 D_β 的环，这两个环的运算时间为6 u.t.和4 u.t.。图2-6b所示的对应图 G_d 中存在这样一个环，它经过了对应于 D_α 和 D_β 两节点的环。该环的边的权重之和为6，该值也是图2-6a中 G 中的两个环的最大运算时间。在 G_d 中，一个环路的边的数量等于该环的节点数量，也等于 G 中该环的延时单元数量。因此 G_d 中环 c 的环均值为

$$\frac{G\text{ 中包含 }c\text{ 中延时单元的所有环中的最大运算时间}}{G\text{ 中这些环的延时单元数目}}$$

这是 G 中包含有环 c 中延时单元的环路的最大环路边界。G_d 的最大环均值是 G 中所有环的最大环路边界，即为 G 的迭代边界。

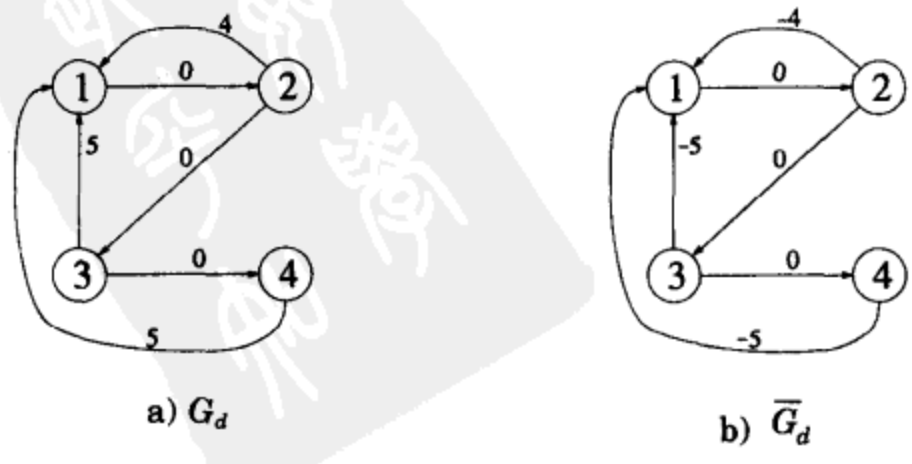
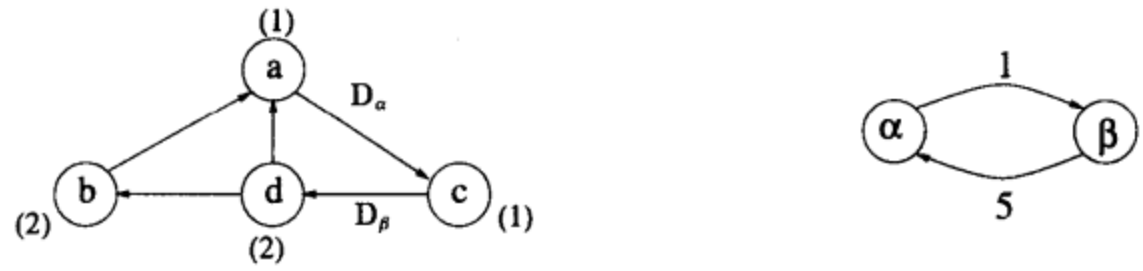


图2-5 图2-2中DFG的图形



a) 图形 G: 具有包含延时单元 D_α 和 D_β 的两个环　　b) 对应的图形 G_d

图 2-6

为了计算 G_d 的最大环均值，可从 G_d 构建图形 $\overline{G}_d$，只需将 G_d 边的权重乘上−1即可。即 $\overline{G}_d$的拓扑与 G_d 一样，而 $\overline{G}_d$中边 $i\to j$ 的权重 $\overline{w}(i,j)$由下式确定: $\overline{w}(i,j)=-w(i,j)$，其中 $w(i,j)$ 是 G_d 中的边 $i\to j$ 的权重。图2-5b中即为图2-2中的DFG对应的 $\overline{G}_d$。G_d 的最大环均值即为 $\overline{G}_d$的MCM与−1的乘积。所以 G 的迭代周期能够按照这样的方法计算: 找出 $\overline{G}_d$的MCM，然后将其与−1相乘。

通过先建立 $d+1$ 个向量序列 $\mathbf{f}^{(m)}$，$m=0,1,\cdots,d$，其中每个向量的大小为 $d\times 1$，可以找到 $\overline{G}_d$的MCM。任意指定 $\overline{G}_d$的一个参考节点（将该节点称作 s）。初始向量 $\mathbf{f}^{(0)}$ 如下确定: $f^{(0)}(s)=0$，$\mathbf{f}^{(0)}$ 的其余的项全部设为∞。假定节点1取为图2-5b中的图形 $\overline{G}_d$的参考节点，那么

$$\mathbf{f}^{(0)}=\begin{bmatrix}0\\ \infty\\ \infty\\ \infty\end{bmatrix}$$

其余的向量 $\mathbf{f}^{(m)}$，$m=1,2,\cdots,d$，按照下面的公式递归计算:

$$f^{(m)}(j)=\min_{i\in I}(f^{(m-1)}(i)+\bar{w}(i,j)) \tag{2-2}$$

51 ~ 52

其中 $\overline{w}(i,j)$是 $\overline{G}_d$中边 $i\to j$ 的权重，I 表示在 $\overline{G}_d$中从节点 i 到 j 存在一条边（$i\to j$）的节点集合。图2-5b $\overline{G}_d$中可以找到的向量序列为

$$\mathbf{f}^{(1)}=\begin{bmatrix}\infty\\ 0\\ \infty\\ \infty\end{bmatrix}\quad \mathbf{f}^{(2)}=\begin{bmatrix}-4\\ \infty\\ 0\\ \infty\end{bmatrix}\quad \mathbf{f}^{(3)}=\begin{bmatrix}-5\\ -4\\ \infty\\ 0\end{bmatrix}\quad \mathbf{f}^{(4)}=\begin{bmatrix}-8\\ -5\\ -4\\ \infty\end{bmatrix}$$

例如 $f^{(4)}(1)$如下计算

$$\begin{aligned}f^{(4)}(1) &= \min(f^{(3)}(2)-w(2,1),f^{(3)}(3)-w(3,1),f^{(3)}(4)-w(4,1))\\ &= \min(-4-4,\infty-5,0-5)=-8.\end{aligned}$$

根据向量 $\mathbf{f}^{(m)}$，$m=0,1,\cdots,d$，迭代边界可以如下计算

$$T_\infty=-\min_{i\in\{1,2,\cdots,d\}}\left(\max_{m\in\{0,1,\cdots,d-1\}}\left(\frac{f^{(d)}(i)-f^{(m)}(i)}{d-m}\right)\right)$$

本例中，$d=4$，表2-1列出了下式对应 $1\leqslant i\leqslant 4$，$0\leqslant m\leqslant 3$ 的值。

$$\frac{f^{(4)}(i)-f^{(m)}(i)}{4-m}$$

在某些例子中，我们可能会遇到 $f^{(d)}(i)-f^{(m)}(i)=\infty-\infty$。在这种情况下，$\infty-\infty$ 应该认为是0。例如 $f^{(4)}(4)-f^{(1)}(4)=\infty-\infty$，所以表2-1中 $i=4$ 和 $m=1$ 时的值为 $(\infty-\infty)/3=0/3=0$。

表2-1 $\frac{f^{(4)}(i)-f^{(m)}(i)}{4-m}$ **的值** $(1\leqslant i\leqslant 4,\ 0\leqslant m\leqslant 3)$

	$m=0$	$m=1$	$m=2$	$m=3$	$0\leqslant m\leqslant 3$中 $\frac{f^{(4)}(i)-f^{(m)}(i)}{4-m}$ 的最大值
$i=1$	-2	$-\infty$	-2	-3	-2
$i=2$	$-\infty$	$-5/3$	$-\infty$	-1	-1
$i=3$	$-\infty$	$-\infty$	-2	$-\infty$	-2
$i=4$	$\infty-\infty$	$\infty-\infty$	$\infty-\infty$	∞	∞

利用表2-1中的右边的列，可以得到

$$T_\infty=-\min\{-2,-1,-2,\infty\}=2$$

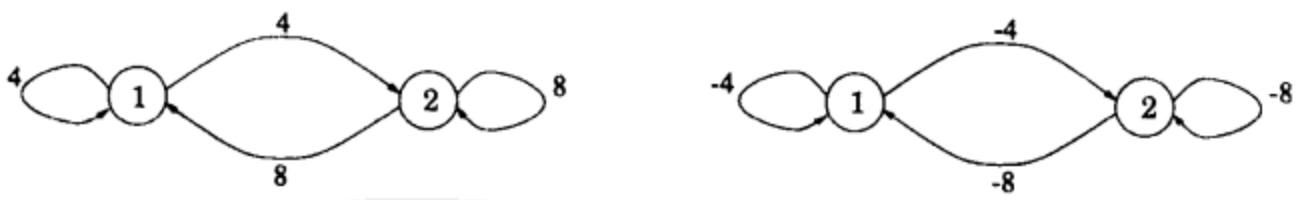
与预期的一样。

另外一个例子，考虑图2-4中的DFG，相应的图形 G_d 和 $\overline{G}_d$ 如图2-7所示。向量集合 $\mathbf{f}^{(m)}$，$m=0,1,2$ 为

$$\mathbf{f}^{(0)}=\begin{bmatrix}0\\ \infty\end{bmatrix}\quad \mathbf{f}^{(1)}=\begin{bmatrix}-4\\ -4\end{bmatrix}\quad \mathbf{f}^{(2)}=\begin{bmatrix}-12\\ -12\end{bmatrix}$$

表2-2列出了下式对应 $1\leqslant i\leqslant 2$，$0\leqslant m\leqslant 1$ 的值：

$$\frac{f^{(2)}(i)-f^{(m)}(i)}{2-m}$$



a) G_d　　b) $\overline{G}_d$

图2-7 图2-4中DFG的图形

表2-2 $\frac{f^{(2)}(i)-f^{(m)}(i)}{2-m}$ **的值** $(1\leqslant i\leqslant 2,\ 0\leqslant m\leqslant 1)$

	$m=0$	$m=1$	$0\leqslant m\leqslant 1$的 $\frac{f^{(2)}(i)-f^{(m)}(i)}{2-m}$ 最大值
$i=1$	$-12/2$	$-8/1$	-6
$i=2$	$-\infty$	$-8/1$	-8

利用表2-2中右边的列，可以确定

$$T_\infty=-\min\{-8,-6\}=8$$

从原始DFG构建 G_d 和 $\overline{G}_d$ 的时间复杂度为 $O(de)$。注意这与LPM算法中计算 $\mathbf{L}^{(1)}$ 的时间复杂度相等，这是因为这些是对等的问题。计算 G_d 的MCM需要的时间为 $O(de_d)$，其中 e_d 是 G_d 的边数。采用MCM算法计算迭代边界所需总的时间复杂度为 $O(de + de_d)$。 53 ~ 54

2.5 多速率数据流图的迭代边界

到目前为止，我们只考虑了单速率DFG（single-rate DFG，SRDFG），即每次迭代中每个节点正好执行一次的DFG。另外一种DFG叫做多速率DFG（multirate DFG，MRDFG）[7]，即在一次迭代中允许每个节点执行多次，而且在一次迭代中不要求2个节点执行相同的次数（参考第1.4.3节）。可以用2个步骤的方法来计算多速率DFG的迭代边界。这种2个步骤的方法是

(1) 构建一个等价于MRDFG的SRDFG。

(2) 采用LPM或者MCM算法计算等价SRDFG的迭代边界。

MRDFG的迭代边界与等价SRDFG的迭代边界是一样的。本节中对MRDFG进行定义，并提出根据MRDFG构造等价的SRDFG的一种算法（步骤1），使得可以使用本章中描述的两种算法之一来计算等价SRDFG（步骤2）的迭代边界。

一个MRDFG中从节点 U 到节点 V 的边如图2-8所示。O_{UV} 表示调用一次节点 U 在边上产生的样值数目，I_{UV} 表示调用一次节点 V 在边上所消耗的样值数目，i_{UV} 的值表示边的延时数。

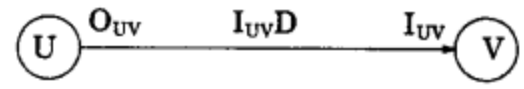


图2-8 多速率DFG中的一条边 $U \to V$

如果在一次迭代中节点 U 和 V 分别调用了 k_U 和 k_V 次，那么在一次迭代中，从节点 U 到节点 V 的边所产生的样值数目为 $O_{UV}k_U$，在一次迭代中节点 V 在边上消耗的样值数目为 $I_{UV}k_V$。为了避免边上样值的积累和不足，一次迭代中产生的样值必须与一次迭代中消耗的样值相等。这种关系可以用下式表示:

$$O_{UV}k_U = I_{UV}k_V \tag{2-3}$$

55

为了确定在一次迭代中每个节点必须执行的次数，对于MRDFG中的每个节点写出公式(2-3)后得到的方程组必须有解，因此节点的调用次数必须互素。例如图2-9中MRDFG的方程组为

$$\begin{aligned} 4k_a &= 3k_b \\ k_b &= 2k_c \\ k_c &= k_c \\ 3k_c &= 2k_a \end{aligned}$$

其解为 $k_a = 3$，$k_b = 4$，$k_c = 2$。一旦节点调用的次数确定，就可以构造与MRDFG的等价SRDFG。

为简化表达，用 a / b 表示除法的商的整数部分，用 $a \% b$ 表示余数。例如10 / 3 = 3，10 % 3 = 1。从数学上可以写作 $a/b = \left\lfloor \frac{a}{b} \right\rfloor$，$a\%b = a - \left\lfloor \frac{a}{b} \right\rfloor b$，其中 $\lfloor x \rfloor$ 表示 x 向下取整，即小于或者等于 x 的最大整

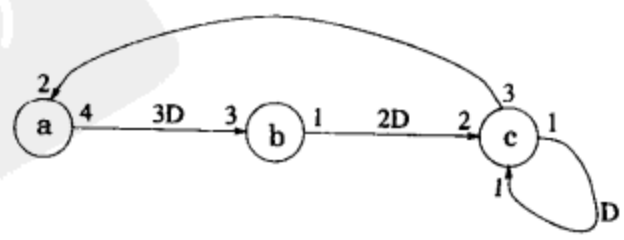


图2-9 多速率DFG

数。图2-10所示的是从MRDFG构造一个等价的SRDFG的算法。

1. 对于MRDFG中的每个节点U
2. For $k = 0$ to k_{U-1}
3. 在SRDFG中画一个与MRDFG中节点U有相同计算时间的节点U^k
4. 对于MRDFG中的每条边 $U \xrightarrow{i_{UV}} V$
5. For $j = 0$ to $O_{UV}k_{U-1}$
6. 在SRDFG中画一条延时为 $(j + i_{UV}) / (I_{UV}k_V)$ 的边 $U^{j/O_{UV}} \rightarrow V^{((j+i_{UV})/I_{UV})\%k_V}$

图2-10 构建MRDFG的等价SRDFG的算法

对于图2-9所示的DFG，可以用图2-10中的算法构造图2-11中的DFG。图2-11中SRDFG的迭代边界与图2-9中的MRDFG的迭代边界相等。

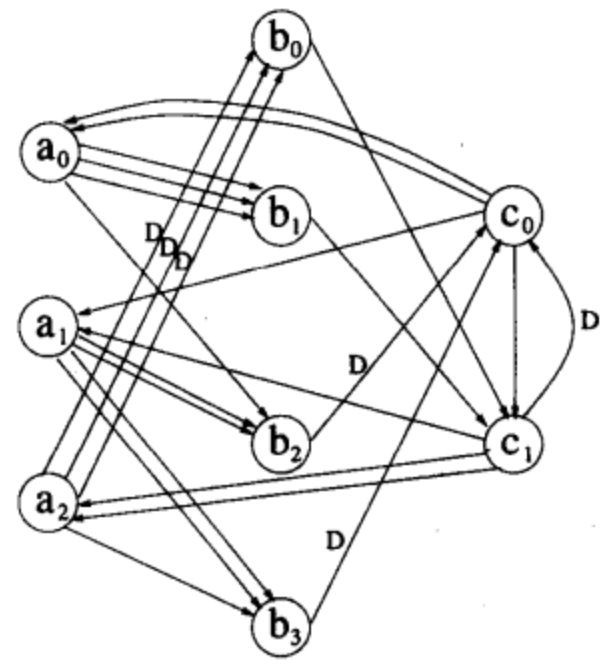


图2-11 图2-9中MRDFG的等价SRDFG

有各种技巧可用来减少等价SRDFG中的节点和边界数目，而不影响图形的迭代边界。这些约减可以减少确定等价SRDFG迭代边界的复杂度，从而直接减少确定MRDFG迭代边界的复杂度。有兴趣的读者可以在[4]中找到降低等价SRDFG复杂度的算法。

2.6 结论

DSP程序都具有从 $n = 0$ 执行到 $n = \infty$ 时的性质。这些程序常常用DFG来表示，它们或者是递归的，或者是非递归的。如果DFG是递归的，对于该DSP程序的硬件实现，迭代边界是其最小采样周期的基本极限。我们已经讨论了两种用来计算迭代边界的算法。LPM算法求迭代边界的时间复杂度为 $O(d^4 + de_d)$，MCM算法求迭代边界的时间复杂度为 $O(de + de_d)$。因为对于大多数情况，$e_d < d^2$，所以MCM算法通常比LPM算法要快。

56 ~ 57

多速率DFG的迭代边界可以先构造等价单速率DFG，然后用本章描述的两种算法之一来计算该单速率DFG的迭代边界。图2-10中的算法可以用来从多速率DFG构造等价的单速率DFG。

2.7 习题

1. 如图2-12中的DFG，括号中的数字表示节点的运算时间。用下面的方法计算该DFG的迭代边界：
   (a) LPM算法
   (b) MCM算法
2. 假定图2-13中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。

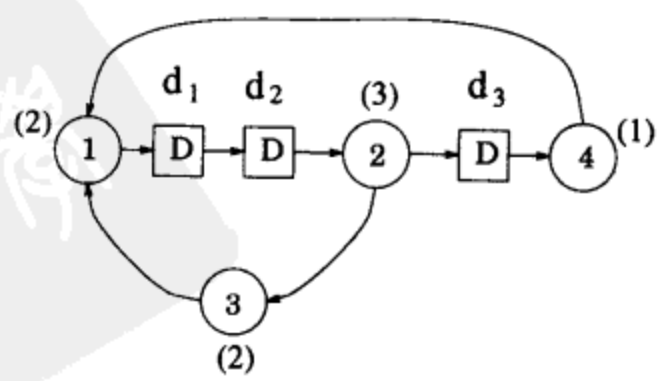


图2-12 题1的DFG

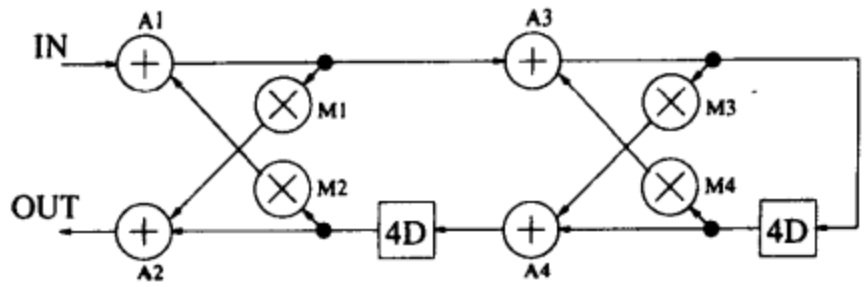


图2-13 4层全通8阶IIR数字滤波器

58

3. 假定图2-14中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。

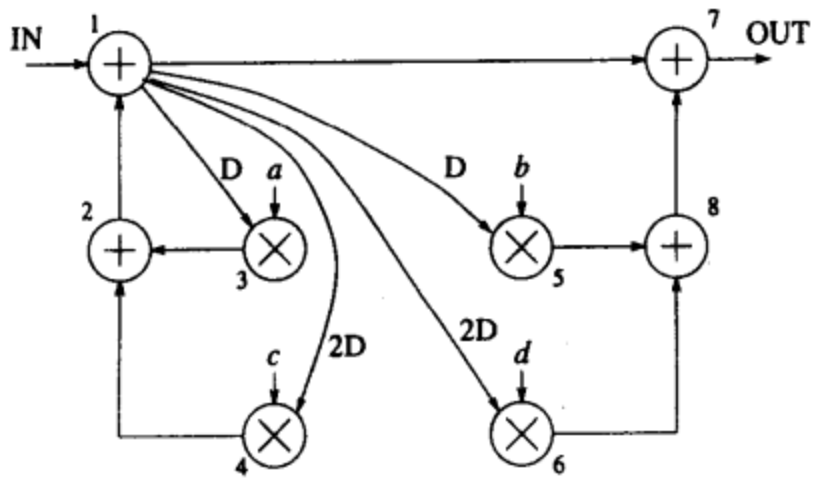


图2-14 biquad滤波器

4. 假定图2-15中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。

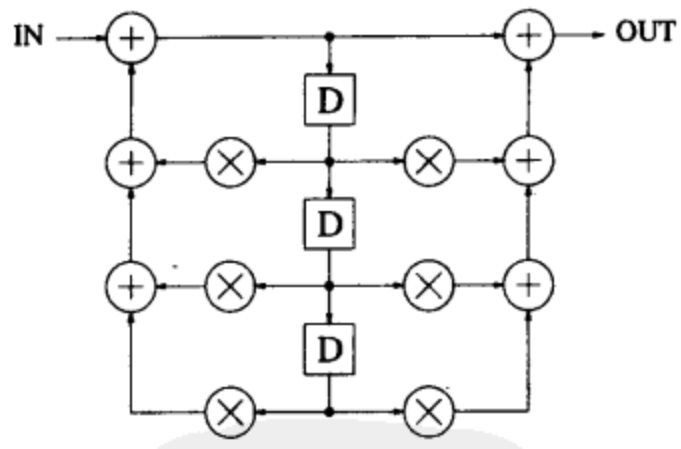


图2-15 3阶直接形式IIR滤波器

5. 假定图2-16中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。
6. 假定图2-17中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。
7. 假定图2-18中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。
8. 假定图2-11中的DFG的加法和乘法分别需要1 u.t.和2 u.t.的运算时间，重复第一题的计算。
9. 假定图2-19中的MRDFG中每个节点的执行时间为1 u.t.，确定其迭代边界。必须采用下面的步骤:

59

(a) 构造等价SRDFG

(b) 用本章介绍的任何一种算法确定等价SRDFG的迭代边界。

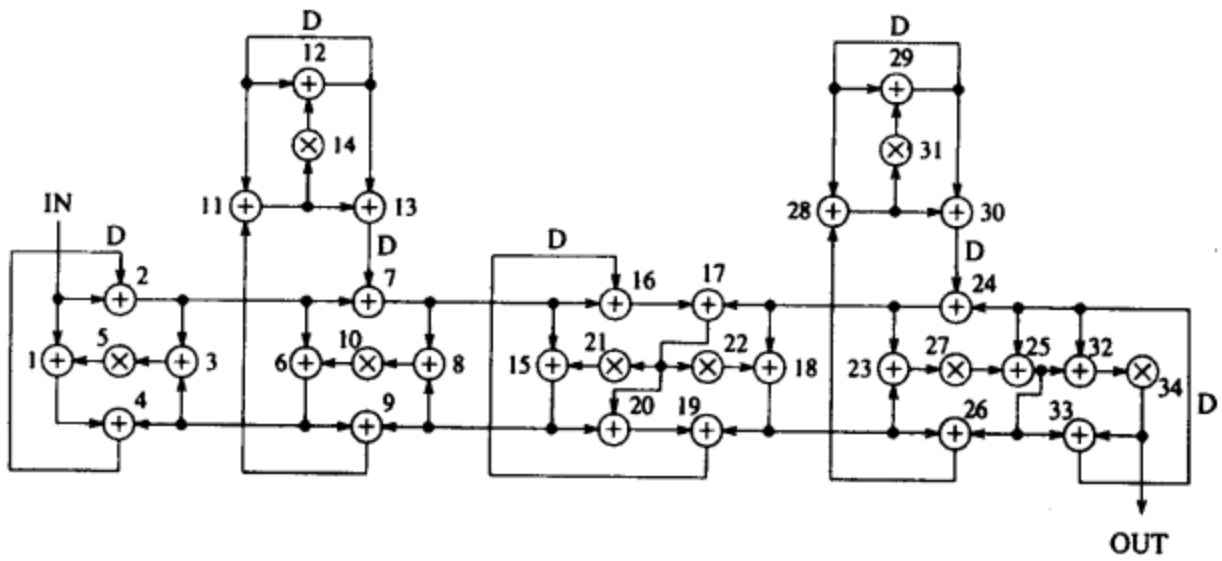


图2-16 4阶波形数字椭圆滤波器

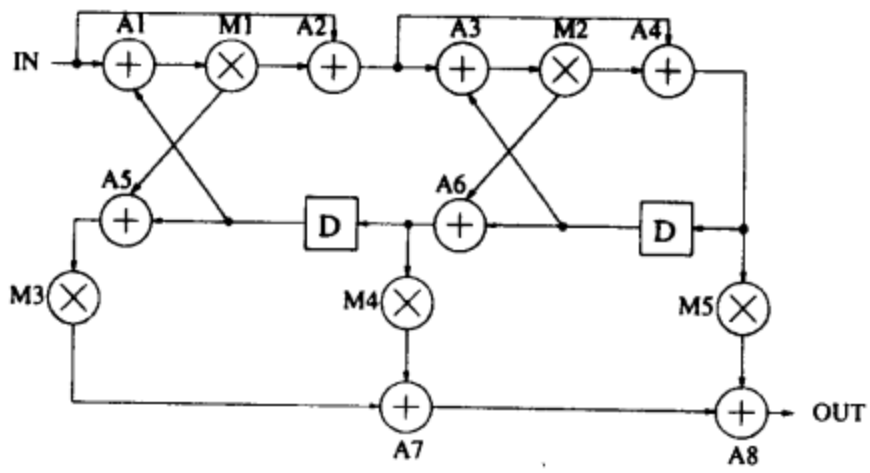


图2-17 题6中的格型滤波器

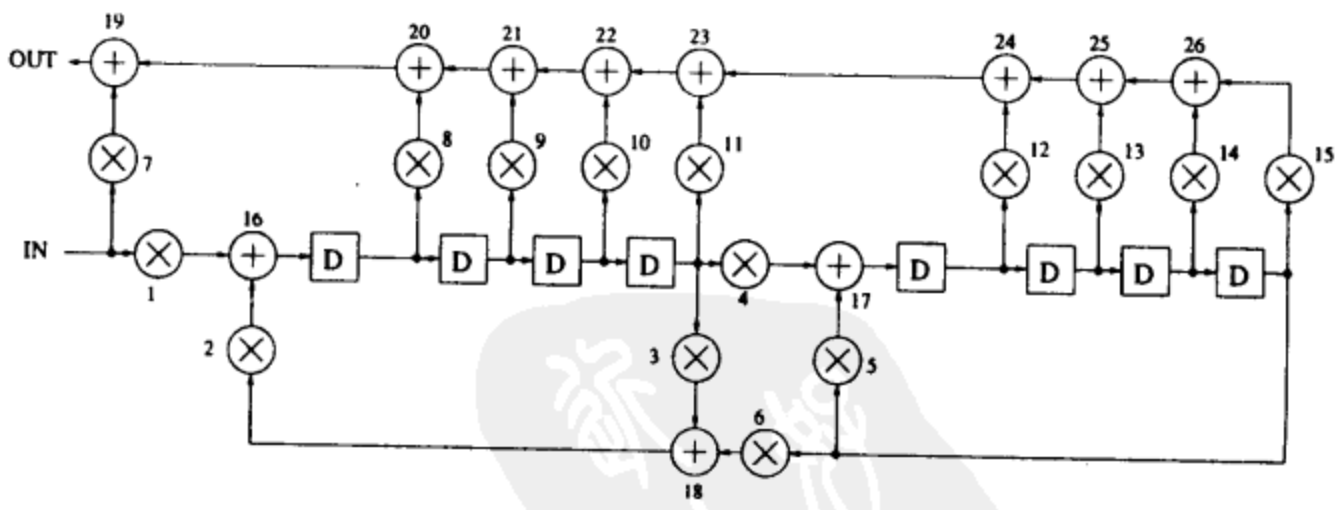


图2-18 题7中的归一化格型滤波器

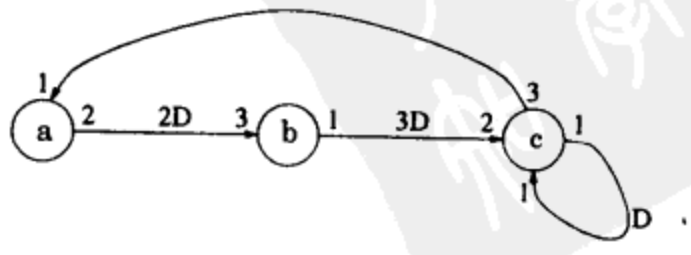


图2-19 题9中的DFG

参考文献

1. M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Trans. on Circuits and Systems*, vol. CAS-28, no. 3, pp. 196–202, March 1981.
2. K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. on Computers*, vol. 40, no. 2, pp. 178–195, Feb. 1991.
3. S. H. Gerez, S. M. Heemstra de Groot, and O. E. Herrmann, "A polynomial-time algorithm for the computation of the iteration-period bound in recursive data-flow graphs," *IEEE Trans. on Circuits and Systems–I: Fundamental Theory and Applications*, vol. 39, no. 1, pp. 49–52, Jan. 1992.
4. K. Ito and K. K. Parhi, "Determining the minimum iteration period of an algorithm," *Journal of VLSI Signal Processing*, vol. 11, pp. 229–244, 1995.
5. D. Y. Chao and D. T. Wang, "Iteration bounds of single-rate data flow graphs for concurrent processing," *IEEE Trans. on Circuits and Systems–I: Fundamental Theory and Applications*, vol. 40, no. 9, pp. 629–634, Sept. 1993.
6. R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.
7. E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. on Computers*, vol. C-36, no. 1, pp. 24–35, Jan. 1987.
8. D. A. Schwartz and T. P. Barnwell, III, "Cyclo-static solutions: optimal multiprocessor realizations of recursive algorithms," in *Proc. of IEEE VLSI Signal Processing Workshop*, 1986.

60
~
61

第3章 流水线与并行处理

3.1 引言

流水线变换导致了关键路径的缩短，从而可以提高时钟速度或采样速度，或者可以在同样速度下降低功耗。在并行处理中，多个输出在一个时钟周期内并行地计算。这样，有效采样速度提高到与并行级数相当的倍数。与流水线类似，并行处理也能够用来降低功耗。

考虑3阶有限冲激响应(FIR)数字滤波器

$$y(n) = ax(n) + bx(n-1) + cx(n-2) \tag{3-1}$$

其框图实现示于图3-1。

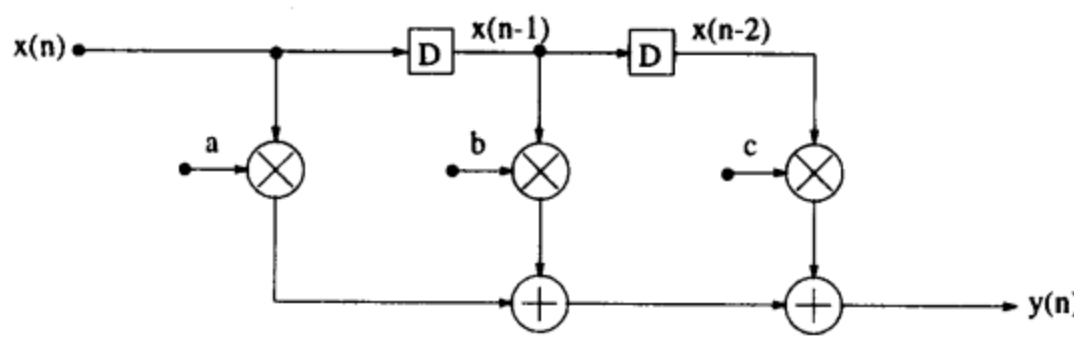


图3-1 一个3阶FIR滤波器

关键路径（或者处理一个新样点所需要的最小时间）由1个乘法与2个加法时间来限定。即如果 T_M 是乘法所用时间、T_A 是加法操作需要的时间，则“采样周期”(T_{sample})由下式给出

$$T_{sample} \geqslant T_M + 2T_A \tag{3-2}$$

因而，采样频率(f_{sample})(也称为吞吐速率或迭代速率)由下式给出

$$f_{sample} \leqslant \frac{1}{T_M + 2T_A} \tag{3-3}$$

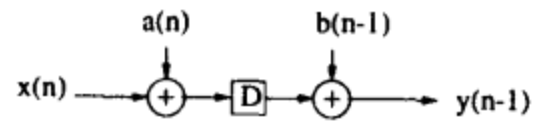
注意，图3-1所示的*直接形式结构*（direct-form structure）仅当不等式(3-2)满足时才可以使用。但是如果某些实时应用要求较快的输入速率（采样率）时，这个结构就不能用了！在这种情况下，有效关键路径需要利用流水线或者并行处理来减小。

流水线采用沿着数据通路引入流水线锁存器的方法来减小*有效关键路径*（effective critical path）。流水线已经用在了结构设计[1]~[4]和编译器综合[5]~[10]等教科书或文章中。并行处理提高采样率是采用复制硬件的方法，以使几个输入能够并行地处理，而几个输出能够在同一时间产生出来[11]~[13]。考虑图3-2a的简单结构，其关键路径的计算时间为$2T_A$，图3-2b示出了它的2级流水线结构，其中一个锁存器插入在两个加法器之间，从而使关键路径减少了一半。它的2级并行处理结构如图3-2c所示，其中相同的硬件复制为两套使得两个输入

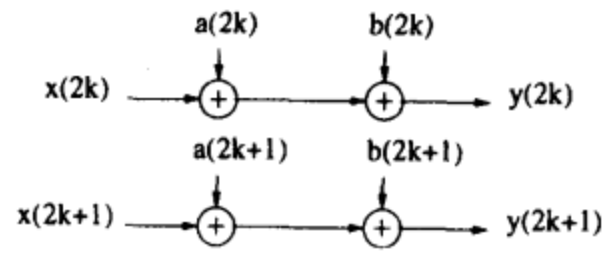
能同时进行处理和两个输出能同时产生出来。因此采样率增为原来的二倍。

本章是这样组织的：第3.2节和第3.3节分别提出流水线与并行处理的方法，它们都是针对非递归计算的，如FIR数字滤波器等。第3.4节叙述运用流水线与并行处理在相同采样速度下使用较低的电源电压来降低功耗。

a) 数据通路



b) 数据通路a的2级流水线结构



c) 数据通路a的并行处理结构

图 3-2

3.2 FIR数字滤波器的流水线

考虑公式(3-1)所表征的3阶FIR滤波器，其流水线实现是通过引入两个附加锁存器而得到的，如图3-3所示。

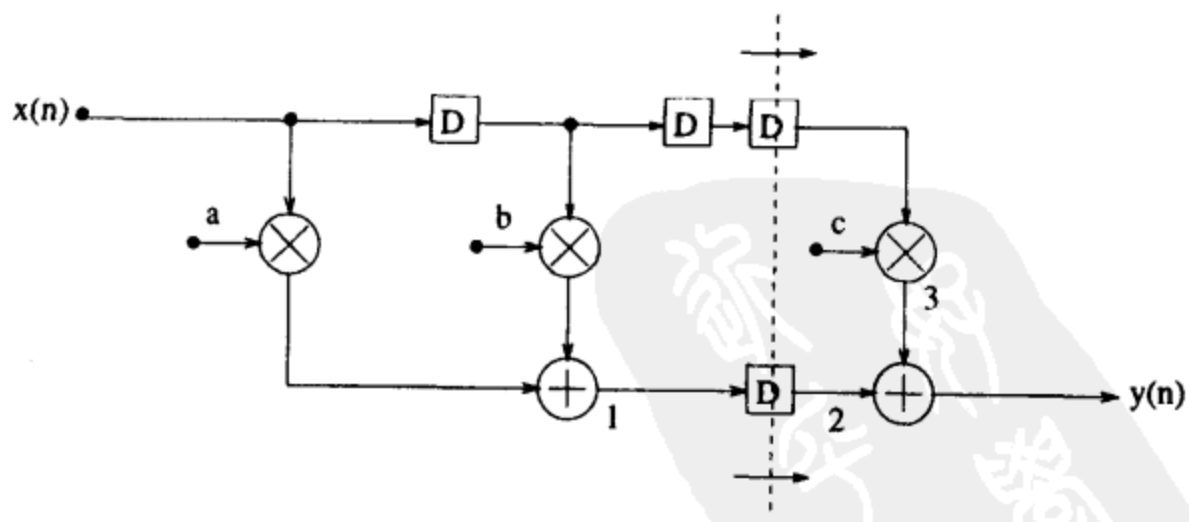


图3-3 流水线FIR滤波器，其中垂直虚线代表一个前馈割集

关键路径现在由 $T_M + 2T_A$ 减小为 $T_M + T_A$。在这种安排下，当左边的加法器启动当前迭代计算的同时，右边的加法器正在完成前次迭代结果的计算。这个流水线系统的事件调度情况示于表3-1。在这个系统中，任一时刻两个连贯的输出以交替方式被计算出来。

表3-1 图3-3中的流水线FIR滤波器事件调度情况

时钟	输入	节点1	节点2	节点3	输出
0	$x(0)$	$ax(0)+bx(-1)$	—	—	—
1	$x(1)$	$ax(1)+bx(0)$	$ax(0)+bx(-1)$	$cx(-2)$	$y(0)$
2	$x(2)$	$ax(2)+bx(1)$	$ax(1)+bx(0)$	$cx(-1)$	$y(1)$
3	$x(3)$	$ax(3)+bx(2)$	$ax(2)+bx(1)$	$cx(0)$	$y(2)$

64 ~ 65

必须注意到，在一个 M 级流水线系统中，从输入到输出的任一路径上的延时元件数目是 $(M-1)$，它要大于在原始时序电路中同一路径上的延时元件数。虽然流水线减小了关键路径，但是它付出了增加迟滞（latency）的代价。迟滞实质上是流水线系统第一个输出数据的时间与原来时序系统第一个输出数据时间相比的滞后。例如，若迟滞为一个时钟周期，那么在一个1级流水线系统中，第 k 个输出在第 $(k+1)$ 个时钟周期可用。可见，流水线的两个主要缺点是增加了锁存器数目和增加了系统的迟滞。

下述要点需要注意:

(1) 一个架构的速度（或时钟周期）由任意两个锁存器间、或一个输入与一个锁存器间、或一个锁存器与一个输出间、或输入与输出间路径中最长的路径限定。

(2) 这个最长的路径或“关键路径”可以通过在架构中适当插入流水线锁存器来减小。

(3) 流水线锁存器只能按照穿过任一图的前馈割集（feed-forward cutset）的方式插入。

为了说明第3点，需要引入两个定义:

- **割集**: 割集是一个图的边的集合，如果从图中移去这些边，图就成为不相连接的了。
- **前馈割集**: 如果数据在割集的所有边上都沿前进的方向移动，这个割集就称为前馈割集。例如在图3-3中用于对FIR滤波器实现流水线的割集就是一个前馈割集。

可以在任何FIR滤波器结构的前馈割集上不影响算法功能地任意插入锁存器。

例3.2.1 在图3-4a所示的信号流图(SFG)中，假定每个节点的计算时间为1u.t.

(a) 计算关键路径的运算时间。

(b) 插入3个额外的延时元件，如图3-4b所示，使关键路径减小为2u.t。这是一个有效的流水线吗？如果不是，请得到一个关键路径为2u.t.的合适流水线电路。

解

(a) 关键路径(任何两个锁存器间的最长路径)是($A1\rightarrow A3\rightarrow A5\rightarrow A4\rightarrow A6$)，其计算时间为4u.t.

(b) 这不是一个有效的流水线。令图3-4b中的虚线表示一个前馈割集，我们可以看到，只有3个锁存器穿过割集插入。虽然关键路径比原来的系统减少了一半，但是功能已经变化了。为了得到一个合适的流水线电路，流水线锁存器应该插入到前馈割集的所有边上，图3-4c示出了一个有效的流水线电路，其关键路径为2u.t.

66

说明：在图3-4c中由前馈割集加延时元件导致一个2级流水线。在2级流水线系统中，从输入到输出的任一路径中的延时元件数目都增加了1个。 ■

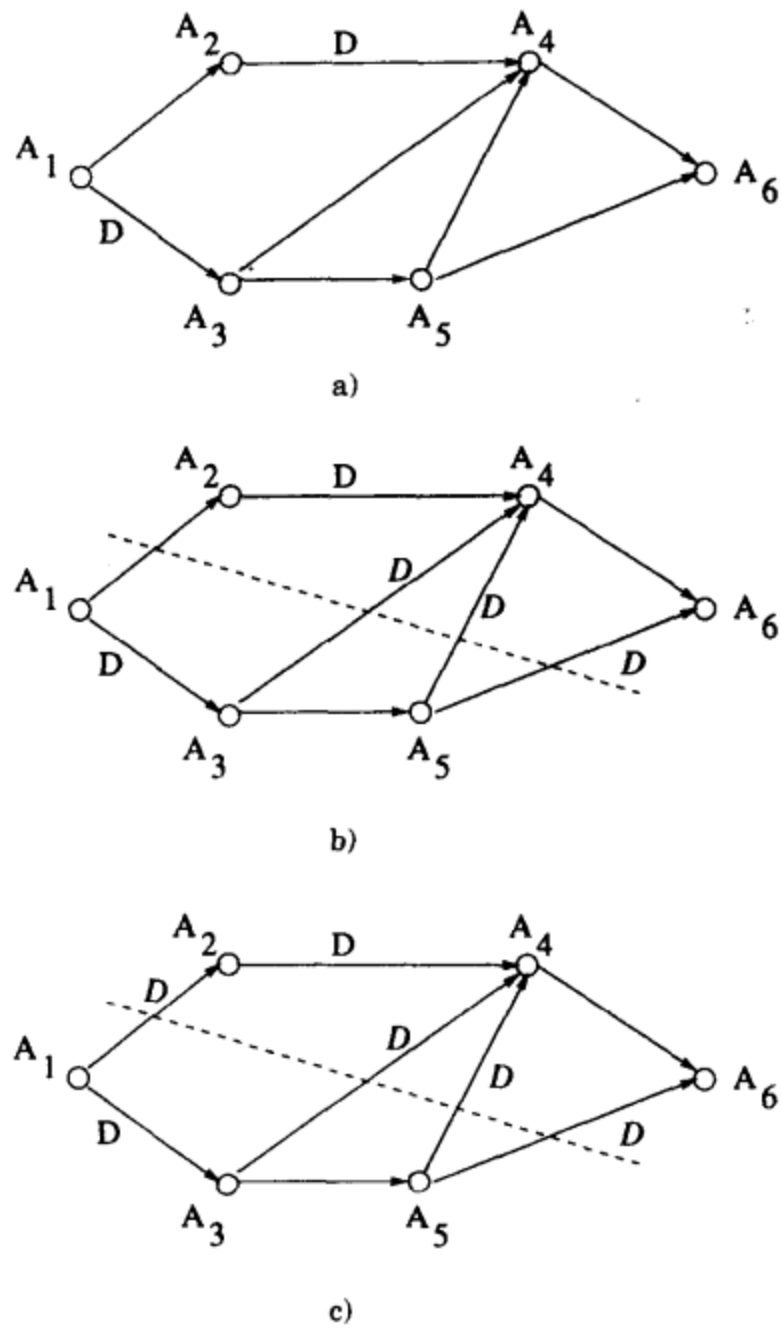


图3-4 例3.2.1的信号流图

3.2.1 数据广播结构

原始的3阶FIR滤波器的关键路径能够通过转置结构来减小，而不需要引入任何的流水线锁存器。转置定理陈述如下:

“反转一个给定SFG中所有边的方向并且互换输入与输出端口，系统的功能保持不变。”

一个3阶FIR滤波器表示为图3-5所示的SFG形式。

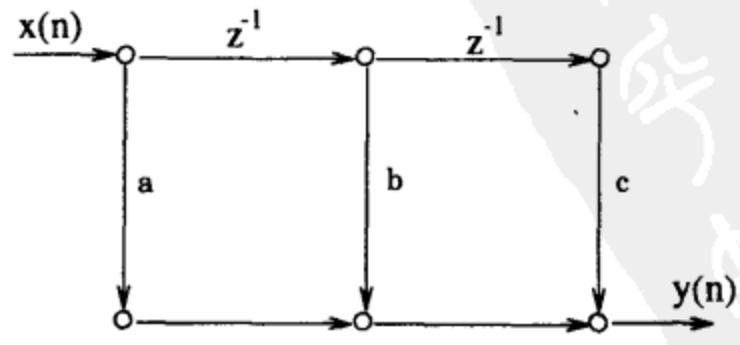


图3-5 FIR滤波器的信号流图表示

转置滤波器的SFG见图3-6，其等效框图见图3-7。

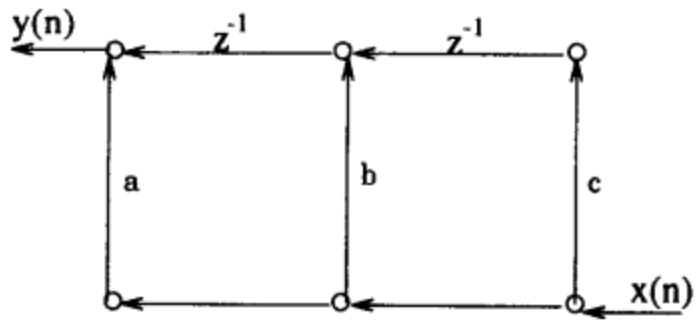


图3-6 FIR滤波器的转置SFG表示

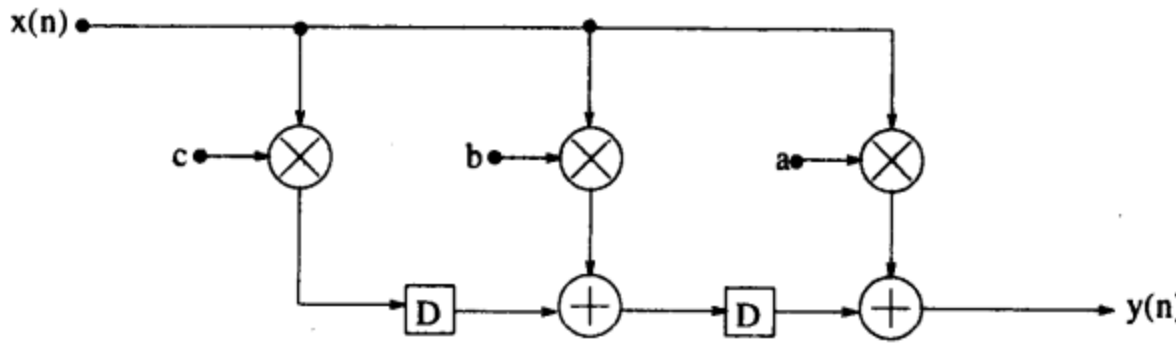


图3-7 FIR滤波器的数据广播结构

这导致了数据广播结构，其数据不是存储下来，而是同时广播到所有的乘法器。注意，此时我们得到的关键路径为 $T_M + T_A$，与图3-3相同。

3.2.2 细粒度流水线

令 T_M = 10u.t.，T_A = 2u.t.，所希望的时钟周期为$(T_M + T_A)/2$，即6个单位时间(u.t.)。在这种情况下，乘法器分拆为两个较小的单元，其处理时间分别为6u.t.和4u.t.。现在通过在穿过乘法器的水平割集上插入锁存器的方法可以达到所希望的时钟速度，这种方法称为细粒度流水线。3阶数据广播FIR滤波器的细粒度流水线版本见图3-8。

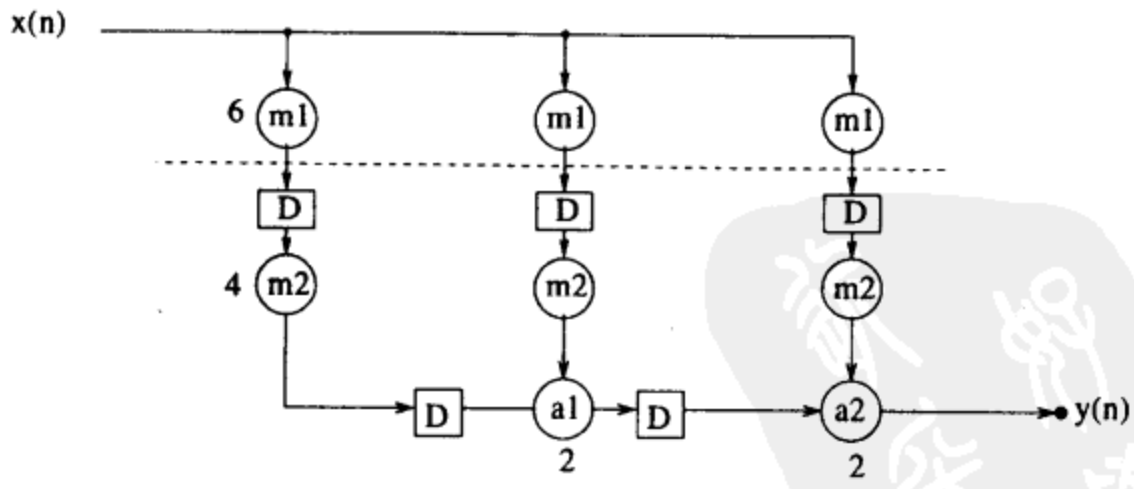


图3-8 FIR滤波器的细粒度流水线

3.3 并行处理

注意到并行处理与流水线技术互为对偶的这一特点是十分有趣的，若一个计算能够排成流水线，它也能并行地处理。两种技术都发掘了计算中可供利用的并发性，只是方式不同。

67 ~ 69

当一组互不相关的计算能够在一个流水线系统中按交替方式计算时，则它们也能够利用复制的硬件按并行处理的模式计算。

考虑公式(3-1)所描述的3阶FIR滤波器，这个系统是一个单输入单输出(SISO)系统，可描述如下:

$$y(n) = ax(n) + bx(n-1) + cx(n-2) \tag{3-4}$$

为了获得一个并行处理结构，SISO系统必须转换为MIMO（多输入多输出）系统。例如，下列方程组描述一个每个时钟周期有3个输入的并行系统（即并行处理的级数 $L = 3$）。

$$\begin{aligned} y(3k) &= ax(3k) + bx(3k-1) + cx(3k-2) \\ y(3k+1) &= ax(3k+1) + bx(3k) + cx(3k-1) \\ y(3k+2) &= ax(3k+2) + bx(3k+1) + cx(3k) \end{aligned} \tag{3-5}$$

此处 k 表示时钟周期数。可以看出，在第 k 个时钟周期，有3个输入 $x(3k)$，$x(3k+1)$ 和$x(3k+2)$被处理，同时输出中产生3个样点。并行处理系统也称为*块处理系统*，而每个时钟周期内处理的输入个数称为*块尺寸*。由于MIMO的结构，在任意一条线处插入一个锁存器会产生一个有效延时，等于 L 个对应于采样率的时钟周期。每个延时元件称为一个*块延时*（也称为 L 级*减慢*，L-slow）。例如，把信号 $x(3k)$ 延迟一个时钟周期将导致信号 $x(3k-3)$ 而非$x(3k-1)$，因为 $x(3k-1)$ 已经是另一条输入线的输入。3级并行FIR滤波器的框图架构示于图3-9，其细节示于图3-10。

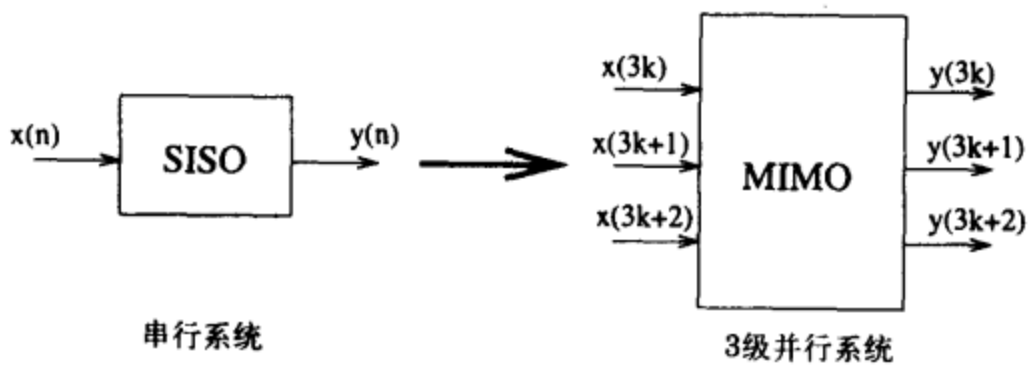


图3-9 一个块处理的例子

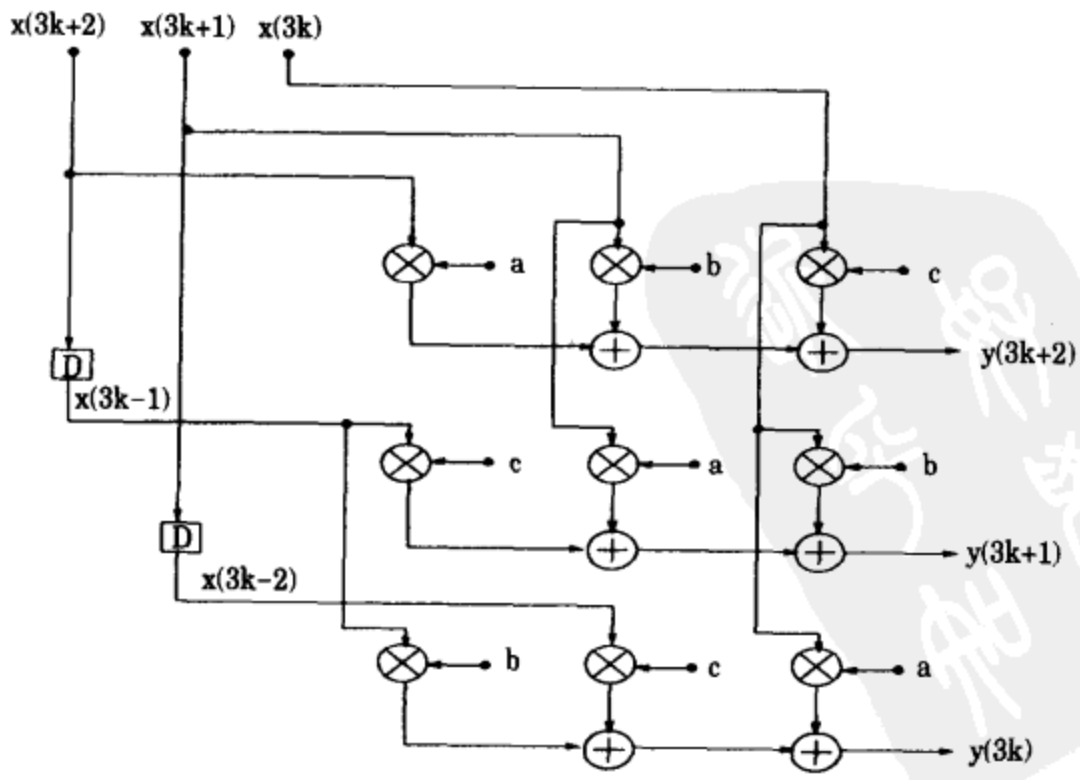


图3-10 块尺寸为3的3阶FIR滤波器的并行处理架构

注意，块或并行处理系统的关键路径保持不变，而且时钟周期 (T_{clk}) 必须满足

$$T_{clk} \geqslant T_M + 2T_A \tag{3-6}$$

但是，由于3个样点是在一个时钟周期内而不是三个时钟周期内处理的，因此迭代周期要由下式确定:

$$T_{iter} = T_{sample} = \frac{1}{L}T_{clk} \geqslant \frac{1}{3}(T_M + 2T_A) \tag{3-7}$$

重要的是要理解在并行系统中 $T_{clk} \neq T_{sample}$，而在流水线系统中 $T_{clk} = T_{sample}$。图3-11给出了一个完整的并行处理系统，它包含串-并转换器和并-串转换器，其细节分别示于图3-12和3-13中。

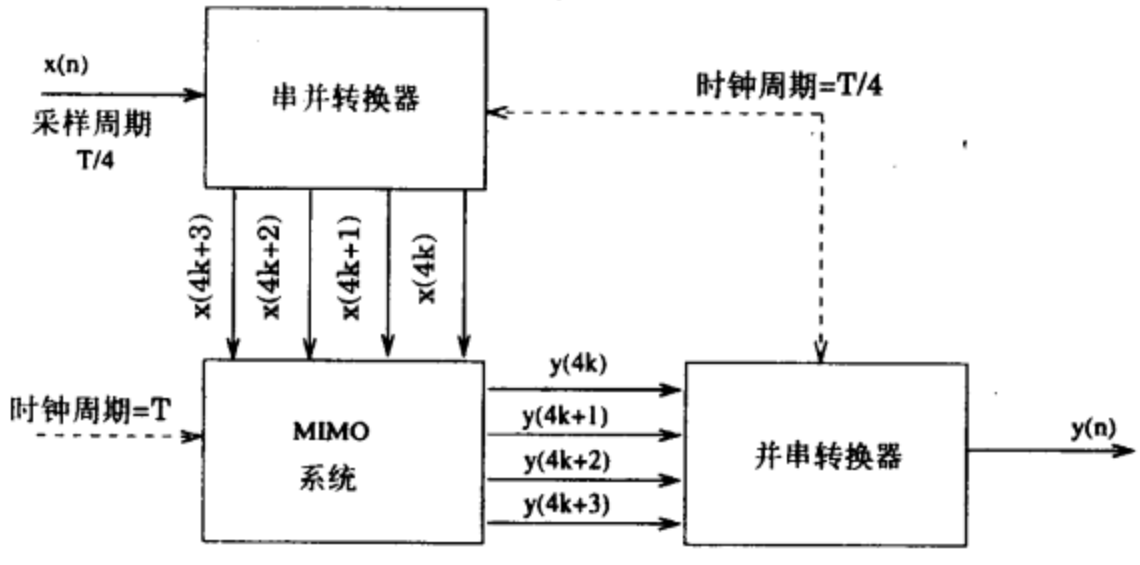


图3-11 块尺寸为4的完全并行处理系统

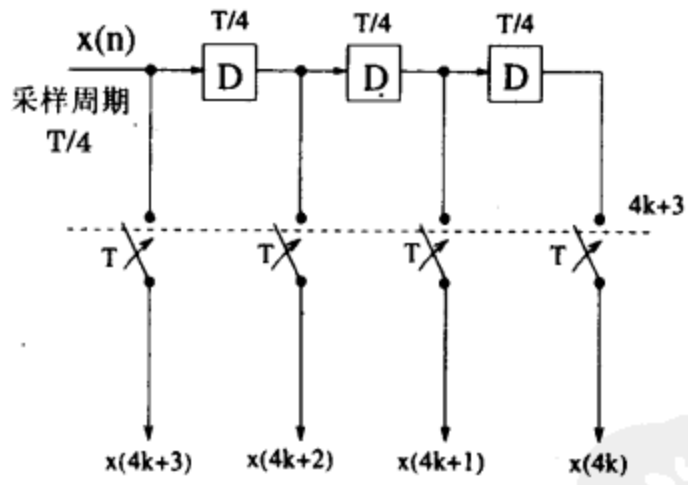


图3-12 串-并转换器

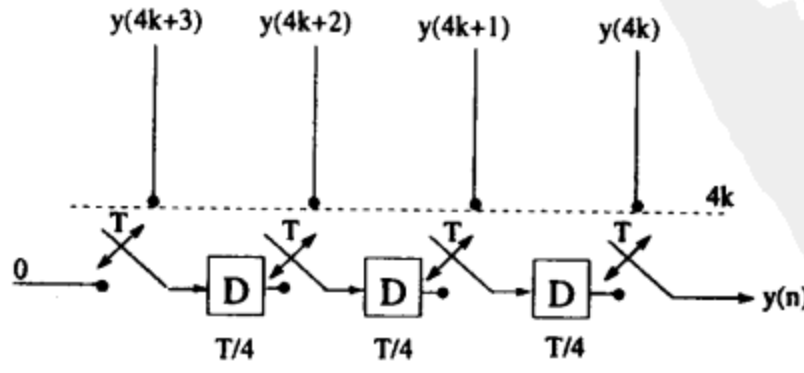


图3-13 并-串转换器

现在人们会问，当能够用流水线达到同样好的效果时，为什么还要并行处理呢？为什么要复制和使用这么多硬件呢？回答是，流水线存在着一个基本的限制，就是输入/输出(I/O)的瓶颈问题。

考虑图3-14所示的芯片组。例如，若假定输出管腿、输入管腿和两个芯片之间连线的延时总和为8 nsec（纳秒），那么 T_{clk} 必须大于或等于8 nsec。若关键路径的计算时间小于8 nsec，则I/O延时的限制将占主导地位，该系统为通信受限的系统。这实质上意味着，流水线仅在关键路径计算时间大于通信或I/O延时边界时才可以使用，一旦达到此边界后，流水线就不能进一步提高速度了。这时，流水线必须结合并行处理才能进一步提高该架构的速度。作为一个例子，考虑图3-10的并行滤波器。假定一个乘法的计算时间(T_M)是10 u.t.，一个加法的计算时间为2 u.t.。细粒流水线可用到并行滤波器中来进一步缩小关键路径。在这种情况下，乘法器分拆为两个较小的单元 $m1$ 和 $m2$，其计算时间分别为7u.t.和3u.t.。流水线锁存器插入到穿过乘法器的水平割集上，如图3-15所示。虽然这些水平割集看起来似乎是无效的，但是实际上它们是有效的，因为去掉这些割集的边就断开了元件间的连接。于是通过并行处理与流水线的结合，采样周期减至

$$T_{iter} = T_{sample} = \frac{1}{LM}T_{clk} = \frac{1}{6}(T_M + 2T_A) \tag{3-8}$$

并行处理也被用来通过减慢时钟来减少功耗，这种方法减少功耗是由于时钟方面的原因，相比之下，流水线系统需要工作在更快的时钟下，才能保持等价的吞吐率或采样速度。进一步说，更不希望使用细粒度流水线，如位级流水线，因为硬件开销与迟滞时间都会由于锁存器的显著增加而增加。

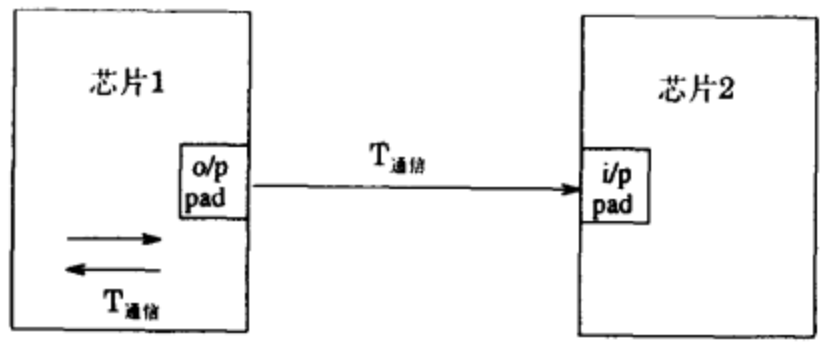


图3-14 一个芯片组

3.4 流水线与并行处理的功耗减低

利用流水线和并行处理有两个主要的优点:

- 高速度
- 低功耗

由前面章节已经看出流水线与并行处理能够增加采样速度。现在考虑在采样速度不需要增加的情况下如何利用这些技术来降低功耗[14]。

在往下叙述之前，先回顾一下两个公式，一个是计算CMOS电路传播延时的公式，另一个是计算功耗的公式。传播延时 T_{pd} 与在关键路径上各种晶体管栅极和杂散电容的充放电荷密切相关。对CMOS电路，传播延时可写为:

$$T_{pd} = \frac{C_{charge}V_0}{k(V_0 - V_t)^2} \tag{3-9}$$

其中 C_{charge} 表示在单个时钟周期里充放电的电容，即沿着关键路径的电容，V_o 是电源电压，V_t 是阈值电压。参数 k 是工艺参数 μ、W/L 和 C_{ox} 的函数。CMOS电路的功耗可用下列方程来估计:

$$P = C_{total} V_0^2 f \tag{3-10}$$

其中 C_{total} 代表电路中的总电容，V_o 是电源电压，f 是电路的时钟频率。注意，公式(3-9)式和公式(3-10)式是建立在简单近似的基础上的，因此只适用于一级近似的分析。

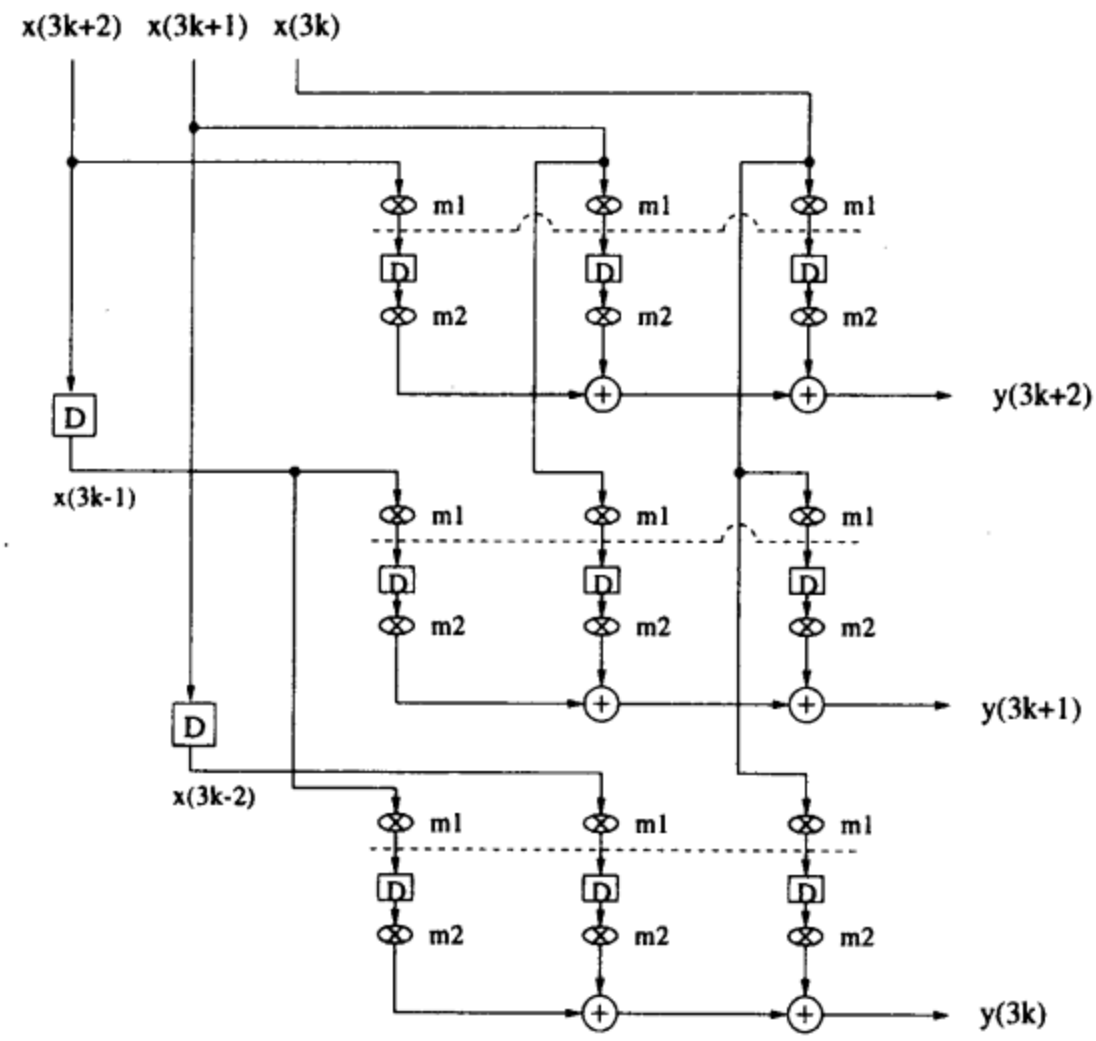


图3-15 3阶FIR滤波器细粒度流水线与并行处理相结合的架构

3.4.1 用流水线降低功耗

如前所述，流水线结构可以用来降低FIR滤波器的功耗。令

$$P_{seq} = C_{total} V_0^2 f \tag{3-11}$$

70 ~ 74

表示原始滤波器的功耗。注意 $f = 1/T_{seq}$，其中 T_{seq} 是原始时序滤波器的时钟周期。现在考虑一个 M 级流水线系统，其关键路径缩短为原始路径长度的 $1/M$，一个时钟周期内充放电电容减小为 C_{charge}/M，注意总电容并没有变化。如果时钟速度保持不变，即时钟频率 f 保持不变，在原来对电容 C_{charge} 充放电的同样时间内，现在只需要对 C_{charge}/M 进行充放电（见图3-16）。这意味着，电源电压可以降低到 βV_0，其中 β 是一个小于1的常数。这样，流水线滤波器的功耗将为

$$P_{pip} = C_{total} \beta^2 V_0^2 f = \beta^2 P_{seq}. \tag{3-12}$$

因此，和原始系统相比流水线系统的功耗降低了 β^2 倍。

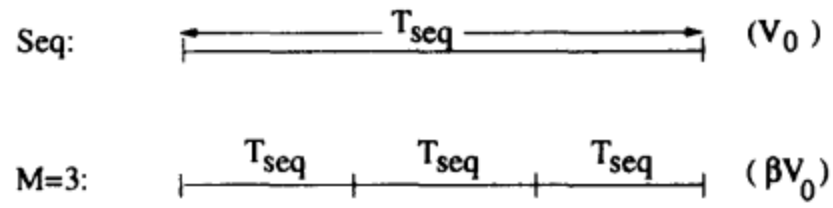


图3-16 原始系统和3级流水线系统的关键路径

功耗降低因子 β 可以通过考察原始滤波器和流水线滤波器传播延时之间的关系来确定。原始滤波器的传播延时是

$$T_{seq} = \frac{C_{charge}V_0}{k(V_0 - V_t)^2} \tag{3-13}$$

流水线滤波器的传播延时是

$$T_{pip} = \frac{\frac{C_{charge}}{M}\beta V_0}{k(\beta V_0 - V_t)^2} \tag{3-14}$$

应该注意的是，时钟周期 T_{clk} 通常被设置为等于电路中的最大传播延时 T_{pd}。因为对于这两个滤波器来说使用相同的时钟速度，根据公式(3-13)和公式(3-14)，从下列二次方程可以解出 β，

$$M(\beta V_0 - V_t)^2 = \beta(V_0 - V_t)^2 \tag{3-15}$$

一旦得到了 β，流水线滤波器降低的功耗就可以由公式(3-12)算出。

例3.4.1 考虑图3-7所示3阶FIR滤波器及其细粒度流水线版本（见图3-8）。假设乘法操作需要10单位时间(u.t.)，加法操作需要2 u.t.。出于估计功耗的目的，假定乘法器的电容是加法器的5倍。在细粒度流水线结构的滤波器中，乘法器被分成2部分，$m1$和$m2$，它们的计算时间分别为6 u.t.和4 u.t.，它们的电容分别为加法器的3倍和2倍。假设器件阈值电压为0.6V。同时假定非流水线滤波器工作在5V电源电压下。

75

(a) 如果时钟周期保持不变，流水线滤波器的电源电压为多少？

(b) 流水线结构滤波器的功耗是原滤波器的百分之多少？

解

(a) 令 C_M 为一个乘法器的电容，C_A 为一个加法器的电容。对于原始滤波器，关键路径上的充电电容为

$$C_{charge} = C_M + C_A = 6C_A \tag{3-16}$$

对于流水线滤波器，关键路径上的充电电容为

$$C_{charge} = C_{m1} = C_{m2} + C_A = 3C_A \tag{3-17}$$

其中 C_{m1} 和 C_{m2} 分别是乘法器的 $m1$ 部分和 $m2$ 部分的电容。这里流水线级数 $M = 2$，流水线滤波器的充电电容是原始滤波器的一半。假定流水线滤波器工作在电源电压 βV_0下，将 $M = 2$，$V_0 = 5.0$，$V_t = 0.6$代入公式(3-15)：

$$50\beta^2 - 31.36\beta + 0.72 = 0 \tag{3-18}$$

解公式(3-18)得到

$$\beta = 0.6033 \text{ 或 } \beta = 0.0239 \tag{3-19}$$

注意，$\beta = 0.0239$是不可用的解，因为这种情况对应的电源电压为0.1195V，低于阈值电压，器件在这种电源电压下永远是关断的。所以，流水线滤波器的电源电压应该是

$$V_{pip} = \beta V_0 = 3.0165 \text{ V} \tag{3-20}$$

(b) 因为流水线滤波器的总电容与原始滤波器的总电容相同，而且两个滤波器工作在同样的时钟频率下，根据公式(3-12)，

$$Ratio = \beta^2 = 36.4\% \tag{3-21}$$ ■

3.4.2 用并行处理降低功耗

和流水线一样，并行处理也可以通过降低电源电压来降低功耗。在一个 L 路并行系统中，充电电容通常不变，而总电容增大 L 倍。为了保持同样的采样速率，L 级并行电路的时钟周期必须增加到 LT_{seq}，其中 T_{seq} 是由公式(3-13)决定的时序电路的传播延时。这意味着 C_{charge} 的充电时间是 LT_{seq} 而不是 T_{seq}。换句话说，同样的电容有了更长的充电时间（见图3-17）。这就意味着电源电压可以降低到 βV_0。 76

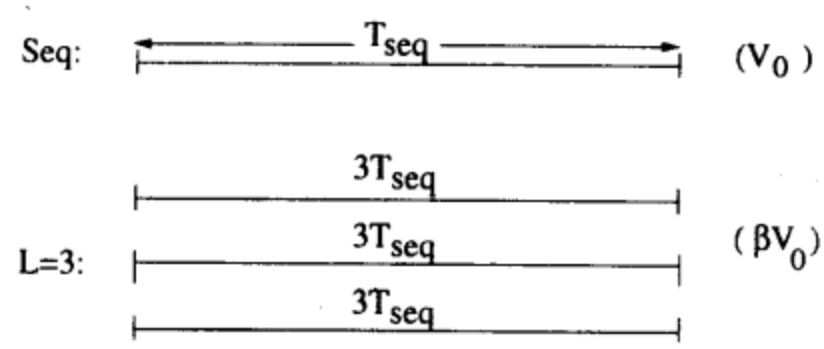


图3-17 顺序流水线系统和3级流水线系统的关键路径

对传播延时的考虑可以再次用来计算 L 级并行系统的电源电压。原始系统的传播延时由公式(3-13)给出，而 L 级并行系统的传播延时由下式给出：

$$LT_{seq} = \frac{C_{charge}\beta V_0}{k(\beta V_0 - V_t)^2} \tag{3-22}$$

根据公式(3-13)和公式(3-22)，可以得到下列二次方程来求出 β：

$$L(\beta V_0 - V_t)^2 = \beta(V_0 - V_t)^2 \tag{3-23}$$

一旦求出 β，L 路并行系统的功耗可以计算如下：

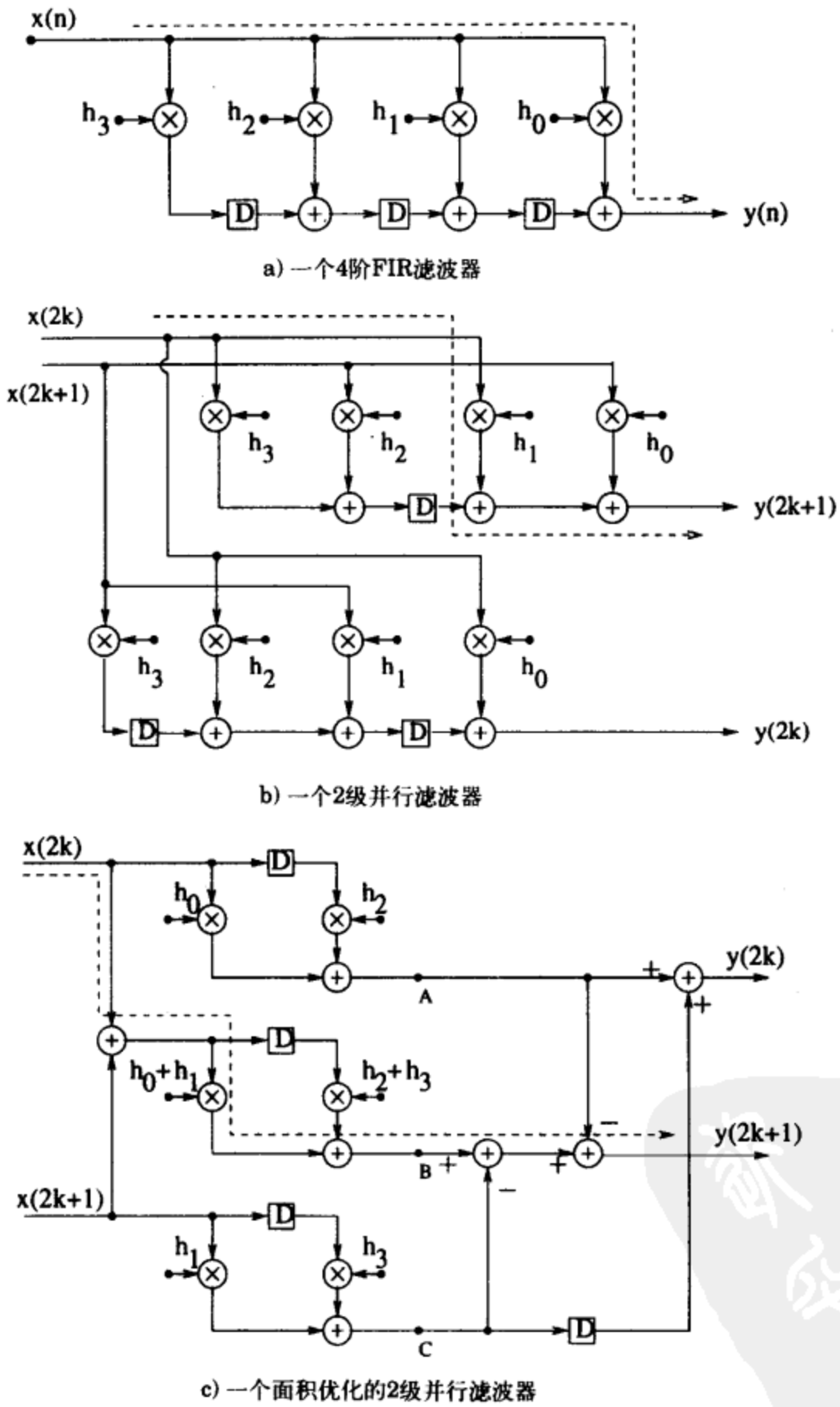
$$\begin{aligned} P_{par} &= (LC_{charge})(\beta V_0)^2 \frac{f}{L} \\ &= \beta^2 C_{charge} V_0^2 f \\ &= \beta^2 P_{seq} \end{aligned} \tag{3-24}$$

其中 P_{seq} 是由公式(3-11)给出的原始时序系统的功耗。所以，和流水线系统一样，L 路并行系统的功耗降低为原时序系统的 β^2 倍。

例3.4.2 考虑图3-18a所示的4阶FIR滤波器，其2级并行结构如图3-18b所示。并行滤波器正

好包含了2个复制的原始滤波器。图3-18中的虚线表示关键路径。假定乘法操作需要8 u.t.，加法操作需要1 u.t.。为计算功耗，假定乘法器的电容是加法器电容的8倍，两种结构都工作在同样的采样周期9 u.t.下，器件阈值电压为0.45V。同时还假定原始的时序线滤波器工作在3.3V原始电源电压下。

77



a) 一个4阶FIR滤波器

b) 一个2级并行滤波器

c) 一个面积优化的2级并行滤波器

图 3-18

(a) 2级并行滤波器的电源电压为多少？

(b) 2级并行滤波器的功耗是原始滤波器的百分之多少？

解

(a) 令 C_M 为一个乘法器的电容，C_A 为一个加法器的电容。对于原始滤波器，关键路径上的充电电容为：

$$C_{charge} = C_M + C_A = 9C_A \tag{3-25}$$

对于2级并行滤波器，关键路径上的充电电容为：

$$C_{charge} = C_M + 2C_A = 10C_A \tag{3-26}$$

假定流水线滤波器工作在电源电压 βV_0 下。注意到原始的时序滤波器和2级并行滤波器的充电电容不相等，我们不能直接使用公式(3-23)。根据公式(3-13)，2级并行滤波器的传播延时由下式决定：

$$\begin{aligned} T_{seq} &= \frac{9C_A V_0}{k(V_0 - V_t)^2} \\ T_{par} &= \frac{10C_A \beta V_0}{k(\beta V_0 - V_t)^2} \end{aligned} \tag{3-27}$$

原始的时序滤波器的关键路径是1次乘法和1次加法，即9 u.t.，这意味着采样周期 $T_{sample} = T_{seq}$。将等式 $T_{par} = 2T_{sample} = T_{seq}$ 和公式(3-27)结合起来，有：

$$5\beta(V_0 - V_t)^2 = 9(\beta V_0 - V_t)^2 \tag{3-28}$$

将 $V_0 = 3.3\text{V}$、$V_t = 0.45\text{V}$ 代入公式(3-28)得到：

$$98.01\beta^2 - 67.3425\beta + 1.8225 = 0 \tag{3-29}$$

解公式(3-29)，得

$$\beta = 0.6589 \text{ 或 } \beta = 0.0282 \tag{3-30}$$

对于2级并行滤波器，惟一可行的电源电压是 $\beta V_0 = 2.17437\text{V}$。(另一个根 $\beta = 0.0283$ 被舍去，因为 $\beta V_0 = 0.09306\text{V}$，小于阈值电压。)

(b) 根据公式(3-24)

$$Ratio = \beta^2 = 43.41\% \tag{3-31}$$ ■

78 ~ 79

例3.4.3 考虑图3-18a中的4阶FIR滤波器，其2级并行结构如图3-18b。图3-18c中示出的是另一个2级并行结构，其效率高于图3-18b中的2级并行结构，因为它需要6个乘法器而不是图3-18b中的2级并行结构所需要的8个。各项假定如同例3.4.2，目标是比较在采样周期9 u.t.的情况下原始的时序滤波器和它的2级并行结构的功耗情况。其中，2级并行滤波器可以使用较低的电源电压，以达到9 u.t.的有效采样周期，但是电源电压必须大于或等于0.9V。

(a) 跟踪计算路径，验证2级并行滤波器能够正确计算输出 $y(2k)$ 和 $y(2k+1)$。

(b) 计算使2级并行滤波器能够正常工作的电源电压。

(c) 使用(b)的结果，计算2级并行滤波器的功耗是原始时序滤波器功耗的百分之多少。

解

(a) 所给定的4阶FIR滤波器的系统方程如下：

$$y(n) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + h_3 x(n-3) \tag{3-32}$$

定义节点 A、B 和 C 的输出分别为 y_A、y_B 和 y_C，得到：

$$\begin{aligned} y_A &= h_0x(2k) + h_2x(2k-2) \\ y_B &= (h_0+h_1)(x(2k)+x(2k+1)) + (h_2+h_3)(x(2k-2)+x(2k-1)) \\ y_C &= h_1x(2k+1) + h_3x(2k-1) \end{aligned} \tag{3-33}$$

那么

$$\begin{aligned} y(2k) &= y_A + [y_C \ after\ 1\ block\ delay] \\ &= h_0x(2k) + h_1x(2k-1) + h_2x(2k-2) + h_3x(2k-3) \\ y(2k+1) &= y_B - y_A - y_C \\ &= h_0x(2k+1) + h_1x(2k) + h_2x(2k-1) + h_3x(2k-2) \end{aligned} \tag{3-34}$$

很容易验证上述等式满足系统方程公式(3-32)。

(b) 令 C_M 表示一个乘法器的电容，C_A 表示一个加法器的电容。对于时序滤波器，关键路径上的充电电容由公式(3-25)给出。对于2级并行滤波器，关键路径上的充电电容为：

$$C_{charge} = C_M + 4C_A = 12C_A \tag{3-35}$$

80 由例3.4.2，已经知道 $T_{sample} = T_{seq} = 9$ u.t.。为了使2级并行滤波器在采样周期9 u.t.下工作，必须满足 $T_{par} = 2T_{sample} = 2T_{seq}$，这导致：

$$\frac{2\cdot 9C_AV_0}{k(V_0-V_t)^2} = \frac{12C_A\beta V_0}{k(\beta V_0 - V_t)^2} \tag{3-36}$$

将 $V_0 = 3.3$V、$V_t = 0.45$V代入公式(3-36)，得到：

$$32.67\beta^2 - 25.155\beta + 0.6075 = 0 \tag{3-37}$$

解公式(3-37)，得到：

$$\beta = 0.745 \ \text{或}\ \beta = 0.025 \tag{3-38}$$

2级并行滤波器应该工作在电源电压 $\beta V_0 = 2.4585$V下。(另一个根无效，因为 $\beta V_0 = 0.0825$V小于阈值电压。)

(c) 令 $C_{total}^{(seq)}$ 为4阶FIR滤波器的总电容，令 $C_{total}^{(par)}$ 为2级并行滤波器的总电容。根据公式(3-10)，4阶FIR滤波器和2级并行滤波器的功耗如下：

$$P_{seq} = C_{total}^{(seq)}V_0^2 f_{seq},\ P_{par} = C_{total}^{(par)}V_{par}^2 f_{par} \tag{3-39}$$

其中 $C_{total}^{(seq)} = 4C_M + 3C_A = 35C_A$，$C_{total}^{(par)} = 6C_M + 7C_A = 55C_A$。由(b)中的解答，得到 $f_{par} = f_{seq}/2 = f_s/2$，其中 f_s 是采样频率。因此，有：

$$\begin{aligned} P_{seq} &= 35C_A\cdot V_0^2\cdot f_s \\ P_{par} &= 55C_A\cdot \beta^2V_0^2\cdot \frac{1}{2}f_s \end{aligned} \tag{3-40}$$

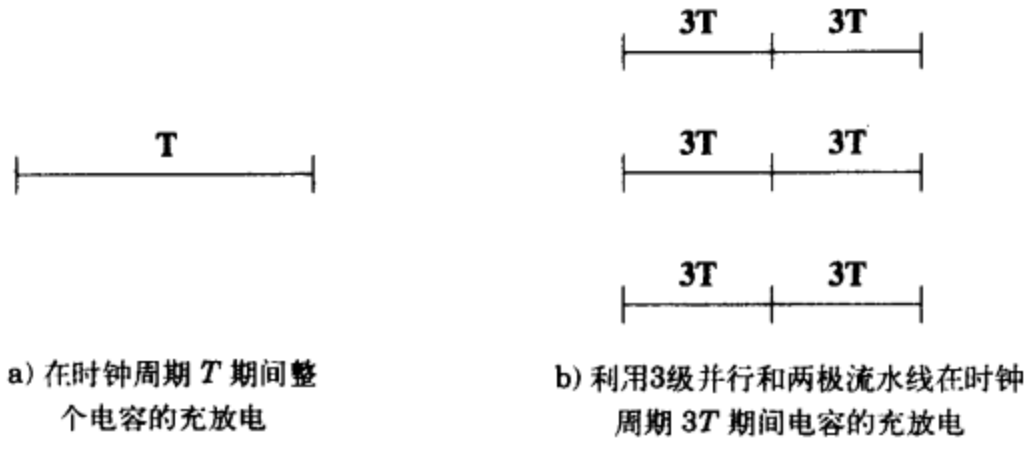
所以

$$Ratio = \frac{P_{par}}{P_{seq}} = \frac{0.5\cdot 55\beta^2}{35} = 43.6\% \tag{3-41}$$ ■

注意，虽然图3-18b中和图3-18c中的结构所消耗的功率相近，但是图3-18c中的结构消耗的面积却小得多。

3.4.3 流水线和并行处理的结合

流水线技术和并行处理技术可以结合起来降低功耗。原理是一样的，即流水线降低1个时钟周期内充电放电电容，而并行处理则增加对原电容的充电放电时钟周期（见图3-19）。



a) 在时钟周期 T 期间整个电容的充放电

b) 利用3级并行和两极流水线在时钟周期 $3T$ 期间电容的充放电

图 3-19

并行流水线滤波器的传播延时如下:

$$LT_{pd} = \frac{(C_{charge}/M)\beta V_0}{k(\beta V_0 - V_t)^2} = \frac{LC_{charge}V_0}{k(V_0 - V_t)^2}. \tag{3-42}$$

根据该方程，得到下列二次方程:

$$ML(\beta V_0 - V_t)^2 = \beta(V_0 - V_t)^2 \tag{3-43}$$

它可用来求解 β。作为一个例子，考虑 $L = M = 2$, $V_0 = 5\text{V}$, $V_t = 0.6\text{V}$ 的情况。利用公式(3-43)，得出 β 约等于0.4，这表示功耗降低为原来的0.16。应该注意的是，电源电压并不能通过使用更多级的流水线和并行处理而无限地降低，因为存在一个由工艺参数和噪声容限决定的电源电压下限。

3.5 结论

本章介绍了非递归数字滤波器中的流水线和并行处理方法。这两种方法都可以用来提高滤波器的采样频率。在流水线中，流水线锁存器放置在SFG中的前馈割集处，使关键路径的计算时间降低。其结果使时钟频率得以提高，从而采样频率提高。在并行处理中，复制原始的串行系统的硬件，得到一个MIMO并行系统。在这种情况下，时钟频率不变，采样频率却增加了。此外，还说明了流水线和并行处理在低功耗设计中的应用。基本思路是利用降低电源电压的方法，用提高后的采样速度换取功耗的降低。利用快速算法，并行FIR滤波器能够以小于并行级数线性增加的硬件代价实现（见第9章）。使用“超前”(look-ahead)技术的递归数字滤波器中的流水线和并行处理将在第10章中介绍。

81 ~ 82

3.6 习题

1. 考察图3-20中的DFG，假定每步操作需要时间都是 T。
   (a) 该系统可达到的最大采样速率是多少?
   (b) 在适当的前馈割集位置插入流水线寄存器，使系统的采样速度能近似达到 $1/T$，请清楚地标出前馈割集，并计算所需要流水线寄存器的总数目。

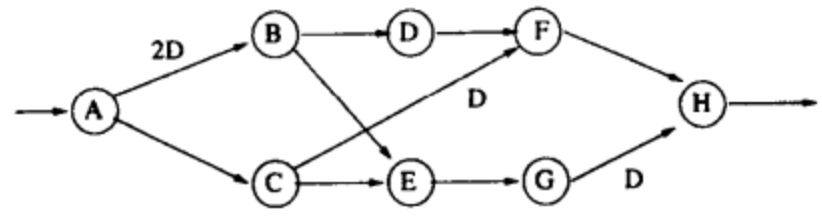


图3-20 习题1数据流图

2. 考察图3-21中所示IIR数字滤波器框图。假定乘法操作时间为2 u.t.，加法操作时间为1 u.t.。

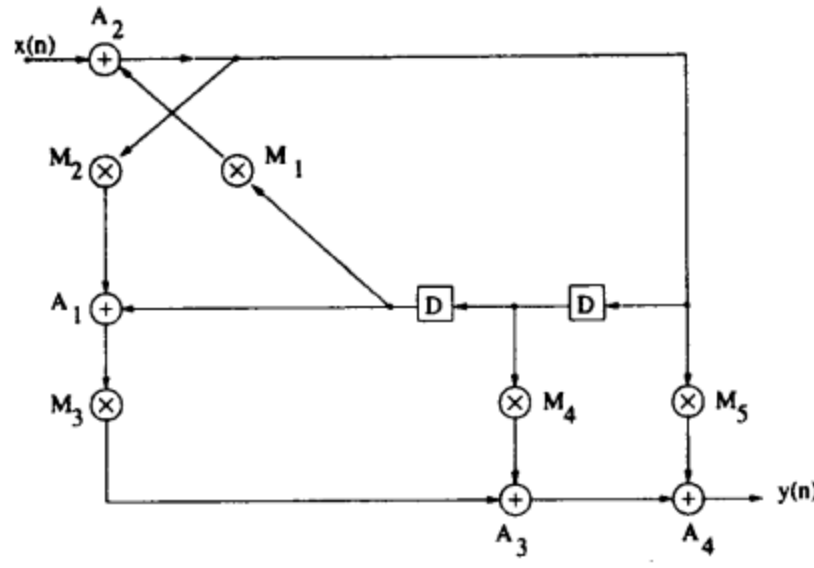


图3-21 习题2数字IIR滤波器

83

(a) 计算IIR滤波器的关键路径。

(b) 在适当的前馈割集处中插入锁存器，使IIR滤波器变成流水线结构，其关键路径延时降低为3 u.t.。

3. 考察图3-22所示的非递归信号处理结构，找出一种与该算法等价的数据广播实现方法，来改善系统的速度，不要使用附加的锁存器，计算广播结构的吞吐率或采样速度。(提示：图3-22中的2输入1输出系统中不能进行转置操作。)

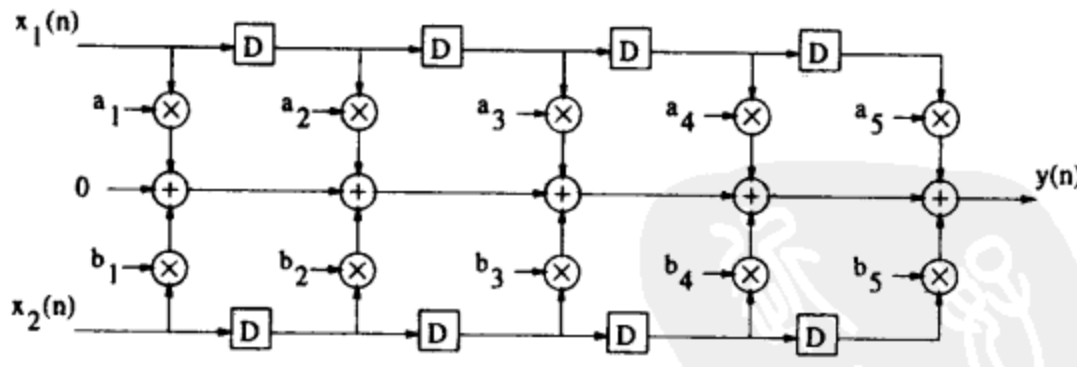


图3-22 习题3数据流图

4. 考察图3-23中的规模为3 × 3的2维FIR数字滤波器。找出一种等价的数据广播FIR滤波器结构。

$$y(n_1, n_2) = \sum_{i=0}^{2}\sum_{j=0}^{2} a_{ij}x(n_1 - i, n_2 - j)$$

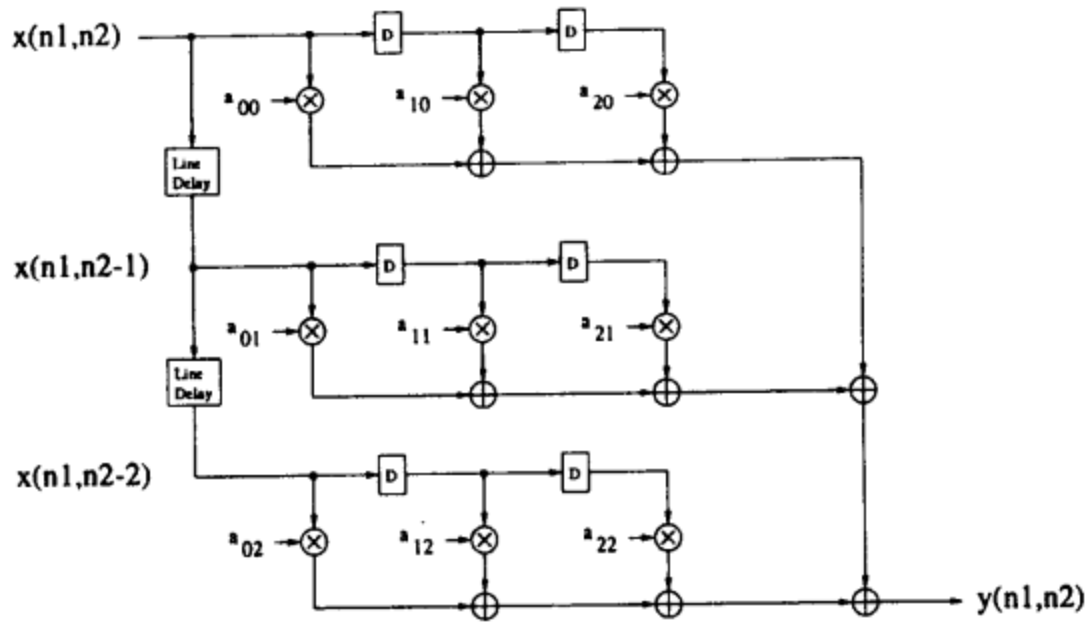


图3-23 习题4的二维FIR滤波器

5. 考察FIR滤波器一个直接形式的实现

$$y(n) = ax(n) + bx(n-2) + cx(n-3) \tag{3-44}$$

假设一次乘加操作所需时间是 T。

(a) 在该滤波器中加入流水线，使时钟周期约等于 T。

(b) 取分组尺寸为3，画出分组滤波器的结构。用流水线实现该分组滤波器，使时钟周期约为 T，问系统采样速率是多少?

(c) 用流水线实现(b)中的分组滤波器，使时钟周期约为 $T/2$，指出适当的割集，并清楚标出输出。这时系统采样速率是多少?

6. 使用广播滤波器结构重做习题5。在每种情况下，你能节省多少锁存器?

7. 考察6阶FIR滤波器

$$y(n) = ax(n) + bx(n-4) + cx(n-6) \tag{3-45}$$

(a) 画出该滤波器的拓扑结构，使时钟周期不超过一次乘加的时间。注意不要增加新的锁存器。

(b) 对(a)中结构，画出分组规模为3时的结构。重新分配这个分组结构，使时钟周期是一次乘加时间的四分之一。其中，假定乘法计算时间是加法计算时间的3倍。

8. 考察递归滤波器

$$x(n) = ax(n-2) + u(n) \tag{3-46}$$

(a) 将乘加操作分成2级流水线实现，要求首先将乘加分开成2个部分，并且重新分配环路中的延时元件。

(b) 使用相同的硬件将公式(3-46)和公式(3-47)中的计算

$$y(n) = by(n-2) + v(n) \tag{3-47}$$

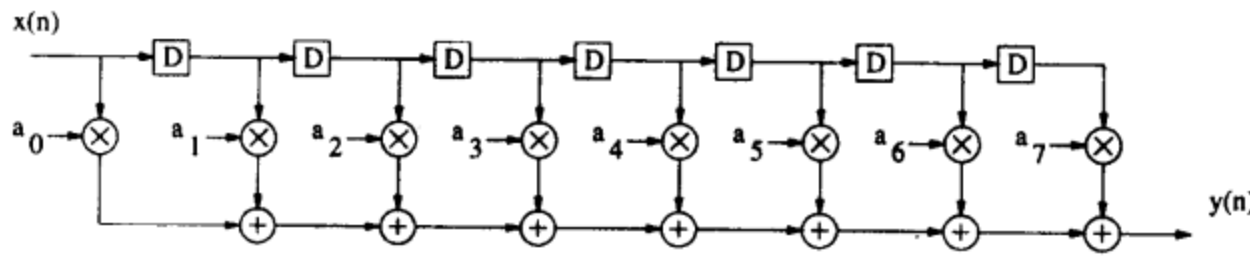
交替进行。将乘加操作用4级流水线实现，画出这种实现所需的全部电路。

9. 一个系统的功耗需要利用流水线方法至少降到原来的1/5。对于阈值电压为0.4V、初始电源电压为5V的情况来说，该系统需要采用几级流水线? 流水线系统的电源电压是多少?

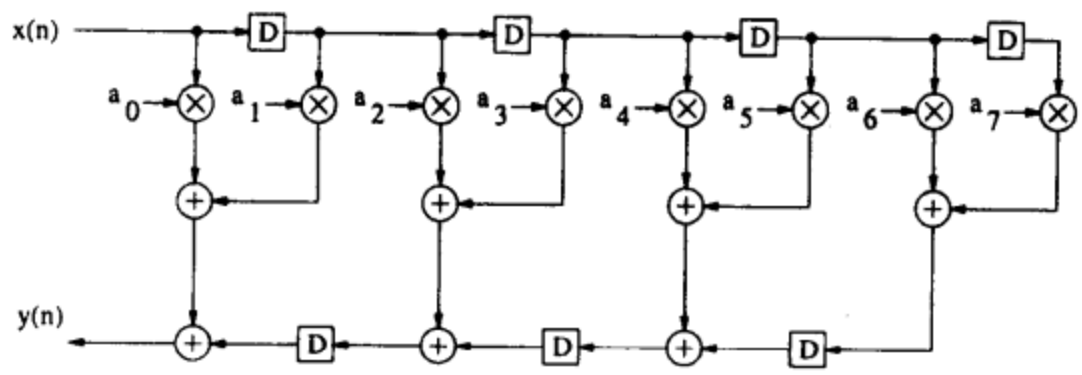
84 ~ 85

10. 一个8阶FIR滤波器的两种实现如图3-24。假定乘法器的关键路径(或传播延时)为加法器的两倍，即 $T_m = 2T_a$，因而乘法器充电电容是加法器的两倍。进而假定乘法器的总电容是加法器的10倍，即 $C_m = 10C_a$。图3-24a中直接形式结构的关键路径是 $T_m + 7T_a = 9T_a$。图3-24b中的结构可以工作在较低的电源电压下，而保持时钟周期或采样周期 $9T_a$ 的约束。因此图3-24b的结构可以用来降低功耗。假如图3-24a的结构工作在4V电源电压下，工艺阈值电压为0.5V，电源电压大于1.2V以保证可容忍的噪声容限。

请问：当图3-24b所示结构达到所希望的 $9T_a$ 采样周期时，最小的工作电源电压是多少？与图3-24a的结构相比，计算图3-24b所示结构在功耗上降低的百分比。其中可以忽略在计算关键路径或功耗时延时元件的传播延时和电容。



a)



b)

86

图3-24 习题10中一个8阶FIR滤波器的两种实现

11. 考虑总电容为 C_{total} 的数据通路。这个数据通路采用 M 级流水线。用 C_{latch} 表示一级流水线所用锁存器的总电容。流水线系统工作在较低的电源电压下以降低功耗。假设两个系统都工作在相同速度下，且锁存器的传播延时可忽略不计。令 $C_{total} = 10\ C_{latch}$，$V_{dd} = 4V$，$V_t = 0.6V$。对不同的 M 计算流水线系统相对原始时序系统功耗的百分比。功耗最小时所对应的最佳 M 是多少？

12. 计算某一计算系统的功耗降低。其中该系统采用了4级流水线和块尺寸为4的并行处理结构，并保持了与原始系统相同的采样速率。这里假定该原始系统工作在5V电源电压，CMOS工艺的阈值电压 V_t 为0.4V。计算这个并行-流水线系统与原始系统相比的功耗。该并行-流水线系统的电源电压是多少？

13. 考虑一个工作在时钟式采样周期为 T 的FIR滤波器。该习题不必知道滤波器的阶就可以求解。滤波器电路工作在5V电源电压下，采用阈值电压 V_t 为0.4V的CMOS工艺。用户要求滤波器的采样速度增加4倍，即新系统应当达到 $T/4$ 的采样周期。此外功耗

也应当降低，即使可能以增加面积为代价。为了降低功耗可以使用较低的电源电压，假定一个可调电源能够提供1.0V到5V的电压。电源电压不能低于1.0V。

同时提高速度并降低功耗可以通过块处理达到，块尺寸是4的倍数。例如，我们可以用一个块尺寸为8的并行FIR滤波器，使这个系统工作在$2T$时钟周期下，从而达到采样周期为$T/4$的目的。类似地，我们可以用块尺寸为$4p$的并行滤波器，使滤波器工作在pT时钟周期下，此处p是一个任意正整数。这样的滤波器就能够工作在较低的电源电压下来降低功耗。

(a) 选择什么样的p值或块尺寸可以得到功耗最低的电路？计算采用该p值时的电源电压与功耗。

(b) 如果目标不是降低功耗，而是减小面积-功耗积，应当选取什么样的p值或块尺寸？计算采用此p值时的电源电压、功耗和面积-功耗积。对于面积计算，假定与乘法器相比加法器的代价可以忽略。 87

14. 把图3-18c所示的电路用流水线实现，在适当的前馈位置插入锁存器，使关键路径缩短为1/2，到6 u.t.。认为电路元件的电容是正比于其路径的长度。计算该电路的功耗与图3-18a所示原始时序电路功耗的百分比。(利用与例3.4.2和例3.4.3中相同的参数)

15. 考虑利用流水线和并行处理而以相同速度工作的电路的功耗降低。令 V_0 为原始时序系统的电源电压，β 代表一个 L 级并行、M 级流水线系统的电源电压缩小因子，即该系统工作在 βV_0 电源电压下。令 β_1 为相同速度工作的 M 级流水线系统的电源电压缩小因子，即该系统工作在 $\beta_1 V_0$ 电源电压下。令 β_2 是 L 级并行系统的电源电压缩小因子，该 L 级并行系统与工作在 $\beta_1 V_0$ 电源电压下的时序电路在相同速度下工作。证明 $\beta = \beta_1 \beta_2$。

参考文献

1. J. Hennessy and D. Patterson, *Computer Architecture A Quantitative Approach, 2nd ed.*. Morgan Kaufmann Publishers, 1996.
2. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.
3. P. M. Kogge, *The Architecture of Pipelined Computers*. McGraw-Hill, 1981.
4. H. T. Kung, "Why systolic architectures?" *IEEE Computers Magazine*, vol. 15, pp. 37–45, Jan. 1982.
5. U. Banerjee et al., "Time and parallel processor bounds for fortran like loops," *IEEE Trans. on Computers*, vol. 28, pp. 660–670, 1979.
6. N. Jouppi and D. Wall, "Available instruction level parallelism for superscalar and super-pipelined machines," in *Proc. of 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, (Boston), pp. 272–282, May 1989.
7. M. Lam, "Software pipelining: an effective scheduling technique for VLIW machines," in *Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, (Atlanta), pp. 318–328, June 1988.
8. D. A. Padua and M. J. Wolfe, "Advanced compiler organizations for supercomputers," *Communications of the ACM*, vol. 29, pp. 1184–1201, 1986.

9. B. R. Rau et al., "Efficient code generation for horizontal architectures: compiler techniques and architectural support," in *Proc. of 9th International Symposium on Computer Architecture*, 1982.

88

10. M. E. Wolf and M. S. Lam, "A loop transformation theory and an algorithm to maximize parallelism," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, pp. 452–471, 1991.

11. K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters–part I: pipelining using scattered look-ahead and decomposition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1099–1117, July 1989.

12. K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters–part II: pipelined incremental block filtering," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1118–1135, July 1989.

13. C. W. Wu and P. R. Cappello, "Application-specific CAD of VLSI second-order sections," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 813–825, 1988.

14. A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–483, April 1992.

89

第4章 重 定 时

4.1 引言

重定时[1]是一种变换技术，用于改变电路中延时元件的位置，但不会影响电路的输入/输出特性。例如，考虑图4-1a的IIR滤波器，用下式描述:

$$\begin{aligned} w(n) &= ay(n-1)+by(n-2) \\ y(n) &= w(n-1)+x(n) \\ &= ay(n-2)+by(n-3)+x(n) \end{aligned}$$

图4-1b的滤波器用下式描述:

$$\begin{aligned} w_1(n) &= ay(n-1) \\ w_2(n) &= by(n-2) \\ y(n) &= w_1(n-1)+w_2(n-1)+x(n) \\ &= ay(n-2)+by(n-3)+x(n) \end{aligned}$$

虽然图4-1a和图4-1b的延时元件在不同的位置，但是这两个滤波器的输入/输出特性都是相同的，二者可以利用重定时技术互相导出。

重定时在同步电路设计中有很多应用，包括缩短电路的时钟周期、减少电路中寄存器的数目、降低电路的功耗和逻辑综合的规模。缩短时钟周期和减少寄存器数目的重定时在本章中会详细讨论。对于逻辑综合的重定时技术不在本书讨论范围之内。

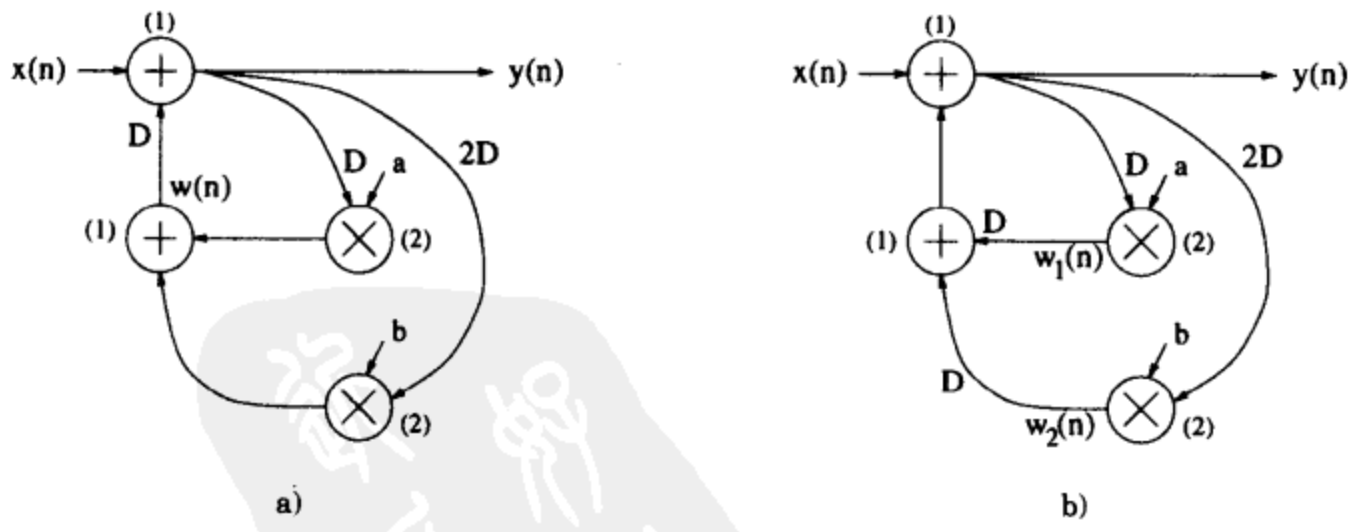


图4-1 一个IIR滤波器的两种版本，括号中的数字表示各节点的计算时间

重定时可以通过减少关键路径的计算时间来增加电路的时钟速率。大家还记得关键路径的定义是: 包括零延时在内的所有路径中具有最长计算时间的路径。而关键路径计算时间是电路的时钟周期的下界。图4-1a滤波器的关键路径通过了一个乘法器和一个加法器，计算时间为3 u.t.，因此这个滤波器不能在低于3 u.t.的时钟周期下工作。而图4-1b中重定时后的滤波器，其关键路径通过了两个加法器，计算时间为2 u.t.，因此这个滤波器能够工作在2 u.t.

的时钟周期下。通过对图4-1a的滤波器进行重定时，成为图4-1b的滤波器，时钟周期从3 u.t.缩小为2 u.t.，即缩小了33%。时钟周期最小化的一个多项式时间算法将在第4.4.2节中描述。

重定时可以改变电路中的寄存器数目。图4-1a滤波器用了4个寄存器，而图4-1b滤波器用了5个寄存器。因为重定时能够既影响时钟周期又影响寄存器数，所以有时希望同时考虑这两个参数。在给定时钟周期下使寄存器数最小化的多项式时间算法将在第4.4.3节中描述。

重定时可以用来通过减少开关动作降低电路功耗，这种开关动作造成静态CMOS电路的动态功耗[2]。在具有大电容的节点输入处插入寄存器能够减少这些节点的开关动作率，从而导出低功耗解决方案(见第17.5.4节)。

92

在第4.2节中给出重定时的数学描述，并讨论重定时的某些性质。用于求解不等式系统的技术在第4.3节介绍。这些技术的应用放在第4.4节，描述对一个电路进行重定时的算法如何达到诸如时钟周期最小化等目标。

4.2 定义与性质

4.2.1 重定时的定量描述

重定时将一个电路 G 映射到一个重定时的电路 G_r。在本章重定时的上下文中，电路、图和DFG这些术语经常交叉使用在本章中。一个重定时的解用图中每个节点 V 的值 $r(V)$ 来表征。令 $w(e)$ 表示在原始图 G 中边 e 的权重，$w_r(e)$ 表示在重定时后的图 G_r 中边 e 的权重。重定时后图中边 $U \xrightarrow{e} V$ 的权重由原始图中边的权重计算如下:

$$w_r(e) = w(e) + r(V) - r(U) \tag{4-1}$$

为了说明某些正式的重定时概念，图4-1a中的滤波器表示为图4-2a，而图4-1b重定时后的滤波器表示为图4-2b。重定时的值 $r(1)=0$, $r(2)=1$, $r(3)=0$, $r(4)=0$ 可以用来从图4-2a的DFG得到图4-2b重定时后的DFG。例如，重定时后DFG中边 $3 \xrightarrow{e} 2$ 包含延时数为:

93

$$\begin{aligned} w_r(3 \xrightarrow{e} 2) &= w(3 \xrightarrow{e} 2) + r(2) - r(3) \\ &= 0 + 1 - 0 = 1 \end{aligned}$$

而边 $2 \xrightarrow{e} 1$ 包含延时数为:

$$\begin{aligned} w_r(2 \xrightarrow{e} 1) &= w(2 \xrightarrow{e} 1) + r(1) - r(2) \\ &= 1 + 0 - 1 = 0 \end{aligned}$$

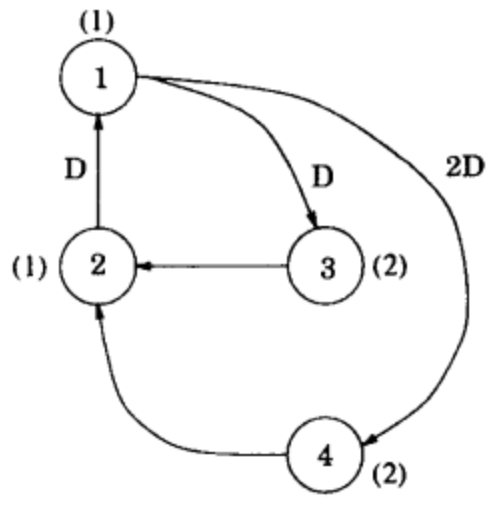
如果 $w_r(e) \geq 0$ 的条件对所有边都保持成立，重定时的解就是可行的。把图4-2a映射到图4-2b的解是可行的，因为图4-2b所有边的权重都是非负的。而 $r(1)=0$, $r(2)=-1$, $r(3)=0$ 和 $r(4)=0$ 这样的解则是不可行的，因为在重定时后的系统中边3与2包含延时是负的。

$$\begin{aligned} w_r(3 \xrightarrow{e} 2) &= w(3 \xrightarrow{e} 2) + r(2) - r(3) \\ &= 0 + (-1) - 0 = -1 \end{aligned}$$

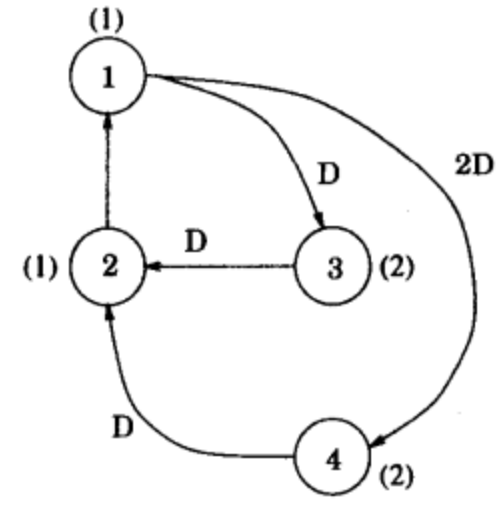
4.2.2 重定时的性质

重定时的几个性质可以从重定时方程（4-1）导出。在考虑这些性质之前先回顾一下路径和环

路的概念。一条路径是一个由边和节点顺序排成的序列 $V_0 \xrightarrow{e_0} V_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-2}} V_{k-1} \xrightarrow{e_{k-1}} V_k$。路径 p 的权重是 $w(p)=\sum_{i=0}^{k} w(e_i)$，而路径的计算时间是 $t(p)=\sum_{i=0}^{k} t(V_i)$。一个环路是一个封闭的路径 $V_0 \xrightarrow{e_0} V_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-2}} V_{k-1} \xrightarrow{e_{k-1}} V_0$。环路 c 的权重为 $w(c)=\sum_{i=0}^{k-1} w(e_i)$，而环路的延时为 $t(c)=\sum_{i=0}^{k-1} t(V_i)$。



a) 一个DFG



b) 利用 $r(1)=0, r(2)=1, r(3)=0$ 和 $r(4)=0$ 获得的重定时后的DFG

图 4-2

性质4.2.1 重定时的路径 $p=V_0 \xrightarrow{e_0} V_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-2}} V_{k-1} \xrightarrow{e_{k-1}} V_k$ 的权重是 $w_r(p)=w(p)+r(V_k)-r(V_0)$。

重定时路径的权重是:

$$
\begin{aligned}
w_r(p) &= \sum_{i=0}^{k-1} w_r(e_i) \\
&= \sum_{i=0}^{k-1} (w(e_i)+r(V_{i+1})-r(V_i)) \\
&= \sum_{i=0}^{k-1} w(e_i) + \left(\sum_{i=0}^{k-1} r(V_{i+1}) - \sum_{i=0}^{k-1} r(V_i)\right) \\
&= w(p)+r(V_k)-r(V_0)
\end{aligned}
$$

94

例如，在图4-2a中路径2→1→3的延时数为2，而在图4-2b的重定时DFG中这一路径的延时数为 $2+r(3)-r(2)=2+0-1=1$。

性质4.2.2 重定时不改变环路中的延时数。

这是性质4.2.1的一个特殊情况，此处 $V_k=V_0$。重定时后的环路 c 的权重是 $w_r(c)=w(c)+r(V_0)-r(V_0)=w(c)$。在图4-2中，环路1→3→2→1在未重定时和重定时的DFG中都包含了2个延时，而环路1→4→2→1在未重定时和重定时的DFG中都包含3个延时。

性质4.2.3 重定时不改变DFG中的迭代边界。

性质4.2.4 每个节点的重定时值增加一个常数 j 不会改变从 G 到 G_r 的映射。

对每个节点用 $r(V)+j$ 代替 $r(V)$ 之后，G_r 中重定时后的边 $U \xrightarrow{e} V$ 的权重为:

$$w_r(e) = w(e) + (r(V) + j) - (r(U) + j) = w(e) + r(V) - r(U)$$

该权重对于 j 的任何数值（包括 $j = 0$）都是相同的。回顾由图4-2a的未重定时DFG得到图4-2b的重定时DFG使用了重定时值 $r(1)=0$，$r(2)=1$，$r(3)=0$ 和 $r(4)=0$。例如，给这些重定时值加一个常数“-38”，得到新的重定时值 $r(1)=-38$，$r(2)=-37$，$r(3)=-38$ 和 $r(4)=-38$。这些值可以用来从图4-2a产生图4-2b的重定时DFG。

4.3 不等式求解系统

给定一组 N 个变量的 M 个不等式，其中每个不等式的形式为 $r_i - r_j \leqslant k$，k 为整数。在附录A中给出的最短路径算法中的任何一个都可以用来确定解是否存在，如果解的确存在的话，便可以求出这个解。利用下面的步骤可以完成这一工作。

(1) 画一个约束图

1) 对 N 个变量中的每一个变量 r_i，$i = 1, 2, \cdots, N$，画一个节点

2) 画节点 $N+1$

3) 对每个不等式 $r_i - r_j \leqslant k$，画一条从节点 j 到节点 i 的边 $j \to i$，长度为 k。

95

4) 对每个节点 i，$i = 1, 2, \cdots, N$，画一条从节点 $N+1$ 到节点 i 的边 $N+1 \to i$，长度为0。

(2) 利用最短路径算法求解

1) 当且仅当约束图不包含负环路时，不等式系统才有解。

2) 如果解存在，一个解为从节点 $N+1$ 到节点 i 的最小长度 r_i。

例4.3.1 本例中，我们说明如何利用最短路径算法求解一个不等式数目为 $M = 5$、变量数目为 $N = 4$ 的不等式系统：

$$\begin{aligned} r_1 - r_2 &\leqslant 0 \\ r_3 - r_1 &\leqslant 5 \\ r_4 - r_1 &\leqslant 4 \\ r_4 - r_3 &\leqslant -1 \\ r_3 - r_2 &\leqslant 2 \end{aligned}$$

第一步是画出约束图，如图4-3所示。

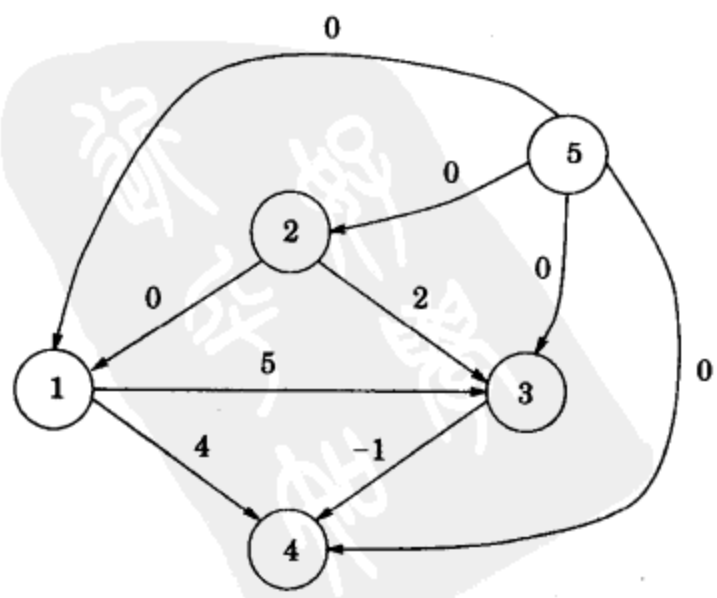


图4-3 例4.3.1的约束图

利用Bellman－Ford算法(附录A中A-2节中描述)，其中原点 U 是节点5，我们发现，没有负环路，因而可以通过考察 $r^{(4)}(V)$ 找到解。由 $r^{(4)}(1)=0$，$r^{(4)}(2)=0$，$r^{(4)}(3)=0$，$r^{(4)}(4)=-1$ 和 $r^{(4)}(5)=0$，该不等式系统的一个解可以确定为 $r_1=0$，$r_2=0$，$r_3=0$ 和 $r_4=-1$。 96

利用Floyd－Warshall算法(在附录A的A-3节中描述)，我们发现没有操作环路，因此求解可以通过考察矩阵

$$\mathbf{R}^{(6)}=\begin{bmatrix} \infty & \infty & 5 & 4 & \infty \\ 0 & \infty & 2 & 1 & \infty \\ \infty & \infty & \infty & -1 & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & 0 & -1 & \infty \end{bmatrix}$$

而得到。这里矩阵 $\mathbf{R}^{(6)}$ 中的元素 U、V 是 $r^{(6)}(U, V)$。$\mathbf{R}^{(6)}$ 最下行给出 $r^{(6)}(5, V)$，其中 $V=1, 2, 3, 4, 5$。不等式系统的解由 $r^{(6)}(5, V)$，$V=1, 2, 3, 4$ 给出，为 $r_1=0$，$r_2=0$，$r_3=0$，$r_4=-1$。 ■

当求解不等式系统时，可能有多个不等式左端相同的情况，这会造成在步骤1(3)中边的并行。例如，两个不等式 $r(1)-r(2)\leqslant 9$ 和 $r(1)-r(2)\leqslant 7$，将导致由节点2到节点1的两个边的权重分别为9和7。当这种情况发生时，应当选择这些不等式中的最大限制，而避免去画步骤1(3)中的那些并行的边。例如两个不等式 $r(1)-r(2)\leqslant 9$ 和 $r(1)-r(2)\leqslant 7$可以简单地用 $r(1)-r(2)\leqslant 7$来代表，因为它是两个不等式中的最大限制。所以步骤1(3)只画出了一条由节点2到节点1、权重为7的边。

4.4 重定时技术

本节考虑重定时所使用的一些技术。首先考虑称之为割集重定时和流水线的两种特殊情况，然后再考虑两个用于重定时的算法，一个使时钟周期最小化，另一个使寄存器数目最小化，二者都是实现电路所需要的。

4.4.1 割集重定时和流水线

割集重定时是一种有用的技术，它是重定时的一种特殊情况。割集是一组边，可以把它们从图中移走而产生两个不相连接的子图。割集重定时只影响割集中边的权重。若两个不相连的子图标为 G_1 和 G_2，则割集重定时包含两个内容，一个是从 G_1 到 G_2 的每条边都增加 k 个延时，另一个是从 G_2 到 G_1 的每条边都减去 k 个延时。例如，图4-4a中虚线所示的一个割集，其3条边是2→1，3→2和1→4。去掉该割集的这3条边后看到两个子图 G_1 和 G_2，如图4-4b所示。对于 $k=1$，割集重定时的结果如图4-4c。从 G_1 到 G_2 的边是3→2和1→4，每条边都加1个延时。从 G_2 到 G_1 的边是2→1，从这条边减1个延时。 97

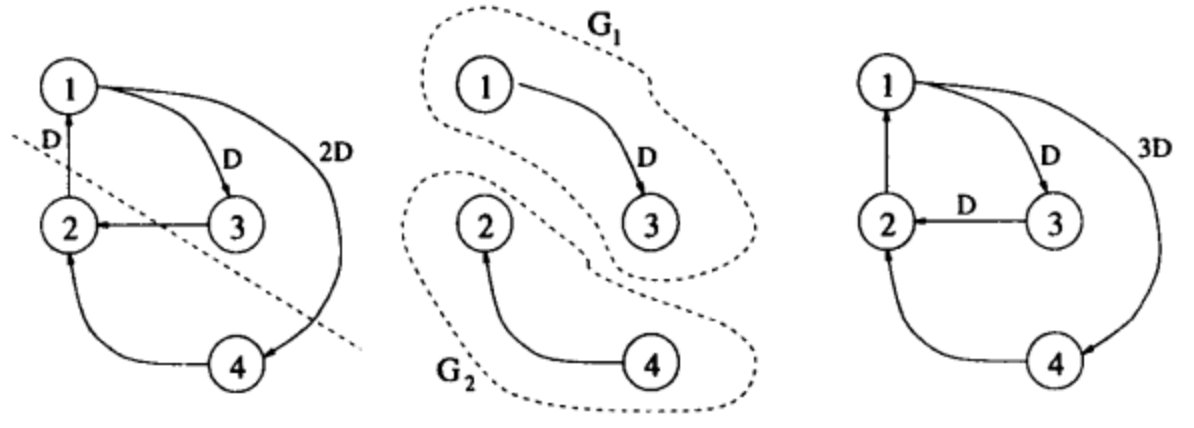
割集重定时是重定时的一种特殊情况，这种情况下子图 G_1 中每个节点重定时的值都为 j，而子图 G_2 中每个节点重定时值为 $j+k$。根据性质4.2.4，j 值是不重要的。在图4-4的例子中，由值 $j=0$ 和 $k=1$ 产生 $r(1)=0$，$r(2)=1$，$r(3)=0$ 和 $r(4)=1$，这样，就将图4-4a的DFG映射为图4-4c的DFG。任何 j 值都会产生与此相同的重定时图。

对于重定时图的可行性来说，$w_r(e)\geqslant 0$ 对于 G_r 中的所有边 e 都是必须保持成立的。令 $e_{1,2}$ 代表从 G_1 到 G_2 的一条边，而 $e_{2,1}$ 代表从 G_2 到 G_1 的一条边。因为割集重定时给从 G_1

到 G_2 的每条边都增加 k 个延时，$w_r(e_{1,2}) \geqslant 0 \Rightarrow w(e_{1,2}) + k \geqslant 0$ 必须保持成立。类似地，因为要从 G_2 到 G_1 的每条边 $e_{2,1}$ 都要减去 k 个延时，$w_r(e_{2,1}) \geqslant 0 \Rightarrow w(e_{2,1}) - k \geqslant 0$ 必须成立。把这两个不等式结合起来，并考虑割集的所有边，得到

$$-\min_{G_1 \xrightarrow{e} G_2} \{w(e)\} \leqslant k \leqslant \min_{G_2 \xrightarrow{e} G_1} \{w(e)\}$$

作为割集重定时给出可行解对 k 的要求。在图4-4中，$\min_{G_1 \xrightarrow{e} G_2}\{w(e)\} = \min\{0, 2\} = 0$ 和 $\min_{G_2 \xrightarrow{e} G_1}\{w(e)\} = 1$，则 $0 \leqslant k \leqslant 1$ 必须成立。因为 $k = 0$ 不产生任何效果，所以 $k = 1$是能产生可行的重定时图的惟一 k 值，$k = 1$的重定时图如图4-4c所示。

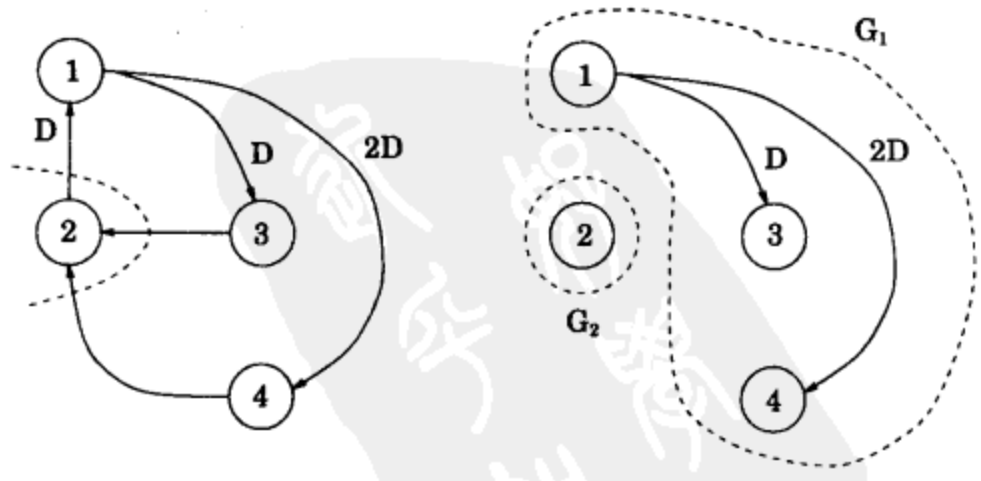


a) 未重定时的DFG，虚线表示它的一个割集　b) 去掉该割集的边形成的两个子图 G_1 和 G_2　c) 利用 $k = 1$ 的割集重定时找到的重定时图

图 4-4

有趣的是看看如下特殊情况会发生什么：选择一个割集，使子图 G_2 只有一个节点，而子图 G_1 包括其余节点并减去与 G_2 节点连接的那些边。这种割集的一个例子如图4-5a，其两个子图如图4-5b所示。子图 G_2 包含一个节点2，而子图 G_1 包含了原图的其余部分，并除去与节点2相连接的那3条边。这种情况的割集重定时采用给指向节点2的边加上1个延时，而从节点2指向其他节点的边减去1个延时，结果如图4-5c所示。因此，这种割集重定时的特殊情况包括：选择一个节点作为一个割集、从该节点的每个出行边上减1个延时而给该节点的每个入行边上加1个延时。

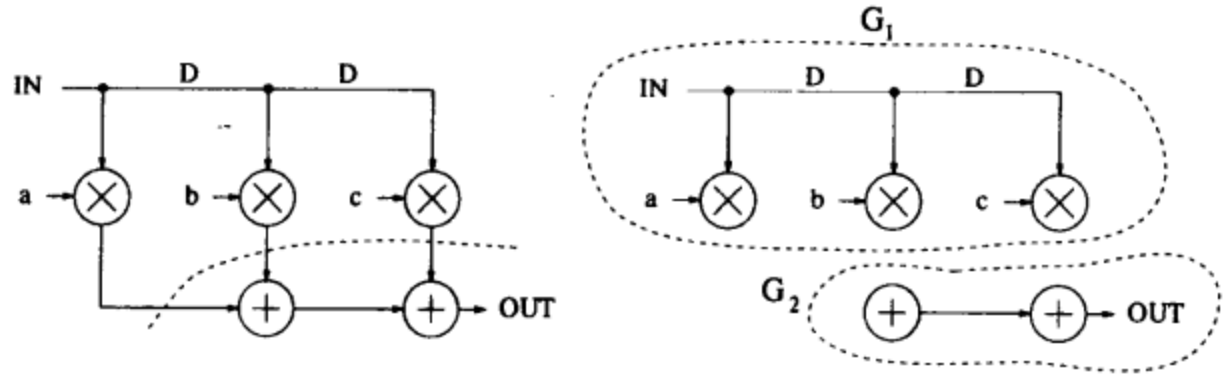
98



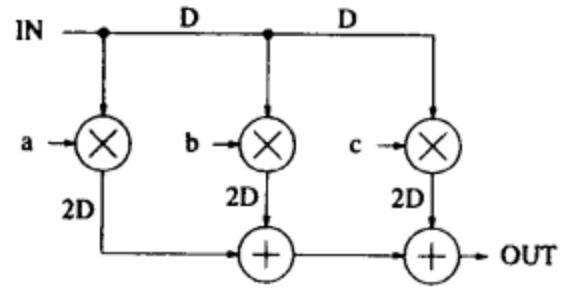
a) 未重定时的DFG，虚线示出它的一个割集　b) 去掉该割集的边形成的两个子图 G_1 和 G_2

图 4-5

流水线也是割集重定时的一个特殊情况，此处没有从子图 G_2 到子图 G_1 的边，即流水线应用到了一个无环路图的情况。这些割集称为前馈割集（forward cutset），前馈割集如图4-6a所示，该图分为两个子图，如图4-6b所示。割集中所有3条边都从 G_1 指向 G_2，并且对它进行 $k=2$ 的割集重定时，使割集中每条边增加2个延时，产生如图4-6c所示的重定时图(或在此种情况下也称为流水线图)。这表明了流水线可以看成是重定时的一个特殊情况。



a) 未重定时的DFG，虚线示出它的一个割集　　b) 去掉该割集的边形成的两个子图 G_1 和 G_2



c) 利用 $k=2$的割集重定时得到的图

图 4-6

割集重定时常常与减速（slow-down）结合使用，步骤是首先用 N 个延时取代DFG中的每个延时，以产生DFG的 N倍减速（N-slow）版本；然后对 N 倍减速的DFG进行割集重定时。注意在一个 N 倍减速系统中，$N-1$ 个空操作(或0样点)必须在每个有用信号样点后交替插入，以保持算法的正确功能。例如，考虑图4-7a的100阶格型滤波器，其关键路径为101个加法器和两个乘法器。假定加法和乘法时间分别为1 u.t.和2 u.t.，则最小抽样周期是(101)(1)+(2)(2)=105 u.t.。该电路的2倍减速(2-slow)版本示于图4-7b，割集重定时可以用来得到图4-7c的电路。重定时电路的关键路径有两个加法器和两个乘法器，计算时间为(2)(1)+(2)(2)=6 u.t.，因为这个电路是2倍减速的，则最小抽样周期是2×6 = 12 u.t.。在本例中，减慢与割集重定时把采样周期从105 u.t. 减少到12 u.t.。

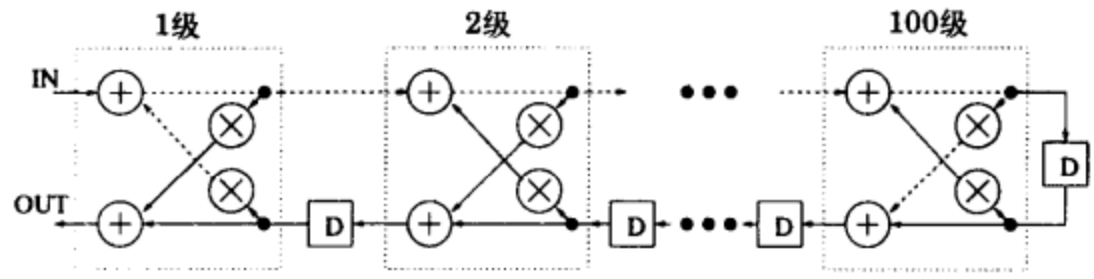
概括地说，割集重定时是重定时的一个特殊情况，流水线是割集重定时的一个特殊情况。割集重定时和流水线都是图形化技术，可以用来以简单的方式完成复杂的重定时操作。

4.4.2 时钟周期最小化的重定时

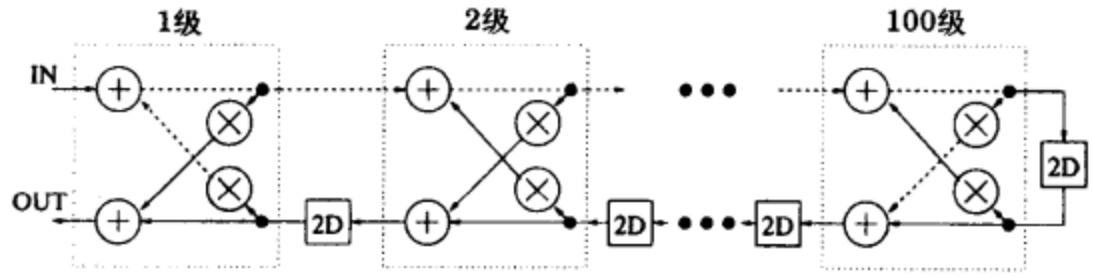
本节提出一种重定时算法，用来使同步电路的时钟周期最小化。对于电路 G，最小可行的时钟周期是关键路径的计算时间，关键路径是在所有无延时元件的路径中计算时间最长的路径。从数学上看，最小可用时钟周期 $\Phi(G)$ 定义为

99 ~ 100

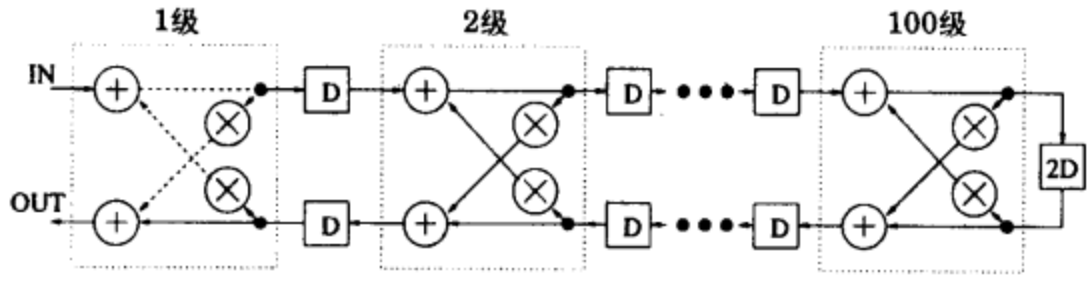
$$\Phi(G) = \max\{t(p) : w(p) = 0\}$$



a) 最小采样周期为105 u.t.的100级格型滤波器



b) 电路的2倍减速版本



c) 关键路径为6 u.t.的2倍减速电路的重定时版本，最小采样周期为12 u.t

图4-7 本图三个滤波器中每一个的关键路径都用虚线表示

（假设加法和乘法时间分别为1 u.t.和2 u.t.）

本节提出的算法找到一个重定时的解 r_0，使 $\Phi(G_{r0}) < \Phi(G_r)$，$\Phi(G_r)$ 为任何一个其他重定时的解 r。

在这个算法中用到了 $W(U, V)$ 和 $D(U, V)$ 这两个量，$W(U, V)$ 是从节点 U 到节点 V 的任一路径上寄存器的最少数目，而 $D(U, V)$ 是从 U 到 V 以 $W(U, V)$ 为权重的、所有路径中最长的计算时间。用公式表示如下：

$$\begin{aligned} W(U,V) &= \min\{w(p) : U \overset{p}{\rightsquigarrow} V\} \\ D(U,V) &= \max\{t(p) : U \overset{p}{\rightsquigarrow} V \text{ and } w(p) = W(U,V)\} \end{aligned}$$

下面的算法可以用来计算 $W(U, V)$ 和 $D(U, V)$。

(1) 令 $M = t_{max}n$，此处 t_{max} 是 G 中节点的最长计算时间，n 是 G 中节点数。

(2) 构成一个新图 G'，它与 G 相同，只是对所有的边 $U \xrightarrow{e} V$，其权重用 $w'(e) = M\,w(e) - t(U)$ 代替。

101

(3) 求解 G' 上所有 U、V 对（pairs）的最短路径问题。令 S'_{UV} 为从 U 到 V 的最短路径。

(4) 若 $U \neq V$，则 $W(U, V) = \left\lceil \dfrac{S'_{UV}}{M} \right\rceil$，$D(U, V) = MW(U, V) - S'_{UV} + t(V)$。

若 $U = V$，则 $W(U, V) = 0$，$D(U, V) = t(U)$。$\lceil x \rceil$ 的值是 x 的向上取整，为大于或等于 x 的最小整数。

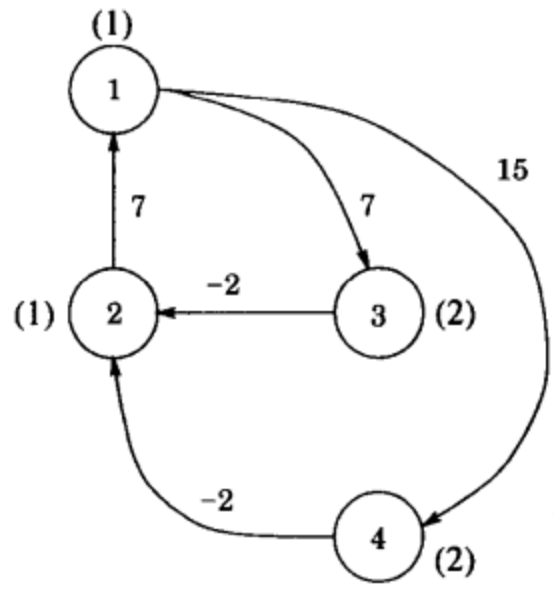


图4-8 对于图4-2a DFG构成的图G'，用来计算 $W(U, V)$ 和 $D(U, V)$

为了说明这个算法，计算图4-2a DFG的 $W(U, V)$ 和 $D(U, V)$ 值。第一步，$t_{max} = 2$，$n = 4$，所以 $M = 8$。新图 G'（在第二步找到的）如图4-8所示。对于 G' 所有的最短路径问题的解可以用Floyol-Warshall算法找到，该算法在附录A第A-3节中叙述。这个解（在第三步找到的）和 $W(U, V)$、$D(U, V)$ 的值（在最后一步找到的）示于表4-1。

表4-1 对于图4-2a的DFG找到的 S'_{UV}、W(U, V) 和 D(U, V) 的值

S'_{UV}	1	2	3	4
1	12	5	7	15
2	7	12	14	22
3	5	-2	12	20
4	5	-2	12	20

$W(U, V)$	1	2	3	4
1	0	1	1	2
2	1	0	2	3
3	1	0	0	3
4	1	0	2	0

$D(U, V)$	1	2	3	4
1	1	4	3	3
2	2	1	4	4
3	4	3	2	6
4	4	3	6	2

$W(U, V)$ 和 $D(U, V)$ 的值用于确定是否存在一个能够达到所期望时钟周期的重定时解。给定一个期望的时钟周期 c，当下列约束保持成立时：

(1)（可行性约束）对 G 的每条边 $U \xrightarrow{e} V$ 来说，都有 $r(U) - r(V) \leq w(e)$。

(2)（关键路径约束）对 G 中所有的顶点 U、V 来说，$r(U) - r(V) \leq W(U, V) - 1$，以满足 $D(U, V) > c$。

存在一个可行的重定时解 r。可行性约束强制重定时图中每条边上的延时数必须是非负的，而关键路径约束强制 $\Phi(G) \leq c$。若 $D(U, V) > c$，则对关键路径来说，必须保持 $W(U, V) + r(V) - r(U) \geq 1$，以使计算时间小于或等于 c，关键路径约束就是由这一点引出来的。对于图

4-2a的DFG来说，若 c 选取为3，对每个边 $U \xrightarrow{e} V$ 的不等式 $r(U)-r(V) \leqslant w(e)$ 为

$$\begin{aligned} r(1)-r(3) &\leqslant 1 \\ r(1)-r(4) &\leqslant 2 \\ r(2)-r(1) &\leqslant 1 \\ r(3)-r(2) &\leqslant 0 \\ r(4)-r(2) &\leqslant 0 \end{aligned} \tag{4-2}$$

而对 G 中所有的顶点 U、V 来说，根据从表4-1查到满足 $D(U, V) > 3$的数值，得到不等式 $r(U) - r(V) \leqslant W(U, V)-1$ 如下：

$$\begin{aligned} r(1)-r(2) &\leqslant 0 \\ r(2)-r(3) &\leqslant 1 \\ r(2)-r(4) &\leqslant 2 \\ r(3)-r(1) &\leqslant 0 \\ r(3)-r(4) &\leqslant 2 \\ r(4)-r(1) &\leqslant 0 \\ r(4)-r(3) &\leqslant 1 \end{aligned} \tag{4-3}$$

102~103

如果公式（4-2）和公式（4-3）有解，那么这个解就是一个可行的重定时解，使电路能够在时钟周期 $c = 3$ 下工作。对解的检查可以通过构造一个约束图并按第4-3节描述的方法求解单源(single-source)最短路径问题来完成。公式4-2和4-3的12个不等式的约束图示在图4-9中。利用Bellman-Ford算法或Floyd-Warshall算法可以确定，图4-9不包含负值环，因此对 $c = 3$的重定时解就是原点在节点5的单源最短路径问题的解，即 $r(1) = r(2) = r(3) = r(4) = 0$。这个结果是直观的，因为图4-2a的DFG已有 $\Phi(G) = 3$，所以不需要进行重定时就能达到时钟周期为3 u.t.的目标。

公式（4-2）和（4-3）的12个不等式有解这一事实表明，对于图4-2a电路的最小时钟周期小于或等于3。前面已经表明了，对于某些 U、V 对来说，DFG的最小时钟周期是 $D(U, V)$ [1]。为了找到可行的最小时钟周期，我们可以试探求解 c 等于每个 $D(U, V)$ 特定值的重定时方程，进而找出一个解的最小 c 值。由表4-1列出的 $D(U, V)$ 值可以看到，$D(U, V)$ 的特定值是1、2、3、4、6。$c = 1$ 的时钟周期是不可行的，因为节点3和节点4的计算时间大于1 u.t.。而 $c = 3$ 的时钟周期是可行的，因为图4-9的约束图有解，所以为了确定图4-2a的DFG最小的可行时钟周期，仅有 $c = 2$ 的值是必须检查的。

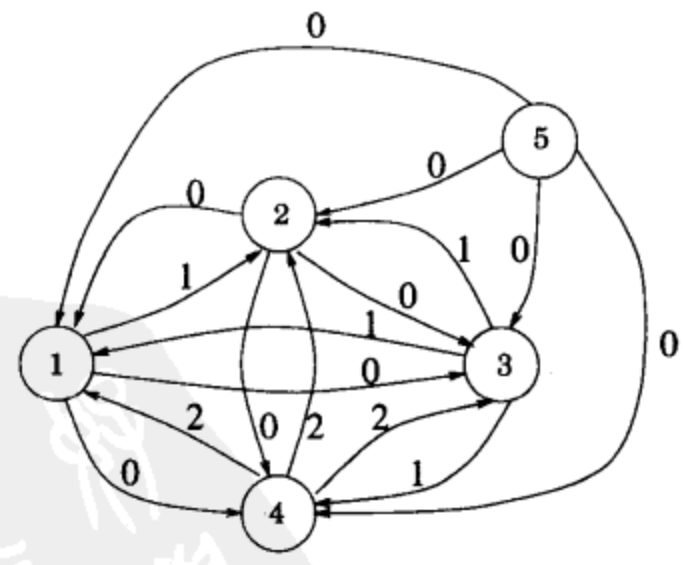


图4-9 公式（4-2）和公式（4-3）中不等式的约束图

对于 $c = 2$，每条边 $U \xrightarrow{e} V$ 的不等式 $r(U)-r(V) \leqslant w(e)$ 由公式（4-2）给出（这些不等式是DFG的函数，而不是 c 的函数）；对 G 中所有顶点 U、V 来说，满足 $D(U, V) > 2$ 的不等式 $r(U) - r(V) \leqslant W(U, V) -1$是

104

$$
\begin{aligned}
r(1)-r(2) &\leqslant 0\\
r(1)-r(3) &\leqslant 0\\
r(1)-r(4) &\leqslant 1\\
r(2)-r(3) &\leqslant 1\\
r(2)-r(4) &\leqslant 2\\
r(3)-r(1) &\leqslant 0\\
r(3)-r(2) &\leqslant -1\\
r(3)-r(4) &\leqslant 2\\
r(4)-r(1) &\leqslant 0\\
r(4)-r(2) &\leqslant -1\\
r(4)-r(3) &\leqslant 1
\end{aligned}
\tag{4-4}
$$

时钟周期 $c=2$ 的、可行的重定时解的方程由公式（4-2）和公式（4-4）给出。把公式（4-2）和公式（4-4）中左端相同的那些式子结合起来，可得到如下一组方程:

$$
\begin{aligned}
r(1)-r(3) &\leqslant 0\\
r(1)-r(4) &\leqslant 1\\
r(2)-r(1) &\leqslant 1\\
r(3)-r(2) &\leqslant -1\\
r(4)-r(2) &\leqslant -1\\
r(1)-r(2) &\leqslant 0\\
r(2)-r(3) &\leqslant 1\\
r(2)-r(4) &\leqslant 2\\
r(3)-r(1) &\leqslant 0\\
r(3)-r(4) &\leqslant 2\\
r(4)-r(1) &\leqslant 0\\
r(4)-r(3) &\leqslant 1
\end{aligned}
\tag{4-5}
$$

公式（4-5）的12个不等式的约束图示于图4-10。利用Bellman-Ford算法或Floyd-Warshall算法可以确定该图不包含负值环路，因此 $c=2$ 的重定时解就是以节点5为原点的单源最短路径问题的解，为 $r(1)=-1$，$r(2)=0$，$r(3)=-1$，$r(4)=-1$。这个解的时钟周期 $\Phi(G_r)=2$ 是图4-2a的重定时版本，如图4-2b所示。

寻找最小时钟周期重定时解的一般方法是，首先计算 $W(U, V)$ 和 $D(U, V)$，然后对 $D(U, V)$ 的值排序，再对这些值做二进制搜索（binary search），找出具有时钟周期的重定时解。计算 $W(U, V)$ 和 $D(U, V)$ 的值可以利用Floyd-Warshall算法在 $O(n^3)$ 时间内完成。$D(U, V)$ 不同值的最大数目是 n^2 个（若 $D(U, V)$ 所有值都不相同时），所以二进制搜索用掉 $O(\log n)$步。利用Floyd-Warshall算法或Bellman-Ford算法检查这些值是否大于或等于最小时钟周期用掉 $O(n^3)$ 时间。因此，找到最小时钟周期的重定时电路所需要的时间复杂性是 $O(n^3\log n)$。注意，如果迭代边界已经知道的话，则仅需要考虑 $D(U, V)$ 中大于或等于迭代边界的那些值就够了。 105

4.4.3 使寄存器数最小化的重定时

本节提出一种寻找重定时解的算法，它使用的寄存器数目最少，而仍能满足时钟周期的约束条件。

如果一个节点有几条输出边载带着相同的信号，实现这些边所需要的寄存器数是这些边中寄存器数最多的那个边上的寄存器数目。这一点表明于图4-11中。在图4-11a中直截了当的实现方法使用了1+3+7=11个寄存器，而在图4-11b中比较聪明的实现方法则使用了max(1, 3, 7)=7个寄存器。利用这个概念，在重定时图中实现节点 V 的输出边所需要的寄存器数目是

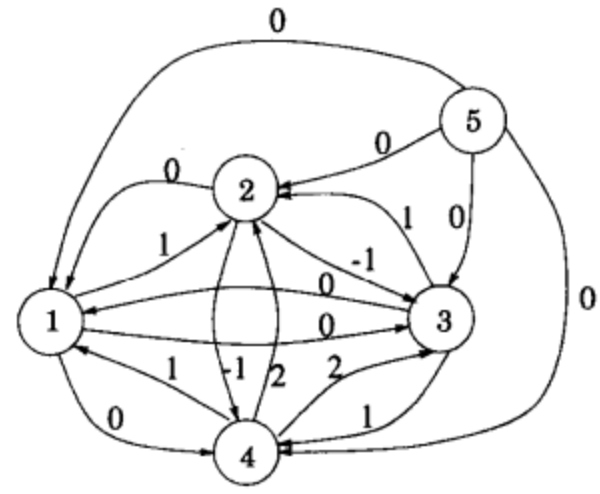
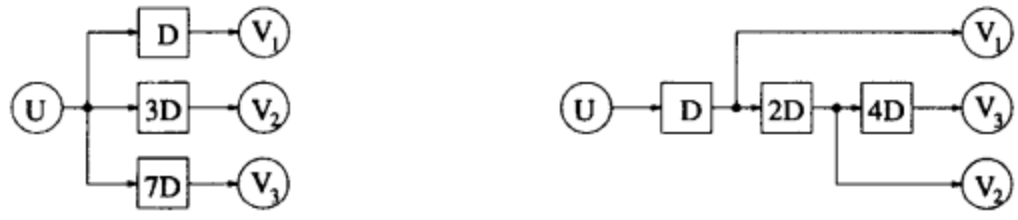


图4-10 公式（4-5）不等式的约束图

$$R_V = \max_{V \xrightarrow{e} ?}\{w_r(e)\}$$

在重定时后的电路中，总寄存器的代价是 $COST = \sum R_V$。



a) 扇出的实现，利用了1+3+7=11个寄存器

b) 扇出的实现，利用了max(1, 3, 7)=7个寄存器

图 4-11

在时钟周期不大于 c 的约束条件下，用来最小化寄存器数的公式表达是：最小化 $COST=\sum R_V$，满足条件：

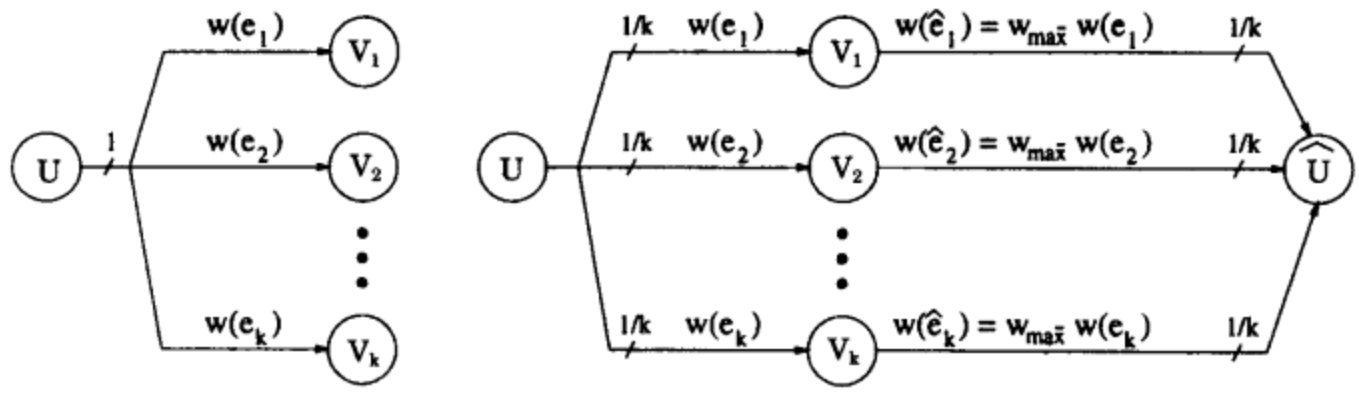
(1)(扇出约束)对所有的节点 V 和所有的边 $V \xrightarrow{e} ?$，$R_V \geqslant w_r(e)$。

(2)(可行性约束)对每条边 $U \xrightarrow{e} V$，$r(U) - r(V) \leqslant w(e)$。

(3)(时钟周期约束)对所有顶点 U、V，$r(U) - r(V) \leqslant W(U, V) - 1$，以满足 $D(U, V) > c$。

扇出约束用来简单地确认是否 $R_V = \max_{V \xrightarrow{e} ?}\{w_r(e)\}$。计算 $W(U, V)$ 和 $D(U, V)$ 的算法以及对可行性约束和时钟周期约束的描述已在第4.4.2节中叙述过了。

虽然这个重定时的公式表达确实可以在时钟周期约束下使寄存器数目最小化，但是它的形式并不是能够利用线性规划(Linear Programming, LP)技术可解的形式，因为某些解可能不是整数。为了得到能够利用LP技术可解形式的公式表达，用一种“小机构（gadget）”来表示带有多个输出边的节点[1]。这个“小机构”示于图4-12。图4-12a示出了一个带有 k 个输出边的扇出节点。图4-12b的“小机构”用来对扇出节点建模。对 k 条边中的每条边 e_i $(1 \leqslant i \leqslant k)$ 有一个相关联的权重 $w(e_i)$，$w(e_i)$ 可由DFG得知。节点 $\hat{U}$ 是一个计算时间为零 $[t(\hat{U}) = 0)]$ 的虚拟节点，边 $\hat{e}_i$ $(1 \leqslant i \leqslant k)$ 是引入的虚拟边，这样寄存器最小化问题的重定时就可以按线性规划问题来建模。边 $\hat{e}_i$ 的权重定义为 $w(\hat{e}_i) = w_{max} - w(e_i)$，此处 $w_{max} = \max_{1 \leqslant i \leqslant k} w(e_i)$。除了权重外，在这个模型中，每条边还有一个与其相关的幅宽 β。这个幅宽简言之就是一个数，以使图4-12b的“小机构”能够用来适当地模型化重定时DFG中的边 e_i $(1 \leqslant i \leqslant k)$ 所需要的存储器（会在稍后对其加以叙述）。e_i 和 $\hat{e}_i$ $(1 \leqslant i \leqslant k)$ 中每一条边的幅宽为 $\beta = 1/k$，如图4-12b所示。



a) 带有 k 个输出边的节点 U　　b) 用于给节点 U 建模的“小机构”

图 4-12

利用图4-12的扇出模型，重定时的公式表达是：最小化 $COST = \sum_e \beta(e)\, w_r(e)$，满足条件

(1)（可行性约束）对每条边 $U \xrightarrow{e} V$，$r(U) - r(V) \leqslant w(e)$。

(2)（时钟周期约束）对所有顶点 U、V，$r(U) - r(V) \leqslant W(U, V) - 1$，以满足 $D(U, V) > c$。

$COST$ 的表达式可以重写为：

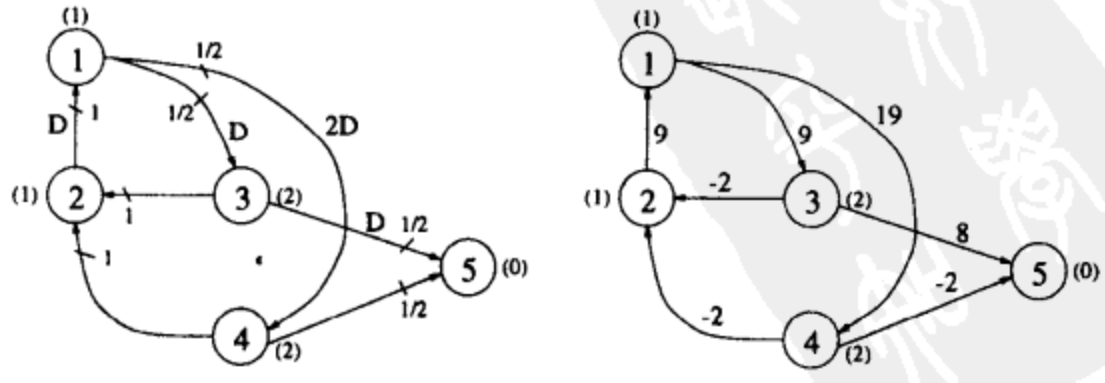
$$
\begin{aligned}
COST &= \sum_e \beta(e) w_r(e) \\
&= \sum_e \beta(e)(w(e) + r(V) - r(U)) \\
&= \sum_e \beta(e) w(e) + \sum_e \beta(e)(r(V) - r(U)) \\
&= K + \sum_e \beta(e)(r(V) - r(U)) \\
&= K + \sum_V r(V) \left(\sum_{? \xrightarrow{e} V} \beta(e) - \sum_{V \xrightarrow{e} ?} \beta(e) \right)
\end{aligned}
$$

因为 K 是常数，公式表达可以写成最小 $COST' = \Sigma_V r(V)(\Sigma_{? \xrightarrow{e} V} \beta(e) - \Sigma_{V \xrightarrow{e} ?} \beta(e))$，满足条件：

(1)（可行性约束）对每条边 $U \xrightarrow{e} V$，$r(U) - r(V) \leqslant w(e)$。

(2)（时钟周期约束）对所有顶点 U、V，为满足 $D(U, V) > c$，$r(U) - r(V) \leqslant W(U, V) - 1$。

为了说明这一点，对图4-2a的DFG进行重定时，以使寄存器数目最小化，而保持时钟周期 $\Phi(G_r) \leqslant 2$。第一步是利用节点1的扇出模型重画这个DFG，如图4-13a所示，因为节点1的输出边不只一条。第二步是计算4-13a DFG的 $W(U, V)$ 和 $D(U, V)$。利用在第4.4.2节中所叙述的步骤，得到 $M = 10$，G' 示于图4-13b，而 S'_{UV}、$W(U, V)$ 和 $D(U, V)$ 的值列于表4-2。 106 ~ 108



a) 利用图4-12中的扇出“小机构”重画图4-2a的DFG　　b) 用于计算 $W(U, V)$ 和 $D(U, V)$ 的 G' 图

图 4-13

表4-2 图4-13a中DFG的 S'_{UV}，W(U, V) 和 D(U, V) 的值

S'_{UV}	1	2	3	4	5
1	16	7	9	19	17
2	9	16	18	28	26
3	7	-2	16	26	8
4	7	-2	16	26	-2
5	-	-	-	-	-

$W(U,V)$	1	2	3	4	5
1	0	1	1	2	2
2	1	0	2	3	3
3	1	0	0	3	1
4	1	0	2	0	0
5	-	-	-	-	0

$D(U,V)$	1	2	3	4	5
1	1	4	3	3	3
2	2	1	4	4	4
3	4	3	2	6	2
4	4	3	6	2	2
5	-	-	-	-	0

该问题的线性规划公式表达为：最小化

$$
\begin{aligned}
COST' &= r(1)(1-1)+r(2)(2-1)+r(3)(1/2-3/2)+r(4)(1/2-3/2) \\
&\quad +r(5)(1-0) \\
&= r(2)-r(3)-r(4)+r(5)
\end{aligned}
$$

满足可行性约束

109

$$
\begin{aligned}
r(1)-r(3) &\leqslant 1 \\
r(1)-r(4) &\leqslant 2 \\
r(2)-r(1) &\leqslant 1 \\
r(3)-r(2) &\leqslant 0 \\
r(3)-r(5) &\leqslant 1 \\
r(4)-r(2) &\leqslant 0 \\
r(4)-r(5) &\leqslant 0
\end{aligned}
$$

和时钟周期约束

$$
\begin{aligned}
r(1)-r(2) &\leqslant 0 \\
r(1)-r(3) &\leqslant 0 \\
r(1)-r(4) &\leqslant 1 \\
r(1)-r(5) &\leqslant 1 \\
r(2)-r(3) &\leqslant 1 \\
r(2)-r(4) &\leqslant 2 \\
r(2)-r(5) &\leqslant 2 \\
r(3)-r(1) &\leqslant 0 \\
r(3)-r(2) &\leqslant -1
\end{aligned}
$$

$$
\begin{aligned}
r(3)-r(4) &\leqslant 2\\
r(4)-r(1) &\leqslant 0\\
r(4)-r(2) &\leqslant -1\\
r(4)-r(3) &\leqslant 1
\end{aligned}
$$

这个线性规划问题在形式上是最小成本流(minimum-cost flow)问题，其算法已存在（参见[3]），但可能相当复杂。为了求解这一问题，可以使用如GAMS[4]这类的线性规划软件包，利用GAMS，该问题的解是 $r(1)=1$，$r(2)=2$，$r(3)=1$，$r(4)=0$ 和 $r(5)=0$，其重定时后的图利用了节点1两条输出边之间适当的寄存器共享，如图4-14所示。

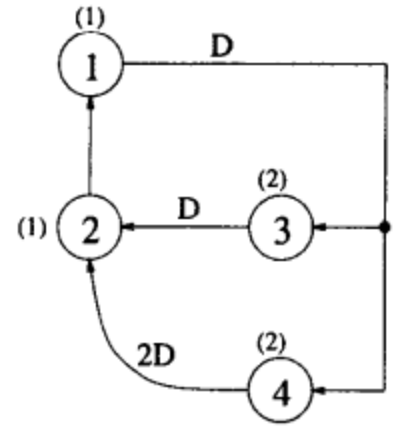


图4-14 在保持时钟周期为2 u.t.的条件下使寄存器数最小化，对图4-2a DFG重定时的结果

余下的问题是给出图4-12b中的扇出“小机构”来对多输出边节点所用存储器数量进行适当地建模。目标是使扇出“小机构”中 e_i 和 $\hat{e}_i$ 所有边的权重（乘以幅宽 β 因子）之和等于“小机构”中边 e_i 上权重的最大值。换言之，目标是给出

$$\sum_{i=1}^{k}(w_r(e_i)\beta(e_i)+w_r(\hat{e}_i)\beta(\hat{e}_i))=w_{r,max}$$

此处 $w_{r,max}=\max_{1\leqslant i\leqslant k} w_r(e_i)$。这一点可以用下面三个步骤来证明。

第一步：假定 $w_r(e_j)=w_{r,max}$，证明 $w_r(e_j)=w_{max}+r(\hat{U})-r(U)$。

第二步：证明

$$\sum_{i=1}^{k}(w_r(e_i)\beta(e_i)+w_r(\hat{e}_i)\beta(\hat{e}_i))=w_{max}+r(\hat{U})-r(U)$$

第三步：结合第一、二步，得到

$$\sum_{i=1}^{k}(w_r(e_i)\beta(e_i)+w_r(\hat{e}_i)\beta(\hat{e}_i))=w_{max}+r(\hat{U})-r(U)=w_r(e_j)=w_{r,max}$$

第三步是试探性的(假定第一、二步可以证明)，第一、二步则不是试探性的，下面描述这两步细节。

第一步的细节：假定 $w_r(e_j)=w_{r,max}$。在原始的DFG中，每条路径 $p_i=U\xrightarrow{e_i}V_i\xrightarrow{\hat{e}_i}\hat{U}$ 的权重是

$$
\begin{aligned}
w(p_i) &= w(e_i)+(w_{max}-w(e_i))\\
&= w_{max}
\end{aligned}
$$

在重定时的DFG中，每条路径 p_i 的权重是

$$
\begin{aligned}
w_r(p_i) &= w_r(e_i)+w_r(\hat{e}_i)\\
&= (w(e_i)+r(V_i)-r(U))+\left(w_{max}-w(e_i)+r(\hat{U})-r(V_i)\right)\\
&= w_{max}+r(\hat{U})-r(U)
\end{aligned}
$$

110 ~ 111

求解最小化 $COST$，使 $r(\hat{U})$ 尽量小，而仍然强制 $w_r(\hat{e}_i) \geq 0$，$1 \leq i \leq k$。因此，对于至少一条边 $\hat{e}_i$ 有 $w_r(\hat{e}_i)=0$。因为公式 $w_r(p_i)=w_r(e_i)+w_r(\hat{e}_i)=w_{max}+r(\hat{U})-r(U)$ 的值对于 $1 \leq i \leq k$ 是相同的，所以具有 $w_r(e_j)=w_{r,max}$ 的边 e_j 在路径 p_j 中，其 $w_r(\hat{e}_i)=0$。因此，$w_r(e_j)=w_r(p_j)=w_{max}+r(\hat{U})-r(U)$。

第二步的细节：在扇出模型中边的代价是：

$$
\begin{aligned}
COST &= \sum_{i=1}^{k}(w_r(e_i)\beta(e_i)+w_r(\hat{e}_i)\beta(\hat{e}_i)) \\
&= \frac{1}{k}\left(\sum_{i=1}^{k}(w_r(e_i)+w_r(\hat{e}_i))\right) \\
&= \frac{1}{k}(k)\left(w_{max}+r(\hat{U})-r(U)\right)
\end{aligned}
$$

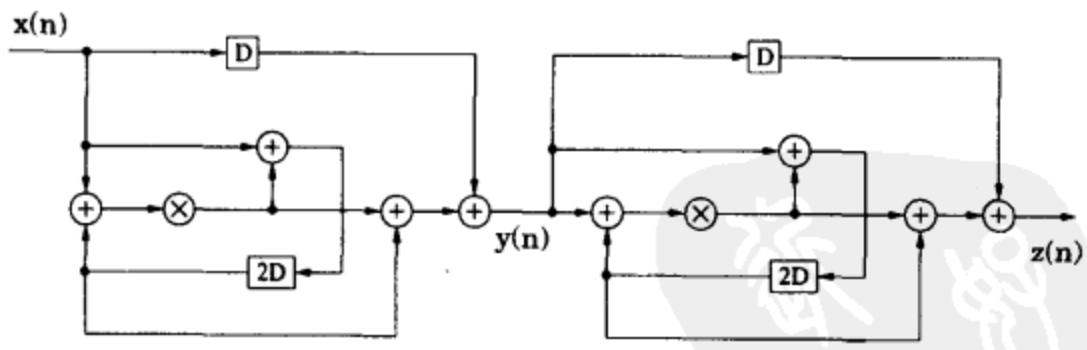
■

4.5 结论

同步系统可以利用重定时来缩小关键路径或时钟周期、减少存储或延时元件数目、或者降低功耗。最短路径算法可以用来得到重定时的解，如果这个解存在的话。所有可能的重定时解也可以通过利用[5]，[6]中所叙述的方法穷尽地（exhaustively）进行反复重定时来获得。重定时也可以用来作为折叠和计算舍入噪声的预处理步骤，折叠和计算舍入噪声分别在第6章和第11章中讨论。利用重定时来降低功耗的问题包含在第17.5.4节的内容中。多速率DFG的重定时在第6.5节中论及多速率折叠的内容时讨论。二维DFG的重定时已经在[7]，[8]中阐述过了。

4.6 习题

1. 考虑图4-15所示的数字滤波器。假定每个乘法操作需要20 nsec、每个加法操作需要8 nsec。



112

图4-15 习题1中的波数字滤波器结构

(a) 通过考察，计算这个滤波器的迭代周期边界。

(b) 关键路径是哪一条？

(c) 对这个滤波器进行手工排流水线和/或重定时。

2. 考虑图4-16所示的DFG，其中每个节点处的数字表示其执行时间。

(a) 该DFG的最大采样率是多少？

(b) 对于该DFG所描述的系统，采样周期的基本限制是什么？

(c) 对该DFG进行手工重定时，使时钟周期最小化。

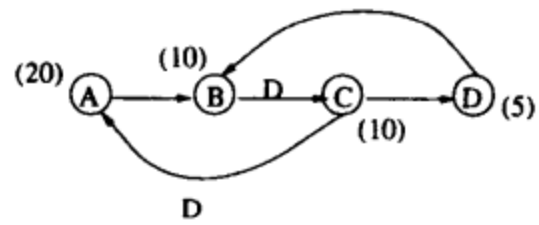


图4-16 习题2的数据流图

3. 考虑图4-17所示的4级流水线全带通8阶IIR数字滤波器的DFG。假定加法和乘法分别需要1 u.t.和2 u.t.。

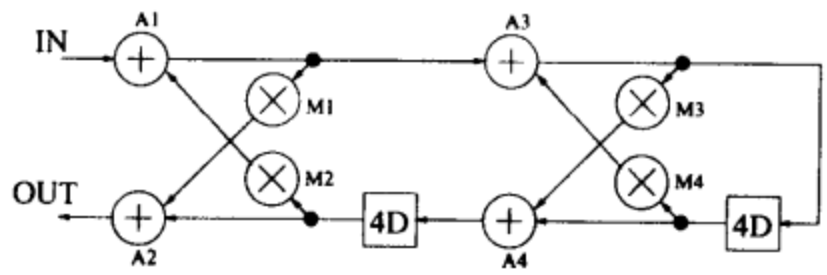


图4-17 4级流水线全带通8阶IIR数字滤波器

(a) 通过考察，计算迭代边界。

(b) 计算电路的关键路径时间。

(c) 通过考察对该系统进行手工操作排流水线和/或重定时，使关键路径达到2 u.t.。

4. 考虑图4-18所示的6阶正交滤波器结构。在该结构中所有的操作都是CORDIC(Coordinate Rotation Digital Computer，坐标旋转数字计算机)的旋转操作，这些旋转操作都是正交归一的。假定每个CORDIC旋转操作需要T nsec。

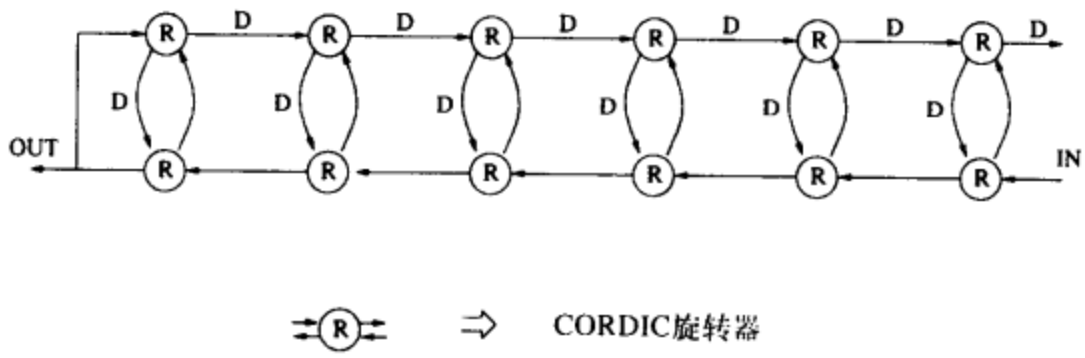


图4-18 习题4的6阶正交滤波器

该滤波器的迭代边界是多少？关键路径是哪条？对该滤波器进行手工排流水线和/或重定时，以便达到计算时间为$2T$ nsec的关键路径。明确地示出用于重定时的所有割集位置。

5. 图4-19示出的DFG描述了一个4阶IIR数字滤波器，它是由两节2阶滤波器级连来实现的。假定每个乘法需要2 u.t.而每个加法需要1 u.t.。

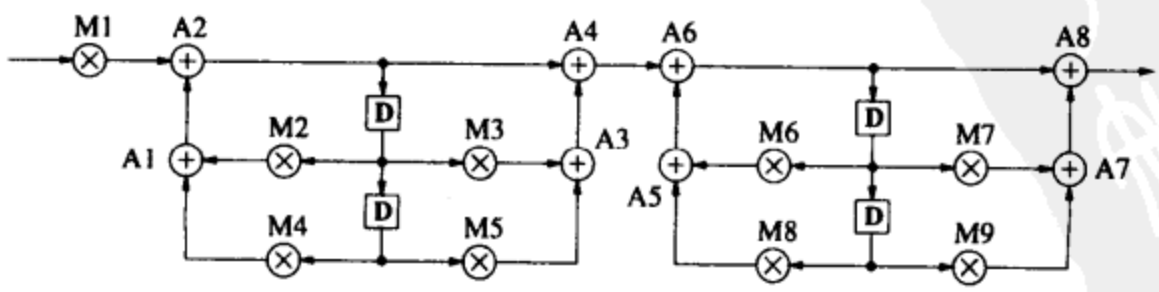


图4-19 习题5中4阶IIR滤波器的DFG

(a) 这个DFG的关键路径是哪条？这个DFG的迭代边界是多少？

(b) 对这个DFG进行手工重定时和排流水线，使时钟周期最小化。用流水线和重定时得到的最小时钟周期是多少？

6. 对于图4-20中的双二阶滤波器，假定加法和乘法分别需要1 u.t.和2 u.t.。有120个重定时的特解，对该滤波器重定时使时钟周期为4 u.t.。通过考察找出其中的10个解来。

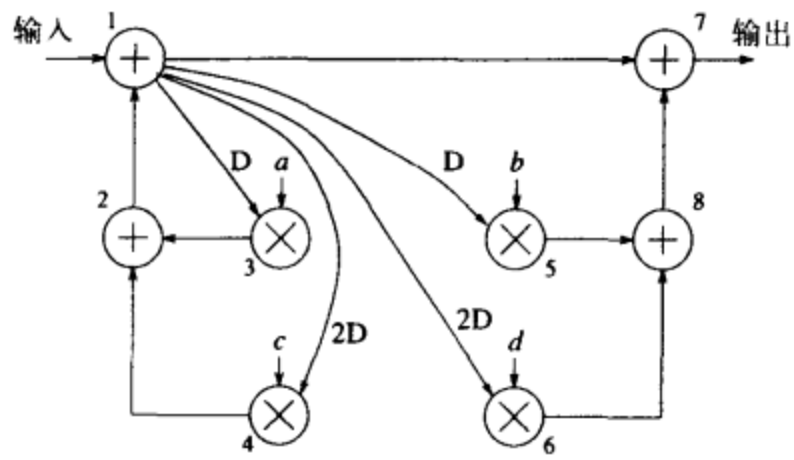


图4-20 双二阶滤波器，节点旁边的数字是习题7中所用的节点标号，表示执行时间

7. 对于图4-20中的DFG，找出重定时后的DFG，满足 $r(1)=0$，$r(2)=0$，$r(3)=-1$，$r(4)=-2$，$r(5)=-1$，$r(6)=0$，$r(7)=0$，$r(8)=0$。

8. 画一个约束图，并用它来确定下列不等式系统是否有解。如果有解的话，利用

(a) Bellman-Ford算法

(b) Floyd-Warshall算法

求出这个解来。

$$
\begin{aligned}
r_1 - r_2 &\leqslant 1 \\
r_3 - r_1 &\leqslant 3 \\
r_4 - r_1 &\leqslant 2 \\
r_4 - r_3 &\leqslant -1 \\
r_3 - r_2 &\leqslant 1 \\
r_5 - r_1 &\leqslant 2 \\
r_3 - r_5 &\leqslant -1 \\
r_4 - r_5 &\leqslant -2
\end{aligned}
$$

9. 画一个约束图，并用它来确定下列不等式系统是否有解。若有解的话，利用

(a) Bellman-Ford算法

(b) Floyd-Warshall算法

求出这个解来。

$$
\begin{aligned}
r_1 - r_2 &\leqslant 1 \\
r_3 - r_1 &\leqslant 3 \\
r_4 - r_1 &\leqslant 2 \\
r_4 - r_3 &\leqslant -1 \\
r_3 - r_2 &\leqslant 1 \\
r_5 - r_1 &\leqslant 2 \\
r_3 - r_5 &\leqslant -6 \\
r_4 - r_5 &\leqslant -2
\end{aligned}
$$

10. 对于图3-21（见第3章习题2）所示的IIR数字滤波器进行重定时/排流水线，以便把关键路径缩小为2 u.t.。 113～115

11. 本题讨论递归DFG的块间流水线问题。

(a) 重定时图4-21所示的系统，以便实现块间流水线，即块间通讯的每条边应当至少有一个延时元件。

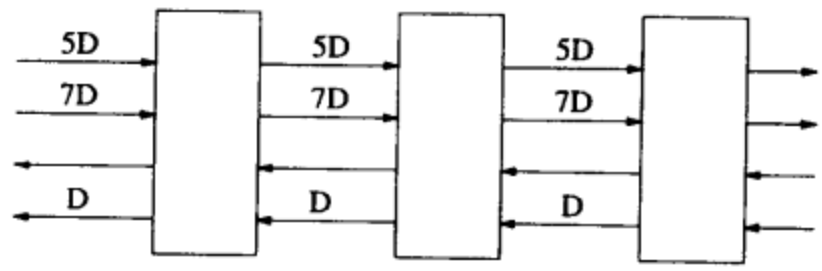


图4-21 习题11(a)中所用的递归DFG

(b) 为了对图4-22所示的系统得到块间流水线，使用一种适当地减慢的方法，然后再使用重定时。这个系统的硬件实现效率如何？

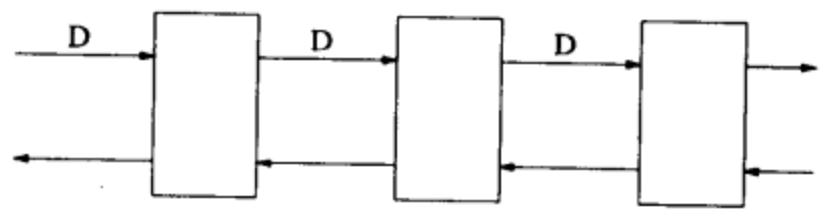


图4-22 习题11(b)中所用的递归DFG

12. 考虑图4-23中示出的N级归一化的格型滤波器。令$N=25$，即滤波器有25个模块，并假定加法和乘法分别需要1 u.t.和2 u.t.。

(a) 计算这个滤波器的关键路径和最小时钟周期。

(b) 利用2倍减速的变换，重定时这个滤波器，使得它在模块级构成流水线，即模块之间的每条线有至少一个延时。重定时后滤波器的时钟周期和采样周期各是多少？

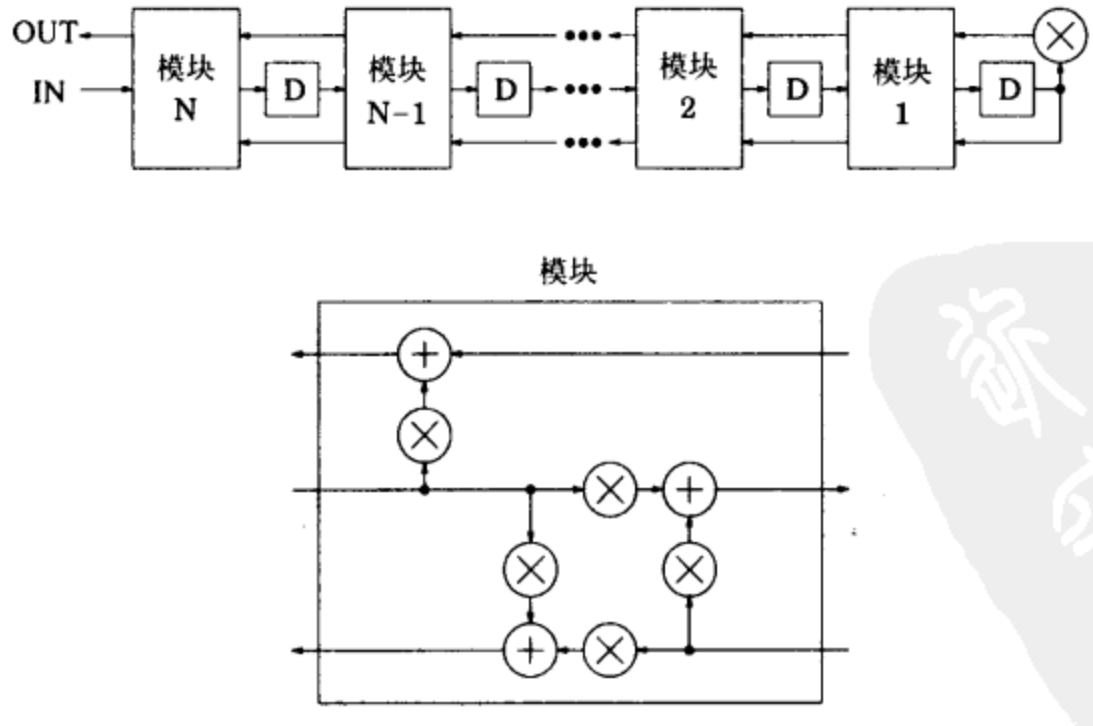


图4-23 N级归一化格型滤波器

13. 本题的目的是写一段程序，对一个DFG进行重定时，使时钟周期最小化。假定加法和

乘法分别需要1 u.t.和2 u.t.。把重定时的约束公式化，并利用你的程序来确定图4-24中直接形式的3阶IIR滤波器的最小时钟周期。

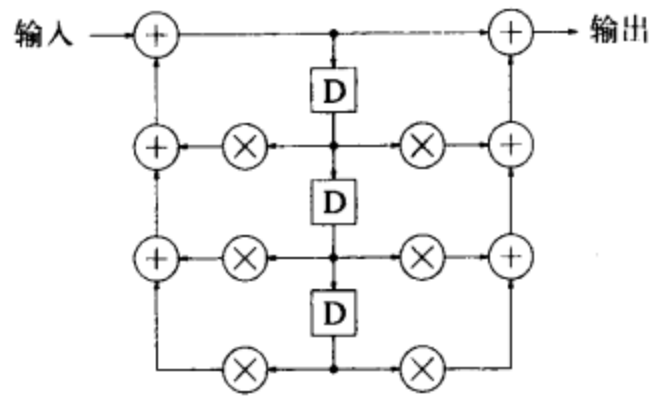


图4-24 直接形式的3阶IIR滤波器

14. 图4-25中的DFG表示一个5阶的波数字椭圆滤波器，据此重做习题13。
15. 图2-17中的DFG表示一个使用单乘法器的基本的格型IIR数字滤波器，据此重做习题13。
16. 把习题13中所用的重定时DFG方法系统化，用于使寄存器数目最小化，而达到时钟周期为4 u.t.的目标。假定加法和乘法分别需要1 u.t.和2 u.t.。
17. 把习题13中所用的重定时DFG方法系统化，用于使寄存器数目最小化，而达到时钟周期为2 u.t.的目标。假定加法和乘法分别需要1 u.t.和2 u.t.。

116～117

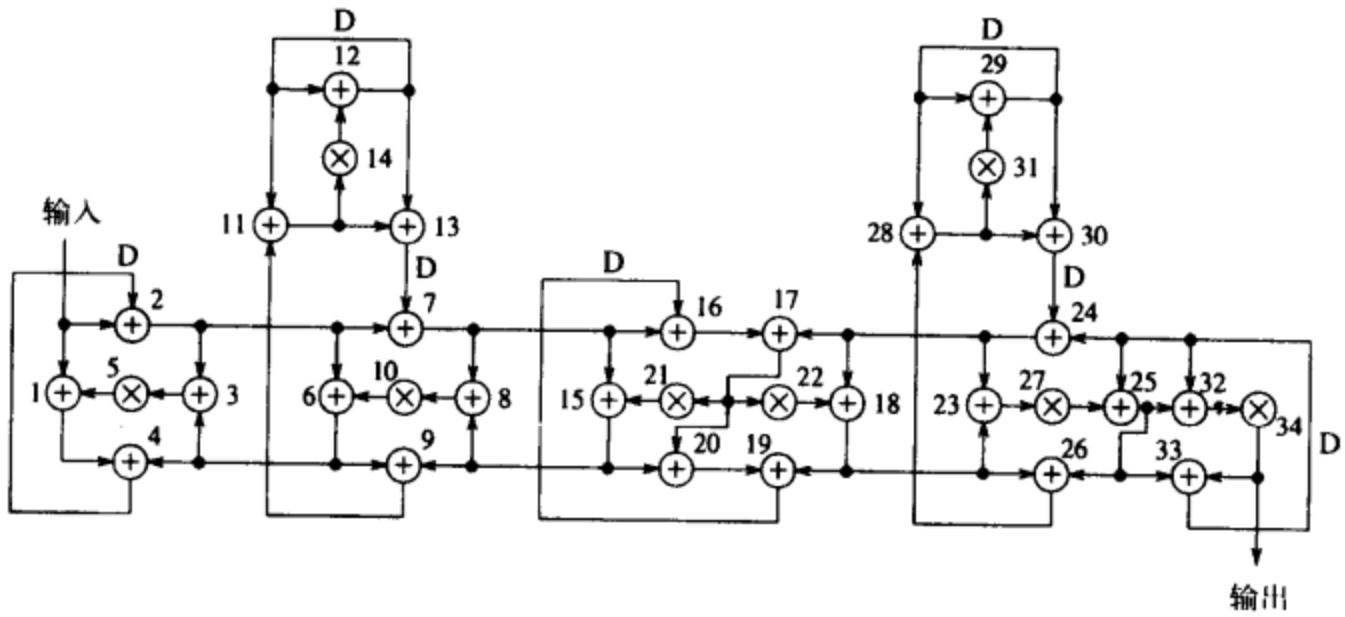


图4-25 一个5阶波数字椭圆滤波器

参考文献

1. C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," in *Third Caltech Conference on VLSI*, pp. 87–116, 1983.
2. J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. of IEEE International Conference on Computer Aided Design*, pp. 398–402, 1993.
3. E. L. Lawler, *Combinatorial Optimization: Networks and Matriods.* Holt, Rinehard and Winston, 1976.
4. A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A User's Guide, Release 2.25.* The Scientific Press, 1992.
5. S. Simon, E. Bernard, M. Sauer, and J. Nossek, "A new retiming algorithm for circuit design," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, (London, England), May 1994.

6. T. C. Denk and K. K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *IEEE Trans. on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 45, no. 7, pp. 821-838, July 1998.

7. T. C. Denk and K. K. Parhi, "Two-dimensional retiming", *IEEE Trans. on VLSI Systems*, vol. 7, 1999.

8. N. Passos and E. H.-M. Sha, "Full parallelism in uniform nested loops using multi-dimensional retiming," in *Proc. of International Conference on Parallel Processing*, 1994.

安徽大学藏

第5章 展　开

5.1 引言

展开是一种可以应用于DSP 程序的转换技术，它产生一个新的程序来描述原有程序的多次迭代。更具体地说，以展开因子J（unfolding factor J）展开一个DSP程序，就会产生一个以原程序连续迭代 J 次的新程序。展开也被称为环路展开，应用于编译理论中[1]。

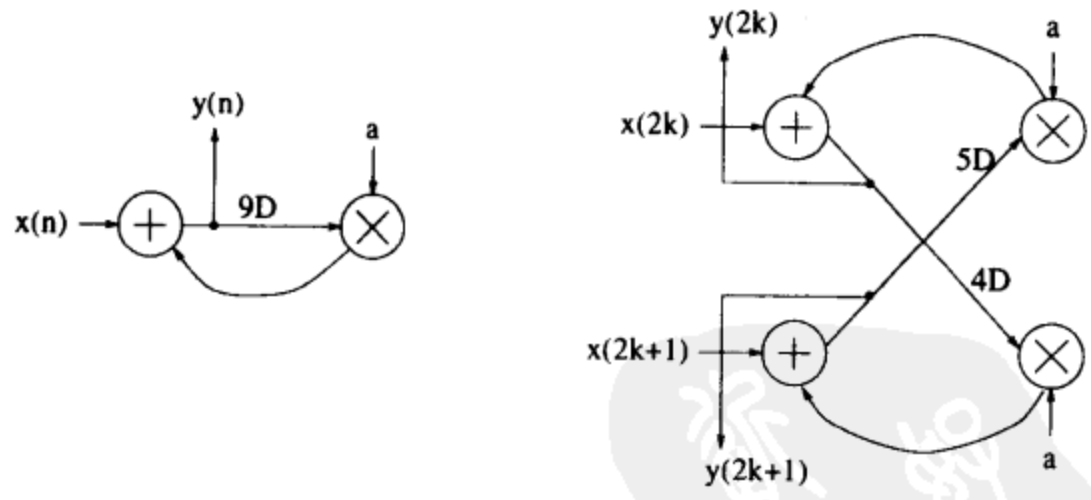
例如：考虑以下DSP程序：

$$y(n) = ay(n-9) + x(n) \tag{5-1}$$

其中 n=0 到∞，如图5-1a所示。将序号 n 用 $2k$ 代替，得到 $y(2k)=ay(2k-9)+x(2k)$，k=0到∞。类似地，用 $(2k+1)$ 代替序号 n，得到 $y(2k+1)=ay(2k-8)+x(2k+1)$，k=0到∞。至此，得到的两个方程

$$\begin{aligned} y(2k) &= ay(2k-9) + x(2k) \\ y(2k+1) &= ay(2k-8) + x(2k+1) \end{aligned} \tag{5-2}$$

和公式(5-1)描述的是同一程序。公式(5-2）的程序描述了公式(5-1)程序的两个连续迭代。例如：当 k=7 时，公式(5-2)为：$y(14)=ay(5)+x(14)$ 和 $y(15)=ay(6)+x(15)$。这些方程和公式(5-1)所描述的当 n=14 和 n=15 时的两个连续迭代是相同的。公式(5-2)中的程序描述了公式(5-1)中程序的一个二阶展开（2-unfolding）版本。



a) 描述 $y(n)=ay(n-9)+x(n)$, n=0到∞的DSP原始程序

b) 描述 $y(2k)=ay(2k-9)+x(2k)$ 和 $y(2k+1)=ay(2k-8)+x(2k+1)$, n=0到∞的二阶展开DSP程序

图 5-1

在展开系统中，每个延迟是 J 倍降速（J-slow）的，这就是说，如果输入对一个延迟单元的信号是 $x(kJ+m)$，则该延迟单元的输出是 $x((k-1)J+m)=x(kJ+m-J)$。公式(5-2)对应的2阶展开程序如图5-1b所示，运用 J 倍降速延迟的概念，图5-1b描述了所希望的运算 $y(2k)=ay(2(k-5)+1)+x(2k)$ 和 $y(2k+1)=ay(2(k-4)+0)+x(2k+1)$。

图5-1b所示的2阶展开程序是通过代入 $n=2k$ 和 $n=2k+1$ 到公式(5-1)中得到的。虽然这种方法可以得到正确的2阶展开程序，但是写出原始程序和它的 J 阶展开程序方程式，并进一步画出相应的展开数据流图（DFG）通常是比较繁琐的。特别地，对于较大的 J，情况尤其如此。本章中，我们描述了一种基于图形的技术，它能直接展开DFG和生成 J 阶展开程序的DFG[2]，而无需写出原程序和展开程序的表达式。

展开技术可应用于高速和低功耗VLSI的架构设计[3]~[5]。一个应用是展开程序以发现其中的并发性，以便在一个更小的迭代周期中完成程序，从而增加实现该架构的吞吐率。另一个应用是在字级或位级设计并行架构。在字级，这些架构可用于把字串行架构设计为字并行架构，从而增加该结构的吞吐率或减小功耗（见第3章）。在位级，展开技术可用于将位串行架构设计为位并行的数字串行架构，达到和字级展开同样的目的[6], [7]。

120

在第5.2节中给出了基于图形的算法，用于在无需写出原程序的 J 次连续迭代表达式的情况下展开一个DFG。展开技术的一些属性在第5.3节中讨论，第5.4节探讨了关键路径、展开阶数和重定时的关系。第5.5节讨论了展开技术的应用。本章中我们始终假设一个节点不能拆为两个具有更小运算时间的子节点。

5.2 一种展开算法

本节讨论了以 J 为展开因子展开DFG的算法。给出这个算法前，首先介绍一些符号和展开后的DFG的基本属性。

操作 $\lfloor x \rfloor$ 表示对 x 向下取整，即取小于或等于 x 的最大整数。例如，$\left\lfloor \frac{11}{4} \right\rfloor = 2$，$\left\lfloor \frac{25}{3} \right\rfloor = 8$。操作 $a\%b$ 表示 a 除以 b 的余数，其中 a 和 b 是整数。例如，11%4=3，25%3=1。

对原始DFG中的每一个节点 U，在 J 阶展开DFG中都有 J 个具有相同功能的节点 U。例如，对于图5-1a中的加法器，在图5-1b所示的2阶展开DFG中存在两个相同的加法器。类似地，图5-1b的两个乘法器与图5-1a中的乘法器相对应。

原始DFG的每一条边，在 J 阶展开DFG中都有 J 条相应的边。从图5-1中可以看到，原始DFG中从加法器到乘法器的一条边，导致了展开DFG中从加法器到乘法器的两条边。

请记住，J 阶展开程序的DFG总是包含了相当于原始DFG的 J 倍数量的节点和边，以下的展开算法可用于构建一个 J 阶展开DFG。

展开算法

(1) 对于原始DFG中的每一个节点 U，画 J 个节点 U_0，U_1，…，U_{J-1}。

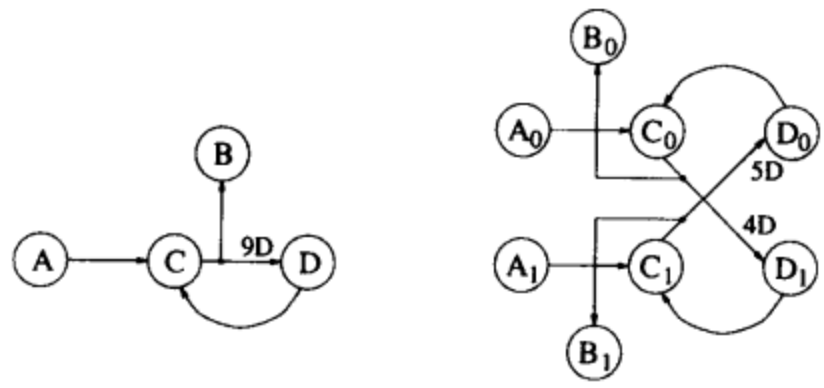
(2) 对原始DFG中每一条有 w 延迟的边 $U \rightarrow V$，画 J 条有 $\left\lfloor \frac{i+w}{J} \right\rfloor$ 延迟的 $U_i \rightarrow V_{(i+w)\%J}$ 边，$i=0$，1，…，$J-1$。

这种展开算法正确性的证明在本节末给出。

为了演示这种展开算法，构建了图5-1a中DSP程序的DFG，这个DFG以2为展开系数展开，得到了图5-1b中的DSP程序。表示图5-1a中DSP程序的DFG如图5-2a所示。在这个DFG中，节点 A 和 B 分别表示输入和输出，节点 C 和 D 分别表示相加和乘 a。为了以2为展开系数展开这个DFG，根据展开算法第一步，首先画出8个节点 A_i、B_i、C_i 和 D_i, $i=0$，1。

121

画出这些节点后，需要进行展开算法的第二步。对于没有延迟的边 $U \to V$，这一步简化为画出 J 个没有延迟的边 $U_i \to V_i$, $i=0, 1, \ldots, J-1$。例如，由图5-2a中没有延迟的边 $D \to C$ 得到图5-2b中的2阶展开DFG中的没有延迟的两条边 $D_0 \to C_0$ 和 $D_1 \to C_1$。对图5-2a中具有 $w=9$ 延迟的边 $C \to D$，画出具有 $\left\lfloor \frac{9+0}{2} \right\rfloor$ 延迟的边 $C_0 \to D_{(9+0)\%2}$ 和具有 $\left\lfloor \frac{9+1}{2} \right\rfloor$ 延迟的边 $C_1 \to D_{(9+1)\%2}$，它们分别对应于图5-2b中具有4个延迟的边 $C_0 \to D_1$ 和具有5个延迟的边 $C_1 \to D_0$。

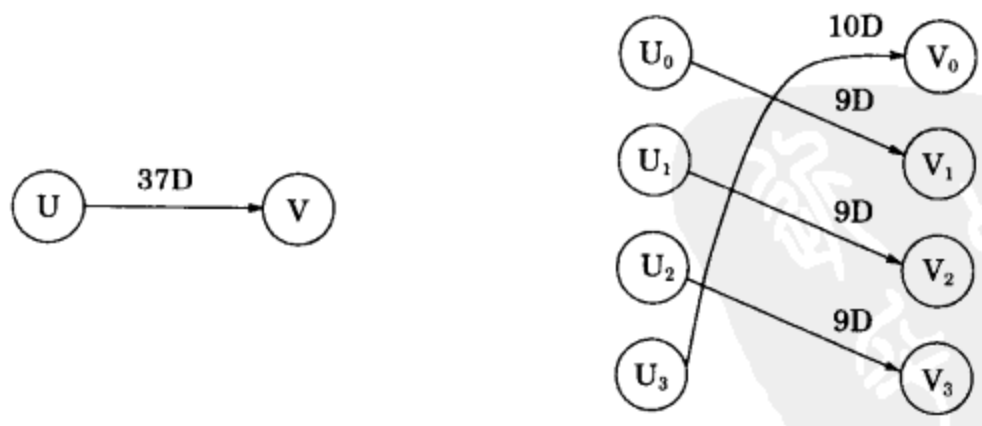


a) 与图5-1a所示DSP程序相对应的DFG　　b) 与图5-1b所示二阶展开DSP程序相对应的二阶展开DFG

图 5-2

由图5-2，二阶展开DFG中的节点 C_0 和 C_1 表示相加运算，因为原始DFG中的节点 C 表示相加（记住，图5-2a中的DFG表示图5-1a中的DSP程序）。类似地，二阶展开DFG中的节点 D_0 和 D_1 表示与常数 a 相乘，因为原始DFG中的节点 D 表示乘以常数 a。原始DFG中的节点 A 表示输入 $x(n)$。展开DFG中节点 A_i 的第 k 次迭代，执行在原始DFG中节点 A 的第 $(Jk+i)$ 次迭代，$i=0, 1, \ldots, J-1$，$k=0$ 到 ∞。因此，节点 A_0 对应于输入样值 $x(2k+0)$，而节点 A_1 对应于输入样值 $x(2k+1)$。类似地，节点 B_0 对应于输出样值 $y(2k+0)$，而节点 B_1 对应于输出样值 $y(2k+1)$。由此得到如图5-2b所示的二阶展开DFG，对应于图5-1b所示的二阶展开DSP程序。

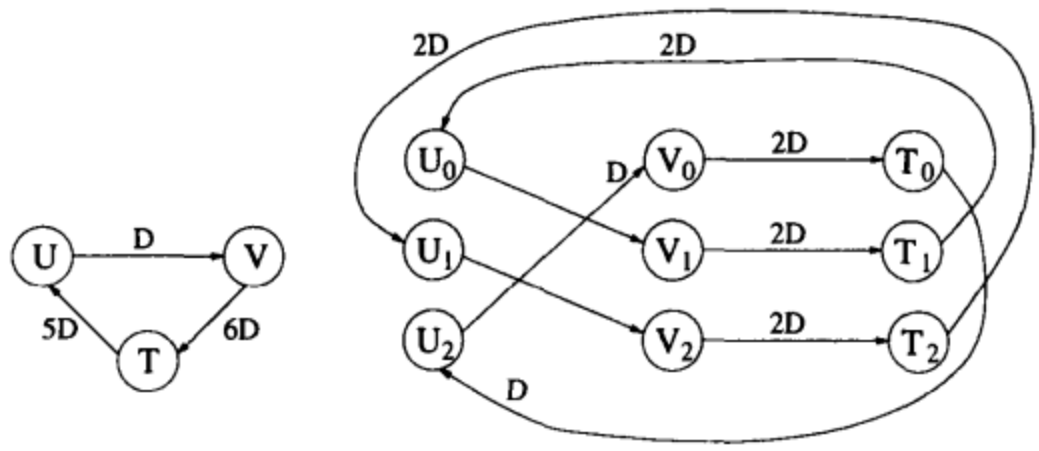
图5-3和图5-4给出了另外两个展开的例子。图5-3a中的DFG以4为展开因子展开得到了如图5-3b所示的DFG，图5-4a中的DFG以3为展开因子展开得到了如图5-4b所示的DFG。



a) 原始DFG　　b) 4阶展开DFG

图 5-3

当 $w<J$ 时，展开原始DFG中一条延迟为 w 的边，相当于在 J 阶展开DFG中生成了 $J-w$ 条没有延迟的边和 w 条延迟为1的边。



a) 原始DFG　　b) 3阶展开DFG

图 5-4

展开的一个重要性质是展开保持DSP程序的优先约束不变（unfolding preserves precedence constraints of a DSP program）。在原始DFG中的 e 条边清楚地表示出在原有程序一次迭代中的优先约束，而在 J 阶展开DFG中的 Je 条边则清楚地表示出程序的 J 次迭代有同样的约束。为了表明本节描述的展开算法的确在生成展开DFG时保持了与原始DFG相同的优先约束，我们需要表明：在展开DFG中延迟为 $\left\lfloor \frac{i+w}{J} \right\rfloor$ 的边 $U_i \to V_{(i+w)\%J}$ 对应于原始DFG中延迟为 w 的边 $U \to V$。

展开变换是基于这样的事实，即 J 阶展开DFG中节点 U_i 第 k 次迭代执行的是原始DFG

122 ~ 123

中节点 U 的第（$Jk+i$）次迭代。在展开DFG中，由于边 $U_i \to V_{(i+w)\%J}$ 上的延迟 $\left\lfloor \frac{i+w}{J} \right\rfloor$，节点 U_i 第 k 次迭代的输出被节点 $V_{(i+w)\%J}$ 第 $\left(k+\left\lfloor \frac{i+w}{J} \right\rfloor\right)$ 次迭代使用。节点 U_i 第 k 次迭代对应于节点 U 第（$Jk+i$）次迭代，而节点 $V_{(i+w)\%J}$ 第 $\left(k+\left\lfloor \frac{i+w}{J} \right\rfloor\right)$ 次迭代对应于节点 V 第 $\left(J\left(k+\left\lfloor \frac{i+w}{J} \right\rfloor\right)+(i+w)\%J\right)$ 次迭代。因此，在原始DFG中，节点 U 第（$Jk+i$）次迭代的输出被节点 V 第 $\left(J(k+\left\lfloor \frac{i+w}{J} \right\rfloor)+(i+w)\%J\right)$ 次迭代所使用，则 $U \to V$ 边的延迟为

$$J\left(k+\left\lfloor \frac{i+w}{J} \right\rfloor\right)+(i+w)\%J-(Jk+i)$$

它可以简化为

$$\left(J\left\lfloor \frac{i+w}{J} \right\rfloor+(i+w)\%J\right)-i$$

式子 $\left(J\left\lfloor \frac{i+w}{J} \right\rfloor+(i+w)\%J\right)$ 的值为 $i+w$，因此，边 $U \to V$ 的延迟正如预计算那样为 $(i+w)-i=w$。

5.3 展开的属性

本节讨论展开算法的一些基本属性。这些属性对于思考第5.5节中展开算法的应用是很有用的。

属性5.3.1 *展开算法保持DFG中的延迟数目不变。*

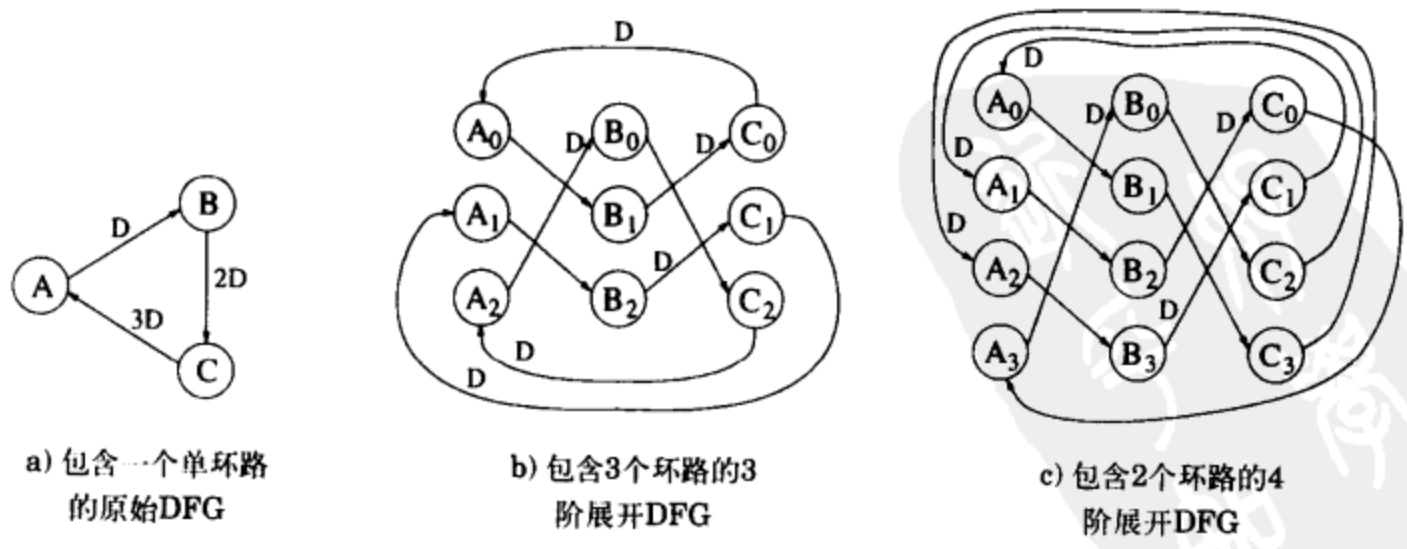
该属性基于以下事实，即 J 阶展开DFG中边 $U_i \to V_{(i+w)\%J}$, $i=0, 1, \ldots, J-1$ 的延迟之和与原始DFG中边 $U \to V$ 的延迟数相同。用数学表达式可表达为

$$\left\lfloor \frac{w}{J} \right\rfloor + \left\lfloor \frac{w+1}{J} \right\rfloor + \cdots + \left\lfloor \frac{w+J-1}{J} \right\rfloor = w \tag{5-3}$$

上式的证明留作练习（见习题3）。这个属性可由图5-2、图5-3和图5-4观察得到。

观察展开环路时发生的现象十分有趣，假定 l 是原始DFG中延迟为 w_l 的一个环路，A 为 l 中的节点。环路 l 可表示为延迟为 w_l 的路径 $A \xrightarrow{l} A$。如果经过 l 环路 p 次（$p \geqslant 1$），就会得到延迟为 pw_l 的路径 $A \xrightarrow{l} A \xrightarrow{l} \ldots \xrightarrow{l} A$。开始于节点 $A_i (0 \leqslant i \leqslant J-1)$，展开后相对应的路径在 J 阶展开DFG中是 $A_i \to A_{(i+pw_l)\%J}$，而且如果 $i=(i+pw_l)\%J$，这个路径就形成了展开DFG中一个环路。我们想找到 p 的最小值（即环路 l 所经过的最小次数），以便使得展开DFG中的路径 $A_i \to A_{(i+pw_l)\%J}$ 是一个环路。 124

例如，图5-5a中的DFG包含一个延迟为 $w_l = 6$ 的单环 $l = A \to B \to C \to A$。图5-5b和图5-5c分别给出了一个3阶展开和4阶展开的DFG。对于 $J=3$ 的情况，式 $i=(i+pw_l)\%J$ 变成式 $i=(i+6p)\%3$，当 $i=0, 1, 2$，$p=1$ 时成立。因此，展开该环路 $p=1$ 次就得到3阶展开DFG中的一个环。从 A_0 开始展开环路 l 得到环路 $A_0 \to B_1 \to C_0 \to A_0$。从 A_1 展开环路 l 得到环路 $A_1 \to B_2 \to C_1 \to A_1$，从 A_2 展开环路 l 得到环路 $A_2 \to B_0 \to C_2 \to A_2$。3阶展开DFG包括3个这样的环路。对于 $J=4$ 的情况，式 $i=(i+pw_l)\%J$ 变成式 $i=(i+6p)\%4$，当 $i=0, 1, 2, 3$，$p=2$ 时成立，而 $p=1$ 时不成立。因此，展开该环路 $p=2$ 次得到4阶展开DFG中的一个环。从 A_0 开始，展开环路 $l'=A \to B \to C \to A \to B \to C \to A$ 得到环路 $A_0 \to B_1 \to C_3 \to A_2 \to B_3 \to C_1 \to A_0$。从 A_1 开始，展开环路 l' 得到环路 $A_1 \to B_2 \to C_0 \to A_3 \to B_0 \to C_2 \to A_1$。4阶展开DFG包括这样2个环路。



a) 包含一个单环路的原始DFG

b) 包含3个环路的3阶展开DFG

c) 包含2个环路的4阶展开DFG

图 5-5

展开一个环路时，无需经过展开的过程，就可以确定展开环路的数目。这个数目可以由

使得路径 $A_i \to A_{(i+pw_l)\%J}$ 成为一个环路的最小 p 值来确定，即满足 $i=(i+pw_l)\%J$ 的最小 p 值。以下两个引理表明了这一点。第一个引理只给出结论，没有证明。

引理5.3.1 $i=(i+pw_l)\%J \Leftrightarrow pw_l=qJ$，$q$ 为整数。

125 **引理5.3.2** 满足 $pw_l = qJ$ 的最小正整数 p 为 $J/\gcd(w_l, J)$。

引理5.3.2的证明：式子 $pw_l = qJ$ 可以写为 $\left(\frac{a}{w_l}\right)w_l=\left(\frac{a}{J}\right)J$。$p=\frac{a}{w_l}$ 和 $q=\frac{a}{J}$ 必须为整数，因为 $p=\frac{a}{w_l}$ 必须尽可能地小，所以 a 必须尽可能地小，因此，$a=\mathrm{lcm}\{w_l, J\}$。运用 $\mathrm{lcm}\{w_l, J\}\ \gcd\{w_l, J\} = w_lJ$，最小 p 值为 $p=\frac{\mathrm{lcm}\{w_l, J\}}{w_l}=\frac{J}{\gcd\{w_l, J\}}$。

由引理5.3.1知，当且仅当 $pw_l = qJ$ 时，路径 $A_i \to A_{(i+nw_l)\%J}$ 才是一个环路。满足这一点的最小 p 值等于形成一个展开环路所需要经过环路 l 的次数，引理5.3.2表明这个数等于 $J/\gcd(w_l, J)$，其中，$\gcd(a, b)$ 表示 a 和 b 的最大公约数。因此，展开环路包含了环路 l 中每个节点的 $J/\gcd(w_l, J)$ 个拷贝，展开DFG总共包含了每个节点的 J 个拷贝，因此，一定有由于展开 l 环路而得到的 $J/(J/\gcd(w_l, J))=\gcd(w_l, J)$ 个展开后的环路。展开后的DFG包含了 w_l 个延迟（由属性5.3.1），$\gcd(w_l, J)$ 个展开环路中的每一个都包含了 $w_l / \gcd(w_l, J)$ 个延迟。这被总结为以下的属性。

属性5.3.2 *原始DFG中延迟为 w_l 的环路 l 的 J 阶展开得到展开后的DFG中的 $\gcd(w_l, J)$ 个环路，$\gcd(w_l, J)$ 个环路中的每个环路包含了 $w_l/\gcd(w_l, J)$ 个延迟以及环路 l 中出现的每个节点的 $J/\gcd(w_l, J)$ 个拷贝。*

以下属性描述了展开对迭代边界的影响。

属性5.3.3 *展开一个迭代边界为 T_∞ 的DFG，会得到一个迭代边界为 JT_∞ 的 J 阶展开DFG。*

很明显，既然 J 阶展开DFG代表了原始DFG的 J 次迭代，那么 J 阶展开DFG的迭代边界就应该是原始DFG迭代边界的 J 倍。数学上，原始DFG的迭代边界为 $T_\infty=\max_l\left\{\frac{t_l}{w_l}\right\}$，利用特性5.3.2，可以得到展开后的DFG的迭代边界为

$$T'_\infty = \max_l\left\{\frac{(J/\gcd(w_l, J))t_l}{w_l/\gcd(w_l, J)}\right\} = J\max_l\left\{\frac{t_l}{w_l}\right\} = JT_\infty.$$

作为一个例子，假设图5-1中的加法和乘法分别需要3 u.t.和6 u.t.。图5-1a中原有的DFG
126 的迭代边界为9/9=1 u.t.，图5-1b中的2阶展开DFG的迭代边界为18/9=2 u.t.。

5.4 关键路径、展开和重定时

本节将原始DFG的关键路径 G 与 J 阶展开DFG的关键路径 G_J 联系起来。同时探讨重定时后的DFG的 J 阶展开版本 $(Gr)_J$ 与 J 阶展开DFG的 $(G_J)_r$ 重定时后的关系[8]~[10]。

属性5.4.1 *考虑原始DFG中延迟为 w 的路径。当 $w<J$ 时，该路径的 J 阶展开将得到 $(J-w)$ 个无延迟路径和 w 个延迟为1的路径。*

推论5.4.1 *原始DFG中包含 J 个或更多延迟的任何路径都能生成 J 个路径，其每个路径具*

有1个或更多延迟。因此，原始DFG中具有 J 个或更多延迟的路径不能生成一条在J阶展开DFG中的关键路径。

运用属性5.4.1，我们可以重定时原有的DFG，以便使得重定时后的DFG的 J 阶展开满足特定关键路径的计算时间 c。如果原始DFG存在一条路径，其计算时间为 c、延迟元件少于 J 个，那么，展开后的DFG的关键路径就能够为 c。这就得出用于在重定时DFG的展开版本中为缩短关键路径而采取重定时的关键路径约束。如果 $D(U, V) > c$，那么 $Wr(U, V) = W(U, V) + r(V) - r(U) > J$，或者 $r(U)-r(V) < W(U, V)-J$。通常的可行性约束 $w(e) + r(V) - r(U) < 0$ 应和关键路径约束一起使用。

引理5.4.1　任何通过重定时 J 阶展开DFG的 G_J 所能得到的可行时钟周期，都可以通过直接对原始DFG的 G 重定时和以 J 为展开因子的展开来得到。

证明：令 r' 是DFG的展开 G_J 的一个合理重定时，它所导致的关键路径为 c。令 r 是 G 的一个重定时，其定义为：

$$r(U) = \sum_{i=0}^{J-1} r'(U_i)$$

我们来证明，r 是对原始DFG可行的重定时，它使得原始DFG的 J 阶展开版本 G_r 具有一条关键路径 c。先考虑重定时 r 的可行性证明。为此考虑 G 中延迟为 w 的边 $U \to V$。因为 r' 是对展开后的DFG的一个合理重定时，所以

$$r'(U_i) - r'(V_{(i+w)\%J}) \leqslant \left\lfloor \frac{i+w}{J} \right\rfloor$$

把上式中由 $i=0$ 到 $J-1$ 所得到的 J 个不等式相加，得到 $r(U) - r(V) \leqslant w$，从而证明 r 是可行的重定时。再考虑 J 阶展开DFG的关键路径为 c 的证明。因为 r' 满足展开后的DFG的关键路径约束，亦即，

127

$$r'(U_i) - r'(V_{(i+w)\%J}) \leqslant \left\lfloor \frac{i+w}{J} \right\rfloor - 1$$

因此，如果 $D(U_i \to V_{(i+w)\%J}) > c$，则对 i 从0到 $J-1$ 的所有式子求和，可得

$$r(U) - r(V) \leqslant W(U,V) - J$$

这正是所要得到的不等式。

5.5 展开的应用

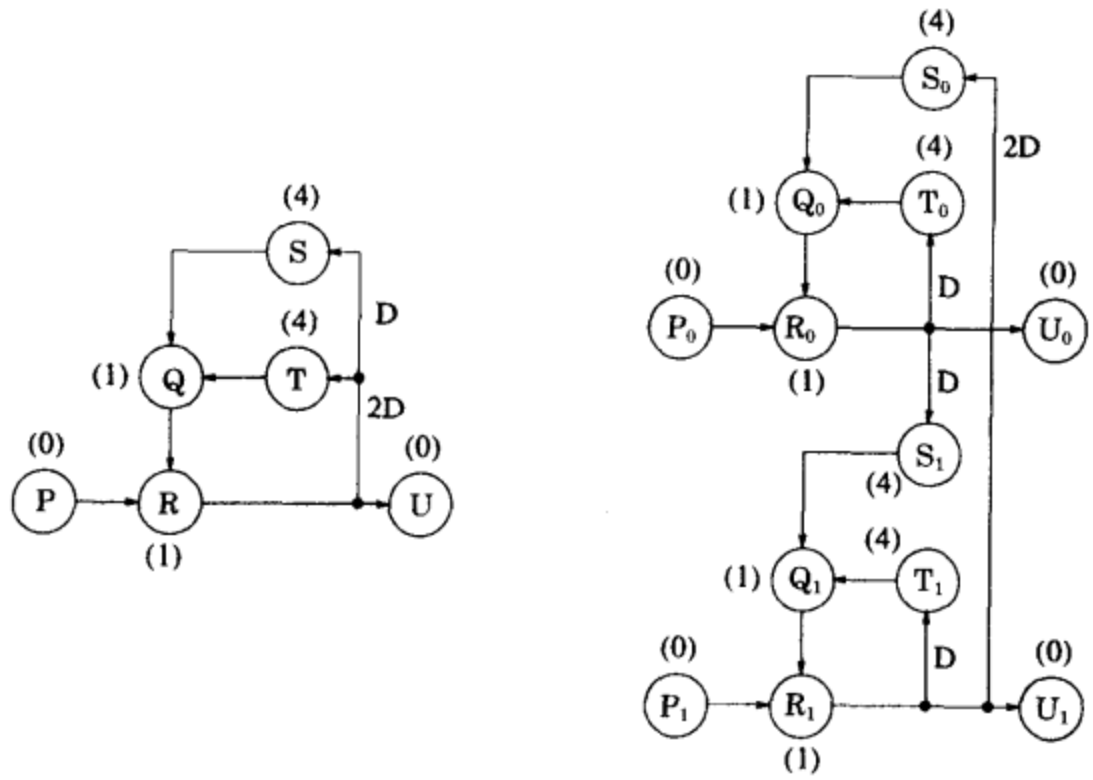
5.5.1 采样周期的缩短

迭代边界 T_∞ 是指一个递归DSP程序的迭代周期的下限。即使有无穷多个处理器，DSP程序的实现绝不可能达到小于迭代边界的迭代周期。迭代边界的完整描述见第2章。在某些情况下，在不用展开时，DSP程序不能实现等于迭代边界的迭代周期。本节讨论两种这样的情况，展开算法使得DSP算法可以用等于迭代边界的迭代周期实现。

第一种迭代周期不能等于迭代边界的情况是当DFG中存在某个节点的计算时间大于 T_∞ 时（这里假设该节点不能分拆为两个计算时间更少的子节点）的情况。这种情况下，甚至重

定时也不能把DFG中关键路径的计算时间减少到 T_∞。例如，图5-6a中的DFG的迭代边界为 T_∞=3，但是，节点 S 和 T 都需要4 u.t.计算时间，因此重定时后的采样周期为4u.t.。这个DFG可以以2为展开因子展开得到图5-6b中的DFG。这个展开后的DFG的迭代边界为 T_∞=6，它的关键路径为6 u.t.。展开后的DFG在6u.t.内完成原有DSP程序的两次迭代，这样，展开后的DFG的采样周期为6/2=3u.t.，它等于原始DFG的迭代边界。综上所述，由于节点计算时间大于迭代边界，使得图5-6a中原始DFG的采样周期无法等于迭代边界，然而图5-6b中展开后的DFG，可以使得采样周期等于原始DFG的迭代边界。总而言之，如果节点 U 的计算时间 t_U大于迭代边界 T_∞，就需要使用$\lceil t_U/T_\infty \rceil$阶展开，$\lceil x \rceil$运算表示对 x 向上取整，即取大于或等于 x 的最小整数。在上述例子中，t_U=4 和 T_∞=3，因此，采用了$\lceil 4/3 \rceil$阶展开（即2阶展开）。

128



a) 原始DFG的迭代边界为 T_∞=3，最小采样周期为4

b) 2阶展开DFG的迭代边界为 T_∞=6，最小采样周期为6/2=3

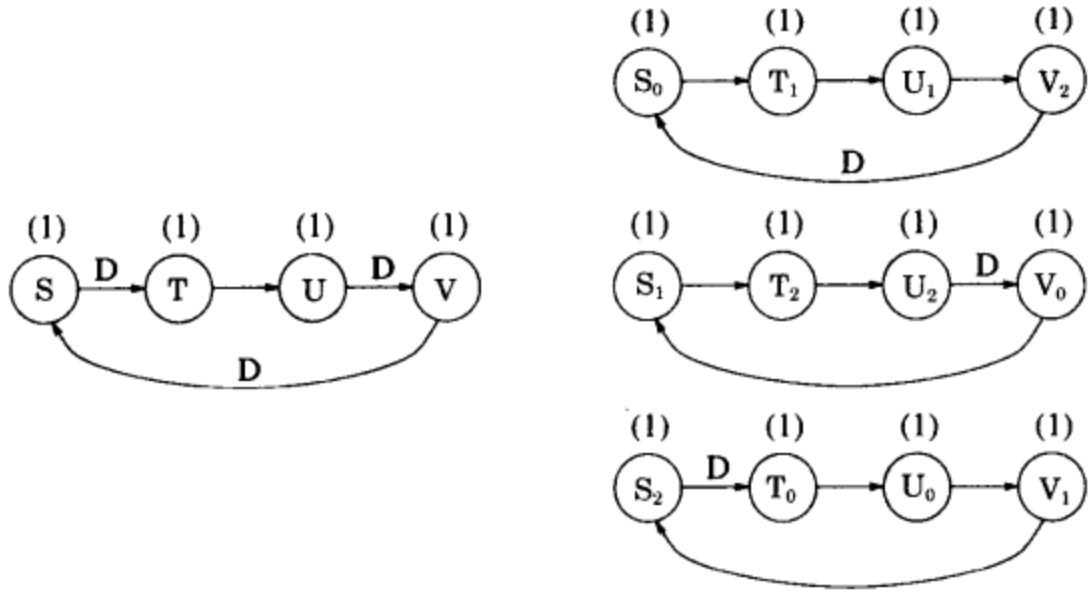
图 5-6

第二种迭代周期不能等于迭代边界的情况是迭代周期不为整数时的情况。图5-7a给出了这种情况的一个简单例子，其中DFG的迭代边界为 T_∞=4/3；然而，即使重定时也无法获得一条小于2 u.t.的关键路径。该DFG可以以3为展开因子展开得到图5-7b中的DFG。这个展开后的DFG的迭代边界为 T_∞=4，它的关键路径为4 u.t.。展开后的DFG在4 u.t.时间内3次迭代原有DSP程序，因此，展开后的DFG的采样周期为4/3 u.t.，即等于原始DFG的迭代边界。综上所述，由于迭代边界不为整数，使得图5-7a中原始DFG的采样周期不能等于迭代边界，然而对图5-7b中展开后的DFG，可以使采样周期等于原始DFG的迭代边界。总而言之，如果关键环路边界的形式为 t_l/w_l，其中 t_l 和 w_l 互为素数，则应采用 w_l 阶展开。例如，若 t_l=60，w_l=45，则 t_l/w_l 可以写成 t_l/w_l=4/3，所以应采用3阶展开。

也可能会发生最长的节点计算时间大于迭代边界 T_∞ 而 T_∞ 不为整数的情况。这种情况下，使得迭代周期能够等于迭代边界的最小展开因子是能使 JT_∞ 成为（大于或等于最长节点计算时间的）整数时 J 的最小值。例如，若 T_∞=4/3 并且最长节点计算时间为6，则使得迭代周期等

129

于迭代边界的最小展开因子为 J=6，因为它是满足 J (4/3) 为整数并且 J (4/3) > 6 的最小 J 值。



a) 原始DFG的迭代边界为 T_∞=4/3，最小采样周期为2

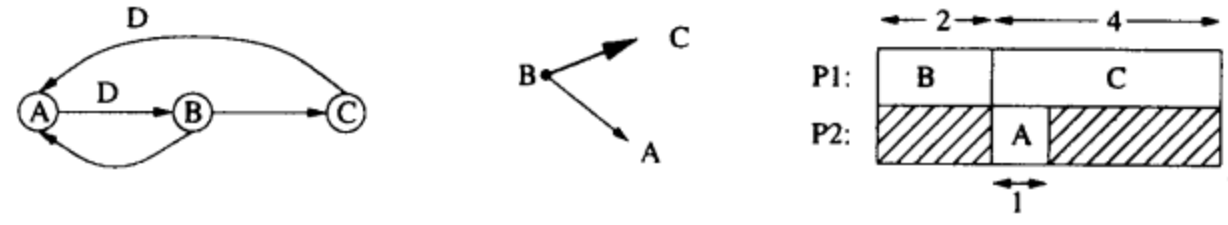
b) 3阶展开DFG的迭代边界为 T_∞=4，最小采样周期为4/3

图 5-7

每一条环路延迟都为1的DFG称为*理想速率*（perfect-rate）DFG。一个理想速率DFG总是可以调整使得DFG的迭代周期等于迭代边界。如果一个DFG不是理想速率的，则总可以展开它，使得展开后的DFG是理想速率的。为了明白这一点，请回忆属性5.3.2。属性5.3.2指出，J 阶展开一个延迟为 w_l 的环路 l 时，在展开之后的DFG中生成了gcd(w_l，J)个环路，而每一个这样的环路都包含了 w_l / gcd(w_l，J) 个延迟。因此，若 gcd(w_l，J) = w_l（即 J 为 w_l 的倍数），则展开后的每条环路的延迟就为1，并且展开后的DFG是理想速率的。因为 gcd(w_l，J) = w_l 对原始DFG中的所有环路 l 都成立，所以 J 必须是每一条环路 l 的延迟 w_l 的倍数。选择 J 等于环路延迟数的最小公倍数，就保证了 J 是每一条环路 l 的延迟 w_l 的倍数，从而保证了展开后的DFG是理想速率的。例如，图5-6a中环路延迟的最小公倍数（lcm）是lcm{2，3}=6，因而以6为展开因子展开这个DFG时，就能保证展开后的DFG的采样周期等于迭代边界。然而前面的例子表明，展开因子2也可用来使得采样周期等于迭代边界，因此，环路延迟的最小公倍数不是保证展开后的DFG的采样周期等于迭代边界所必需的最小展开因子（事实上，它是展开因子的上限）。请注意，最小公倍数独立于节点计算时间。如何确定最小展开因子，以及如何把展开和重定时结合起来，这些问题不在本书讨论范围之内。 130

展开可以用于缩短可编程DSP的迭代周期。通过删除DFG中所有带有延迟的边，可以得到无环路优先图，周期调度表即可由此建立（见附录B）。若一个 J 阶展开DFG的周期调度表的周期为 T，则平均迭代周期为 T/J。调度表可以是*重叠的*，或是*非重叠的*（overlapped or nonoverlapped），在非重叠调度表中，调度表的周期等于无环路优先图的关键路径。在有重叠调度表中，调度表的周期小于无环路优先图的关键路径[2]。例如，考虑图5-8a所示DFG，其无环路优先图和周期调度表分别如图5-8b和图5-8c所示。这种情况的调度表是非重叠的。该DFG的迭代周期为3.5 u.t.，非重叠周期调度表的周期为6 u.t.。注意，这里无环路优先图中粗线标出的路径表示关键路径，对应的2阶展开DFG的无环路优先图和周期调度表如图5-9所示。它的有重叠周期调度表的周期为7 u.t.；因此，平均迭代周期为3.5 u.t.，即等于迭代

边界。读者可以自行验证，把节点B的输入边去掉延迟，而在节点 B 的每一条输出边上加上延迟来对图5-8a中DFG进行重定时，可以把关键路径从6 u.t.缩短到4 u.t.。进而，对这个重定时后的DFG进行2阶展开，可以得到一个以等于迭代边界的平均迭代周期为周期的非重叠调度表。通常，构建一个重叠调度表比构建一个非重叠调度表要复杂得多。

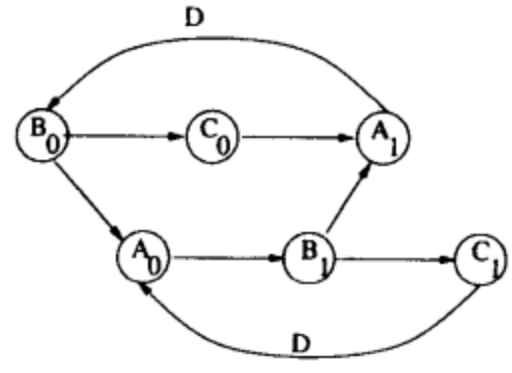


a) 一个节点 A、B 和 C 的执行时间分别为1 u.t.、2 u.t. 和 4 u.t.的DFG

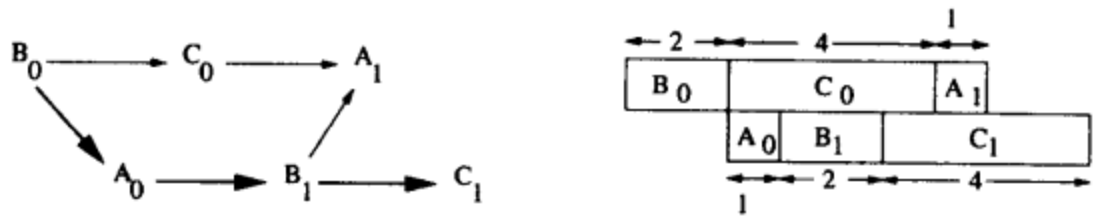
b) 对应的无环路优先图

c) 周期为6 u.t.的非重叠周期调度表

图 5-8



a) 2阶展开DFG



b) 它的无环路优先图

c) 周期为7 u.t.的重叠周期调度表

图 5-9

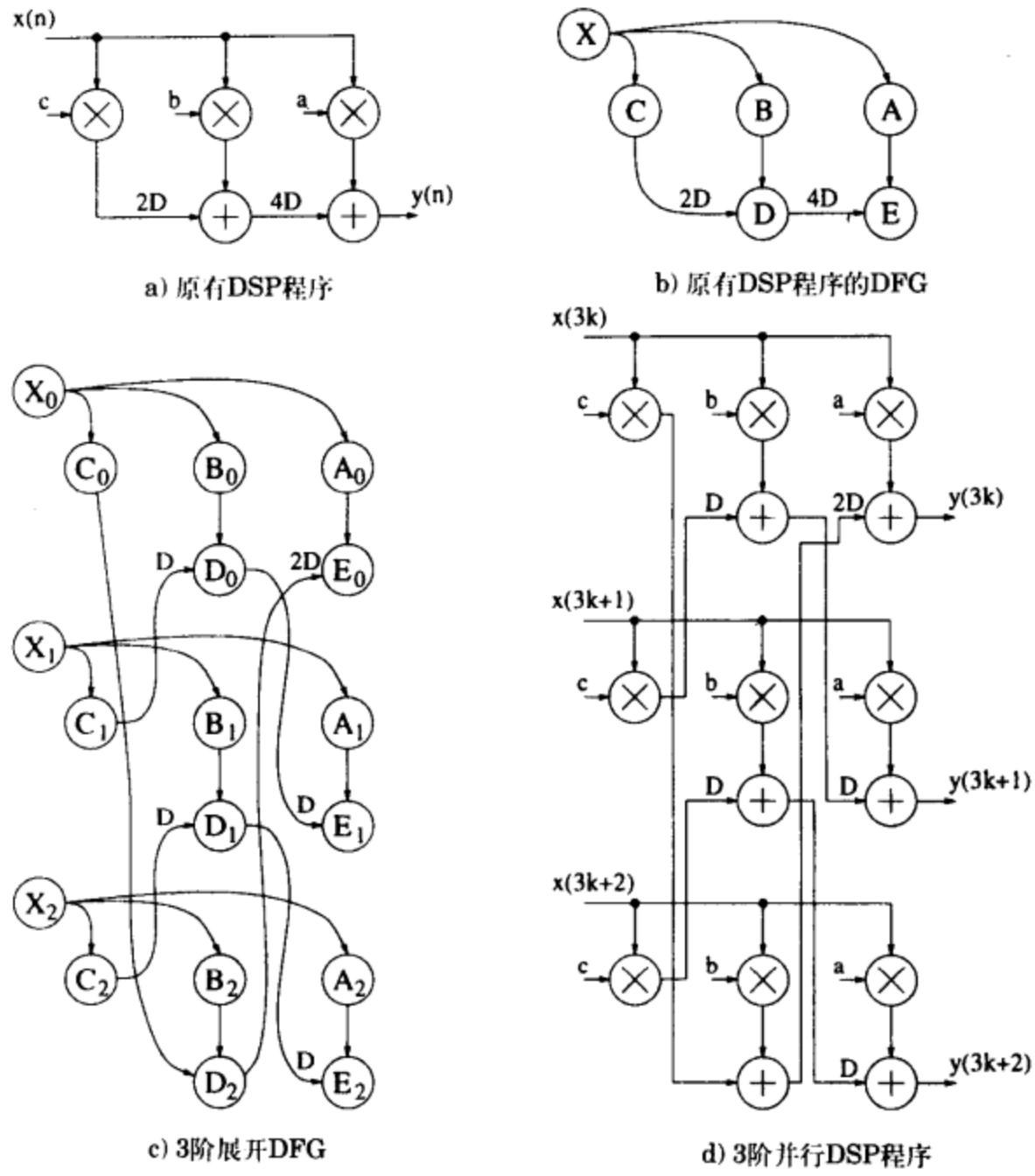
5.5.2 并行处理

展开变换的一个直接应用就是从串行处理架构设计出并行处理架构。在字级上，意味着字并行架构可以从字串行架构设计出来。在位级，意味着位并行和数字串行的架构可以从位串行架构设计出来。以下各节讨论了如何应用展开技术来设计并行处理架构。

5.5.2.1 字级并行处理

展开变换可以用于从字串行处理架构设计出字并行处理架构。这已由图5-1表明。图5-1a中原有的架构每一个时钟周期处理一个输入样点，并计算出一个输出样点，而5-1b的2阶展开架构每个时钟周期处理两个输入样点，并计算出两个输出样点。通常，以 J 为展开因子展开字串行架构，会生成一个每个时钟周期处理 J 个字的字并行架构。注意如第3章所述的并行

处理架构可以用展开技术来设计。



a) 原有DSP程序　b) 原有DSP程序的DFG

c) 3阶展开DFG　d) 3阶并行DSP程序

图 5-10

作为另一个例子，考虑图5-10a所示的DSP程序 $y(n)=ax(n)+bx(n-4)+cx(n-6)$。为了生成每个时钟周期能够处理多个字的架构，第一步是画出如图5-10b所示的DFG，下一步是展开这个DFG，3阶展开DFG如图5-10c所示，相应的DSP程序如图5-10d所示，图5-10d中的DSP程序是图5-10a中DSP程序的3路并行处理版本。

131 ~ 133

5.5.2.2 位级并行处理

本节的目的是表明如何利用展开变换，由位串行架构导出位并行、数字串行架构。位串行、位并行和数位串行处理方式的描述将在下面给出。假设 W 为数据的字长。

- *位串行处理*（bit-serial processing）：一个时钟周期处理一位，一个完整的字在 W 个时钟周期内处理完成。
- *位并行处理*（bit-parallel processing）：一个 W 位的字在一个时钟周期内处理完成。
- *数字串行处理*（digit-serial processing）：一个时钟周期处理 N 位，一个字在 W/N 个时钟周期内处理完成。参数 N 称为*数字尺寸*（digit size）。

图5-11给出了 $W=6$ 时3种处理方式的不同之处。对于数字串行结构，一个时钟周期处理 N 位或数字尺寸的比特数。我们从假设N是W的因子开始，这个限制将在本节后面去掉。

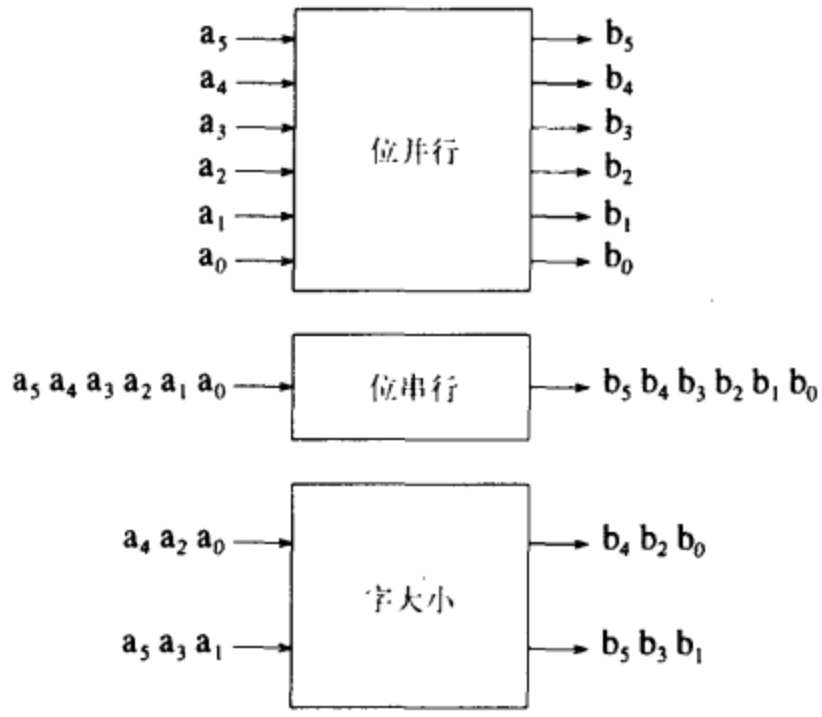


图5-11 字长 $W=6$ 的位并行、位串行和数字并行处理方式

图5-12给出了一个字长 $W=4$ 时的位串行加法器架构。这个加法器把两个4位数 $a_3a_2a_1a_0$和 $b_3b_2b_1b_0$ 相加得到和的4位数 $s_3s_2s_1s_0$，其中位0是最低位，位3是最高位。为了得到数字串行或位并行加法器架构，位串行加法器必须展开。位串行加法器在多数字串行架构中可算典型，它包含一条带有开关（在硬件上对应于一个多路选择器）的边。因此，首先讨论把带有开关的边展开的过程，然后将其用于从位串行架构出发来设计数字串行和位并行的架构。

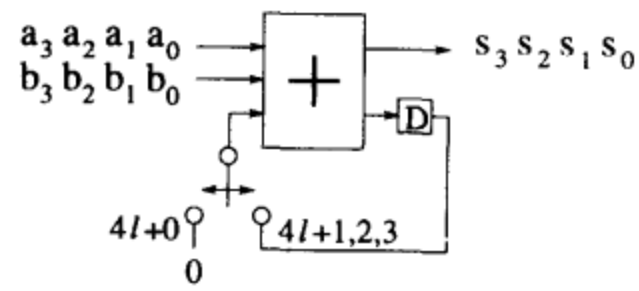


图5-12 字长 $W=4$ 的位串行加法 $s=a+b$

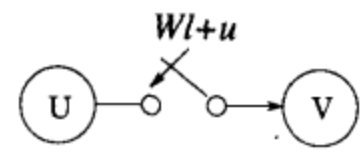


图5-13 一个开关

考虑图5-13中的边$U\rightarrow V$。为了以展开因子J展开这条边，做两个基本假设：

- 字长 W 是展开因子 J 的倍数，即 $W=W'J$。
- 所有进出开关的边都不含延迟。

记住这两个假设，图5-13中的边就可以通过以下两步展开：

(1) 将开关时刻写为

$$Wl+u=J\left(W'l+\left\lfloor\frac{u}{J}\right\rfloor\right)+(u\%J)$$

(2) 展开后的图中，从节点 $U_{u\%J}$ 到节点 $V_{u\%J}$ 画一条不带延迟的边，它在 $W'l+\left\lfloor\frac{u}{J}\right\rfloor$ 时刻开关。

对这一点的证明留作一个练习（见习题18）。例如，假定 $W=12$，$u=7$，考虑以 $J=3$ 为展开因子展开图5-13中的开关。这是满足以上两个假设的，因为 $W=W'J=>12=(4)(3)$，而且边不带延迟。开关时刻可以写为

$$12l+7=3\left(4l+\left\lfloor\frac{7}{3}\right\rfloor\right)+(7\%3)=3(4l+2)+1$$

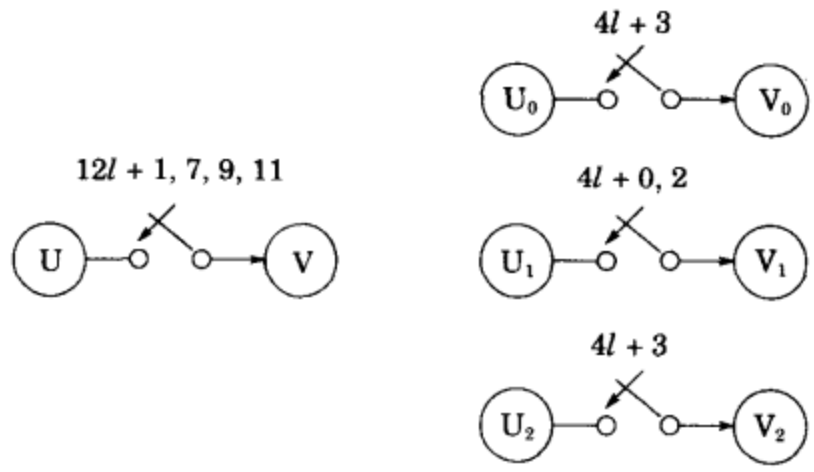
134 ~ 135

在展开后的DFG中，展开后的边是从节点 U_1 到节点 V_1 的边，并且在 $4l+2$ 时刻开关。

如果开关有多个开关时刻，那么每一个开关时刻都要单独处理。图5-14a中的开关在4个开关时刻 $12l+1$、$12l+7$、$12l+9$ 和 $12l+11$ 闭合。为了以 $J=3$ 展开DFG，这些开关时刻写为

$$\begin{aligned}12l+1 &= 3(4l+0)+1\\ 12l+7 &= 3(4l+2)+1\\ 12l+9 &= 3(4l+3)+0\\ 12l+11 &= 3(4l+3)+2\end{aligned}$$

展开后的DFG如图5-14b所示。



a) 原始DFG　　b) 3阶展开DFG

图 5-14

如果一条边包含一个开关和一个正的延迟数，可以用一个虚节点将这个问题简化为该条边不带延迟的情形，图5-15a中的DFG包含边 $A\rightarrow C$，它有一个开关和两个延迟。展开前，这个DFG可以用一个如图5-15b中所示的虚节点 D 重画。为了以 $J=3$ 展开，这个DFG中的6个开关时刻可以写为

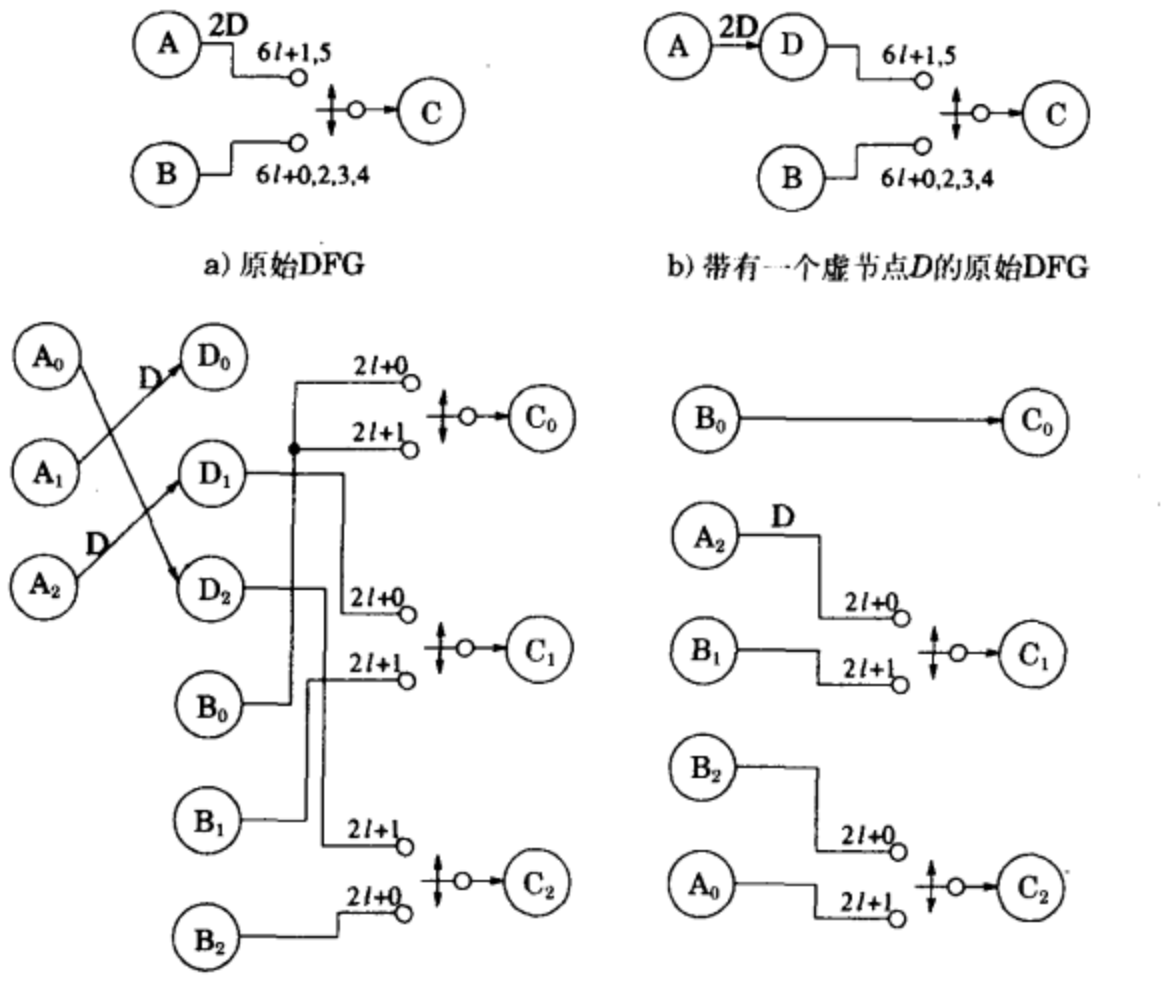
$$\begin{aligned}6l+0 &= 3(2l+0)+0\\ 6l+1 &= 3(2l+0)+1\\ 6l+2 &= 3(2l+0)+2\\ 6l+3 &= 3(2l+1)+0\\ 6l+4 &= 3(2l+1)+1\\ 6l+5 &= 3(2l+1)+2\end{aligned}$$

136

展开后的DFG如图5-15c所示。虚节点 D_i，$i=0$，1，2，以及“死”节点 A_1 都可以消除，如图5-15d中经简化之后的展开DFG所示。可以看到，展开包含开关的电路不再保留延迟。

运用这些技术展开开关，位并行和数字串行架构可以从位串行架构系统得到。可以利用图5-12中的位串行加法器来表明这一点。与这个加法器相对应的DFG如图5-16所示，这个DFG的4阶展开和2阶展开如图5-17所示，这些展开后的DFG的架构如图5-18所示。图5-18a中的4阶展开架构是一个简单的位并行串行进位加法器。图5-18b中的2阶展开结构是数字尺

寸等于2的数字串行加法器。



a) 原始DFG　　b) 带有一个虚节点D的原始DFG

c) 展开后的DFG　　d) 去掉虚节点和死节点的展开后的DFG

图 5-15

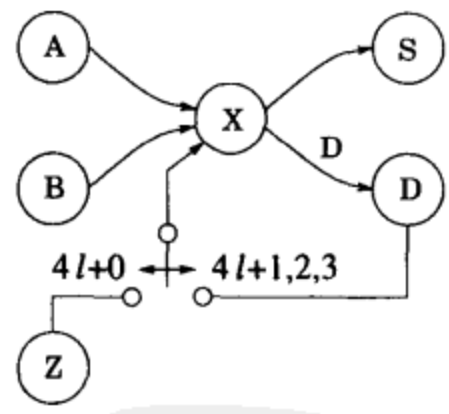
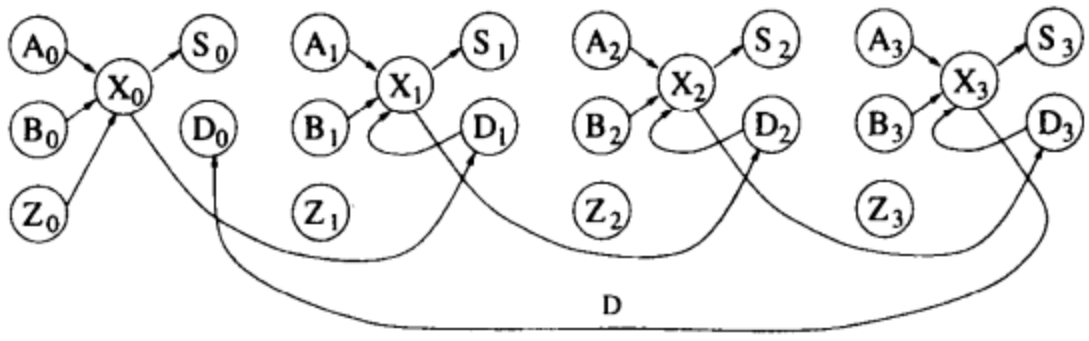


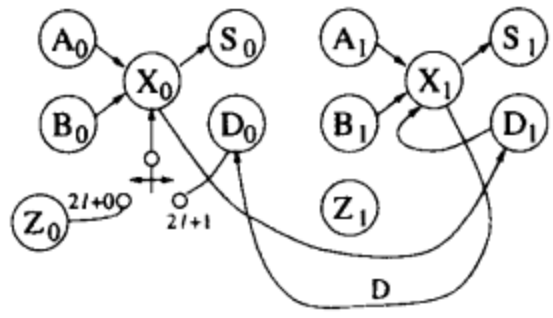
图5-16 与图5-12中的位串行加法器相对应的DFG

如果字长 W 不是展开因子 J 的倍数，DFG仍然可以展开，即求 $L=\mathrm{lcm}\{W, J\}$，并且将开关时刻 $Wl+u$ 替换为 L/W 个开关时刻 $Ll+u+wW$，$w=0$ 到 $L/W-1$。现在，开关的周期从 W 变为 L，并且每次开关扩展了 L/W 倍。既然开关周期是展开因子的倍数，展开可以如前所述那样进行。通过将图5-16中的DFG以 $J=3$ 为展开因子展开表明了这一点，这个DFG重画于5-19a，使得所有的开关时刻都是 $12l+m$ 的形式，因为 $L=\mathrm{lcm}\{4, 3\}=12$。请注意，开关时刻$4l$等效于$12l$，$12l+4$和$12l+8$。类似地，开关时刻$4l+1$扩展为$12l+1$，$12l+5$和$12l+9$；$4l+2$扩展为$12l+2$，$12l+6$和$12l+10$；$4l+3$扩展为$12l+3$，$12l+7$和$12l+11$。3阶展开DFG示于5-19b。这就得到了图5-19c所示的数字尺寸为3的数字串行加法器。因为字长为4，该加法器每4个时钟周期处理3个字。注意，这是一个字长不为数字尺寸倍数的数字串行架构的例子，

说明使用不是字长因子的数字尺寸需要更复杂的控制电路，但是该电路平均4/3个时钟周期处理一个字。如果数字尺寸限定为字长的因子，那么本例中字长需要扩展为6位（使用两个符号扩展位），则每两个时钟周期才能处理一个字。

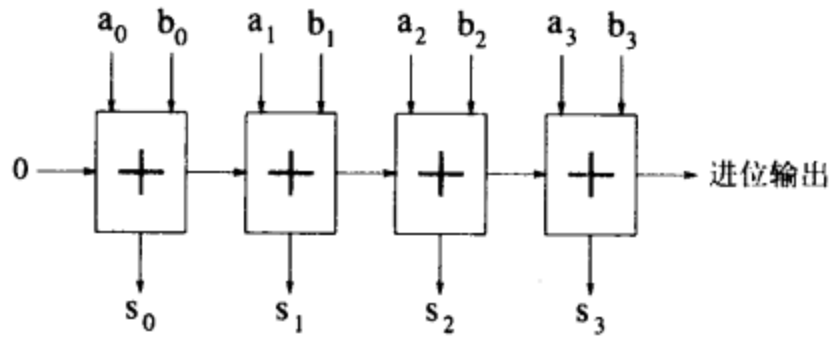


a) 用展开因子 $J=4$ 展开图5-16DFG所得到的DFG

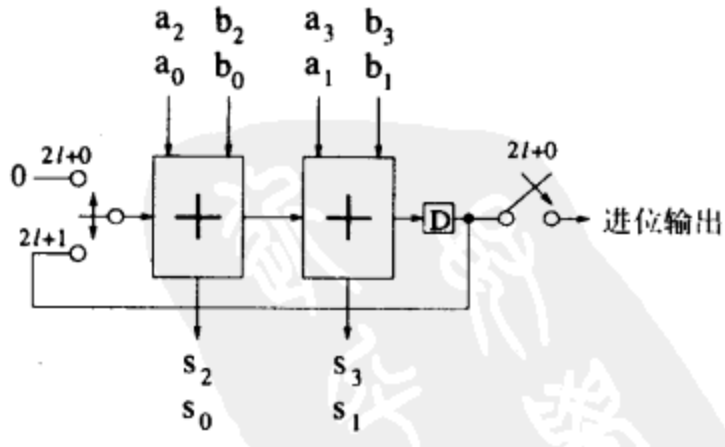


b) 用展开因子 $J=2$ 展开图5-16DFG所得到的DFG

图 5-17



a) 用J=4展开图5-12中的位串行加法器所得到的位并行加法器



b) 用J=2展开图5-12中的位串行加法器所得到的数字串行加法器

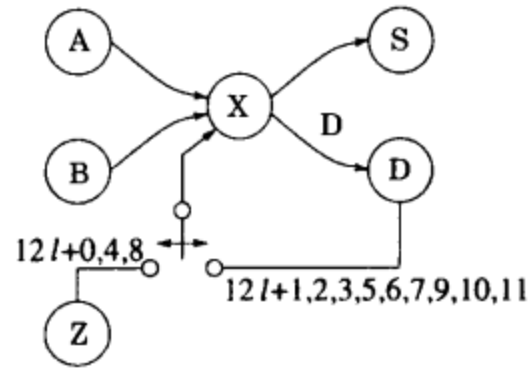
图 5-18

137 ~ 139

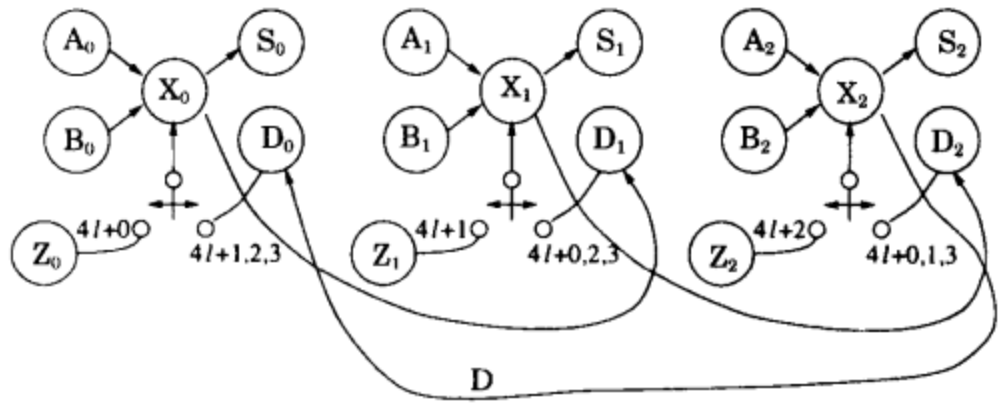
5.6 结论

可以看出，展开变换可以揭示隐藏在用DFG描述的数字信号处理系统中的并发性。因此，

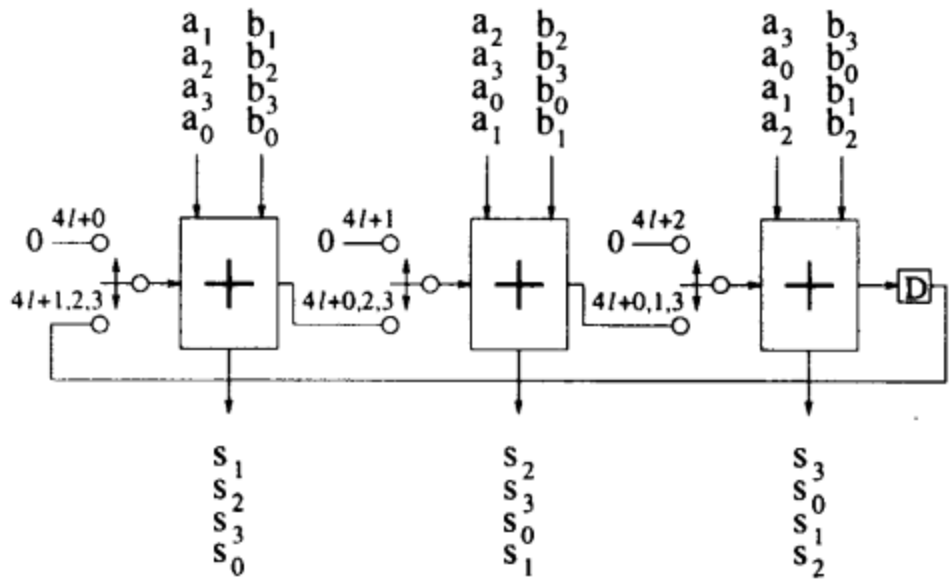
展开技术可以用来缩短DSP算法的迭代周期。展开可以用来生成高速低功耗应用中的字并行型架构。在可编程DSP中，展开技术可以用于减少计算一个输出样值所需的时钟周期数。



a) 重画的图5-16 DFG，以便使它兼容展开因子 J=3



b) 3阶展开DFG



c) 数字尺寸为3的数字串行加法器

图 5-19

5.7 习题

1. 以3和4为展开因子展开图5-20中的DFG。
2. 以2和5为展开因子展开图5-21中的DFG。
3. 证明公式5-3中的展开保持延迟数不变的关系。
4. 本题试图证明，一个由两个基本环路组成的复环路，不会产生一个新的迭代边界。为此，考虑计算时间分别为 T_1 和 T_2、延迟元件数分别为 N_1 和 N_2 的两个环路。令

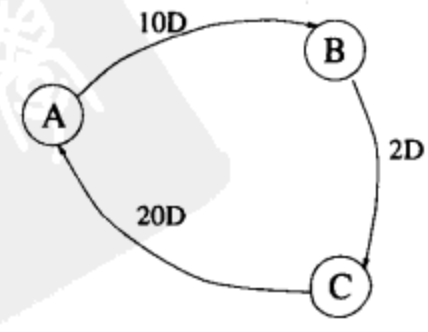


图5-20 习题1的DFG

$T_1/N_1>T_2/N_2$ 成立。证明

$$\frac{T_1+T_2}{N_1+N_2}<\frac{T_1}{N_1} \tag{5-4}$$

5. 本题的目的是为了证明，J 阶展开DFG的关键路径是 J 的单调非减函数[11]。为证明这一点，请证明 J 阶展开DFG的关键路径大于或等于 $J-1$ 阶展开DFG的关键路径。

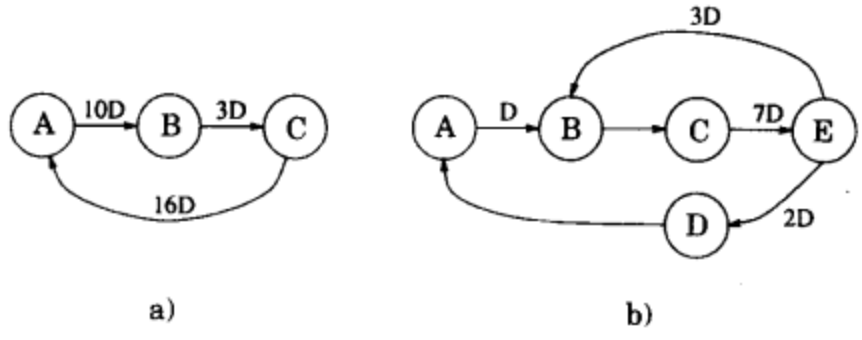


图5-21　习题2的DFG

6. 证明以下迭代算法可以算出一个非递归DFG的能够达到其迭代周期 T 要求的最小展开系数。假设没有使用流水线与/或重定时来缩短关键路径（提示：使用习题5的结果）。

迭代算法:

迭代，直到 $T_{crit}<JT$

{

$$J=\left\lceil\frac{T_{crit}}{T}\right\rceil$$

$T_{crit}=J$ 阶展开DFG的关键路径

}

7. 考虑图5-22中的DFG。找出所需的最小展开系数，使展开后所得到的等效DFG能够达到所需的8 u.t.迭代周期。不要在这个DFG上使用流水线与/或重定时。找出最小的 J，使得 J 阶展开DFG的关键路径小于或等于 $8J$。最小展开系数可由习题6给出的算法来计算得出。

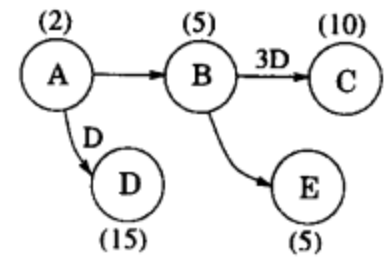


图5-22　习题7的DFG，括号中的数字表示节点的计算时间

140
~
142

8. 在本题中，我们希望表明，改变重定时与展开的顺序，对减小展开后的DFG的关键路径是不重要的。对图5-23中的DFG完成(a)与(b):

(a) 以展开系数 $J=2$ 展开DFG，然后，对展开后的DFG进行重定时以减小时钟周期。

(b) 令本题(a)中决定的重定时函数表示为r'。令重定时函数r为：$r(A)=r'(A_0)+r'(A_1)$；$r(B)=r'(B_0)+r'(B_1)$；$r(C)=r'(C_0)+r'(C_1)$；$r(D)=r'(D_0)+r'(D_1)$。证明：以 r 对图5-23中的DFG进行重定时得到的DFG，其展开后，将得到与本题(a)中相同的关键路径。要证明这一点，首先只需证明，如果 $D(U,V)\geq c$，则对于原始DFG的重定时版本中所有的 U、V 节点对，$W(U,V)+r(V)-r(U)\geq J$ 都成立。

9. 对图5-24中的DFG进行重定时，使得重定时后的DFG的4阶展开版本的关键路径达到14 u.t.。假设加法和乘法分别需要2 u.t.和4 u.t.。

10. 对图5-25中的DFG进行重定时，使得重定时后的DFG的3阶展开版本的关键路径达到4 u.t.。假设加法和乘法分别需要1 u.t.和2 u.t.。

11. 对于图5-24所示的8阶4级流水线全极点格型滤波器，假设加法和乘法分别需要2 u.t.和4 u.t.。
    (a) 用重定时使得DFG的关键路径最小。
    (b) 确定最小展开因子 J，使得 J 阶展开DFG经过重定时后的关键路径为 JT_∞，其中 T_∞是图5-24中原始DFG的迭代边界。

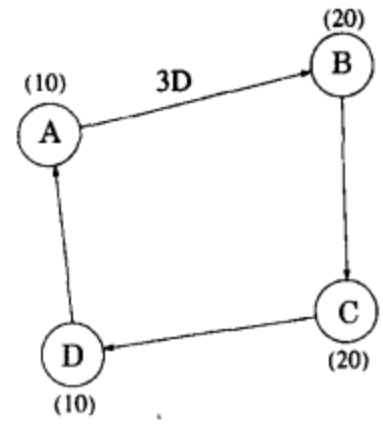


图5-23 习题8的DFG

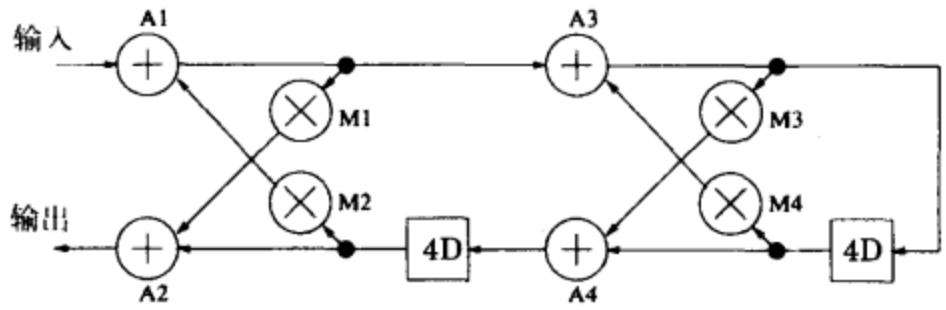


图5-24 4级流水线全带通8阶IIR数字滤波器

12. 重复习题11，假设加法和乘法分别需要1 u.t.和2 u.t.。

13. 对于图5-25中的DFG，寻找最小展开因子 J，使得 J 阶展开DFG经过重定时后的关键路径为 JT_∞，假设加法和乘法分别需要1 u.t.和2 u.t.。

14. 画出FIR滤波器 $y(n)=ax(n)+bx(n-2)+cx(n-5)$ 的直接形式和数据广播形式的DFG，并且以 $J=3$ 为展开因子将这两个DFG展开。

15. 利用 $J=4$，重复习题14。

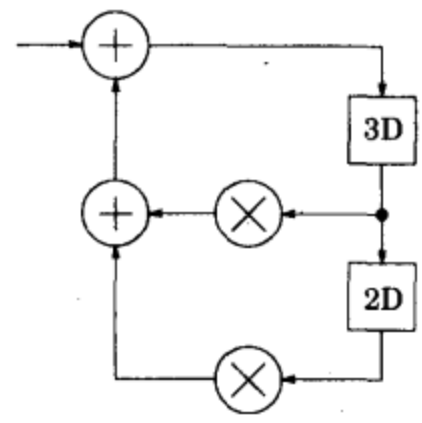


图5-25 习题13中所用的DFG

16. 考虑图5-26中的DFG。括号中的数字是节点的计算时间。
    (a) 该DFG的迭代边界是多少？实际的迭代周期又是多少？
    (b) 重定时该DFG使其迭代周期最小。重定时后的DFG的实际迭代周期是多少？
    (c) 将原始DFG和重定时后的DFG都以2为因子展开。它们的实际的迭代周期分别是多少？
    (d) 确定最小展开因子 J，使得 J 阶展开DFG（由原始DFG展开）经过重定时后的关键路径为 JT_∞，其中 T_∞ 是图5-26中原始DFG的迭代边界。以这个最小展开因子展开该DFG，使得重定时后它的关键路径为 JT_∞。

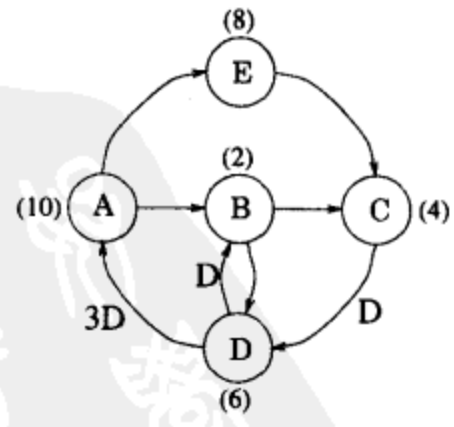


图5-26 习题16的DFG

17. 本题涉及软件中的环路展开（解环）问题，以求在固定硬件结构和给定算法（程序）的情况下使硬件利用效率最大化。它是把展开技术运用于带有3个MAC（乘法累加）

单元的可编程DSP（PDSP）中的并行计算。我们的目的是想利用展开算法，尽可能地提高使用图5-27中PDSP架构的并行FIR滤波应用的硬件使用率。该PDSP存在一个数据瓶颈，也就是说在一个时钟周期内只能从随机存取存储器（RAM）中取出一个操作数作为输入。假定只有输入数据需要从这个产生瓶颈效应的数据总线上取得，而滤波器的系数存储在其他本地存储单元中，并且可以由每个MAC单元在一个时钟周期内自由地存取。画出关于一个7抽头FIR滤波器采用合适展开因子进行计算的调度表，使得硬件使用率最大。

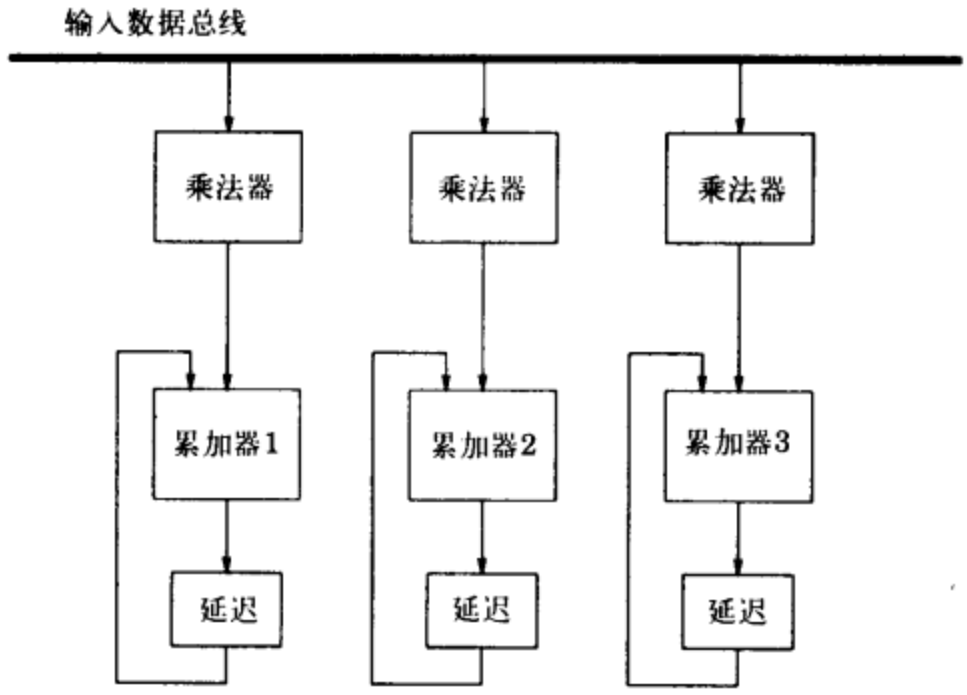


图5-27　PDSP架构

18. 证明当字长 W 是展开因子 J 的倍数时，第5.5.2节所描述的展开开关的算法是正确的。

19. 以2、3、4为展开因子展开图5-28中的电路。

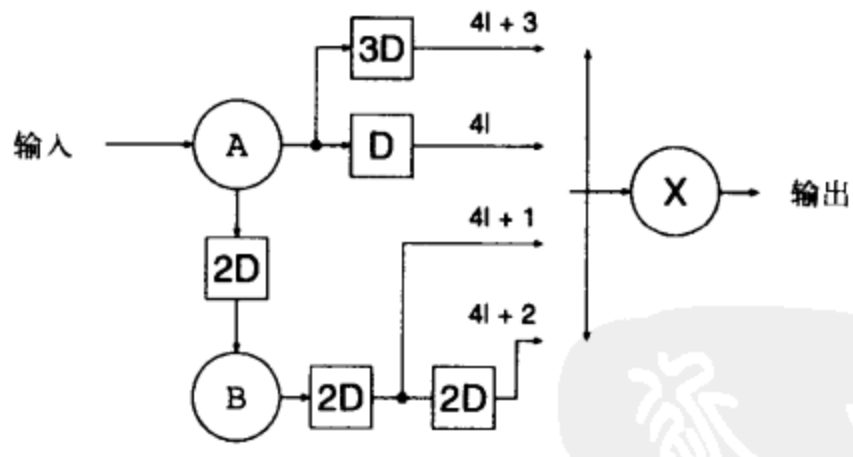


图5-28　习题19的DFG

20. 运用展开变换，求图5-29所示计算的一个2路并行结构。

21. 进行如下展开运算:

(a) 以2和5为展开因子展开图5-30a中的DFG。

(b) 以4为展开因子展开图5-30b中的DFG。

22. 以4和5为因子展开图5-31中的字长为12的位串行加法器，得到相应的数字串行加法器。

23. 图5-32所示的位串行系统实现字长为5位的 $y(n)=\frac{9}{8}y(n-1)+x(n)$。以2为因子展开该

系统，得到一个数字尺寸为2的数字串行架构。

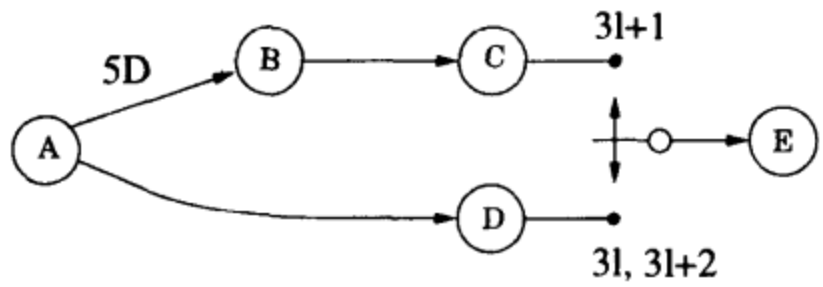


图5-29 习题20的DFG

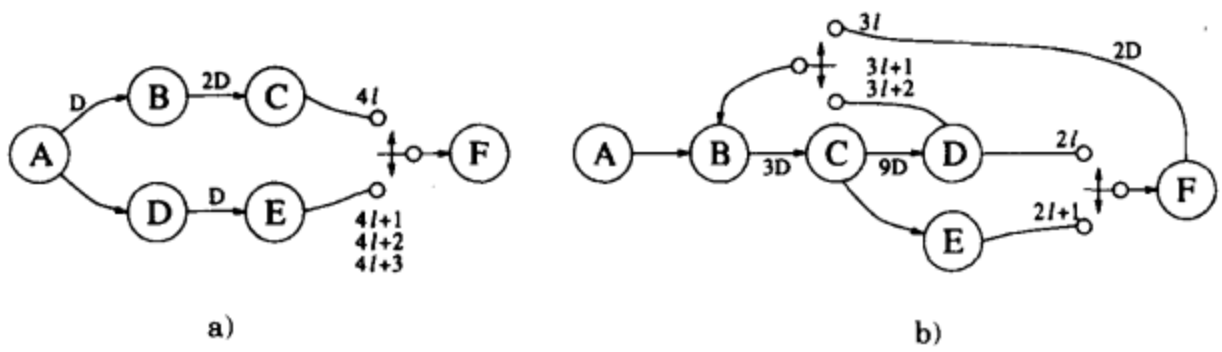


图5-30 习题21的DFG

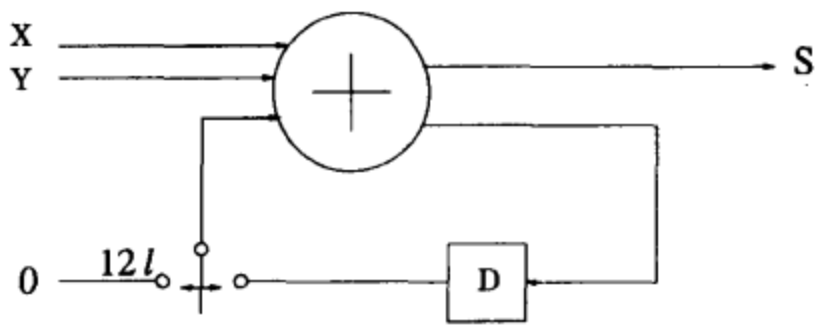


图5-31 习题22的DFG

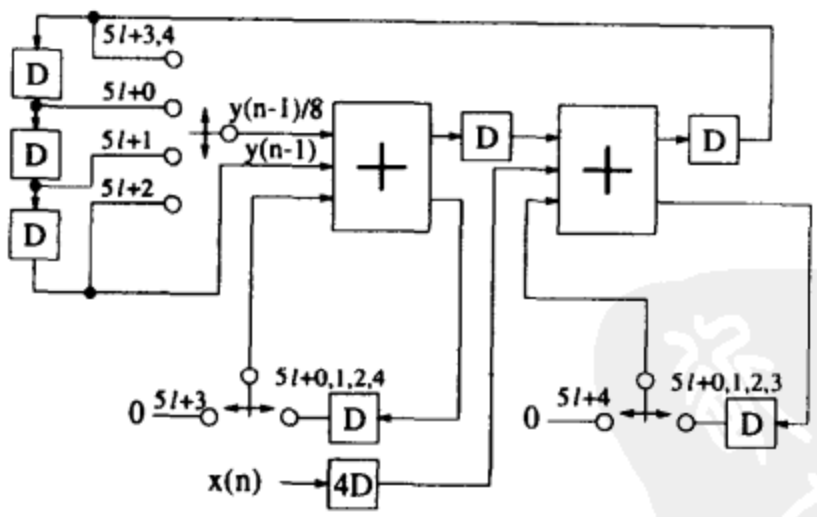


图5-32 习题23的DFG

参考文献

1. A. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools.* Addison-Wesley, 1986.

2. K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. on Computers,*

vol. 40, no. 2, pp. 178–195, Feb. 1991.

3. K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters–part I: pipelining using scattered look-ahead and decomposition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1099–1117, July 1989.
4. K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters–part II: pipelined incremental block filtering," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1118–1135, July 1989.
5. A. P. Chandrakasan, A. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, April 1992.
6. K. K. Parhi, "A systematic approach for design of digit-serial signal processing architectures," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 4, pp. 358–375, April 1991.
7. R. I. Hartley and K. K. Parhi, *Digit-Serial Computation*. Kluwer, 1995.
8. L.-F. Chao and E. Sha, "Retiming and unfolding data-flow graphs," in *Proc. of 1992 International Conference on Parallel Processing, part II*, (St. Charles, IL), pp. 33–40, Aug. 1992.
9. L.-G. Jeng and L.-G. Chen, "Rate-optimal DSP synthesis by pipeline and minimum unfolding," *IEEE Trans. on VLSI Systems*, vol. 2, no. 1, pp. 81–88, March 1994.
10. D.-J. Wang and Y.-H. Hu, "Fully static multiprocessor array realizability criteria for real-time recurrent DSP applications," *IEEE Trans. on Signal Processing*, vol. 42, no. 5, pp. 1288–1292, May 1994.
11. L. E. Lucke and K. K. Parhi, "Data-flow transformations for critical path time reduction in high-level DSP synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1063–1068, July 1993.

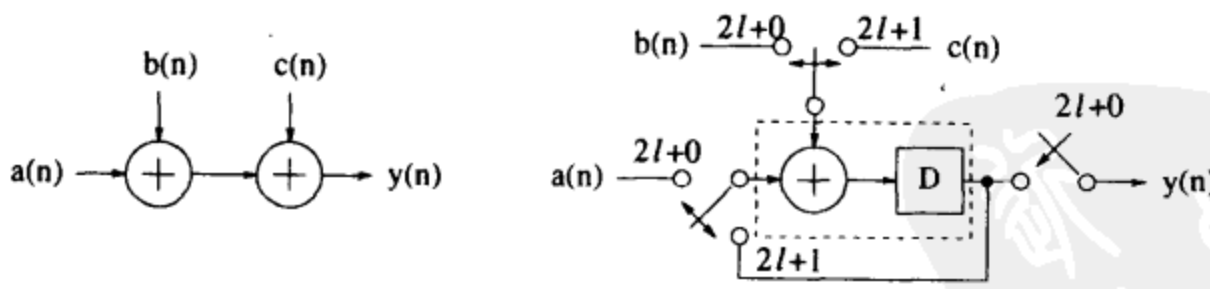
143 ~ 148

第6章 折　叠

6.1 引言

在DSP架构的综合时，通过减少功能单元（例如加法器和乘法器）、寄存器、多路选择器和互连线使得集成电路的硅片面积最小是很重要的。折叠变换用于系统地确定DSP架构中的控制电路，使得单个功能单元（例如流水线加法器）[1][2]能够分时复用于多个算法运算（例如加法运算）。通过在一个功能单元上执行多个算法运算，功能单元的数目减少，使得集成电路能够以较小的硅片面积来实现。虽然折叠技术可用于在单时钟或多时钟下工作的DSP架构的综合，但本章仅涉及单时钟下工作的电路综合。

图6-1给出了两个加法运算可以在一个流水线硬件加法器上分时复用的例子。图6-1a中的DSP程序计算 $y(n) = a(n) + b(n) + c(n)$。图6-1b给出了一个流水线加法器分时复用于图6-1a中的两个加法运算情况。图6-1b中的分时复用硬件的工作如下。在周期0时，样值 $a(0)$ 和 $b(0)$ 输入到加法器，其和 $(a(0) + b(0))$ 存储于延迟元件中直到周期1，这时 $(a(0) + b(0))$ 和 $c(0)$ 一起输入到加法器，其和$(a(0)+b(0)+c(0))$ 存储于延迟元件中直到周期2，然后 $(a(0) + b(0) + c(0))$ 输出，并且加法器计算出中间结果 $(a(1) + b(1))$。这个过程如表6-1所示一直持续。每两个时钟周期产生一个输出样值，同时，每个输入信号样值被采用一次（例如 $a(i)$ 在周期$2i$被采用）。因此，输入样值在变化前必须保持两个时钟周期的时间。通常，总是认为折叠应用的输入数据在变化前可以保持N个周期，其中N是在硬件的一个功能单元上执行算法操作的个数。图6-1b中的结构十分简单，可以用ad hoc技术来设计；然而，通常的DSP程序要比图6-1a中的程序复杂得多。在这种情况下，本章所描述的系统折叠技术就可以用来设计这种分时复用架构。



a) 含有2个加法运算的简单DSP程序

b) 通过一级流水线将2个加法运算折叠为单个硬件加法器的折叠架构

图 6-1

在DSP架构中，折叠技术提供了一种以时间换取面积的方法。实现图6-1a中的DSP程序的一个方法是运用两个硬件加法器，并在两个加法器中间插入一个延迟。这种实现方法需要两个硬件加法器、并在需要执行一次加法运算的时间 T_{add} 里完成该程序的一次迭代运算。另一方面，图6-1b中的折叠实现方法，应用一个硬件加法器，但是需要在 $2T_{add}$ 时间里完成一

次迭代运算。通常，折叠技术可以在增加N倍计算时间的代价下，减少硬件功能单元的数目到$1/N$。有两种极端的情况，一个是采用完全的并行实现方式（即每个算法运算在硬件里都分配有自己的功能单元），另一个是只应用一个处理单元（即整个程序都在一个功能单元上执行）。

表6-1 图6-1b中折叠硬件的前6个周期操作

周期	加法器输入（左部）	加法器输入（顶部）	系统输出
0	$a(0)$	$b(0)$	–
1	$a(0)+b(0)$	$c(0)$	–
2	$a(1)$	$b(1)$	$a(0)+b(0)+c(0)$
3	$a(1)+b(1)$	$c(1)$	–
4	$a(2)$	$b(2)$	$a(1)+b(1)+c(1)$
5	$a(2)+b(2)$	$c(2)$	–

尽管折叠变换可以减少体系结构中的功能单元数目，但是它也可能导致一个采用了大量寄存器的结构。为了避免采用过量的寄存器，需要采用一定的技术，计算出实现折叠DSP所需寄存器的最小数目，并把数据分配给这些寄存器[3]。这些技术将在本章中详细讨论。折叠变换和寄存器最小化一起运用，就能不但减少功能单元的数目，也能保证折叠架构需要的存储器面积最少。

本章从折叠以及折叠的重定时的数学描述开始，见第6.2节。第6.3节讨论了设计最少存储器的DSP架构的通用技术，这些技术在第6.4节中应用于折叠结构。第6.5节讨论了折叠变换系统地应用于多速率DSP程序的结构综合[4]。

6.2 折叠变换

折叠变换提供了一种系统的技术，以设计若干个算法运算分时复用于一个功能单元的硬件控制电路。折叠方程的推导是该技术的基础，它将在本节中和用于折叠的重定时方程的推导包含在一起，后者用于对折叠前的DFG进行重定时。

考虑图6-2a中延迟为 $w(e)$ 的从节点 U 到 V 的边 e。分别调度节点 U 和 V 的 l 次迭代在 $Nl+u$ 和 $Nl+v$ 时刻执行，u 和 v 是节点 U 和 V 的折叠阶数，满足$0\leqslant u$，$v\leqslant N-1$。节点的折叠阶数是节点在硬件中调度执行的时间划分。在节点 U 和 V 执行的功能单元分别表示为 H_U 和 H_V。注意，N 表示折叠到一个功能单元的运算数目，也称为折叠因子。如果 H_U 是 P_U 级流水线结构，那么节点 U 的 l 次迭代结果在 $Nl+u+P_U$ 时刻出现。因为 $U\xrightarrow{e}V$ 边有 $w(e)$ 个延迟，节点 U 的 l 次迭代结果被节点 V 在 $N(l+w(e))+v$ 时刻执行的 $(l+w(e))$ 次迭代运算所使用。因此，中间结果必须存储

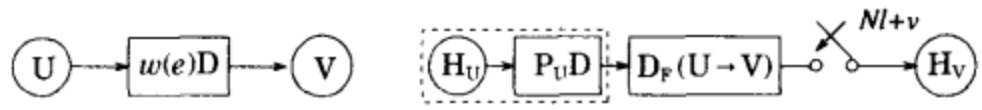
150 ~ 151

$$D_F(U\xrightarrow{e}V)=[N(l+w(e))+v]-[Nl+P_U+u]=Nw(e)-P_U+v-u \tag{6-1}$$

个单位时间，它独立于迭代次数 l。$U\xrightarrow{e}V$ 边实现为结构中从 H_U 到 H_V 的一条带有 $D_F(U\xrightarrow{e}V)$ 延迟的路径，该路径上的数据是 $Nl+v$ 时刻 H_V 的输入，如图6-2b所示。

折叠集是由相同的功能单元所执行运算的有序集。每一个折叠集包含N个单元，其中的一些可能是空运算。折叠集中处于第 j（$0\leqslant j\leqslant N-1$）个位置的运算，由功能单元在第 j 个时间段实行。例如，考虑 $N=3$ 的折叠集 $S_1=\{A_1, 0, A_2\}$。A_1 运算属于折叠集 S_1，其折叠序数为

0（也表示为$(S_1|0)$），A_2 运算属于折叠集 S_1，其折叠序数为2（也表示为$(S_1|2)$）。因为 S_1 中的位置1为空操作，执行 A_1 和 A_2 的运算功能单元在 $3l+1$ 时刻没有利用。



a) 带有 $w(e)$ 延迟的边 $U \xrightarrow{e} V$

b) 相应的折叠数据通道数据已含有 P_U 级流水线的功能单元 H_U 为起点，经过 $D_F(U \xrightarrow{e} V)$延迟，在 N_l+v 时刻由开关输入功能单元 H_V

图 6-2

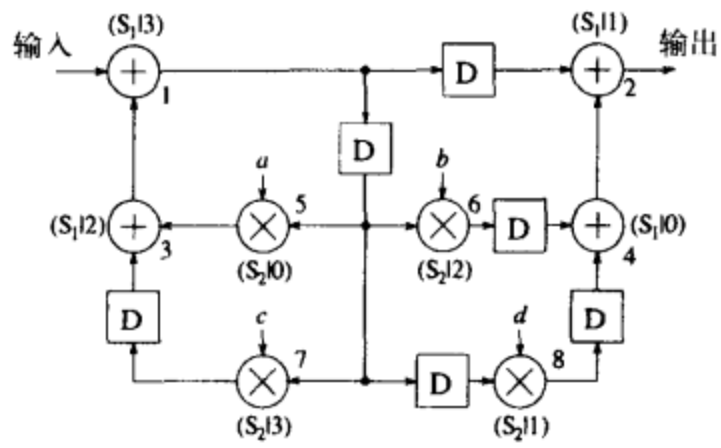


图6-3　指定了合理的折叠集的重定时双2次节滤波器

系统折叠技术的使用可以以折叠图6-3中的重定时后的双2次节滤波器为例。假定加法和乘法分别需要1 u.t.和2 u.t.，并且一级流水线加法器和二级流水线乘法器在此应用（即 $P_A = 1$和 $P_M = 2$）。功能单元可以由1 u.t.周期来定时，该滤波器用图中所示的折叠集，以N=4为折叠因子进行折叠。折叠因子N=4意味着折叠后的硬件的迭代周期为4 u.t.，即在折叠结构中的双2次节滤波器的每个节点恰好每4 u.t.执行一次。这也意味着折叠后的硬件中的一个功能单元要执行DSP程序中的4个运算（节点）。为了表明这一点，图6-3 中的折叠集可以写为 $S_1 = \{4, 2, 3, 1\}$ 和 $S_2 = \{5, 8, 6, 7\}$。折叠集 S_1 只包含加法运算，该折叠集中的节点由同一个硬件加法器执行。类似地，折叠集 S_2 只包含乘法运算，该折叠集中的节点由同一个硬件乘法器执行。例如，折叠结构中的节点 3于$4l+2$ 时刻，在折叠集 S_1 中完成该运算的加法器上执行。折叠结构示于图6-4。这是通过图6-3中的数据流图得到的，方法是给DFG中11条边的每一条，写出如公式（6-1）的折叠方程。这11个方程是

$$
\begin{aligned}
D_F(1 \to 2) &= 4(1) - 1 + 1 - 3 = 1 \\
D_F(1 \to 5) &= 4(1) - 1 + 0 - 3 = 0 \\
D_F(1 \to 6) &= 4(1) - 1 + 2 - 3 = 2 \\
D_F(1 \to 7) &= 4(1) - 1 + 3 - 3 = 3 \\
D_F(1 \to 8) &= 4(2) - 1 + 1 - 3 = 5 \\
D_F(3 \to 1) &= 4(0) - 1 + 3 - 2 = 0 \\
D_F(4 \to 2) &= 4(0) - 1 + 1 - 0 = 0 \\
D_F(5 \to 3) &= 4(0) - 2 + 2 - 0 = 0 \\
D_F(6 \to 4) &= 4(1) - 2 + 0 - 2 = 0 \\
D_F(7 \to 3) &= 4(1) - 2 + 2 - 3 = 1 \\
D_F(8 \to 4) &= 4(1) - 2 + 0 - 1 = 1
\end{aligned}
\tag{6-2}
$$

例如，$D_F(1\rightarrow 8)=5$ 表示在折叠后的DFG中有一条延迟为5，从加法器到乘法器的边。因为这条边终止于节点8，该节点在图6-3中的折叠阶数为1，因此折叠后的边在 $4l+1$ 时刻输入到折叠后的DFG的乘法器中。这条折叠后的边如图6-4所示。

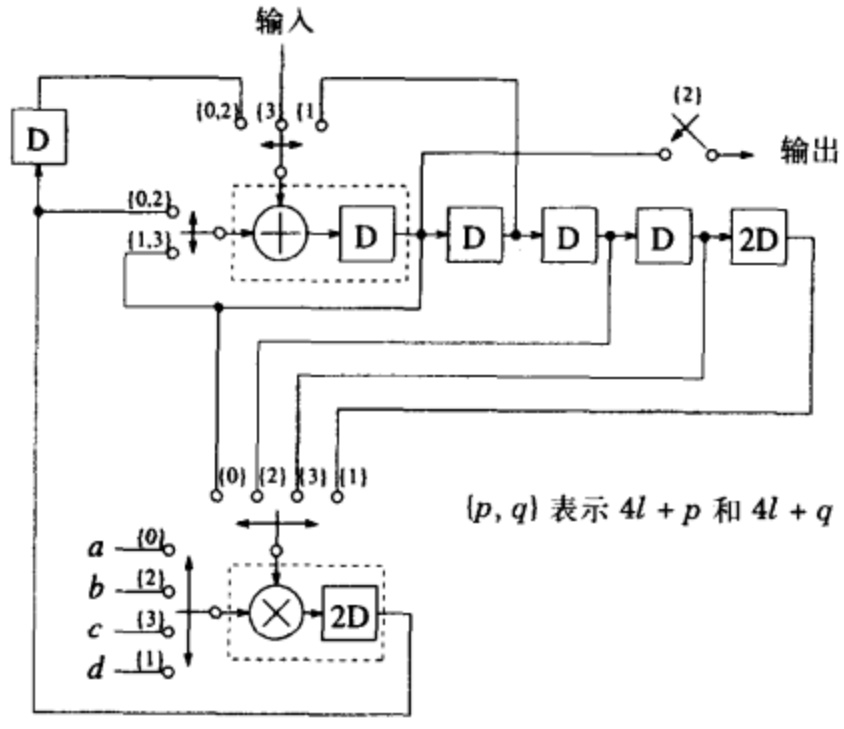


图6-4 用图6-3中的折叠集折叠后的双2次节滤波器

对于一个可实现的折叠系统来说，$D_F(U\xrightarrow{e}V)\geqslant 0$对DFG中的所有边都必须成立。一旦指定了合理的折叠集，就可运用重定时来满足这个属性或者决定该折叠集是否可行。注意到图6-2a中的边 $U\xrightarrow{e}V$ 用公式（6-1）折叠。通过重定时，边 $U\xrightarrow{e}V$ 的延迟数由 $w(e)$ 变为

$$w_r(e)=w(e)+r(V)-r(U) \tag{6-3}$$

其中 $w_r(e)$ 是重定时后的DFG的边 $U\xrightarrow{e}V$ 的延迟数，$r(X)$ 表示节点 X 的重定时之值（见第4章）。令 $D_F'(U\xrightarrow{e}V)$ 表示通过重定时DFG的边 $U\xrightarrow{e}V$ 折叠得到的折叠延迟数目。为了保证折叠后硬件中的相应边具有非负延迟，约束 $D_F'(U\xrightarrow{e}V)\geqslant 0$必须成立，这意味着

$$Nw_r(e)-P_U+v-u\geqslant 0 \tag{6-4}$$

这个约束保证了重定时后的折叠集是合理的。将公式（6-3）和公式（6-4）结合起来，就得到

$$N(w(e)+r(V)-r(U))-P_U+v-u\geqslant 0$$

将 $D_F(U\rightarrow V)$ 代入 $Nw(e)-P_U+v-u$，以求解 $r(U)-r(V)$，得到

$$r(U)-r(V)\leqslant\frac{D_F(U\xrightarrow{e}V)}{N}$$

因为节点的重定时数值限定为整数，该式可重写为

$$r(U)-r(V)\leqslant\left\lfloor\frac{D_F(U\xrightarrow{e}V)}{N}\right\rfloor \tag{6-5}$$

其中 $\lfloor x\rfloor$ 表示 x 的整数部分，即小于或等于 x 的最大整数。

一旦通过公式（6-5）找到DFG的约束集（DFG中的每条边都有一个这样的约束），就可

152 ~ 154

以运用第4.3节描述的方法来判断解是否存在，如果真的存在，找到这个解。例如，考虑图6-5中的双2次节DFG。假定1级流水线加法器和2级流水线乘法器（即 $P_A=1$ 和 $P_M=2$）存在。图6-5a中DFG的折叠方程（见公式（6-1））和折叠约束的重定时（见公式（6-5）），在表6-2中给出。根据第4.3节中解系统不等式的方法，第一步是画出约束图，表6-2的右列中不等式的约束图参见图6-6。如果该约束图包含负环路，那么折叠约束的重定时无解。如果没有负环路，就存在一个解，它是节点9到节点 i 的最短路径。最短路径算法，如Bellman-Ford算法（见附录A的第A.2节）或Floyd-Warshall算法（见附录A的第A.3节），可以用来判定是否存在负环路，并且在不存在负环路的情况下求得一个解。运用这些算法中的任何一个就可发现这个约束集有解，其中一个解是 $r(1)=-1$, $r(2)=0$, $r(3)=-1$, $r(4)=0$, $r(5)=-1$, $r(6)=-1$, $r(7)=-2$ 和 $r(8)=-1$。重定时DFG如图6-3所示，重定时DFG的折叠方程由公式（6-2）给出，折叠DFG如图6-4所示。

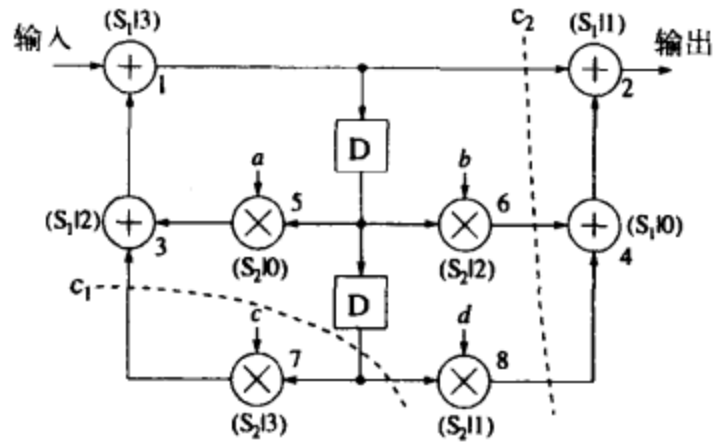


图6-5　原有DFG折叠后带有负的边延迟，图6-3表示折叠后的DFG经过重定时的所有边延迟非负

表6-2　图6-5中DFG的折叠方程和折叠约束的重定时

边	折叠方程	折叠约束的重定时
1→2	$D_F(1\to2)=-3$	$r(1)-r(2)<-1$
1→5	$D_F(1\to5)=0$	$r(1)-r(5)<-0$
1→6	$D_F(1\to6)=2$	$r(1)-r(6)<-0$
1→7	$D_F(1\to7)=7$	$r(1)-r(7)<1$
1→8	$D_F(1\to8)=5$	$r(1)-r(8)<1$
3→1	$D_F(3\to1)=0$	$r(3)-r(1)<0$
4→2	$D_F(4\to2)=0$	$r(4)-r(2)<0$
5→3	$D_F(5\to3)=0$	$r(5)-r(3)<0$
6→4	$D_F(6\to4)=-4$	$r(6)-r(4)<-1$
7→3	$D_F(7\to3)=-3$	$r(7)-r(3)<-1$
8→4	$D_F(8\to4)=-3$	$r(8)-r(4)<-1$

对诸如双2次节滤波器这样的简单DFG的折叠重定时，也可通过观察进行。由表6-2，带负延迟 $D_F(U\to V)$ 的边为 1→2, 6→4, 8→4 和 7→3。请注意，在边 $U\to V$ 上增加（减去）w 延迟，就会相应地增加（减少）Nw 于 $D_F(U\to V)$。这样，$D_F(1\to2)$, $D_F(6\to4)$, $D_F(8\to4)$ 和 $D_F(7\to3)$ 可以通过在每条这样的边上增加一个延迟，成为非负的边。这可以通过对图6-5中标为 $c1$ 和 $c2$ 的割集进行割集重定时得到。读者可以验证通过这种方式得到的重定时DFG与图6-3中的DFG完全相同。

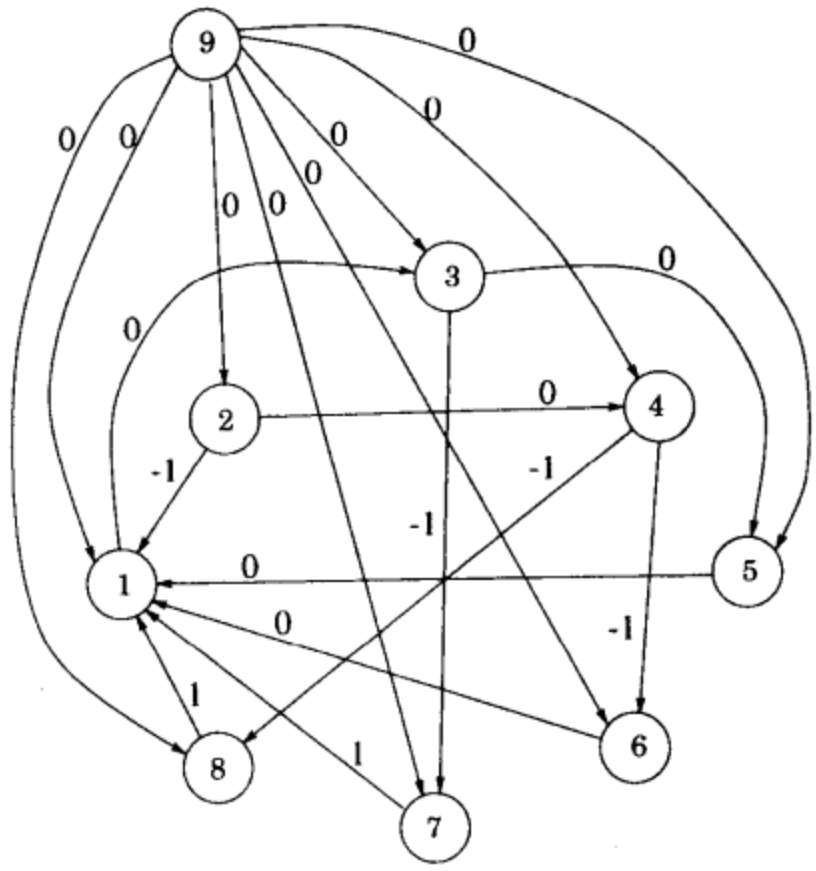


图6-6 表6-2右列中的不等式集表示的约束图，节点9为主节点

155 ~ 156

注意到用展开因子4展开图6-4中的折叠电路，就会得到图6-3中的DFG，它是图6-5中的DFG的重定时。通常地，原有DFG和折叠DFG（由折叠因子N综合得到）的N阶展开DFG彼此互为重定时与/或流水线结构。并且，任何一个DFG都能以因子N展开，展开后的DFG可以通过很多可能的折叠集折叠产生一系列的结构。一个适当的折叠集可以用来得到原有的DFG。

在硬件综合时，折叠可用于功能单元的连接绑定。折叠集通常由调度分配算法得到（见附录B）。

6.3 寄存器最小化技术

本节重点讨论如何使用于DSP架构中的寄存器数目最少，使寄存器占有的硅面积较小。这里给出了用于计算最少寄存器数以及将数据分配给这些寄存器的方法。第6.4节讨论如何将这些技术用于设计尽可能少的寄存器数的折叠架构。

6.3.1 寿命分析

寿命分析是计算用硬件实现DSP算法所需的最少寄存器数的过程。一个样值（也称为变量）从产生的时刻到被应用的时刻是激活的。变量被应用后，就消亡了。变量在它激活的每个单位时间里都要占据一个寄存器。在寿命分析中，算出每个单位时间里的激活变量数，就可确定任意单位时间里的最大激活变量数[5]。这即是实现DSP程序所需的最小寄存器数。

例如，考察一个生成3个变量 a、b 和 c 的DSP程序。变量 a 在 $n \in \{1, 2, 3, 4\}$ 单位时间里激活，变量 b 在 $n \in \{2, 3, 4, 5, 6, 7\}$ 单位时间里激活，变量 c 在 $n \in \{5, 6, 7\}$ 单位时间里激活。假设之前和之后的程序迭代的变量寿命不和变量 a、b 或 c 的寿命交叉，那么在单位时间1、2、3、4、5、6、7里的激活变量数分别是1、2、2、2、2、2、2。实现DSP程序所需的最小寄存器数，即是任意时刻的最大激活变量数，即 $\{1, 2, 2, 2, 2, 2, 2\} = 2$。本节将讨

论寿命分析的图形技术。

线性寿命图　用于图形化表示线性模式（相对应的是循环模式）下的每个变量的寿命。图6-7a就是一个线性寿命图的例子。水平线表示时钟周期（或单位时间），垂直线表示变量的寿命。请注意，寿命图中的变量在它所产生的时钟周期里是非激活的，而在它被应用的时钟周期里是激活的。例如，变量 a 在周期0产生，在周期4被应用，则该变量在周期1、2、3、4里是激活的。寿命图右侧给出了每个周期里的激活变量数。任意时刻的最大激活变量数是2，所以实现给定DSP程序的最小寄存器数是2。 157

通常，DSP程序是周期性的。给定DSP程序的一次迭代的变量 a、b 和 c 的寿命有可能会与其他迭代中的同名变量的寿命重叠。例如，令迭代周期 N=6。a_i、b_i、c_i 表示DSP程序的 i 次迭代的变量 a、b、c。图6-7b给出了DSP程序的3次连续迭代的寿命图。a_1 的寿命与 b_0 和 c_0 的寿命重叠，因此考虑到DSP程序的周期性，该DSP程序的实现需要3个寄存器。在图6-7a中没有考虑DSP程序的周期性时只需要2个寄存器。因此，DSP程序的周期性在寿命分析时必须考虑。幸运的是，在考虑周期性时，无需画出几次迭代的寿命。可以画出0次迭代的变量寿命，令在 $n>N$ 时刻的激活变量数为0次迭代在周期 $n-kN$ 时的激活变量数之和，其中 k 为大于等于0的所有整数。如图6-7c所示，周期7的激活变量数为0次迭代在周期7和周期1的激活变量数之和，即2＋1＝3。

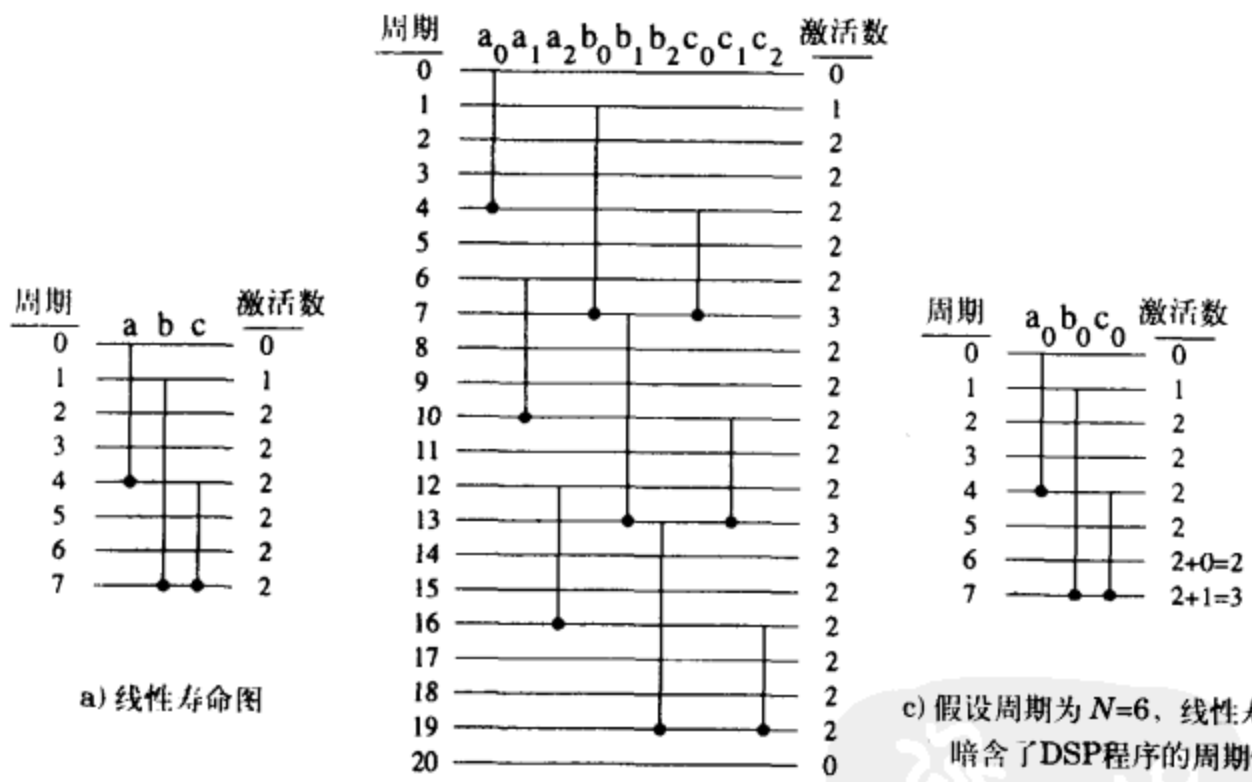


图　6-7

通常，寿命分析由建立如表6-3所示的寿命表开始。该表表示3×3矩阵的转置运算，例如

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

矩阵由行排序转为列排序。表6-3中的每个变量都有一个输入时间 T_{input}，一个0延时输出时间 T_{zlout}，即忽略系统延时的变量的输出时间。0延时系统的每个变量在 $T_{dif}f = T_{zlout} - T_{input}$ 个

时钟周期里是激活的；然而，如果寿命期是负的，就违反了因果性。例如，变量 d 在 $T_{input}=3$ 时输入，在 $T_{zlout}=1$ 时输出，则它在 $1-3=-2$ 个单位时间里是激活的，而这是不可能的。为使系统满足因果性，需要给系统添加延时。延时 T_{lat} 是 T_{diff} 最大负值的幅度，在表6-3中 $T_{lat}=-(-4)=4$。变量的实际输出时间是 $T_{output}=T_{zlout}+T_{lat}$，变量的寿命期是 $(T_{input}\rightarrow T_{output})$。运用寿命期就可画出寿命图。表6-3的寿命图如图6-8所示。该DSP程序的周期为 N=9。

表6-3 3×3矩阵转置运算的寿命

变量	T_{input}	T_{zlout}	T_{diff}	T_{output}	寿 命 $(T_{input}\rightarrow T_{output})$
a	0	0	0	4	$0\rightarrow 4$
b	1	3	2	7	$1\rightarrow 7$
c	2	6	4	10	$2\rightarrow 10$
d	3	1	−2	5	$3\rightarrow 5$
e	4	4	0	8	$4\rightarrow 8$
f	5	7	2	11	$5\rightarrow 11$
g	6	2	−4	6	$6\rightarrow 6$
h	7	5	−2	9	$7\rightarrow 9$
i	8	8	0	12	$8\rightarrow 12$

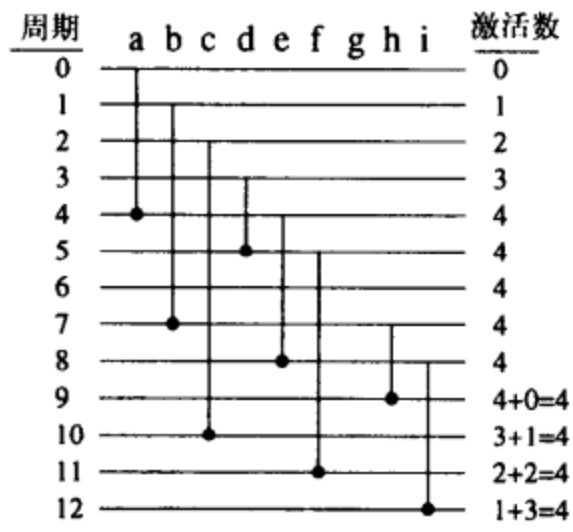


图6-8 3×3矩阵转置的周期为 N=9 的线性寿命图，变量的寿命周期如表6-3所示

另一种用寿命图表示DSP程序的周期行为的方法是循环寿命图。在周期为N的循环寿命图中，点 i（i 是满足 $0<i<N-1$ 的整数）表示时间块 i 和所有的 $\{(Nl+i)\}$ 时刻，l 为非负整数。例如，N=8 时，时间块 i=3 表示 {3, 11, 19, ...} 时刻。与图6-8的线性寿命图相对应的循环寿命图如图6-9所示。括号里的数字表示每个时间块的激活变量数。请注意，在单位时间 j 产生并于单位时间 k 被应用的变量，在单位时间 j+1 到 k 是激活的，因为变量在它产生的单位时间里是不需要存储的。

6.3.2 采用前向-后向寄存器分配的数据分配技术

一旦确定了实现DSP程序所需的最小寄存器数，就需要将数据分配给这些寄存器。前向-后向寄存器分配是将数据分配给最小寄存器数的分配方案[6]。本节描述了前向-后向寄存器分配，并用之产生实现DSP程序所需的最小寄存器架构，也可以运用其他分配方案[7][8]。

寄存器分配可以通过分配表来完成。分配方案给出了变量是如何指派给分配表中的寄存

器的。图6-10b给出了用前向-后向方案给一个3 × 3矩阵转置分配数据的分配表。进行前向-后向寄存器分配的步骤如下。

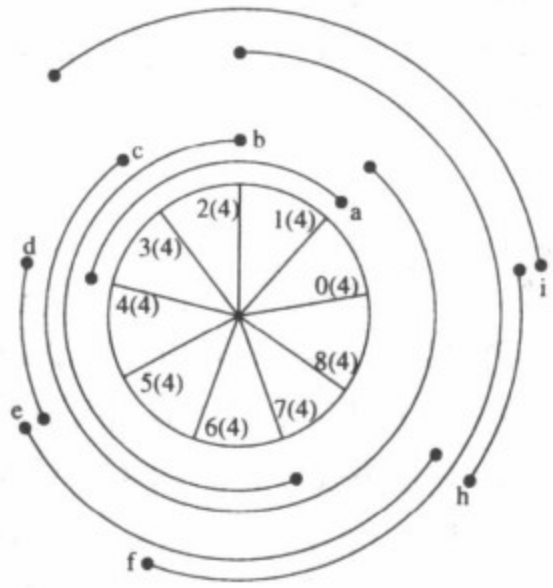


图6-9　3 × 3矩阵转置的环形寿命图相应的线性寿命图如图6-8所示，变量的寿命周期如表6-3所示

cycle	input	R1	R2	R3	R4	output
0	a					
1	b	a				
2	c	b	a			
3	d	c	b	a		
4	e	d	c	b	a	a
5	f	e	d	c	b	d
6	g	f	e		c	g
7	h		f	e		
8	i	h		f	e	e
9		i	h		f	h
10			i			
11				i		
12					i	i

a) 3 × 3矩阵转置经过1到4步前向-后向寄存器分配后的分配表

cycle	input	R1	R2	R3	R4	output
0	a					
1	b	a				
2	c	b	a			
3	d	c	b	a		
4	e	d	c	b	a	a
5	f	e	d	c	b	d
6	g	f	e	b	c	g
7	h	c	f	e	b	b
8	i	h	c	f	e	e
9		i	h	c	f	h
10			i	f	c	c
11				i	f	f
12					i	i

b) 分配完成后的分配表

图 6-10

158 ~ 161

步骤1：通过寿命分析确定最小寄存器数。

步骤2：在寿命期开始所对应的时间步输入每个变量。如果给定周期里有多个变量输入，这些变量就分配给多个寄存器，使得寿命最长的变量分配给初始寄存器，其他变量按寿命递减逐个分配给寄存器。

步骤3：每个变量都以这种前向方式分配，直至变量消亡或者到达末尾寄存器。在前向分配中，如果寄存器 i 在当前周期里存有某变量，则寄存器 $i+1$ 在下一个周期存有相同的变量。如果寄存器 $i+1$ 不可用，则变量分配给第一个可用的前向寄存器。

步骤4：因为分配是周期性的，当前迭代的分配在接下来的迭代中还会重复。因此，如果 R_j 在周期 l 被某个变量占据，则 R_j 会在周期 $l+N$ 占有相应的变量，N 表示分配的周期。因此，对任意 j 和 l，R_j 在单位时间 $l+N$ 时发生重叠。

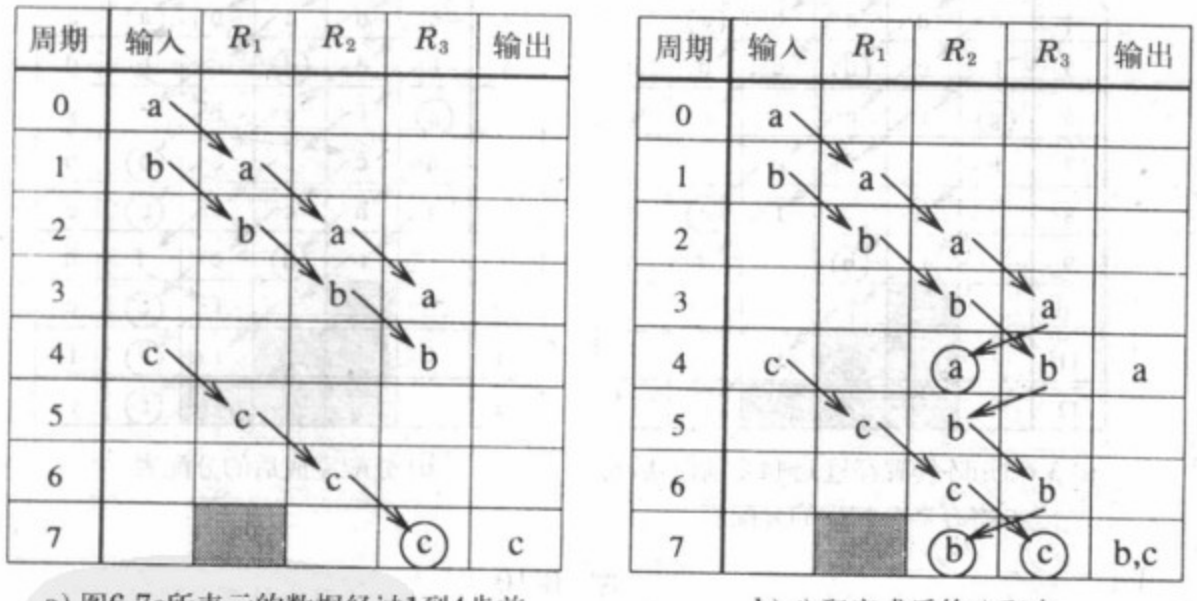
步骤5：对到达末尾寄存器但仍没有消亡的变量，可以计算出它的剩余寿命，然后基于先到先得的原则，将这些变量以后向的方式分配给寄存器。如果后向分配中有多个寄存器可用，首先尝试选择在末尾寄存器和该寄存器之间已进行过后向分配的寄存器。如果有不止一个寄存器符合后向分配，在候选寄存器中，选择有最少但足够多的前向寄存器来完成变量分配的那个寄存器。一个变量经过后向分配后，再进行前向分配，直至消亡或再次到达末尾寄存器。

步骤6：按要求重复步骤4和5直至分配完成。

图6-10给出了3×3矩阵转置的前向-后向分配表，表6-3给出了它的寿命表。步骤1在第6.3.1节中已进行，确定3×3矩阵转置需要4个寄存器（见图6-8和6-9）。步骤2、3、4如图6-10a所示。步骤2中每个变量都在它寿命开始所对应的周期里输入。本例中没有一个周期里有超过一个输入变量。步骤3中变量都用前向方式分配。步骤4中的重叠避免了和后向分配的冲突。例如，寄存器R_2在周期11混叠，因为该寄存器在周期11-9=2时被变量 a 占有。步骤5如图6-10b所示。本步骤中，变量进行后向分配。例如，变量 b 被后向分配给R_3，因为这是惟一可用于后向分配的寄存器。在变量 b、c、f 被后向分配后，再进行前向分配直至它们的消亡，至此分配完成。

162

作为前向-后向分配的另一个例子，图6-7c中的数据的分配表如图6-11所示。图6-11a给出了前向-后向寄存器分配的步骤1到步骤4之后的分配表，图6-11b给出了分配完成之后的分配表。



a) 图6-7c所表示的数据经过1到4步前向-后向寄存器分配后的分配表

b) 分配完成后的分配表

图 6-11

通过分配表可以综合出存储器单元架构，3×3矩阵转置架构如图6-12所示。这个架构完全由图6-10b中的分配表而来。例如，图6-10b中的变量 b 在周期5时从R_4到R_3的后向分配，在图6-12中是用在$9l+5$时刻闭合的、从R_4输出到R_3输入的反馈路径来实现的。该架构中变量 f 的反馈路径的闭合时刻为$9l+0$，因为该反馈发生在图6-10b中的周期9，所有反馈路径的闭合时刻都有$9l+m$形式，$0 \leqslant m \leqslant 8$。

另外一个例子是图6-13给出的与图6-11b中的分配表相对应的架构。

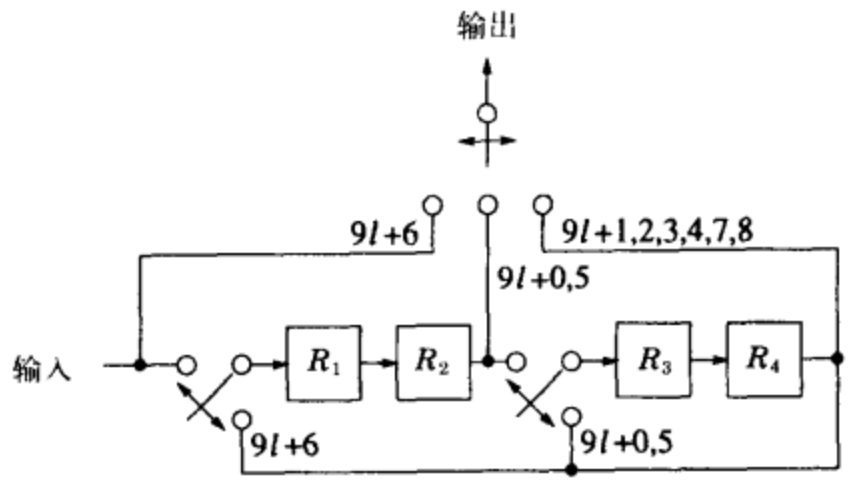


图6-12 与图6-10b中的分配表相对应的3 × 3矩阵转置的架构

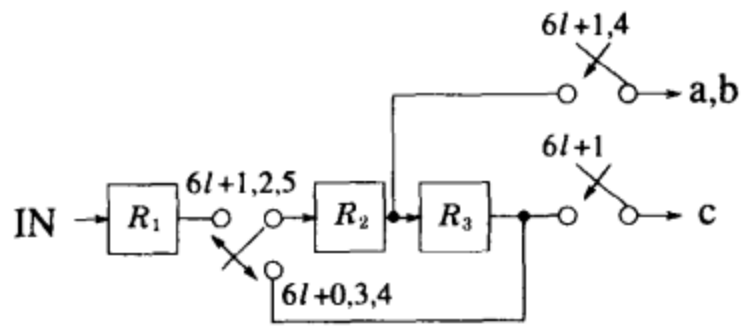


图6-13 与图6-11b中的分配表相对应的架构

6.4 折叠架构的寄存器最小化

第6.3节描述的寄存器最小化技术，可用于综合折叠架构中的控制电路，使得所需寄存器尽可能少。基本过程如下：

（1）进行折叠的重定时（见第6.2节）。

（2）写出折叠方程（见第6.2节）。

（3）用折叠方程构造寿命表。

（4）画出寿命图并确定所需寄存器数（见第6.3.1节）。

（5）进行前向-后向寄存器分配（见第6.3.2节）。

（6）画出最小寄存器数的折叠架构。

本节用2个例子来说明该过程。

6.4.1 双2次节滤波器例子

本例为图6-5中的DFG构造了一个最小寄存器架构。经过重定时和折叠之后的DFG如图6-3所示，该DFG的折叠方程由公式（6-2）给出。未经过寄存器最小化的折叠架构如图6-4所示。该架构用了6个寄存器（未计加法器和乘法器内部的3个流水线寄存器）。 163~164

因为折叠的重定时已完成并且折叠方程已由公式（6-2）写出，下一步便是构建如表6-4所示的寿命表。寿命表对DFG的每一个节点的寿命（$T_{input} \rightarrow T_{output}$）都有相应项说明。节点$U$的 T_{input} 为 $u+P_U$，其中 u 是 U 的折叠阶数，P_U 是执行 U 的功能单元的流水线级数。T_{input} 是硬件为DSP程序的0次迭代产生数据的时间。例如，节点1的 T_{input} 为3+1=4。节点U

的 T_{output} 为 $u + P_U + \max_V \{(D_F(U \to V)\}$，$\{D_F(U \to V)\}$ 表示所有由节点 U 开始的边中最长的折叠路径延迟。T_{output} 的值为该节点0次迭代之值被使用的最迟时间。例如，节点1的 T_{output} 为 $3 + 1 + \max\{1, 0, 2, 3, 5\} = 3 + 1 + 5 = 9$。请注意，在该寿命表中不必给输出时间增加延迟，因为折叠的重定时步骤已经保证了系统的因果性。

表6-4 图6-3所示的重定时双2次节滤波器的寿命

节点	$T_{input} \to T_{output}$
1	$4 \to 9$
2	–
3	$3 \to 3$
4	$1 \to 1$
5	$2 \to 2$
6	$4 \to 4$
7	$5 \to 6$
8	$3 \to 4$

图6-14是表6-4所示的寿命表对应的线性寿命图。N=4 是运算周期。由寿命图可知，折叠结构需要2个寄存器。接下来进行前向-后向寄存器分配。分配表如图6-15所示。表中变量 n_i 表示节点 i 的输出。表中只给出了3个变量 n_1、n_7 和 n_8，因为只有这些变量的寿命不为0。表中显示变量 n_1 在周期9输出。请注意，该变量也在周期4、5、6、8时输出。为表述清楚，表中只给出了每个变量的最迟输出时间。

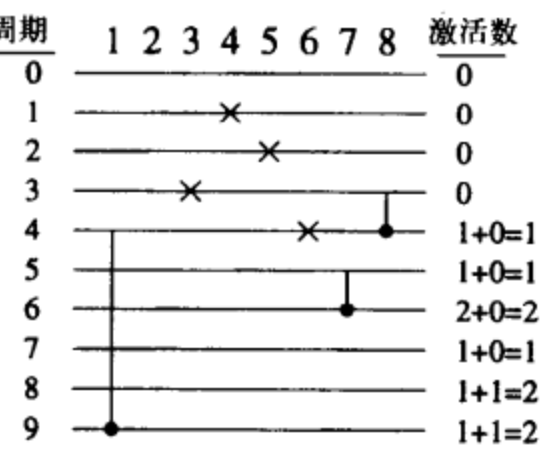


图6-14 与表6-4中的寿命表相对应的寿命图

通过图6-15的分配表和公式（6-2）所示的折叠方程，可以最后综合出图6-16中的架构。例如，折叠架构中的边1→2有 $D_F(1 \to 2) = 1$ 延迟。该边始于节点1，经过一个延迟后，变量 n_1 位于图6-15中的寄存器R_1，因此存在一条从R_1到折叠架构中的加法器的边，该边在$4l$+1时刻与加法器导通，因为节点2的折叠阶数为1。另一个例子是，边1→7有 $D_F(1 \to 7) = 3$ 延迟，变量 n_1 在2个延迟后位于R_2，因此存在一条从R_2到折叠架构中的乘法器的边。该边在$4l$+3时刻与乘法器导通，因为节点7的折叠阶数为3。本例中折叠架构中的寄存器数从6（见图6-4）减少为2（见图6-16）。

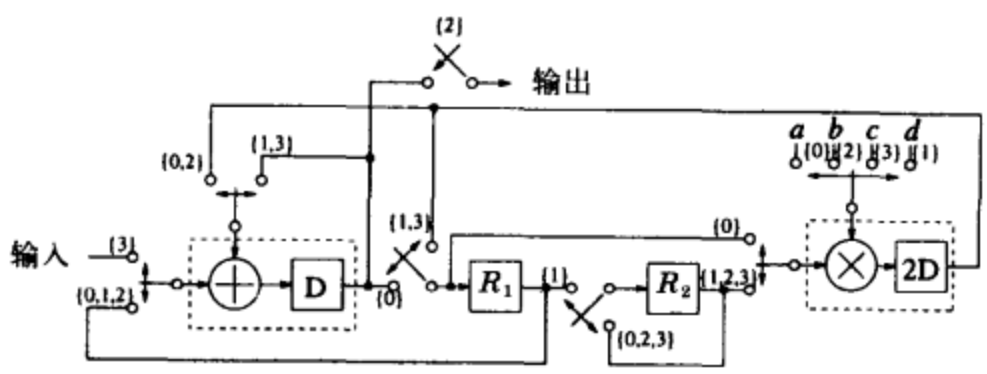
6.4.2 IIR滤波器例子

本例中，用来计算 $y(n) = ay(n-3) + by(n-5) + x(n)$ 的图6-17a中的IIR滤波器是折叠架构。逐一执行第6.4节开头所描述的所有6个步骤，从而找出使用最小寄存器数的折叠架构。

第一步，进行折叠的重定时。表6-5给出了折叠方程和折叠重定时不等式。运用第4.3节的技术，求解折叠重定时不等式，可得重定时方案为 $r(1)=0$, $r(2)=0$, $r(3)=-2$ 和 $r(4)=-1$。图6-17b给出了重定时后的DFG。

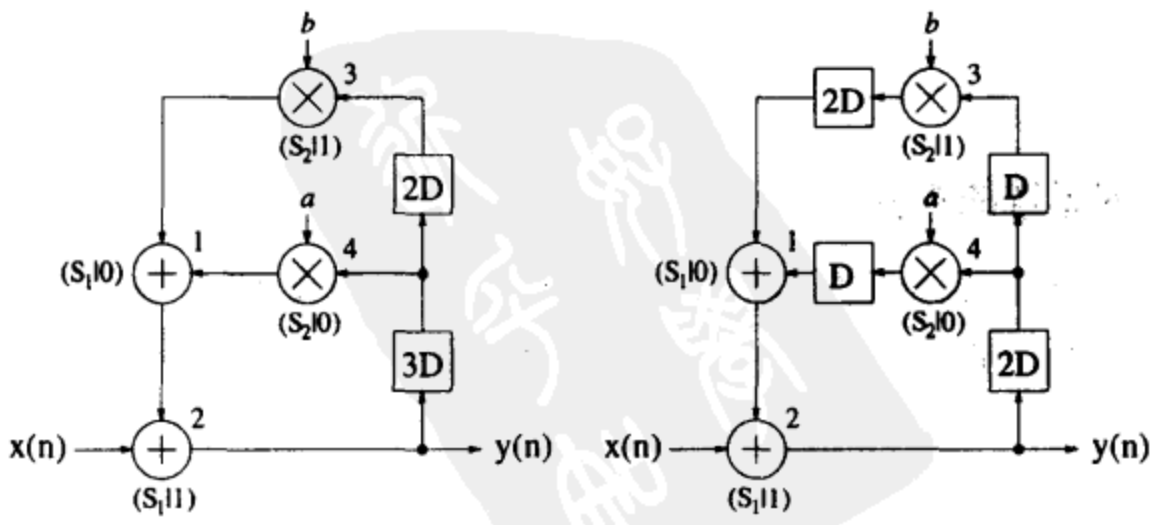
周期	输入	R_1	R_2	输出
0				
1				
2				
3	n_8			
4	n_1	n_8		n_8
5	n_7	n_1		
6		n_7	n_{11}	n_7
7			n_{11}	
8			n_{11}	
9			n_1	n_1

图6-15　折叠双2次节滤波器的分配表



$\{p,q\}$表示$4l+p$和$4l+q$

图6-16　用最少寄存器数，即2个实现图6-3中DFG的折叠双2次节滤波器



a) 计算 $y(n)=ay(n-3)+by(n-5)+x(n)$ 的IIR滤波器

b) 经过重定时的折叠边延迟为非负的滤波器

图　6-17

表6-5 图6-17a中DFG的折叠方程和它的重定时

边	折叠方程	折叠方程的重定时
$1 \to 2$	$D_F(1 \to 2) = 0$	$r(1) - r(2) \leqslant 0$
$2 \to 3$	$D_F(2 \to 3) = 9$	$r(2) - r(3) \leqslant 4$
$2 \to 4$	$D_F(2 \to 4) = 4$	$r(2) - r(4) \leqslant 2$
$3 \to 1$	$D_F(3 \to 1) = -3$	$r(3) - r(1) \leqslant -2$
$4 \to 1$	$D_F(4 \to 1) = -2$	$r(4) - r(1) \leqslant -1$

图6-17b中的重定时DFG的折叠方程为

$$\begin{aligned} D_F(1 \to 2) &= 2(0) - 1 + 1 - 0 = 0 \\ D_F(2 \to 3) &= 2(3) - 1 + 1 - 1 = 5 \\ D_F(2 \to 4) &= 2(2) - 1 + 0 - 1 = 2 \\ D_F(3 \to 1) &= 2(2) - 2 + 0 - 1 = 1 \\ D_F(4 \to 1) &= 2(1) - 2 + 0 - 0 = 0 \end{aligned} \tag{6-6}$$

表6-6 公式（6-6）中方程的寿命图

节点	$T_{input} \to T_{output}$
1	$1 \to 1$
2	$2 \to 7$
3	$3 \to 4$
4	$2 \to 2$

表6-6给出了这些方程的寿命表，图6-18给出了寿命图，图6-19a给出了分配表。由分配表和公式（6-6）中的折叠方程，可以综合出图6-19b中的架构。与不经过寄存器最少化的简单实现需要6个寄存器相比，该架构只需要3个寄存器（不包括流水线寄存器）。

165 ~ 168

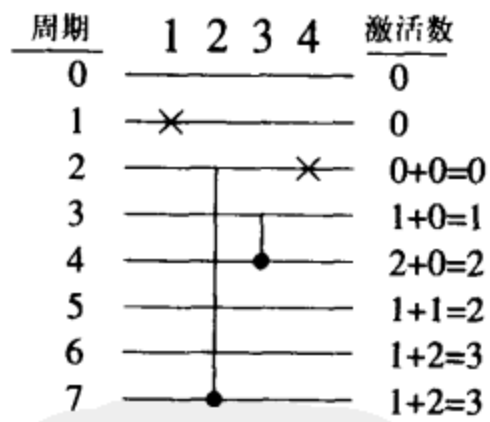


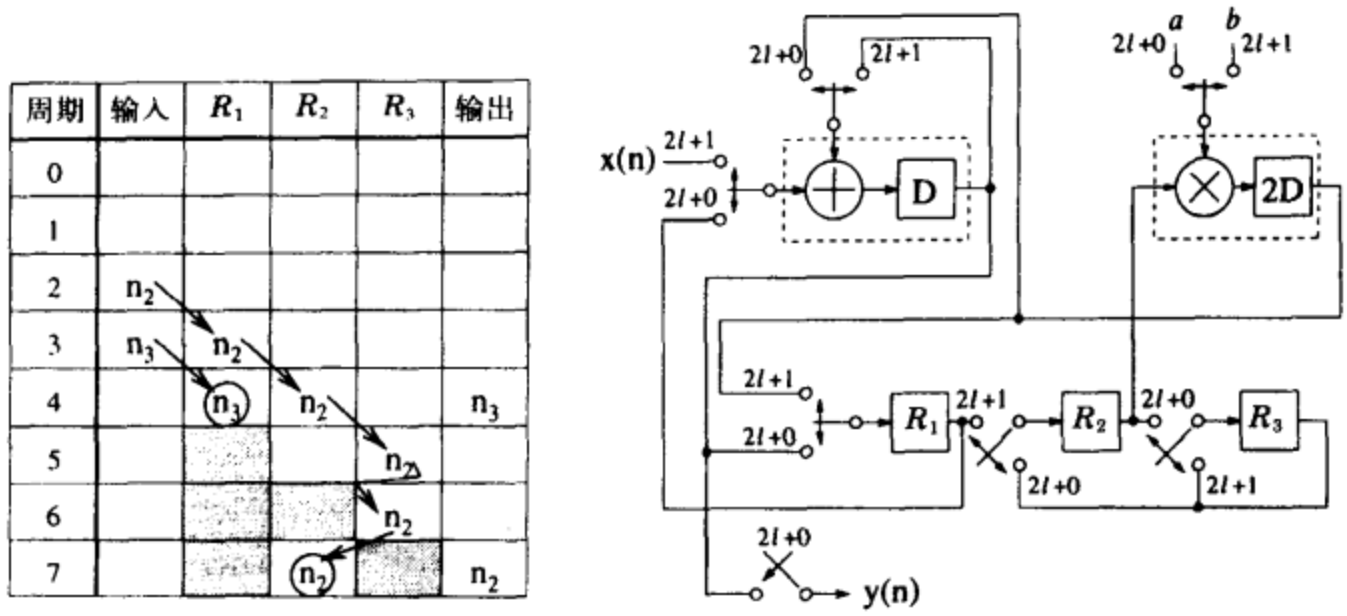
图6-18 与表6-6相对应的寿命图

6.5 多速率系统的折叠

多速率DSP算法描述包含了抽取器和/或扩展器（见第1.2.9节）。图6-20给出了一个抽取器和一个扩展器。抽取器遵循输入输出关系 $y_D(n) = x(Mn)$，扩展器遵循输入输出关系

$$y_E(n) = \begin{cases} x(\frac{n}{M}) & \text{如果 } n \text{ 是 } M \text{ 的倍数} \\ 0 & \text{否则} \end{cases}$$

抽取器从M个样值中抽出$M-1$个，而扩展器在2个非0样值中插入$M-1$个0样值。抽取器和扩展器都有改变抽样速率的作用。



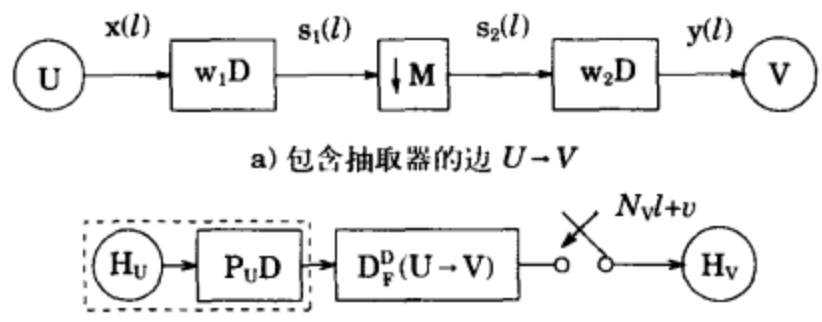
a) IIR滤波器例子的分配表　　b) IIR滤波器例子的最小寄存器架构

图 6-19

a) M 倍抽取　　b) M 倍扩展

图 6-20

公式（6-1）给出了不含抽取器或扩展器的边的折叠方程。也可推出含有抽取器或扩展器的边的折叠方程[9]。本节推出了包含抽取器的边的方程；边包含扩展器的情况留作练习（见习题22）。



a) 包含抽取器的边 $U\to V$

b) 折叠后的边

图 6-21

考虑图6-21(a)中的边 $U\to V$，节点 U 的输出经过 w_1 延迟，M 倍抽取，w_2 延迟，到达节点 V。令节点 U 的l次迭代在单位时间 N_Ul+u 时执行，节点 V 的l次迭代在单位时间 N_Vl+v 时执行，折叠阶数 u 和 v 满足 $u\in[0,\ N_U)$ 和 $v\in[0,\ N_V)$。图6-21a中所标记的信号有如下关系 169 ~ 170

$$
\begin{aligned}
s_1(l) &= x(l-w_1)\\
s_2(l) &= s_1(Ml)=x(Ml-w_1)\\
y(l) &= s_2(l-w_2)=x(M(l-w_2)-w_1)
\end{aligned}
$$

它表明，被节点 V 在l次迭代中使用的样值 $y(l)$，产生于节点 U 的 $(Ml-(Mw_2+w_1))$ 次迭代。样值 $y(l)$ 在 N_Vl+v 时刻被 H_V 使用，在 $N_U(Ml-(Mw_2+w_1))+u$ 时刻由 H_U 算出。如果 H_U 是 P_U 级流水线，则 $y(l)$ 在 $N_U(Ml-(Mw_2+w_1))+u+P_U$ 时刻是可用的。因此，$y(l)$ 必

须存储

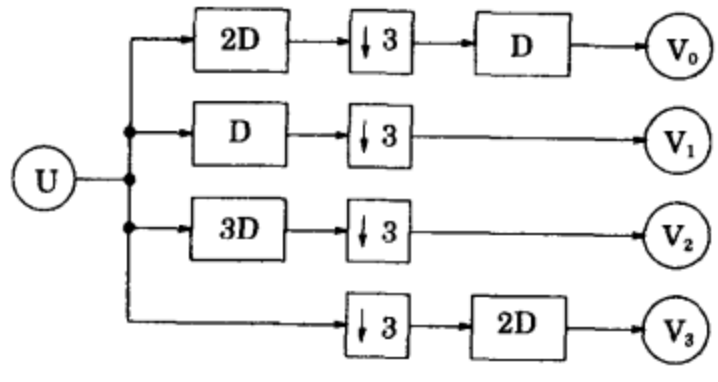
$$\begin{aligned} D_F^D(U \to V) &= [N_V l + v] - [N_U(Ml - (Mw_2 + w_1)) + u + P_U] \\ &= (N_V - MN_U)l + N_U(Mw_2 + w_1) - P_U + v - u \end{aligned}$$

个单位时间。和单速率情形一样，该表达式应该与迭代次数 l 无关。这可以通过令 $N_V = MN_U$ 得到，它意味着节点 V 每执行一次，节点 U 要执行 M 次。这与我们的直觉吻合，因为节点 U 的输出在到达节点 V 之前要进行 M 倍抽取。有了 $N_V = MN_U$，折叠方程变为

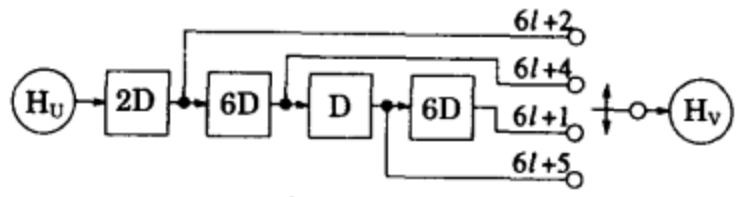
$$D_F^D(U \to V) = N_U(Mw_2 + w_1) - P_U + v - u \tag{6-7}$$

与迭代次数 l 无关。

因为节点 V 在 $Nvl + v$ 时刻调度执行硬件运算符 H_V，如图6-21b所示，边 $U \to V$ 上的数据在 $Nvl + v$ 时刻输入到 H_V。例如，M=1 的情形，即抽取器不影响数据流，w_1 和 w_2 可合并为 $w = w_1 + w_2$，$N_U = N_V = N$，N 是节点 U 和 V 的迭代周期，将这些式子代入公式（6-7），就得到了和公式（6-1）一样的单速率折叠方程。



a) 边含有抽取器的DFG



b) 折叠架构

图 6-22

对于折叠边包含抽取器的例子，考虑图6-22a中的DFG。节点的折叠因子或迭代周期为 $N_U = 2$ 和 $N_{V0} = N_{V1} = N_{V2} = N_{V3} = 6$。折叠阶数为 $u = 1$，$v_0 = 1$，$v_1 = 2$，$v_2 = 4$，$v_3 = 5$。节点 U 指定给一级流水线处理器，即 $P_U = 1$。折叠方程为

$$\begin{aligned} D_F^D(U \xrightarrow{e_0} V_0) &= 2(3(1) + 2) - 1 + 1 - 1 = 9 \\ D_F^D(U \xrightarrow{e_1} V_1) &= 2(3(0) + 1) - 1 + 2 - 1 = 2 \\ D_F^D(U \xrightarrow{e_3} V_2) &= 2(3(0) + 3) - 1 + 4 - 1 = 8 \\ D_F^D(U \xrightarrow{e_3} V_3) &= 2(3(2) + 0) - 1 + 5 - 1 = 15 \end{aligned}$$

如果4个节点 Vi，i=1，2，3，4，都由相同的功能单元 H_V 执行，则折叠结构如图6-22b所示。寄存器数可以通过寿命分析减少。

为使公式（6-7）有用，D_F^D（$U \to V$）≥ 0在给定的可行的调度下必须成立。折叠的重定时可以以和单速率DFG（SRDFGs）相同的方式用于多速率DFG（MRDFGs）。在得出包含抽

取器的边的折叠重定时的约束之前，先回顾一下多速率的基本属性。

惰性恒等变换在多速率DSP的理论和实现中十分有用[4]。这些恒等变换的特例如图6-23所示。这些关系用来推得重定时折叠MRDFG的条件。

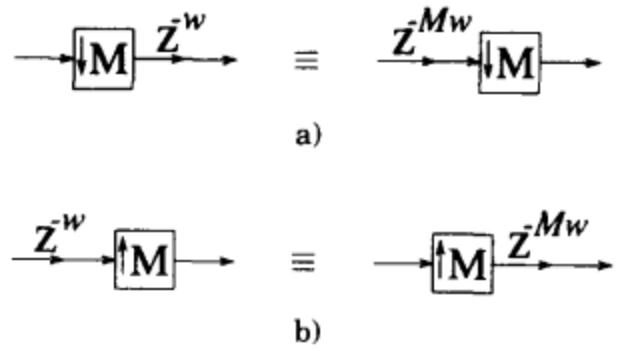


图6-23　利用恒等量在多速率系统中进行延迟的重分布

171 ~ 172

DFG可以通过重定时满足条件 $D_F^D\ (U \to V) \geqslant 0$。令 w_1' 和 w_2' 为边 $U \to V$ 重定时后的延迟。运用公式（6-7），可得折叠边重定时后的延迟为

$$D_F'^D(U \to V) = N_U(Mw_2' + w_1') - P_U + v - u$$

w_1' 和 w_2' 与 w_1 和 w_2 的关系为

$$w_1' = w_1 + Mr(D_{uv}) - r(U)$$

和

$$w_2' = w_2 + r(V) - r(D_{uv})$$

其中 $r(u)$ 和 $r(v)$ 分别是节点 U 和 V 的重定时数值，即从节点的每个输出边减少一个延迟和从节点的每个输入边增加一个延迟的次数。根据图6-23a的惰性恒等变换，抽取器的重定时数值，$r(D_{uv})$，是输出减少一个延迟和输入增加 M 个延迟的次数。代入 w_1' 和 w_2' 的表达式，得到

$$\begin{aligned} D_F'^D(U \to V) &= N_U[M(w_2 + r(V) - r(D_{uv})) + w_1 \\ &\quad + Mr(D_{uv}) - r(U)] - P_U + v - u \\ &= D_F^D(U \to V) + N_U(Mr(V) - r(U)) \end{aligned}$$

它与 $r(D_{uv})$ 无关。我们可以通过令 $D_F^D\ (U \to V) \geqslant 0$ 来重定时折叠DFG，得到

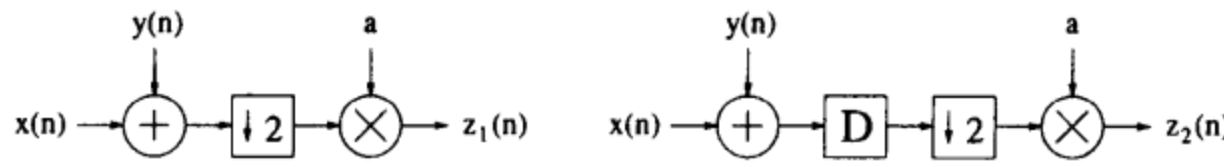
$$r(U) - Mr(V) \leqslant \frac{D_F^D(U \to V)}{N_U}$$

因为 M、$r(U)$ 和 $r(V)$ 都限定为整数，又可写为

$$r(U) - Mr(V) \leqslant \left\lfloor \frac{D_F^D(U \to V)}{N_U} \right\rfloor \tag{6-8}$$

由于MRDFG的周期时变特性，对它进行重定时必须小心。例如，考虑图6-24a中MRDFG。如果我们重定时该DFG，令加法器的重定时数值为-1，乘法器的重定时数值为0，我们就得到图6-24b中的DFG。问题是这两个电路的功能完全不同。在单速率情形下，重定时输入节点只会引起输出信号的延迟，而这个例子却表明，重定时MRDFG的输入节点会完全改变电路的功能。

173



a) 计算 $z_1(n) = a(x(2n) + y(2n))$ 的MRDFG　　b) 计算 $z_2(n) = a(x(2n-1) + y(2n-1))$ 的过程经过重定时后的架构

图 6-24

6.6 结论

折叠是设计时分复用架构的系统变换技术，它是[10][11]中技术的派生。虽然折叠电路只需较少的硅面积，但可通过运用功能单元的细粒图流水线以较高的速度运行。对于折叠因子较小的系统来说，这不会引起抽样率的净损失。折叠集可通过任何调度和分配技术（见附录B）设计。用于编译理论的寿命分析，可用来减少折叠电路的存储单元数目。本章集中讨论了标量或一维变量存储单元的减少。含有矢量或多维信号的系统的寄存器最少化，不在本书的讨论范围之内（见[12][13]）。一维多速率系统的折叠也涉及到了。对于多维单速率和多速率系统的折叠，读者可参考[14]。

6.7 习题

1. 考虑6阶FIR滤波器

$$y(n) = \sum_{i=0}^{5} h_i x(n-i)$$

用图6-25中的数据广播表实现。该滤波器以2为折叠因子，

$$S_0 = \{MA5, MA4\},\quad S_1 = \{MA3, MA2\},\quad S_2 = \{MA1, MA0\}.$$

为折叠集实现。

(a) 设计折叠架构。

174 (b) 构建与折叠架构相应的调度表，并验证该折叠架构能产生所需的滤波器输出样值。

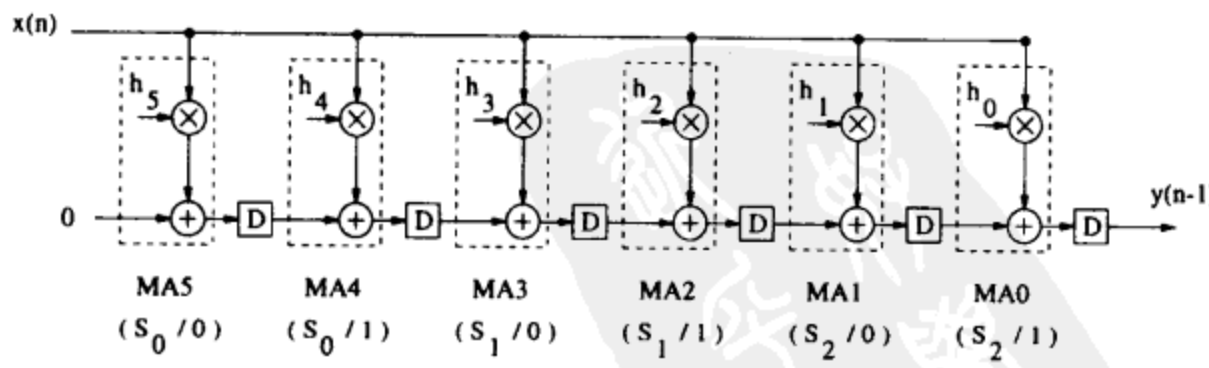


图6-25 习题1的6阶数据广播FIR滤波器

2. 对图6-26中的DFG进行折叠重定时，使得下面所示的折叠集在折叠架构中产生非负边延迟。假设折叠因子 N=5，并假设每个乘法器为2级流水线，每个加法器为1级流水线。每个运算由周期为1u.t.的时钟定时。乘法运算需要2u.t.，加法运算需要1u.t.。ϕ表示空操作。

$$
\begin{aligned}
S_{M1} &= \{M_2, M_1, M_3, M_6, M_7\} \\
S_{M2} &= \{M_4, \phi, M_5, M_8, M_9\} \\
S_{A1} &= \{A_4, \phi, A_1, A_2, A_3\} \\
S_{A2} &= \{A_5, A_6, A_7, A_8, \phi\}
\end{aligned}
$$

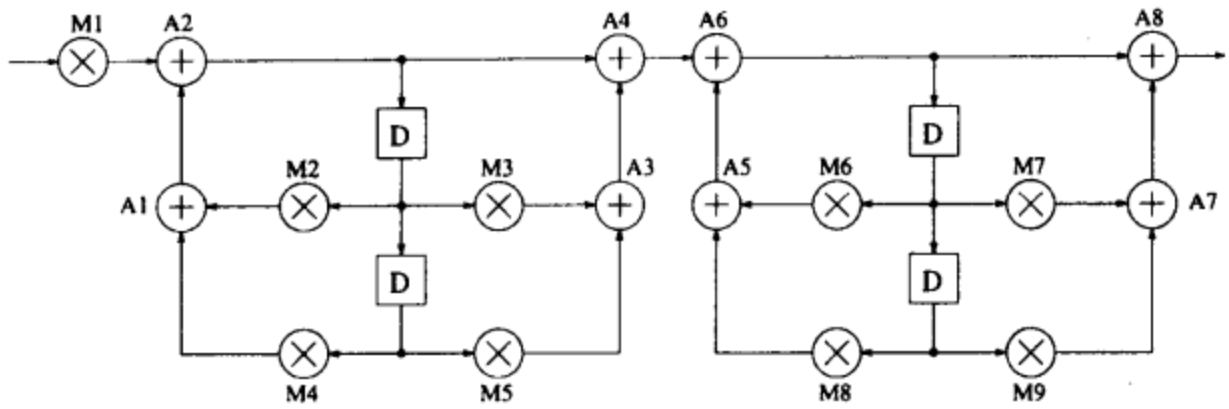


图6-26 习题2所要折叠的DFG

3. 用习题2中给出的折叠集对习题2得到的重定时DFG进行折叠。
4. 采用与习题2相同的假设，对图6-27中的DFG进行折叠，假设迭代周期 N=4 时的折叠集如下:

$$
\begin{aligned}
S_M &= \{M_4, M_1, M_3, M_2\} \\
S_A &= \{A_1, A_3, A_4, A_2\}
\end{aligned}
$$

175

(a) 进行折叠重定时，使得折叠集在折叠架构中产生非负边延迟。
(b) 折叠重定时后的DFG。

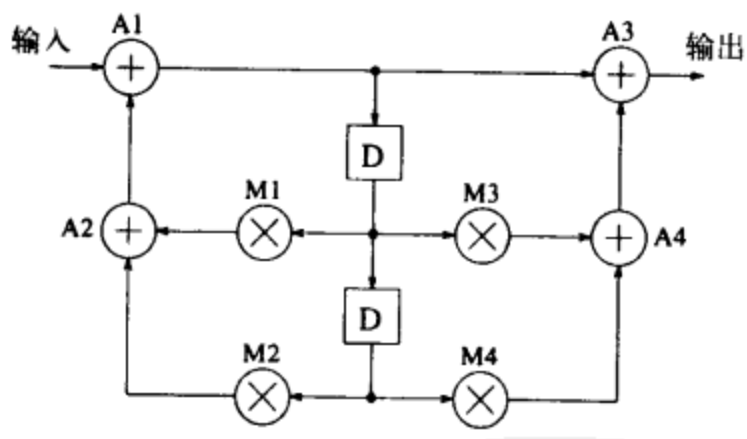


图6-27 习题4、5、6所要折叠的DFG

5. 用折叠集

$$
\begin{aligned}
S_M &= \{M_2, M_1, M_3, M_4\} \\
S_A &= \{A_1, A_4, A_3, A_2\}
\end{aligned}
$$

重复习题4。

6. 用折叠集

$$
\begin{aligned}
S_M &= \{M_4, M_1, M_2, M_3\} \\
S_A &= \{A_1, A_4, A_3, A_2\}
\end{aligned}
$$

重复习题4。

7. 本题的目的是用图6-28所描述的折叠集，对该图中的格型滤波器进行折叠。假设乘法

运算映射为2级流水线乘法硬件，加法运算映射为1级流水线加法器。硬件架构需用周期为1u.t.的时钟定时。

(a) 进行折叠重定时，使得所有折叠后的边延迟非负。

(b) 折叠重定时后的DFG。

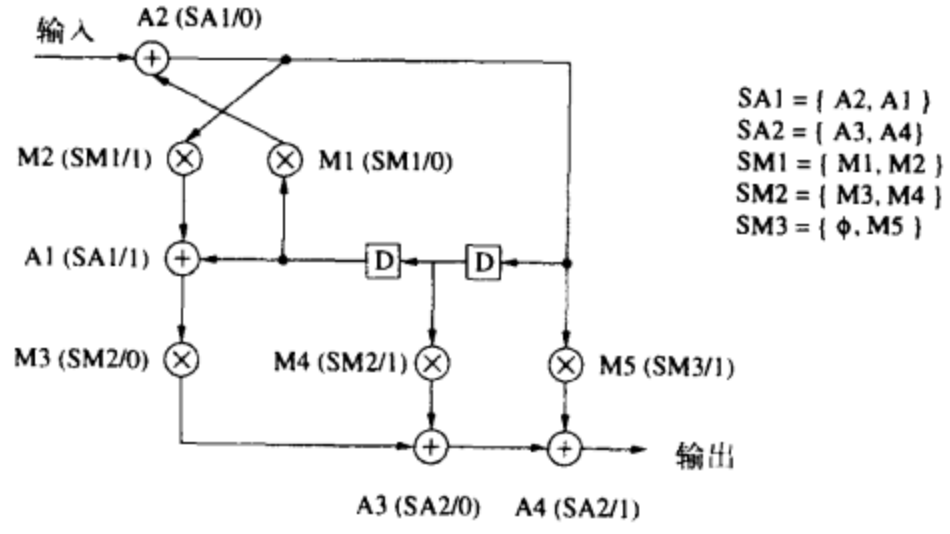


图6-28 习题7用到的格型滤波器

8. 表6-7给出了数据格式转换器的变量名、输入时间、理想0延迟输出时间。我们的问题是设计一个满足这些约束的架构。得到延迟（T_{lat}）、最小寄存器数、分配表和相应的用前向-后向分配法所综合出的架构。

9. 采用与习题2相同的假设，对图6-29中的DFG进行折叠，假设迭代周期 N=2 时的折叠集如下：

$$
\begin{aligned}
S_{M1} &= \{M_2, M_1\} \\
S_{M2} &= \{M_3, M_4\} \\
S_{A1} &= \{A_1, A_2\} \\
S_{A2} &= \{A_4, A_3\}
\end{aligned}
$$

(a) 进行折叠重定时，使得折叠集在折叠架构中产生非负边延迟。

(b) 折叠重定时后的DFG。

(c) 最小化寄存器数并重新设计折叠架构。

表6-7 习题8讨论的转换器的指标

变量名	输入时间单位	零延迟输出时间单位
x_0	0	0
x_1	0	6
x_2	2	0
x_3	2	6
x_4	4	0
x_5	4	6
x_6	6	3
x_7	6	9
x_8	8	3
x_9	8	9
x_{10}	10	3
x_{11}	10	9

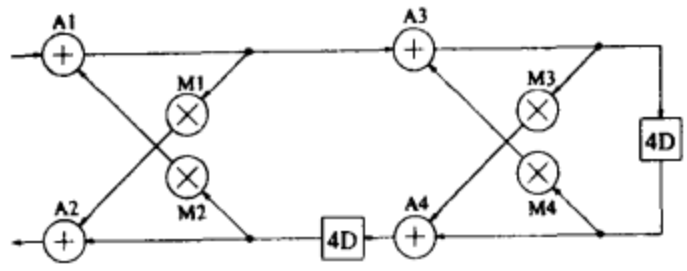


图6-29　习题9~13所要折叠的DFG

10. 令迭代周期 N=2，用折叠集

$$\begin{aligned} S_{M1} &= \{M_2, M_1\} \\ S_{M2} &= \{M_4, M_3\} \\ S_{A1} &= \{A_1, A_2\} \\ S_{A2} &= \{A_3, A_4\} \end{aligned}$$

重复习题9。

11. 令迭代周期 N=2，用折叠集

$$\begin{aligned} S_{M1} &= \{M_3, M_1\} \\ S_{M2} &= \{M_2, M_4\} \\ S_{A1} &= \{A_1, A_3\} \\ S_{A2} &= \{A_4, A_2\} \end{aligned}$$

重复习题9。

12. 令迭代周期 N=4，用折叠集

$$\begin{aligned} S_M &= \{M_3, M_1, M_2, M_4\} \\ S_A &= \{A_1, A_3, A_4, A_2\} \end{aligned}$$

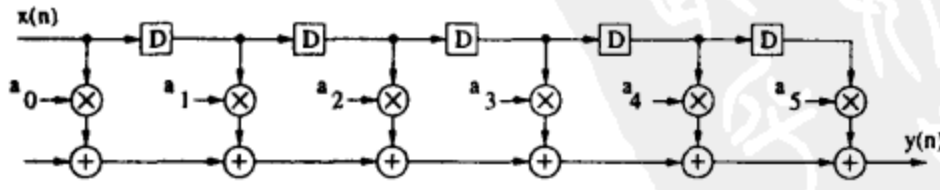
重复习题9。

13. 令迭代周期 N=4，用折叠集

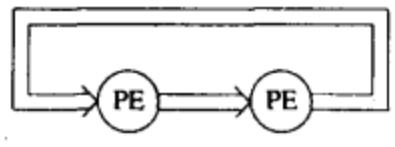
$$\begin{aligned} S_M &= \{M_4, M_1, M_2, M_3\} \\ S_A &= \{A_1, A_4, A_3, A_2\} \end{aligned}$$

重复习题9。

14. 本题讨论图6-30所示的6阶FIR滤波器的实现，它应用了包含2个处理元件的环形架构（PEs），每个PE包含了运用折叠技术的乘法加法器。由于环形架构的循环特性，硬件并没有充分利用。



a) 6阶FIR滤波器



b) 环形结构

图6-30　习题14图示

折叠架构硬件的充分利用，可以通过实现一个2阶展开的6阶滤波器来实现。首先，6阶FIR滤波器，如图6-31所示，以2为系数展开。这两个独立运算可以通过交织的方式来调度。2阶展开滤波器的以6为折叠因子的折叠集如下：

$$S1 = \{a_0^0, a_4^1, a_2^0, a_0^1, a_4^0, a_2^1\}$$
$$S2 = \{a_3^1, a_1^0, a_5^1, a_3^0, a_1^1, a_5^0\}$$

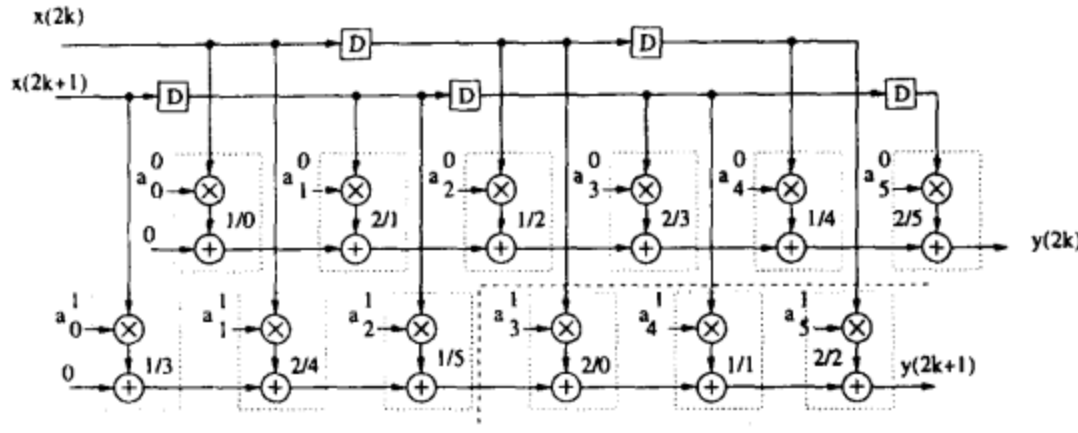


图6-31 习题14的2阶展开6阶FIR滤波器

其中，上标为 i 的系数用于计算 $y(2k+i)$，i=0，1。图6-31所示为折叠集。为了折叠，乘法和它相应的加法运算被看作一个单独运算。

(a) 验证为折叠进行的重定时，会导致DFG的流水线中的延迟元件置于前向割集，如图6-31所示。

(b) 为6阶FIR滤波器设计折叠架构。

(c) 为最小化所需寄存器数，将 $x(2k)$ 和 $x(2k+1)$ 的并行延迟线替换为输入 $x(n)$ 的单个延迟线。假设 $x(n)$ 在 $3n$ 时刻被抽样。证明该折叠电路所需最小寄存器数为7。

(d) 用基于前向-后向分配得到的最小寄存器数，综合出折叠架构。

15. 动态规划（DP）用来求解通信和控制、人工智能和运算研究（见第1.2.8节）等问题。DP问题具有以下属性，即由初始迭代到第 $i+j$ 次迭代的最优方案，必然是由初始迭代到 i 次迭代的最优方案和 i 次迭代到 $i+j$ 次迭代的最优方案所组成。在信号处理中，DP通常用于通信系统中的维特比解码器和语音识别系统中隐性马尔可夫模型[15]。考虑由

$$x_i(n+1) = \max_j [x_j(n) + a_{ji}(n)],\ i, j = 1, 2, \cdots, N \tag{6-9}$$

给出的N态DP问题，其中，$x_i(n)$ 为 n 次迭代的状态 i，变量 $\{a_{ji}\}$ 为格点或路径系数。公式（6-9）中的基本运算为相加-比较-选择（ACS）。为了更新N个状态值，N态DP问题需要 N^2 次ACS运算。N=4 的DFG如图6-32所示，它对应一个环形脉动结构。系数 a_{ji} 存于节点 A_{ji}，DFG由右边的 i 条边环绕限制，并与左边同样数量的边相连接。

(a) 写出图6-32中的DFG的 n 次迭代时所有的ACS单元所执行的运算，并验证该DFG沿着第1、2、3、4列，分别更新了 $x_1(n+1)$、$x_2(n+1)$、$x_3(n+1)$ 和 $x_4(n+1)$。

(b) 以4为折叠因子折叠DFG，流水线级数为1，折叠集如下：

$$\begin{aligned} S_1 &= \{A_{41}, A_{31}, A_{21}, A_{11}\} \\ S_2 &= \{A_{12}, A_{42}, A_{32}, A_{22}\} \\ S_3 &= \{A_{23}, A_{13}, A_{43}, A_{33}\} \\ S_4 &= \{A_{34}, A_{24}, A_{14}, A_{44}\} \end{aligned}$$

(提示：你应该得到一个环形脉动结构。)

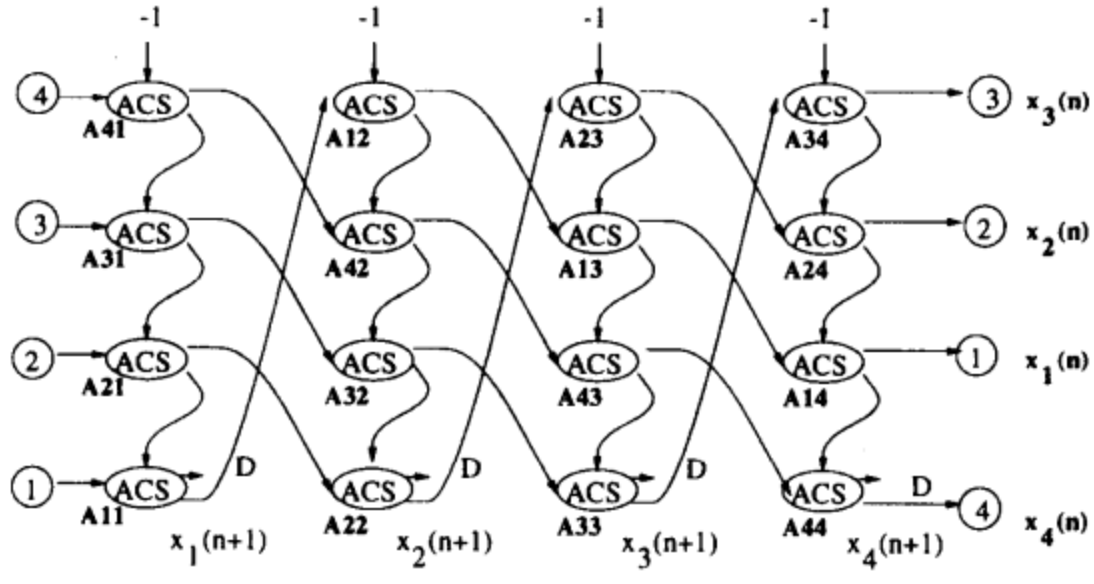
176 ~ 180

(c) 以8为折叠因子折叠DFG，流水线级数为2，折叠集如下：

$$S_1 = \{A_{41}, A_{23}, A_{31}, A_{13}, A_{21}, A_{43}, A_{11}, A_{33}\}$$
$$S_2 = \{A_{12}, A_{34}, A_{42}, A_{24}, A_{32}, A_{14}, A_{22}, A_{44}\}$$

(d) 以16为折叠因子，将DFG中的所有节点折叠到一个处理元件上，流水线级数为4，折叠集如下：

$$S_1 = \{A_{41}, A_{12}, A_{23}, A_{34}, A_{31}, A_{42}, A_{13}, A_{24}, A_{21}, A_{32}, A_{43}, A_{14}, A_{11}, A_{22}, A_{33}, A_{44}\}$$



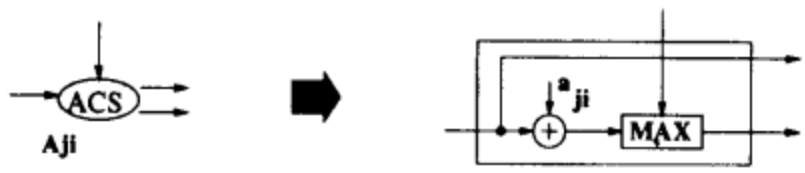


图6-32　习题15中DP的DFG

16. 对图6-33中的格型滤波器，令折叠因子为N=8，折叠集为：

$$S_A = \{A_1, A_4, A_7, A_2, A_3, A_5, A_8, A_6\}$$
$$S_M = \{M_4, M_1, M_5, \phi, \phi, M_2, M_3, \phi\}$$

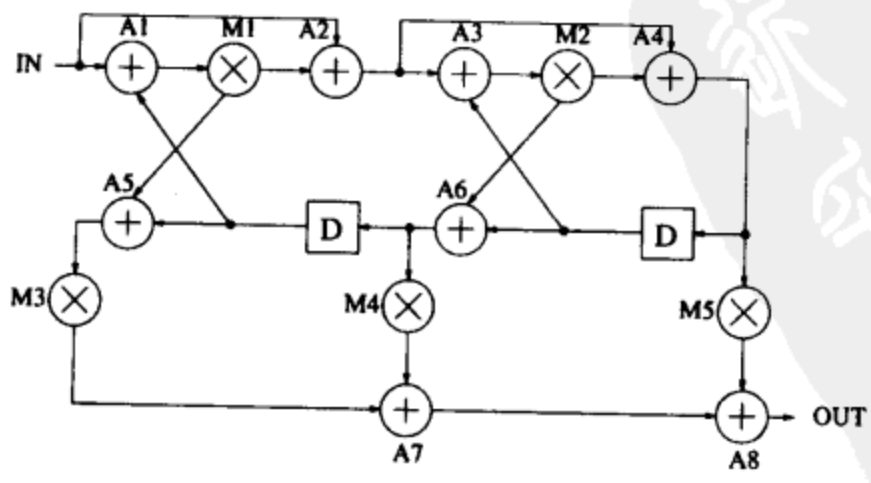


图6-33　习题16用到的格型滤波器

假设加法和乘法分别需要1 u.t.和2 u.t.。

(a) 为了折叠进行重定时，使得所有折叠后的边含有非负延迟。

(b) 综合出该系统的折叠架构。

(c) 运用寿命分析，最小化架构中的延迟元件。用前向-后向分配法将变量分配给最小数目的延迟元件。重新画出综合出的折叠架构。

17. 以 $N=2$ 为折叠因子，折叠图6-34所示的格型滤波器。如果必要，可为了折叠进行重定时。

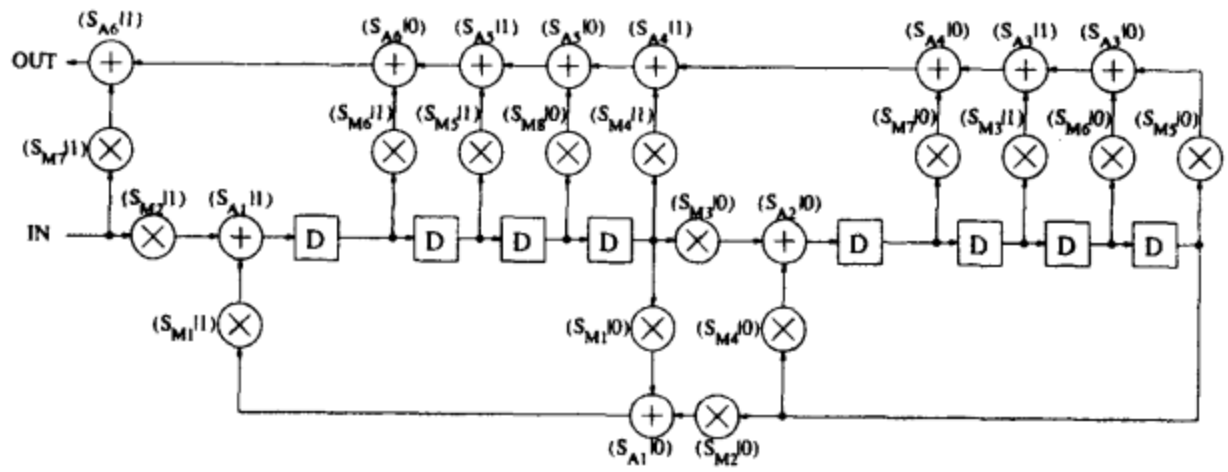


图6-34 习题17用到的归一化格型滤波器

18. 以4为折叠因子，

$$S_{M_1} = \{M1, M2, M3, M4\}$$
$$S_{A_1} = \{A3, A4, A1, A2\}$$

为折叠集，设计图6-35所示的流水线/重定时双2次节滤波器的折叠架构。以4为展开因子展开该折叠DFG，验证展开后的DFG与图6-35中的流水线/重定时双2次节DFG相同。

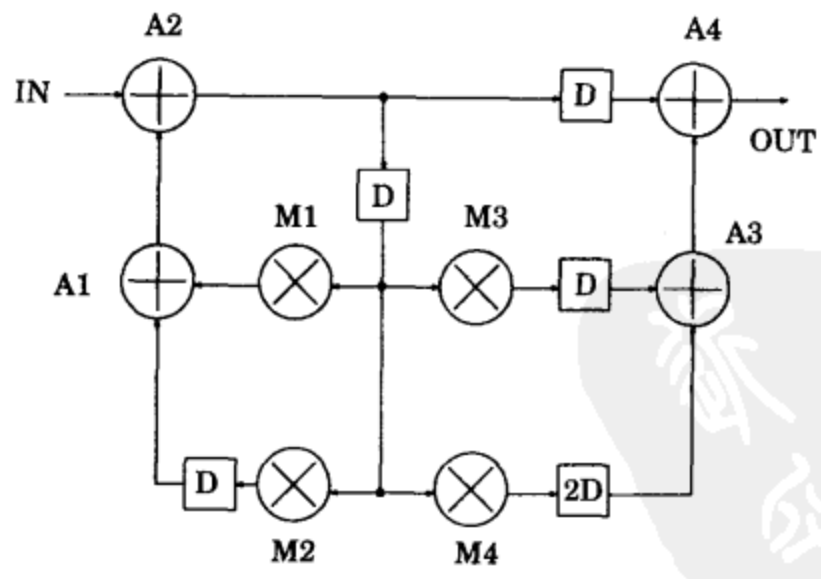


图6-35 习题18用到的双2次节滤波器

19. 考虑习题7中折叠后的格型滤波器。由习题7所得折叠边延迟，通过寿命分析，确定为了存储每个功能单元（功能单元即加法器或乘法器）的输出所需的最小寄存器数。然后用前向-后向寄存器分配为每个功能单元综合出一个存储单元。用这些最小寄存器存储单元画出完整的折叠架构。将架构中的寄存器数与习题7中得到的寄存器数相

比较。

20. 令 w_l 和 t_l 为环路 l 的延迟数和计算时间。假设计算时间为 t 的节点，在 t 级流水线处理器上执行。

(a) 证明

$$\sum_l D_F(U \to V) = Nw_l - t_l$$

一定成立，N为折叠因子。

(b) 利用(a)的结果证明，折叠因子N不能小于迭代边界T_∞。

21. 考虑图6-36所示的4阶重定时正交滤波器结构，其中所有计算都是CORDIC旋转操作。该结构用一个硬件CORDIC旋转算子实现，该算子可以用折叠技术来得到。令折叠因子为N，本题不要求寄存器数最小化。

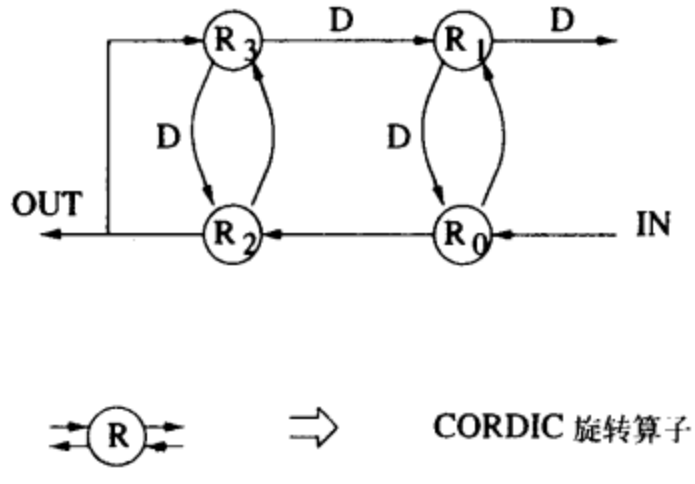


图6-36　习题21的DFG

假设每个CORDIC旋转操作在3.3V电源电压下需要T纳秒时间。折叠架构应该设计使得$4T$的抽样周期可以实现，而功耗则通过使用更低电源电压和尽可能多的功能单元的流水线来最小化。假设工艺的阈值电压为0.5V，为保持高噪声抑制度，工艺要求的最小电源电压为2V。

(a) 首先考虑使用细粒流水线CORDIC旋转算子的可能性。假设CORDIC旋转算子的流水线级数为P。系统地证明:

i. 如果折叠因子N等于4，P不能大于2。(提示：利用环路给算法添加的约束，见习题20(a)。)

ii. 对 P=2 和 N=4，没有可行的折叠集。

(b) 用1态流水线CORDIC旋转算子和折叠集 $S = \{R_0, R_1, R_2, R_3\}$ 设计折叠架构。

(c) 本部分将用到2级流水线CORDIC旋转算子。以5为折叠因子和折叠集 $S = \{R_0, R_3, \varphi, R_1, R_2\}$ 设计折叠架构。请注意2级流水线CORDIC旋转算子的关键路径为$0.5T$，而$0.8T$的时钟周期已经足够满足抽样周期为$4T$的条件。因此该架构可以在更低的电源电压下工作，从而减小它的功耗。

(d) 假设锁存器、多路选择器和总线不消耗任何功耗。在1阶近似下比较(b)和(c)中的设计功耗。忽略扇出带来的容性负载效应和毛刺效应。为此，假设(b)中的设计在3.3V电源电压下工作。(提示：首先计算出(c)中设计的近似电源电压。)

(e) 功耗可以通过使用更长流水线CORDIC旋转算子 ($P > 2$) 和更大的折叠因子 ($N > 5$)

来减小吗？用一或两个例子解释你的答案。

181 ~ 184

22. 考虑图6-37中的边 $U \to V$，节点 U 的输出在到达节点 V 之前，经过了 w_1 个延迟，L 倍扩展和 w_2 个延迟。令节点 U 的 l 次迭代在 $N_U l + u$ 时间执行，节点 V 的 l 次迭代在 $Nvl + u$ 时间执行，折叠阶数满足 $u \in [0, N_U)$ 和 $u \in [0, N_v)$。
    (a) 用信号 y 写出 $x(l)$ 的表达式。
    (b) 具有 P_U 级流水线的功能单元 H_U，在什么时刻产生节点 U 的 l 次迭代结果？该样值在什么时刻被功能单元 H_V 使用？
    (c) 利用(b)的结果，得出图6-37中的边的折叠方程。使得该方程与迭代次数 l 无关的 N_U 和 N_V 的关系是什么？
    (d) 推导为折叠这个边所需的重定时不等式。

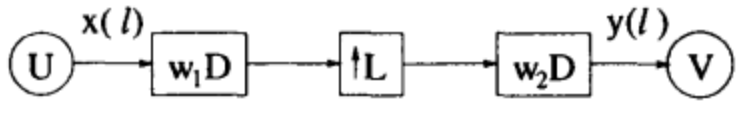


图6-37 包含扩展器的边 $U \to V$

23. 本题利用多速率迭代设计了一个离散小波变换架构(见第1.2.10节)。折叠图6-38所示的3阶小波滤波器的3级正交将小波变换分析滤波器的阶数离散。该DFG已经经过重定时，使所有折叠后的边具有非负延迟。该折叠架构的硬件利用率如何？该架构的调度表在表6-8给出。调度表中的 X 表示空操作。

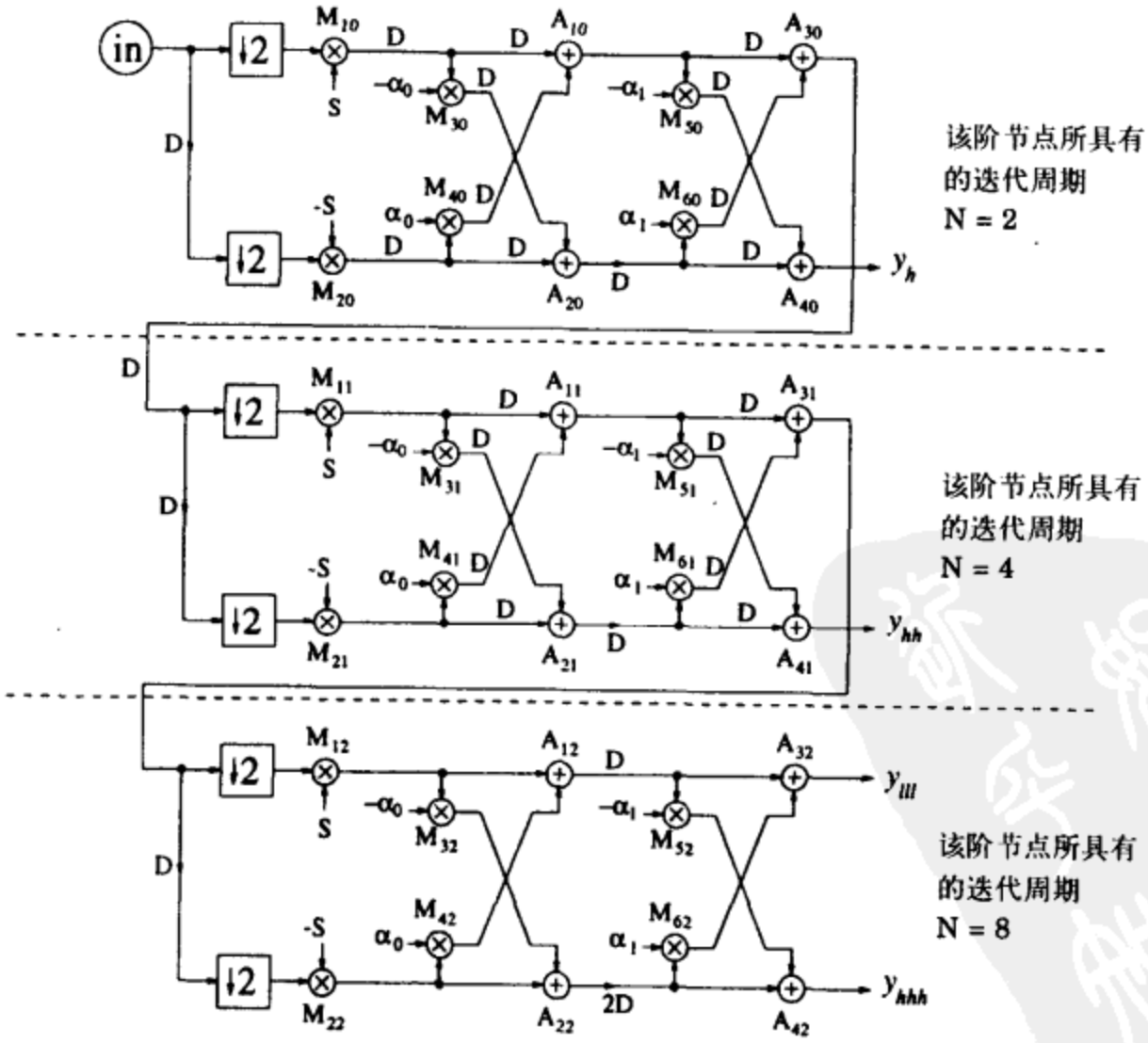


图6-38 采用3阶小波滤波器的3级正交离散小波变换分析滤波器组

表6-8 周期为8的3级正交DWT例子的调度表

	0	1	2	3	4	5	6	7
Processor M_1	M_{10}	M_{11}	M_{10}	M_{12}	M_{10}	M_{11}	M_{10}	X
Processor M_2	M_{20}	M_{21}	M_{20}	M_{22}	M_{20}	M_{21}	M_{20}	X
Processor M_3	M_{30}	X	M_{30}	M_{31}	M_{30}	M_{32}	M_{30}	M_{31}
Processor M_4	M_{40}	X	M_{40}	M_{41}	M_{40}	M_{42}	M_{40}	M_{41}
Processor M_5	M_{52}	M_{50}	M_{51}	M_{50}	X	M_{50}	M_{51}	M_{50}
Processor M_6	M_{62}	M_{60}	M_{61}	M_{60}	X	M_{60}	M_{61}	M_{60}
Processor A_1	A_{10}	A_{11}	A_{10}	X	A_{10}	A_{11}	A_{10}	A_{12}
Processor A_2	A_{20}	A_{21}	A_{20}	X	A_{20}	A_{21}	A_{20}	A_{22}
Processor A_3	A_{31}	A_{30}	A_{32}	A_{30}	A_{31}	A_{30}	X	A_{30}
Processor A_4	A_{41}	A_{40}	A_{42}	A_{40}	A_{41}	A_{40}	X	A_{40}

参考文献

1. K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, Jan. 1992.
2. K. K. Parhi, "High-level algorithm and architecture transformations for DSP synthesis," *Journal of VLSI Signal Processing*, vol. 9, no. 1, pp. 121–143, Jan. 1995.
3. K. K. Parhi, "Calculation of minimum number of registers in arbitrary life time chart," *IEEE Trans. on Circuits and Systems–II*, vol. 41, no. 6, pp. 434–436, June 1994.
4. P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Prentice Hall, 1993.
5. A. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.
6. K. K. Parhi, "Systematic synthesis of DSP data format converters using life time analysis and forward-backward register allocation," *IEEE Trans. on Circuits and Systems - II*, vol. 39, no. 7, pp. 423–440, July 1992.
7. A. Tucker, "Coloring a family of circular arcs," *SIAM Journal of Applied Mathematics*, vol. 29, no. 3, pp. 493–502, Nov. 1975.
8. L. Stok and J. A. G. Jess, "Foreground memory management in data path synthesis," *International Journal of Circuit Theory and Applications*, vol. 20, no. 3, pp. 235-255, 1992.
9. T. C. Denk and K. K. Parhi, "Synthesis of folded pipelined architectures for multirate DSP algorithms," *IEEE Trans. on VLSI Systems*, vol. 6, no. 4, Dec. 1998.
10. E. F. Girczyc, "Loop winding–a data flow approach to functional pipelining," in *Proc. of IEEE Symposium on Circuits and Systems*, pp. 382–385, May 1987.
11. G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man, "An efficient microcode compiler for application specific DSP processors," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, pp. 925–937, 1990.
12. F. Franssen, F. Balasa, M. van Swaaij, F. Catthoor, and H. De Man, "Modeling

185 ~ 186

multi-dimensional data and control flow," *IEEE Trans. on VLSI Systems*, vol. 1, pp. 319–327, Sept. 1993.

13. F. Balasa, F. Catthoor, and H. De Man, "Dataflow-driven memory allocation for multi-dimensional signal processing systems," in *Proc. of IEEE International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 31–34, Nov. 1994.

14. V. Sundararajan and K. K. Parhi, "Synthesis of folded, pipelined architectures for multi-dimensional multirate systems", in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Seattle, WA), pp. 3089–3092, May 1998.

15. K. K. Parhi, "Pipelining in dynamic programming architectures," *IEEE Trans. on Signal Processing*, pp. 1442–1450, 1991.

187

第7章 脉动结构设计

7.1 引言

脉动结构（也称脉动阵列）表示一种有节奏地计算并通过系统传输数据的处理单元（processing elements，PE）网络。这些处理单元规则地泵入泵出数据以维持规则的数据流（[1]，[2]）。脉动系统的特征是模块化和规则化，这对于VLSI设计是很重要的性质。脉动阵列可作为和主计算机结合的协处理器，从主计算机接受数据样本送到处理单元，并将最终结果返回主计算机（见图7-1）。这个操作类似心脏的血液流动，因此命名为“脉动”。

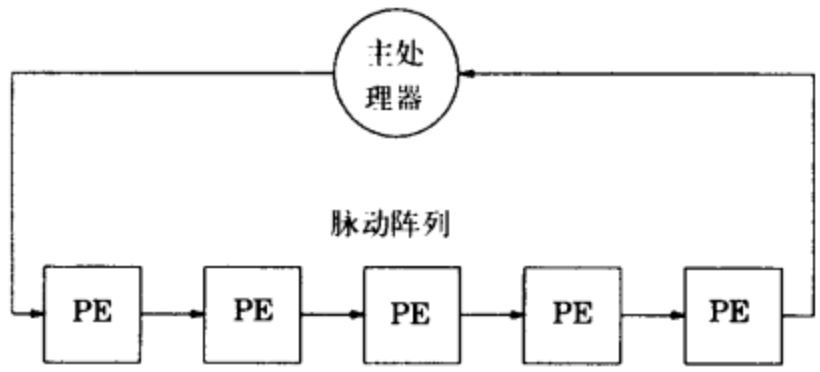


图7-1 脉动系统的基本原理

典型情况下，脉动阵列的所有处理单元是相同且全流水的，也就是处理部件的所有通信边沿都包含延时单元，且整个系统通常只包含局部互联[3]。但是，为了扩大脉动阵列的用途，可引入一些弛豫。这些弛豫包括使用局部和邻位（靠近但不是最近）互联，使用数据广播操作，以及在系统中使用不同的处理单元，尤其在边界上。通过这些弛豫，可设计出一系列适合DSP应用的模块化、规则且有效的数据驱动脉动结构。

本章给出脉动结构设计的方法原理，其中，对任意给定的规则迭代算法，用线性映射或投影技术可设计出多种脉动结构。线性脉动映射方法原理在第7.2节介绍，并用于第7.3节的推导FIR滤波器脉动阵列。第7.4节说明如何选择调度矢量。第7.5节推导矩阵相乘问题的几种脉动阵列。第7.6节扩展映射技术以适应规则的迭代算法，进而产生有延时的依赖图（dependence graph, DG）。

7.2 脉动阵列设计方法原理

脉动结构是将线性映射技术用于规则依赖图（[3]~[7]）上进行设计的。依赖图的边表示优先约束。如果依赖图任意一个节点沿某个方向的边的存在表示依赖图所有节点该方向的边的存在，则称依赖图（dependence graph, DG）是规则的。

例7.2.1 分析图7-2所示的一个3阶FIR滤波器的依赖图：

$$y(n) = \omega_0 x(n) + \omega_1 x(n-1) + \omega_2 x(n-2) \tag{7-1}$$

这个依赖图有3条基本边：$[0\ 1]^T$方向的边表示上移的输入、$[1\ 0]^T$方向的边表示右移的系

数、而结果沿$[1-1]^T$方向移动。因为依赖图的所有节点包含着同样的三条边，因此该依赖图是规则的。 ■

依赖图是一种空间表示，不分派时间步骤给任何运算。一般情况下，对应 $t = 0$的平面。映射技术把空间表示变换为一种空间-时间表示，其中每个节点映射到特定的处理单元，且调度到特定的时间步骤下。

189 ~ 190

脉动设计方法原理将N维依赖图映射到一种较低维的脉动结构。本章中，只考虑1级映射，就是将N维依赖图映射到（$N-1$）维脉动阵列。

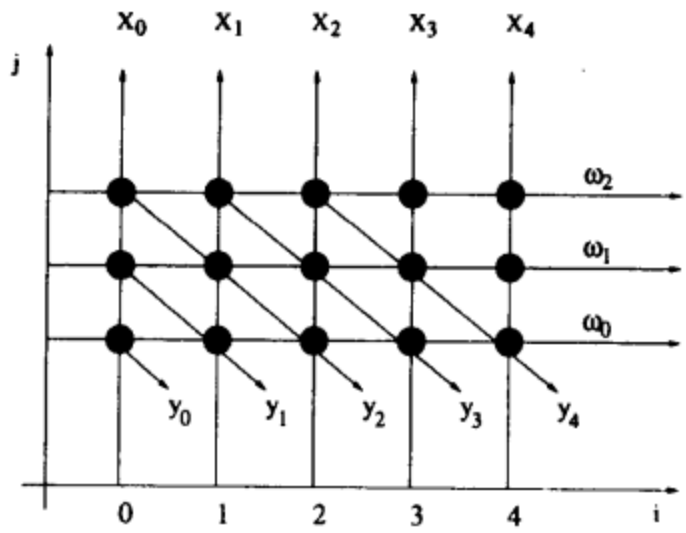


图7-2 FIR滤波器的空间表示

现在定义脉动阵列设计中涉及到的基本矢量。

- 投影矢量（也称迭代矢量），$\mathbf{d} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$。

  位移为 $\mathbf{d}$ 或 $\mathbf{d}$ 的倍数的两个节点由同一个处理器执行。
- 处理器空间矢量，$\mathbf{P}^T = (p_1\ p_2)$。

  任何标识为 $I^T = (i, j)$ 的节点将如下被处理器处理。

$$\mathbf{p}^T I = \begin{pmatrix} p_1 & p_2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

- 调度矢量，$\mathbf{s}^T = (s_1\ s_2)$。

  任何标识为 I 的节点在时间 $\mathbf{s}^T I$ 执行。
- 硬件利用率，$HUE = 1/|\mathbf{s}^T\mathbf{d}|$。

  这是因为用同样的处理器执行的两次任务相隔$|\mathbf{s}^T\mathbf{d}|$时间单位。

对一个给定问题可通过选择不同的投影、处理器空间矢量和调度矢量设计出很多脉动结构。这些矢量必须满足下面推导的可行性限制条件。

- 处理器空间矢量和投影矢量必须互相正交。如果点 A 和 B 相差投影矢量（也就是 $I_A - I_B$）和 $\mathbf{d}$ 相同，那么它们必须用同样的处理器处理。也就是说，$\mathbf{p}^T I_A = \mathbf{p}^T I_B$ 由此得：

191

$$\mathbf{p}^T(I_A - I_B) = 0 \Rightarrow \mathbf{p}^T\mathbf{d} = 0 \tag{7-2}$$

- 如果 A 和 B 映射到同一个处理器，则它们不能在同一时间执行，即：

$$\mathbf{s}^T I_A \neq \mathbf{s}^T I_B, \text{也就是，} \mathbf{s}^T\mathbf{d} \neq 0 \tag{7-3}$$

- 边映射：如果依赖图中或空间表示中包含边 $\mathbf{e}$，则在脉动阵列中引入一条延时为 $\mathbf{s}^T\mathbf{e}$ 的边 $\mathbf{p}^T\mathbf{e}$。

空间表示或依赖图可以变换为空间-时间表示或几何表示，只要将空间维度中的一个维度认为是时间维度即可[8]。对于一个二维（2D）的依赖图，一般的变换采用 $i' = t = 0$，$j' = \mathbf{p}^T I$ 和 $t' = \mathbf{s}^T I$，或者等效为：

$$\begin{pmatrix} i' \\ j' \\ t' \end{pmatrix} = T\begin{pmatrix} i \\ j \\ t \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ & \mathbf{p}^T & 0 \\ & \mathbf{s}^T & 0 \end{pmatrix}\begin{pmatrix} i \\ j \\ t \end{pmatrix} \tag{7-4}$$

在空间-时间表示中，j' 轴表示处理器轴（回顾二维依赖图映射到一维脉动阵列），且 t' 表示调度的时间步骤。

7.3 FIR脉动阵列

本节用线性映射技术导出一系列用于FIR数字滤波器的脉动阵列。

7.3.1 设计 B_1（输入广播，结果移动，权重保持）

脉动设计 B_1 通过如下选择投影矢量、处理器矢量和调度矢量得到：

$$\mathbf{d} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\quad \mathbf{p}^T = \begin{pmatrix} 0 & 1 \end{pmatrix},\quad \mathbf{s}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{7-5}$$

根据这些定义，可得到：

- 任何标识为 $I^T = (i, j)$ 的节点映射到处理器

$$\mathbf{p}^T I = \begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} i \\ j \end{pmatrix} = j$$

因此，所有水平线上的节点映射到同一个处理器。 192

- 任何标识为 $I^T = (i, j)$ 的节点的执行时间为：

$$\mathbf{s}^T I = \begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} i \\ j \end{pmatrix} = i$$

- 因为

$$\mathbf{s}^T\mathbf{d} = \begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1$$

因此有：

$$HUE = \frac{1}{|\mathbf{s}^T\mathbf{d}|} = 1 \tag{7-6}$$

- 边映射：对应权重、输入和结果的3条基本边可映射为表7-1的脉动阵列对应的端口。

因此，可构造出 B_1 脉动阵列设计的框图如图7-3所示。这个结构的低级别实现如图7-4所示。

表7-1 设计 B_1 的边的映射表

e^T	p^Te	s^Te
wt(1 0)	0	1
i/p(0 1)	1	0
result(1 −1)	−1	1

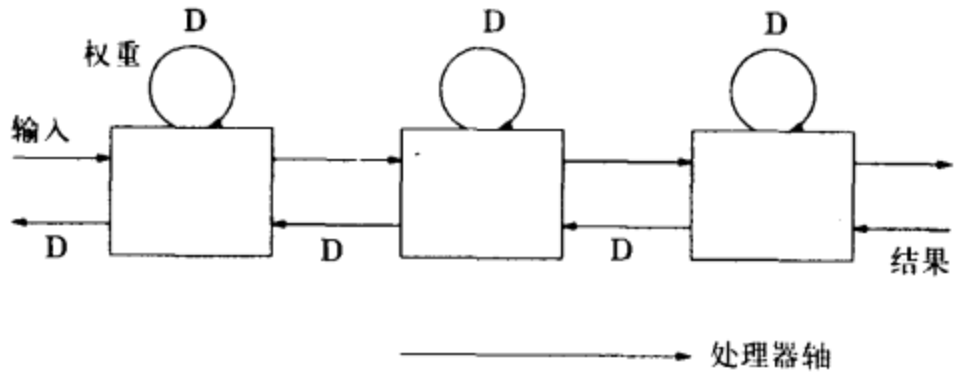


图7-3 设计 B_1 的框图

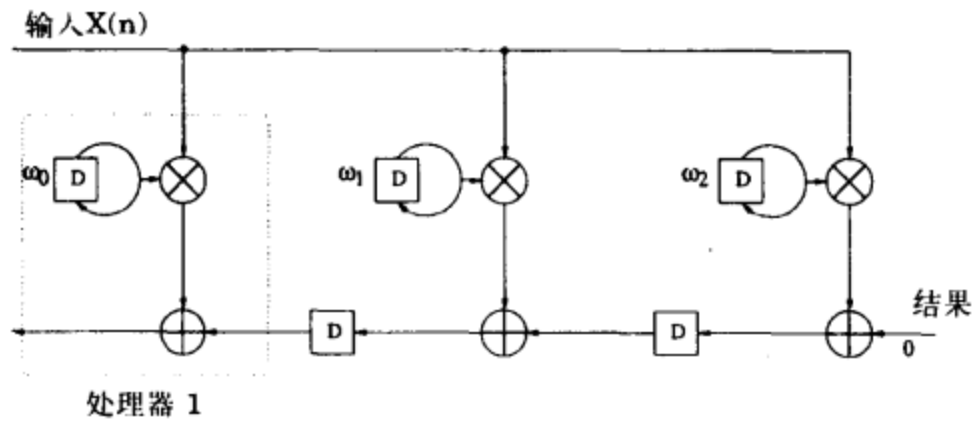


图7-4 设计 B_1 的低级别实现

设计 B_1 的空间-时间表示如图7-5所示，由图可知输入的 x 值可同时在所有的处理器得到。特定的说法是输入数据“广播”到各处理器。

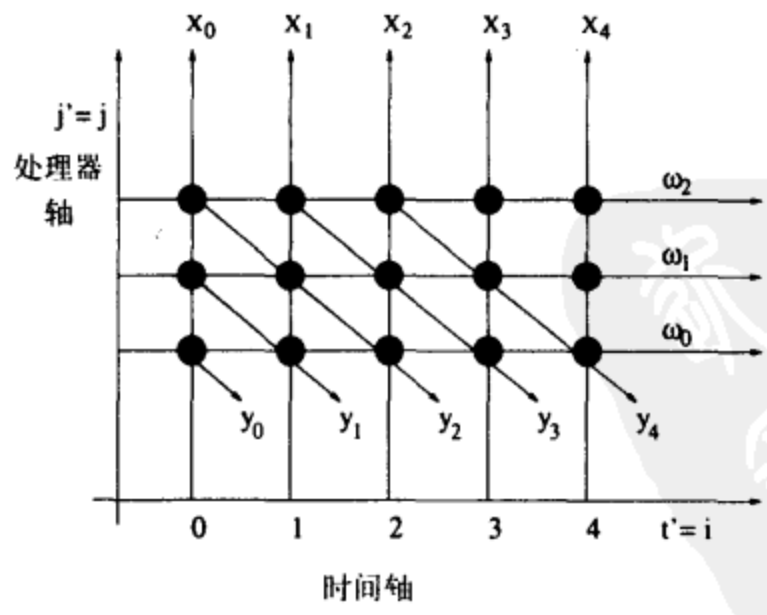


图7-5 设计 B_1 的空间-时间表示

类似地，权重 ω_i 出现在各处理器相同的空间坐标。因此，ω_i 值“保持”。而输出 y_i 出现在各处理器不同的空间和时间，即输出“移动”。

7.3.2 设计 B_2（输入广播，权重移动，结果保持）

设计 B_2 选择的投影矢量、处理器矢量和调度矢量为:

$$\mathbf{d}=\begin{pmatrix}1\\-1\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}1 & 1\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}1 & 0\end{pmatrix} \tag{7-7}$$

显然这些矢量满足前面的限制条件且

$$\mathbf{s}^T\mathbf{d}=\begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}1\\-1\end{pmatrix}=1$$

193 ~ 194

因此，$HUE=1$。设计 B_2 的边的映射如表7-2所示。

表7-2 设计 B_2 的边的映射

$\mathbf{e}$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt(1, 0)	1	1
i/p(0, 1)	1	0
result(1, −1)	0	1

可以看到设计 B_2 中移动的是权重而不是 B_1 中的结果，但两种情况的输入都被广播。该设计的脉动阵列如图7-6所示，低级别实现如图7-7所示。

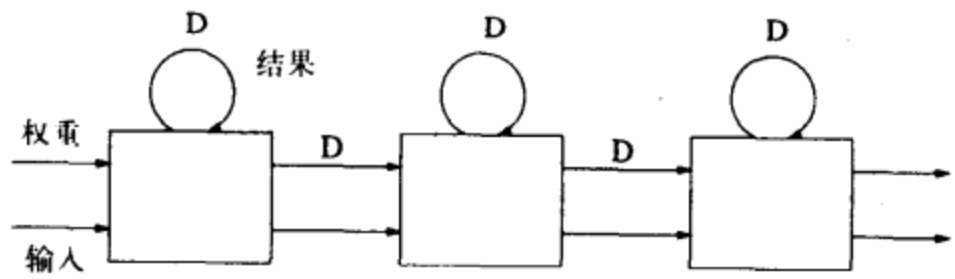


图7-6 设计 B_2 的框图

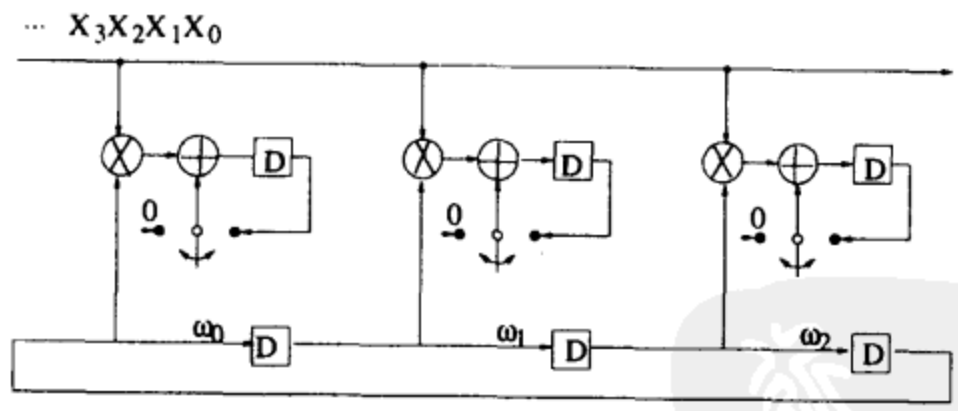


图7-7 设计 B_2 的低级别实现

应用空间-时间变换，有:

$$\begin{aligned} j' &= \mathbf{p}^T\begin{pmatrix}i\\j\end{pmatrix}=i+j \\ t' &= \mathbf{s}^T\begin{pmatrix}i\\j\end{pmatrix}=i \end{aligned} \tag{7-8}$$

因此，B_2 的空间-时间表示如图7-8所示。根据该图，显然有输入数据 x_i 同时出现在各处理器。所以输入数据是广播的；权重 ω_i 出现在各处理器不同的空间和时间，所以权重“移 195

动”；而输出 y_i 保持在相同的处理器。

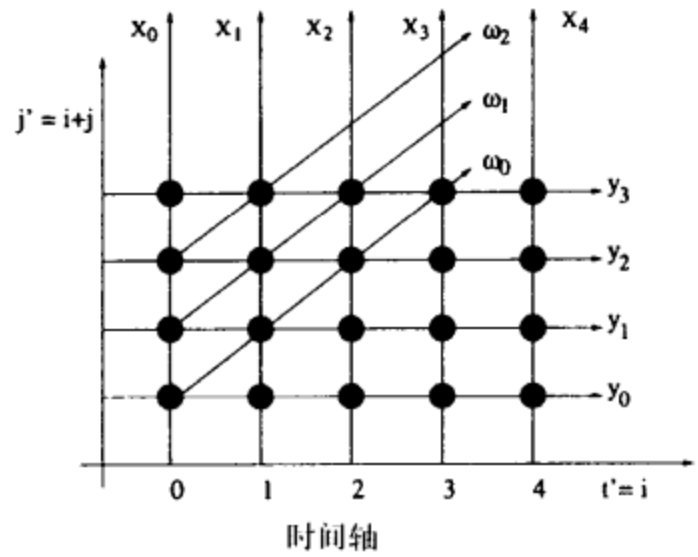


图7-8 设计 B_2 的空间-时间表示

7.3.3 设计F（结果扇入，输入移动，权重保持）

设计 F 选择的投影矢量、处理器矢量和调度矢量为:

$$\mathbf{d}=\begin{pmatrix}1\\0\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}0 & 1\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}1 & 1\end{pmatrix} \tag{7-9}$$

这种情况下，硬件利用率HUE等于1。同样，应用这些矢量得到边映射如表7-3所示。根据表7-3，显然权重 ω_i 的空间固定，输入矢量 x_i 以1延时单位从左向右移动。输出 y_i 无延时地从右向左移动。脉动阵列和低级别实现分别如图7-9和图7-10所示。这个结构的输出即为“扇入”。

表7-3 设计F的边映射

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt(1, 0)	0	1
i/p(0, 1)	1	1
result(1, −1)	-1	0

196 也可以用如下的几何（空间-时间）变换验证:

$$j'=\mathbf{p}^T\begin{pmatrix}i\\j\end{pmatrix}=j,\quad t'=\mathbf{s}^T\begin{pmatrix}i\\j\end{pmatrix}=i+j$$

因此，设计 F 的空间-时间表示如图7-11所示。根据几何表示，有权重保持、输入移动和输出扇入。

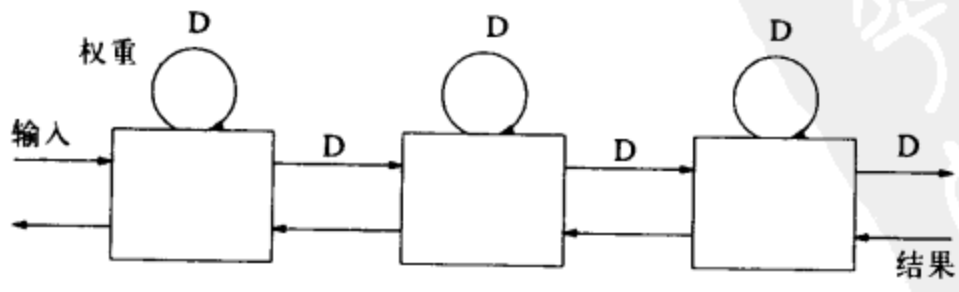


图7-9 设计 F 的框图

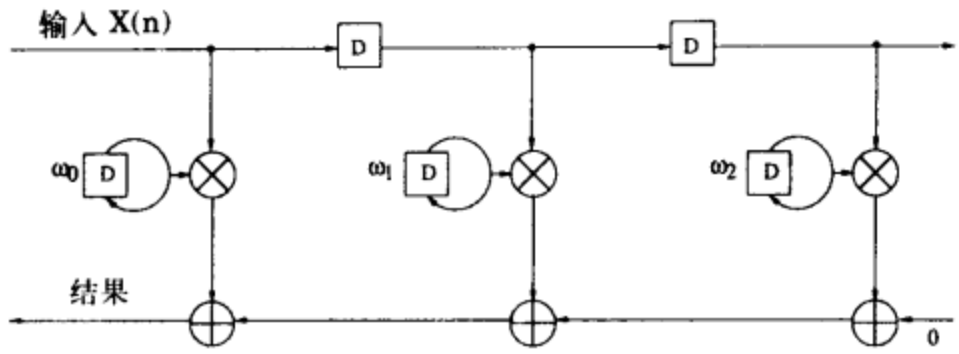


图7-10 设计 F 的低级别实现

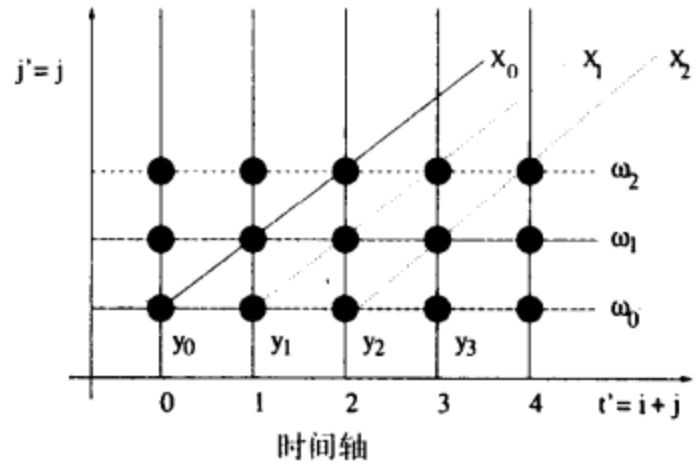


图7-11 设计 F 的空间-时间表示

197

7.3.4 设计 R_1（结果保持，输入和权重反向移动）

设计 R_1 选择的投影矢量、处理器矢量和调度矢量为:

$$\mathbf{d}=\begin{pmatrix}1\\-1\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}1 & 1\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}1 & -1\end{pmatrix} \tag{7-10}$$

以及

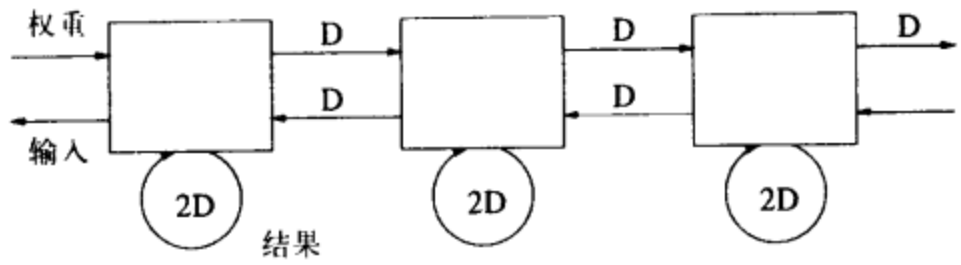
$$\mathbf{s}^T\mathbf{d}=\begin{pmatrix}1 & -1\end{pmatrix}\begin{pmatrix}1\\-1\end{pmatrix}=2$$

所以，硬件利用率为1/2。设计 R_1 的边映射总结为表7-4，设计 R_1 的脉动阵列结构和低级别实现分别如图7-12和图7-13所示。注意输入的方向已经反转以保证 $\mathbf{s}^T\mathbf{e}>0$ 对于所有边都成立。这个操作只在沿着边的计算没有优先约束时才可进行。和设计 B_2 相似，结果在同一个处理器中累加。但是，3个输出是在6个时钟周期内完成。所以，输出的平均速率是每两个时钟周期产生一个输出。

表7-4 设计 R_1 的边映射

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt(1, 0)	1	1
i/p(0, −1)	−1	1
result(1, −1)	0	2

采用相同的分析可推导出另外5种脉动阵列设计。下面几节总结了这些设计所用的投影矢量、处理器矢量和调度矢量，具体细节不再重复。



198

图7-12 设计 R_1 的框图

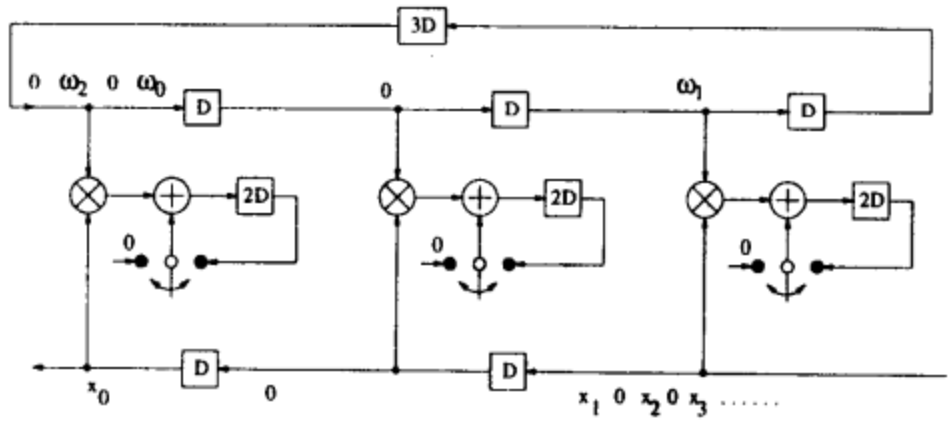


图7-13 设计 R_1 的低级别实现

7.3.5 设计 R_2 和双 R_2（结果保持，输入和权重同方向但不同速度移动）

设计 R_2 和双 R_2 的投影矢量和处理器矢量相同:

$$\mathbf{d} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad \mathbf{p}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} \tag{7-11}$$

设计 R_2 和双 R_2 的调度矢量不同，分别为:

$$R_2: \ \mathbf{s}^T = \begin{pmatrix} 2 & 1 \end{pmatrix}; \qquad 双\ R_2: \ \mathbf{s}^T = \begin{pmatrix} 1 & 2 \end{pmatrix} \tag{7-12}$$

这两个设计的硬件利用率相同，都为$HUE = 1$。边映射总结为表7-5。注意设计双 R_2 的结果边已经反转以保证 $\mathbf{s}^T\mathbf{e} \geqslant 0$。

表7-5 设计 R_2 和双 R_2 的边映射

R_2			Dual R_2		
$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$	$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt(1, 0)	1	2	wt(1, 0)	1	1
i/p(0, 1)	1	1	i/p(0, 1)	1	2
result(1, −1)	0	1	result(−1, 1)	0	1

199

7.3.6 设计 W_1（权重保持，输入和结果反向移动）

投影矢量、处理器矢量和调度矢量为:

$$\mathbf{d} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ \mathbf{p}^T = \begin{pmatrix} 0 & 1 \end{pmatrix}, \ \mathbf{s}^T = \begin{pmatrix} 2 & 1 \end{pmatrix} \tag{7-13}$$

硬件利用率为$HUE = 1/2$。边映射表如表7-6所示。

表7-6 设计 W_1 的边映射

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt (1, 0)	0	2
i/p (0, 1)	1	1
result (1, −1)	−1	1

7.3.7 设计 W_2 和双 W_2（权重保持，输入和结果同方向但不同速度移动）

设计 W_2 和双 W_2 的投影矢量和处理器矢量相同；调度矢量不同，值如下：

$$\mathbf{d}=\begin{pmatrix}1\\0\end{pmatrix},\ \mathbf{p}^T=(0\ \ 1)$$
$$W_2:s^T=(1\ \ 2);\qquad 双\ W_2:\mathbf{s}^T=(1\ \ -1) \tag{7-14}$$

两种设计的硬件利用率都等于1。边映射总结为表7-7。注意设计双 W_2 的输入边和结果边都被反转以保证 $\mathbf{s}^T\mathbf{e}>0$。

表7-7 设计 W_2 和双 W_2 的边映射

W_2			双 W_2		
$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$	$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt(1, 0)	0	1	wt(1, 0)	0	1
i/p(0, 1)	1	2	i/p(0, −1)	−1	1
result(−1, 1)	1	1	result(1, −1)	−1	2

200

7.3.8 应用变换的关联脉动设计

用同样的投影矢量和处理器空间矢量，但不同的调度矢量得到的FIR脉动结构可通过变换互相推导，例如边反转、结合、减速、重定时和流水线。边反转变换表示如果沿着某有向的边没有优先约束，则在依赖图中反转边的方向。结合可用于多个元素累加时改变加法的顺序，也就是 $(a+b)+c=a+(b+c)$。读者可根据第3章和第4章获得其他变换的定义。

可以发现设计 F 可通过对设计 B_1 应用割集重定时得到，如图7-14所示；设计 W_1 可由设计 F 先减速为原来的二分之一，再应用割集重定时得到，等等。这些脉动阵列间的互相推导将作为本章末的习题。

7.4 调度矢量的选择

对任何指定的投影矢量、处理器矢量和调度矢量，都可用线性映射技术来设计脉动阵列。本节将讨论用调度不等式选择可行的调度矢量的方法。基于选出的调度矢量 $\mathbf{s}^T$，可根据方程式（7-3）和（7-2）选择投影矢量 $\mathbf{d}$ 和处理器空间矢量 $\mathbf{p}^T$，从而得到所要的脉动阵列。

7.4.1 基于调度不等式选择 $\mathbf{s}^T$

分析依赖关系 $X\rightarrow Y$，

$$X:\ I_x=\begin{pmatrix}i_x\\j_x\end{pmatrix}\longrightarrow Y:\ I_y=\begin{pmatrix}i_y\\j_y\end{pmatrix} \tag{7-15}$$

其中 I_x 和 I_y 分别是节点 X 和节点 Y 的标识。对应该依赖关系的调度不等式定义为:

$$S_y \geqslant S_x + T_x \tag{7-16}$$

其中 T_x 是计算节点 X 的时间，而 S_x、S_y 分别是节点 X、Y 的调度时间。调度方程可分如下两类:

(1) 线性调度，其中:

$$\begin{aligned} S_x &= \mathbf{s}^T I_x = \begin{pmatrix} s_1 & s_2 \end{pmatrix} \begin{pmatrix} i_x \\ j_x \end{pmatrix} \\ S_y &= \mathbf{s}^T I_y = \begin{pmatrix} s_1 & s_2 \end{pmatrix} \begin{pmatrix} i_y \\ j_y \end{pmatrix} \end{aligned} \tag{7-17}$$

(2) 仿射调度，其中:

$$\begin{aligned} S_x &= \mathbf{s}^T I_x + \gamma_x = \begin{pmatrix} s_1 & s_2 \end{pmatrix} \begin{pmatrix} i_x \\ j_x \end{pmatrix} + \gamma_x \\ S_y &= \mathbf{s}^T I_y + \gamma_y = \begin{pmatrix} s_1 & s_2 \end{pmatrix} \begin{pmatrix} i_y \\ j_y \end{pmatrix} + \gamma_y \end{aligned} \tag{7-18}$$

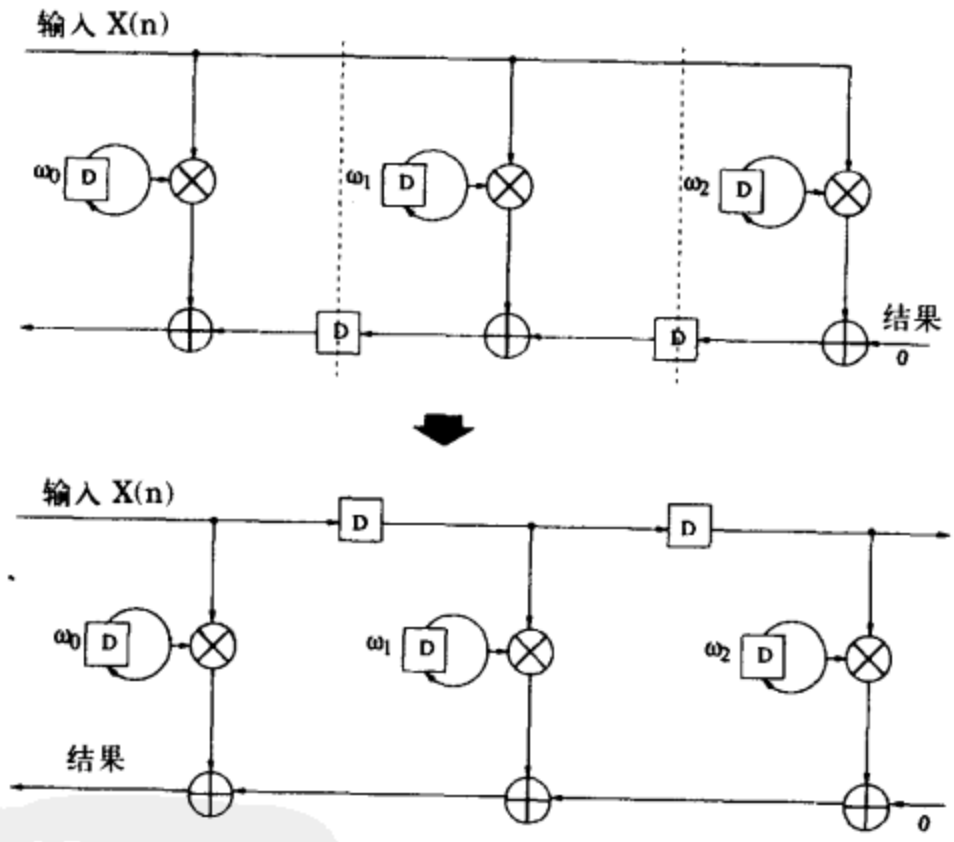


图7-14 根据设计B_1用割集重定时推导的设计F

应用前面的定义，仿射调度的调度方程可改写为:

$$\mathbf{s}^T I_y + \gamma_y \geqslant \mathbf{s}^T I_x + \gamma_x + T_x \tag{7-19}$$

注意线性调度的调度方程可通过设置 γ_x 和 γ_y 等于零得到。定义从节点 X 到节点 Y 的边为 $\mathbf{e}_{x-y} = I_y - I_x$。则边的调度不等式表示如下:

201 ~ 202

$$\mathbf{s}^T \mathbf{e}_{x-y} + \gamma_y - \gamma_x \geqslant T_x \tag{7-20}$$

因此，对依赖图的每一条基本边，可得到一个调度不等式，解这些不等式即可得到调度矢量 $\mathbf{s}^T$。所以调度矢量选择由2个步骤组成:

(1) 找出所有基本边。可利用简化依赖图（reduced dependence graph, RDG）得到基本边，简化依赖图则利用相应问题的规则迭代算法(regular iterative algorithm, RIA)描述构造。这两个方面都将在下面的小节中详细讨论。

(2) 根据公式（7-20）建立调度不等式，且解之得到可用的 $\mathbf{s}^T$。

7.4.2 RIA描述

本节介绍规则迭代算法（RIA），并举例说明用RIA建立RDG的方法。

RIA有两种标准形式，定义如下:

- 如果所有方程的输入下标都相同，称该RIA为标准输入RIA形式;
- 如果所有输出下标，即左侧下标都相同，称该RIA为标准输出RIA形式。

应用同样的FIR滤波器例子，RIA描述为:

$$\begin{aligned} W(i+1,j) &= W(i,j) \\ X(i,j+1) &= X(i,j) \\ Y(i+1,j-1) &= Y(i,j)+W(i+1,j-1)X(i+1,j-1) \end{aligned} \tag{7-21}$$

显然该FIR滤波器问题不能表示为标准输入RIA形式。但可以表示为标准输出RIA描述，如公式（7-22）所示。

$$\begin{aligned} W(i,j) &= W(i-1,j) \\ X(i,j) &= X(i,j-1) \\ Y(i,j) &= Y(i-1,j+1)+W(i,j)X(i,j) \end{aligned} \tag{7-22}$$

根据这些方程，可得到图7-15的RDG。

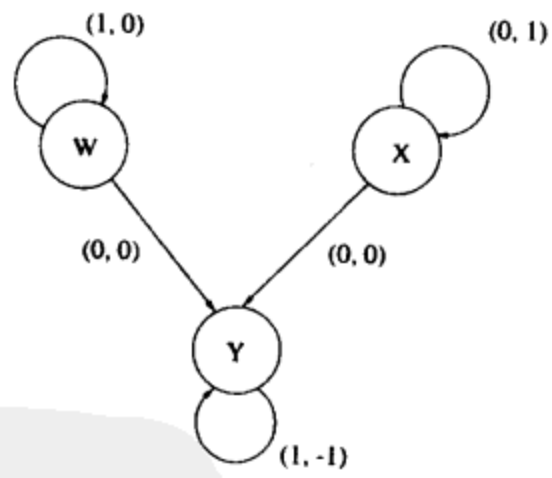


图7-15 FIR滤波器例子的简化依赖图

7.4.3 应用RDG的调度矢量和脉动阵列设计

本节分析应用RDG建立调度不等式，应用调度不等式选择调度矢量以及应用调度矢量的脉动映射。具体方法将用上面的FIR滤波器例子来说明。由于这种公式表示的一般性，它适用于各种操作的不同的计算时间。

例7.4.1 假设进行乘法、加法和通信的时间如下:

$$T_{mult}=5,\quad T_{add}=2,\quad T_{com}=1$$

回顾依赖图中一条边的调度不等式为:

$$\mathbf{s}^T\mathbf{e} + \gamma_y - \gamma_x \geqslant T_x$$

其中：

$$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}$$

图7-15的RDG中有5条边。每条边的调度不等式如下：

$$W \to Y: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \gamma_y - \gamma_w \geqslant 0$$

$$X \to X: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad s_2 + \gamma_x - \gamma_x \geqslant 1$$

$$W \to W: \quad \mathbf{e} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad s_1 + \gamma_w - \gamma_w \geqslant 1$$

$$X \to Y: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \gamma_y - \gamma_x \geqslant 0$$

203 ~ 204

$$Y \to Y: \quad \mathbf{e} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad s_1 - s_2 + \gamma_y - \gamma_y \geqslant 5 + 2 + 1$$

对于线性调度，$\gamma_x = \gamma_y = \gamma_\omega = 0$。简化这些方程可得：

$$s_1 \geqslant 1, \quad s_2 \geqslant 1, \quad s_1 - s_2 \geqslant 8$$

所以，其中一组解为：

$$s_2 = 1, s_1 = 8 + 1 = 9 \Rightarrow \mathbf{s}^T = (9, 1) \tag{7-23}$$

现在，选择 $\mathbf{d} = (1, -1)$ 令 $\mathbf{s}^T\mathbf{d} \neq 0$ 并选择 $\mathbf{p}^T$ 令 $\mathbf{p}^T\mathbf{d} = 0$。选择 $\mathbf{p}^T = (1, 1)$。因为：

$$\mathbf{s}^T\mathbf{d} = \begin{pmatrix} 9 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = 8$$

因此 $HUE = 1/8$。现在，可用表7-8对边进行映射。脉动阵列结构如图7-16所示。 ■

表7-8 例7.4.1的边映射

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
wt(1, 0)	1	9
i/p(0, 1)	1	1
result(1, −1)	0	8

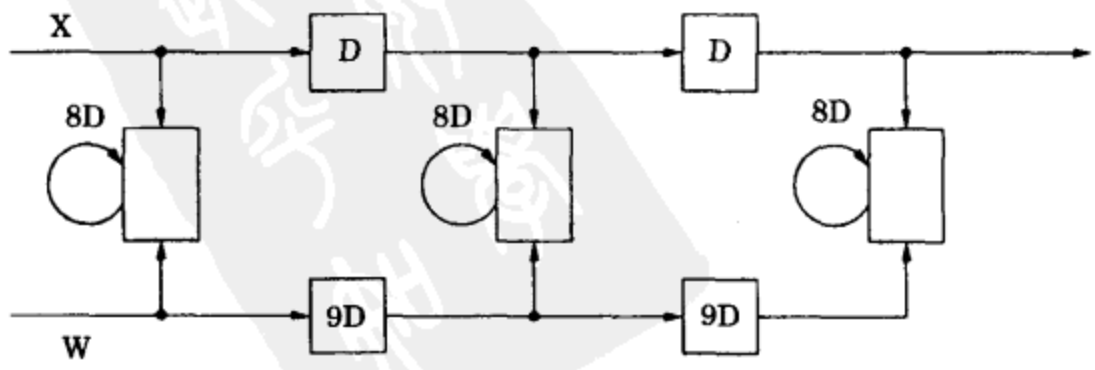


图7-16 例7.4.1的脉动阵列结构

7.5 矩阵乘法与二维脉动阵列设计

本节介绍用于矩阵-矩阵乘法的的脉动阵列设计 [4]。这个问题的依赖图对应一个三维空间表示。线性投影将用于设计矩阵间乘法的二维脉动阵列。 205

给定两个矩阵 A 和 B，可用 $C = AB$ 表示它们的积，其中 A、B 和 C 都是 $n \times n$ 矩阵。对于 $n = 2$，有:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

这些方程可用图7-17所示的空间表示法来表示。

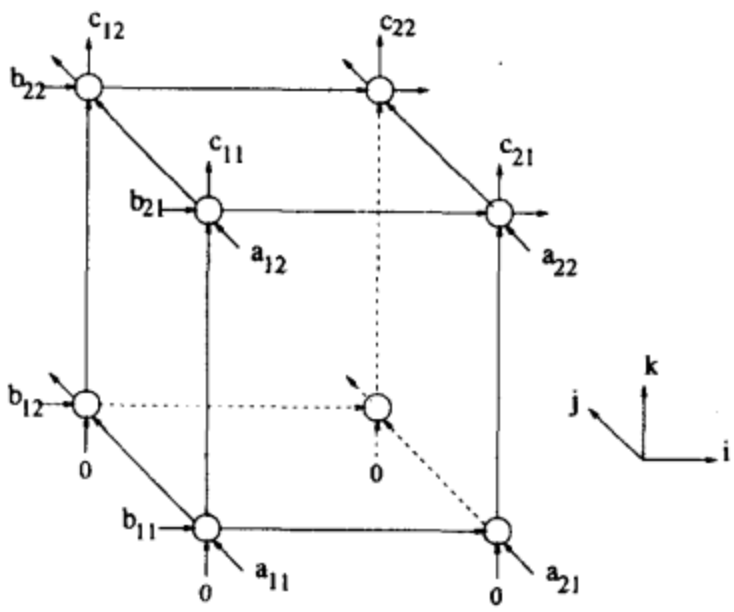


图7-17 矩阵积计算的脉动阵列结构

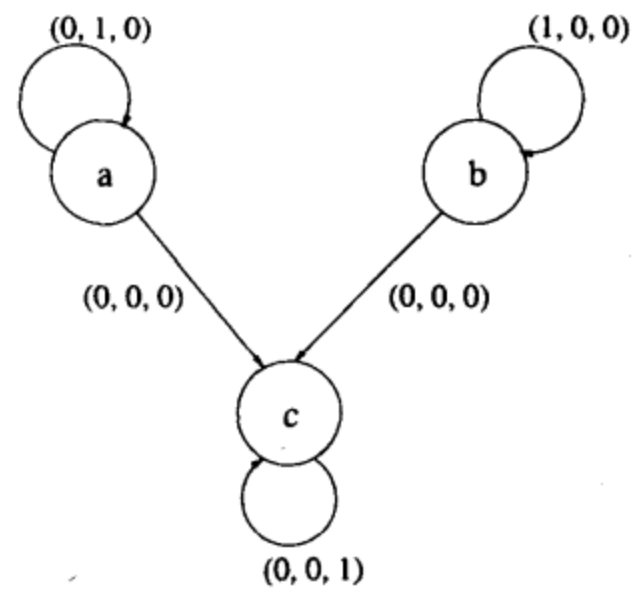


图7-18 矩阵积计算的简化依赖图

根据此空间图，迭代可以写成如下的标准输出RIA形式:

$$\begin{aligned} a(i,j,k) &= a(i,j-1,k) \\ b(i,j,k) &= b(i-1,j,k) \\ c(i,j,k) &= c(i,j,k-1) + a(i,j,k)b(i,j,k) \end{aligned}$$

对应的RDG如图7-18所示。再针对RDG的每条边都应用调度不等式 $\mathbf{s}^T\mathbf{e} + \gamma_y - \gamma_x \geqslant T_x$，且假设 $T_{mult\text{-}add} = 1$ 以及 $T_{com} = 0$，有:

$$a \to a: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad s_2 \geqslant 0$$

$$b \to b: \quad \mathbf{e} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad s_1 \geqslant 0$$

$$c \to c: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad s_3 \geqslant 1$$

$$a \to c: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \gamma_c - \gamma_a \geqslant 0$$

$$b \to c: \quad \mathbf{e} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \gamma_c - \gamma_b \geqslant 0$$

对于线性调度，$\gamma_a = \gamma_b = \gamma_c = 0$。考虑如下的可能解:

• 解1:

206 ~ 207

$$\mathbf{s}^T = (1,1,1), \quad \mathbf{d} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad \mathbf{P}^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{P}^T\mathbf{d} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\mathbf{s}^T\mathbf{d} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 1$$

所以，HUE为1。依赖图的边映射总结为表7-9。该设计对应的二维脉动阵列结构如图7-19所示。

表7-9 矩阵间乘法解1和解2的边映射

	解 1		解 2	
$\mathbf{e}^T$	$\mathbf{P}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$	$\mathbf{P}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
a(0, 1, 0)	(0, 1)	1	(0, 1)	1
b(1, 0, 0)	(1, 0)	1	(1, 0)	1
c (0, 0, 1)	(0, 0)	1	(1, 1)	1

• 解2: 考虑如下的投影矢量、调度矢量和处理器空间矢量:

$$\mathbf{s}^T = (1,1,1), \quad \mathbf{d} = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \quad \mathbf{P}^T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

根据这个设计，

$$\mathbf{P}^T\mathbf{d} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\mathbf{s}^T\mathbf{d} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = 1$$

所以，$HUE = 1$。依赖图的边映射总结为表7-9。对应的二维脉动阵列设计如图7-20所示。

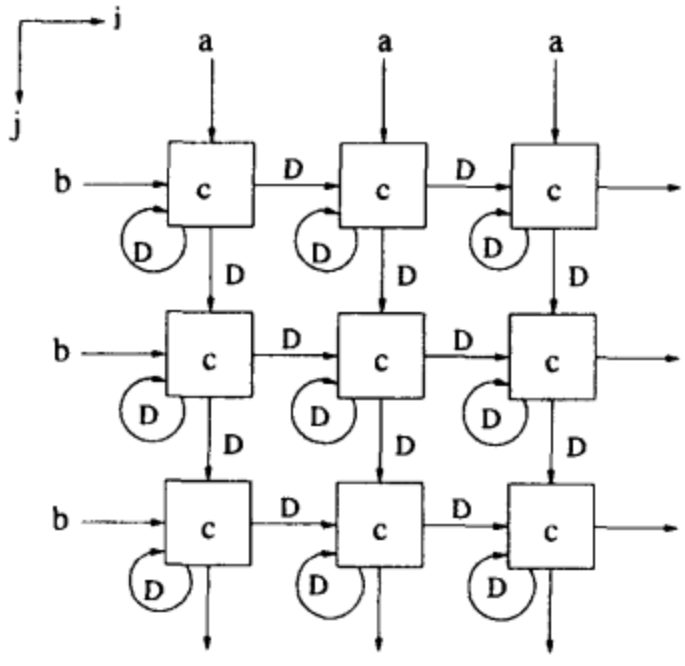


图7-19 S. Y. Kung的二维脉动阵列[3]

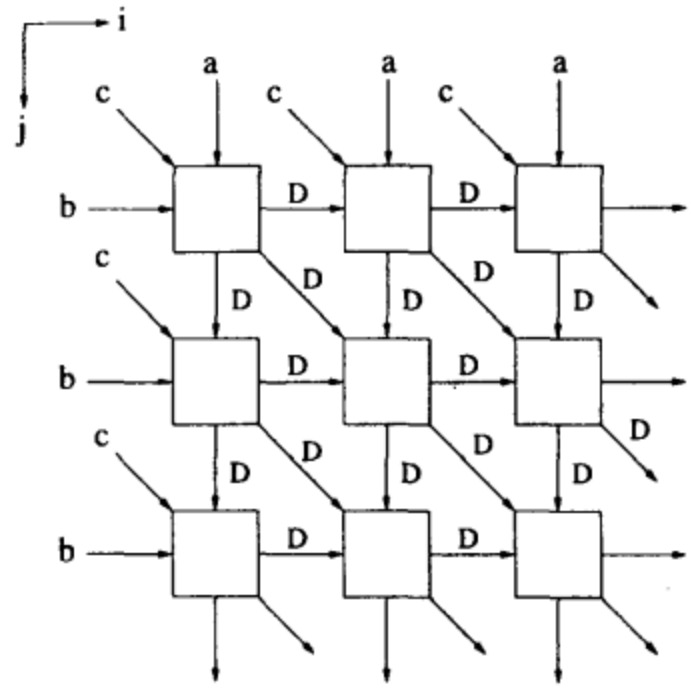


图7-20 Weiser-Davis的二维脉动阵列[4]

下面是其他一些解:

- 解3:

$$\mathbf{s}^T = (1,1,1),\quad \mathbf{d} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix},\quad \mathbf{P}^T = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

这组解可导出*Schreiher-Rao*二维脉动阵列。

- 解4:

$$\mathbf{s}^T = (1,1,1),\quad \mathbf{d} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},\quad \mathbf{P}^T = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

208 ~ 209

这组解可导出*Kung-Leiserson*脉动阵列。

- 解5:

$$\mathbf{s}^T = (1,2,1),\quad \mathbf{d} = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix},\quad \mathbf{P}^T = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

- 解6:

$$\mathbf{s}^T = (1,1,1),\quad \mathbf{d} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix},\quad \mathbf{P}^T = \begin{pmatrix} 1 & -1 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

- 解7:

$$\mathbf{s}^T = (1,2,1),\quad \mathbf{d} = \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix},\quad \mathbf{P}^T = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

7.6 包含延迟的空间表示脉动设计

脉动映射方法可改进后用于空间表示的延时元件[9]。在这类空间表示中的优先约束包含迭代间和迭代内优先约束。这种方法原理还可用于多级别脉动映射（例如矩阵间乘法的一维脉动阵列设计）。

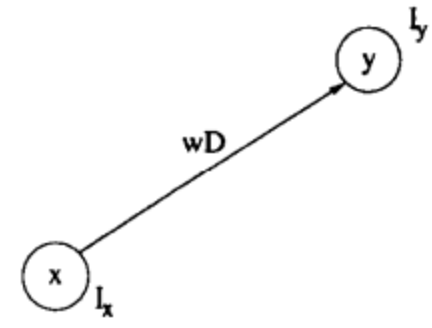


图7-21 有延时依赖图

分析图7-21所示的依赖图，其中 I_x、I_y 分别是节点 x 和 y 的标识，ω 为从节点 x 到节点 y 延时元件的个数。严格说来，有延时的依赖图应该叫做数据流图。但这里我们仍然称之为依赖图。在推导有延时依赖图每条边的调度约束之前，需要定义如下参数:

210

- N': 映射到一个处理器的节点个数。
- $\mathbf{s}^T\mathbf{d}$: 如果一个处理器在 $\mathbf{s}^T I$ 时间计算节点 x，则在 $\mathbf{s}^T(I+d)$ 时间计算离x为$\mathbf{d}$的节点。

现在，假设节点 x 第 l 次迭代在 T_l 时间进行，则节点 x 的第 $(l+1)$ 次迭代在 $T_l + N'|\mathbf{s}^T\mathbf{d}|$ 时间执行。因此，迭代周期为 $N'|\mathbf{s}^T\mathbf{d}|$。节点 x 第 l 次迭代调度时间步骤为 $\mathbf{s}^T I_x + lN'|\mathbf{s}^T\mathbf{d}|$，节点 y 第 $(l+w)$ 次迭代调度时间步骤为 $\mathbf{s}^T I_y + (l+w)N'|\mathbf{s}^T\mathbf{d}|$。从而导出调度不等式:

$$\mathbf{s}^T I_y + (l+w)N'|\mathbf{s}^T\mathbf{d}| \geqslant \mathbf{s}^T I_x + lN'|\mathbf{s}^T\mathbf{d}| + T_x$$

化简后得到:

$$\mathbf{s}^T\mathbf{e} + wN'|\mathbf{s}^T\mathbf{d}| \geqslant T_x \tag{7-24}$$

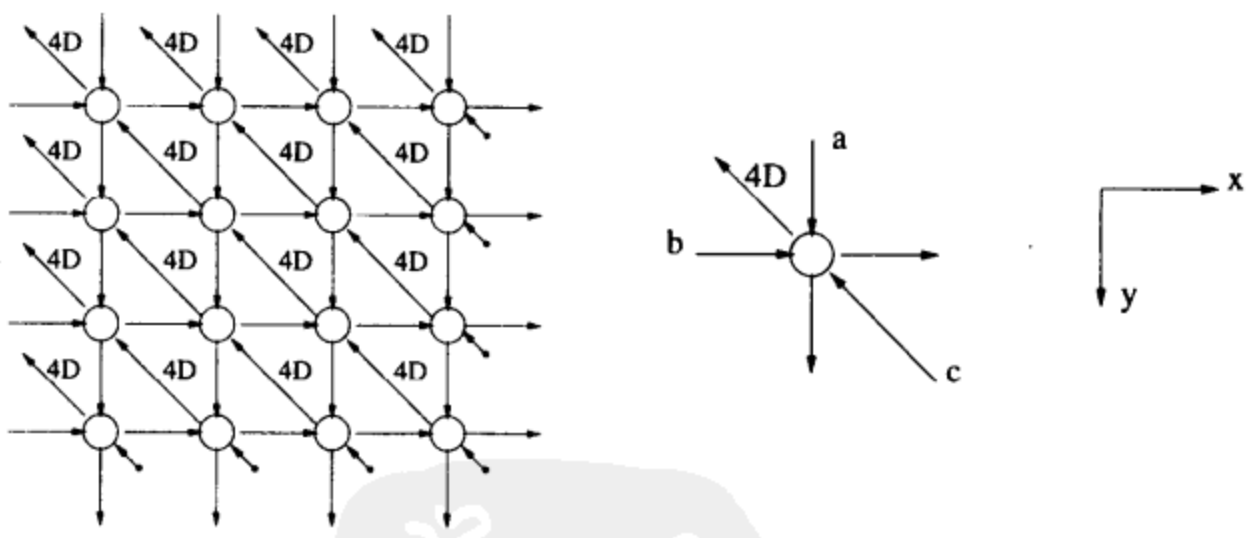


图7-22 依赖图及其一个单元

211 **例7.6.1** 分析图7-22所示的依赖图的一个单元。假设 $T_x = 1$ 并对三条基本边应用调度不等式，有:

$$\mathbf{e_a} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} s_1 & s_2 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} + 0 \cdot N'\left|S^T\mathbf{d}\right| \geqslant 1 \Rightarrow s_2 \geq 1$$

$$\mathbf{e_b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad s_1 \geqslant 1$$

$$\mathbf{e_c} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad -s_1 - s_2 + 4N'(s_1 d_1 + s_2 d_2) \geqslant 1$$

假设：

$$\mathbf{d} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

所以，同一列的所有任务都映射到同一个处理器。进一步假设依赖图每列有4个节点，即 $N' = 4$。将这些值代入前述不等式，有：

$$\begin{aligned} -s_1 - s_2 + 16(s_2) &\geqslant 1 \\ 15s_2 - s_1 &\geqslant 1 \end{aligned}$$

选择 $s_1 = s_2 = 1$，即 $\mathbf{s}^T = (1, 1)$，则$HUE = 1$。选择 $\mathbf{p}^T = (1, 0)$。则基本边的边映射可求得如表7-10。该脉动阵列的框图如图7-23所示。 ■

表7-10 例7.6.1脉动阵列设计的边映射

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e} + wN'\lvert\mathbf{s}^T\mathbf{e}\rvert$
a(0, 1)	0	1
b(1, 0)	1	1
c(−1, −1)	−1	14

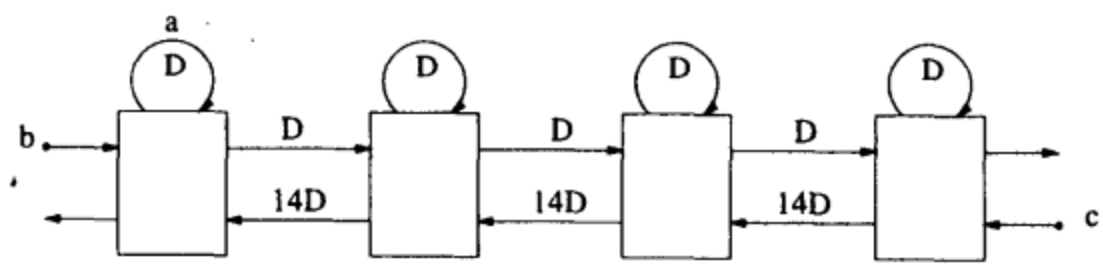


图7-23 FIR滤波器例子的简化依赖图

212

7.7 结论

本章介绍了脉动结构设计方法原理，对于任一种给定的规则迭代算法，通过线性映射或者投影技术，可以设计出多种脉动结构。并以FIR滤波器和矩阵间乘法为例说明了该方法原理。关于多级别脉动映射理论，读者可参考[10]。

7.8 习题

1. 对于图7-24所示的依赖图：
   (a) 下面各组调度和投影中哪组是可行的？
      i. $\mathbf{s} = [1\ 0]^T$，$\mathbf{d} = [1\ 0]^T$；
      ii. $\mathbf{s} = [1\ 2]^T$，$\mathbf{d} = [2\ -1]^T$；
      iii. $\mathbf{s} = [1\ 1]^T$，$\mathbf{d} = [1\ 0]^T$；
      iv. $\mathbf{s} = [1\ -2]^T$，$\mathbf{d} = [1\ 0]^T$。
   (b) 推导每个可行组的投影脉动阵列。
2. 画出设计 R_1 的空间-时间映射。
3. 用一种或者多种变换，如边反转、结合、减速、重定时和流水线，根据设计 B_2 推导

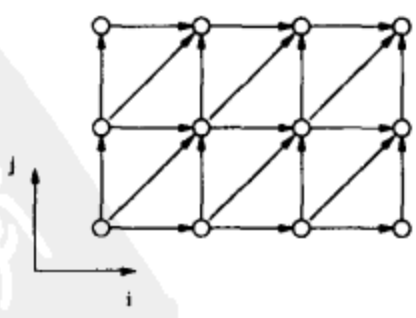


图7-24 习题1的图

设计 R_1。

4. 对 R_2 和双 R_2 两个设计，画出它们的结构框图和空间-时间映射。
5. 用一种或者多种变换，如边反转、结合、减速、重定时和流水线，给出由 R_2 推导双 R_2 的所有步骤。

213

6. 画出设计 W_1 的空间-时间表示和脉动结构，其中 $\mathbf{d}=\begin{bmatrix}1\\0\end{bmatrix}$、$\mathbf{p}^T=[0\ 1]$ 和 $\mathbf{s}^T=[2\ 1]$。
7. 根据设计 F 用边反转、结合、减速、重定时和流水线等变换推导设计 W_1。
8. 对设计 W_2 和双 W_2，画出它们的结构框图和空间-时间映射。为什么它们互为对偶结构?
9. 考虑图7-25所示的5阶卷积滤波器的依赖图。假设输入和系数可同时广播到所有处理器（无延时），或者可将延时用于全脉动设计。进一步假设乘-加处理器需要4个时间单位，即每个处理器包含一个4级流水线乘-加法器。
   (a) 写出所有调度矢量元素 $\mathbf{s}^T=[s_1\ \ s_2]$ 必须满足的不等式，选出硬件利用率最高的调度矢量。
   (b) 画出对应迭代矢量 $\mathbf{d}^T=[1\ \ 0]$ 的脉动结构，使用(a)部分选出的调度矢量。该阵列的硬件利用率是多少?
   (c) 对 $\mathbf{d}^T=[1\ \ -1]$ 重复(b)部分。

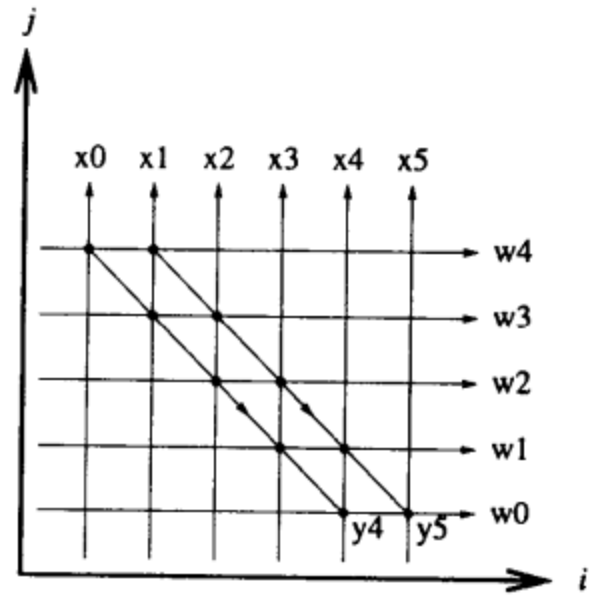


图7-25 习题9的图

10. 本题考虑图7-26所示的依赖图的全流水（即所有互连边至少包含一个延时元件）脉动结构设计。
   (a) 用投影矢量 $\mathbf{d}^T=[0\ \ 1]$ 设计全流水脉动结构，求出处理器置换和调度矢量。要求调度矢量的选择必须使硬件利用率最大。明确给出脉动设计所需的合适的处理单元数目。
   (b) 对投影矢量 $\mathbf{d}^T=[1\ \ 1]$ 重复(a)部分。

214

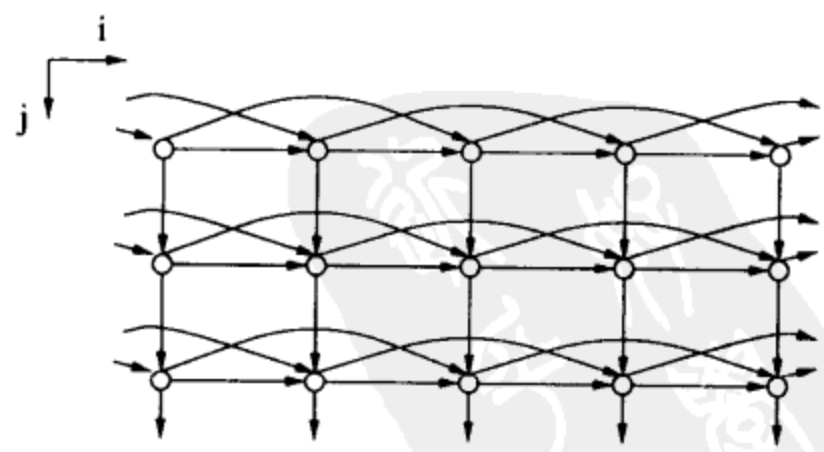


图7-26 习题10的依赖图

11. 矢量量化（Vector Quantization, VQ）常用于语音、图像、视频编码和语音识别中的数据压缩。第1章的第1.2.7小节有关于VQ的概述。本题考虑VQ的搜索过程的脉动结构设计，描述如下:

$$ind_n = (\min_{0 \leqslant j \leqslant N-1} d_j)^{-1} = (\max_{0 \leqslant j \leqslant N-1} \sum_{i=0}^{k-1} (x_i^n y_{ji} + e_j))^{-1} \tag{7-25}$$

其中求逆表示“输出达到最小的标识 ind_n”，n 表示时间步骤。搜索过程还可等效表示为如下的有比较的矩阵-矢量乘法公式[3]:

$$\begin{aligned} \mathbf{D} &= [d_0\ d_1\ \cdots\ d_{N-1}]^T = \mathbf{Y}\mathbf{x} + \mathbf{e} \\ ind_n &= (\mathbf{Max}\{d_i\})^{-1} \end{aligned} \tag{7-26}$$

其中 $\mathbf{Y} = \{y_{ji}\}$ 是 $N \times k$ 矩阵，以第 j 个码矢 y_j^T 为其第 j 行，$\mathbf{x}$为输入矢量且 $\mathbf{e} = [e_0 e_1\ \ldots\ e_{N-1}]^T$（见第1.2.7节）。公式（7-26）计算的依赖图、2个基本处理单元的定义、乘法累加（multiply-accumulate, MAC）和比较/选择见图7-27。输入矢量$\mathbf{x}$沿 i 方向传播，且码矢 y_{ji} 的元素预载到依赖图的 (j, i) 节点。初值 $\{e_j\}$ 从顶行输入到第 j 列。本题假设 $k = 4$ 和 $N = 16$。

(a) 应用依赖图和边的信息，用投影矢量 $\mathbf{d} = [1\ \ 0]^T$、调度矢量 $\mathbf{s} = [1\ \ 1]^T$ 和处理器矢量 $\mathbf{p} = [0\ \ 1]^T$ 推导公式（7-26）计算的脉动结构。假设每个处理器都有局部存储器。这种情况下，局部存储器（码矢的存储单元）应组成重复循环队列（移位寄存器或模余寻址模式的只读存储器(ROM)）画出存储器结构且明确给出每个位置的初始内容。 215

(b) 用投影矢量 $\mathbf{d} = [0\ \ 1]^T$、$\mathbf{s} = [1\ \ 1]^T$ 和 $\mathbf{p} = [1\ \ 0]^T$ 推导对应公式（7-26）计算的另一种脉动结构。注意这个结构需要选择器，给出每个局部存储器位置的内容。

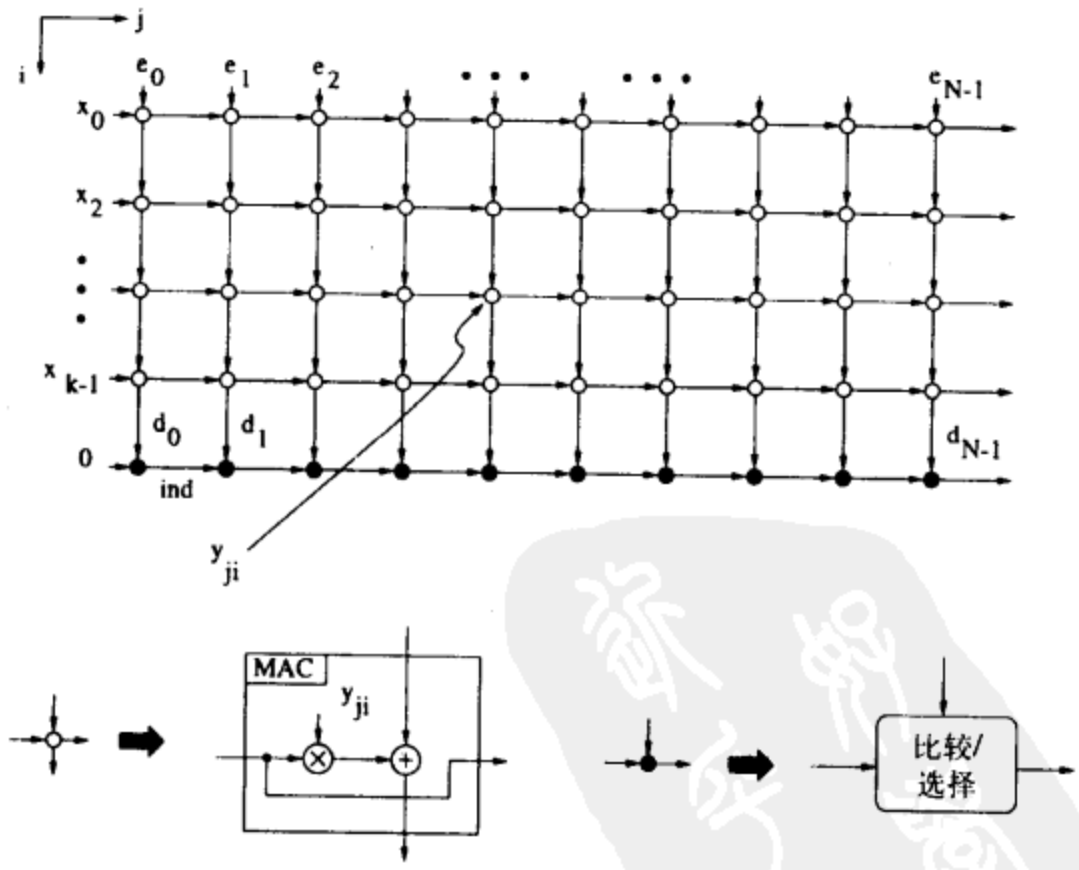


图7-27 第11题矢量量化的依赖图和处理单元

12. 本题关于矩阵-矢量乘法的脉动结构设计，重点是Toeplitz矩阵矢量乘法和它在数字相关器设计上的应用。本题假设每个处理器单元只能访问自己的局部存储器，且矩阵A的元素为常数。

(a) 考虑如下的矩阵-矢量乘法:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{7-27}$$

216

其数据依赖图如图7-28所示，其中系数矩阵**A**的元素 a_{ij} 存于位置 (i, j)。画出矩阵-矢量乘法的空间-时间表示和脉动结构，其中:

$$\mathbf{d}^T = [1\ 0], \quad \mathbf{p}^T = [0\ 1], \quad \mathbf{s}^T = [1\ 1] \tag{7-28}$$

假设存储器局部化到每个处理单元，明确给出局部存储器的内容。

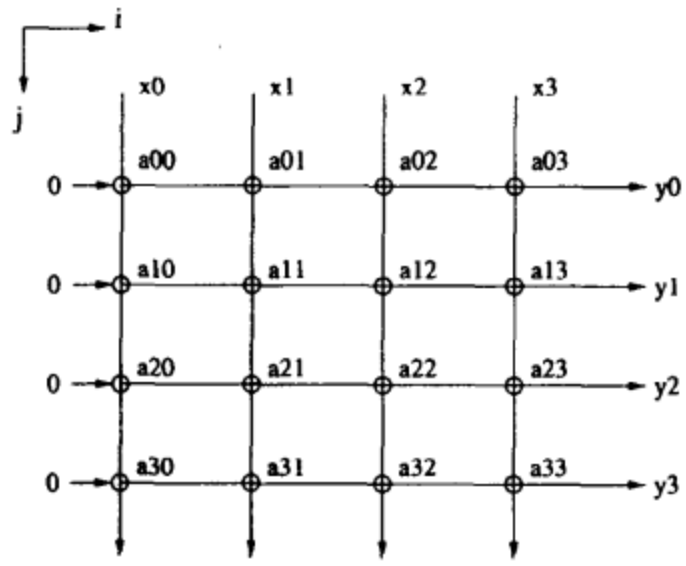


图7-28 第12题(a, b)的矩阵-矢量乘法的依赖图

(b) 数字相关操作给定如下:

$$y_n = \sum_{k=0}^{N-1} x_k a_{n+k} \tag{7-29}$$

其中 $n = 0, 1, 2, \cdots, N-1$。也可写成矩阵-矢量乘法形式，其中矩阵为Toeplitz矩阵，即对角线方向的所有元素都相等。例如一个4级数字相关器可写为:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a_3 & a_2 & a_1 & a_0 \\ a_4 & a_3 & a_2 & a_1 \\ a_5 & a_4 & a_3 & a_2 \\ a_6 & a_5 & a_4 & a_3 \end{bmatrix} \cdot \begin{bmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \tag{7-30}$$

注意矩阵**A**中只有7个不同的元素。将这个Toeplitz矩阵**A**代入图7-28的依赖图且将该数字相关器算法映射到脉动阵列，分别使用:

i. $\mathbf{d}^T = [1\ \ 1]$, $\mathbf{p}^T = [1\ \ -1]$, $\mathbf{s}^T = [1\ \ 0]$;

217

ii. $\mathbf{d}^T = [1\ \ 1]$, $\mathbf{p}^T = [1\ \ -1]$, $\mathbf{s}^T = [0\ \ 1]$;

iii. $\mathbf{d}^T = [1\ \ 1]$, $\mathbf{p}^T = [1\ \ -1]$, $\mathbf{s}^T = [1\ \ 1]$。

画出每个设计的空间-时间表示和脉动结构。比较这三种设计和(a)部分数字相关器设计的硬件复杂度（局部存储单元、处理器单元、延时元件等的数目）以及吞吐量。(提示: (b)部分设计可以使用固定系数乘法器，而如果不考虑Toeplitz矩阵的特殊性质，

(a)部分设计必须用变系数乘法器。)

(c) 本部分要表明，一个算法可用多种依赖图来表示，可用不同投影矢量导出相同脉动结构。数字相关操作也可用与FIR滤波操做相似的方法描述，即用图7-29所示的依赖图。根据这个依赖图，对于(a)部分和(b)部分得到的数字相关器的脉动设计，求出投影矢量 **d**、处理器矢量 **p** 和调度矢量 **s**。

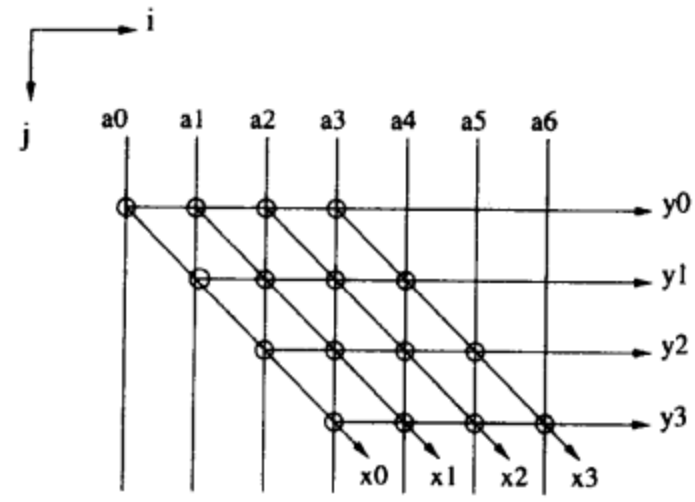


图7-29 习题12(c)的依赖图

(d) 如果使用全局存储器，即处理单元公用的存储器，相对于数字相关器而言，(b)部分设计与(a)部分设计相比，其优点和缺点是什么？

13. 对于以下迭代矢量画出矩阵间乘法的高级结构（行矢量表示迭代矢量的转置）。不用画出完整的结构，只要给出处理器间的数据流方向。对于每个阵列，选择达到最好迭代周期的调度矢量。每个结构的硬件利用率是多少？ 218

(a) [2 1 0];

(b) [2 −1 0];

(c) [2 −1 1]。

14. 离散小波变换(discrete wavelet transform, DWT)是一种有限能量序列的离散时间、离散数值范围分解，用于表示随时间变化的能量频谱（见第1.2.10节）。DWT可作为一系列卷积和降抽样来递归计算。在每个倍频程级别 j，输入序列 $S_{j-1}(n)$ 分别输入到系数分别为 $G(n)$ 和 $H(n)$ 的低通滤波器和高通滤波器。倍频程 j 的计算可表示如下：

$$\begin{aligned} S_j(n) &= \sum_k g_k S_{j-1}(2n-k) \\ W_j(n) &= \sum_k h_k S_{j-1}(2n-k) \end{aligned} \tag{7-31}$$

其中 n 为抽样标识，j 为倍频程标识。图7-30给出一个3倍频程DWT的框图。通常，由于降抽样操作，DWT的特征表现为一个不一致的数据依赖结构。[11]中引入了一些非线性变换，首先规则化DWT算法，再将规则化的三维算法变换为二维索引空间（依赖图）。

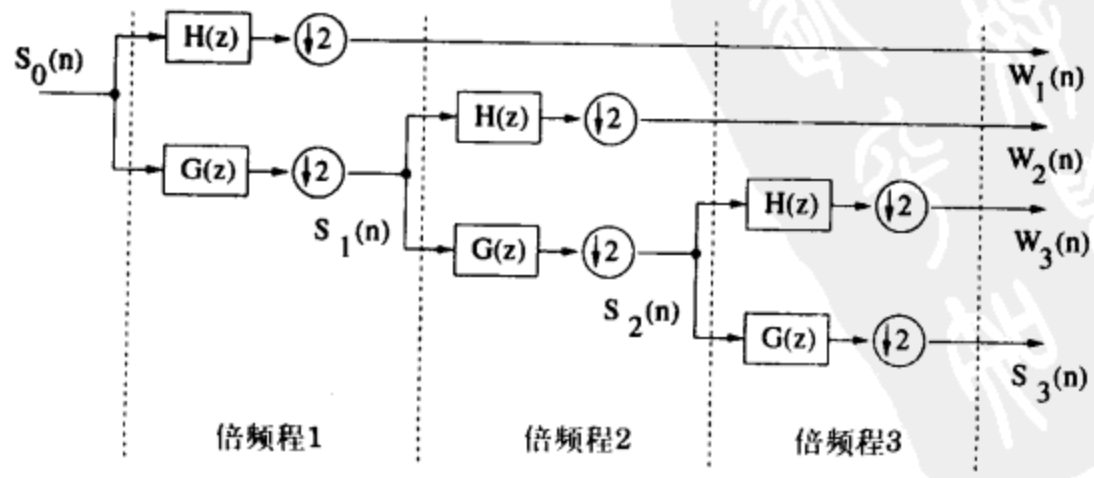


图7-30 DWT的框图

考虑一个3倍频程DWT，其第一、第二、第三倍频程计算的依赖图分别如图7-31、7-32、7-33所示；或者他们也可合并为一个依赖图，如图7-34所示，n 作为沿 x 轴的标识，k 作为沿 y 轴的标识，其中 n 和 k 和公式（7-31）相同。注意第 j-1 倍频程的输出 S 作为第 j 倍频程的输入。

219

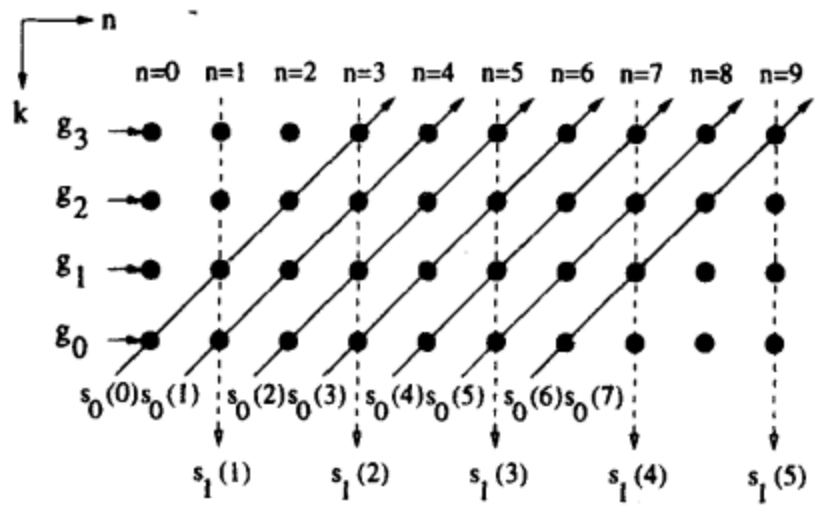


图7-31 第1倍频程的计算的依赖图

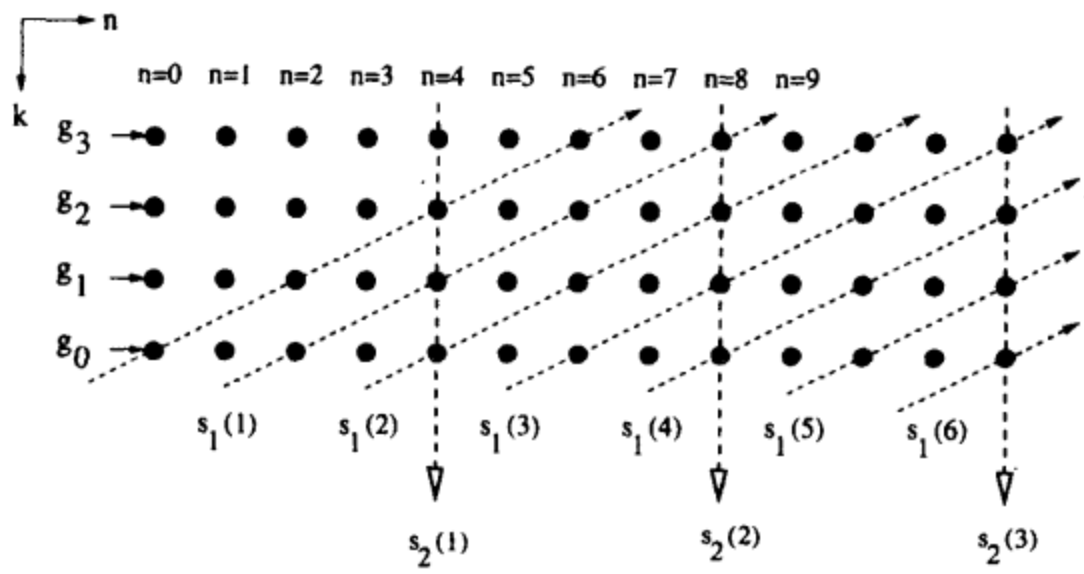


图7-32 第2倍频程的计算的依赖图

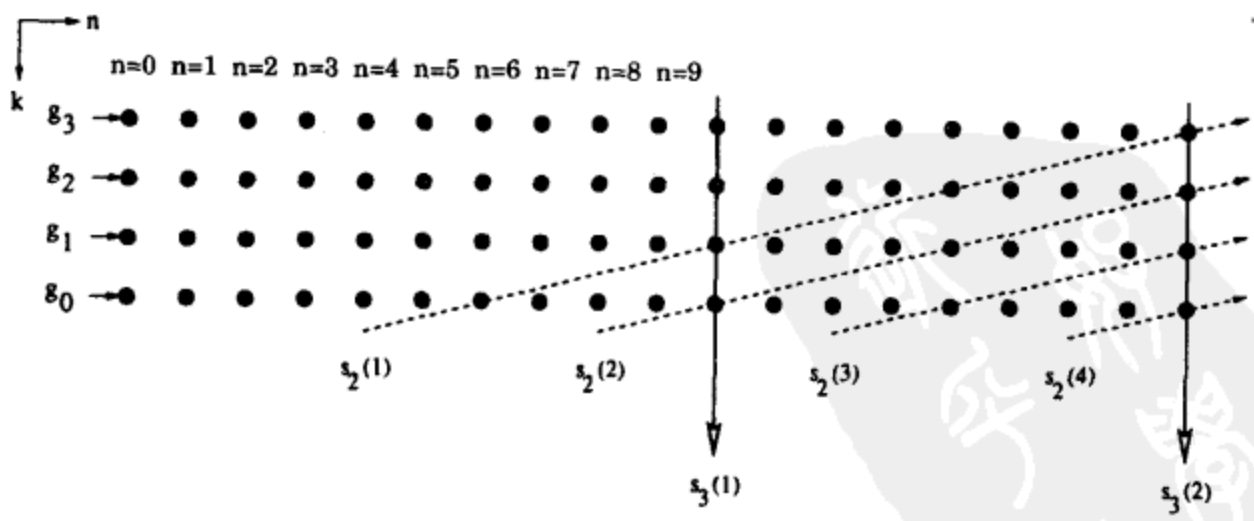


图7-33 第3倍频程的计算的依赖图

(a) 写出 j = 1，2，3 以及 n = 2 ，3的方程 $S_j(n)=\sum_{k=0}^{3} g_k S_{j-1}(2n-k)$。在图7-34的依赖图中定位这些计算。

(b) 第 j (j = 1，2，3) 倍频程的输出边矢量为 $\mathbf{e}=[0\ \ 1]^T$。第1倍频程的输入边矢量

为 $\mathbf{e} = [1\ \ -1]^T$。确定下面矢量的 $\mathbf{e}$:

i. 第2倍频程的输入;

ii. 第3倍频程的输入;

iii. 权重 g_i。

(c) 边矢量 $\mathbf{e} = [1\ \ 0]^T$ 描述了第1倍频程的输出送入第2倍频程的输入的转换; 类似地, $\mathbf{e} = [2\ \ 0]^T$ 描述了第2倍频程的输出送入第3倍频程的输入的转换。根据这个信息以及(b)部分得到的边矢量, 求出满足下面条件的DWT的脉动结构:

$$\mathbf{p} = [0\ 1]^T,\ \mathbf{s} = [1\ 1]^T,\ \mathbf{d} = [1\ 0]^T \tag{7-32}$$

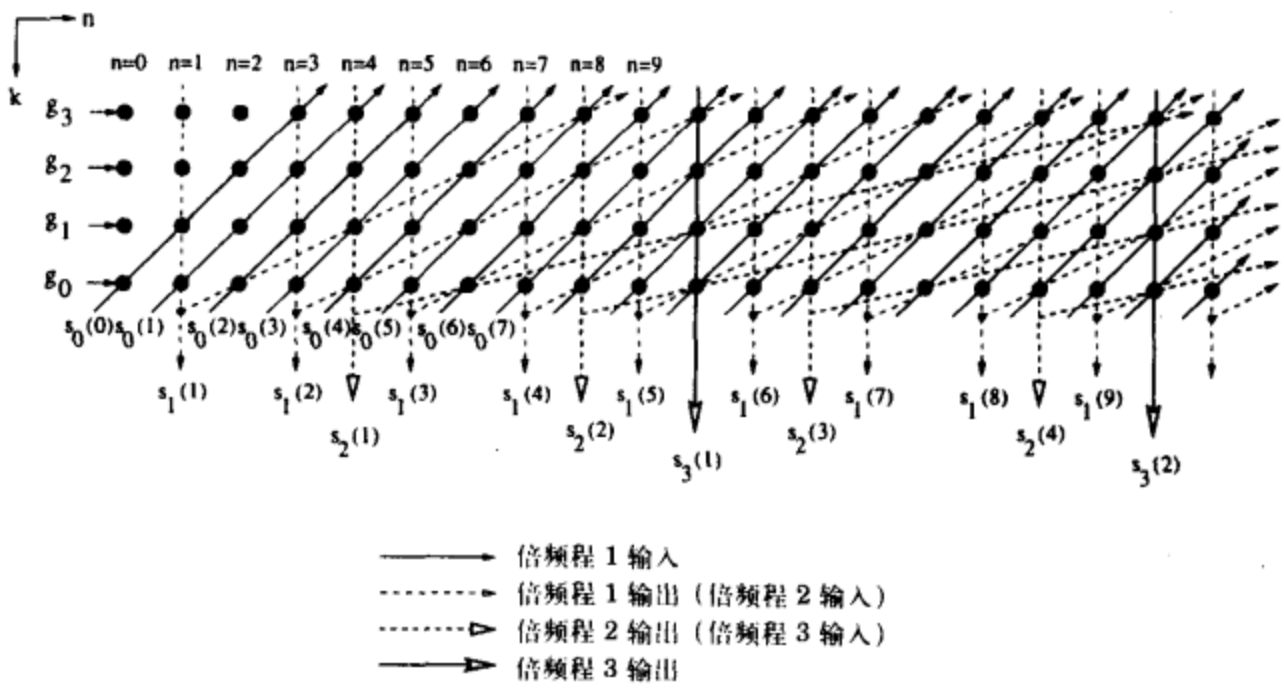


图7-34 第1、2、3倍频程的计算的依赖图

15. 考虑图7-35所示的如下滤波操作计算的空间表示:

$$y(n) = ax(n) + bx(n-2) + cx(n-4) + dx(n-6) \tag{7-33}$$

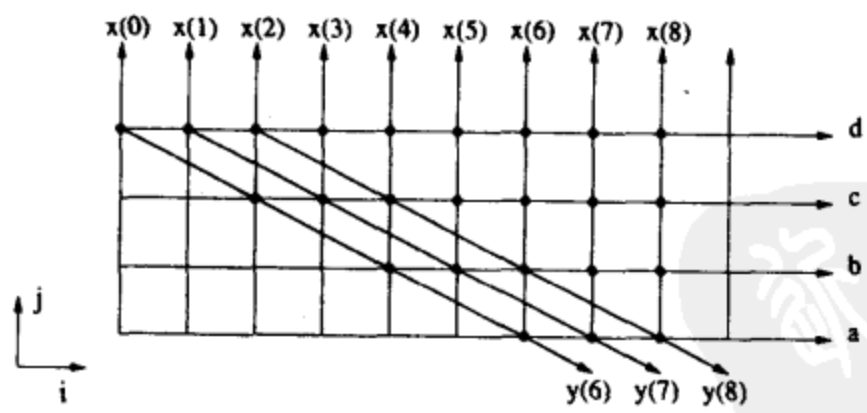


图7-35 第15题的依赖图

(a) 明确导出该算法的一个等价RIA描述。给出简化依赖图且写出如果乘-加操作需要2时间单位的调度不等式。假设输入和权重系数可用广播方式传播。

(b) 选择投影矢量 $\mathbf{d}^T = [1\ \ 0]$, 选择处理器矢量 $\mathbf{p}$。求一组调度不等式的解, 令设计的脉动结构需要最少延时次数。画出脉动结构。 220~221

(c) (b)部分结构的硬件利用率是多少? 如果此结构不能达到100%的硬件利用率, 求

另一种能达到100%硬件效率的脉动结构。(提示：尝试反转某些空间表示的方向是否有效。)

16. 本题要求对一些边包含延时的依赖图设计脉动阵列。考虑图7-36所示的依赖图。

(a) 写出调度矢量元素 $\mathbf{s}^T = [s_1 \ \ s_2]$ 必须满足的所有等式。

(b) 对于迭代矢量 $\mathbf{d}^T = [0 \ \ 1]$，求令硬件利用率最大的 s_1 和 s_2 的值。对于这个调度矢量，求脉动阵列。该阵列的硬件利用率是多少？

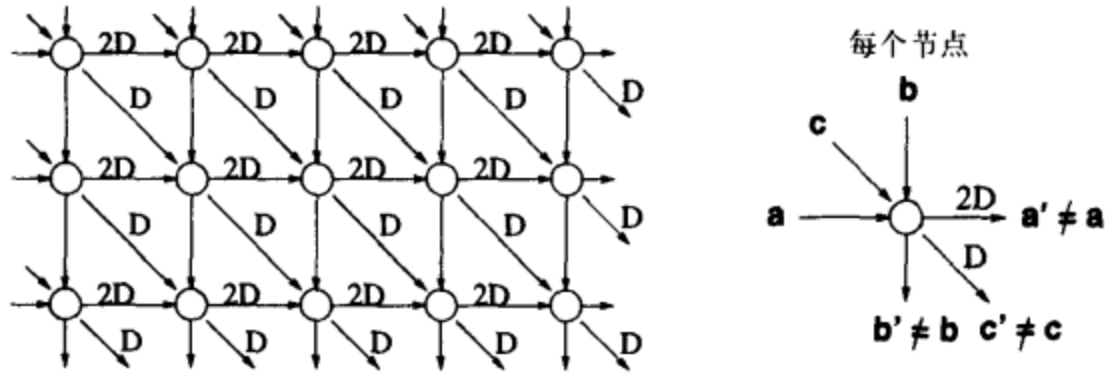


图7-36 第16题图

17. 对 $\mathbf{d}^T = [1 \ \ 1]$ 重复例题7.6.1。

18. 考虑运动估值计算：

$$s(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |x(i,j) - y(i+m, j+n)|, \ 0 \leqslant m,n \leqslant 2p$$
$$v = (m,n)|u, \ u = min_{(m,n)} s(m,n) \tag{7-34}$$

对 $N = 3$ 和 $p = 2$ 应用图7-37和图7-38所示的二维和一维脉动结构[12]。这些结构中的“点”表示延时因素。

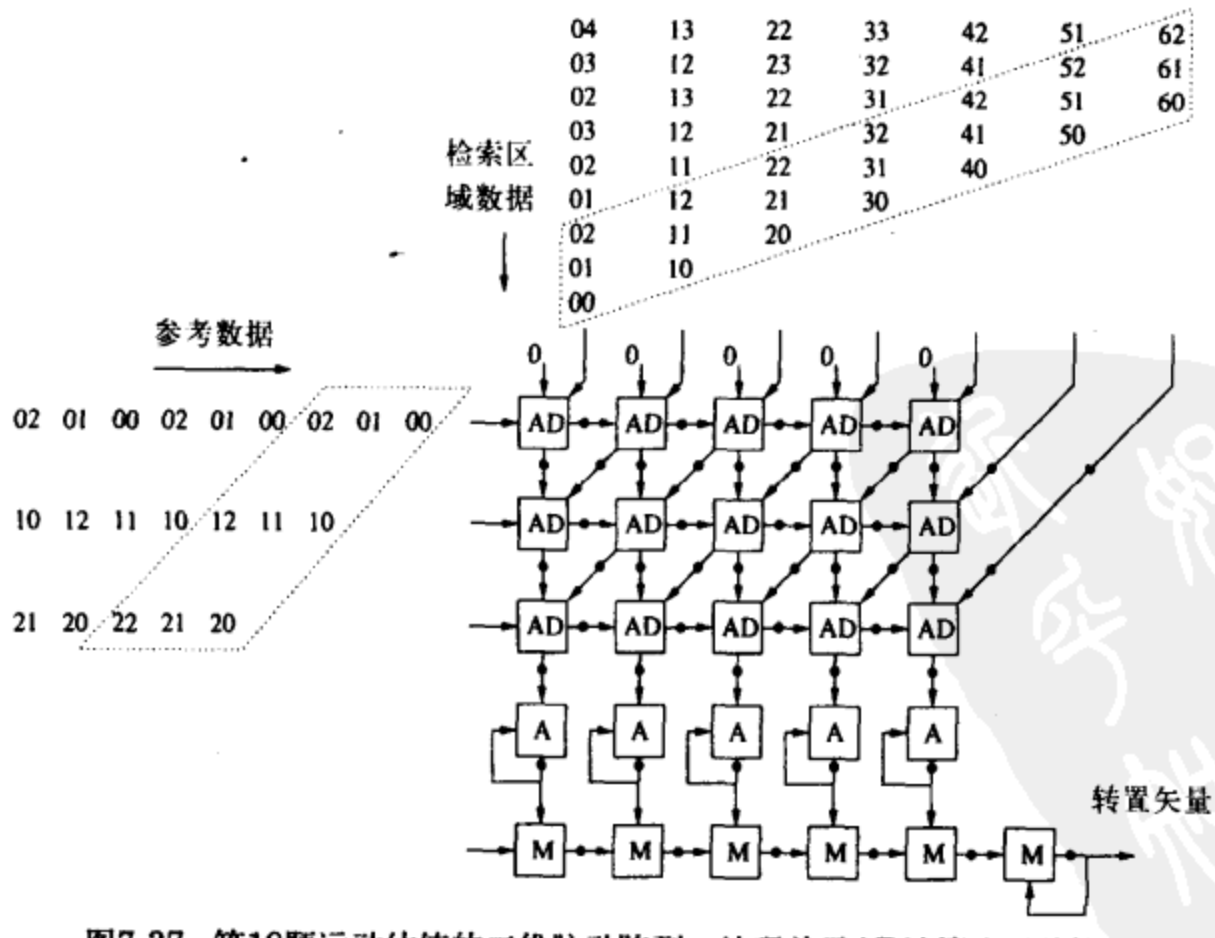


图7-37 第18题运动估值的二维脉动阵列。处理单元AD计算绝对差值；A 计算和；M 比较 $s(m,n)$ 并选择转置矢量索引为 $x(i,j)$ 和 $y(i+m, j+m)$

(a) 验证这些脉动结构的操作。

(b) 每个脉动阵列中计算一次运动矢量需要多少个时钟周期? 222

19. 考虑运动估值计算:

$$s(m,n)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|x(i,j)-y(i+m,j+n)|,\ 0\leqslant m,n\leqslant 2p \tag{7-35}$$

$$v=(m,n)|u,\ u=min_{(m,n)}s(m,n)$$

对 $N=3$ 和 $p=2$ 的二维和一维脉动结构分别如图7-39和图7-40所示[12]。这些结构中的“点”表示延时元件。

(a) 验证这些脉动结构的操作。

(b) 每个脉动阵列中计算一次运动矢量需要多少个时钟周期?

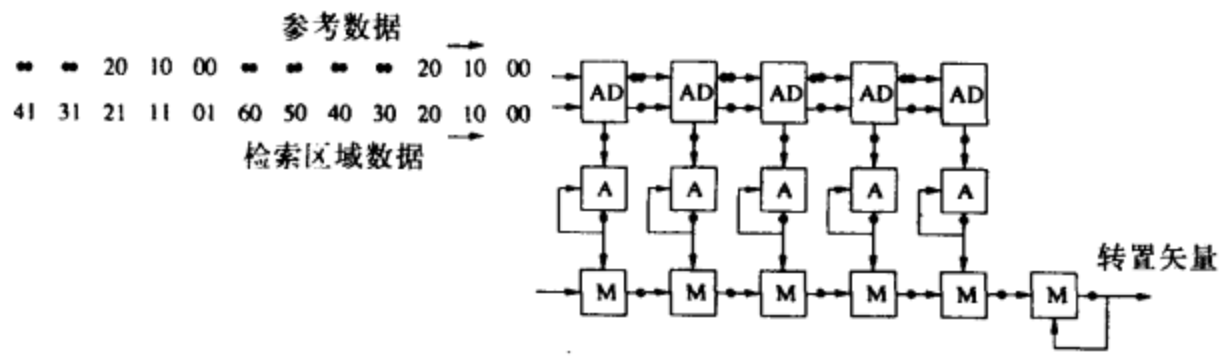


图7-38 第18题运动估值的一维脉动阵列

223 ~ 224

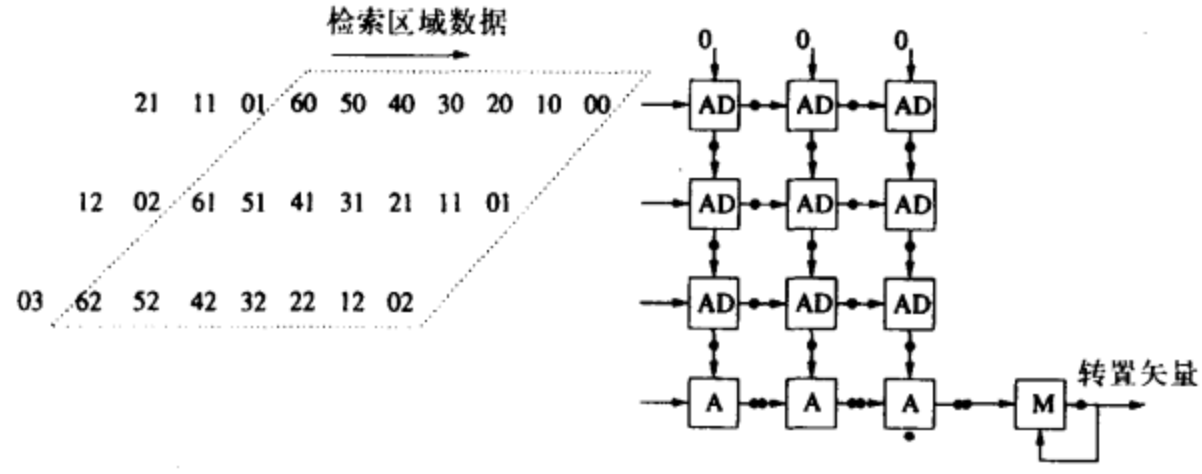


图7-39 第19题运动估值的二维脉动阵列

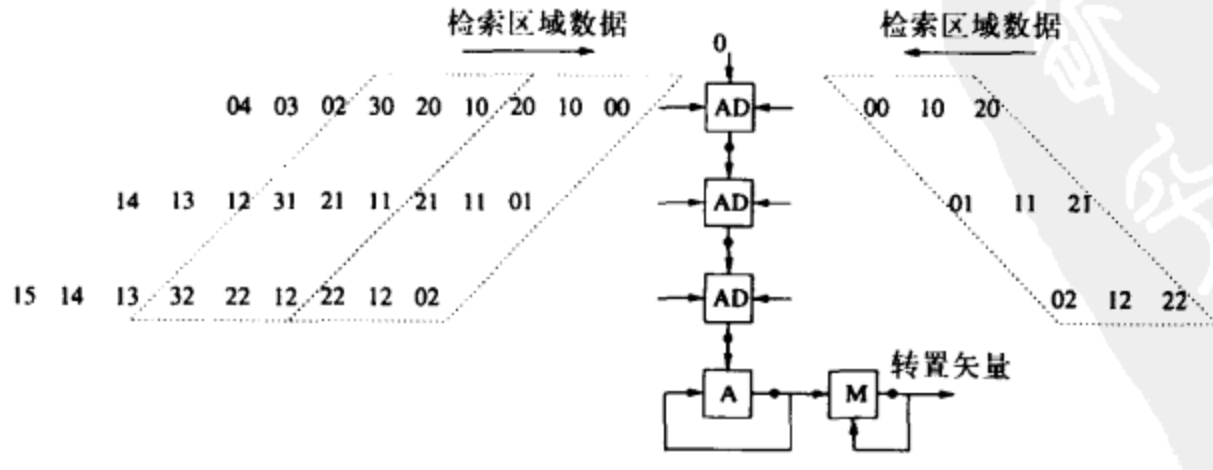


图7-40 第19题运动估值的一维脉动阵列

参考文献

1. H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Sparse Matrix Symposium, SIAM*, pp. 256–282, 1978.
2. H. T. Kung, "Why systolic architectures?" *IEEE Computers Magazine*, vol. 15, pp. 37–45, Jan. 1982.
3. S. Y. Kung, *VLSI Array Processors*. Prentice Hall, 1988.
4. H. V. Jagadish, S. K. Rao, and T. Kailath, "Array architecture for iterative algorithms," *Proc. IEEE*, vol. 75, no. 9, pp. 1304–1321, Sept. 1987.
5. P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations," in *Proc. of 11th Annual Symposium on Computer Architecture*, pp. 208–214, June 1984.
6. D. I. Moldovan and J. A. B. Fortes, "Partioning and mapping of algorithms into fixed size systolic arrays," *IEEE Trans. on Computers*, pp. 1–12, Jan. 1986.
7. G. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. on Computers*, pp. 67–77, Jan. 1985.
8. P. R. Cappello and K. Steiglitz, "Unifying VLSI array design with linear transformation of space time," *Advances Comput. Res.*, vol. 2, pp. 23–65, 1984.
9. K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, Jan. 1992.
10. Y. T. Hwang and Y. H. Hu, "MSSM–a design aid for multi-stage systolic mapping," *Journal of VLSI Signal Processing*, vol. 4, pp. 125–145, 1992.
11. J. Fridman and E. S. Manolakos, "Discrete wavelet transform: data dependence analysis and synthesis of distributed memory and control array architectures," *IEEE Trans. on Signal Processing*, vol. 45, pp. 1291–1308, May 1997.
12. T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. on Circuits and Systems*, no. 10, pp. 1301–1308, Oct. 1989.

225 ~ 226

第8章 快速卷积

8.1 引言

本章讨论如何应用较少的乘法运算实现卷积算法。这些被称为快速卷积的算法，属于算法强度缩减的范畴，即在增加诸如加法运算这样的弱运算数目的代价下，尽可能地减少诸如乘法运算这样的强运算数目。这种实现方法无论对于可编程硬件还是专用硬件的实现，都是非常适合的。

从下述的复数乘法例子，可以显见快速算法对于缩减运算复杂度的重要性。假设 $(a+jb)(c+jd)=e+jf$，其中 $(a+jb)$ 是信号样值，$(c+jd)$ 是系数。这可以表示为矩阵形式:

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

直接实现需要4次乘法和2次加法。然而，运用下式

$$\begin{aligned} ac-bd &= a(c-d)+d(a-b) \\ ad+bc &= b(c+d)+d(a-b) \end{aligned} \tag{8-1}$$

系数矩阵可以分解为一个2×3、3×3和3×2矩阵的乘积:

227

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c-d & 0 & 0 \\ 0 & c+d & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \tag{8-2}$$

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}$ 是只需要一次加法并可以预先计算的矩阵。注意到 $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ b \\ a-b \end{bmatrix}$。$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ 是需要2次加法的可以延后计算的矩阵。还注意到，乘法的次数即是3×3对角线矩阵的元素个数。因此，运算的复杂度被缩减为3次乘法和3次加法。一次乘法被一次加法替换，从而大大节省了硬件开销。假定参数 $c-d$ 和 $c+d$ 是事先计算好的。

本章给出了设计快速短卷积算法[1][2]的两种著名方法。第8.2节[3]讨论了基于拉格朗日插值的Cook-Toom算法。第8.3节[4]讨论了基于中国剩余定理(China Remainder Theorem, CRT)的Winograd算法。第8.4节阐述了用短卷积算法导出长序列卷积的高计算效率的实现形式。第8.5节讨论了环形卷积及其与线性卷积的关系。第8.6节讲述了如何通过观察来设计快速卷积算法。

8.2 Cook-Toom算法

Cook-Toom算法是用于多项式相乘的线性卷积算法。它基于拉格朗日插值定理。拉格朗日插值定理的表述如下:

“令 $\beta_0, \ldots, \beta_n$ 为 $n+1$ 个不同点的集合，并给定 $f(\beta_i)$，其中 $i=0, 1, \ldots, n$。当在 β_i 点求

值时，有且仅有一个阶数小于等于 n 的多项式 $f(p)$ 在该点的值为 $f(\beta_i)$，其中 $i = 0, 1, \ldots, n$。该多项式由下式给出：

$$f(p) = \sum_{i=0}^{n} f(\beta_i) \frac{\Pi_{j \neq i}(p - \beta_j)}{\Pi_{j \neq i}(\beta_i - \beta_j)} \tag{8-3}$$

228 考虑N点序列 $h = \{h_0, h_1, \ldots, h_{N-1}\}$ 和 L 点序列 $x = \{x_0, x_1, \ldots x_{L-1}\}$。$h$ 和 x 的线性卷积可以表示为如下的多项式相乘形式：

$$s(p) = h(p)x(p) \tag{8-4}$$

其中 $h(p) = h_{N-1} p^{N-1} + \ldots + h_1 p + h_0$，$x(p) = x_{L-1} p^{L-1} + \ldots + x_1 p + x_0$ 和 $s(p) = s_{L+N-2} p^{L+N-2} + \ldots + s_1 p + s_0$。输出多项式 $s(p)$ 的阶数为 $L+N-2$，并有 $L+N-1$ 个不同的系数。因此，该式可以由它在 $L+N-1$ 个不同点上的值惟一地确定。令 $\beta_0, \beta_1, \ldots, \beta_{L+N-2}$ 为 $L+N-1$ 个不同的实数。如果已知 $s(\beta_i)$，其中 $i = 0, 1, \ldots, L+N-2$，则 $s(p)$ 可以由拉格朗日插值定理计算得出

$$s(p) = \sum_{i=0}^{L+N-2} s(\beta_i) \frac{\prod_{j \neq i}(p - \beta_j)}{\prod_{j \neq i}(\beta_i - \beta_j)} \tag{8-5}$$

可以证明，公式（8-5）在给定 $s(p)$ 的条件下是 $s(\beta_i)$ 的惟一解，其中 $i = 0, 1, \ldots, L+N-2$。

利用Cook-Toom算法计算公式（8-4）中的线性卷积的步骤小结于下述算法。

算法8.2.1 Cook-Toom算法

(1) 选择 $L+N-1$ 个不同的实数 $\beta_0, \beta_1, \ldots, \beta_{L+N-2}$。

(2) 计算 $h(\beta_i)$ 和 $x(\beta_i)$，$i = 0, 1, \ldots, L+N-2$。

(3) 计算 $s(\beta_i) = h(\beta_i)\, x(\beta_i)$, $i = 0, 1, \ldots, L+N-2$。

(4) 用公式8-5计算 $s(p)$。

请注意，快速卷积算法的目的在于减少乘法的复杂度。如果适当地选择 β_i，$i = 0, 1, \ldots, L+N-2$，那么计算 $h(\beta_i)$ 和 $x(\beta_i)$ 的步骤2将只涉及到小常数的加法和乘法(例如，2的正数或负数次幂)。当 β_i's 为小数时，将忽略乘法运算。(然而必须记住的是，当问题的规模增加时，这些加法和乘法运算所带来的硬件复杂度的增加是不能被忽略的)。乘法运算在步骤3中用来计算 $s(\beta_i) = h(\beta_i)\, x(\beta_i)$，$i = 0, 1, \ldots, L+N-2$。这里只需要 $L+N-1$ 次通用乘法运算。因此，乘法的次数在增加加法次数的代价下，由 $O(LN)$ 减少为 $L+N-1$。由于加法器和乘法器相比，只需要更小的面积和计算时间，因此Cook-Toom算法可以大大节省硬件的复杂度，从而用来
229 得到计算效率较高的实现方式。

例8.2.1 用Cook-Toom算法构建一个2×2的卷积算法，β=0, ±1。

解 写出2×2卷积的多项式相乘形式

$$s(p) = h(p)x(p) \tag{8-6}$$

其中 $h(p) = h_0 + h_1 p$, $x(p) = x_0 + x_1 p$ 和 $s(p) = s_0 + s_1 p + s_2 p^2$。直接实现方式可以表示为下述的矩阵形式：

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} h_0 & 0 \\ h_1 & h_0 \\ 0 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \tag{8-7}$$

它需要4次乘法和1次加法。接下来，Cook-Toom算法可用于构建一个乘法次数缩减后的高效卷积算法。首先，算出 $h(\beta_i)$ 和 $x(\beta_i)$，$i=0,1,2$，从而得到

$$\begin{aligned} \beta_0 &= 0, & h(\beta_0) &= h_0, & x(\beta_0) &= x_0 \\ \beta_1 &= 1, & h(\beta_1) &= h_0+h_1, & x(\beta_1) &= x_0+x_1 \\ \beta_2 &= -1, & h(\beta_2) &= h_0-h_1, & x(\beta_2) &= x_0-x_1 \end{aligned} \tag{8-8}$$

然后，$s(\beta_0)$, $s(\beta_1)$ 和 $s(\beta_2)$ 由下式计算得出

$$\begin{aligned} s(\beta_0) &= h(\beta_0)x(\beta_0) \\ s(\beta_1) &= h(\beta_1)x(\beta_1) \\ s(\beta_2) &= h(\beta_2)x(\beta_2) \end{aligned} \tag{8-9}$$

它需要3次乘法。由公式（8-5）中的拉格朗日插值公式，可得

$$\begin{aligned} s(p) &= s(\beta_0)\frac{(p-\beta_1)(p-\beta_2)}{(\beta_0-\beta_1)(\beta_0-\beta_2)} + s(\beta_1)\frac{(p-\beta_0)(p-\beta_2)}{(\beta_1-\beta_0)(\beta_1-\beta_2)} \\ &\quad + s(\beta_2)\frac{(p-\beta_0)(p-\beta_1)}{(\beta_2-\beta_0)(\beta_2-\beta_1)} \\ &= s(\beta_0)(-p^2+1) + s(\beta_1)\frac{p^2+p}{2} + s(\beta_2)\frac{p^2-p}{2} \\ &= s(\beta_0) + p(\frac{s(\beta_1)}{2} - \frac{s(\beta_2)}{2}) + p^2(-s(\beta_0) + \frac{s(\beta_1)}{2} + \frac{s(\beta_2)}{2}) \\ &= s_0 + ps_1 + p^2 s_2 \end{aligned} \tag{8-10}$$

230

上述计算得到了矩阵形式

$$\begin{aligned} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} s(\beta_0) \\ \frac{s(\beta_1)}{2} \\ \frac{s(\beta_2)}{2} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} h(\beta_0) & 0 & 0 \\ 0 & \frac{h(\beta_1)}{2} & 0 \\ 0 & 0 & \frac{h(\beta_2)}{2} \end{bmatrix} \begin{bmatrix} x(\beta_0) \\ x(\beta_1) \\ x(\beta_2) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 \\ 0 & 0 & \frac{h_0-h_1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \end{aligned} \tag{8-11}$$

计算过程如下：

(1) $H_0 = h_0$，$H_1 = \dfrac{h_0+h_1}{2}$，$H_2 = \dfrac{h_0-h_1}{2}$。（事先计算得出）

(2) $X_0 = x_0$，$X_1 = x_0 + x_1$，$X_2 = x_0 - x_1$。

(3) $S_0 = H_0 X_0$，$S_1 = H_1 X_1$，$S_2 = H_2 X_2$。

(4) $s_0 = S_0$，$s_1 = S_1 - S_2$，$s_2 = -S_0 + S_1 + S_2$。

该算法需要3次乘法和5次加法。因此，在增加了4次加法运算的代价下，减少了1次乘法运算。

■

例8.2.2 考虑2×3的线性卷积

$$s(p) = h(p)x(p) \tag{8-12}$$

其中 $h(p) = h_0 + h_1 p$, $x(p) = x_0 + x_1 p + x_2 p^2$。用Cook-Toom算法构建该线性卷积的高效实现方式。

解 $s(p)$ 的阶数为3。令 $s(p) = s_0 + s_1 p + s_2 p^2 + s_3 p^3$。令 $\beta_0 = 0$，$\beta_1 = 1$，$\beta_2 = -1$，$\beta_3 = 2$。则

$$\begin{array}{lll} \beta_0 = 0, & h(\beta_0) = h_0, & x(\beta_0) = x_0 \\ \beta_1 = 1, & h(\beta_1) = h_0 + h_1, & x(\beta_1) = x_0 + x_1 + x_2 \\ \beta_2 = -1, & h(\beta_2) = h_0 - h_1, & x(\beta_1) = x_0 - x_1 + x_2 \\ \beta_3 = 2, & h(\beta_2) = h_0 + 2h_1, & x(\beta_1) = x_0 + 2x_1 + 4x_2 \end{array} \tag{8-13}$$

和

231

$$\begin{array}{c} s(\beta_0) = h(\beta_0)x(\beta_0) \\ s(\beta_1) = h(\beta_1)x(\beta_1) \\ s(\beta_2) = h(\beta_2)x(\beta_2) \\ s(\beta_3) = h(\beta_3)x(\beta_3) \end{array} \tag{8-14}$$

需要4次乘法。由公式（8-5）中的拉格朗日插值公式，可得

$$\begin{aligned} s(p) &= s(\beta_0)\frac{(p-\beta_1)(p-\beta_2)(p-\beta_3)}{(\beta_0-\beta_1)(\beta_0-\beta_2)(\beta_0-\beta_3)} \\ &\quad + s(\beta_1)\frac{(p-\beta_0)(p-\beta_2)(p-\beta_3)}{(\beta_1-\beta_0)(\beta_1-\beta_2)(\beta_1-\beta_3)} \\ &\quad + s(\beta_2)\frac{(p-\beta_0)(p-\beta_1)(p-\beta_3)}{(\beta_2-\beta_0)(\beta_2-\beta_1)(\beta_2-\beta_3)} \\ &\quad + s(\beta_3)\frac{(p-\beta_0)(p-\beta_1)(p-\beta_2)}{(\beta_3-\beta_0)(\beta_3-\beta_1)(\beta_3-\beta_2)} \\ &= \frac{s(\beta_0)}{2}(p^3-2p^2-p+2) + \frac{s(\beta_1)}{-2}(p^3-p^2-2p) \\ &\quad + \frac{s(\beta_2)}{-6}(p^3-3p^2+2p) + \frac{s(\beta_3)}{6}(p^3-p) \\ &= s(\beta_0) \\ &\quad + p(-\frac{s(\beta_0)}{2} + s(\beta_1) - \frac{s(\beta_2)}{3} - \frac{s(\beta_3)}{6}) \\ &\quad + p^2(-s(\beta_0) + \frac{s(\beta_1)}{2} + \frac{s(\beta_2)}{2}) \\ &\quad + p^3(\frac{s(\beta_0)}{2} - \frac{s(\beta_1)}{2} - \frac{s(\beta_2)}{6} + \frac{s(\beta_3)}{6}) \end{aligned} \tag{8-15}$$

最后，卷积运算可表示为矩阵-矢量形式

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & 2 & -2 & -1 \\ -2 & 1 & 3 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \frac{s(\beta_0)}{2} \\ \frac{s(\beta_1)}{2} \\ \frac{s(\beta_2)}{6} \\ \frac{s(\beta_3)}{6} \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & 0 & 0 \\ \underline{-1} & 2 & -2 & \underline{-1} \\ -2 & 1 & 3 & 0 \\ \underline{1} & -1 & -1 & \underline{1} \end{bmatrix} \begin{bmatrix} \frac{h_0}{2} & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 \\ 0 & 0 & \frac{h_0-h_1}{6} & 0 \\ 0 & 0 & 0 & \frac{h_0+2h_1}{6} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ \underline{1} & 1 & \underline{1} \\ \underline{1} & -1 & \underline{1} \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{8-16}$$

计算过程如下：

232

(1) $H_0 = \frac{h_0}{2}$, $H_1 = \frac{h_0+h_1}{2}$, $H_2 = \frac{h_0-h_1}{6}$, $H_3 = \frac{h_0+2h_1}{6}$。(事先计算得出)

(2) $X_0 = x_0$, $X_1 = (x_0+x_2)+x_1$, $X_2 = (x_0+x_2)-x_1$, $X_3 = x_0+2x_1+4x_2$。

(3) $S_0 = H_0X_0$, $S_1 = H_1X_1$, $S_2 = H_2X_2$, $S_3 = H_3X_3$。

(4) $s_0 = 2S_0$, $s_1 = -(S_0+S_3)+2(S_1-S_2)$, $s_2 = -2S_0+S_1+3S_2$, $s_3 = (S_0+S_3)-(S_1+S_2)$。

该算法与直接实现方式所需要的6次乘法和2次加法相比，只需要4次乘法和12次加法。■

讨论

注意到，前置加法矩阵中的某些加法运算可以共享，例如，例8.2.2中的公式（8-16）画下划线的部分 $x_0 + x_2$。当计算加法次数时，只需要1次而不是2次。同理，对后置加法矩阵也成立，例如 $S_0 + S_3$ 可以共享。

如果将卷积看作是3个抽样数据通过一个2阶FIR滤波器，将 h_0、h_1 看作滤波器系数，将 x_0、x_1、x_2 看作数据序列。那么，每次使用滤波器时，H_0、H_1、H_2、H_3 不需要重新计算。它们可以事先计算一次就存储起来。因此，当计算运算次数时，可以忽略这些运算。

由例8.1.1和8.2.2可以看出，Cook-Toom算法可以理解为矩阵的分解。通常，卷积可以表示为矩阵-矢量形式

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} h_0 & 0 & 0 \\ h_1 & h_0 & 0 \\ 0 & h_1 & h_0 \\ 0 & 0 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{8-17}$$

或者

$$\mathbf{s} = \mathbf{T}\mathbf{x} \tag{8-18}$$

方程（8-18）通常可表示为

$$\mathbf{s} = \mathbf{CHDx} \tag{8-19}$$

其中，**C** 是后置加法矩阵，**D** 是前置加法矩阵，**H** 是主对角线元素为 H_i 的对角线矩阵，$i = 0, 1, \ldots, L+N-2$。由公式（8-18）和（8-19）可得 **T** = **CHD**。这意味着Cook-Toom算法提供了一种将卷积矩阵 **T** 因式分解为一个后置加法矩阵、一个对角线矩阵和一个前置加法矩阵之积的方法，使得通用乘法的数目完全由对角线矩阵 **H** 的非零对角元素的数目来决定。 233

虽然乘法的次数减少了三分之一，但是加法的次数增加了。

现在，可以修改Cook-Toom算法，以便进一步地减少加法运算的次数。定义

$$s'(p) = s(p) - s_{L+N-2}p^{L+N-2} \tag{8-20}$$

注意到，$s(p)$ 的阶数为 $L+N-2$，s_{L+N-2} 是它的最高次项系数。因此，$s'(p)$ 的阶数为 $L+N-3$。修改后的Cook-Toom算法小结如下。

算法8.2.2 修改后的Cook-Toom算法

(1) 选择 $L+N-2$ 个不同的实数 $\beta_0, \beta_1, \ldots, \beta_{L+N-3}$。

(2) 计算 $h(\beta_i)$ 和 $x(\beta_i)$，$i = 0, 1, \ldots, L+N-3$。

(3) 计算 $s(\beta) = h(\beta_i)\, x(\beta_i)$，$i = 0, 1, \ldots, L+N-3$。

(4) 计算 $s'(\beta_i) = s(\beta_i) - s_{L+N-2}\, \beta_i^{L+N-2}$，$i = 0, 1, \ldots, L+N-3$。

(5) 用公式8-5计算 $s'(p)$。

(6) 计算 $s(p) = s'(p) + s_{L+N-2}\, p^{L+N-2}$。

例8.2.3 用修改后的Cook-Toom算法导出2×2卷积算法，其中 $\beta_0=0$ 和 $\beta_1=-1$，并和例8.2.1中的结果相比较。

解 考虑

$$s'(p) = s(p) - h_1 x_1 p^2 \tag{8-21}$$

在 $\beta_0=0$ 和 $\beta_1=-1$ 处的拉格朗日插值。首先找出 $s'(\beta_i) = h(\beta_i)\, x(\beta_i) - h_1\, x_1\, \beta_i^2$。

$$\begin{aligned} \beta_0 = 0, \quad & h(\beta_0) = h_0, \quad & x(\beta_0) = x_0 \\ \beta_1 = -1, \quad & h(\beta_1) = h_0 - h_1, \quad & x(\beta_1) = x_0 - x_1 \end{aligned} \tag{8-22}$$

和

$$\begin{aligned} s'(\beta_0) = \quad & h(\beta_0)x(\beta_0) - h_1 x_1 \beta_0^2 = h_0 x_0 \\ s'(\beta_1) = \quad & h(\beta_1)x(\beta_1) - h_1 x_1 \beta_1^2 = (h_0 - h_1)(x_0 - x_1) - h_1 x_1 \end{aligned} \tag{8-23}$$

234

在不计 $h_1\, x_1$ 乘法时需要2次乘法。对 $s'(p)$ 应用公式（8-5）中的拉格朗日插值公式，可得

$$\begin{aligned} s'(p) &= s'(\beta_0)\frac{(p-\beta_1)}{(\beta_0-\beta_1)} + s'(\beta_1)\frac{(p-\beta_0)}{(\beta_1-\beta_0)} \\ &= s'(\beta_0) + p(s'(\beta_0) - s'(\beta_1)) \end{aligned} \tag{8-24}$$

因此，

$$\begin{aligned} s(p) &= s'(p) + h_1 x_1 p^2 \\ &= s_0 + s_1 p + s_2 p^2 = s_0 + s_1 p \end{aligned} \tag{8-25}$$

最后可得，

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s'(\beta_0) \\ s'(\beta_1) \\ h_1 x_1 \end{bmatrix} \tag{8-26}$$

注意到

$$\begin{bmatrix} s'(\beta_0) \\ s'(\beta_1) \\ h_1 x_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s(\beta_0) \\ s(\beta_1) \\ h_1 x_1 \end{bmatrix} \tag{8-27}$$

因此，

$$\begin{aligned}\begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s(\beta_0) \\ s(\beta_1) \\ h_1 x_1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 \\ 0 & h_0 - h_1 & 0 \\ 0 & 0 & h_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}\end{aligned} \tag{8-28}$$

计算过程如下：

(1) $H_0 = h_0$，$H_1 = h_0 - h_1$，$H_2 = h_1$。（事先计算得出）

(2) $X_0 = x_0$，$X_1 = x_0 - x_1$，$X_2 = x_1$。

(3) $S_0 = H_0 X_0$，$S_1 = H_1 X_1$，$S_2 = H_2 X_2$。

(4) $s_0 = S_0$，$s_1 = S_0 - S_1 + S_2$，$s_2 = S_2$。

运算次数包括3次乘法和3次加法。与例8.2.1中的卷积算法相比，加法运算减少了2次，而乘法运算的次数保持不变。 ■ 235

例8.2.4 用修改后的Cook-Toom算法导出2×3卷积算法，其中 $\beta = 0, \pm 1$，并和例8.2.2中的算法的复杂度相比较。

解 因为 $s'(p)$ 的阶数为 $L+N-3=2$，所以有3个不同的实数 β_i's，例如 $\beta_0 = 0$，$\beta_1 = 1$，$\beta_2 = -1$，就足以算出。那么

$$\begin{aligned} \beta_0 = 0, &\quad h(\beta_0) = h_0, &\quad x(\beta_0) = x_0 \\ \beta_1 = 1, &\quad h(\beta_1) = h_0 + h_1, &\quad x(\beta_1) = x_0 + x_1 + x_2 \\ \beta_2 = -1, &\quad h(\beta_2) = h_0 - h_1, &\quad x(\beta_1) = x_0 - x_1 + x_2 \end{aligned} \tag{8-29}$$

$$\begin{aligned} s'(\beta_0) &= h(\beta_0)x(\beta_0) - h_1 x_2 \beta_0^3 \\ s'(\beta_1) &= h(\beta_1)x(\beta_1) - h_1 x_2 \beta_1^3 \\ s'(\beta_2) &= h(\beta_2)x(\beta_2) - h_1 x_2 \beta_2^3 \end{aligned} \tag{8-30}$$

在不计 $h_1 x_2$ 乘法时需要3次乘法运算。对 $s'(p)$ 应用公式（8-5）中的拉格朗日插值公式，可得

$$\begin{aligned} s'(p) &= s'(\beta_0)\frac{(p-\beta_1)(p-\beta_2)}{(\beta_0-\beta_1)(\beta_0-\beta_2)} + s'(\beta_1)\frac{(p-\beta_0)(p-\beta_2)}{(\beta_1-\beta_0)(\beta_1-\beta_2)} \\ &\quad + s'(\beta_2)\frac{(p-\beta_0)(p-\beta_1)}{(\beta_2-\beta_0)(\beta_2-\beta_1)} \\ &= s'(\beta_0) + p\left(\frac{s'(\beta_1)}{2} - \frac{s'(\beta_2)}{2}\right) + p^2\left(-s'(\beta_0) + \frac{s'(\beta_1)}{2} + \frac{s'(\beta_2)}{2}\right) \end{aligned} \tag{8-31}$$

因此，

$$\begin{aligned} s(p) &= s'(p) + h_1 x_2 p^3 \\ &= s_0 + s_1 p + s_2 p^2 + s_3 p^3 \end{aligned} \tag{8-32}$$

最后可得

236

$$
\begin{aligned}
\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s'(\beta_0) \\ \frac{s'(\beta_1)}{2} \\ \frac{s'(\beta_2)}{2} \\ h_1 x_2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{1}{2} \\ 0 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&\quad \times \begin{bmatrix} h_0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 \\ 0 & 0 & \frac{h_0-h_1}{2} & 0 \\ 0 & 0 & 0 & h_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_0+x_1+x_2 \\ x_0-x_1+x_2 \\ x_2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 \\ 0 & 0 & \frac{h_0-h_1}{2} & 0 \\ 0 & 0 & 0 & h_1 \end{bmatrix} \\
&\quad \times \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}
\end{aligned}
\tag{8-33}
$$

计算过程如下：

(1) $H_0 = h_0$，$H_1 = \dfrac{h_0 + h_1}{2}$，$H_2 = \dfrac{h_0 - h_1}{2}$，$H_3 = h_1$。（事先计算得出）

(2) $X_0 = x_0$，$X_1 = (x_0 + x_2) + x_1$，$X_2 = (x_0 + x_2) - x_1$，$X_2 = x_2$。

(3) $S_0 = H_0 X_0$，$S_1 = H_1 X_1$，$S_2 = H_2 X_2$，$S_3 = H_3 X_3$。

(4) $s_0 = S_0$，$s_1 = S_1 - S_2 - S_3$，$s_2 = -S_0 + S_1 + S_2$，$s_3 = S_3$。

与例8.2.2中需要4次乘法和12次加法相比，该计算需要4次乘法和7次加法。 ■

以乘法次数为表征，Cook-Toom算法是高效的。然而正如早前所提到的，当问题的规模增加时，它的效率就不高了，因为对一个大系统而言，当输出序列的样值数目很大时，β 就不再仅是 $0, \pm 1, \pm 2, \pm 4$ 了。这样得到的前置和后置加法矩阵就不再是简单矩阵了。当面临更大的问题时，下节所描述的Winograd算法更具效率。

8.3 Winograd算法

Winograd短卷积算法是基于整数环的中国剩余定理（CRT），它的描述如下：

“如果已知一个非负整数在给定的不同模数下对应的余数，若这些模数互素并且已知该整数小于这些模数之积，则该非负整数可惟一确定。”

237

任何域上的一个多项式环内都存在CRT。整数环和多项式环上的CRT总结如下。

定理8.3.1 整数的CRT

已知 $c_i = R_{mi}[c]$，$i = 0, 1, \ldots, k$，其中 mi 为互素的模数，则

$$
c = \sum_{i=0}^{k} c_i N_i M_i \bmod M \tag{8-34}
$$

其中 $M=\prod_{i=0}^{k} m_i$，$M_i=M/m_i$ 和 N_i 是下式的解，

$$N_i M_i + n_i m_i = GCD(M_i, m_i) = 1 \tag{8-35}$$

假定 $0 \leqslant c < M$。符号 $R_{mi}[c]$ 表示 c 被 mi 除后的余数。

定理8.3.2 多项式的CRT

已知 $c^{(i)}(p)=R_{m^{(i)}(p)}[c(p)]$，$i=0,1,\ldots,k$，其中 $m^{(i)}(p)$ 为互素的模数，则

$$c(p)=\sum_{i=0}^{k} c^{(i)}(p) N^{(i)}(p) M^{(i)}(p) \ mod\ M(p) \tag{8-36}$$

其中 $M(p)=\prod_{i=0}^{k} m^{(i)}(p)$，$M^{(i)}(p)=\dfrac{M(p)}{m^{(i)}(p)}$ 和 $N^{(i)}(p)$是下式的解

$$N^{(i)}(p) M^{(i)}(p) + n^{(i)}(p) m^{(i)}(p) = GCD(M^{(i)}(p), m^{(i)}(p)) = 1 \tag{8-37}$$

假定 $c(p)$ 的阶数小于 $M(p)$ 的阶数。 ■

为了求解得出公式(8-35)和(8-37)中的 N_i 和 $N^{(i)}(p)$，需要分别应用整数环和多项式环上的欧几里得最大公约数算法(GCD)。欧几里得GCD算法在附录C中有详细的描述。

考虑对整数运用CRT的例子。选择模数 $m_0=3, m_1=4, m_2=5$。则 $M=m_0 m_1 m_2=60$ 和 $M_i=\dfrac{M}{m_i}$。那么

$$\begin{aligned} m_0 &= 3, \quad M_0 = 20, \quad (-1)20+7(3)=1 \\ m_1 &= 4, \quad M_1 = 15, \quad (-1)15+(4)4=1 \\ m_2 &= 5, \quad M_2 = 12, \quad (-2)12+(5)5=1 \end{aligned} \tag{8-38}$$

其中 N_i 和 n_i 由欧几里得GCD算法得到。假定整数 C 满足 $0 \leqslant C < M$，令 $c_i=R_{mi}[c]$。CRT表明对任意 $0 \leqslant c < M$，满足 c 取值的 M 与满足余数矢量 (c_0, c_1, c_2) 取值的 M 之间存在一一对应的关系。(注意到，$0 \leqslant c_0 < m_0$，$0 \leqslant c_1 < m_1$，$0 \leqslant c_2 < m_2$) 整数 c 可计算为 238

$$\begin{aligned} c &= \sum_{i=0}^{k} c_i N_i M_i \ mod\ M \\ &= (-20c_0 - 15c_1 - 24c_2)\ mod\ 60 \end{aligned} \tag{8-39}$$

例8.3.1 若 $c=17$，$c_0=R_3(17)=2$，$c_1=R_4(17)=1$，$c_2=R_5(17)=2$，

$$\begin{aligned} c &= (-20*2-15*1-24*2)\ mod\ 60 \\ &= -103\ mod\ 60 \\ &= 17 \end{aligned} \tag{8-40}$$

多项式对模数 $p^i+f(p)$ 的余数，其中 $deg\, f(p) \leqslant i-1$ 可以通过将多项式中的 p^i 替代为 $-f(p)$ 来得到。 ■

例8.3.2

(a) $R_{x+2}\left[5x^2+3x+5\right]=5(-2)^2+3(-2)+5=19$

(b) $R_{x^2+2}\left[5x^2+3x+5\right]=5(-2)+3x+5=3x-5$

(c) $R_{x^2+x+2}\left[5x^2+3x+5\right]=5(-x-2)+3x+5=-2x-5$ ■

现在考虑多项式环上的CRT。考虑式

$$s(p)=h(p)x(p)\ mod\ m(p) \tag{8-41}$$

的计算，其中 $m(p)=m^{(0)}(p)m^{(1)}(p)\ldots m^{(k)}(p)$ 和 $m^{(i)}(p)$ 是一对互素数。假定 $M^{(i)}(p)=m(p)/m^{(i)}(p)$ 和 $N^{(i)}(p)$ 可由公式8-37得到。令 $s^{(i)}(p)=R_{m_{(i)}(p)}[s(p)]$，$i=0,1,\ldots,k$。只要 $deg\ s(p)<deg\ m(p)$，多项式 $s(p)$ 可惟一确定

$$s(p)=\sum_{i=0}^{k}s^{(i)}(p)N^{(i)}(p)M^{(i)}(p)\ mod\ m(p) \tag{8-42}$$

注意到当 $deg\ h(p)\,x(p)<deg\ m(p)$ 时，

$$s(p)=h(p)x(p)\ mod\ m(p)=h(p)x(p) \tag{8-43}$$

方程（8-41）是线性卷积的另一种表述。线性卷积的高效实现可以通过适当地选择和分解多项式 $m(p)$ 来构建。这就得到了Winograd快速卷积算法。Winograd卷积算法的结构总结如下。

239

算法8.3.1 Winograd算法

(1) 选择多项式 $m(p)$，使其阶数高于 $h(p)\,x(p)$ 的阶数，并且把它分解成为 $k+1$ 个互素的实系数多项式，即 $m(p)=m^{(0)}(p)\,m^{(1)}(p)\ldots m^{(k)}(p)$。

(2) 令 $M^{(i)}(p)=\dfrac{m(p)}{m^{(i)}(p)}$ 并且运用欧几里得GCD算法求解公式（8-37）得到 $N^{(i)}(p)$。

(3) 计算

$$\begin{aligned} h^{(i)}(p) &= h(p)\ mod\ m^{(i)}(p) \\ x^{(i)}(p) &= x(p)\ mod\ m^{(i)}(p) \end{aligned} \tag{8-44}$$

其中 $i=0,1,\ldots,k$。

(4) 计算

$$s^{(i)}(p)=h^{(i)}(p)x^{(i)}(p)\ mod\ m^{(i)}(p) \tag{8-45}$$

其中 $i=0,1,\ldots,k$。

(5) 用公式（8-42）计算 $s(p)$。

注意只有步骤4中的表示短卷积的多项式之积 $h^{(i)}(p)\,x^{(i)}(p)$ 需要乘法。

例8.3.3 考虑例8.2.2中的2×3的线性卷积。用 $m(p)=p(p-1)(p^2+1)$ 的Winograd算法构建一个高效的实现方式。

解 令

$$\begin{aligned} m^{(0)}(p)&=p \\ m^{(1)}(p)&=p-1 \\ m^{(2)}(p)&=p^2+1 \end{aligned} \tag{8-46}$$

并且 $M^{(i)}(p)=\dfrac{m(p)}{m^{(i)}(p)}$，$i=0,1,2$。应用关系

$$N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}(p) = 1 \tag{8-47}$$

构建下面的表格:

i	$m^{(i)}(p)$	$M^{(i)}(p)$	$n^{(i)}(p)$	$N^{(i)}(p)$
0	p	$p^3 - p^2 + p - 1$	$p^2 - p + 1$	-1
1	$p - 1$	$p^3 + p$	$-\frac{1}{2}(p^2 + p + 2)$	$\frac{1}{2}$
2	$p^2 + 1$	$p^2 - p$	$-\frac{1}{2}(p - 2)$	$\frac{1}{2}(p - 1)$

计算余数 240

$$\begin{aligned} h^{(0)}(p) &= h_0, & x^{(0)}(p) &= x_0 \\ h^{(1)}(p) &= h_0 + h_1, & x^{(1)}(p) &= x_0 + x_1 + x_2 \\ h^{(2)}(p) &= h_0 + h_1 p, & x^{(2)}(p) &= (x_0 - x_2) + x_1 p \end{aligned} \tag{8-48}$$

和

$$\begin{aligned} s^{(0)}(p) &= h_0 x_0 = s_0^{(0)} \\ s^{(1)}(p) &= (h_0 + h_1)(x_0 + x_1 + x_2) = s_0(1) \\ s^{(2)}(p) &= (h_0 + h_1 p)((x_0 - x_2) + x_1 p) \ mod \ (p^2 + 1) \\ &= h_0(x_0 - x_2) - h_1 x_1 + (h_0 x_1 + h_1(x_0 - x_2))p \\ &= s_0^{(2)} + s_1^{(2)} p \end{aligned} \tag{8-49}$$

注意到，计算 $s^{(0)}(p)$ 需要一次乘法，$s^{(1)}(p)$ 需要一次乘法。$s^{(2)}(p)$ 的计算需要4次乘法。然而，可以如下所示进一步减少为3次乘法。

$$\begin{bmatrix} s_0^{(2)} \\ s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 \\ 0 & h_1 - h_0 & 0 \\ 0 & 0 & h_0 + h_1 \end{bmatrix} \begin{bmatrix} x_0 + x_1 - x_2 \\ x_0 - x_2 \\ x_1 \end{bmatrix}$$

那么

$$\begin{aligned} s(p) &= \sum_{i=0}^{2} s^{(i)}(p) N^{(i)}(p) M^{(i)}(p) \ mod \ m(p) \\ &= [-s^{(0)}(p)(p^3 - p^2 + p - 1) + \frac{s^{(1)}(p)}{2}(p^3 + p) \\ &\quad + \frac{s^{(2)}(p)}{2}(p^3 - 2p^2 + p)] \ mod \ (p^4 - p^3 + p^2 - p) \end{aligned} \tag{8-50}$$

将 $s^{(0)}(p)$，$s^{(1)}(p)$ 和 $s^{(2)}(p)$ 代入公式8-50，将 $s(p)$ 的系数以表格形式写出

p^0	p^1	p^2	p^3
$s_0^{(0)}$	$-s_0^{(0)}$	$s_0^{(0)}$	$-s_0^{(0)}$
0	$\frac{s_0^{(1)}}{2}$	0	$\frac{s_0^{(1)}}{2}$
0	$\frac{s_0^{(2)}}{2}$	$-s_0^{(2)}$	$\frac{s_0^{(2)}}{2}$
0	$\frac{s_1^{(2)}}{2}$	0	$-\frac{s_1^{(2)}}{2}$

241

因此有

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & 0 & -2 & 0 \\ -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} s_0^{(0)} \\ \frac{1}{2}s_0^{(1)} \\ \frac{1}{2}s_0^{(2)} \\ \frac{1}{2}s_1^{(2)} \end{bmatrix} \tag{8-51}$$

注意到，

$$\begin{bmatrix} s_0^{(0)} \\ \frac{1}{2}s_0^{(1)} \\ \frac{1}{2}s_0^{(2)} \\ \frac{1}{2}s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$
$$\begin{bmatrix} h_0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{h_1-h_0}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1}{2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_0+x_1+x_2 \\ x_0+x_1-x_2 \\ x_0-x_2 \\ x_1 \end{bmatrix} \tag{8-52}$$

最后有

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 1 & -1 \\ 1 & 0 & -2 & 0 & 2 \\ -1 & 1 & 0 & -1 & -1 \end{bmatrix}$$
$$\begin{bmatrix} h_0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{h_1-h_0}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{8-53}$$

这就完成了Winograd卷积算法的构建。该算法与直接实现需要6次乘法和2次加法相比，只需要5次乘法和11次加法。 ■

Winograd算法的乘法次数高度依赖于 $m^{(i)}(p)$ 的阶数。因此，$m(p)$ 的阶数要尽可能地小。根据CRT可得，极端情形是当 $deg\ m(p) = deg\ s(p) + 1$ 时。然而，通过选择 $m(p)$ 的阶数等于 $s(p)$ 的阶数，并且对 $s'(p) = s(p) - h_{N-1}\, x_{L-1}\, m(p)$ 应用CRT，可以得到一个更高效率的Winograd算法。注意到 $s'(p) \bmod m(p) = s(p) \bmod m(p)$。改进后的Winograd算法将在下面讨论。

242

算法8.3.2 改进后的Winograd算法

(1) 选择多项式 $m(p)$，使其阶数等于 $s(p)$ 的阶数，并且把它分解成为 $k+1$ 个互素的实系数多项式，即

$$m(p) = m^{(0)}(p)m^{(1)}(p)\cdots m^{(k)}(p) \tag{8-54}$$

(2) 令 $M^{(i)}(p) = \dfrac{m(p)}{m^{(i)}(p)}$ 并且运用欧几里得GCD算法求解公式（8-37）得到 $N^{(i)}(p)$。

(3) 计算

$$\begin{aligned} h^{(i)}(p) &= h(p) \bmod m^{(i)}(p) \\ x^{(i)}(p) &= x(p) \bmod m^{(i)}(p) \end{aligned} \tag{8-55}$$

其中 $i = 0, 1, \ldots, k$。

(4) 计算

$$s'^{(i)}(p) = h^{(i)}(p)x^{(i)}(p) \bmod m^{(i)}(p) \tag{8-56}$$

其中 $i = 0, 1, \ldots, k$。

(5) 用公式（8-42）计算 $s'(p)$。

(6) 计算 $s(p) = s'(p) + h_{N-1}\, x_{L-1}\, m(p)$。

例8.3.4 用 $m(p) = p(p-1)(p+1)$ 的改进后的Winograd算法构建一个2×3的卷积算法。

解 令

$$\begin{aligned} m^{(0)}(p) &= p \\ m^{(1)}(p) &= p-1 \\ m^{(2)}(p) &= p+1 \end{aligned} \tag{8-57}$$

并且 $M_{(i)}(p) = \dfrac{m(p)}{m^{(i)}(p)}$，$i = 0, 1, 2$。应用公式（8-47）中的关系式构建以下表格

i	$m^{(i)}(p)$	$M^{(i)}(p)$	$n^{(i)}(p)$	$N^{(i)}(p)$
0	p	p^2-1	p	-1
1	$p-1$	p^2+p	$-\frac{1}{2}(p+2)$	$\frac{1}{2}$
2	$p+1$	p^2-p	$-\frac{1}{2}(p-2)$	$\frac{1}{2}$

计算余数

$$\begin{aligned} h^{(0)}(p) &= h_0, & x^{(0)}(p) &= x_0 \\ h^{(1)}(p) &= h_0 + h_1, & x^{(1)}(p) &= x_0 + x_1 + x_2 \\ h^{(2)}(p) &= h_0 - h_1, & x^{(2)}(p) &= x_0 - x_1 + x_2 \end{aligned} \tag{8-58}$$

243

和

$$\begin{aligned} s'^{(0)}(p) &= h_0 x_0 \\ s'^{(1)}(p) &= (h_0 + h_1)(x_0 + x_1 + x_2) \\ s'^{(2)}(p) &= (h_0 - h_1)(x_0 - x_1 + x_2) \end{aligned} \tag{8-59}$$

因为 $m^{(i)}(p)$ 的阶数为1，$s'^{(i)}(p)$ 是阶数为0的多项式，即常数。因此有

$$\begin{aligned} s(p) &= s'(p) + h_1 x_2 m(p) \\ &= s'^{(0)}(-p^2+1) + \frac{s'^{(1)}}{2}(p^2+p) + \frac{s'^{(2)}}{2}(p^2-p) + h_1 x_2 (p^3 - p) \\ &= s'^{(0)} + p\left(\frac{s'^{(1)}}{2} - \frac{s'^{(2)}}{2} - h_1 x_2\right) \\ &\quad + p^2\left(-s'^{(0)} + \frac{s'^{(1)}}{2} + \frac{s'^{(2)}}{2}\right) + p^3(h_1 x_2) \end{aligned} \tag{8-60}$$

该算法可以写成矩阵形式

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s'(0) \\ \frac{s'(1)}{2} \\ \frac{s'(2)}{2} \\ h_1 x_2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 \\ 0 & 0 & \frac{h_0-h_1}{2} & 0 \\ 0 & 0 & 0 & h_1 \end{bmatrix} \tag{8-61}$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

该算法需要4次乘法和7次加法。 ■

8.4 迭代卷积

迭代卷积算法迭代运用高效的短卷积算法来构建长卷积[3]。尽管这些算法不会得到最小的乘法复杂度，但它们能在乘法和加法之间得到很好的平衡。

244

例8.4.1 用2×2的短卷积构建一个4×4的线性卷积算法。

解 令 $h(p)=h_0+h_1p+h_2p^2+h_3p^3$，$x(p)=x_0+x_1p+x_2p^2+x_3p^3$ 和 $s(p)=h(p)\,x(p)$。首先我们需要将4×4卷积分解成2×2卷积。定义 $h_0'(p)=h_0+h_1p$，$h_1'(p)=h_2+h_3p$，$x_0'(p)=x_0+x_1p$，$x_1'(p)=x_2+x_3p$和 $q=p^2$。那么我们有

$$\begin{aligned} h(p) &= h_0'(p)+h_1'(p)p^2, \quad \text{即.} \quad h(p)=h(p,q)=h_0'(p)+h_1'(p)q \\ x(p) &= x_0'(p)+x_1'(p)p^2, \quad \text{即.} \quad x(p)=x(p,q)=x_0'(p)+x_1'(p)q \end{aligned} \tag{8-62}$$

和

$$\begin{aligned} s(p) &= h(p)x(p)=h(p,q)x(p,q) \\ &= (h_0'(p)+h_1'(p)q)(x_0'(p)+x_1'(p)q) \\ &= s(p,q)=s_0'(p)+s_1'(p)q+s_2'(p)q^2 \end{aligned} \tag{8-63}$$

因此4×4卷积分解成为了2级嵌套2×2卷积。让我们从顶层2×2卷积开始，它用变量q来表示。用前一节得到的2×2卷积算法，有

$$\begin{bmatrix} s_0'(p) \\ s_1'(p) \\ s_2'(p) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_0'(p) & 0 & 0 \\ 0 & h_0'(p)-h_1'(p) & 0 \\ 0 & 0 & h_1'(p) \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0'(p) \\ x_1'(p) \end{bmatrix} \tag{8-64}$$

其中用到了3次多项式乘法，1次1阶多项式加法和2次2阶多项式加法。这3次用来计算 $s'_0(p)$，$s'_1(p)$ 和 $s'_2(p)$ 的多项式乘法，又是2×2卷积，每个需要3次乘法和3次加法。注意到 $s'_0(p)$ 和

$s'_1(p)$ 以及 $s'_1(p)$ 和 $s'_2(p)$ 的交叉项需要2次加法。因此，该4×4迭代卷积算法用到的运算总数为9次乘法和19次加法。 ■

迭代卷积算法的过程总结如下。

算法8.4.1 迭代卷积算法

(1) 将长卷积分解为若干级短卷积。 245

(2) 构建短卷积的快速卷积算法。

(3) 用短卷积算法迭代(层次)实现长卷积。

分解后的短卷积阶数会影响最后得到的长卷积结构的复杂度。

8.5 循环卷积

循环卷积又称为环形卷积。令 $h=\{h_0, h_1, \dots h_{n-1}\}$ 为滤波器系数，$x=\{x_0, x_1, \dots x_{n-1}\}$ 为数据序列。循环卷积可表示为:

$$s(p)=h\bigcirc_n x=h(p)x(p)\ mod\ (p^n-1) \tag{8-65}$$

输出序列表示为

$$s_i=\sum_{k=0}^{n-1}h_{((i-k))}x_k,\ i=0,1,\cdots,n-1 \tag{8-66}$$

其中 $((i-k))$ 表示 $(i-k)\ mod\ n$。循环卷积可以按照被 p^{n-1} 约简后的线性卷积来计算。(该线性卷积有 $2n-1$ 个不同的输出样值)。计算循环卷积的另外一种方法是用 $m(p)=p^{n-1}$ 的CRT来计算，这种方法简单得多。

例8.5.1 用 $m(p)=p^4-1=(p-1)(p+1)(p^2+1)$ 的CRT构建一个4×4的循环卷积算法。

解 令 $h(p)=h_0+h_1p+h_2p^2+h_3p^3$ 和 $x(p)=x_0+x_1p+x_2p^2+x_3p^3$。令 $m^{(0)}(p)=p-1$，$m^{(1)}(p)=p+1$，$m^{(2)}(p)=p^2+1$。并且 $M^{(i)}(p)=\dfrac{m(p)}{m^{(i)}(p)}$。用公式（8-37）中的关系构建以下表格

i	$m^{(i)}(p)$	$M^{(i)}(p)$	$n^{(i)}(p)$	$N^{(i)}(p)$
0	$p-1$	p^3+p^2+p+1	$-\frac{p^2+2p+3}{4}$	$\frac{1}{4}$
1	$p+1$	p^3-p^2+p-1	$\frac{p^2-2p+3}{4}$	$-\frac{1}{4}$
2	p^2+1	p^2-1	$\frac{1}{2}$	$-\frac{1}{2}$

计算余式

$$\begin{aligned}
h^{(0)}(p)&=h_0+h_1+h_2+h_3=h_0^{(0)}\\
h^{(1)}(p)&=h_0-h_1+h_2-h_3=h_0^{(1)}\\
h^{(2)}(p)&=(h_0-h_2)+(h_1-h_3)p=h_0^{(2)}+h_1^{(2)}p\\
x^{(0)}(p)&=x_0+x_1+x_2+x_3=x_0^{(0)}\\
x^{(1)}(p)&=x_0-x_1+x_2-x_3=x_0^{(1)}\\
x^{(2)}(p)&=(x_0-x_2)+(x_1-x_3)p=x_0^{(2)}+x_1^{(2)}p
\end{aligned} \tag{8-67}$$

246

和

$$
\begin{aligned}
s^{(0)}(p) &= h^{(0)}(p)x^{(0)}(p) = h_0^{(0)}x_0^{(0)} = s_0^{(0)} \\
s^{(1)}(p) &= h^{(1)}(p)x^{(1)}(p) = h_0^{(1)}x_0^{(1)} = s_0^{(1)} \\
s^{(2)}(p) &= s_0^{(2)} + s_1^{(2)}p = h^{(2)}(p)x^{(2)}(p) \ mod\ (p^2+1) \\
&= (h_0^{(2)}x_0^{(2)} - h_1^{(2)}x_1^{(2)}) + (h_0^{(2)}x_1^{(2)} + h_1^{(2)}x_0^{(2)})p
\end{aligned} \tag{8-68}
$$

注意到计算 $s_0^{(0)}$ 和 $s_0^{(1)}$ 各需要1次乘法，计算 $s_0^{(2)}$ 和 $s_1^{(2)}$ 需要4次乘法。然而，运用表达式

$$
\begin{aligned}
s_0^{(2)} &= h_0^{(2)}x_0^{(2)} - h_1^{(2)}x_1^{(2)} \\
&= h_0^{(2)}(x_0^{(2)} + x_1^{(2)}) - (h_0^{(2)} + h_1^{(2)})x_1^{(2)} \\
s_1^{(2)} &= h_0^{(2)}x_1^{(2)} + h_1^{(2)}x_0^{(2)} \\
&= h_0^{(2)}(x_0^{(2)} + x_1^{(2)}) + (h_1^{(2)} - h_0^{(2)})x_0^{(2)}
\end{aligned} \tag{8-69}
$$

或

$$
\begin{bmatrix} s_0^{(2)} \\ s_1^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} h_0^{(2)} & 0 & 0 \\ 0 & h_1^{(2)} - h_0^{(2)} & 0 \\ 0 & 0 & h_0^{(2)} + h_1^{(2)} \end{bmatrix} \begin{bmatrix} x_0^{(2)} + x_1^{(2)} \\ x_0^{(2)} \\ x_1^{(2)} \end{bmatrix} \tag{8-70}
$$

乘法次数减少为3。那么

$$
\begin{aligned}
s(p) &= \sum_{i=0}^{2} s^{(i)}(p)N^{(i)}(p)M^{(i)}(p) \ mod\ m(p) \\
&= s_0^{(0)}\frac{p^3+p^2+p+1}{4} + s_0^{(1)}\frac{p^3-p^2+p-1}{-4} \\
&\quad + s_0^{(2)}\frac{p^2-1}{-2} + s_1^{(2)}p\frac{p^2-1}{-2} \\
&= \frac{s_0^{(0)}}{4} + \frac{s_0^{(1)}}{4} + \frac{s_0^{(2)}}{2} + p(\frac{s_0^{(0)}}{4} - \frac{s_0^{(1)}}{4} + \frac{s_1^{(2)}}{2}) \\
&\quad + p^2(\frac{s_0^{(0)}}{4} + \frac{s_0^{(1)}}{4} - \frac{s_0^{(2)}}{2}) + p^3(\frac{s_0^{(0)}}{4} - \frac{s_0^{(1)}}{4} - \frac{s_1^{(2)}}{2})
\end{aligned} \tag{8-71}
$$

247

有

$$
\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{s_0^{(0)}}{4} \\ \frac{s_0^{(1)}}{4} \\ \frac{s_0^{(2)}}{2} \\ \frac{s_1^{(2)}}{2} \end{bmatrix} \tag{8-72}
$$

注意到

$$\begin{bmatrix} \frac{s_0^{(0)}}{4} \\ \frac{s_0^{(1)}}{4} \\ \frac{s_0^{(2)}}{2} \\ \frac{s_1^{(2)}}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{h_0^{(0)}}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0^{(1)}}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0^{(2)}}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{h_1^{(2)}-h_0^{(2)}}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0^{(2)}+h_1^{(2)}}{2} \end{bmatrix} \begin{bmatrix} x_0^{(0)} \\ x_0^{(1)} \\ x_0^{(2)}+x_1^{(2)} \\ x_0^{(2)} \\ x_1^{(2)} \end{bmatrix} \tag{8-73}$$

因此有

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{h_0+h_1+h_2+h_3}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0-h_1+h_2-h_3}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0-h_2}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{-h_0+h_1+h_2-h_3}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1-h_2-h_3}{2} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{8-74}$$

248

该算法需要5次乘法和15次加法，而直接实现

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} h_0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_3 \\ h_3 & h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{8-75}$$

需要16次乘法和12次加法。(注意到循环卷积矩阵是循环行列式矩阵)。 ■

高效循环卷积算法通常可以很容易地扩展构建为高效线性卷积。

例8.5.2 用例8.5.1中的4×4循环卷积算法构建一个3×3的线性卷积。

解 令3点系数序列为 $h=\{h_0, h_1, h_2\}$，并且3点数据序列为 $x=\{x_0, x_1, x_2\}$。首先，将它们扩展为4点序列

$$h=\{h_0,h_1,h_2,0\},\ x=\{x_0,x_1,x_2,0\} \tag{8-76}$$

则 h 和 x 的3×3线性卷积为

$$\begin{bmatrix} h_0x_0 \\ h_1x_0+h_0x_1 \\ h_2x_0+h_1x_1+h_0x_2 \\ h_2x_1+h_1x_2 \\ h_2x_2 \end{bmatrix} \tag{8-77}$$

并且 h 和 x 的4×4循环卷积 $h\bigcirc_4 x$ 为

$$\begin{bmatrix} h_0x_0+h_2x_2 \\ h_1x_0+h_0x_1 \\ h_2x_0+h_1x_1+h_0x_2 \\ h_2x_1+h_1x_2 \end{bmatrix} \tag{8-78}$$

因此有 $s(p)=h(p)\,x\,(p)=h\bigcirc_4 x+h_2\,x_2\,(p^4-1)$。对 $h\bigcirc_4 x$ 应用例8.5.1中的结果，就可得到如下的线性卷积算法：

249

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 & -1 \\ 1 & -1 & 1 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{h_0+h_1+h_2}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0-h_1+h_2}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{h_0-h_2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-h_0+h_1+h_2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1-h_2}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & h_2 \end{bmatrix} \tag{8-79}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

它需要6次乘法和16次加法。 ■

讨论 3×3线性卷积算法也可通过改进后的 $m(p)=p^4-1=(p-1)\,(p+1)\,(p^2+1)$ 的Winograd算法来构建。然而，因为我们已经得到了例8.5.1中的4×4循环卷积算法，因此，运用已有的循环卷积算法比用改进的Winograd算法的设计流程要简单得多。

通常，一个高效的线性卷积算法可用于得到一个高效的循环卷积算法。相反地，一个高效的循环卷积算法也可以用来推得一个高效的线性卷积算法。

8.6 通过观察设计快速卷积算法

不是所有的高效卷积算法都可以由Cook-Toom或Winograd算法生成。有时，通过观察得到一个机灵的分解，可以生成一个更好的算法。

例8.6.1 通过观察构建一个3×3的快速卷积算法。

解 3×3线性卷积可以写为

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} h_0x_0 \\ h_1x_0 + h_0x_1 \\ h_2x_0 + h_1x_1 + h_0x_2 \\ h_2x_1 + h_1x_2 \\ h_2x_2 \end{bmatrix} \tag{8-80}$$

250

它需要一次乘法和4次加法。运用以下表达式

$$\begin{aligned} s_1 &= h_1x_0 + h_0x_1 = (h_0 + h_1)(x_0 + x_1) - h_0x_0 - h_1x_1 \\ s_2 &= h_2x_0 + h_1x_1 + h_0x_2 = (h_0 + h_2)(x_0 + x_2) - h_0x_0 + h_1x_1 - h_2x_2 \\ s_3 &= h_2x_1 + h_1x_2 = (h_1 + h_2)(x_1 + x_2) - h_1x_1 - h_2x_2 \end{aligned} \tag{8-81}$$

3×3线性卷积可以写为

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} h_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & h_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_0 + h_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 + h_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & h_1 + h_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{8-82}$$

它需要6次乘法和10次加法。这个高效算法无法由Cook-Toom或Winograd算法得到。 ■

8.7 结论

本章介绍了各种短快速卷积算法，包括Cook-Toom算法、改进的Cook-Toom算法、Winograd算法、改进的Winograd算法。基于这些快速短卷积算法，可以通过迭代卷积方法得到长卷积算法。还讨论了循环卷积和线性卷积的关系，以及它们如何互相导出。快速卷积算法构成了快速并行FIR滤波器设计的基础(见第9.2.2.3节)。

8.8 习题

1. 用最少的乘法次数计算3个复数之积

$$x + jy = (a + jb)(c + jd)(e + jf) \tag{8-83}$$

2. 用2次乘法计算 $f_1 = ax + by$ 和 $f_2 = bx + ay$。

251

3. 用6次实数乘法计算下式给出的 f_1 和 f_2

$$\begin{aligned} f_1 &= (a_1 + ja_2)(x_1 + jx_2) + (b_1 + jb_2)(y_1 + jy_2) \\ f_2 &= (b_1 + jb_2)(x_1 + jx_2) + (a_1 + ja_2)(y_1 + jy_2) \end{aligned}$$

4. 用 $\beta = 0$、1、2 的Cook-Toom算法构建2×2卷积算法。
5. 用 β_i's 为 0、1、−1、2、−2 的Cook-Toom算法构建3×3卷积算法。
6. 用 β_i's 为 0、1、−1、2 的改进的Cook-Toom算法构建3×3卷积算法。

7. 用 $\beta_i^{'}s$ 为 0、1、2、-2 的改进的Cook-Toom算法构建3 × 3卷积算法。
8. 用Cook-Toom算法构建一个卷积算法，用于一个4阶FIR滤波器对3点数据序列进行滤波，即4 × 3的线性卷积。
9. 用Cook-Toom算法构建一个3 × 4的卷积算法。
10. 用改进的Cook-Toom算法构建一个4 × 3的线性卷积算法。
11. 用 $m(p) = p(p-1)(p+1)$ 的Winograd算法构建一个2 × 2的卷积算法。
12. 用 $m(p) = p(p-1)$ 的改进的Winograd算法构建一个2 × 2的卷积算法。
13. 为 $m(p) = p^2(p+1)(p-1)(p^2+1)$ 的4 × 3点的线性卷积设计一个Winograd卷积。
14. 用 $m(p) = p(p+1)(p-1)(p^2+1)$ 的改进的Winograd算法设计一个4 × 3点的线性卷积。
15. 用 $m(p) = p^3-1 = (p-1)(p^2+p+1)$ 构建一个3点环形卷积算法。
16. 用 $m(p) = p(p+1)(p^2+1)$ 的改进的Winograd算法计算线性卷积 $h = \{h_0, h_1\} * x = \{x_0, x_1, x_2, x_3\}$。
17. 用习题15的结果，计算一个2 × 3点的线性卷积。算出所需的乘法和加法的次数。将结果与由Cook-Toom算法或Winograd算法得出的线性卷积算法相比较。（提示：注意到 $h * x = h \bigcirc_3 x + h_1 x_2 (p^3-1)$。）

252

18. 利用2 × 2和3 × 2的高效短卷积算法，采用迭代卷积算法计算6 × 4卷积。请注意，所用到的短卷积算法的阶数很重要。计算出所有可能的阶数需要的乘法和加法次数并得出最优的结果。（提示：对3 × 2卷积运用 $h(p) = h_0^{'}(p^3) + ph_1^{'}(p^3) + p^2 h_2^{'}(p^3) + p^2 h_2^{'}(p^3)$，其中 $h^{'}(q) = h_i + h_{i+3} q$ 和 $q=p^3$。）
19. 例8.6.1中给出的3 × 3快速卷积需要6次乘法和10次加法。通过观察，得到采用 $h_0 x_0$、$h_1 x_1$、$h_2 x_2$、$(h_0 + h_1)(x_0 + x_1)$、$(h_1 + h_2)(x_1 + x_2)$ 和 $(h_0 + h_1 + h_2)(x_0 + x_1 + x_2)$ 项的另一个3 × 3卷积算法。证明该卷积算法需要6次乘法和9次加法。

参考文献

1. R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, 1985.
2. R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, 1989.
3. R. C. Agarwal and J. W. Cooley, "New algorithms for digital convolution," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASAP-25, pp. 392–410, 1977.
4. S. Winograd, "On computing the discrete Fourier transform," *Math. Comp.*, vol. 32, pp. 175–199, 1978.

253

第9章 滤波器和变换中的算法强度缩减

9.1 引言

本章介绍了算法强度缩减在并行FIR滤波器、离散余弦变换（DCT）以及并行秩-阶滤波器中的应用。强度缩减利用共享子结构达到了缩减硬件复杂度的效果。在一个VLSI实现或者一个可编程DSP实现的迭代周期中，用这种变换可以降低其硅面积或功耗。

在任何数字信号处理系统中，FIR滤波器是最基本的处理单元之一。FIR滤波器被用于DSP应用中，其范围覆盖从视频、图像处理到无线通信的应用。在某些应用中，如视频处理中，FIR滤波器电路必须能够在高频下工作，而在其他一些应用中，例如蜂窝电话的应用中，FIR滤波器电路又必须低功耗且能工作在适中的频率上。将并行或块处理用于数字FIR滤波器，可以提高原始滤波器的有效吞吐量，或者降低原始滤波器的功耗。通常，并行处理应用于FIR滤波器中将涉及原始滤波器中硬件单元的重复使用。如果原始滤波器所需的面积为 A，那么 L 并行电路就需要 $L \times A$ 的面积。换句话说，电路面积随着块尺寸的增加线性增长。在很多设计情况下，由于设计面积的限制，由并行处理带来的硬件开销是无法容忍的。因此，实现比传统并行FIR滤波器结构面积消耗更小的并行FIR滤波器结构是非常有益的。在9.2节中，算法强度缩减被用于并行FIR滤波器设计，电路面积增加小于线性增加。更小块尺寸的快速滤波算法被开发，并联接起来用于设计更大块尺寸的并行FIR滤波器。

离散余弦变换被广泛的用于数字信号处理中。它已经成为图像编码的主要工具，并已经成功地应用于高分辨率图像的编码。由于离散余弦变换重要性日益增长，因此为了加速运算以实现实时和高吞吐量，人们做出了许多努力。在9.3节中，我们用算法结构变换和频率抽取方法设计了快速离散余弦变换结构。

秩-阶滤波器属于非线性滤波器，在消除诸如脉冲噪声和闪烁噪声的非高斯噪声时非常有用。在9.4节中，讨论了并行可编程秩-阶滤波器的设计。在用最小数量的比较互换单元设计这些结构中，采用了共享子结构。

9.2 并行FIR滤波器

本节讨论采用算法强度缩减变换的方法来实现并行FIR滤波器。

9.2.1 并行FIR滤波器的多相式分解表示

本节阐述了并行FIR滤波器的多相式分解表示，这是用于多速率信号处理的一种技术[1]。在时域，一个N抽头的FIR滤波器可表示为:

$$y(n) = h(n) * x(n) = \sum_{i=0}^{N-1} h(i)x(n-i),\ n = 0, 1, 2, \cdots, \infty \tag{9-1}$$

其中，$\{x(n)\}$ 是一个无限长的输入序列，序列 $\{h(n)\}$ 包含了长度为N的FIR滤波器的系数，或者在 z 域中可以表示为:

$$Y(z)=H(z)X(z)=\sum_{n=0}^{N-1}h(n)z^{-n}\sum_{n=0}^{\infty}x(n)z^{-n} \tag{9-2}$$

输入序列 $\{x(0), x(1), x(2), x(3), \ldots\}$可以被分解为偶数部分和奇数部分，如下式所示:

256

$$\begin{aligned} X(z) &= x(0)+x(1)z^{-1}+x(2)z^{-2}+x(3)z^{-3}+\cdots \\ &= x(0)+x(2)z^{-2}+x(4)z^{-4}+\cdots \\ &\quad +z^{-1}[x(1)+x(3)z^{-2}+x(5)z^{-4}+\cdots] \\ &= X_0(z^2)+z^{-1}X_1(z^2) \end{aligned} \tag{9-3}$$

其中 $X_0(z^2)$ 和 $X_1(z^2)$ 分别为 $x(2k)$ 和 $x(2k+1)$ $(0\leqslant k<\infty)$ 的 z 变换。在公式（9-3）中，$X(z)$被分解为两个多相式。同样，N长度的滤波器系数 $H(Z)$ 可以被分解为:

$$H(Z) = H_0(z^2)+z^{-1}H_1(z^2) \tag{9-4}$$

其中 $H_0(z^2)$ 和 $H_1(z^2)$ 长度分别为 $N/2$，对应于偶数子滤波器和奇数子滤波器。例如，6抽头FIR滤波器 $H(z)=h(0)+h(1)z^{-1}+h(2)z^{-2}+h(3)z^{-3}+h(4)z^{-4}+h(5)z^{-5}$ 的偶数和奇数子滤波器分别为: $H_0(z)=h(0)+h(2)z^{-1}+h(4)z^{-2}$ 和 $H_1(z)=h(1)+h(3)z^{-1}+h(5)z^{-2}$。偶数部分的输出序列 $y(2k)$ 和奇数部分的输出序列 $y(2k+1)$ $(0\leqslant k<\infty)$ 的计算如下式所示:

$$\begin{aligned} Y(z) &= Y_0(z^2)+z^{-1}Y_1(z^2) \\ &= (X_0(z^2)+z^{-1}X_1(z^2))\,(H_0(z^2)+z^{-1}H_1(z^2)) \\ &= (X_0(z^2)H_0(z^2)+z^{-1}(X_0(z^2)H_1(z^2) \\ &\quad +X_1(z^2)H_0(z^2))+z^{-2}X_1(z^2)H_1(z^2) \end{aligned} \tag{9-5}$$

例如:

$$\begin{aligned} Y_0(z^2) &= X_0(z^2)H_0(z^2)+z^{-2}X_1(z^2)H_1(z^2) \\ Y_1(z^2) &= X_0(z^2)H_1(z^2)+X_1(z^2)H_0(z^2) \end{aligned} \tag{9-6}$$

其中 $Y_0(z^2)$ 和 $Y_1(z^2)$ 分别对应时域里的 $y(2k)$ 和 $y(2k+1)$。公式（9-6）中的滤波运算在每次迭代中，处理两个输入 $x(2k)$ 和 $x(2k+1)$，并产生两个输出 $y(2k)$ 和 $y(2k+1)$，所以被称为2并行FIR滤波器。这个2并行FIR滤波器可用矩阵形式表示如下:

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix}=\begin{bmatrix} H_0 & z^{-2}H_1 \\ H_1 & H_0 \end{bmatrix}\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \tag{9-7}$$

图9-1给出了获得的2并行FIR滤波器结构，这需要$2N$次乘法和$2(N-1)$次加法。

对于三相多相式分解，输入序列 $X(z)$ 和滤波器系数 $H(z)$ 可以分解如下:

$$X(z)=X_0(z^3)+z^{-1}X_1(z^3)+z^{-2}X_2(z^3) \tag{9-8}$$

$$H(z)=H_0(z^3)+z^{-1}H_1(z^3)+z^{-2}H_2(z^3) \tag{9-9}$$

257 其中 $X_0(z^3)$、$X_1(z^3)$ 和 $X_2(z^3)$ 分别对应时域的 $x(3k)$、$x(3k+1)$ 和 $x(3k+2)$; $H_0(z)$，$H_1(z)$ 和 $H_2(z)$ 是 $H(z)$ 的三个子滤波器。其输出可以计算如下:

$$
\begin{aligned}
Y(z) &= Y_0(z^3) + z^{-1}Y_1(z^3) + z^{-2}Y_2(z^3) \\
&= (X_0 + z^{-1}X_1 + z^{-2}X_2)(H_0 + z^{-1}H_1 + z^{-2}H_2) \\
&= X_0H_0 + z^{-1}(X_0H_1 + X_1H_0) + z^{-2}(X_0H_2 + X_1H_1 + X_2H_0) \\
&\quad + z^{-3}(X_1H_2 + X_2H_1) + z^{-4}X_2H_2 \\
&= X_0H_0 + z^{-3}(X_1H_2 + X_2H_1) + z^{-1}(X_0H_1 + X_1H_0 + z^{-3}X_2H_2) \\
&\quad + z^{-2}(X_0H_2 + X_1H_1 + X_2H_0)
\end{aligned}
\tag{9-10}
$$

因此，

$$
\begin{aligned}
Y_0(z^3) &= X_0(z^3)H_0(z^3) + z^{-3}X_1(z^3)H_2(z^3) + z^{-3}X_2(z^3)H_1(z^3) \\
Y_1(z^3) &= X_0(z^3)H_1(z^3) + X_1(z^3)H_0(z^3) + z^{-3}X_2(z^3)H_2(z^3) \\
Y_2(z^3) &= X_0(z^3)H_2(z^3) + X_1(z^3)H_1(z^3) + X_2(z^3)H_0(z^3)
\end{aligned}
\tag{9-11}
$$

其中 $Y_0(z^3)$、$Y_1(z^3)$ 和 $Y_2(z^3)$ 分别对应 $y(3k)$、$y(3k+1)$ 和 $y(3k+2)$。这种3并行FIR滤波器一次迭代处理3个采样输入 $x(3k)$、$x(3k+1)$ 和 $x(3k+2)$、产生3个采样输出 $y(3k)$、$y(3k+1)$和 $y(3k+2)$，它可用矩阵形式表示如下：

$$
\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-3}H_2 & z^{-3}H_1(z^3) \\ H_1 & H_0 & z^{-3}H_2(z^3) \\ H_2(z^3) & H_1(z^3) & H_0(z^3) \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix}
\tag{9-12}
$$

图9-2给出了获得的3并行FIR滤波器结构，它需要$3N$次乘法和$3(N-1)$次加法。

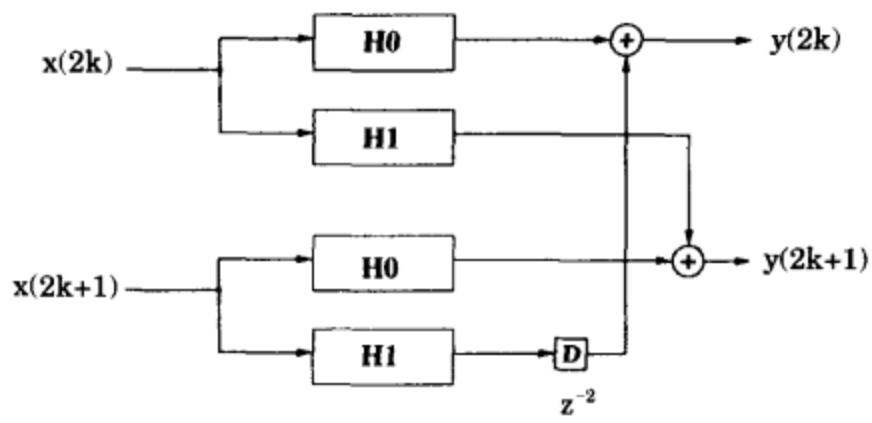


图9-1 传统的2并行FIR滤波器实现

通常，多相式分解可以通过将 $X(z)$、$H(z)$ 和 $Y(z)$ 分解为如下的 L 个子序列来获得 L 并行FIR滤波器：

258

$$
\begin{aligned}
X_i(z) &= \sum_{k=0}^{\infty} z^{-k}x(Lk+i),\ i=0,1,\cdots,L-1 \\
H_j(z) &= \sum_{k=0}^{N/L-1} z^{-k}h(Lk+j),\ j=0,1,\cdots,L-1 \\
Y_l(z) &= \sum_{k=0}^{\infty} z^{-k}y(Lk+l),\ l=0,1,\cdots,L-1
\end{aligned}
\tag{9-13}
$$

由L个输入子序列 $x(Lk+i)$ $(0 \le i \le L-1,\ 0 \le k < \infty)$，通过合并 L 个子滤波器，可以计算 L 个输出子序列 $y(Lk+i)$ $(0 \le i \le L-1,\ 0 \le k < \infty)$ 如下：

$$\sum_{l=0}^{L-1} Y_l(z^L)z^{-l} = \sum_{i=0}^{L-1} X_i(z^L)z^{-i} \sum_{j=0}^{L-1} H_j(z^L)z^{-j} \tag{9-14}$$

或者更清楚地表示为:

$$\begin{aligned} Y_k &= z^{-L} \sum_{i=k+1}^{L-1} H_i X_{L+k-i} + \sum_{i=0}^{k} H_i X_{k-i},\ 0 \leqslant k \leqslant L-2 \\ Y_{L-1} &= \sum_{i=0}^{L-1} H_i X_{L-1-i} \end{aligned} \tag{9-15}$$

上式也可以写成矩阵形式如下:

$$\mathbf{Y} = \mathbf{HX} \tag{9-16}$$

$$\begin{bmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{L-1} \end{bmatrix} = \begin{bmatrix} H_0 & z^{-L}H_{L-1} & \cdots & z^{-L}H_1 \\ H_1 & H_0 & \ddots & z^{-L}H_2 \\ \vdots & \vdots & \ddots & \vdots \\ H_{L-1} & H_{L-2} & \cdots & H_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{L-1} \end{bmatrix} \tag{9-17}$$

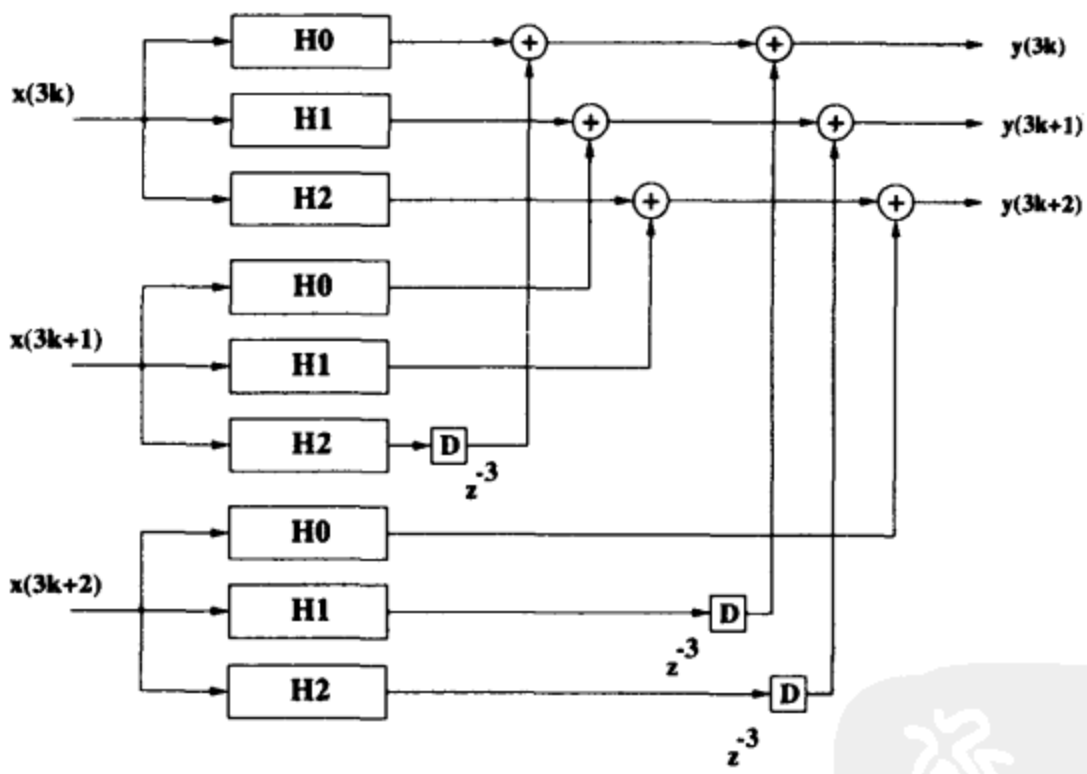


图9-2 传统的3并行FIR滤波器的实现

应该注意的是**H**是一个准循环行列式矩阵[1]。这种L并行FIR滤波器需要 L^2 次子滤波运算，每次运算的长度为N/L，并需要N/L次乘法-累加运算。因此，这种 L 并行FIR滤波器需要 $L^2\ N/L$ 次或者 LN 次乘法-累加运算，其运算次数与块尺寸L成线性关系。虽然多相式表示法没有缩减并行滤波器的复杂性，但它可以用于获得快速并行FIR滤波器。

9.2.2 快速FIR算法

由Winograd的著作[2]可知，2个$L-1$阶多项式相乘，可仅用（$2L-1$）个乘积项来实现

[3]。这种乘法数量的缩减是以加法数量的增加为代价来实现的。因为FIR滤波器的多相表达形式中的乘积项等于块表达形式的滤波运算，于是暗示了并行FIR滤波器可近似用 $(2L-1)$个长度为N/L的FIR滤波器来实现。

一个被称为快速FIR算法（FFAs）的相对比较新型的算法就是依赖于这种方法来生成复杂度降低的并行滤波结构[6]。采用这种方法，L并行滤波器可以近似用 $(2L-1)$ 个长度为 (N/L) 的滤波运算来实现。获得的并行滤波结构将需要 $(2N-N/L)$次乘法。例如，如果我们设 $N = 4$、$L = 2$，传统的2并行方法将需要8次乘法，而2并行快速滤波方法仅仅需要6次乘法。当N值很大时，FFAs可以大大缩减乘法的数量。

259 ~ 260

用快速FIR算法，本节阐述了硬件有效的并行FIR滤波器的起源。

9.2.2.1 2并行和3并行低复杂度FIR滤波器

2并行快速FIR滤波器 公式(9-6)中的2并行FIR滤波器需要$2N$次乘法和加法运算。注意公式(9-6)也可写成如下形式:

$$\begin{aligned} Y_0 &= H_0X_0 + z^{-2}H_1X_1 \\ Y_1 &= (H_0+H_1)(X_0+X_1) - H_0X_0 - H_1X_1 \end{aligned} \tag{9-18}$$

这个2并行快速滤波器包括5个子滤波器；然而其中的 H_0X_0 和 H_1X_1 是公共项，在计算 Y_0 和 Y_1 时可以共用。该低复杂度2并行滤波器结构如图9-3所示，它用三个截然不同的长度为$N/2$的子滤波器和4个预/后处理加法运算来计算一个2输出的块。它需要 $3N/2 = 1.5N$ 次乘法和 $3(N/2-1) + 4 = 1.5N + 1$ 次加法，然而对于直接从多相式分解导出的传统并行滤波器需要$2N$次乘法和$2(N-1)$次加法。例如，当N=8、$H = \{h_0, h_1, h_2, \ldots, h_6, h_7\}$ 时，它的三个子滤波器为:

$$\begin{aligned} H_0 &= \{h_0, h_2, h_4, h_6\} \\ H_1 &= \{h_1, h_3, h_5, h_7\} \\ H_0 + H_1 &= \{h_0+h_1, h_2+h_3, h_4+h_5, h_6+h_7\} \end{aligned} \tag{9-19}$$

这里 H_0+H_1 可预先计算出来。这个低复杂度的2并行滤波器需要12次乘法和13次加法，然而对于传统并行滤波器则需要16次乘法和14次加法运算。

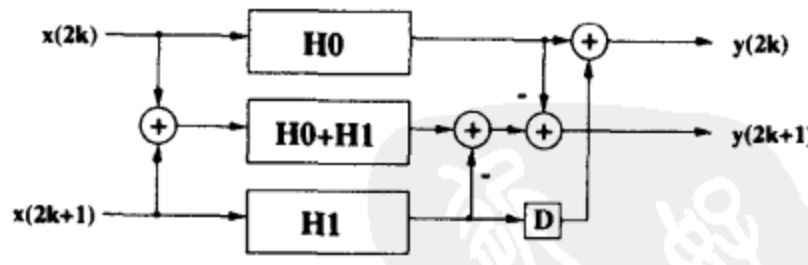


图9-3 缩减复杂度的2并行FIR滤波器的实现

公式（9-18）中的2并行滤波器也可以用矩阵形式表示如下:

$$\mathbf{Y_2} = \mathbf{Q_2H_2P_2X_2}$$

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & z^{-2} \\ -1 & 1 & -1 \end{bmatrix} diag \begin{bmatrix} H_0 \\ H_0+H_1 \\ H_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} \tag{9-20}$$

261

注意，符号

$$diag\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix}$$

表示一个$N \times N$对角线矩阵，其对角线元素为 $a_1, a_2, \ldots, a_N$。在这个矩阵表达式中，$\mathbf{Q}_2$ 是一个后处理矩阵，$\mathbf{P}_2$ 是一个预处理矩阵。$\mathbf{P}_2$ 决定了输入的结合方式，$\mathbf{Q}_2$ 决定了滤波器输出的结合方式以正确产生并行输出。应该注意的是FFA的应用对角化了公式（9-7）的准循环行列式矩阵。$\mathbf{H}_2$ 的对角线上的元素是这个并行FIR滤波器所需的子滤波器。

可以得到许多等价的并行FIR滤波器结构。例如，2并行滤波器可以用子滤波器 H_0、H_0-H_1 和 H_1 来实现（见习题1）。而且，所有的并行滤波器都可以转换成其他等价结构（见9.2.2.2节）。在可编程DSP的实现中，这些滤波器的性能比较相近，而在硬件VLSI的实现中，它们的性能却有很大的差别。例如，在一个窄带低通滤波器中，子滤波器 $H_0 + H_1$ 可能需要比 H_0-H_1 更多的非零比特[7]。与此类似，在窄带高通滤波器中，子滤波器 $H_0 + H_1$ 可能需要更少的非零比特。因而，对于一个特定的频谱，可以选择合适的滤波器结构来缩减面积和功耗。值得注意的是系数的和或者差也可以用于缩减序列滤波器的强度（见习题20）。另外，不同结构也将导致不同的舍入噪声特性。选择合适的滤波器结构也可以减小舍入噪声。

3并行快速FIR滤波器 快速3并行FIR算法可以通过递归应用2并行快速FIR算法来得到。

$$\begin{aligned} Y &= Y_0 + z^{-1}Y_1 + z^{-2}Y_2 \\ &= (X_0 + z^{-1}X_1 + z^{-2}X_2)(H_0 + z^{-1}H_1 + z^{-2}H_2) \\ &= (X_0 + z^{-1}V)(H_0 + z^{-1}W) \end{aligned} \tag{9-21}$$

其中 $V = X_1 + z^{-1} X_2$ 且 $W = H_1 + z^{-1} H_2$。用公式（9-18）中的快速FIR算法，可以计算 Y 如下:

$$\begin{aligned} Y &= H_0X_0 + z^{-2}VW + z^{-1}((H_0 + W)(X_0 + V) - H_0X_0 - VW) \\ &= [H_0X_0 + z^{-2}VW] \\ &\quad + z^{-1}[(H_0 + H_1 + z^{-1}H_2)(X_0 + X_1 + z^{-1}X_2) - H_0X_0 - VW] \end{aligned} \tag{9-22}$$

262

其中 $VW = (X_1 + z^{-1} X_2)(H_1 + z^{-1} H_2)$ 可以计算如下:

$$\begin{aligned} VW = &[H_1X_1 + z^{-2}H_2X_2] \\ &+ z^{-1}[(H_1 + H_2)(X_1 + X_2) - H_1X_1 - H_2X_2] \end{aligned} \tag{9-23}$$

将公式（9-23）代入公式（9-22）中，我们得到

$$\begin{aligned} Y = &[H_0X_0 + z^{-2}([H_1X_1 + z^{-2}H_2X_2] \\ &+ z^{-1}[(H_1 + H_2)(X_1 + X_2) - H_1X_1 - H_2X_2])] \\ &+ z^{-1}[((H_0 + H_1) + z^{-1}H_2)((X_0 + X_1) + z^{-1}X_2) - H_0X_0 \\ &- ([H_1X_1 + z^{-2}H_2X_2] + z^{-1}[(H_1 + H_2)(X_1 + X_2) - H_1X_1 - H_2X_2])] \\ = &H_0X_0 \\ &+ z^{-1}[(H_0 + H_1)(X_0 + X_1) - H_0X_0 - H_1X_1] \\ &+ z^{-2}[H_1X_1 + (H_0 + H_1)X_2 + H_2(X_0 + X_1) \\ &- (H_1 + H_2)(X_1 + X_2) + H_1X_1 + H_2X_2] \end{aligned} \tag{9-24}$$

$$
\begin{aligned}
&+z^{-3}[(H_1+H_2)(X_1+X_2)-H_1X_1-H_2X_2+H_2X_2-H_2X_2]\\
&+z^{-4}[H_2X_2]\\
=\ &H_0X_0-z^{-3}H_2X_2+z^{-3}[(H_1+H_2)(X_1+X_2)-H_1X_1]\\
&+z^{-1}([(H_0+H_1)(X_0+X_1)-H_1X_1]-[H_0X_0-z^{-3}H_2X_2])\\
&+z^{-2}([(H_0+H_1+H_2)(X_0+X_1+X_2)]-[(H_0+H_1)(X_0+X_1)\\
&-H_1X_1]-[(H_1+H_2)(X_1+X_2)-H_1X_1])
\end{aligned}
$$

获得的3并行FFA滤波器如下:

$$
\begin{aligned}
Y_0 &= H_0X_0-z^{-3}H_2X_2+z^{-3}[(H_1+H_2)(X_1+X_2)-H_1X_1]\\
Y_1 &= [(H_0+H_1)(X_0+X_1)-H_1X_1]-[H_0X_0-z^{-3}H_2X_2]\\
Y_2 &= [(H_0+H_1+H_2)(X_0+X_1+X_2)]\\
&\quad -[(H_0+H_1)(X_0+X_1)-H_1X_1]\\
&\quad -[(H_1+H_2)(X_1+X_2)-H_1X_1]
\end{aligned}
\tag{9-25}
$$

上式也可用矩阵形式写成:

$$
\mathbf{Y_3}=\mathbf{Q_3H_3P_3X_3}
$$

$$
\mathbf{Y_3}=\begin{bmatrix} Y_0\\ Y_1\\ Y_2\end{bmatrix},\mathbf{Q_3}=\begin{bmatrix} 1 & 0 & z^{-3} & 0\\ -1 & 1 & 0 & 0\\ 0 & -1 & -1 & 1\end{bmatrix}\begin{bmatrix} 1 & 0 & -z^{-3} & 0 & 0 & 0\\ 0 & -1 & 0 & 1 & 0 & 0\\ 0 & -1 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}
$$

263

$$
\mathbf{H_3}=diag\begin{bmatrix} H_0\\ H_1\\ H_2\\ H_0+H_1\\ H_1+H_2\\ H_0+H_1+H_2\end{bmatrix},\mathbf{P_3}=\begin{bmatrix} 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 1 & 1\end{bmatrix},\mathbf{X_3}=\begin{bmatrix} X_0\\ X_1\\ X_2\end{bmatrix}
\tag{9-26}
$$

其中 $\mathbf{P}_3$ 和 $\mathbf{Q}_3$ 分别为预处理和后处理矩阵，$\mathbf{H}_3$ 为对角化了的子滤波器矩阵。因此，可用6个长度为$N/3$的子滤波器构建3并行FIR滤波器，包括 H_0X_0、H_1X_1、H_2X_2、$(H_0+H_1)(X_0+X_1)$、$(H_1+H_2)(X_1+X_2)$ 和 $(H_0+H_1+H_2)(X_0+X_1+X_2)$ 以及3个预处理和7个后处理加法器。如图9-4所示。

所有的计算需要$2N$次乘法和$2N+4$次加法。比较传统3并行结构和降低复杂度的3并行结构的成本，显然降低复杂度的滤波结构比传统结构节省了近33%。

9.2.2.2 转置的并行滤波器

任何并行FIR滤波器结构通过转置操作（或转置）都可以得到另一个并行等价结构。通常，转置的结构具有相同的硬件复杂度，而有限字长性能却不同。本节阐述了转置的并行FIR滤波器的推导。

考虑以矩阵形式 **Y=HX** 表达的并行滤波运算，如公式（9-16）所示，其中 **H** 是一个 $L\times L$ 矩阵。通过将 **H** 矩阵转置并交换矢量 **X** 和 **Y** 的位置，可得到该并行滤波算法的等价实现。

$$
\mathbf{Y}_F=\mathbf{H}^T\mathbf{X}_F
$$

其中 $\mathbf{X_F}$ 和 $\mathbf{Y_F}$ 表示如下:

$$
\begin{aligned}
\mathbf{X}_F &= [\ X_{L-1}\quad X_{L-2}\quad \cdots\quad X_0\ ]^T\\
\mathbf{Y}_F &= [\ Y_{L-1}\quad Y_{L-2}\quad \cdots\quad Y_0\ ]^T
\end{aligned}
\tag{9-27}
$$

例如，公式（9-7）中的2并行FIR滤波器可以通过使用 $\mathbf{H}^T$ 和交换输入输出矢量重新表述为:

$$\begin{bmatrix} Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} H_0 & H_1 \\ z^{-2}H_1 & H_0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_0 \end{bmatrix} \tag{9-28}$$

公式（9-20）中2并行快速滤波器的转置变换

$$\mathbf{Y}_2 = \mathbf{Q}_2\mathbf{H}_2\mathbf{P}_2\mathbf{X}_2$$

可得到另一个等价结构

$$\begin{aligned} \mathbf{Y}_{2_F} &= (\mathbf{Q}_2\mathbf{H}_2\mathbf{P}_2)^T\mathbf{X}_{2F} \\ &= \mathbf{P}_2^T\mathbf{H}_2^T\mathbf{Q}_2^T\mathbf{X}_{2F} \end{aligned}$$

$$\begin{bmatrix} Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} diag \begin{bmatrix} H_0 \\ H_0 + H_1 \\ H_1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ z^{-2} & -1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_0 \end{bmatrix} \tag{9-29}$$

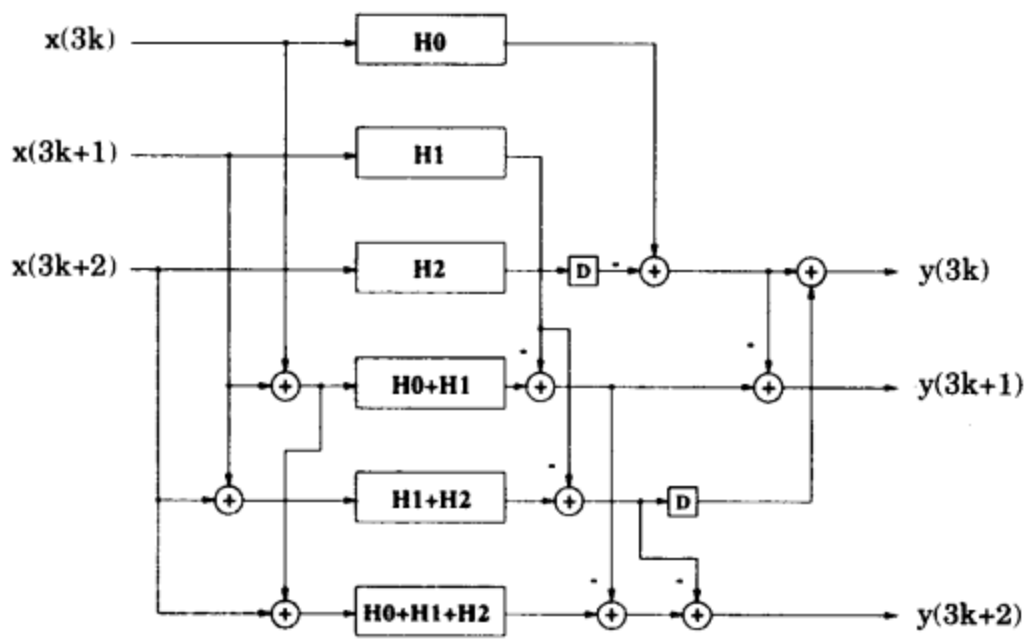


图9-4 降低复杂度的3并行FIR滤波器的实现

注意其滤波算法的计算复杂度保持不变。图9-5c展示了公式（9-29）所描述的降低复杂度的2并行FIR滤波器结构，它通过转置变换而得到。

转置结构也可以通过转置原始并行滤波器的信号流图来得到。如图9-5a和图9-5b所示。通常，对任意的FFA，矩阵转置和信号流图转置都可用来生成其等价并行滤波结构。

9.2.2.3 由线性卷积得到的并行滤波算法

任何 $L \times L$ 卷积算法也可得到 L 并行快速滤波器结构。例如，将一个2×2线性卷积算法矩阵转置

264 ~ 265

$$\begin{bmatrix} s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} h_1 & 0 \\ h_0 & h_1 \\ 0 & h_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \tag{9-30}$$

可用来得到2并行滤波器

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} H_1 & H_0 & 0 \\ 0 & H_1 & H_0 \end{bmatrix} \begin{bmatrix} z^{-2}X_1 \\ X_0 \\ X_1 \end{bmatrix} \tag{9-31}$$

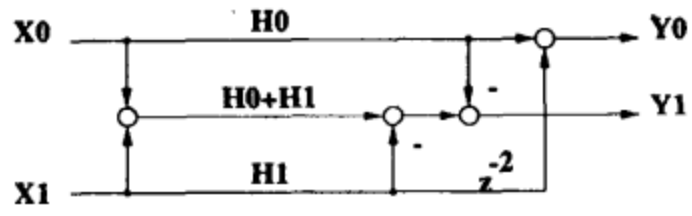
通常、L 并行滤波器

$$\mathbf{Y} = \mathbf{HX}$$

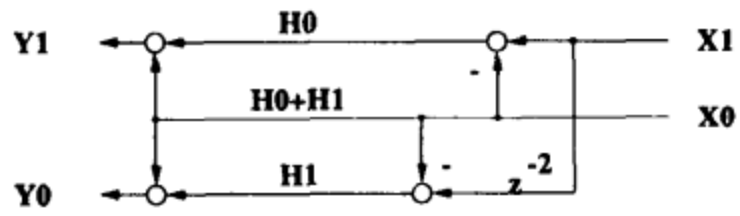
首先表示为如下形式:

$$\begin{bmatrix} Y_0 & Y_1 & \vdots & Y_{L-1} \end{bmatrix}^T$$

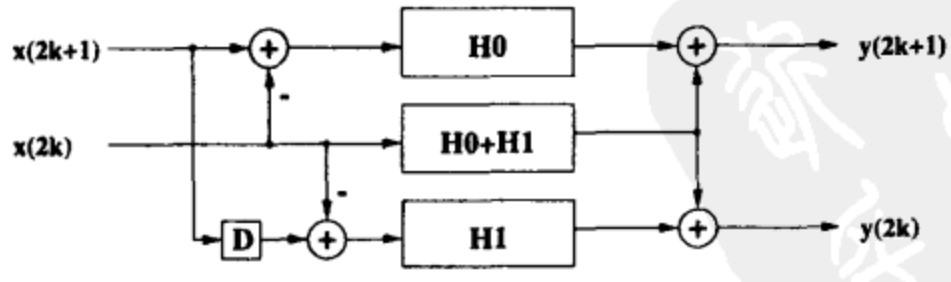
$$= \begin{bmatrix} H_{L-1} & H_{L-2} & \cdots & H_1 & H_0 & 0 & \cdots & 0 \\ 0 & H_{L-1} & \cdots & H_2 & H_1 & H_0 & \cdots & 0 \\ 0 & 0 & \cdots & H_3 & H_2 & H_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & H_{L-1} & H_{L-2} & \cdots & H_0 \end{bmatrix} \begin{bmatrix} z^{-L}X_1 \\ z^{-L}X_2 \\ \cdots \\ z^{-L}X_{L-1} \\ X_0 \\ X_1 \\ \cdots \\ X_{L-1} \end{bmatrix} \tag{9-32}$$



a) 图9-3中2并行FIR滤波器的信号流图



b) 转置后的信号流图



c) 转置后的降低复杂度2并行FIR滤波器的实现框图

图 9-5

注意标准并行滤波算法的这种表示形式与线性卷积的转置形式相似[8]。采用这种思路，就可以得到降低复杂度的并行滤波算法。其基本的思路是从一个理想线性卷积开始，用它的

转置变换得到并行滤波算法。为了用2×2快速卷积得到一个2并行滤波器，考虑下面的理想2×2线性卷积:

$$\mathbf{s} = \mathbf{CHAx}$$

$$\begin{bmatrix} s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} diag \begin{bmatrix} h_1 \\ h_0 + h_1 \\ h_0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} \tag{9-33}$$

注意序列 $\{s\}$、$\{h\}$ 和 $\{x\}$ 中采样的颠倒，保持了卷积的表达式，例如，对于颠倒序列，可以使用相同的矩阵 $\mathbf{C}$ 和 $\mathbf{A}$。求取该算法的转置并且对 s_i、h_i 和 x_i 做适当的替换，我们得到

$$\mathbf{Y} = (\mathbf{CHA})^{\mathrm{T}}\mathbf{X} = \mathbf{QHPX}$$

$$\begin{bmatrix} Y_0 \\ Y_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} diag \begin{bmatrix} H_1 \\ H_1 + H_0 \\ H_0 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} z^{-2}X_1 \\ X_0 \\ X_1 \end{bmatrix} \tag{9-34}$$

获得的2并行结构如图9-5所示。既然原始的2并行FFA基于公式（9-33）的理想线性卷积，那么用这种方法与用2并行FFA直接转置变换所得到的结构相同也就不足为奇了。上一节介绍的技术采用了FFA的直接转置变换，而这一节所采用的技术是线性卷积的转置变换。对于任意的FFA，都可以用这种方法来生成等价并行滤波实现。

267

9.2.2.4 用于大块尺寸的快速并行FIR算法

大尺寸的并行FIR滤波器可以用级联更小尺寸快速并行滤波器的方法来设计。例如，一个 m 并行FFA可以与一个 n 并行FFA级联起来，产生一个 $(m \times n)$ 并行滤波结构。从 m 并行FFA应用得到的一系列FIR滤波器一次一个地被 n 并行FFA应用进一步分解。最后得到的一系列滤波器的长度为 $N/(m \times n)$。当级联这些FFA时，明了滤波结构所需的乘法数量和加法数量是很重要的。下面给出一个 L 并行滤波器 $(L = L_1 L_2 \ldots L_r)$ 所需的乘法数量:

$$M = \frac{N}{\prod_{i=1}^{r} L_i} \prod_{i=1}^{r} M_i \tag{9-35}$$

这里的 r 是所用FFA的数量。L_i 是第 i 步时FFA的块尺寸，M_i 是由第 i 个FFA应用得到的滤波器数量，N 是滤波器的长度。所需的加法器可以计算如下:

$$A = A_1 \prod_{i=2}^{r} L_i + \sum_{i=2}^{r} \left[A_i \left(\prod_{j=i+1}^{r} L_j \right) \left(\prod_{k=1}^{i-1} M_k \right) \right] + \left[\prod_{i=1}^{r} M_i \right] \left[\frac{N}{\prod_{i=1}^{r} L_i} - 1 \right] \tag{9-36}$$

这里 A_i 是第 i 个FFA所需的预/后处理加法器的数量。考虑两个2并行FFA级联的情况。所得到的4并行滤波结构实现时总共需要$9N/4$次乘法和$20+9(N/4-1)$次加法。与传统4并行FIR滤波结构所需的$4N$次乘法相比，降低复杂度的4并行滤波结构节省的硬件（面积）将近44%。

例9.2.1 计算硬件复杂度

在本例中，将计算实现一个24抽头滤波器所需的乘法数和加法数，该滤波器具有 $L=6$ 的块尺寸。对 $L_1 = 2$、$L_2 = 3$ 和 $L_1 = 3$、$L_2 = 2$ 两种情况都分别进行计算。

$$\underline{L_1 = 2,\ L_2 = 3}$$

$$M_1 = 3,\ A_1 = 4$$
$$M_2 = 6,\ A_2 = 10$$

$$M = \tfrac{24}{(2\times3)} \times (3 \times 6) = 72$$
$$A = (4 \times 3) + (10 \times 3) + (3 \times 6)\left[\tfrac{24}{2\times3} - 1\right] = 96$$

268

$$\underline{L_1 = 3,\ L_2 = 2}$$

$$M_1 = 6,\ A_1 = 10$$
$$M_2 = 3,\ A_2 = 4$$

$$M = \tfrac{24}{(3\times2)} \times (6 \times 3) = 72$$
$$A = (10 \times 2) + (4 \times 6) + (6 \times 3)\left[\tfrac{24}{3\times2} - 1\right] = 98$$

■

为了弄清怎样级联FFA，重新考虑一下公式（9-15）是很有用的。考虑设计一个块尺寸为4的并行FIR滤波器。由公式（9-15）我们有：

$$\begin{aligned} Y = &\quad Y_0 + z^{-1}Y_1 + z^{-2}Y_2 + z^{-3}Y_3 \\ = &\quad (X_0 + z^{-1}X_1 + z^{-2}X_2 + z^{-3}X_3)(H_0 \\ &\quad + z^{-1}H_1 + z^{-2}H_2 + z^{-3}H_3) \end{aligned} \tag{9-37}$$

通过首先把2并行FFA应用到公式（9-37），然后把该FFA第二次应用到第一次FFA应用所获得的每一个滤波运算中，就得到了降低复杂度的4并行FIR滤波结构。由公式（9-37）有

$$Y = (X_0' + z^{-1}X_1')(H_0' + z^{-1}H_1')$$

其中，

$$\begin{aligned} X_0' &= X_0 + z^{-2}X_2, \quad X_1' = X_1 + z^{-2}X_3 \\ H_0' &= H_0 + z^{-2}H_2, \quad H_1' = H_1 + z^{-2}H_3 \end{aligned} \tag{9-38}$$

应用1

$$Y = X_0'H_0' + z^{-1}[(X_0' + X_1')(H_0' + H_1') - X_0'H_0' - X_1'H_1'] + z^{-2}X_1'H_1' \tag{9-39}$$

然后，2并行FFA第二次应用于公式（9-39）的每个滤波运算 $X'_0H'_0$，$X'_1H'_1$ 和 $(X'_0 + X'_1)(H'_0 + H'_1)$。

应用2，滤波运算 $X'_0H'_0$

$$\begin{aligned} X_0'H_0' = &\quad (X_0 + z^{-2}X_2)(H_0 + z^{-2}H_2) \\ = &\quad X_0H_0 + z^{-2}[(X_0 + X_2)(H_0 + H_2) \\ &\quad - X_0H_0 - X_2H_2] + z^{-4}X_2H_2 \end{aligned} \tag{9-40}$$

269

应用2，滤波运算 $X'_1H'_1$

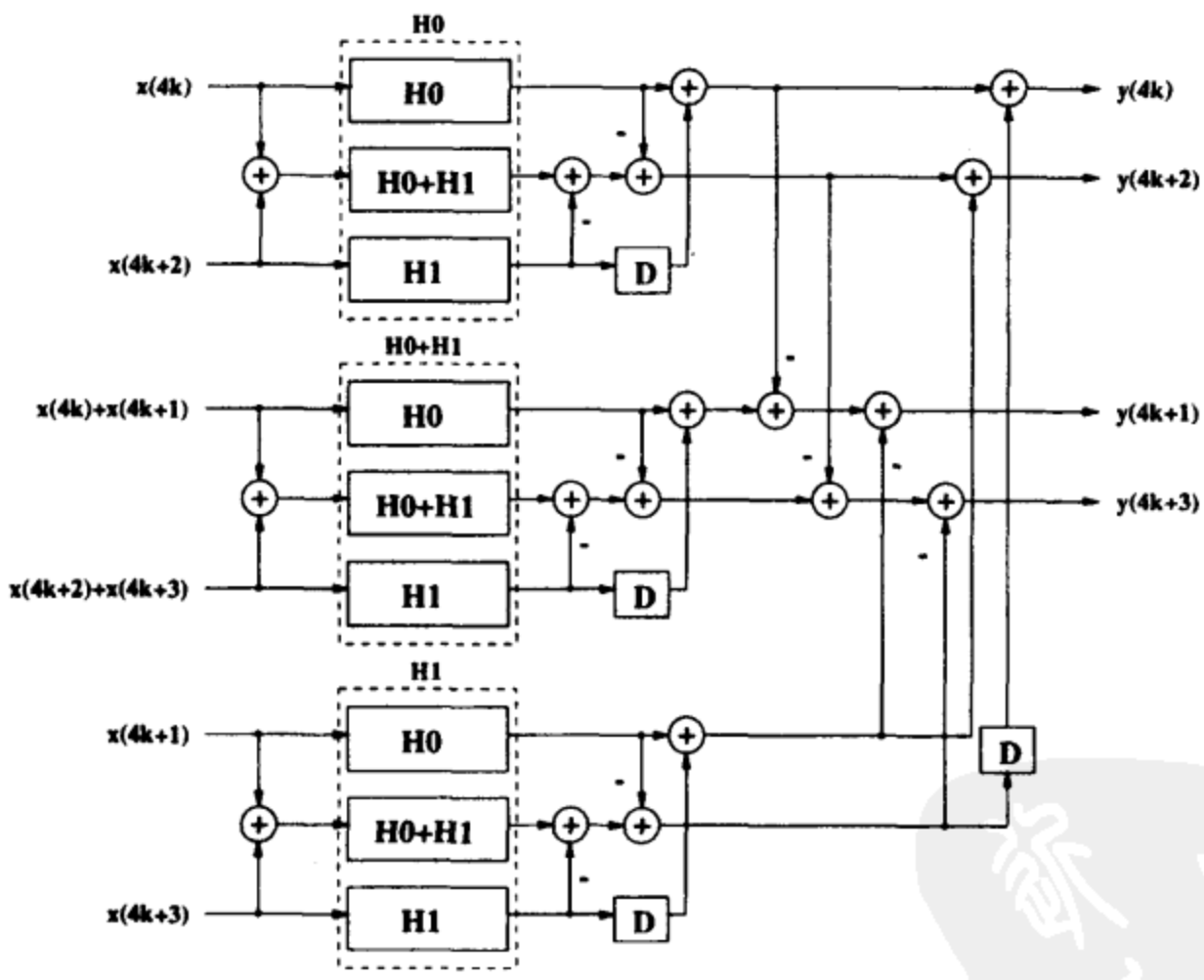
$$
\begin{aligned}
X_1'H_1' = & (X_1 + z^{-2}X_3)(H_1 + z^{-2}H_3) \\
= & X_1H_1 + z^{-2}[(X_1 + X_3)(H_1 + H_3) \\
& -X_1H_1 - X_3H_3] + z^{-4}X_3H_3
\end{aligned} \tag{9-41}
$$

应用2，滤波运算 $(X'_0 + X'_1)(H'_0 + H'_1)$

$$
\begin{aligned}
& (X_0' + X_1')(H_0' + H_1') \\
= & [(X_0 + X_1) + z^{-2}(X_2 + X_3)][(H_0 + H_1) + z^{-2}(H_2 + H_3)] \\
= & (X_0 + X_1)(H_0 + H_1) \\
& +z^{-2}[(X_0 + X_1 + X_2 + X_3)(H_0 + H_1 + H_2 + H_3) \\
& -(X_0 + X_1)(H_0 + H_1) - (X_2 + X_3)(H_2 + H_3)] \\
& +z^{-4}(X_2 + X_3)(H_2 + H_3)
\end{aligned} \tag{9-42}
$$

2并行FFA的第二次应用所得到的4并行滤波结构如图9-6所示，该结构需要9个长度为$N/4$的滤波运算。降低复杂度的4并行FIR滤波器用矩阵形式表示如下：

$$\mathbf{Y_{4p}} = \mathbf{B_4}(\mathbf{I_{3x3}} \otimes \mathbf{Q_4})\mathbf{H_4}(\mathbf{P_2} \otimes \mathbf{P_2})\mathbf{X_{4p}}$$



270

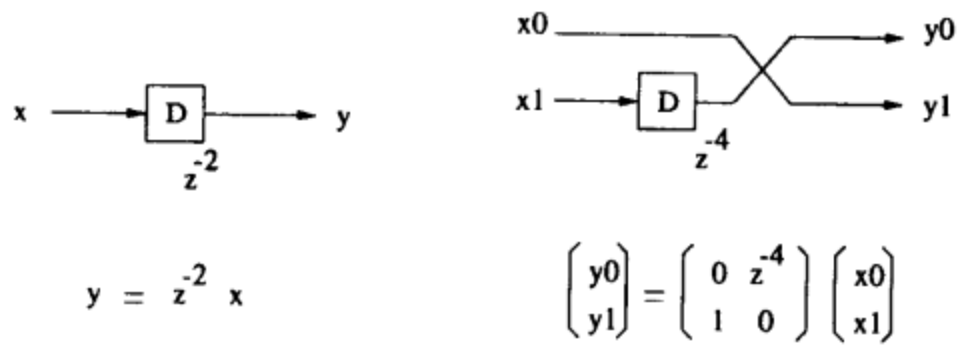
图9-6 降低复杂度的4并行FIR滤波器

其中，

$$
\mathbf{Y_{4p}} = \begin{bmatrix} Y_0 \\ Y_2 \\ Y_1 \\ Y_3 \end{bmatrix}, \quad \mathbf{B_4} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & z^{-4} \\ 0 & 1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 \end{bmatrix}
$$

$$\mathbf{Q_4} = \begin{bmatrix} 1 & 0 & z^{-4} \\ -1 & 1 & -1 \end{bmatrix}, \quad \mathbf{H_4} = diag \begin{bmatrix} H_0 \\ H_0 + H_2 \\ H_2 \\ H_0 + H_1 \\ H_0 + H_1 + H_2 + H_3 \\ H_2 + H_3 \\ H_1 \\ H_1 + H_3 \\ H_3 \end{bmatrix} \tag{9-43}$$

$$\mathbf{P_2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{X_{4P}} = \begin{bmatrix} X_0 \\ X_2 \\ X_1 \\ X_3 \end{bmatrix}$$



a) 2并行设计中的 z^{-2} 表述

b) 得到4并行设计的2伸展表述，该描述中的延迟元素对应 z^{-4}

图9-7 延迟元素的伸展

注意 $\mathbf{X_{4P}}$ 和 $\mathbf{Y_{4P}}$ 分别是 $\mathbf{X_4}$ 和 $\mathbf{Y_4}$ 的序列改变版本。注意通过将1替换为 $\mathbf{I_{2\times2}}$、0替换为 $\mathbf{0_{2\times2}}$ 以及对2并行设计的延迟算子 z^{-2} 进行适当的伸展，可以从 $\mathbf{Q_2}$ 得到 $\mathbf{B_4}$，如图9-7所示（关于伸展的细节，请参见第5章）。⊗ 算子是张量积或者Kronecker积算子[9]。由于张量积可以将大矩阵用更小矩阵表示，所以它在信号处理应用中极其有用。张量积定义如下：

定义9.2.1 张量积

令 $\mathbf{A}$ 和 $\mathbf{B}$ 是两个维数分别为 $k\times l$ 和 $m\times n$ 的任意矩阵。 271

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,l-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,l-1} \\ \cdot & \cdot & \cdots & \cdot \\ a_{k-1,0} & a_{k-1,1} & \cdots & a_{k-1,l-1} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-1} \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-1} \\ \cdot & \cdot & \cdots & \cdot \\ b_{m-1,0} & b_{m-1,1} & \cdots & b_{m-1,n-1} \end{bmatrix}$$

那么矩阵 $\mathbf{A}$ 和 $\mathbf{B}$ 的张量积，$\mathbf{C}=\mathbf{A}\otimes\mathbf{B}$，定义为 $km\times ln$ 矩阵如下：

$$\mathbf{C}=\begin{bmatrix} a_{0,0}\mathbf{B} & a_{0,1}\mathbf{B} & \cdots & a_{0,l-1}\mathbf{B} \\ a_{1,0}\mathbf{B} & a_{1,1}\mathbf{B} & \cdots & a_{1,l-1}\mathbf{B} \\ \cdot & \cdot & \cdots & \cdot \\ a_{k-1,0}\mathbf{B} & a_{k-1,1}\mathbf{B} & \cdots & a_{k-1,l-1}\mathbf{B} \end{bmatrix}$$

采用张量积，可以将降低复杂度的4并行FIR滤波器表示为相对紧缩的形式。应该注意的是，矩阵 $\mathbf{Q}_4$ 和矩阵 $\mathbf{Q}_2$ 除了仅仅在矩阵延迟算子的幂次上不同外，其本质上是相同的。图9-6所示的4并行结构可以被看作为3个分立的2并行FFA，每个2并行FFA都具有两个输出，所有输出通过 $\mathbf{B}_4$ 合并产生4个滤波器输出。如前所述，这种结构比传统的4并行FIR滤波器节省了将近44%的硬件开销。

在本节剩下的部分，我们给出了2个更多并行的滤波结构：6并行FIR滤波器和8并行FIR滤波器。6并行FIR滤波器可以由一个2并行FFA和一个3并行FFA级联而成。该处理过程与生成4并行滤波结构的过程在本质上是相同的。从公式（9-15）开始，应用2并行FFA产生3个滤波运算，然后将3并行FFA逐个应用于这些滤波器中，这样就产生了6并行滤波结构所需的18个滤波运算。注意，当2并行FFA和3并行FFA级联时，总是先应用2并行FFA，因为这样可以实现成本最低（见公式（9-35）和（9-36））。所得到的6并行滤波结构需要18N/6，也就是3N次乘法和42+18(N/6−1) = 3N+24次加法。与传统的6并行滤波结构相比，这种降低复杂度的6并行滤波结构节省了将近50%的面积。降低复杂度的6并行FIR滤波器用矩阵形式表示如下：

272

$$\mathbf{Y}_{6p}=\mathbf{B}_6(\mathbf{I}_{3\times3}\otimes\mathbf{Q}_6)\mathbf{H}_6(\mathbf{P}_2\otimes\mathbf{P}_3)\mathbf{X}_{6p}$$

其中，

$$\mathbf{Y}_{6p}=\begin{bmatrix} Y_0 \\ Y_2 \\ Y_4 \\ Y_1 \\ Y_3 \\ Y_5 \end{bmatrix},\quad \mathbf{B}_6=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z^{-6} \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{Q}_6=\begin{bmatrix} 1 & 0 & z^{-6} & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -z^{-6} & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_6=diag\begin{bmatrix} H_0 \\ H_2 \\ H_4 \\ \vdots \\ H_1+H_3 \\ H_3+H_5 \\ H_1+H_3+H_5 \end{bmatrix},\quad \mathbf{P}_2=\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{9-44}$$

$$\mathbf{P}_3=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},\quad \mathbf{X}_{6p}=\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_1 \\ X_3 \\ X_5 \end{bmatrix}$$

注意其中的 $\mathbf{X}_{6P}$ 和 $\mathbf{Y}_{6P}$ 分别为 $\mathbf{X}_6$ 和 $\mathbf{Y}_6$ 的序列改变版本，除了将延迟算子从 z^{-3} 改为 z^{-6} 以外，$\mathbf{Q}_6$ 和 $\mathbf{Q}_3$ 相同。通过将 $\mathbf{Q}_2$ 中的1替换为 $\mathbf{I}_{3\times3}$、0替换为 $\mathbf{0}_{3\times3}$ 以及 z^{-2} 替换为其3伸展形式，就得到了矩阵 $\mathbf{B}_6$。降低复杂度的6并行FIR滤波器结构如图9-8所示。

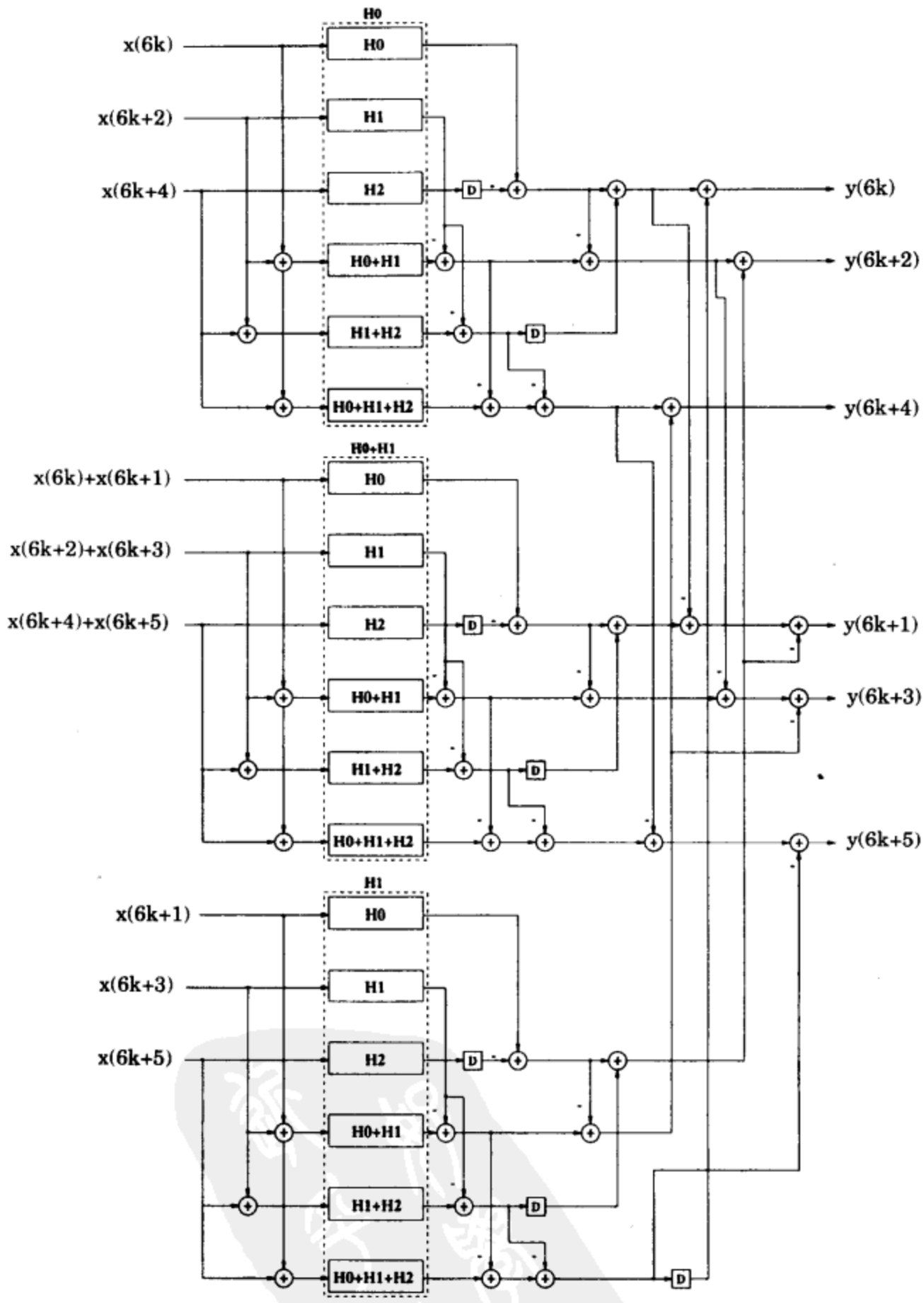


图9-8 降低复杂度的6并行FIR滤波器

该结构中的子滤波器为 H_0、H_0+H_1 和 H_1，这些子滤波器都从原始滤波器得到

8并行FIR 滤波器可通过级联三个2并行FFA得到。这就产生了一个具有27个滤波运算的

滤波结构。8并行FIR滤波结构需要27/N次乘法和76+27(N/8−1)=27/N+49次加法。与传统的8并行FIR滤波器相比，这种滤波结构节省了58%的面积。降低复杂度的8并行FIR滤波器用矩阵形式表示为:

273 ~ 274

$$\mathbf{Y}_{8p} = \mathbf{B}_8(\mathbf{I}_{3\times3} \otimes \mathbf{Q}_{8_2})[\mathbf{I}_{3\times3} \otimes (\mathbf{I}_{3\times3} \otimes \mathbf{Q}_{8_1})]\mathbf{H}_8[\mathbf{P}_2 \otimes (\mathbf{P}_2 \otimes \mathbf{P}_2)]\mathbf{X}_{8p}$$

其中，

$$\mathbf{Y}_{8p} = \begin{bmatrix} Y_0 \\ Y_4 \\ Y_2 \\ Y_6 \\ Y_1 \\ Y_5 \\ Y_3 \\ Y_7 \end{bmatrix}, \quad \mathbf{Q}_{8_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & z^{-8} \\ 0 & 1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 \end{bmatrix}, \quad \mathbf{X}_{8p} = \begin{bmatrix} X_0 \\ X_4 \\ X_2 \\ X_6 \\ X_1 \\ X_5 \\ X_3 \\ X_7 \end{bmatrix}$$

$$\mathbf{B}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z^{-8} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{9-45}$$

$$\mathbf{Q}_{8_1} = \begin{bmatrix} 1 & 0 & z^{-8} \\ -1 & 1 & -1 \end{bmatrix}, \quad \mathbf{P}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{H}_8 = diag \begin{bmatrix} H_0 \\ H_0 + H_4 \\ \vdots \\ H_3 + H_7 \\ H_7 \end{bmatrix}$$

降低复杂度的8并行FIR滤波器结构如图9-9所示。矩阵 $\mathbf{Q}_{8_1}$ 和 $\mathbf{Q}_{8_2}$ 分别与矩阵 $\mathbf{Q}_2$ 和 $\mathbf{Q}_4$ 相同，而 $\mathbf{Q}_2$ 和 $\mathbf{Q}_4$ 分别是用于2并行和4并行FIR滤波器的矩阵表示的。在这两个矩阵中，它们仅仅在延迟算子的幂次上不同。

9.2.2.5 快速并行FIR滤波器的应用

快速并行FIR滤波器可以用在许多应用中以减少乘法运算的数量。它可以用来设计有效的小波变换[10]（见习题16和17）、子波段编码器、自适应均衡器[11](见习题18和19)、二维并行FIR滤波器（见习题23）和视频压缩中的运动估计[12]（见习题24）。

9.3 离散余弦变换和反离散余弦变换

275

离散余弦变换是用于静态和动态视频压缩中的频率变换[13]（见1.2.6节）。本节阐述了基于算法结构变换和频率抽取方法的快速离散余弦变换的实现。

指定数据序列 $x(n)$ (n=0, 1, ... , N−1) 的DCT为 $X(k)$ (k=0, 1, ... , N−1)。DCT和IDCT算法用下列等式来表示:

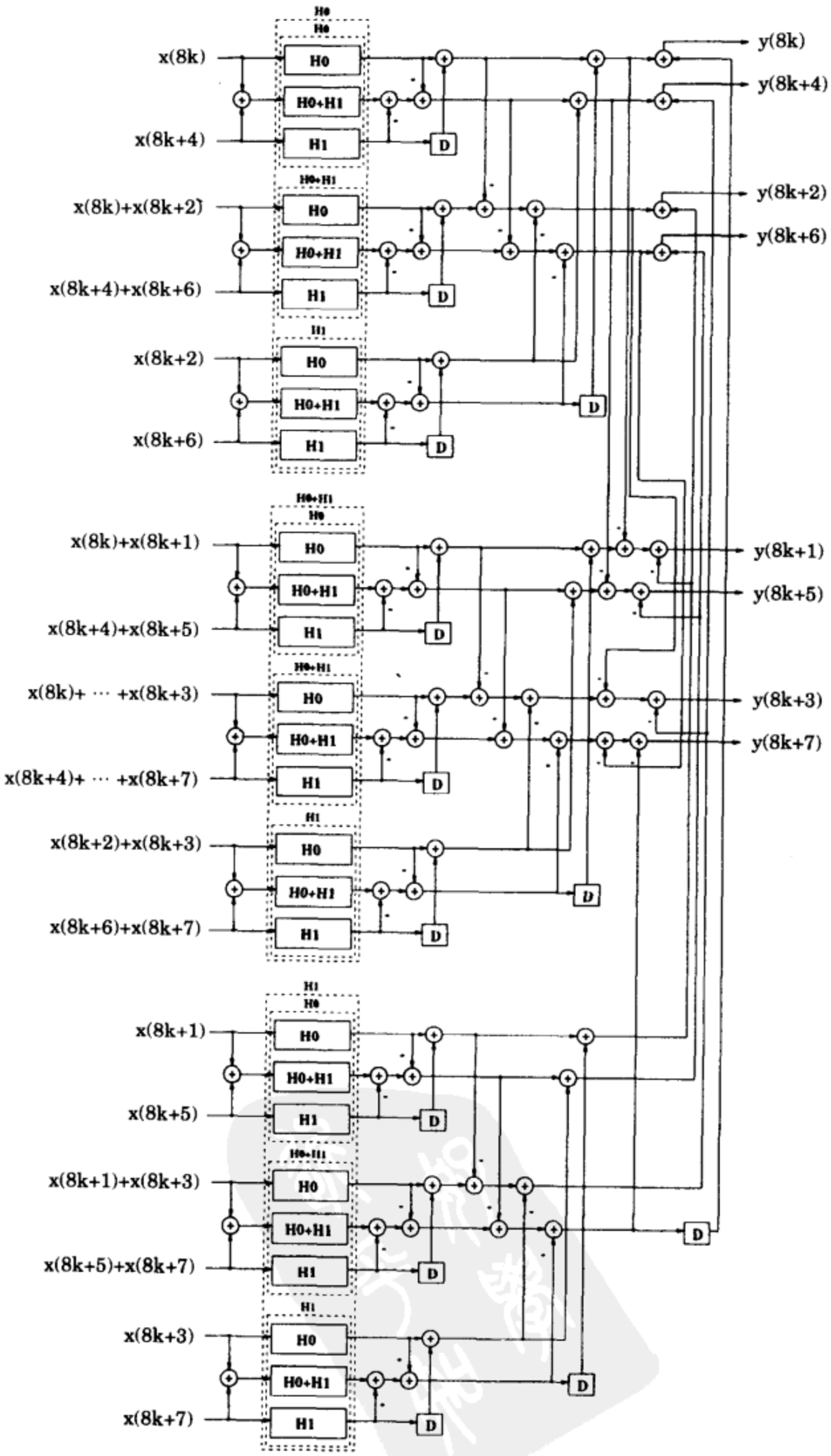


图9-9 降低复杂度的8并行FIR滤波器

DCT:

$$X(k)=e(k)\sum_{n=0}^{N-1}x(n)\cos[\frac{(2n+1)\pi k}{2N}],\ k=0,1,\cdots,N-1 \tag{9-46}$$

IDCT:

$$x(n)=\frac{2}{N}\sum_{k=0}^{N-1}e(k)X(k)\cos[\frac{(2n+1)\pi k}{2N}],\ n=0,1,\cdots,N-1 \tag{9-47}$$

其中

$$e(k)=\begin{cases}\frac{1}{\sqrt{2}}, & \text{如果 } k=0,\\ 1, & \text{否则}\end{cases} \tag{9-48}$$

注意DCT是正交变换，例如，IDCT的变换矩阵是DCT变换矩阵转置的缩放版本，或者相反。因此，DCT结构可以通过转置IDCT得到，例如，将IDCT流图中的箭头反向，IDCT也可以通过转置DCT得到。

容易证明直接实现DCT和IDCT需要 $N(N-1)$ 次乘法运算，即 $O(N^2)$，硬件成本是非常昂贵的。因此，能减少所需计算数量的算法成为DCT特别感兴趣的问题。

9.3.1 算法结构变换

算法结构变换可以用来获得有效的DCT实现，所用乘法的数量急剧减少[14]。算法结构映射技术采用层次方法使得一个给定的算法，或者在系统方法上改变算法的描述适应一个结构。例9.3.1采用这种技术导出了一个8点DCT结构，它的乘法数量从56减少到了13。

例9.3.1 考虑8点DCT

276 ~ 277

$$X(k)=e(k)\sum_{n=0}^{7}x(n)\cos[\frac{(2n+1)\pi k}{16}],\ k=0,1,\cdots,7 \tag{9-49}$$

其中

$$e(k)=\begin{cases}\frac{1}{\sqrt{2}}, & \text{如果} k=0\\ 1, & \text{否则}\end{cases} \tag{9-50}$$

将其用矩阵形式表示如下：

$$\begin{bmatrix}X(0)\\X(1)\\X(2)\\X(3)\\X(4)\\X(5)\\X(6)\\X(7)\end{bmatrix}=\begin{bmatrix}c_4&c_4&c_4&c_4&c_4&c_4&c_4&c_4\\c_1&c_3&c_5&c_7&c_9&c_{11}&c_{13}&c_{15}\\c_2&c_6&c_{10}&c_{14}&c_{18}&c_{22}&c_{26}&c_{30}\\c_3&c_9&c_{15}&c_{21}&c_{27}&c_1&c_7&c_{13}\\c_4&c_{12}&c_{20}&c_{28}&c_4&c_{12}&c_{20}&c_{28}\\c_5&c_{15}&c_{25}&c_3&c_{13}&c_{23}&c_1&c_{11}\\c_6&c_{18}&c_{30}&c_{10}&c_{22}&c_2&c_{14}&c_{26}\\c_7&c_{21}&c_3&c_{17}&c_{31}&c_{13}&c_{27}&c_9\end{bmatrix}\begin{bmatrix}x(0)\\x(1)\\x(2)\\x(3)\\x(4)\\x(5)\\x(6)\\x(7)\end{bmatrix} \tag{9-51}$$

其中 $c_i=\cos(i\pi/16)$。对于8点DCT，算法结构映射的实现分为三步。第一步，在系统方法上修改DCT算法使其更容易实现。利用三角学的性质，我们有

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \tag{9-52}$$

那么，8点DCT可以重新写成

$$\begin{aligned} X(1) &= M_0c_1 + M_1c_7 + M_2c_3 + M_3c_5 \\ X(7) &= M_0c_7 - M_1c_1 - M_2c_5 + M_3c_3 \\ X(3) &= M_0c_3 - M_1c_5 - M_2c_7 - M_3c_1 \\ X(5) &= M_0c_5 + M_1c_3 - M_2c_1 + M_3c_7 \\ X(2) &= M_{10}c_2 + M_{11}c_6 \\ X(6) &= M_{10}c_6 - M_{11}c_2 \\ X(4) &= M_{100}c_4 \\ X(0) &= P_{100}c_4 \end{aligned} \tag{9-53}$$

其中

$$\begin{aligned} M_0 &= x_0 - x_7 & P_0 &= x_0 + x_7 \\ M_1 &= x_3 - x_4 & P_1 &= x_3 + x_4 \\ M_2 &= x_1 - x_6 & P_2 &= x_1 + x_6 \\ M_3 &= x_2 - x_5 & P_3 &= x_2 + x_5 \\ M_{10} &= P_0 - P_1 & P_{10} &= P_0 + P_1 \\ M_{11} &= P_2 - P_3 & P_{11} &= P_2 + P_3 \\ M_{100} &= p_{10} - P_{11} & P_{100} &= P_{10} + P_{11} \end{aligned} \tag{9-54}$$

278

图9-10是根据公式（9-53）和（9-54）得出的具有22次乘法的DCT结构。

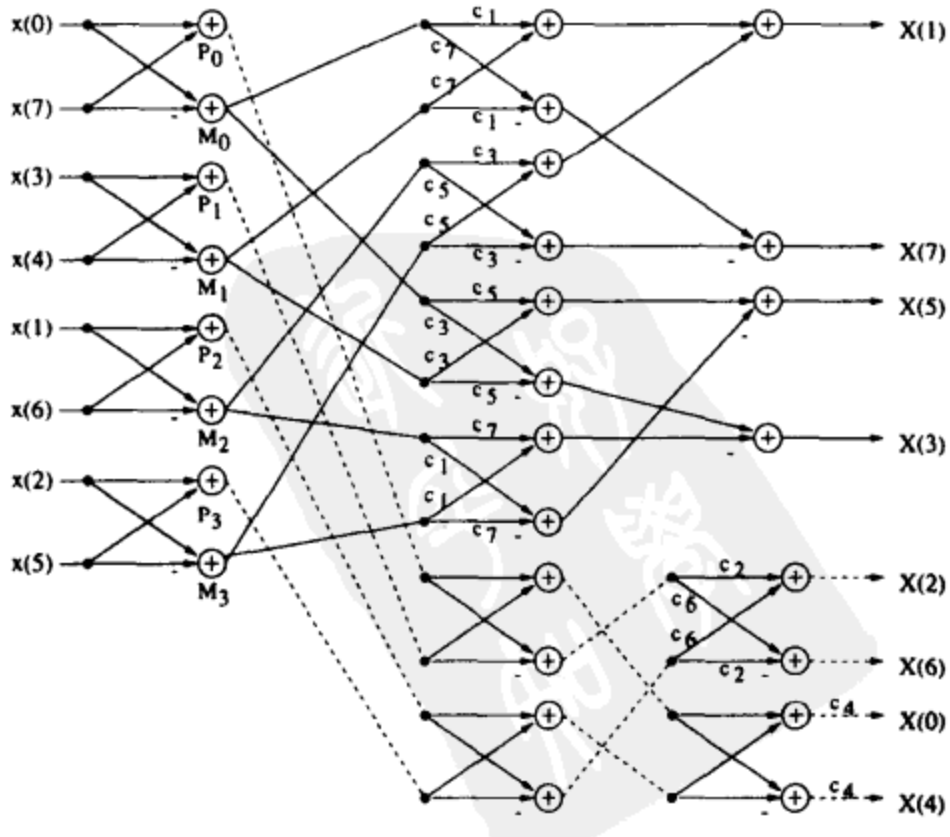


图9-10 例9.3.1中8点DCT的第一步

第二步，将图9-10所示的DCT结构分组为不同的功能单元，每个单元表示为一个块，然后，整个DCT结构被变换为块框图。其中两个主要的块如图9-11所示。变换得到的8点DCT的块框图如图9-12所示。第三步，探索每个不同块的降低复杂度的实现。如图9-13a所示，用3次乘法和3次加法代替4次乘法和2次加法来实现块 $\boxed{XC\pm}$。更进一步，用 $a=\sin\theta$、$b=\cos\theta$ 定义块 $\boxed{XC\pm}$，并且翻转输出为一个旋转块 $\boxed{\mathrm{rot}\theta}$，该块的计算如图9-13b所示。

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{9-55}$$

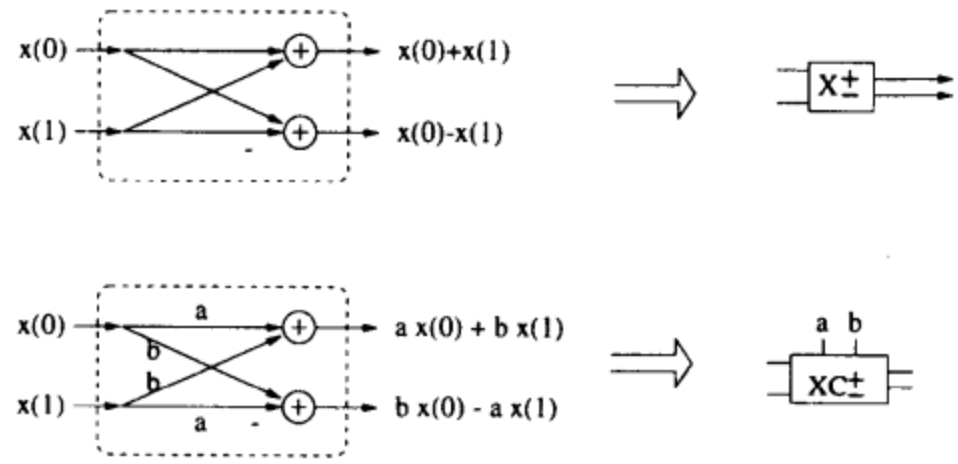


图9-11 例9.3.1中的块定义

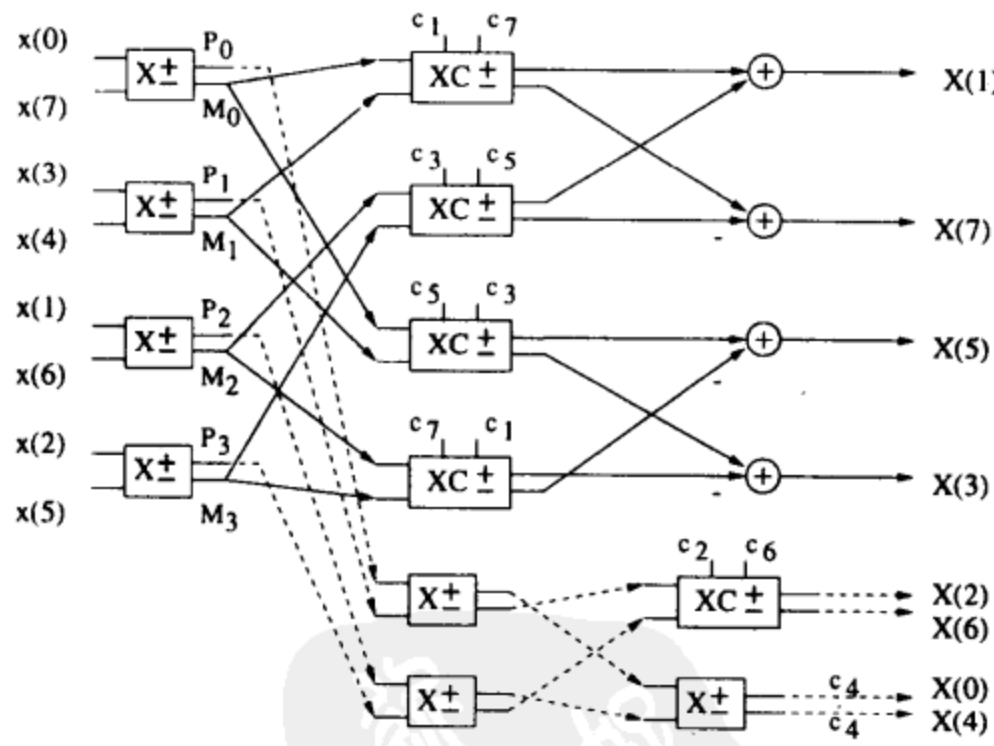


图9-12 例9.3.1中8点DCT结构的第二步

279～280

注意到级联起来的旋转块的相角可以简单相加，如图9-13c中变换块所示。图9-13d是基于一个事实得出的，$\theta=\pi/4$ 的旋转块正好是输出被翻转并乘以$\cos(\pi/4)$的块 $\boxed{X\pm}$。

将图9-13中的所有变换应用到图9-12中的DCT块框图中，我们得到图9-14所示的最终结构，该结构仅仅需要13次乘法。 ■

如果用缩放的DCT，用 $X(k)/c_4$ 的计算替代 $X(k)$ 的计算，乘法器的数量还可以进一步减少到11（见习题29）。

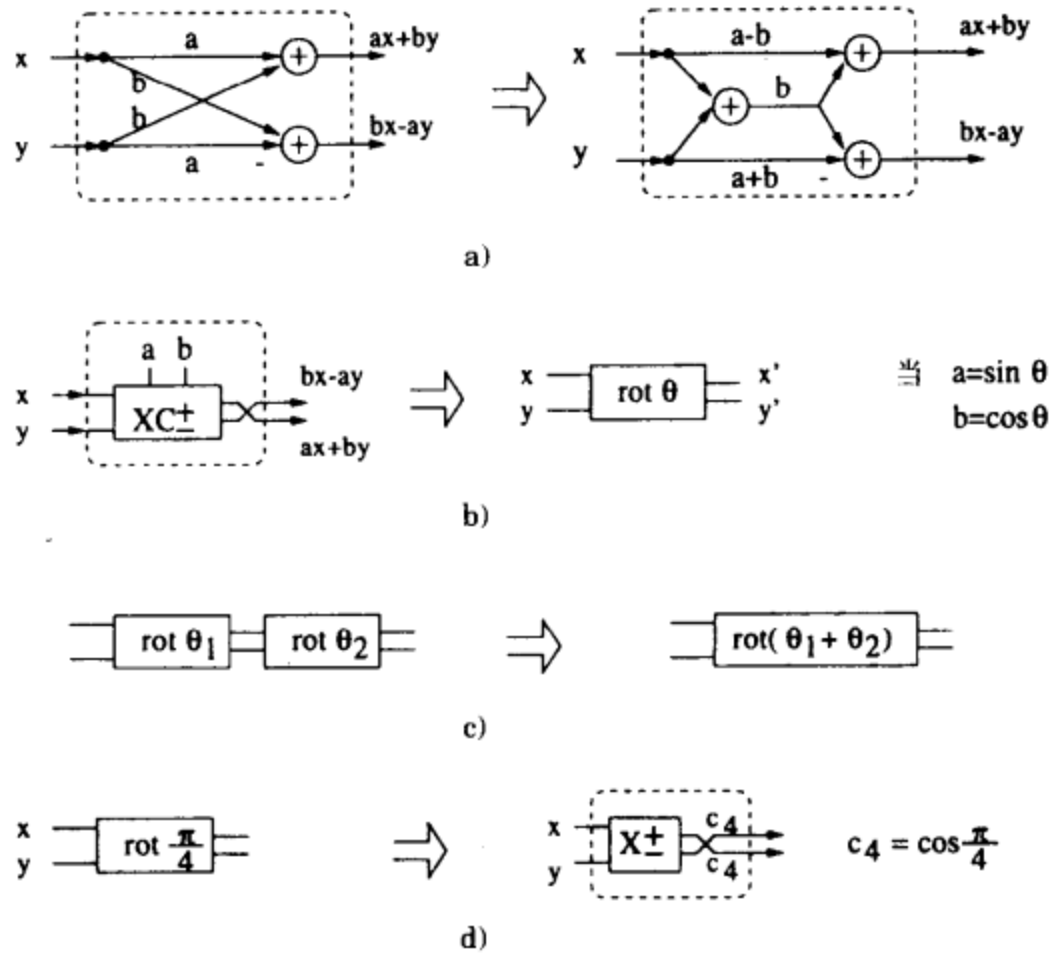


图9-13 例9.3.1中所用的图形变换

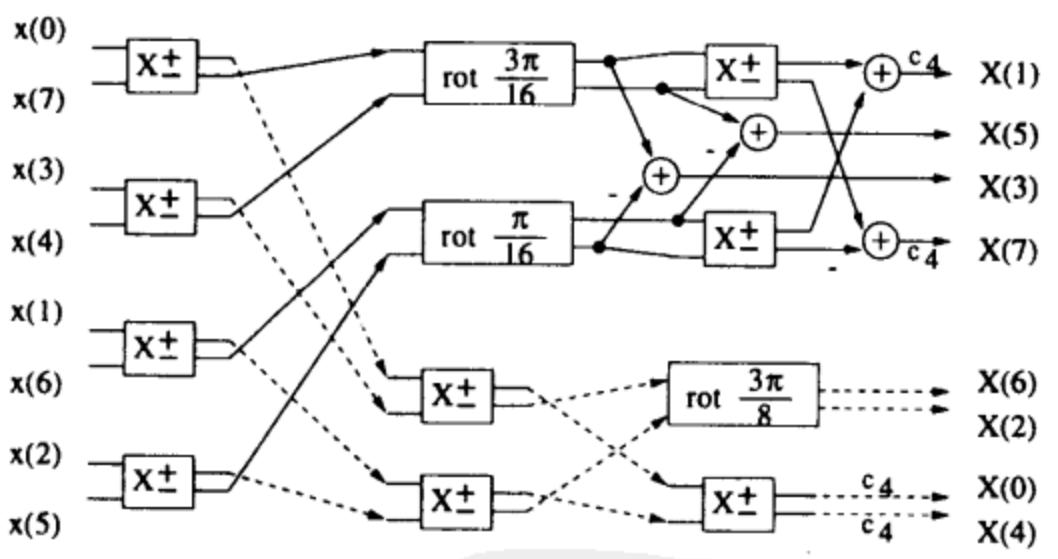


图9-14 例9.3.1中最终的8点DCT结构

9.3.2 对 2^m 点DCT的频率抽取快速DCT

快速 2^m 点DCT/IDCT结构可以通过频率抽取方法得到[15],[16]，频率抽取方法也常常用于DFT。该算法通过2的幂次的分解，将乘法的数量减少到大约 $(N/2)\log_2 N$。

首先考虑快速IDCT计算。为简单起见，在下面的推导过程中忽略了公式（9-47）中的缩放因子$2/N$。定义

$$\hat{X}(k) = e(k)X(k) \tag{9-56}$$

并将 $x(n)$ 按序号 k 分解为奇数和偶数部分如下:

$$
\begin{aligned}
x(n) &= \sum_{k=0}^{N-1} \hat{X}(k)\cos\frac{(2n+1)\pi k}{2N} \\
&= \sum_{k=0}^{N/2-1} \hat{X}(2k)\cos\frac{(2n+1)\pi(2k)}{2N} \\
&\quad + \sum_{k=0}^{N/2-1} \hat{X}(2k+1)\cos\frac{(2n+1)\pi(2k+1)}{2N} \\
&= \sum_{k=0}^{N/2-1} \hat{X}(2k)\cos\frac{(2n+1)\pi k}{2(N/2)} \\
&\quad + \frac{1}{2\cos\frac{(2n+1)\pi}{2N}} \sum_{k=0}^{N/2-1} 2\hat{X}(2k+1)\cos\frac{(2n+1)\pi(2k+1)}{2N}\cos\frac{(2n+1)\pi}{2N}
\end{aligned}
\tag{9-57}
$$

注意

$$
2\cos\frac{(2n+1)\pi(2k+1)}{2N}\cos\frac{(2n+1)\pi}{2N} = \cos\frac{(2n+1)\pi(k+1)}{N} + \cos\frac{(2n+1)\pi k}{N} \tag{9-58}
$$

281~282

由于 $\cos\frac{(2n+1)\pi(N/2-1+1)}{N}=0$，因此

$$
\begin{aligned}
&\sum_{k=0}^{N/2-1} 2\hat{X}(2k+1)\cos\frac{(2n+1)\pi(2k+1)}{2N}\cos\frac{(2n+1)\pi}{2N} \\
&= \sum_{k=0}^{N/2-1} \hat{X}(2k+1)\cos\frac{(2n+1)\pi(k+1)}{N} + \sum_{k=0}^{N/2-1} \hat{X}(2k+1)\cos\frac{(2n+1)\pi k}{N} \\
&= \sum_{k=0}^{N/2-2} \hat{X}(2k+1)\cos\frac{(2n+1)\pi(k+1)}{N} + \sum_{k=0}^{N/2-1} \hat{X}(2k+1)\cos\frac{(2n+1)\pi k}{N}
\end{aligned}
\tag{9-59}
$$

将 $k'=k+1$ 代入公式（9-60）中的第一项，它就变为

$$
\begin{aligned}
&\sum_{k=0}^{N/2-2} \hat{X}(2k+1)\cos\frac{(2n+1)\pi(k+1)}{N} \\
&= \sum_{k'=1}^{N/2-1} \hat{X}(2k'-1)\cos\frac{(2n+1)\pi k'}{N} \\
&= \sum_{k'=0}^{N/2-1} \hat{X}(2k'-1)\cos\frac{(2n+1)\pi k'}{N}
\end{aligned}
\tag{9-60}
$$

其中 $\hat{X}(-1)=0$。那么公式（9-58）的IDCT可以写成

$$
\begin{aligned}
x(n) &= \sum_{k=0}^{N/2-1} \hat{X}(2k)\cos\frac{(2n+1)\pi k}{2(N/2)} \\
&\quad + \frac{1}{2\cos\frac{(2n+1)\pi}{2N}} \sum_{k=0}^{N/2-1} [\hat{X}(2k+1)+\hat{X}(2k-1)]\cos\frac{(2n+1)\pi k}{2(N/2)}
\end{aligned}
\tag{9-61}
$$

定义

$$G(k) = \hat{X}(2k)$$
$$H(k) = \hat{X}(2k+1) + \hat{X}(2k-1), \qquad 当\ k = 0, 1, \cdots, N/2-1 \tag{9-62}$$

以及

$$g(n) = \sum_{k=0}^{N/2-1} \hat{X}(2k) \cos \frac{(2n+1)\pi k}{2(N/2)}$$
$$h(n) = \sum_{k=0}^{N/2-1} [\hat{X}(2k+1) + \hat{X}(2k-1)] \cos \frac{(2n+1)\pi k}{2(N/2)} \tag{9-63}$$
$$当\ n = 0, 1, \cdots, N/2-1$$

283

显然，$G(k)$ 和 $H(k)$ 分别为 $g(n)$ 和 $h(n)$ 的DCT。同时，我们有

$$x(n) = g(n) + \frac{1}{2\cos\frac{(2n+1)\pi}{2N}} h(n) \tag{9-64}$$
$$x(N-1-n) = g(n) - \frac{1}{2\cos\frac{(2n+1)\pi}{2N}} h(n) \tag{9-65}$$
$$当\ n = 0, 1, \cdots, N/2-1$$

由于

$$\cos\frac{(2(N-1-n)+1)\pi k}{N} = \cos\frac{(2n+1)\pi k}{N}$$
$$\cos\frac{(2(N-1-n)+1)\pi}{2N} = -\cos\frac{(2n+1)\pi}{2N} \tag{9-66}$$

因此，公式（9-47）中的N点IDCT被表示为公式（9-63）中两个$N/2$点IDCT的形式。重复这个处理过程，可以进一步分解IDCT直到它表示为2点IDCT的形式。

类似地，DCT算法也可以分解。换句话说，如前面子小节所述，它可以通过转置IDCT来得到。

例9.3.2 构造2点IDCT蝶形结构。2点IDCT可计算如下：

$$\begin{aligned} x(0) &= \hat{X}(0) + \hat{X}(1)\cos\frac{\pi}{4} \\ x(1) &= \hat{X}(0) - \hat{X}(1)\cos\frac{\pi}{4} \end{aligned} \tag{9-67}$$

2点IDCT蝶形结构如图9-15所示。 ■

例9.3.3 用2点IDCT蝶形结构构造8点快速DCT结构。当N=8时，公式（9-62）~（9-66）重写为：

$$G(k) = \hat{X}(2k), \tag{9-68}$$
$$H(k) = \hat{X}(2k+1) + \hat{X}(2k-1),\ k = 0, 1, 2, 3 \tag{9-69}$$

和

$$g(n) = \sum_{k=0}^{3} G(k)\cos\frac{(2n+1)\pi k}{2(N/2)} \tag{9-70}$$

$$h(n) = \sum_{k=0}^{3} H(k)\cos\frac{(2n+1)\pi k}{2(N/2)} \tag{9-71}$$

$$x(n) = g(n) + \frac{1}{2\cos\frac{(2n+1)\pi}{16}}h(n) \tag{9-72}$$

$$x(7-n) = g(n) - \frac{1}{2\cos\frac{(2n+1)\pi}{16}}h(n),\ n=0,1,2,3 \tag{9-73}$$

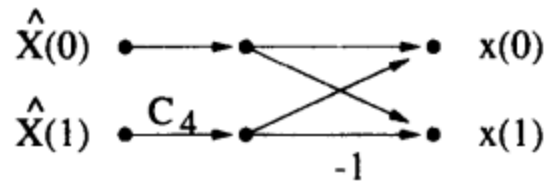


图9-15 2点IDCT蝶形结构

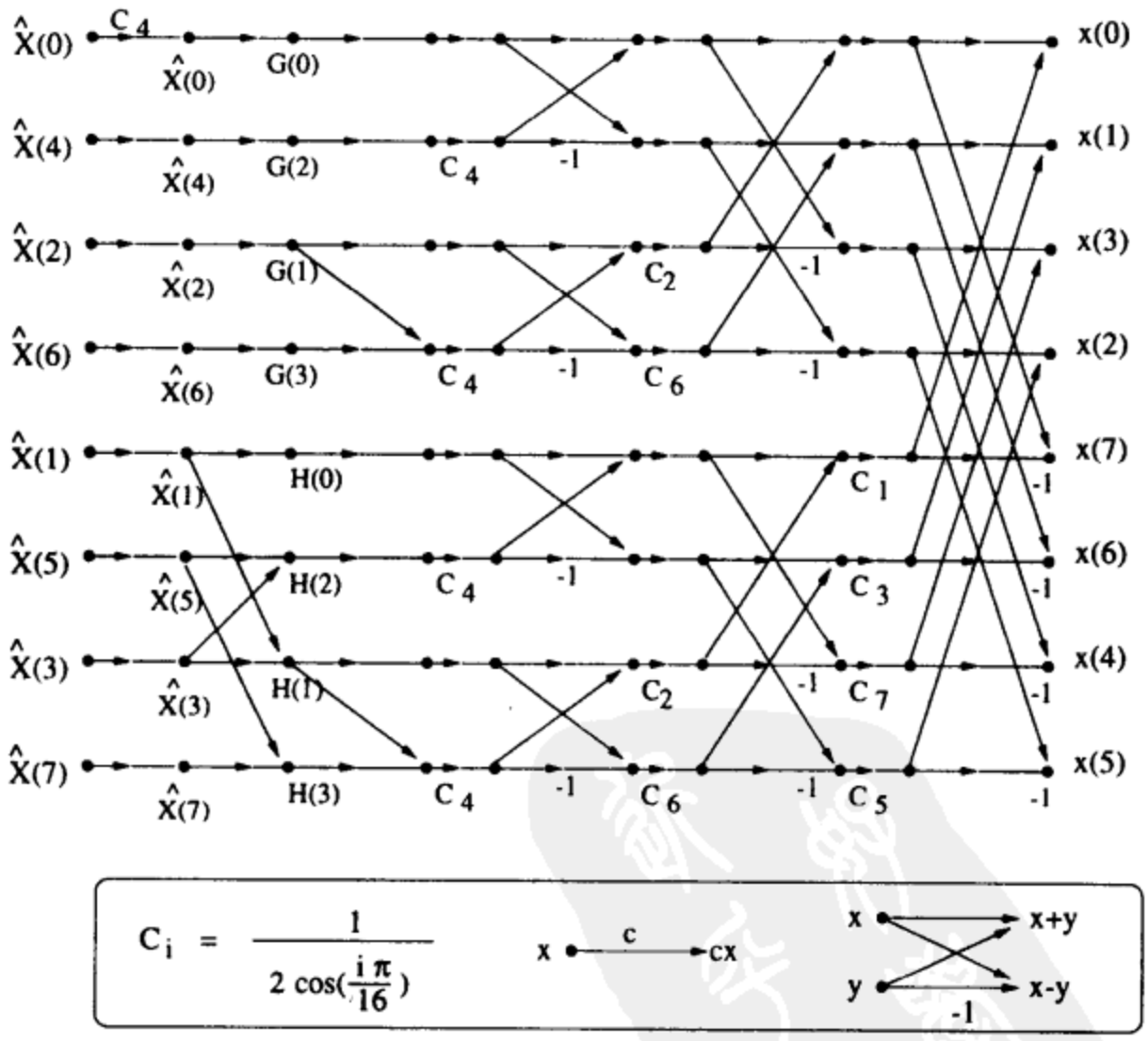


图9-16 8点快速IDCT流图。在本结构中没有包括缩放因子2/N=1/4

图9-16所示的8点快速IDCT仅需要13次乘法操作。转置该结构可以得到快速8点DCT结构如图9-17所示。（注意对于N=8的情况，在两个图中都有 $C_4\frac{1}{2\cos(4\pi/16)}=\cos\frac{\pi}{4}$。） ■

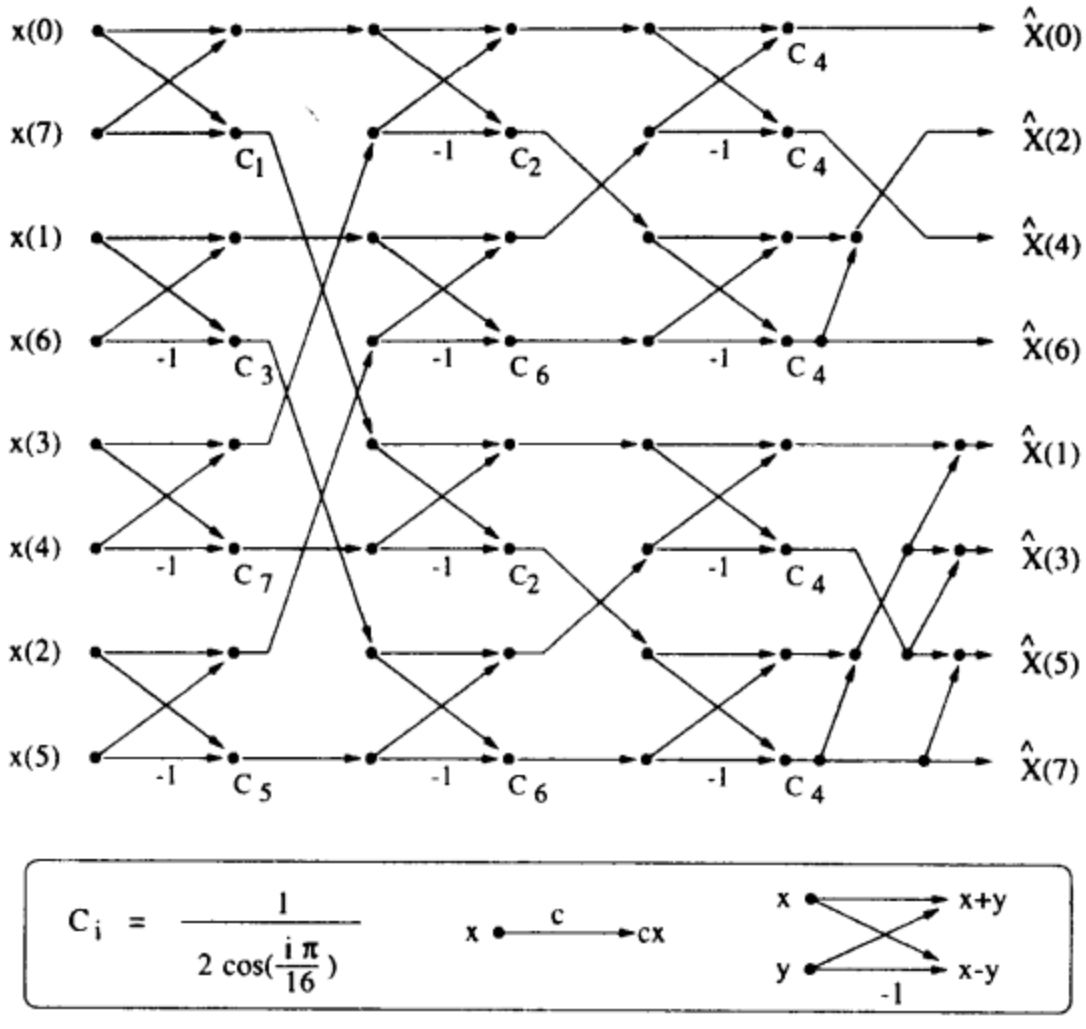


图9-17 8点快速DCT流图

9.4 秩-阶滤波器的并行结构

秩-阶滤波器广泛地应用于图像处理的平滑、减小噪声、边缘检测等方面。秩-阶滤波器是非线性滤波器，它们对输入序列进行分类，并根据它的秩选择输出。秩-阶滤波器的输入输出关系如公式（9-74）所示：

$$y_r(n) = r^{th}\text{秩} \quad [x(n-N), x(n-N+1), \cdots, x(n), \cdots, x(n+N-1), x(n+N)] \tag{9-74}$$

其中 $r=1, 2, \ldots, 2N+1$。该滤波器的窗尺寸为 $W=2N+1$。$W=5$所对应的系统框图如图9-18所示。本节主要关心的是可编程秩-阶滤波器，对于所有的 r 取值，其输出都是可用的。对于 r 的所有取值，所有输出 $\{y_r(n)\}$ 的集合简单地表示为 $y(n)$。

本节提出了基于Batcher奇偶合并分类算法[17]和算法强度缩减设计秩-阶滤波器的并行结构。通过共享不同并行块中的某些合并单元，可以缩减并行秩-阶滤波器的硬件复杂度。

9.4.1 奇偶合并分类结构

在合并分类运算的基础上，秩-阶滤波器根据输入的幅度对其进行分类。本小节介绍奇偶合并分类算法，可有效实现合并分类运算。

奇偶合并分类算法处理2个已经过预分类的序列，并将它们合并为一个分类序列。在该算法中有两个合并单元：奇合并单元和偶合并单元。在所有输入中，下标为奇数的输入到奇合并单元。在所有输入中，下标为偶数的输入到偶合并单元。除了奇合并单元的第一个输出和

284 ~ 286

偶合并单元的最后一个输出外，再次比较每个合并单元的其他输出，奇数合并单元的第一个输出是最大的元素，偶数合并单元的最后一个输出是最小的元素。将奇合并单元的第 i 个输出与偶合并单元的第（i-1）个输出进行比较得到分类后的输出。大的合并分类单元可以用小的合并分类单元来组成。最小的合并分类单元为1×1合并分类单元，通常它被表示为**C&S**并被称为比较和互换单元。合并分类结构的硬件复杂度和延迟的推导作为练习留给读者自己推导（见习题35）。

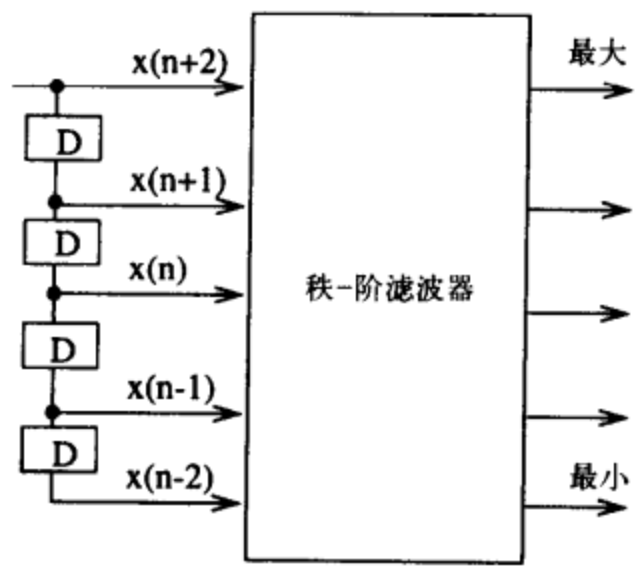


图9-18 一个窗口大小 W=5 的秩-阶滤波器

例9.4.1 图9-19是一个4×4合并分类电路。它包括一个奇2×2合并单元和一个偶2×2合并单元以及一些用于分类的**C&S**单元。2×2合并单元可以用3个**C&S**单元来搭建，如图9-20所示。注意这里假定2个输入序列是经过预分类的，如 $a_1 \geqslant a_2$ 且 $b_1 \geqslant b_2$.（否则，另外需要两个**C&S**单元来依次对它们进行分类，这就意味着一个完整的4元素分类器需要总共5个**C&S**单元。）因此，图9-19所示的4×4合并分类电路需要9个**C&S**单元。 ■

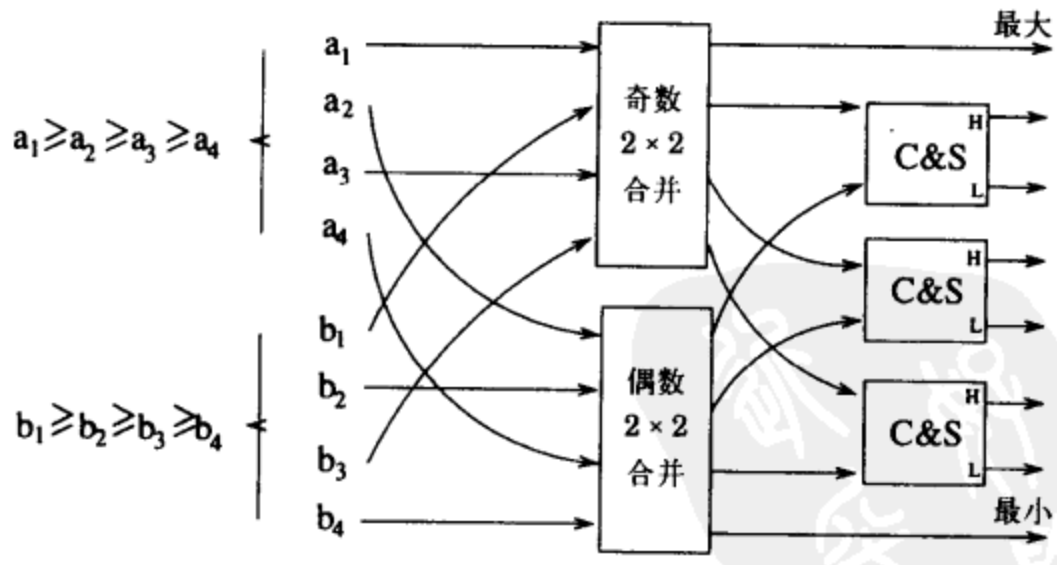


图9-19 一个4×4合并分类电路

例9.4.2 考虑设计如图9-21所示的4×2合并单元。该结构可以很容易从4×4合并单元得来，通过简单的将最后两个数据元素当作哑元，并抛弃所有其输入至少包含这两个哑元之一的**C&S**单元。不难证明一个4×2合并单元需要6个**C&S**单元。 ■

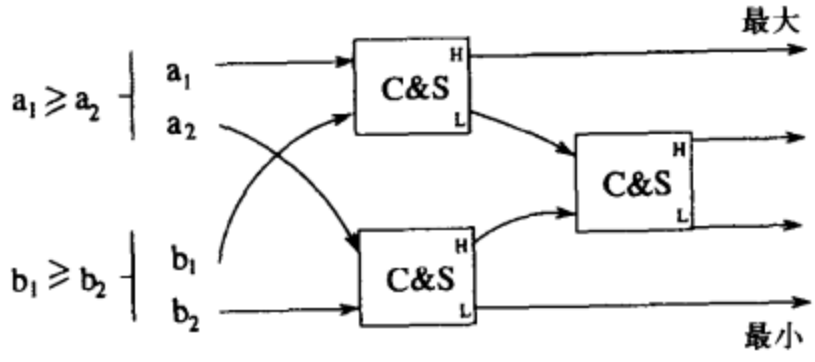


图9-20 2×2合并单元

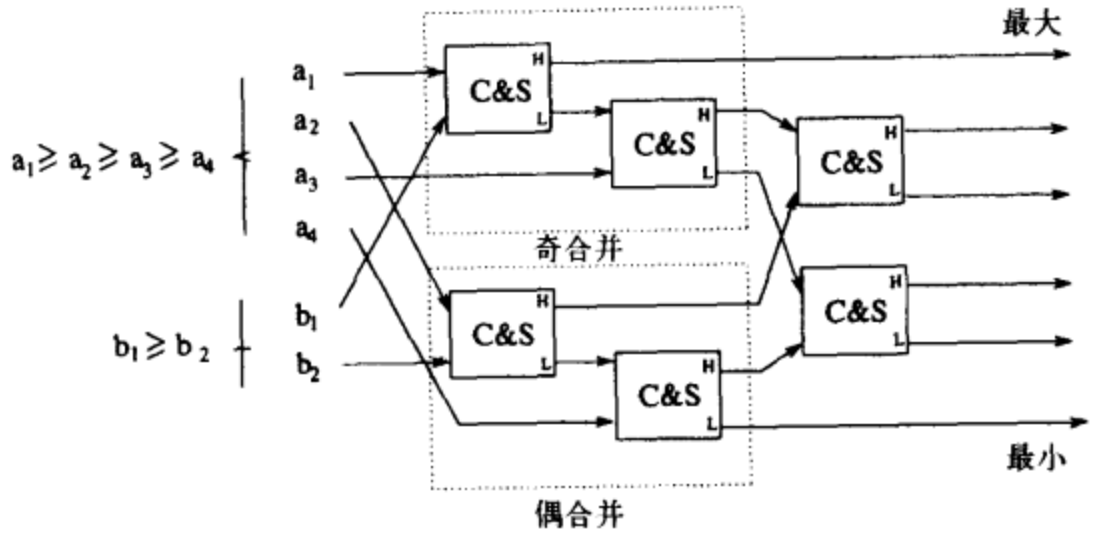


图9-21 4×2合并单元

287 ~ 288

注意一个合并器合并两个输入矢量为一个输出矢量，该输出矢量的元素是升序（或降序）排列。因此，一个 $m \times n$ 合并器与 $n \times m$ 合并器具有相同的硬件复杂度。

9.4.2 秩-阶滤波器结构

本小节讨论基于Batcher奇偶合并分类算法的秩-阶滤波器设计。

例9.4.3 图9-22是一个窗尺寸W=5的秩-阶滤波器结构。注意从滤波器的输入分接出了一根延迟线，用来形成分类器的窗。该秩-阶滤波器总共需要9个**C&S**单元，因为4×1合并单元需要4个**C&S**单元（见图9-23）。 ■

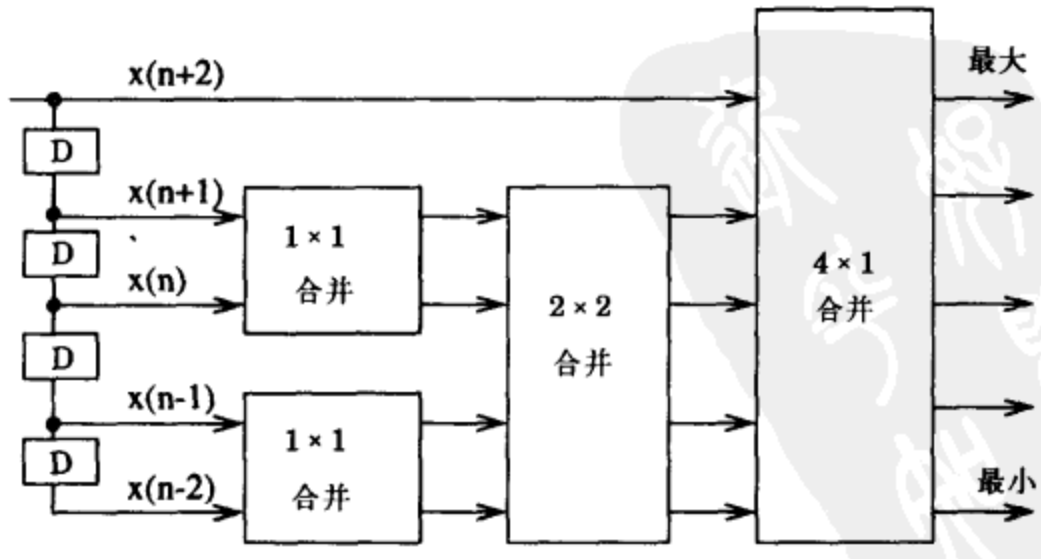


图9-22 W=5，秩-阶滤波器

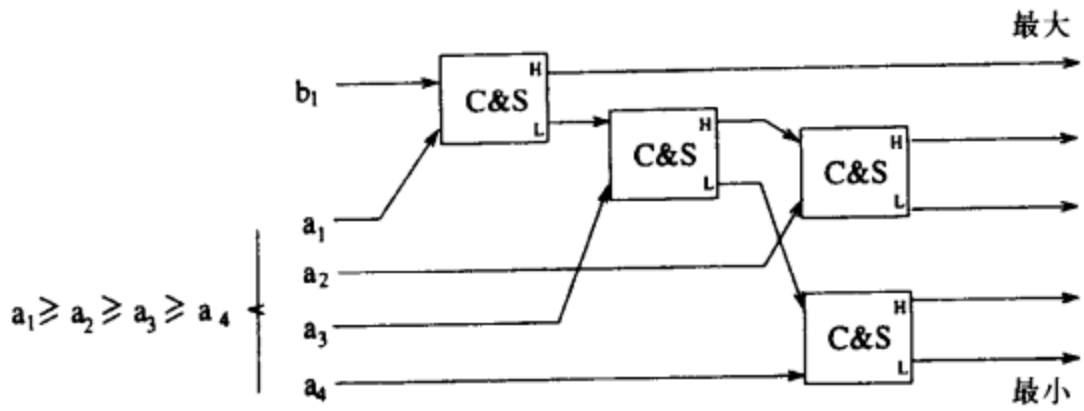


图9-23 4×1合并单元

例9.4.4 考虑窗口尺寸W=7的秩-阶滤波器。其结构如图9-24所示。该滤波器需要一个4×3合并单元，这个合并单元由8个**C&S**单元组成，如图9-25所示。可以看出，总共需要16个**C&S**单元。 ■

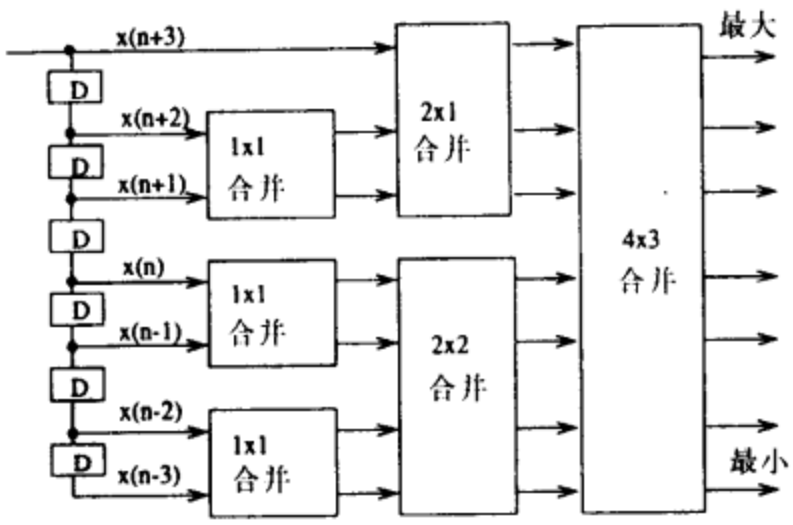


图9-24 W=7秩-阶滤波器

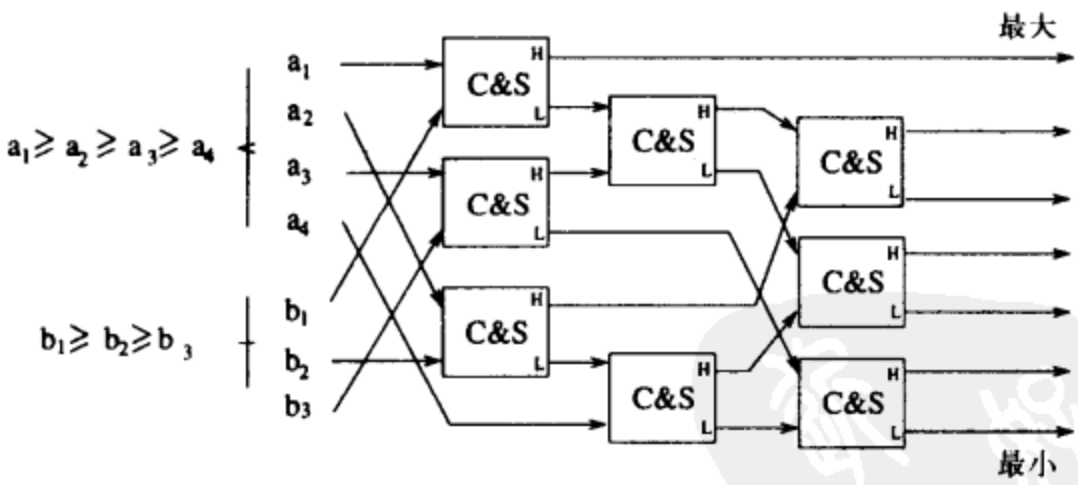


图9-25 4×3合并单元

9.4.3 并行秩-阶滤波器

可以利用块结构内部的共享比较运算来设计有效率的并行秩-阶滤波器，获得的并行滤波器可以充分地降低硬件复杂度[18]。本小节用例子说明了这些采用共享技术的子结构，用于低复杂度并行秩-阶滤波器的设计。系统设计方法可参考文献[18]。

例9.4.5 考虑一个窗尺寸为5并进行2级并行处理（例如，L=2）的并行秩-阶滤波器。并行

秩-阶滤波器的函数表达式如下：

$$\begin{aligned} y_r(2k) &= r^{th}\text{秩}\ [x(2k-2), x(2k-1), x(2k), x(2k+1), x(2k+2)] \\ y_r(2k+1) &= r^{th}\text{秩}\ [x(2k-1), x(2k), x(2k+1), x(2k+2), x(2k+3)] \end{aligned} \tag{9-75}$$

可以通过重复两次相同的硬件来直接实现该滤波器，如图9-26所示，它需要$9\times2=18$个**C&S**单元。但是由于此结构包含了许多冗余的比较操作，因此它并不是一种有效的结构。为了去除这些冗余的比较操作，将公式（9-75）中的序号降低$2k$，并重新写成如图9-27所示的形式。从图9-27中可以看到，在该块结构中，两个相同的硬件块被用于对元素-1、0、1和2的分类。采用这种冗余可以获得有效的结构，如图9-28所示。与图9-26中所需的18个**C&S**单元相比，图9-28所示的结构仅需要13个**C&S**单元。 ■

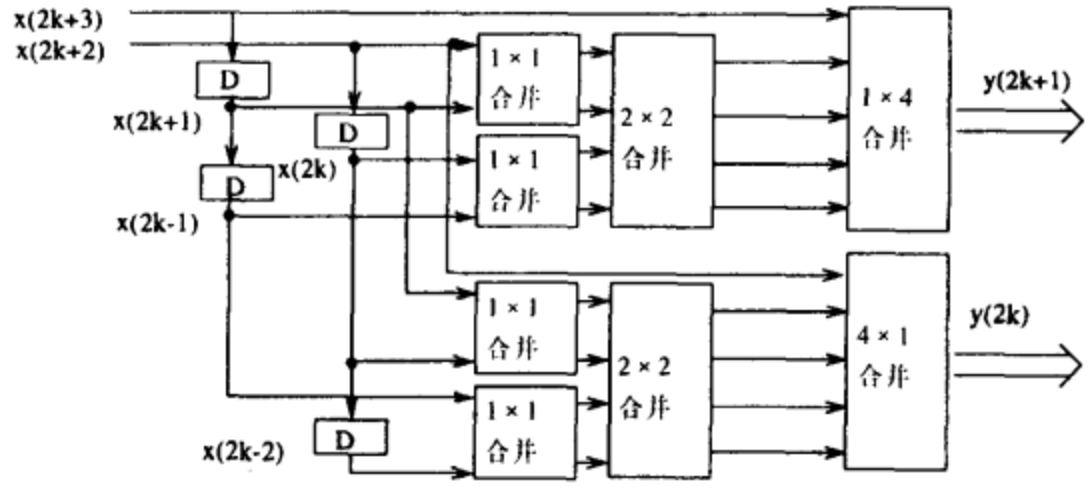


图9-26 W=5、具有两级并行处理的秩-阶滤波器结构

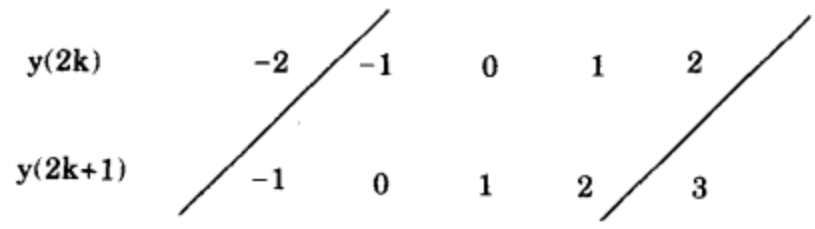


图9-27 W=5、L=2秩-阶滤波器序号

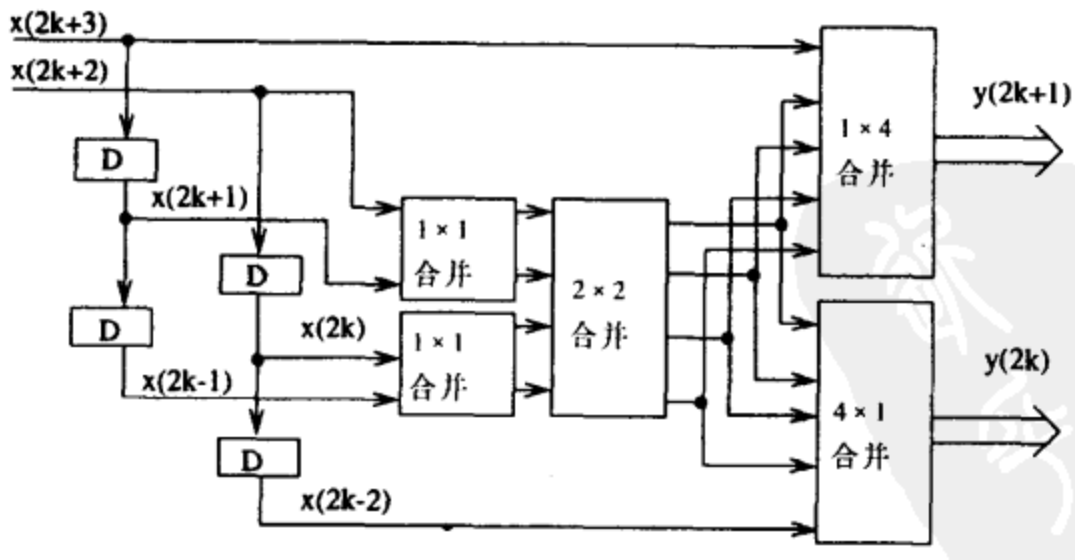


图9-28 具有共享子结构的 W=5、L=2 秩-阶滤波器结构

例9.4.6 考虑一个窗尺寸W=5且块尺寸L=3的并行秩-阶滤波器。在3并行系统中的分类操作可以用图9-29所示的框图来表示(序号$3k$已经被减去)。将输入序列分组如图9-29所示。同组

中的比较操作可以用一个C&S单元来实现。相应的并行滤波器结构如图9-30所示。与标准实现所需的27个C&S单元相比，在这种情况下只需21个C&S单元。每次分组两个采样，就可以得到最有效的并行结构。在本例中，读者可以自己证明，如果将 $y(3k)$、$y(3k+1)$和$y(3k+2)$分为一组，那么也将需要21个C&S单元（见习题38）。 ■

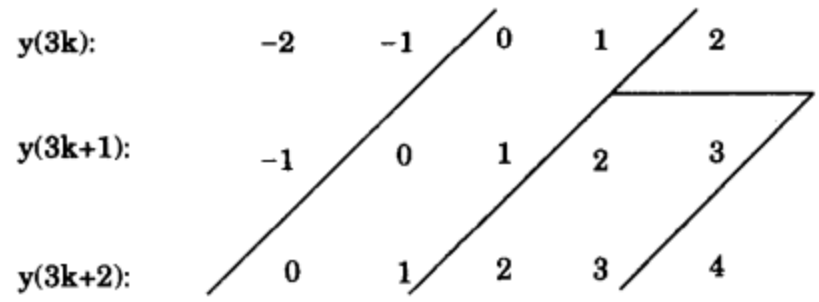


图9-29 W=5、L=3 秩-阶滤波器序号

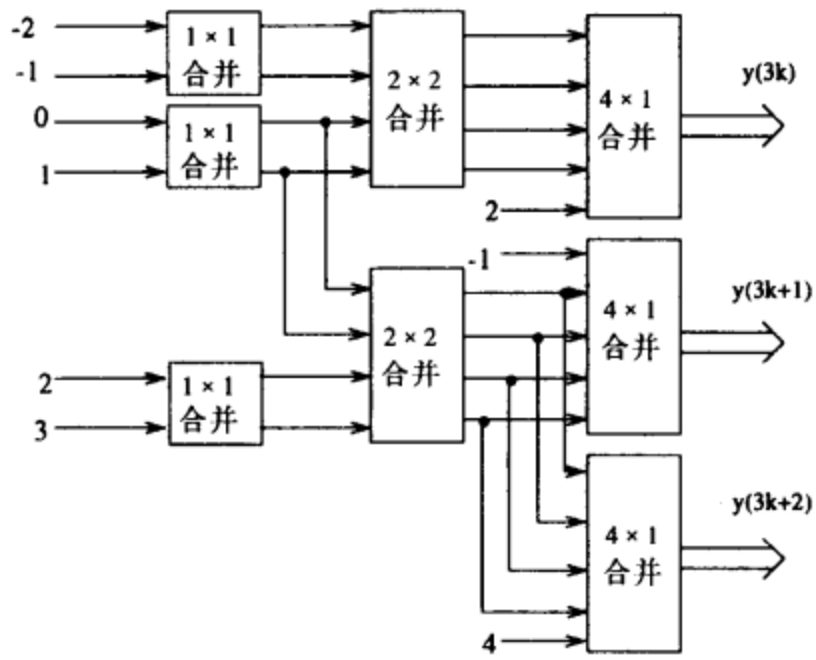


图9-30 具有共享子结构的 W=5、L=3 秩-阶滤波器结构

289~292

从例9.4.5和例9.4.6可以看出，设计有效率的并行秩-阶滤波器的基本思路包括寻找多个输出共同的输入对，接着寻找输入对中的输入对，以此类推。这样做的目的是尽可能多地使用共享子结构。例9.4.7展示了一个包含更多搜索过程的并行秩-阶滤波器的设计。

例9.4.7 考虑设计窗尺寸为7的4并行秩-阶滤波器。图9-31展示了搜索过程。将公共元素进行分组，并以分层的方式对分类过程进行重复，该重复过程是通过替换多个元素作一组来实现的。这种分类方法被用来设计图9-32中的并行结构，与直接实现所需的64个C&S单元相比，它只需要43个C&S单元。 ■

9.4.4 运行次序合并分类器-时间映射技术

到目前为止，Batcher的奇偶合并分类器已经用于秩-阶滤波。从公式（9-74）可以看到：采样窗口中的输入是连续的时间采样；当输入窗口随时间滑动时，只有一个输出离开窗口和一个输入进入窗口。因此，利用输入之间的时间关系，就可以进一步减少合并单元的总数量。运行次序合并分类器采用了这一原理，输入被相同数量的时间步长分离开来的那些单元被映射为一个单一的合并单元，对于合并的每一层次，该映射过程是通过使用附加存储器来实现的[19]。

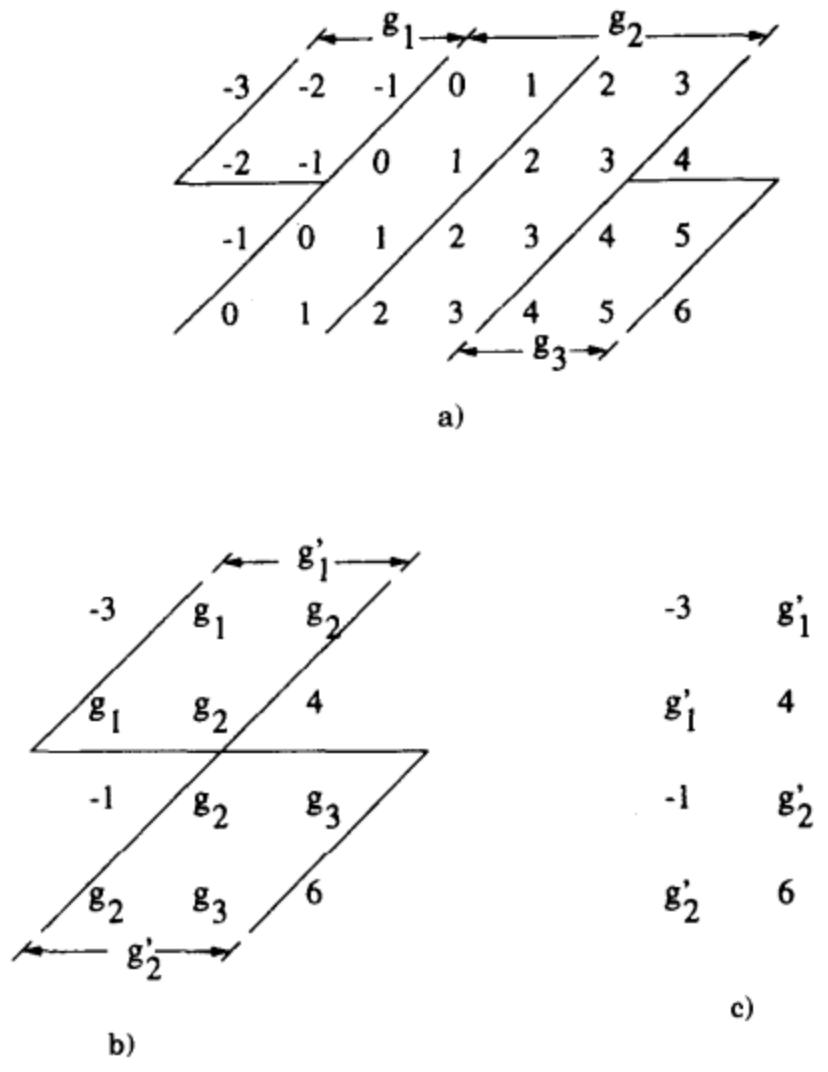


图9-31 对于 W=7、L=4 的秩-阶滤波器的共享比较操作的搜索过程

293

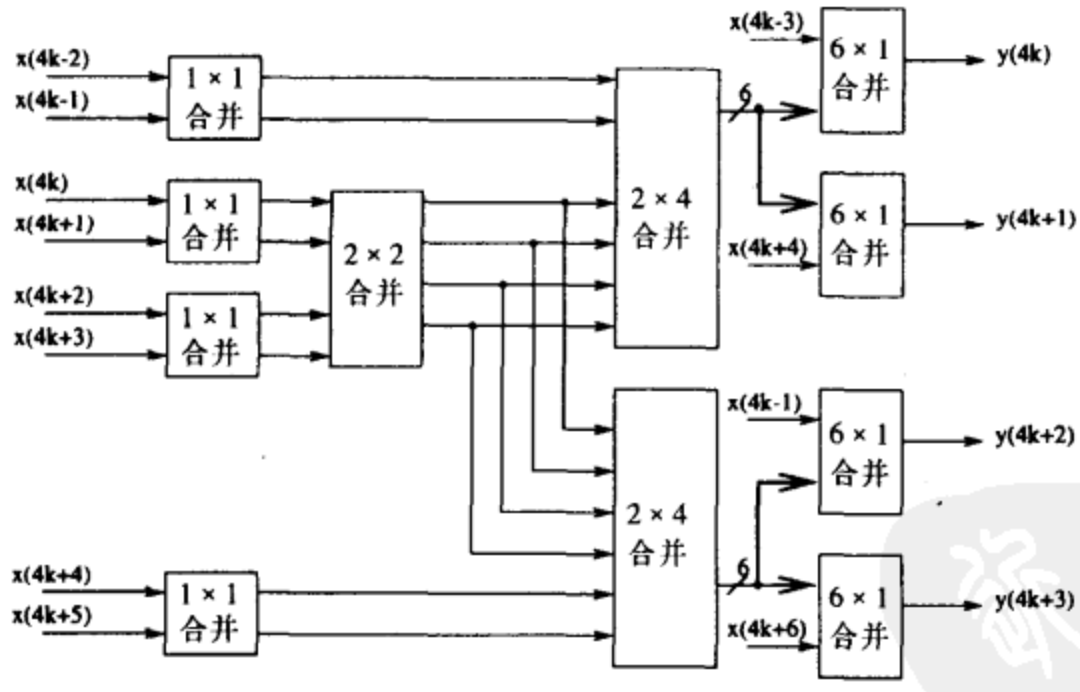


图9-32 具有共享子结构的 W=7、L=4 秩-阶滤波器

然而，由于每个合并层产生一个串行分类输出流，该分类器的延迟等于窗口尺寸W。一种时间映射技术可以并行产生所有的分类输出，如下面的例子所示。

例9.4.8 考虑图9-33所示的窗尺寸W=8的秩-阶滤波器。此滤波器用Batcher的奇偶合并分类器算法来直接实现，需要4个1×1合并器、2个2×2合并器和1个4×4合并器，也即19个C&S单元。经过时间映射后的秩-阶滤波器结构如图9-34所示。可以看出，通过使用时间映

射技术，节省了三个1×1合并器和一个2×2合并器，也即6个C&S单元。 ■

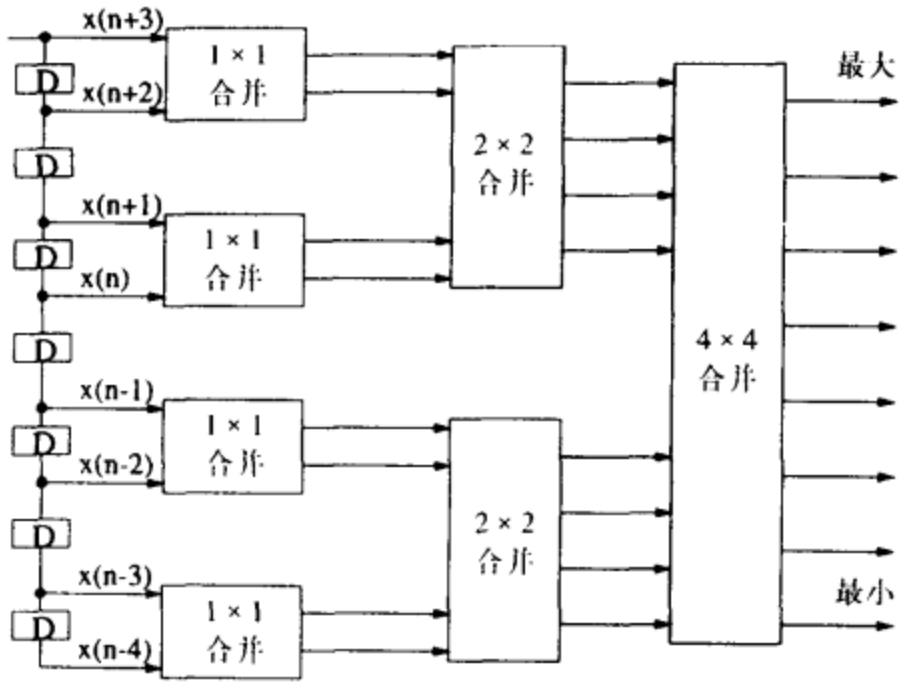


图9-33 W=8 秩-阶滤波器的块结构

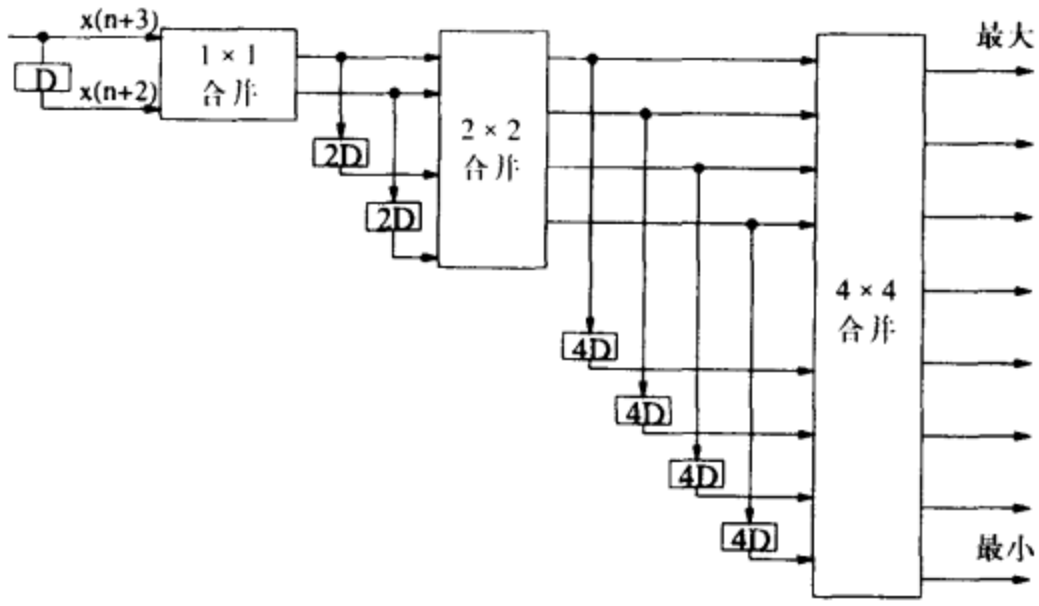


图9-34 经过时间映射后的 W=8 秩-阶滤波器的块结构

当窗口尺寸是2的幂时，该时间映射技术最有效。对于其他的窗口尺寸，可以通过同时仔细采用Batcher合并分类器和时间映射合并分类器来达到节省硬件开销的效果。

9.4.5 低功耗秩-阶滤波器

当维持与时序系统相同的采样频率时，可以通过降低电源电压来降低并行处理所需的功耗。然而，电源电压或功率缩减因子β有一个基本的底限，例如，电源电压βV_0不能低于CMOS器件阈值电压V_t。对于可靠的运算，通常电源电压维持在阈值电压的两倍。对于繁琐的块处理，相同数量的硬件被复制了L次。换句话说，并行系统的面积随着块尺寸L线性增加。因此，并行系统的功耗为 $P_{par} = \beta^2 P_{seq}$，这里的 P_{seq} 代表原始时序系统的功耗。这给功耗降低设置了更低的底限。然而，对于并行秩-阶滤波器，在采用共享子结构减少C&S单元的数量之后，并行滤波器面积（或总电容）的增加比块尺寸L线性增加要小，也即:

294 ~ 295

$$C_{total}^{par} = \alpha L C_{total}^{seq} \tag{9-76}$$

其中$\alpha<1$。因此并行秩-阶滤波器的功耗可以写成

$$P_{par} = \alpha\beta^2 P_{seq} \tag{9-77}$$

因此，通过增加块尺寸L，有可能继续降低功耗，超出电源电压限制的低功耗边界。

例9.4.9 考虑基于奇偶合并，窗尺寸$W=5$的秩-阶滤波器，如图9-22所示。**C&S**单元的总数为9。假设滤波器在**C&S**单元层次上进行流水，一个**C&S**单元的总电容为C_0。那么该时序滤波器的功耗就为

$$P_{seq} = 9C_0V_0^2 f_{seq} \tag{9-78}$$

现在考虑没有对冗余进行最小化的并行滤波器。其总功耗为：

$$P_{par} = \frac{9LC_0\beta^2V_0^2 f_{seq}}{L} = \beta^2 P_{seq} \tag{9-79}$$

注意因为没有对冗余进行最小化，那么其总电容就为$9LC_0$。一旦电源电压或β达到了它的更低的底限，功耗就不能再降低了。

考虑具有共享子结构的并行秩-阶滤波器。它证明了对于L级的块处理，**C&S**单元的总数为

$$\begin{aligned} N_{cs} &= 6L+1, \quad \text{当 } L \text{ 为偶数时} \\ N_{cs} &= 6L+3, \quad \text{当 } L \text{ 为奇数时} \end{aligned} \tag{9-80}$$

（见习题40）。在这种情况下，功耗为（对于L位偶数情况），

$$P_{parallel} = (6+\frac{1}{L})C_0\beta^2V^2 f_{seq} \tag{9-81}$$

注意即使达到了 β 的更低的底限，当 L 增加时功耗还可以降低。当$L\to\infty$时，达到功耗更低的底限。 ■ 296

9.5 结论

本章介绍了算法强度缩减变换方法。这些变换利用了共享子结构，减少了更强运算的数量，但可能是以增加更弱运算的数量为代价。强度缩减已经被用于设计快速并行FIR滤波器、DCT以及并行秩-阶滤波器。在VLSI实现中，这些架构可以减少面积和功耗，或者在可编程DSP实现中，可以缩短迭代周期。强度缩减也可在数字水平上应用，以降低实现的复杂度（见第15章）。

9.6 习题

1. 按照后处理矩阵、对角矩阵和预处理矩阵来表示如下的2并行滤波算法：

$$\begin{aligned} Y_0 &= X_0H_0 + z^{-2}X_1H_1 \\ Y_1 &= X_0H_0 + X_1H_1 - (X_0-X_1)(H_0-H_1) \end{aligned}$$

试用该表达式的转置求出另一个2并行结构。

2. 考虑如下具有单位采样相应的6抽头FIR滤波器，

$$h(n) = \{a, 0, b, b, c, c\} \tag{9-82}$$

和

$$H(z) = a + bz^{-2} + bz^{-3} + cz^{-4} + cz^{-5} \tag{9-83}$$

该序列滤波器有3个不同的系数，可以用3个乘法运算实现，如图9-35所示。该2并行滤波器将不超过6次乘法运算。设计实现该2并行滤波器的结构，并且给出所求滤波器结构中所有乘法器的系数。

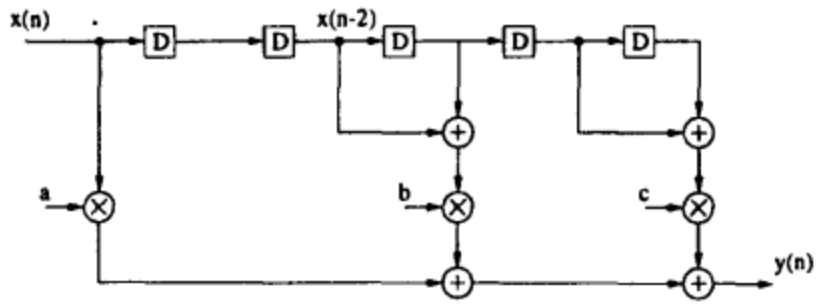


图9-35 习题2的结构图

3. 用9.2.2.1小节中3并行滤波器算法的转置设计一个3并行滤波器结构。
4. 用下面的3并行线性卷积:

$$\begin{bmatrix} s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ -1 & 2 & -2 & -1 & 2 \\ -2 & 1 & 3 & 0 & -1 \\ 1 & -1 & -1 & 1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} diag \begin{bmatrix} h_2/2 \\ (h_2+h_1+h_0)/2 \\ (h_2-h_1+h_0)/6 \\ (h_2+2h_1+4h_0)/6 \\ h_0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_0 \end{bmatrix}$$

求取一个需要5次子滤波运算的3并行滤波器。

5. 用下面的快速2 × 2卷积

$$\begin{aligned} h_0x_0 &= h_0x_0 \\ h_0x_1 + h_1x_0 &= \tfrac{h_0+h_1}{2}(x_0+x_1) - \tfrac{h_0-h_1}{2}(x_0-x_1) \\ h_1x_1 &= -h_0x_0 + \tfrac{h_0+h_1}{2}(x_0+x_1) + \tfrac{h_0-h_1}{2}(x_0-x_1) \end{aligned}$$

和转置运算来设计一个2并行FIR滤波器。

6. 转置习题5中求得的2并行滤波器结构，获得另外一个2并行滤波器。
7. 推导公式(9-35)和公式(9-36)。
8. 证明: 基于2并行快速滤波器算法，块尺寸为 2^n 的块滤波器可以用 $(1.5)^n N$ 次乘法和 $(1.5)^n N^n + 3^{n+1} - 2^{n+2}$ 次加法来实现，其中N表示滤波器的长度。
9. 考虑2并行滤波算法:

$$Y_0 = X_0H_0 + z^{-2}X_1H_1$$
$$Y_1 = X_0H_0 + X_1H_1 - (X_0 - X_1)(H_0 - H_1)$$

基于该算法，设计一个4并行滤波器。

10. 用尺寸为12的块实现36抽头的FIR滤波器。一个12并行FIR滤波器可以用两次2并行快速算法和一次3并行快速算法来产生。如果使用下面的快速算法顺序，求所需的加

297 ~ 298

法和乘法的总次数:

(a) 2 × 2 × 3;

(b) 3 × 2 × 2;

(c) 2 × 3 × 2;

对于每一种情况，求取该滤波器所需的加法器的数目和预处理/后处理所需要的加法器的数目。

11. 考虑利用一个可编程DSP(PDSP)，一个60抽头FIR滤波器的12并行实现。该PDSP可以在一个时钟周期内实现一次加法、一次乘法或一次乘法累加。该12并行滤波器的组织结构为2 × 2 × 3。用这个PDSP，需要多少个时钟周期实现这个12并行滤波器？不包括读取和存储指令所需要的时钟周期。

12. 对于一个用块尺寸为12和36的结构实现的72抽头FIR滤波器，求取获得最优块结构的快速算法顺序，以使得加法和乘法运算次数的总和最小。

13. 用下面的块尺寸和快速算法顺序（假设滤波器的长度为N），计算实现一个FIR滤波器所需要的加法和乘法的次数:

(a) 块尺寸: 9，顺序: 3 × 3;

(b) 块尺寸: 36，顺序: 2 × 2 × 3 × 3;

(c) 块尺寸: 10，顺序: 2 × 5;

(d) 块尺寸: 27，顺序: 3 × 3 × 3。

14. 在同样的采用速率情况下，并行处理可用于减小FIR滤波器的功耗。假设用块尺寸为4的结构实现20抽头（N=20）快速FIR滤波器可以减小滤波器的功耗。假设使用广播FIR滤波器结构。计算功耗减小因子。假设 $V_0 = 5\text{V}$，$V_t = 0.4\text{V}$。

15. 本习题考虑用可编程DSP实现一个100抽头的FIR滤波器。该可编程DSP可以在一个时钟周期内实现一次加法、一次乘法或者一次乘法累加运算。设该可编程DSP的时钟周期为T纳秒。该滤波器可以用直接方式实现，满足实时应用中100T纳秒采样周期的约束，但是，在使用快速并行滤波器算法减小时钟周期数，同时低电源电压运行可编程DSP以成比例地提高时钟周期的情况下，存在一个低功耗实现。

299

(a) 该滤波器用图9-6所示的4并行滤波器结构实现。用该4并行滤波器结构，需要多少个时钟周期计算出一个输出采样值？当运行可编程DSP于低电源电压的情况下，计算功耗降低的百分比。你的实现应该仍然获得一个100T纳秒的采样周期。假设可编程DSP开始时使用的电源电压为5V，并且假设工艺的阈值电压为0.5V。

(b) 用图9-9中所示的8并行结构，重复(a)。

16. 考虑图9-36中所示的离散时间系统的实现，其中 $H(z)$ 是一个N抽头FIR滤波器。该系统的每个输入采样周期需要N次乘法运算和 $(N-1)$ 次加法运算。但是，因为奇数次输出不会被抽取器保留，于是这些输出不必被计算。那么，每个输入采样周期的计算需求就可以减小为 $\frac{N}{2}$ 次乘法和 $\frac{N-1}{2}$ 次加法运算。通过使用并行滤波器结构，计算需求可以更进一步减小。

(a) 通过使用多相表示，推导一个实现该系统的结构，每个输入采样周期仅仅需要计算偶数次输出，需要 $\frac{N}{2}$ 次乘法和 $\frac{N-1}{2}$ 次加法运算。

(b) 用2并行滤波器结构，简化(a)中结构为快速结构，使得需要更少数量的乘法运算。

证明：在该结构中，每个输入采样周期需要 $\frac{3N}{8}$ 次乘法和 $\frac{3N}{8}+1$ 次加法运算。

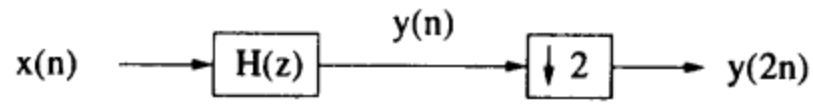


图9-36 习题16的结构图

17. 考虑图9-37中小波滤波器的一阶实现。假设 $H(z)$ 和 $G(z)$ 是N抽头滤波器。在直接实现中，仅仅计算偶数次输出 $v(2n)$ 和 $\omega(2n)$，每个输入采样周期需要N次乘法运算，$(N-1)$ 次加法运算。

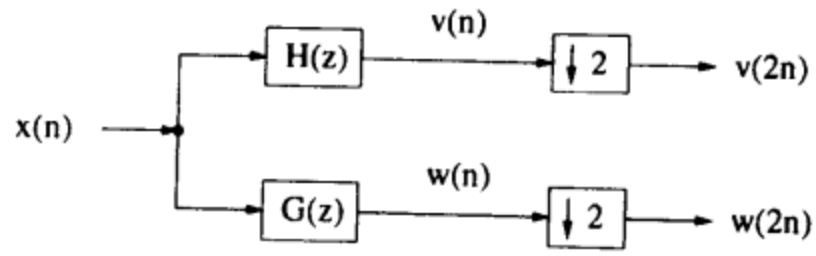


图9-37 习题17中小波滤波器的一阶结构

(a) 用公式（9-18）中的2并行快速FIR滤波器实现该滤波器。证明获得的快速结构每个输入采样周期需要 $\frac{3N}{4}$ 次乘法运算，$\frac{3N}{4}+1.5$ 次加法运算。

300

(b) 证明：用(a)中2并行滤波器的转置

$$\begin{aligned} Y_0 &= (X_1 z^{-2} - X_0)H_1 + (H_0 + H_1)X_0 \\ Y_1 &= (X_1 - X_0)H_0 + (H_0 + H_1)X_0 \end{aligned} \tag{9-84}$$

可以减少每个输入采样周期的加法运算次数为 $\frac{3N}{4}+1$。

18. 一个用于复数二维(2D)自适应滤波器的交叉耦合均衡器描述如下：

$$y(n) = \mathbf{w}^H(n-1)\mathbf{x}(n)$$

其中

$$\begin{aligned} \mathbf{x}(n) &= \mathbf{x}_r(n) + j\mathbf{x}_i(n) \\ \mathbf{w}^H(n) &= \mathbf{w}_r^T(n) - j\mathbf{w}_i^T(n) \\ \mathbf{w}_i^T(n) &= (w_{i1}(n), w_{i2}(n), \cdots, w_{iN}(n)) \end{aligned}$$

$\mathbf{w}^H(n)$ 表示 $w(n)$ 的厄密共轭或转置共轭。类似于 $\mathbf{w}_i(n)$，定义 $\mathbf{w}_r(n)$、$\mathbf{x}_r(n)$ 和 $\mathbf{x}_i(n)$ 为长度N的矢量。在均衡器中的权重通过最小均方值算法来更新：

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mu e^*(n)x(n)$$

其中，$e^*(n)$ 表示 $e(n)$ 的共轭。

(a) 滤波器输出的实部和虚部

$$\begin{aligned} y_r(n) &= \mathbf{w}_r^T(n-1)\mathbf{x}_r(n) + \mathbf{w}_i^T(n-1)\mathbf{x}_i(n) \\ y_i(n) &= \mathbf{w}_r^T(n-1)\mathbf{x}_i(n) - \mathbf{w}_i^T(n-1)\mathbf{x}_r(n) \end{aligned}$$

可以用4个滤波器来计算。采用快速FIR算法，用3个滤波器 $\mathbf{w}_r^T(n-1)+\mathbf{w}_j^T(n-1)$、$\mathbf{w}_r^T(n-1)-\mathbf{w}_i^T(n-1)$ 和 $\mathbf{w}_i^T(n-1)$ 可以获得另外一种实现。

(b) 对于 $\mathbf{w}_r(n)$ 和 $\mathbf{w}_i(n)$，推导权重更新方程。在(a)的计算中，最好是通过更新 $\mathbf{w}_r(n)+\mathbf{w}_i(n)$ 和 $\mathbf{w}_r(n)-\mathbf{w}_i(n)$ 来实现，而不是更新 $\mathbf{w}_r(n)$ 和 $\mathbf{w}_i(n)$。在 $\mathbf{w}_r(n)\pm\mathbf{w}_i(n)$ 更新以后，$\mathbf{w}_i(n)$ 可以通过取上述已更新系数之间差的一半来获得。对于 $\mathbf{w}_r(n)+\mathbf{w}_i(n)$ 和 $\mathbf{w}_r(n)-\mathbf{w}_i(n)$，用4个滤波器表示更新方程。基于快速FIR算法，用3个滤波器表示更新方程。 301

19. 考虑在三维(3D)信号环境中，基于快速滤波器技术、用更少的乘法运算数实现的一个自适应均衡器。设信号 $\mathbf{X}$ 和系数 $\mathbf{W}$ 的3D滤波器运算可以定义为:

$$\begin{aligned}\mathbf{W}^T\mathbf{X} &= (\mathbf{i}\mathbf{w}_i^T+\mathbf{j}\mathbf{w}_j^T+\mathbf{k}\mathbf{w}_k^T)(\mathbf{i}\mathbf{x}_i+\mathbf{j}\mathbf{x}_j+\mathbf{k}\mathbf{x}_k)\\ &= \mathbf{i}(\mathbf{w}_i^T\mathbf{x}_i+\mathbf{w}_k^T\mathbf{x}_j-\mathbf{w}_j^T\mathbf{x}_k)+\mathbf{j}(\mathbf{w}_j^T\mathbf{x}_j+\mathbf{w}_i^T\mathbf{x}_k-\mathbf{w}_k^T\mathbf{x}_i)\\ &\quad+\mathbf{k}(\mathbf{w}_k^T\mathbf{x}_k+\mathbf{w}_j^T\mathbf{x}_i-\mathbf{w}_i^T\mathbf{x}_j)\end{aligned}$$

其中，$\mathbf{w}_r$ 和 $\mathbf{x}_r$ 表示尺寸为N的列矢量。该均衡器需要9个滤波器和6个矢量加法。用4个滤波器和6个矢量加法实现该滤波器。(提示：用滤波器 $\mathbf{w}_i+\mathbf{w}_j+\mathbf{w}_k$，$\mathbf{w}_i+\mathbf{w}_j$，$\mathbf{w}_i+\mathbf{w}_k$ 和 $\mathbf{w}_j+\mathbf{w}_k$。)

20. 在FIR滤波器中，可用具有较小动态范围的一系列变换系数来减小硬件复杂度[20][21]。考虑如下的N抽头FIR滤波器

$$y(n)=\sum_{i=0}^{N-1}h_i x(n-i) \tag{9-85}$$

(a) 对于一个窄带低通滤波器，系数变化很慢。结果，前后连续的系数之间差的动态范围小于原来系数之间差的动态范围。也就是说，系数 $(h_{i+1}-h_i)$ 与 $\{h_i\}$ 相比，需要更少的非零位。给原有传输函数的分子和分母同时乘 $(1-z^{-1})$ 可以获得具有差分系数的传输函数。因此，

$$\begin{aligned}H(z) &= \frac{(\sum_{i=0}^{N-1}h_i z^{-i})(1-z^{-1})}{1-z^{-1}}\\ &= \frac{h_0+\sum_{i=0}^{N-2}(h_{i+1}-h_i)z^{-(i+1)}-h_{N-1}z^{-N}}{1-z^{-1}}\end{aligned}$$

可以用系数为 h_0、h_{N-1} 和 $(N-1)$ 个差分系数的FIR滤波器以及传输函数为 $1/(1-z^{-1})$ 的IIR滤波器来实现。对于$N=6$的情况，划出该滤波器的方框图。

(b) 对于窄带高通滤波器，前后连续的系数之间和的动态范围小于原来系数之间和的动态范围。用类似(a)的方法，基于前后连续的系数之和 $(h_{i+1}+h_i)$，可以获得该FIR滤波器的等效传输函数。用系数之和实现N抽头的FIR窄带高通滤波器。对于$N=6$，画出该滤波器的方框图。 302

21. 众所周知，窄带低通FIR数字滤波器的输入 $x(n)$ 变化较慢，如图9-38所示。在本习题中，我们探索几个用于低功耗实现的替代结构。

(a) 因为 $x(n)$ 变化缓慢，则 $x(n)-x(n-1)$ 的动态范围更小，可以用更小的字长来表示。因此，乘法器的面积可以减小。利用上述的观察结果，获得一个等效的滤波器结构。

(b) 在该部分中，假设用一个窄带低通滤波器给变化缓慢的信号滤波，利用字长的减小，求取一个较小面积和较低功耗的等效滤波器结构。

(c) 如果信号 $x(n)$ 变化很快，并用窄带高通滤波器来滤波，推导一个相应的等效结构。

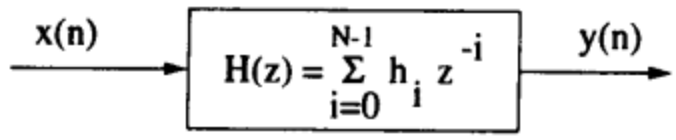


图 9-38

22. 考虑一个通过合理算法选择而设计的低功耗矢量量化器(VQ)(参见1.2.7节)。目的是选择一个需要更少运算的算法。考虑下面列出的全搜索、树搜索和差分树搜索的VQ算法。树搜索的VQ算法获得稍差但可以接受的性能。对于每一种VQ算法，计算乘法和加法/减法运算的数目。假设矢量的维数为16，电报密码本的尺寸为256。

(a) 全搜索矢量量化器，计算

$$\sum_{i=0}^{15}(x_i - c_{ji})^2,\ j = 0 \text{ 到 } 255 \tag{9-86}$$

(b) 树搜索矢量量化器（参见图1-8），计算

303

$$\sum_{i=0}^{15}(x_i - c_{ji})^2,\ j = 2 \times 8 \tag{9-87}$$

其中，对于树的8个层次中的任一层，计算了2个距离。

(c) 差分树搜索矢量量化器，对于树的每一层，计算

$$\begin{aligned} & \textstyle\sum_{i=0}^{15}(x_i - c_{ji})^2 - \sum_{i=0}^{15}(x_i - c_{ki})^2 \\ = \ & \textstyle\sum_{i=0}^{15} 2x_i(c_{ki} - c_{ji}) + \sum_{i=0}^{15}(c_{ji}^2 - c_{ki}^2) \end{aligned} \tag{9-88}$$

在树的每一层，矢量 C_j 和 C_k 表示与两个叶子字相关的电报密码本。假设 $\sum_{i=0}^{15} c_{ji}^2$ 事先已经被计算。

23. 考虑一个用快速并行滤波器技术的二维FIR滤波器的实现:

$$z(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x(i,j)y(i+m,j+n),\ 0 \leqslant m,n \leqslant M-1 \tag{9-89}$$

$z(m,\ n)$ 的直接实现中，每个输出采样需要 N^2 次乘法运算。通过使用快速并行滤波技术[12]，每个输出采样的乘法复杂度可以从 N^2 减小到 $9N^2/16$。

(a) 证明公式（8-89）中的计算可以转换为

$$\begin{bmatrix} z(2m,2n) \\ z(2m+1,2n) \\ z(2m,2n+1) \\ z(2m+1,2n+1) \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_{00} & \mathbf{Y}_{10} & \mathbf{Y}_{01} & \mathbf{Y}_{11} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \\ \mathbf{X}_{01} \\ \mathbf{X}_{11} \end{bmatrix}$$

其中，

$$\mathbf{Y}_{00}=\begin{bmatrix}\mathbf{Y}(2m,2n)\\ \mathbf{Y}(2m+1,2n)\\ \mathbf{Y}(2m,2n+1)\\ \mathbf{Y}(2m+1,2n+1)\end{bmatrix}\quad \mathbf{Y}_{10}=\begin{bmatrix}\mathbf{Y}(2m+1,2n)\\ \mathbf{Y}(2m+2,2n)\\ \mathbf{Y}(2m+1,2n+1)\\ \mathbf{Y}(2m+2,2n+1)\end{bmatrix}$$

$$\mathbf{Y}_{01}=\begin{bmatrix}\mathbf{Y}(2m,2n+1)\\ \mathbf{Y}(2m+1,2n+1)\\ \mathbf{Y}(2m,2n+2)\\ \mathbf{Y}(2m+1,2n+2)\end{bmatrix}\quad \mathbf{Y}_{11}=\begin{bmatrix}\mathbf{Y}(2m+1,2n+1)\\ \mathbf{Y}(2m+2,2n+1)\\ \mathbf{Y}(2m+1,2n+2)\\ \mathbf{Y}(2m+2,2n+2)\end{bmatrix}$$

$$\begin{aligned}\mathbf{X}_{ij}&=\begin{bmatrix}\mathbf{x}_{i,j} & \mathbf{x}_{i,j+2} & \cdots & \mathbf{x}_{i,j+N-2}\end{bmatrix}^T\\ \mathbf{Y}(i,j)&=\begin{bmatrix}\mathbf{y}(i,j) & \mathbf{y}(i,j+2) & \cdots & \mathbf{y}(i,j+N-2)\end{bmatrix}\end{aligned}$$

和

$$\begin{aligned}\mathbf{x}_{ij}&=\begin{bmatrix}x(i,j) & x(i,j+2) & \cdots & x(i,j+N-2)\end{bmatrix}^T\\ \mathbf{y}(i,j)&=\begin{bmatrix}y(i,j) & y(i,j+2) & \cdots & y(i,j+N-2)\end{bmatrix}\end{aligned}$$

304

矢量 $\mathbf{X}_{ij}$ 和 $\mathbf{Y}(i,j)$ 的长度为 $N^2/4$。该计算中的每个输出采样也需要 N^2 次乘法-累计运算。

(b) 验证下面的计算与(a)中二维滤波器运算是相同的:

$$\begin{aligned}&\begin{bmatrix}z(2m,2n)\\ z(2m+1,2n)\\ z(2m,2n+1)\\ z(2m+1,2n+1)\end{bmatrix}\\ &=\begin{bmatrix}(\mathbf{B}(2m,2n)+\mathbf{B}(2m,2n+1))\mathbf{X}_{00}\\ (\mathbf{B}(2m+1,2n)+\mathbf{B}(2m+1,2n+1))\mathbf{X}_{10}\\ (\mathbf{B}(2m,2n+1)+\mathbf{B}(2m,2n+2))\mathbf{X}_{01}\\ (\mathbf{B}(2m+1,2n+1)+\mathbf{B}(2m+1,2n+2))\mathbf{X}_{11}\end{bmatrix}\\ &+\begin{bmatrix}-1 & 0\\ 0 & -1\\ 1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{B}(2m,2n+1)(\mathbf{X}_{00}-\mathbf{X}_{01})\\ \mathbf{B}(2m+1,2n+1)(\mathbf{X}_{10}-\mathbf{X}_{11})\end{bmatrix}\\ &+\begin{bmatrix}-1 & 0\\ 1 & 0\\ 0 & -1\\ 0 & 1\end{bmatrix}\begin{bmatrix}(\mathbf{Y}(2m+1,2n)+\mathbf{Y}(2m+1,2n+1))\mathbf{A}_0\\ (\mathbf{Y}(2m+1,2n+1)+\mathbf{Y}(2m+1,2n+2))\mathbf{A}_1\end{bmatrix}\\ &+\begin{bmatrix}1\\ -1\\ -1\\ 1\end{bmatrix}\mathbf{Y}(2m+1,2n+1)(\mathbf{A}_0-\mathbf{A}_1)\end{aligned}$$

其中,

$$\begin{aligned}&\mathbf{A}_0=\mathbf{X}_{00}-\mathbf{X}_{10},\ \mathbf{A}_1=\mathbf{X}_{01}-\mathbf{X}_{11}\\ &\mathbf{B}(i,j)=\mathbf{Y}(i,j)+\mathbf{Y}(i+1,j)\end{aligned}$$

(c) 证明二维滤波器的计算复杂度可以从 N^2M^2 减小为

$$\begin{aligned}&\tfrac{9}{16}(N^2\cdot M^2)+[\tfrac{5}{4}N^2+2(N+M-2)(N+M-1)\\ &\quad+(N+M-2)^2]+\tfrac{10}{4}M^2\end{aligned}$$

其中，第一项$\left(\frac{9}{16}(N^2 M^2)\right)$是9个滤波运算所需要的运算次数，最后一项$\left(\frac{10}{4}M^2\right)$是后处理所需要的运算次数，中间项是滤波器系数预处理和滤波器输入所需要的运算次数。在这个习题中，一次运算可能是乘法、加法或者乘法-累加运算。

305

24. 本习题考虑基于快速滤波器技术，使用可编程DSP实现的运动估值[12]。用均方误差(MSE)标准来进行运动估值，而不是用平均绝对差（MAD）标准。MSE标准定义为

$$s(m,n)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(x(i,j)-y(i+m,j+n))^2 \quad 0\leqslant m,n\leqslant M-1 \tag{9-90}$$

为了减小计算复杂度，MSE表示为:

$$\begin{aligned}s(m,n)=&\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x(i,j)^2-2\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x(i,j)y(i+m,j+n)\\&+\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}y(i+m,j+n)^2,\ 0\leqslant m,n\leqslant M-1\end{aligned} \tag{9-91}$$

对于每个参考模块，公式(9-91)的第一项是常数。该项对于比较目的而言是不重要的，不必计算。第二项可以用习题23中的快速二维滤波方法实现。公式(9-91)中第三项可以通过迭代计算获得。假设一个运算可以是加法，乘法或者乘法-累加运算中的任何一个。

(a) 对于N=16和M=32，计算公式(9-90)所需的总的运算次数。

(b) 对于N=16和M=32，用习题23中(c)的结论计算二维滤波器快速实现所需要的运算次数。

(c) 证明公式(9-91)中的第三项

$$f(m,n)=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}y(i+m,j+n)^2,\ 0\leqslant m,n\leqslant M-1 \tag{9-92}$$

可以表达为

$$\begin{aligned}f'(m,n)=&\ \sum_{i=0}^{N-1}y(i+m,n)^2\\&\ m=0,\cdots,M-1;\ n=0,\cdots,N+M-2\\f(m,n)=&\ \sum_{j=0}^{N-1}f'(m,j+n)\\&\ 0\leqslant m,n\leqslant M-1\end{aligned} \tag{9-93}$$

证明公式(9-93)可以迭代计算为

306

$$\begin{aligned}f'(m,n)=&\ f'(m-1,n)+y_{m+N-1,n}^2-y_{m-1,n}^2\\f(m,n)=&\ f(m,n-1)+f'(m,n+N-1)-f'(m,n-1)\end{aligned} \tag{9-94}$$

共 $N+2(M-1)\ (N+M-1)\ +\ M(N+2(M-1))$ 次计算。对于N=16和M=32，计算需要的运算次数。

(d) 比较(a)中运算的次数与(b)和(c)中总的运算次数。

25. 用下面的步骤，从公式(9-46)的DCT推导出公式(9-47)的IDCT:

(a) 证明

$$\sum_{k=0}^{N-1}(e^{j\frac{ik\pi}{N}}+e^{-j\frac{ik\pi}{N}})=\begin{cases}2N & i=0\\ 0 & i\text{ 为不等于0的偶数}\\ 2 & i\text{ 为奇数}\end{cases}$$

(b) 用(a)证明

$$\sum_{k=0}^{N-1}\cos\frac{(2l+1)k\pi}{2N}\cos\frac{(2m+1)k\pi}{2N}=\frac{1}{2}+\frac{N}{2}\delta(l-m)$$

(提示：首先表示 $\cos\theta$ 为 $(e^{j\theta}+e^{-j\theta})/2$。)

(c) 用(b)从公式(9-46)推导出公式(9-47)。

26. 用公式(9-55)中的旋转定义，证明图9-13(c)中的变换。
27. 通过转置图9-14中的结构，给出一个8点的IDCT结构。
28. 用转置算法结构获得一个16点的DCT结构。
29. 基于DCT，用变换算法获得一个缩放结构的8点DCT，其中每个输出的缩放因子是 $1/c_4$。计算该缩放的DCT结构所需要的乘法次数。
30. 本习题考虑基于频率抽取方法，快速6点IDCT结构的设计。

(a) 为了计算

$$x(n)=\sum_{k=0}^{2}X(k)\cos\frac{(2n+1)k\pi}{6},\ n=0,1,2$$

首先获得一个3点IDCT蝶型结。

(b) 用(a)中的3点IDCT蝶型结。实现一个6点快速IDCT频率抽取结构。计算该结构所需要乘法和加法运算次数的总和。

307

31. 本习题考虑一个缩放的8点DCT实现，这里计算系数 $X(k)/c_4$，而不是 $X(k)$，用11次乘法运算，把图9-17中的结构变换为一个缩放的DCT结构。
32. 基于频率抽取快速算法，推导出一个16点的IDCT结构。计算所需的乘法和加法运算的次数。
33. 考虑用可编程DSP实现的8点IDCT，该可编程DSP可以在一个时钟周期内执行一次乘法、一次加法或者一次乘法-累加运算。

(a) 考虑用公式(9-53)实现的DCT，对于所有的 $X(k)$，除了 $X(4)$ 和 $X(0)$表示如下

$$\begin{bmatrix}X(0)\\X(4)\end{bmatrix}=\begin{bmatrix}c_4 & c_4\\ c_4 & -c_4\end{bmatrix}\begin{bmatrix}P_{10}\\P_{11}\end{bmatrix}$$

证明：该DCT用可编程DSP可以在35个时钟周期内完成。

(b) 基于图9-17中的快速流图，计算实现该DCT所需要的时钟周期数。

(c) 在(a)和(b)的实现中，哪种实现更好？为什么？

34. 画出3 × 5、4 × 5和2 × 6合并器的结构，这里，两个预先分类的序列被合并形成一个完整的分类序列。
35. 证明用奇偶合并器方法实现的 $2^p\times 2^p$ 合并器需要 $p2^p+1$ 个比较和互换元件，有一

个长度为 $(p+1)T_{CS}$ 关键路径，其中 T_{CS} 表示一个比较和互换单元的计算时间。

36. 画出窗口尺寸为7，块尺寸为2、4和8的并行可编程秩-阶滤波器的完整结构图。在该结构中，需要多少个比较和互换单元？在阈值电压为 $V_t = 0.6\text{V}$、电源电压为 $V_0 = 5\text{V}$ 的情况下，对于一个具有相同采样速率的串行结构，如果使用并行秩-阶滤波器，那么求出并行秩-阶滤波器所需功耗与串行结构所需功耗的百分比是多少？假设这里的串行和并行滤波器是流水线结构，其关键路径受限于一个比较和互换单元。
37. 对于块尺寸为3和5的情况，重复习题36的计算。
38. 一个窗口尺寸为5的3并行结构的秩-阶滤波器已经在例子9.4.6中给出。考虑把 $y(3k)$、$y(3k+1)$ 和 $y(3k+2)$ 当作一组，设计另外一个3并行结构的秩-阶滤波器。画出用于该结构中所有合并器的详细电路。

308

39. 对于一个窗口尺寸为9，块尺寸为3和4的并行秩-阶滤波器，计算所需的比较和互换单元的数目。
40. 证明：对于一个窗口尺寸为5，用子结构共享的L并行秩-阶滤波器，需要的比较和互换单元数为：$6L+1$，（对于偶数L）和$6L+3$（对于奇数L）。
41. 证明：一个窗口尺寸为7，块尺寸为L的并行秩-阶滤波器所需要的比较和互换单元的数目为

$$\begin{aligned} 10.25L+2, &\quad L=4k \\ 10.25L+5.75, &\quad L=4k+1 \\ 10.25L+3.5, &\quad L=4k+2 \\ 10.25L+4.25, &\quad L=4k+3 \end{aligned} \tag{9-95}$$

对于这个块尺寸为L的并行滤波器，求其渐进功耗减小因子是多少？

42. 考虑一个窗口尺寸为5的可编程秩-阶数字滤波器的串行实现。设该系统的采样周期为T。假设该系统运行的电源电压为5V，与工艺相关的阈值电压为0.75V。我们希望设计一个采样周期为$T/2$（例如具有2倍更高的速度）、同时可以降低功耗的实现。到此，假设可利用的电源电压可从1.5V变化到5V。考虑采用2级流水线，块尺寸为L的并行处理来实现该系统。假设L的可能取值为2、4、6、8、10等。记住流水线-并行系统在电源电压小于1.5V时不能运行，而且一定要获得$T/2$的采样周期。求出L如何取值可以获得最小的功耗？如何取值可以获得最小的面积功耗积？
43. 设计一个窗口尺寸为W=7、块尺寸为L=6的并行秩-阶滤波器。计算所需的**C&S**单元数。
44. 设计一个窗口尺寸为W=5、具有最小**C&S**单元需求的秩-阶滤波器结构。（提示：用Batcher的合并-分类器和运行次序合并-分类器。）
45. 设计一个窗口尺寸为W=7、具有最小**C&S**单元需求的秩-阶滤波器结构。用运行次序分类器和子结构共享技术设计块尺寸为L=3和L=5的块秩-阶滤波器。

309

参考文献

1. P. P. Vaidyanathan, *Multirate Systems and Filter Banks.* Prentice Hall, 1993.
2. S. Winograd, "Arithmetic complexity of computations," in *CBMS-NSF Regional Conference Series in Applied Mathematics*, no. 33, SIAM Publications, 1980.

3. R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, 1985.

4. Z.-J. Mou and P. Duhamel, "Short-length FIR filters and their use in fast non-recursive filtering," *IEEE Trans. on Signal Processing*, vol. 39, pp. 1322–1332, June 1991.

5. Z.-J. Mou and P. Duhamel, "Fast FIR filtering: algorithms and implementations," *Signal Processing*, vol. 13, no. 4, pp. 377–384, Dec. 1987.

6. D. A. Parker and K. K. Parhi, "Low-area/power parallel fir digital filter implementations," *Journal of VLSI Signal Processing*, no. 1/2, pp. 75–92, Sept. 1997.

7. J.-G. Chung, Y.-B. Kim, H.-G. Jeong, K. K. Parhi, and Z. Wang, "Efficient parallel FIR filter implementations using frequency spectrum characteristics," in *IEEE International Symposium on Circuits and Systems*, (Monterey, CA), vol. 5, pp. 354–358, June 1998.

8. J. I. Acha, "Computational structures for fast implementation of L-path and L-block digital filters," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 6, pp. 805–812, June 1989.

9. J. Granata, M. Conner, and R. Tolimieri, "The tensor product: a mathematical programming language for FFTs and other fast DSP operations," *IEEE Signal Processing Magazine*, pp. 40–48, Jan. 1992.

10. O. Rioul and P. Duhamel, "Fast algorithms for discrete and continuous wavelet transforms," *IEEE Trans. on Information Theory*, no. 3, pp. 569–586, March 1992.

11. N. Shanbhag and M. Goel, "Low-power adaptive filter architectures and their application to 51.84 mb/s atm-lan," *IEEE Trans. on Signal Processing*, no. 5, pp. 1276–1290, May 1997.

12. Y. Naito, T. Miyazaki, and I. Kuroda, "A fast full-search motion estimation method for programmable processors with a multiply-accumulator," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Atalanta, GA), pp. 3221–3224, 1996.

13. N. Demassieux and F. Jutand, "Orthogonal transforms," in *VLSI Implementations for Image Communications* (P. Pirsch, ed.), pp. 217–250, Elsevier, 1993.

14. C. Loeffler, A. Lightenberg, and G. S. Moschytz, "Algorithm-architecture mapping for custom DSP chips," in *IEEE International Symposium on Circuits and Systems*, pp. 1953–1956, 1988.

15. W. Chen, C. Smith, and S. C. Fralick, "A fast computation algorithm for the discrete cosine transform," *IEEE Trans. on Communication*, pp. 1004–1009, Sept. 1977.

16. B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, pp. 1243–1245, Dec. 1984.

17. K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Comput. Conference*, pp. 307–314, 1968.

18. L. E. Lucke and K. K. Parhi, "Parallel processing architectures for rank order

and stack filters," *IEEE Trans. on Signal Processing*, no. 5, pp. 1178–1189, May 1994.

19. I. Pitas, "Fast algorithms for running ordering and max/min calculation," *IEEE Trans. on Circuits and Systems*, no. 6, June 1989.

20. N. Sankarayya, K. Roy, and D. Bhattacharya, "Algorithms for low power and high speed FIR filter realization using differential coefficients," *IEEE Trans. on Circuits and Systems–II*, vol. 44, no. 6, pp. 488–497, June 1997.

21. S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "Decorrelating (DECOR) transformations for low-power adaptive filters," in *Proc. of 1998 IEEE Symposium on Low-Power Electronics*, (Monterey, CA), pp. 250–255, Aug. 1998.

311

第10章 流水线结构的并行自适应递归滤波器

10.1 引言

任何数字滤波器的频响特性都可以用有限冲击响应（Finite Impulse Response， FIR）或无限冲击响应（Infinite Impulse Response，IIR）数字滤波器来实现。相对来说，在实现某个特定的频响特性时，采用IIR滤波器往往比采用FIR滤波器阶数更低，因此在选择数字滤波器时，通常会首选IIR滤波器。不过，虽然FIR和IIR滤波器对于某个固定环境的频率修整十分有效，但是对于时变系统的滤波效果却不太好。自适应数字滤波器正是为这类时变系统设计的，它可以应用在诸如噪声消除、回波消除、波形生成、系统验证等方面。自适应数字滤波器的系数会随着每次迭代而不断地自行调整，直至最终收敛到某一个固定的值。这也正是自适应滤波器与固定系数的FIR和IIR滤波器的最大区别。

非递归运算通常采用流水线或者并行处理架构，而自适应递归数字滤波器由于本身的反馈回路，无法简单地套用这类架构。本章的论述重点在于如何采用超前计算（look-ahead computation）、增量块处理（incremental block processing）技术、基于最小均方误差（Least-Mean-Square, LMS）流水计算的弛豫超前（relaxed look-ahead）变换以及格型自适应滤波的方法，在自适应递归数字滤波器中实现流水线和并行处理。

本章安排如下：10.2节介绍采用交织滤波操作的多通道IIR数字滤波器中的流水线结构和使用超前计算的单通道IIR数字滤波器中的流水线结构；10.3节介绍如何在使用超前和分解技术的一阶IIR数字滤波器中实现流水线；10.4节介绍如何在使用聚类超前变换和离散超前变换方法的高阶IIR数字滤波器中实现流水线；10.5节介绍如何在使用直接计算技术和增量计算技术的IIR数字滤波器中实现并行处理；10.6节介绍IIR数字滤波器的流水线和并行结构；10.7节介绍如何使用流水线和（或）并行处理技术以及降低供电电压来减少IIR数字滤波器的功耗；10.8节介绍如何在使用弛豫超前变换的自适应数字滤波器中实现流水线。

10.2 数字滤波器中的流水线交织操作

这一节中，以一个简单的一阶递归数字滤波器为例，回顾一下流水线交织操作的概念。将讨论三种形式的操作：

(1) 效率的单通道/多通道交织

(2) 效率的单通道交织

(3) 高效率的多通道交织

在（1）中，因为需要对采样得到的零序列进行交织操作以保持算法的完整性，所以在没有改变算法结构的情况下，在环路中实现了流水线操作，但是这样一来硬件就没有得到充分利用。在（2）和（3）中，对算法的内部结构进行了修改，以便能最大限度地对流水线加以充分利用。

10.2.1 低效率的单通道/多通道交织

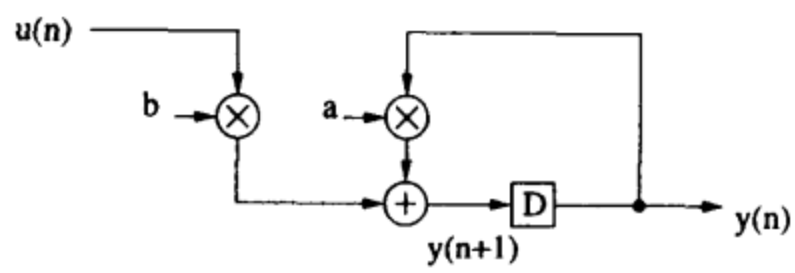
考虑一个一阶线性时不变递归表达式

$$y(n+1) = ay(n) + bu(n) \tag{10-1}$$

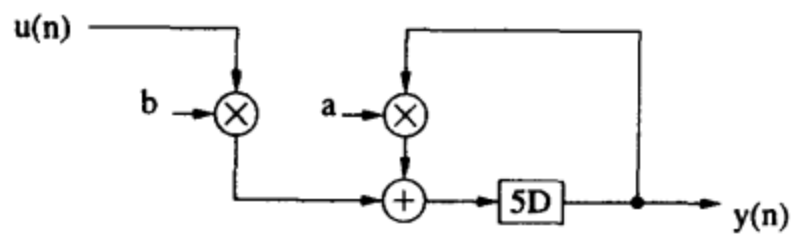
图10-1a是它的信号流图。由这个信号流图得出的迭代周期是 $(T_m + T_a)$，其中，T_m 和 T_a 分别代表字级（word-level）的乘法运算时间和加法运算时间。

考虑通过在环路中（适当的位置）插入（$M-1$）个额外的锁存器，把它用M级流水线实现。这种实现方式的时钟周期原则上可以缩短到原来的$1/M$，但是环路计算的延时和采样周期都会增加到M个时钟周期。举一个$M=5$的例子，如果在时钟周期0的时候从 $y^1(0)$ 状态开始，那么在时钟周期5的时候可以得到 $y^1(1)$。对单个的时间系列，这种情况每隔5个时钟出现一次，即只有总时间的20%是有用的（如果要在每个时钟周期都输入对单个时间序列的采样，就必须采用不同的算法，因为环路内部逻辑延时的个数会发生改变）。所以这种实现方式的采样率是时钟频率的1/5，并不比没有流水线的快（实际上由于额外插入的锁存器带来的延时，会变得更慢）。但是如果5个独立的时间序列都进行滤波的话，尽管这些时间序列都必须在上述的低速率下进行滤波，滤波器的硬件结构还是会得到充分的利用，如图10-1c所示。独立的时间序列可以是级联的每一个一阶或者二阶的级的输出（因为这些级可以被一个

314



a) 简单的一阶递归



b) 环路中插入了（$M-1$）个延时操作后的一阶线性时不变递归（$M=5$）。这种实现方式导致M路交织

时间 (n)	0	1	2	3	4	5	6	7	8	9	10
状态 x(n)	$y^1(0)$	$y^2(0)$	$y^3(0)$	$y^4(0)$	$y^5(0)$	$y^1(1)$	$y^2(1)$	$y^3(1)$	$y^4(1)$	$y^5(1)$	$y^1(2)$

c) b的处理进度的一部分。输入时间序列进行5路交织操作，即5个独立的时间序列被同时滤波。$y^i(n)$ 对应时间是 n 时的第 i 个时间序列

315

图 10-1

前馈割集分开），或者从需要相同滤波操作的独立输入通道输入。例如，对一个由级联的二阶IIR实现的10阶递归滤波器，五部分的输出是独立的，并且能在流水线中进行交织操作（当然，每一个都是在1/5的低速率下）。这样一来，流水线交织的方法就非常适用于要求并行处理的应用。

综合以上所述，如果只有一个延时单元的递归环路，通过插入（$M-1$）个额外的延时单元实现了M级流水线，那么输入数据必须进行M路交织操作，也就是说，（$M-1$）个零时间序列或独立时间序列要和给定的数据流进行交织操作（否则，算法的传输函数将发生改变，并且在单个时间序列可以进行滤波的采样率下，将得不到结果）。这种实现方式在[1]~[3]中也经常被称作M倍降速电路。如果没有提供M个独立的运算以便进行交织操作的话，这种慢速交织方式的硬件结构就没有得到充分利用（实际情况经常是这样）。

10.2.2 高效的单通道交织

除开对独立时间序列进行交织操作不谈，M倍降速电路的实现方式有两个缺点：(1)采样率是时钟频率的$1/M$，(2)没有充分利用处理单元。使用超前变换，[4]，[5]，可以克服上述两种缺点。在这种方法中，对给定的线性递归关系先进行几次迭代以便产生并行性。

考虑公式(10-1)中的一阶线性时不变递归关系。通过对递归关系的变换，可以把 $y(n+2)$ 表示成 $y(n)$ 的函数，得到

$$y(n+2) = a[ay(n) + bu(n)] + bu(n+1) \tag{10-2}$$

这种递归关系的一种实现方式见图10-2a。它的迭代上下限（见第2章）是 $2(T_m + T_a)/2$，和图10-1a是一样的，因为计算量和递归环路中的逻辑延时与图10-1a相比都变成了两倍的关系，所以并没有带来增益。但是，和公式（10-2）等价的另一个递归关系是

$$y(n+2) = a^2y(n) + abu(n) + bu(n+1) \tag{10-3}$$

316

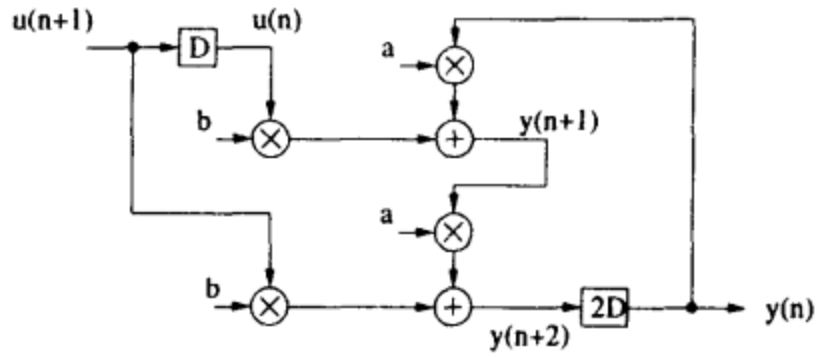
图10-2b描述了这种递归关系。它的迭代周期的上下限，$(T_m + T_a)/2$，是图10-1a中和图10-2a中实现方式的一半。

在公式(10-1)的迭代中应用（$M-1$）步超前计算，可以得到由公式(10-4)表示的等价实现方式，见图10-3a。

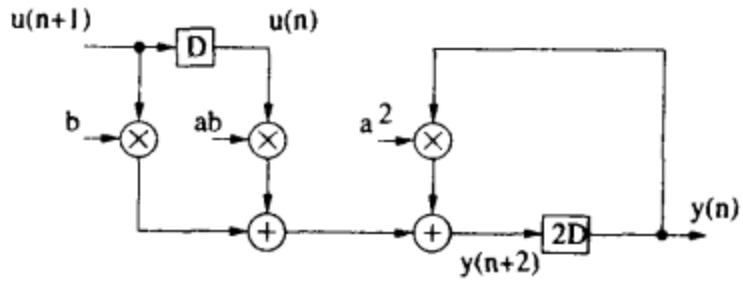
$$y(n+M) = a^My(n) + \sum_{i=0}^{M-1} a^ibu(n+M-1-i) \tag{10-4}$$

注意环路延时由 z^{-M} 引起，而不是 z^{-1}，这表明计算必须在M个而不是1个时钟周期内完成。它的迭代上下限是 $(T_m + T_a)/2$，与比原来的信号流图高M倍的采样率相对应（尽管复杂度和系统的延迟都线性增加了）。图10-3b是$M=5$时图10-3a的处理进度表的一部分。公式(10-4)中的 ab，a^2b，…，$a^{M-1}b$，a^M 各项可以预先计算出来，叫做预先计算项。

公式(10-4)等号右侧的第二项代表了超前计算项，它的复杂度叫做超前复杂度。因为超前计算项不是递归关系的，所以在适当的前馈割集位置上插入锁存器能够实现流水线操作。

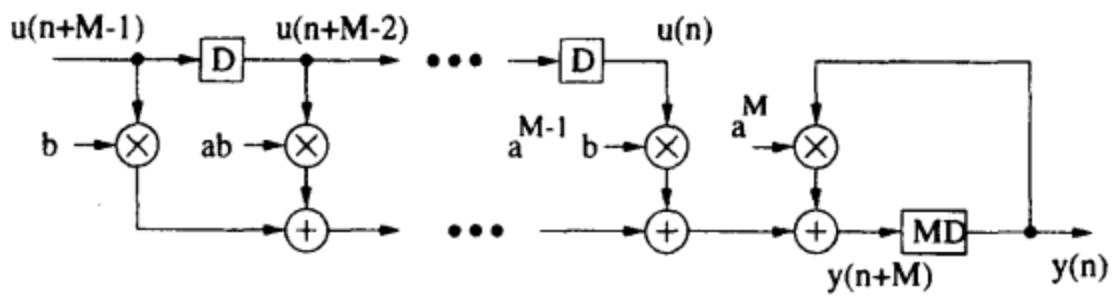


a) 没有应用超前变换的图10-1a的等价实现方法。迭代上下限和图10-1a相同



b) 应用了超前计算的等价的一阶线性时不变递归。迭代上下限是图10-2a结构的两倍

图 10-2



a) 采用$(M-1)$步超前计算得到的等价一阶线性时不变递归

时间 (n)	0	1	2	3	4	5	6	7	8	9	10
状态 $x(n)$	y(−4)	y(−3)	y(−2)	y(−1)	y(0)	y(1)	y(2)	y(3)	y(4)	y(5)	y(6)

b) $M=5$时图10-3a处理进度的一部分

图 10-3

稳态的输入输出特性并没有随着采用超前计算技术而改变。这意味着对大部分旧的输入而言，变换后系统的输出和原来的系统输出将是相同的。但是，只要原来的系统是因果的（在无限精确的前提下），就存在这样的可能性，即重新设置变换后系统的初始状态，可以使变换后系统的输入输出特性和原来系统的输入输出特性对所有的输入而言都是相同的。考虑与图10-3a相对应的图10-3b，以M个独立的初始状态 $y(-M+1)$，

$y(-M+2)$、…，$y(0)$ 开始（$M=5$）。在图10-1所示的原系统中，$y(1)$ 状态根据初始状态 $y(0)$ 来计算， 317~318

$$y(1)=ay(0)+bu(0) \tag{10-5}$$

对变换后的系统(10-4)来说，$y(1)$ 状态根据 $y(-M+1)$ 来计算，

$$y(1)=a^5y(-4)+bu(0) \tag{10-6}$$

$M=5$（因为根据因果性，$u(-4)$，…，$u(-1)$ 都等于0）。根据公式(10-5)和公式(10-6)，

$$y(-4)=a^{-4}y(0) \tag{10-7}$$

可以用相似的分析得到M个初始状态的值

$$y(-i)=a^{-i}y(0),\quad i=1,\ 2,\ \cdots,\ (M-1) \tag{10-8}$$

在变换后的系统中，从M个初始状态开始，后面的M个状态可以用流水线交织的方法计算出来（参看图10-3b）。在这一点上，超前计算可以看作是流水线交织的应用。超前计算允许把单一串行的计算变换成M个独立的并行计算，并且把反馈回路用流水线实现，在保持对硬件有充分利用率的同时达到了对单一时间序列的高速滤波。假设乘法器和加法器能方便地实现流水线，迭代的上下限可以通过时序变换或割集变换[1]~[3]（参看第4章）得到。

10.2.3 高效率的多通道交织

超前计算的概念可以扩展到多个独立通道都需要相同滤波操作的情况。考虑和公式(10-1)相同的一阶线性递归中有2通道与递归环路中有6级流水线的情况。没有使用超前计算技术时，在某一时刻只利用了三分之一的硬件。为了充分利用硬件，迭代两次递归，并对两个时间序列进行交织操作。一般地，如果有P个独立的时间序列，环路的流水线是M级（假设 $M=PQ$），那么递归需要被迭代 $(Q-1)$ 次。对这个例子来说，迭代后的递归由

$$y^i(n+3)=a^3y^i(n)+a^2bu^i(n)+abu^i(n+1)+bu^i(n+2),\quad i=1,\ 2 \tag{10-9}$$

表示。图10-4是采用交织的方法对两个独立时间序列的状态 y^1 和 y^2 进行处理的一部分。 319

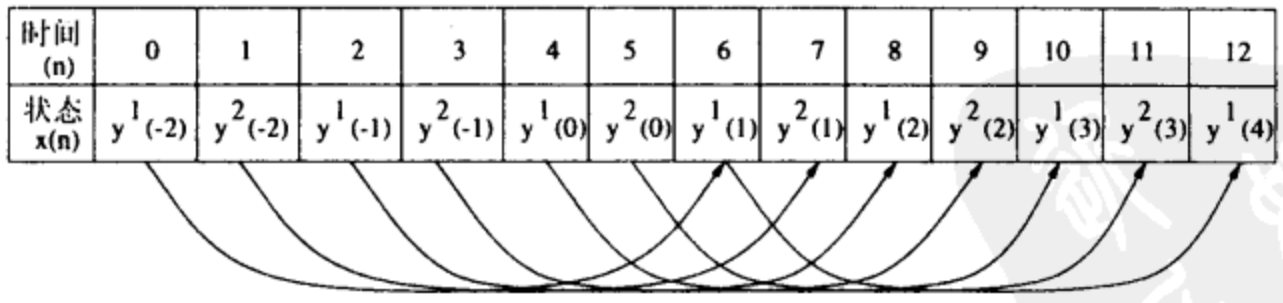


图10-4 采用两步超前计算得到的两通道6级流水线实现方式的处理进度

10.3 一阶IIR数字滤波器中的流水线实现

这一节介绍了使用超前计算技术在一阶递归数字滤波器中实现流水线的方法。在超前计算技术中，介绍了可以相消的极点和零点，它们离原点的距离和原极点相等，并且它们之间相距的角度也相等。流水线的实现要求环路流水线级数的复杂度线性增长，并且必须保证

当原滤波器稳定时，一阶IIR滤波器也是稳定的。这里用分解技术实现由于超前处理而产生的非递归部分，以此获得对于环路流水线级数而言硬件只是呈对数增长的实现方式。分解技术是得到面积优化的实现方式的关键，并使得流水线对于高速VLSI IIR滤波器的实现颇具吸引力。

考虑用公式(10-10)所示传输函数设计的一阶IIR滤波器。

$$H(z)=\frac{1}{1-az^{-1}} \tag{10-10}$$

输出采样 $y(n)$ 可以利用输入采样 $u(n)$ 和以前的输出采样计算出来，如下所示:

$$y(n)=ay(n-1)+u(n) \tag{10-11}$$

因为在关键环路中只有一个延时单元，所以这种递归滤波器的采样率受一个乘加运算所花费时间的限制。

10.3.1 一阶IIR滤波器的超前流水线结构

超前流水线的基本思想是给传输函数添加可以相消的极点和零点以使传输函数的分母中 z^{-1}，…，$z^{-(M-1)}$ 的系数是零。输出采样 $y(n)$ 可以利用输入和输出采样 $y(n-M)$ 来计算,这样关键环路中就会有M级延迟单元，它们反过来可以对关键环路进行流水线操作，滤波器的采样频率就会增加M倍。

320

例10.3.1 考虑一阶滤波器

$$H(z)=\frac{1}{1-az^{-1}} \tag{10-12}$$

它有一个极点位于 $z=a$，$a<1$。在 $z=ae^{\pm(j2\pi/3)}$ 添加极点和零点，就可以得到等价的3级流水线结构且稳定的滤波器，如下式所示

$$H(z)=\frac{1+az^{-1}+a^2z^{-2}}{1-a^3z^{-3}} \tag{10-13}$$

■

10.3.2 在2的幂次分解中实现超前流水线

在2的幂次分解中，一阶IIR滤波器的流水线可以通过 $\log_2 M$ 次变换得到，如例10.3.2所示。

例10.3.2 考虑一阶递归滤波器的传输函数

$$H(z)=\frac{bz^{-1}}{1-az^{-1}} \tag{10-14}$$

通过使用分解技术可以得到使用流水线后等价的传输函数

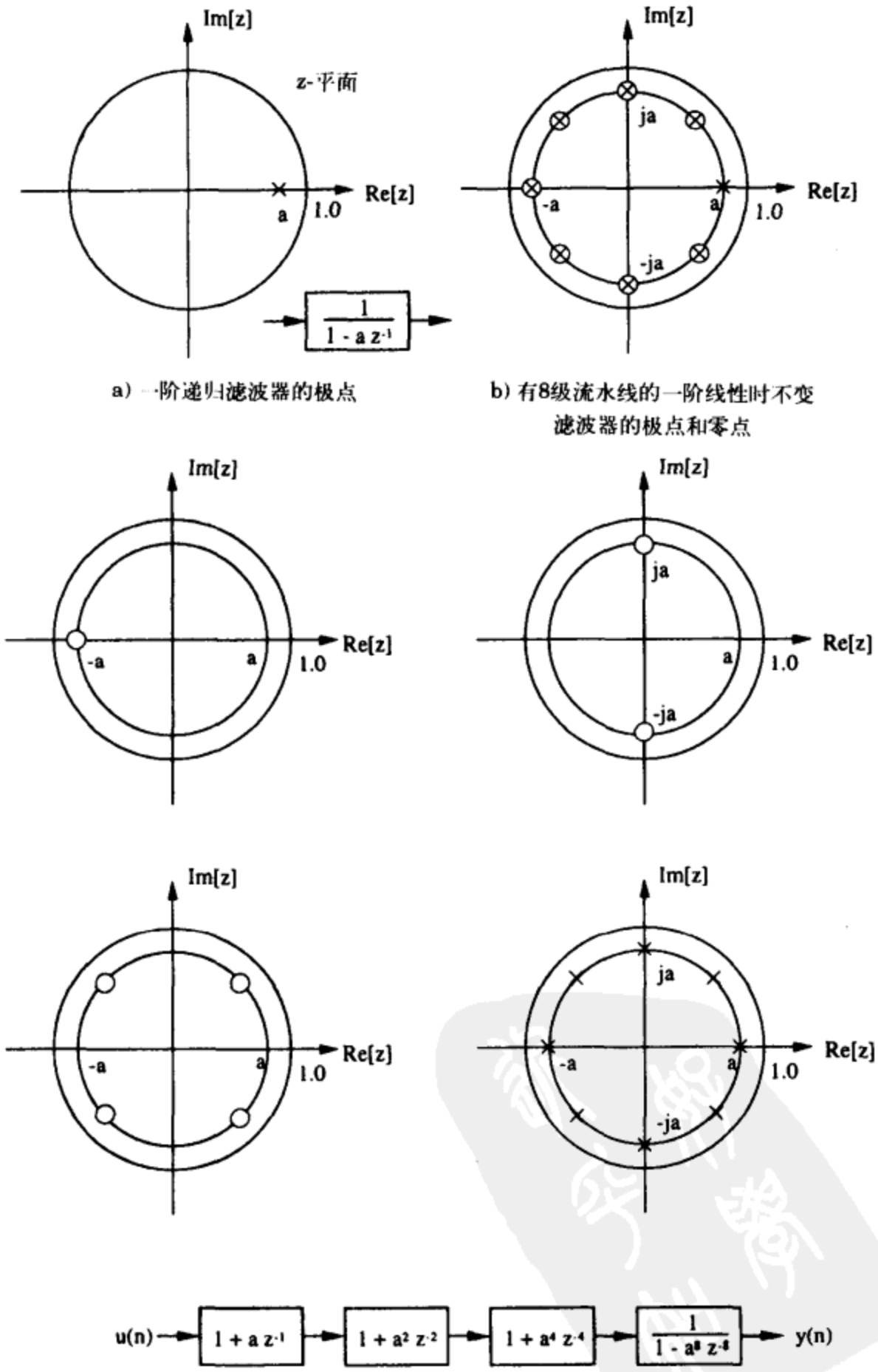
$$H(z)=\frac{bz^{-1}\prod_{i=0}^{\log_2 M-1}(1+a^{2^i}z^{-2^i})}{1-a^Mz^{-M}} \tag{10-15}$$

添加 $(M-1)$ 个极点和零点到相同的位置，可以实现这种流水线。原传输函数只在 $z=a$ 位置有一个极点（参看图10-5a）。而流水线结构的传输函数在a，$ae^{j2\pi/M}$，$ae^{j2(2\pi)/M}$，$ae^{j3(2\pi)/M}$，…，$ae^{j(M-1)(2\pi)/M}$ 位置都有极点（参看图10-5b，$M=8$）。图10-5c是对这些可以消掉的零点的

分解。分解后非递归部分的第 i 级在公式(10-16)位置实现 2^i 个零点

$$z = a\mathrm{e}^{j(2n+1)\pi/2^i},\ \ n = 0,\ 1,\ \cdots,\ (2^i - 1) \tag{10-16}$$

并且只需要一个流水线的乘法操作，而和级数 i 无关。这种流水线实现方式总的复杂度是 $(\log_2 M+2)$ 次乘法。 ■ 321



a) 一阶递归滤波器的极点

b) 有8级流水线的一阶线性时不变滤波器的极点和零点

c) 基于分解技术，在$M=8$的一阶线性时不变滤波器中实现流水线

图 10-5

322

注释

- 也可以用时域方法等价地解释基于分解技术的流水线的实现。原递归滤波器由公式(10-17)表示

$$y(n+1) = ay(n) + bu(n) \tag{10-17}$$

公式(10-18)是用流水线实现后的表达式

$$y(n+M) = a^M y(n) + \sum_{i=0}^{M-1} a^i bu(n+M-1-i) \tag{10-18}$$

以$M=8$为例，有

$$\begin{aligned} y(n+8) &= a^8 y(n) + \sum_{i=0}^{7} a^i bu(n+7-i) \\ &= a^8 y(n) + \sum_{i=0}^{7} a^i f_0(n+7-i) \\ &= a^8 y(n) + \sum_{i=0}^{3} a^{2i} f_1(n+7-2i) \\ &= a^8 y(n) + \sum_{i=0}^{1} a^{4i} f_2(n+7-4i) \end{aligned} \tag{10-19}$$

其中

$$\begin{aligned} f_0(n) &= bu(n) \\ f_1(n) &= af_0(n-1) + f_0(n) \\ f_2(n) &= a^2 f_1(n-2) + f_1(n) \end{aligned} \tag{10-20}$$

- 尽管流水线递归滤波器在无限精度的条件下是稳定的，在有限精度的情况下却对滤波器系数很敏感。在有限精度的实现方式中，M级流水线一阶IIR滤波器的极点在

$$p = (a^M + \Delta)^{1/M} \approx a(1 + \frac{\Delta}{Ma^M}) \tag{10-21}$$

其中，Δ是在表示 a^M 时的有限精度误差。当a的值比较小的时候，这种极点位置就变得更敏感（即极点离原点比较近时）。不过问题不大，因为极点离原点近而造成系统不稳定的情况并不严重。

- 除敏感性问题以外，不精确的零极点相消（参看图10-6）对有限精度的流水线滤波器也会造成影响，带来幅值和相位误差。通过增加字长可以减少这些误差。

10.3.3 在通用分解中实现超前流水线

我们讨论了使用2的幂次分解来用硬件高效实现一阶2的幂级流水线IIR滤波器。分解的思想可以扩展到任意的环路流水线级数M。如果 $M=M_1 M_2 \cdots M_p$，那么非递归部分分别实现(M_1-1)，$M_1(M_2-1)$，…，$M_1 M_2 \cdots M_{p-1}(M_p-1)$ 个零点，总共$(M-1)$个零点。

例10.3.3 考虑公式(10-12)中的一阶传输函数。把它用12级流水线分解实现，可以用下式表示

$$
\begin{aligned}
H(z) &= \frac{\sum_{i=0}^{11} a^i z^{-i}}{1-a^{12}z^{-12}} \\
&= \frac{(1+az^{-1})(1+a^2z^{-2}+a^4z^{-4})(1+a^6z^{-6})}{1-a^{12}z^{-12}}
\end{aligned}
\tag{10-22}
$$

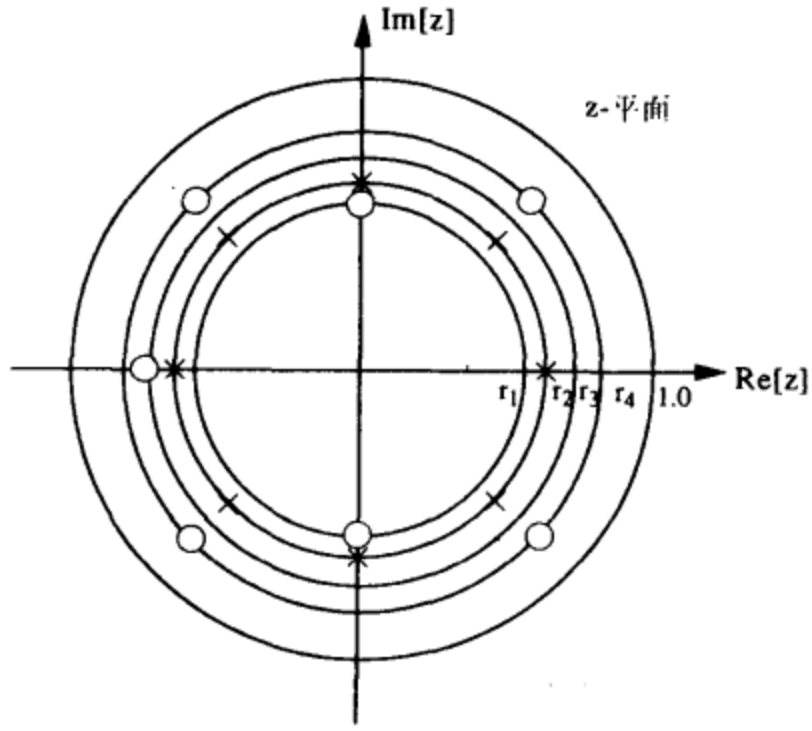


图10-6 在8级环路流水线的一阶IIR超前滤波器中，由于有限精度引起的不精确的零极点相消

323 ~ 324

公式(10-22)中的实现方式和2×3×2的分解相对应。它的零极点配置见图10-7a。对它的11个可以消掉的零点的分解如图10-7b所示，其中三部分分别实现1、4和6个零点。第一部分的零点在$-a$，第二部分的4个零点在 $ae^{\pm j\pi/3}$ 和 $ae^{\pm j2\pi/3}$，第三部分的6个零点在$\pm ja$，$ae^{\pm j\pi/6}$ 和 $ae^{\pm j5\pi/6}$。另外一种分解后的传输函数是

$$
H(z) = \frac{(1+az^{-1})(1+a^2z^{-2})(1+a^4z^{-4}+a^8z^{-8})}{1-a^{12}z^{-12}} \tag{10-23}
$$

它和2×2×3的分解相对应。在这种实现方式中，第一个非递归部分的一个零点在$-a$，第二部分的2个零点在$\pm ja$，第三部分的8个零点在 $ae^{\pm j\pi/6}$，$ae^{\pm j\pi/3}$，$ae^{\pm j2\pi/3}$ 和 $ae^{\pm j5\pi/6}$。3×2×2分解的传输函数是

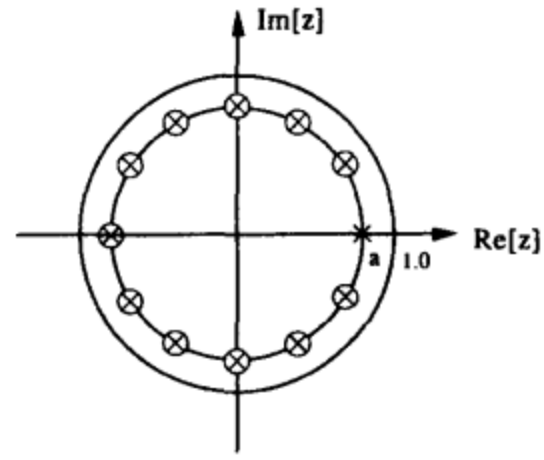
$$
H(z) = \frac{(1+az^{-1}+a^2z^{-2})(1+a^3z^{-3})(1+a^6z^{-6})}{1-a^{12}z^{-12}} \tag{10-24}
$$

三部分分别实现2、3和6个零点。第一部分在 $ae^{\pm j2\pi/3}$ 实现两个零点，第二部分在 $-a$ 和 $ae^{\pm j\pi/3}$ 实现3个零点、第三部分在 $ae^{\pm j\pi/6}$、$\pm ja$ 和 $ae^{\pm j5\pi/6}$ 实现6个零点。 ■

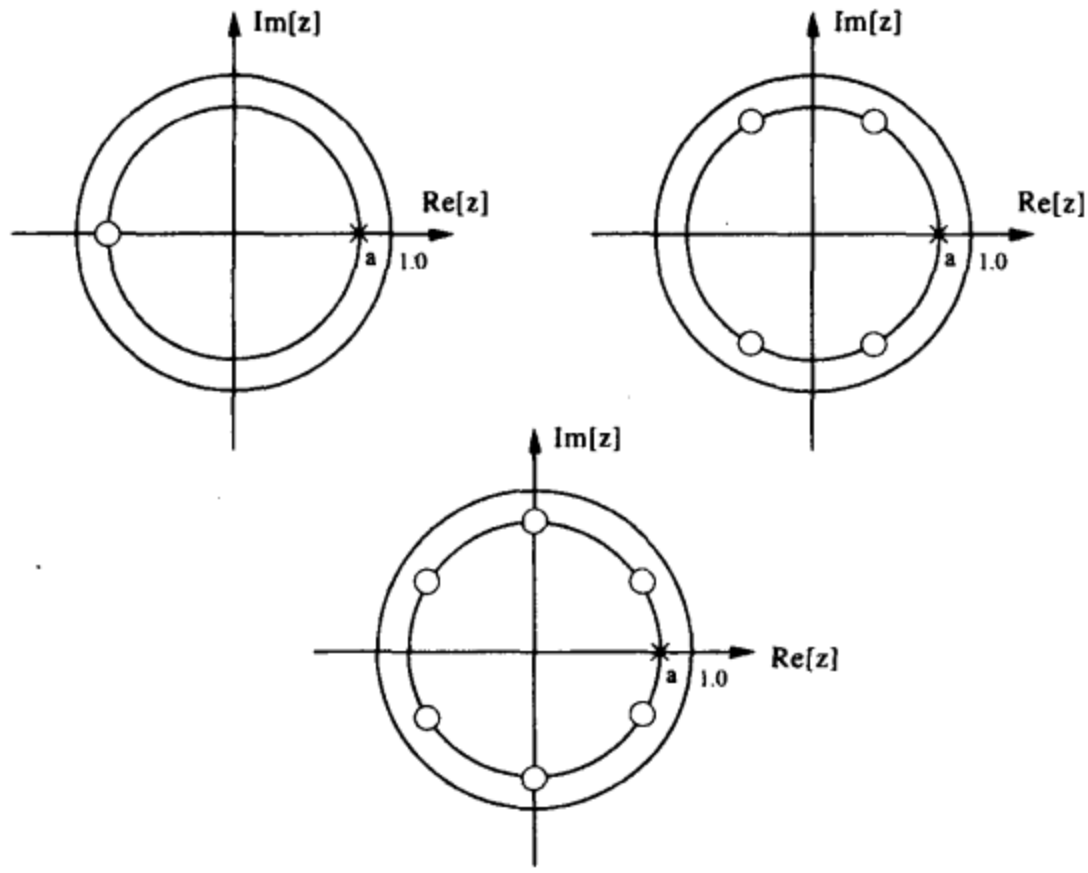
10.4 高阶IIR数字滤波器中的流水线实现

这一节介绍使用聚类和离散超前技术在高阶递归数字滤波器中实现流水线的方法（对一阶IIR滤波器来说，这两种方法变为同一种形式）。对聚类超前而言，流水线的实现要求环路流水线级数线性增长、并且不能保证它总是稳定的。我们将举例说明如何使用离散超前方法来获得稳定的流水线滤波器。然后在高阶IIR滤波器中再次使用分解技术，并使用这种技术来

得到面积优化的高阶滤波器。除了聚类超前和离散超前技术，还讨论了不会带来零极点相消，但是同样可以实现流水线的“受限的滤波器设计”。



a) 1阶12级流水线递归滤波器的极点、零点位置



b) 用2×3×2的分解方法分解流水线滤波器的零点

图 10-7

用公式(10-25)描述一个N阶直接形式递归滤波器的传输函数

$$H(z)=\frac{\sum_{i=0}^{N} b_i z^{-i}}{1-\sum_{i=1}^{N} a_i z^{-i}} \tag{10-25}$$

325 ~ 326

同样，输出采样 $y(n)$ 能被输入采样 $u(n)$、以前的输入和输出采样表示

$$\begin{aligned} y(n) &= \sum_{i=1}^{N} a_i y(n-i)+\sum_{i=0}^{N} b_i u(n-i) \\ &= \sum_{i=1}^{N} a_i y(n-i)+z(n) \end{aligned} \tag{10-26}$$

此外，因为关键环路中包含有一个单一的延时操作或锁存器，所以1个乘法操作和1个加法操作的吞吐量限制了这种递归滤波器实现方式的采样率。

10.4.1 聚类超前流水线

聚类超前流水线的基本思想是给滤波器的传输函数添加可以相消的极点和零点，以便使传输函数分母中 $z^{-1}, \cdots, z^{-(M-1)}$ 的系数是0。输出采样 $y(n)$ 可以用N个以前的输出 $y(n-M)$, $y(n-M-1)$, $\cdots$, $y(n-M-N+1)$ 的聚类来表示[6]。这样一来，这种实现方式的关键环路中就包含了M个延时单元和一个乘法操作。所以，这种环路可以有M级流水线，采样率可以提高M倍。这种实现方式叫做M级聚类超前流水线。

例10.4.1 考虑极点位于 $z=1/2$ 和 $z=3/4$（参看图10-8a）的全极点二阶IIR滤波器。原型滤波器由传输函数(10-27)描述

$$H(z)=\frac{1}{1-\frac{5}{4}z^{-1}+\frac{3}{8}z^{-2}} \tag{10-27}$$

通过消除分母中 z^{-1} 这一项，可以得到一个等价的两级流水线递归数字滤波器。为了达到这个目的，可以让分子分母同时乘以 $(1+5/4z^{-1})$，或等价地在 $z=-5/4$（参看图10-8b）引入一个极点和零点。变换后的传输函数是

$$H(z)=\frac{1+\frac{5}{4}z^{-1}}{1-\frac{19}{16}z^{-2}+\frac{15}{32}z^{-3}} \tag{10-28}$$

其中，分母中 z^{-1} 的系数是0。因此这个滤波器的关键环路包含两个延时单元并能实现两级流水线。类似地，通过分子分母同时乘以 $(1+5/4z^{-1}+19/16z^{-2})$，消除公式(10-27)分母中 z^{-1} 和 z^{-2} 两项，可以实现三级流水线。新的传输函数由下式表示 327

$$H(z)=\frac{1+\frac{5}{4}z^{-1}+\frac{19}{16}z^{-2}}{1-\frac{65}{64}z^{-3}+\frac{57}{128}z^{-4}} \tag{10-29}$$

■

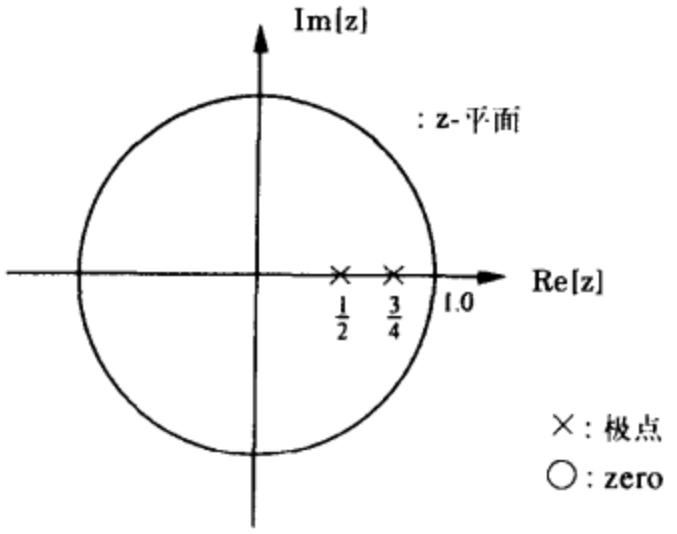
总的来说，可以通过在滤波器的分子、分母中同时乘以 $\sum_{i=0}^{M-1} r_i z^{-i}$，引入 $M-1$ 个附加的可以相消的零极点来实现M级聚类超前流水线递归滤波器。序列 r_i 的定义如下:

$$\begin{aligned} r_{-i} &= 0, \quad 当\ i=1,2,\cdots,N-1 \\ r_0 &= 1 \\ r_i &= \sum_{k=1}^{N} a_k r_{i-k},\ i>0 \end{aligned} \tag{10-30}$$

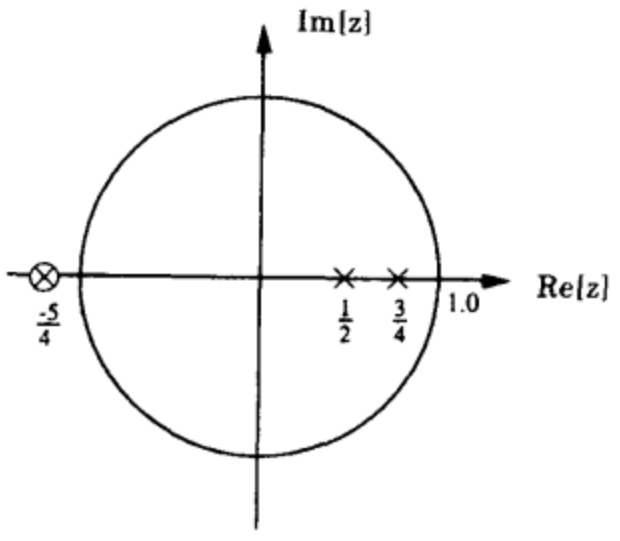
假设M级流水线滤波器的系数是预先计算好的，那么这个流水线滤波器的分子或非递归部分可以用 $(N+M)$ 个乘法运算实现，分母或递归部分可以用N个乘法运算实现。这样，这种流水线实现方式的总复杂度是 $(N+N+M)$ 个乘法运算，和环路流水线级数M、提高的速度或采样率的提高都是呈线性关系的。到目前为止，这种相消的零极点技术已经应用于流水线递归滤波器。但是，增加的极点可能位于单位圆之外，这将改变原滤波器的稳定性。

例10.4.2 公式(10-27)中的二阶滤波器的极点在 $z=0.5$ 和 $z=0.75$（参看图10-8a）。而两级流水线滤波器在 $z=-1.25$ 有一个附加的极点，这个极点位于单位圆外（参看图10-8b）。三级

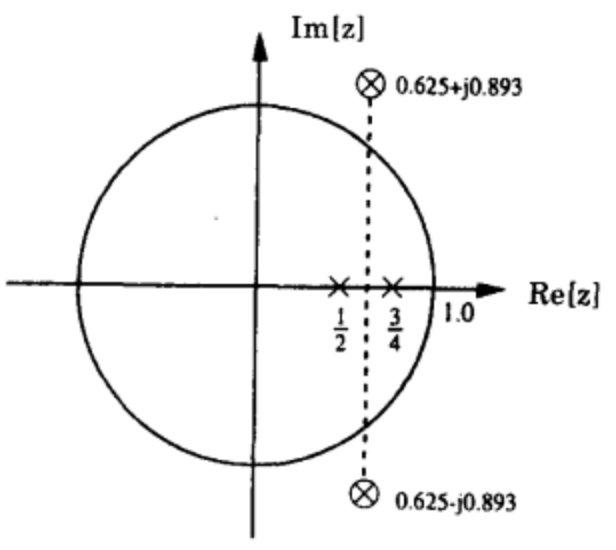
流水线滤波器在 $z=0.625\pm j0.893$ 有两个附加的极点（参看图10-8c）。注意这对复共轭极点也位于单位圆外。所以，尽管公式(10-27)的极点配置是稳定的，公式(10-28)和公式(10-29)中用等价的2级和3级流水线实现时是不稳定的。 ■



a) 2阶稳定的递归滤波器的零极点位置



b) 使用聚类超前方法得到等价的不稳定2阶流水线滤波器的零极点位置



c) 使用聚类超前方法得到等价的不稳定3阶流水线滤波器的零极点位置

图 10-8

10.4.2 稳定的聚类超前滤波器设计

考虑公式(10-31)给出的一个稳定的递归数字滤波器的传输函数

328 ~ 329

$$H(z)=\frac{N(z)}{D(z)} \tag{10-31}$$

聚类超前技术把 $H(z)$ 变换成

$$H(z)=\frac{N(z)P(z)}{D(z)P(z)}=\frac{N(z)P(z)}{1-z^{-M}Q(z)} \tag{10-32}$$

其中 $P(z)$ 代表为了产生所需要的流水线延时M而引入的多余的极点；$(1-z^{-M}Q(z))$ 是应用流水线后的分母。注意这个流水线滤波器的分母只包含 $z^{-(M+i)}$（除常数项外），其中i是小于或等于 $Q(z)$ 的阶数的任意非负整数。前面已经说明公式(10-32)在某一个临界的延时 M_c 时总是一

个稳定的滤波器，因此在 $M>M_c$ 时，是可以保证稳定性的。所以，如果流水线延时M不能产生一个稳定的滤波器，那么应该增大M的值，直到滤波器稳定为止。为了得到最佳的M值和$P(z)$，广泛使用了数值搜索的方法。

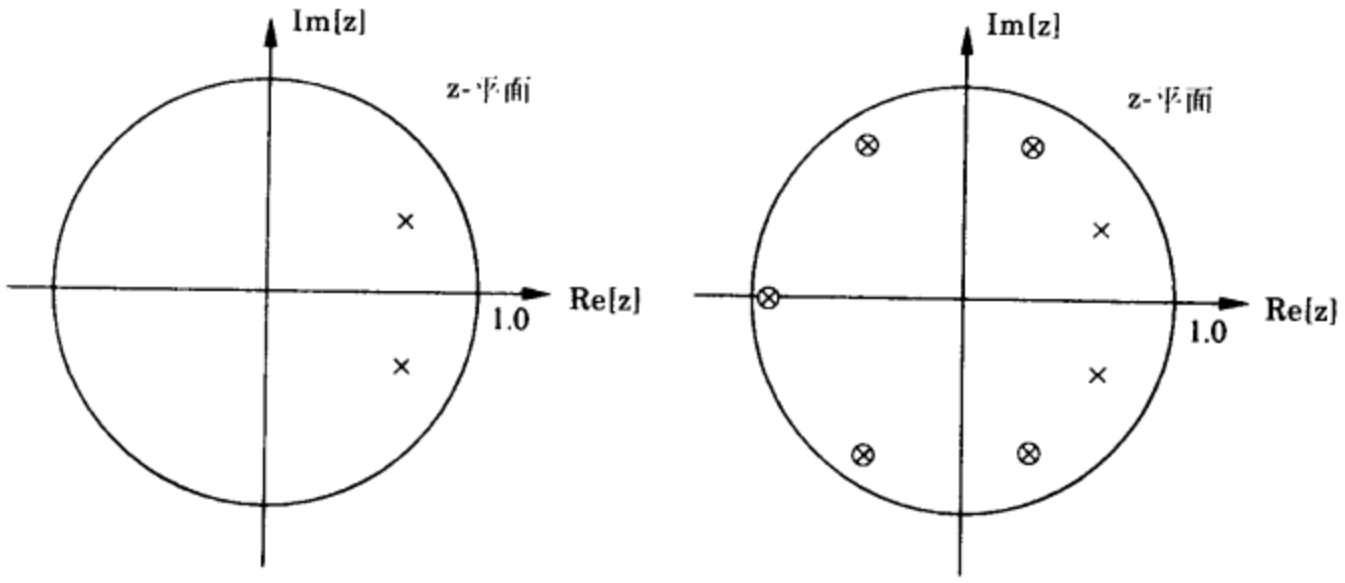
例10.4.3 考虑[7]中给出的用$M=5$实现流水线的二阶传输函数

$$H(z)=\frac{1}{1-1.5336z^{-1}+0.6889z^{-2}} \tag{10-33}$$

通过[7]中给出的稳定的聚类超前方法，我们知道$M=5$不满足稳定性条件。所以增大M使$M=6$，来得到下面稳定的流水线滤波器

$$H(z)=\frac{1+1.5336z^{-1}+1.6630z^{-2}+1.4939z^{-3}+1.1454z^{-4}+0.7275z^{-5}}{1-1.3265z^{-6}+0.5011z^{-7}} \tag{10-34}$$

图10-9a标明了原滤波器在 $z=0.7668\pm j\ 0.3177$ 位置的极点。在 $z=0.9718$，$z=-0.5572\pm j0.7765$ 和 $z=0.2763\pm j0.862$ 添加极点和零点可以得到流水线滤波器，极点-零点位置如图10-9b所示。 ■



a) 一个稳定的二阶递归滤波器的极点-零点位置图

b) 使用稳定的聚类超前方法得到了稳定的6级流水线滤波器的极点-零点位置图

图 10-9

当分母的乘法器数目太多的时候，因为不能用级联或并行的结构实现滤波器，流水线滤波器会有很大的舍入噪声。(注意如果要保持M的话，公式(10-32)就不能被分解成级联或并行的结构。)

10.4.3 离散超前流水线

在离散超前中，公式(10-25)的传输函数中的分母被变换成包含了N项 $z^{-M}, z^{-2M}, \cdots, z^{-NM}$ 的形式。同样，状态 $y(n)$ 由N个以前的离散状态 $y(n-M), y(n-2M), \cdots, y(n-NM)$ 计算得到。在这种超前方法中，对原滤波器中的每一个极点，引入 $(M-1)$ 个相消的极点和零点，这些极点和零点有相同的角度，并且它们到原点的距离和原极点相同。例如，如果原滤波器在 $z=p$ 有一个极点，我们通过在 $z=pe^{j2\pi k/M}$, $(k=1, 2, \cdots, M-1)$ 位置添加 $M-1$ 个极点和零点来实现M级环路流水线。假设传输函数的分母可以进行如下的因式分解

$$D(z)=\prod_{i=1}^{N}(1-p_i z^{-1}) \tag{10-35}$$

使用离散超前方法的流水线可以用下式表示

$$H(z)=\frac{N(z)}{D(z)}=\frac{N(z)\prod_{i=1}^{N}\prod_{k=1}^{M-1}(1-p_i e^{j2\pi k/M}z^{-1})}{\prod_{i=1}^{N}\prod_{k=0}^{M-1}(1-p_i e^{j2\pi k/M}z^{-1})}=\frac{N'(z)}{D'(z^M)} \tag{10-36}$$

现在考虑一些例子来说明如何在递归滤波器中实现离散超前流水线。

例10.4.4 考虑二阶滤波器的传输函数

330 ~ 331

$$H(z)=\frac{1}{1-a_1z^{-1}-a_2z^{-2}} \tag{10-37}$$

一个三级的等效流水线滤波器可以用下式表示

$$H(z)=\frac{1+a_1z^{-1}+(a_1^2+a_2)z^{-2}-a_1a_2z^{-3}+a_2^2z^{-4}}{1-(a_1^3+3a_1a_2)z^{-3}-a_2^3z^{-6}} \tag{10-38}$$

■

例10.4.5 考虑在 $z=re^{\pm j\theta}$ 位置有复共轭极点的二阶滤波器。它的传输函数是

$$H(z)=\frac{1}{1-2r\cos\theta z^{-1}+r^2z^{-2}} \tag{10-39}$$

通过在 $z=re^{\pm j(\theta+2\pi/3)}$ 和 $z=re^{\pm j(\theta-2\pi/3)}$ 位置添加4个附加的极点和零点，可以使这个滤波器实现3级流水线。等效的流水线滤波器的传输函数是

$$H(z)=\frac{1+2r\cos\theta z^{-1}+(1+2\cos 2\theta)r^2z^{-2}+2r^3\cos\theta z^{-3}+r^4z^{-4}}{1-2r^3\cos 3\theta z^{-3}+r^6z^{-6}} \tag{10-40}$$

当 $\theta\neq 2\pi/3$ 时，上式成立。注意当 $\theta=2\pi/3$ 时，因为 $z=re^{\pm j(\theta+2\pi/3)}=re^{\pm j0}$ 和 $z=re^{\pm j(\theta-2\pi/3)}=r$，对实现3级流水线而言，只需要在 $z=r$ 处有一个附加的极点和零点。等效的流水线滤波器为

$$\begin{aligned}H(z)&=\frac{(1-rz^{-1})}{(1+rz^{-1}+r^2z^{-2})(1-rz^{-1})}\\&=\frac{1-rz^{-1}}{1-r^3z^{-3}}\end{aligned} \tag{10-41}$$

■

例10.4.6 考虑在 $z=r_1$ 和 $z=r_2$ 有实极点的二阶滤波器。它的传输函数是：

$$H(z)=\frac{1}{1-(r_1+r_2)z^{-1}+r_1r_2z^{-2}} \tag{10-42}$$

在 $z=r_1e^{\pm j2\pi/3}$ 和 $z=r_2e^{\pm j2\pi/3}$ 位置添加极点（和零点）可以实现三级流水线。实现流水线后的传输函数是：

$$H(z)=\frac{1+(r_1+r_2)z^{-1}+(r_1^2+r_1r_2+r_2^2)z^{-2}+r_1r_2(r_1+r_2)z^{-3}+r_1^2r_2^2z^{-4}}{1-(r_1^3+r_2^3)z^{-3}+r_1^3r_2^3z^{-6}} \tag{10-43}$$

极点位于 $z=1/2$ 和 $z=3/4$ 的3级流水线二阶滤波器的极点－零点位置如图10-10所示。■

因为添加的极点到原点的距离和原滤波器的极点相同，所以如果原滤波器是稳定的，使用离散超前方法就可以得到稳定的流水线滤波器。公式(10-36)中非递归部分的乘法复杂度是

332

$(NM+1)$，递归部分的乘法复杂度是N，总的复杂度是 $(NM+N+1)$ 个流水线乘法，和M是成线性关系的。尽管如此，它还是比聚类超前的要大。同样要指出，因为每一个乘法器都有M级流水线，因此锁存器的复杂度是M的平方。

现在应用分解技术得到流水线的另一种实现方式，这种方式增加的硬件数量是和M成对数关系的。

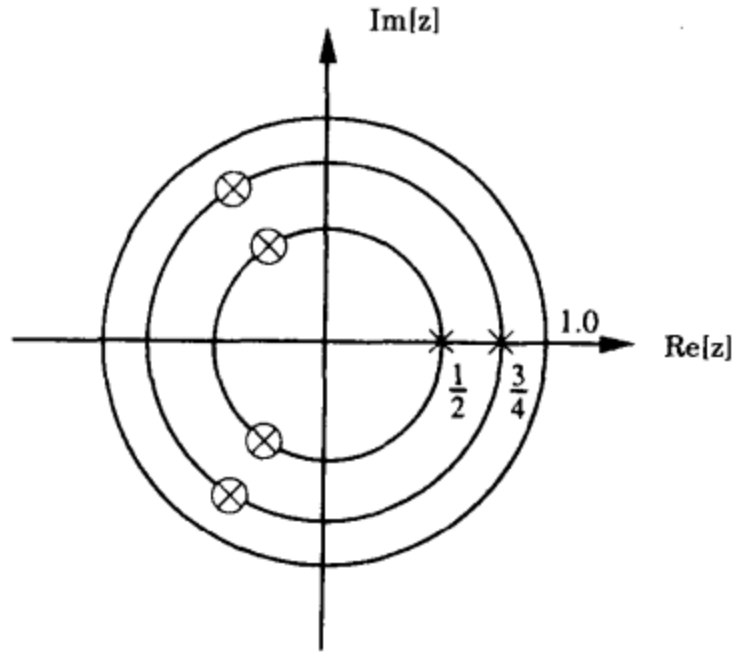


图10-10 应用离散超前方法得到的三级流水线等效稳定滤波器的极点零点位置图

10.4.4 基于2的幂次分解的离散超前流水线

用下式表示一个递归数字滤波器的传输函数

$$H'(z)=\frac{\sum_{i=0}^{N} b_i z^{-i}}{1-\sum_{i=1}^{N} a_i z^{-i}}=\frac{N(z)}{D(z)} \tag{10-44}$$

分子、分母同时乘以 $\left(1-\sum_{i=1}^{N}(-1)^i a_i z^{-i}\right)$ 可以实现两级流水线。等效的实现两级流水线的传输函数是:

$$H'(z)=\frac{\sum_{i=0}^{N} b_i z^{-i}\left(1-\sum_{i=1}^{N}(-1)^i a_i z^{-i}\right)}{\left[1-\sum_{i=1}^{N} a_i z^{-i}\right]\left[1-\sum_{i=1}^{N}(-1)^i a_i z^{-i}\right]}=\frac{N'(z)}{D'(z^2)} \tag{10-45}$$

333

类似地，可以应用这种变换来分别实现4、8和16级流水线。为了实现M级流水线（2的M次幂），要进行 $\log_2 M$ 次这种变换。每次变换都会在把乘法复杂度增加N的同时使速度（采样率）或关键递归环路中流水线的级数加倍。一系列这种变换会在环路流水线级数或速度呈几何增长的同时让硬件的复杂性只有算术关系的增长。

通过应用 $(\log_2 M-1)$ 次这种变换，可以得到等效的传输函数（递归环路中有M级流水线），这个传输函数的乘法复杂度是 $(2N+N\log_2 M+1)$，相对速度的提高或M而言只是一种对数关系。注意尽管乘法操作的个数是对数关系的，但是延时的个数或锁存器的个数是线性的。总的延时或锁存器的个数大约是 $NM(\log_2 M+1)$，其中大约有NM个延时用于非递归部分的实现，大约 $NM\log_2 M$ 个延时用来在 $N\log_2 M$ 个乘法器中的每一个里面实现M级流水线。通过在相

同的位置添加 $N(M-1)$ 个附加的极点和零点，可以得到这种实现方式。为了不把 $N(M-1)$ 个零点用一级非递归的部分实现，探讨一下系数的对称性，并用分解的方法来实现。在分解的实现方法中，第一级实现一个N阶非递归部分，以后的级分别实现$2N$阶，$4N$阶，…，$NM/2$阶非递归部分。由于系数的对称性，每一个非递归部分都要求有N个乘法操作，而和那一部分的阶数无关。下面举一个例子来说明怎样使用离散超前和分解技术在递归滤波器中实现流水线。

例10.4.7 考虑二阶递归滤波器

$$H(z)=\frac{Y(z)}{U(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1-2r\cos\theta z^{-1}+r^2z^{-2}} \tag{10-46}$$

系统的极点位于 $re^{\pm j\theta}$（参看图10-11a）。实现流水线后的传输函数是

$$\begin{aligned}H(z) &= \frac{\left(\sum_{i=0}^{2}b_iz^{-i}\right)}{1-2r^M\cos M\theta z^{-M}+r^{2M}z^{-2M}}\times \\ &\quad \prod_{i=0}^{\log_2 M-1}\left(1+2r^{2^i}\cos 2^i\theta z^{-2^i}+r^{2^{i+1}}z^{-2^{i+1}}\right)\end{aligned} \tag{10-47}$$

当 $\theta\neq 2\pi/M$ 时（当 $\theta=2\pi/M$ 时的流水线实现方式在问题11中讨论）。变换后传输函数的 $2M$个极点位于

[334]

$$z=re^{\pm j(\theta+i(2\pi/M))},\quad i=0,1,2,\cdots,(M-1) \tag{10-48}$$

如图10-11b所示。图10-11c是对流水线二阶滤波器的极点和零点的分解。流水线滤波器的分解实现方法见图10-12。流水线滤波器实现时的复杂度是 $(2\log_2 M+5)$ 个乘法运算。 ■

10.4.5 在通用分解中实现离散超前流水线

和一阶流水线IIR滤波器类似，分解可以消掉的零点，这种技术同样可以扩展到有任意环路流水线级数的高阶流水线IIR滤波器。

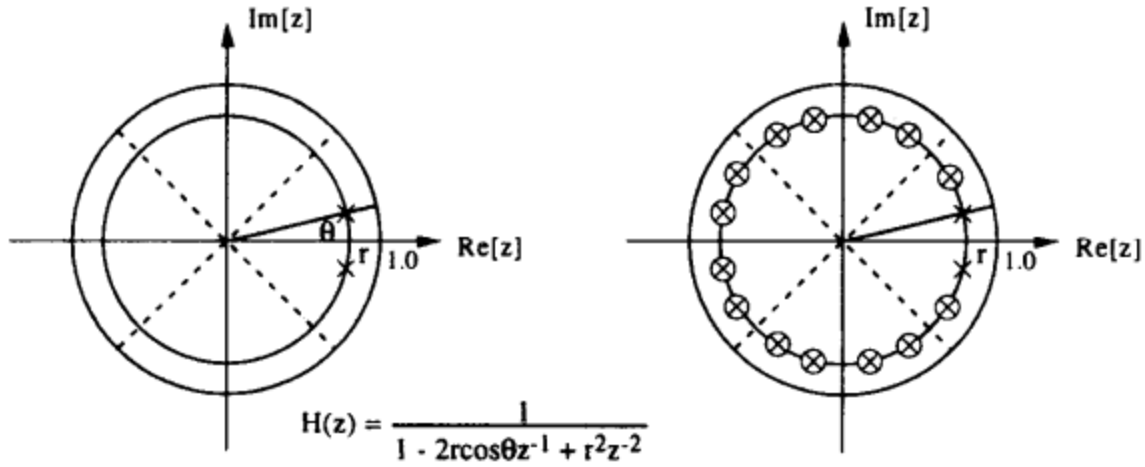
在M级流水线结构的N阶滤波器中，有 $N(M-1)$ 个可以消掉的零点。首先考虑 $M=M_1M_2$ 这种简单分解的情况。这种方式中，系统有 $N(M_1M_2-1)$ 个可以消掉的零点。第一级实现 $N(M_1-1)$个零点，第二级实现 $NM_1(M_2-1)$ 个零点。在 $M_1M_2M_3$ 分解的情况下，第一级实现 $N(M_1-1)$ 个零点，第二级实现 $NM_1(M_2-1)$ 个零点，第三级实现 $NM_1M_2(M_3-1)$ 个零点。总的来说，在一个 $M=M_1M_2\cdots M_P$ 的分解中，P个非递归级分别实现 $N(M_1-1)$, $NM_1(M_2-1)$, …，$NM_1M_2M_{P-1}(M_P-1)$ 个零点，总共 $N(M-1)$ 个零点。通用分解中非递归部分需要大约 NM 个延时和 $N\sum_{i=1}^{P}(M_i-1)$个乘法器（每一个这种乘法器为了实现流水线也要求M个锁存器）。

例10.3.3中已经介绍了如何在一阶IIR滤波器中应用通用分解技术。任何高阶的递归滤波器都能被分解成一阶的组合。和例10.3.3类似的通用分解技术可以应用到这些一阶部分，然后那些复共轭部分就能被组合起来以便得到用实数乘法表示的分解后的形式。

10.4.6 受限的滤波器设计技术

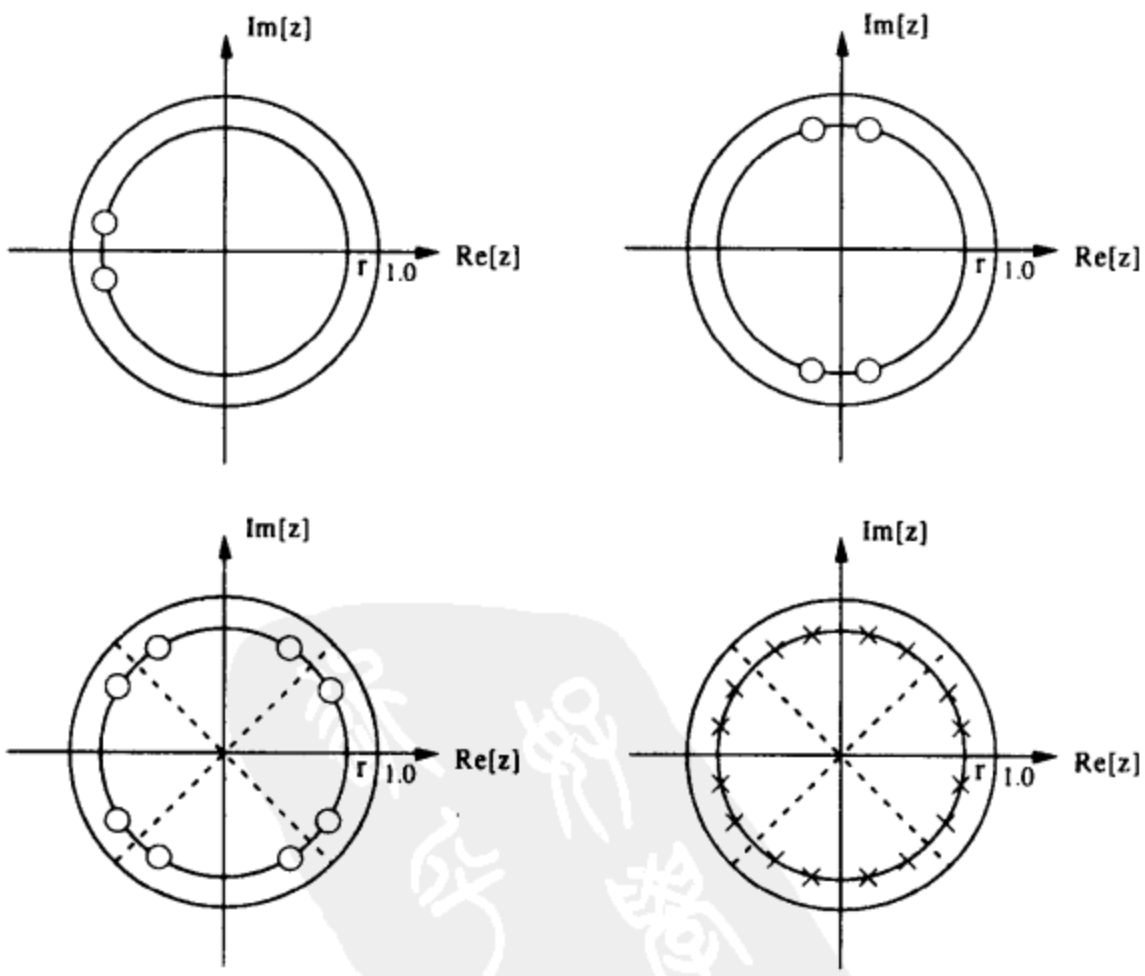
聚类超前和离散超前技术通过极点-零点相消实现流水线。除了那些可以消掉的零点引起

的硬件代价之外，这种流水线滤波器还受有限字长导致的不精确的极点-零点相消的影响。为了避免这些缺点，在分母限定是 z^M 而不是 z 的多项式时，即分母能表示成离散超前的形式时，可以从滤波器的频响特性中直接设计出能进行流水线操作的传输函数。这一节中，简单介绍两种受限滤波器设计技术。受限滤波器设计方法降低了对硬件复杂度的要求，并有更少的输出舍入噪声。把零点放在-1位置不要求乘法操作；把复共轭的零点放在单位圆上，只要求一个乘法操作；这些都可以减少和零点有关的硬件复杂度。 335



a) 二阶滤波器的极点图，滤波器的零点并没有清楚的画出来

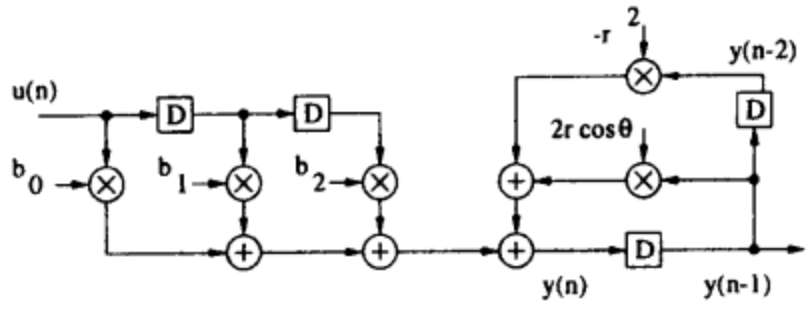
b) 8级环路流水线二阶直接形式滤波器的零极点图



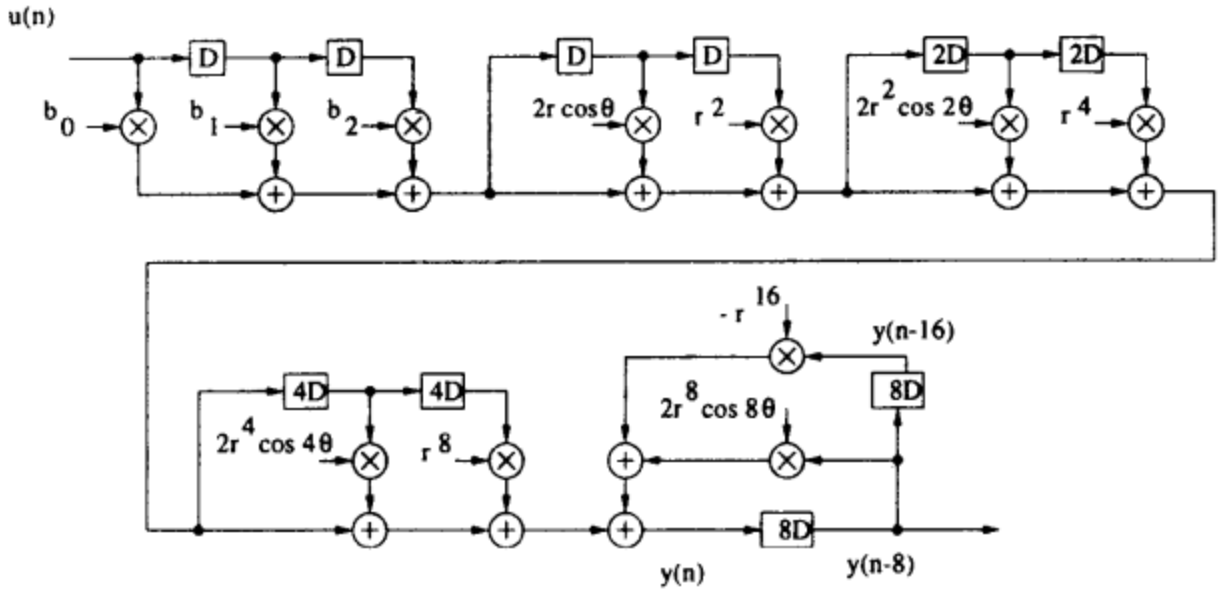
$$H(z) = \frac{(1 + 2r\cos\theta z^{-1} + r^2 z^{-2})(1 + 2r^2\cos 2\theta z^{-2} + r^4 z^{-4})(1 + 2r^4\cos 4\theta z^{-4} + r^8 z^{-8})}{1 - 2r^8\cos 8\theta z^{-8} + r^{16} z^{-16}}$$

c) 二阶流水线滤波器的极点和零点的分解

图 10-11



a)



b)

图10-12 原二阶滤波器的实现和离散超前递归滤波器的实现，它使用了分解技术，有8级递归环路流水线

10.4.6.1 修改的Deczky滤波器设计

[8]中的设计方法首先讲了滤波器的幅度和群延时响应与极点和零点的半径、角度的函数关系。然后针对极点和零点的半径和角度得到了幅度和群延时的偏导数方程。这些偏导数被用于Fletcher-Powell算法来最小化近似误差[9]。Fletcher-Powell算法只使用一阶导数来找到有几个变量的函数的局部极小值。

336～337

为了得到可以实现M级流水线滤波器，它的分母的偏导数用 z^M 的幂而不是 z 的幂重新计算，这些等式被用于Fletcher-Powell算法中[10]。这样，结果得到的传输函数是用 z^M 表示的。

使用修改的Deczky方法进行受限滤波器设计的过程总结如下：

- 对于给定的滤波器指标（通带、阻带、通带纹波、阻带纹波、M级流水线），从1个复数极点对开始滤波器的设计。然后，使分母的阶数是$2M$。我们通常从位于单位圆上的M个零点对开始。这些零点中的一部分被置于单位圆上的阻带内以便抵消由于抽样结构引起的重复极点的影响。
- 如果1个复数极点对不能满足滤波器的指标，可以增加极点和零点的数目，调整极点和零点的初始位置，调整通带和阻带的权重系数。重新设计滤波器。重复这个过程直到满足滤波器的指标。

例10.4.8 考虑一个可以实现4级流水线的低通滤波器，指标为：

通带：$0-0.2\pi$(0.5dB)，阻带：$0.3\pi-\pi$ (20dB)

用修改的Deczky方法得到的传输函数是：

$$N(z)=0.09742(1-1.16548z^{-1}+z^{-2})(1-0.29069z^{-1}+z^{-2})$$
$$\times(1+1.19469z^{-1}+z^{-2})(1+1.75885z^{-1}+z^{-2})$$
$$D(z)=1+0.63459z^{-4}+0.10067z^{-8}$$

注意 $D(z)$ 可以被继续分解成 $D(z)=(1+0.31729z^{-4})^2$。所有的零点都位于单位圆上的阻带内，极点的半径都是0.75并且彼此之间的角度间隔是0.5π。传输函数的零极点图如图10-13所示。注意没有零极点相消时就实现了4级流水线。■

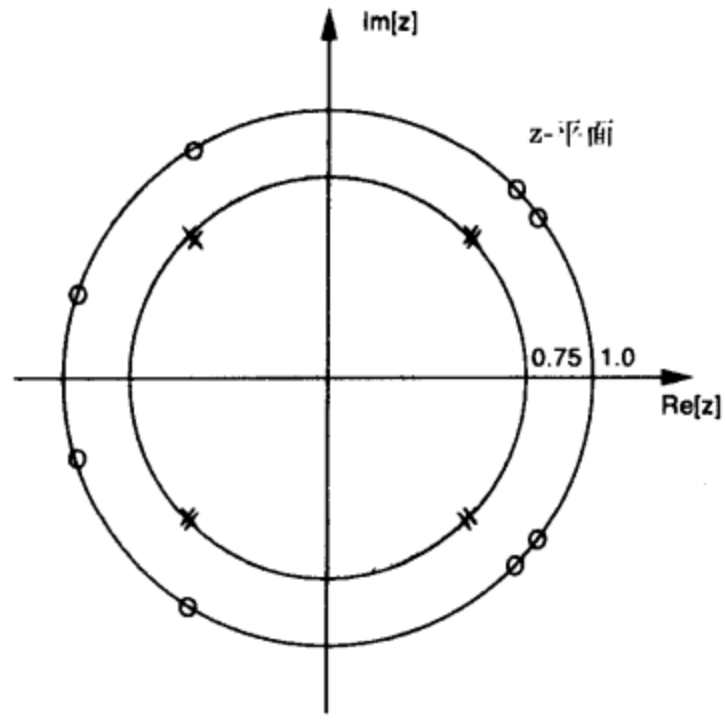


图10-13 例10.4.8中传输函数的极-零点

10.4.6.2 Martinez-Parks 抽样滤波器设计

为了得到能实现流水线的传输函数，也可以应用[11][12]中的抽样滤波器设计方法，这种方法使用等纹波幅度的限制和另外一个限制，即所有的零点都在单位圆上。

[11]中修改后的Remez交换算法通过重复更新分子的数值和分母平方的数值找到了对等纹波的解决方法。这个算法的原理如下：给定通带纹波、通带边界、阻带边界和滤波器的阶数，在选择阻带的纹波时，要满足使最终得到的滤波器都有等纹波这个条件。

在[12]中，对[11]中分母平方的数值方程稍做修改，使所得到的分母只有 z^M 的幂。[12]中的一些例子表明了当滤波器的频响特性是窄带而且边界很陡时，这种方法是很有效的。

10.5 IIR滤波器的并行处理

在这一节，将讨论IIR滤波器的并行处理技术。先讨论一个简单的一阶IIR滤波器的并行处理。之后，将讨论高阶滤波器的并行处理。

例10.5.1 考虑公式(10-49)给出的一阶IIR滤波器的传输函数

$$H(z)=\frac{z^{-1}}{1-az^{-1}} \tag{10-49}$$

其中，$|a|<1$保证稳定性。这个传输函数只在 $z=a$ 有一个极点。相应的输入输出关系可以写成：

$$y(n+1)=ay(n)+u(n) \tag{10-50}$$

考虑在前述滤波器中设计的4并行结构（$L=4$）。注意在并行系统中，每一个延时单元都被当作是一个块延迟，其中每个块的时钟周期是采样周期的4倍。所以和公式(10-50)不同，输入和 $y(n)$ 更新的是 $y(n+4)$。通过对公式(10-50)式中递归的迭代或通过使用超前技术，能得到

338 ~ 339

$$y(n+4)=a^4y(n)+a^3u(n)+a^2u(n+1)+au(n+2)+u(n+3) \tag{10-51}$$

在上式中代入 $n=4k$，

$$y(4k+4)=a^4y(4k)+a^3u(4k)+a^2u(4k+1)+au(4k+2)+u(4k+3) \tag{10-52}$$

相应的结构如图10-14所示。注意并行滤波器中极点位置的不同。原系统的极点在 $z=a$，而并行滤波器的极点在 $z=a^4$，因为 $|a^4|$ 小于或等于 $|a|$（$|a|<1$），所以它距离原点更近。极点位置的改变使系统受舍入噪声的影响大大减少。图10-15是 $L=4$ 时，代入 $n=4k+4$, $4k+5$, $4k+6$, $4k+7$ 得到的直接块处理结构。 ■

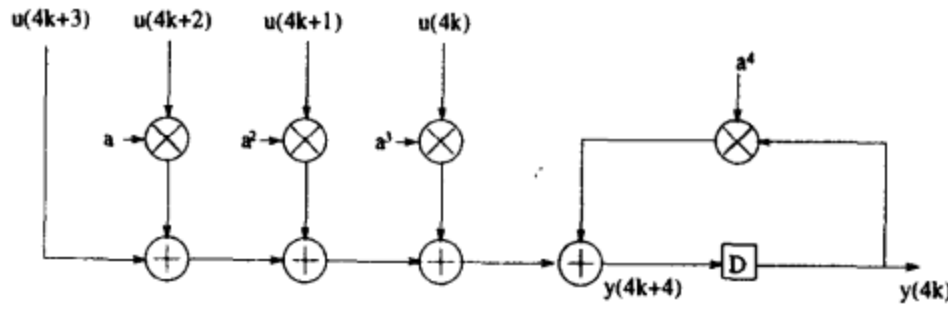


图10-14 一个4并行滤波器中的单状态更新操作

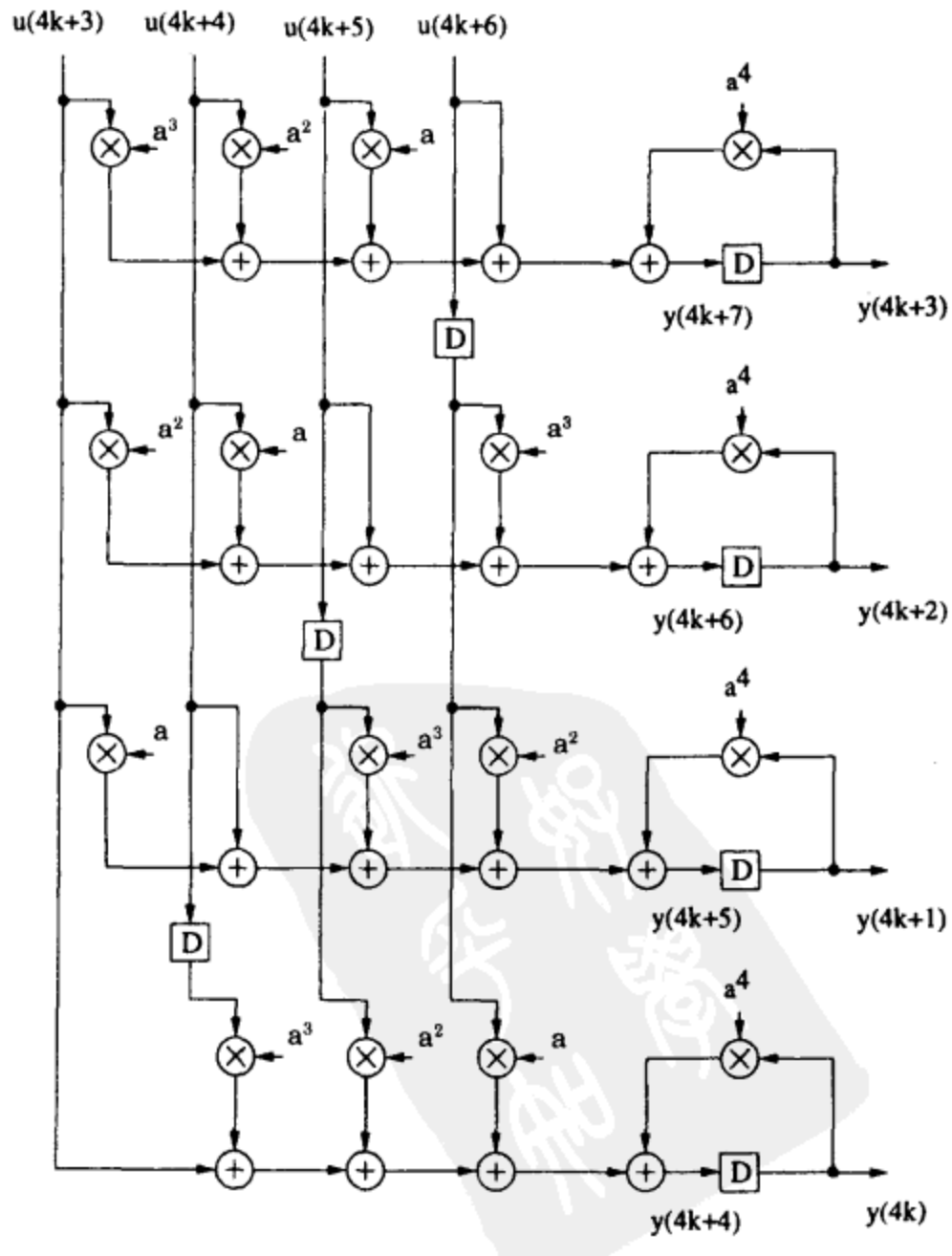


图10-15 一个4并行一阶递归滤波器

因为每个输出要求L个乘加操作，并且共有L个输出，所以这种结构的硬件复杂度是 L^2 个乘加操作。

很明显，直接实现块处理结构的硬件代价是很大的。可以通过并行计算来减少这种成平方关系增长的硬件复杂度。因为在块处理模式中，一个硬件延时单元代表了L个采样延时，所以不能在这里应用基于离散超前的分解技术。但是，可以不去单独计算 $y(4k+1)$，我们可以用 $y(4k)$ 来计算 $y(4k+1)$，用 $y(4k+1)$ 来计算 $y(4k+2)$，用 $y(4k+2)$ 来计算 $y(4k+3)$，这会显著减少硬件的复杂度；代价是系统的整体延迟增加了。这种步进计算 $y(4k+1)$、$y(4k+2)$、$y(4k+3)$ 的技术叫增量块处理技术。

例10.5.2 考虑例10.5.1中的一阶滤波器。为了得到有最小硬件复杂度的4并行滤波器结构，而不是像图10-15中一样只是简单的把原来的硬件结构重复4遍，可以使用增量计算技术来减少硬件复杂度。首先，设计计算 $y(4k)$ 的电路（同图10-14）。然后，利用公式(10-53)从 $y(4k)$ 中得到 $y(4k+1)$

$$y(4k+1) = ay(4k) + u(4k) \tag{10-53}$$

类似地，可以从 $y(4k+1)$ 中得到 $y(4k+2)$，从 $y(4k+2)$ 得到 $y(4k+3)$。完整的结构如图10-16所示。注意硬件复杂度已经从 L^2 减少到 $2L-1$，代价是计算 $y(4k+1)$，$y(4k+2)$ 和 $y(4k+3)$ 增加的时间。 ■

340 ~ 341

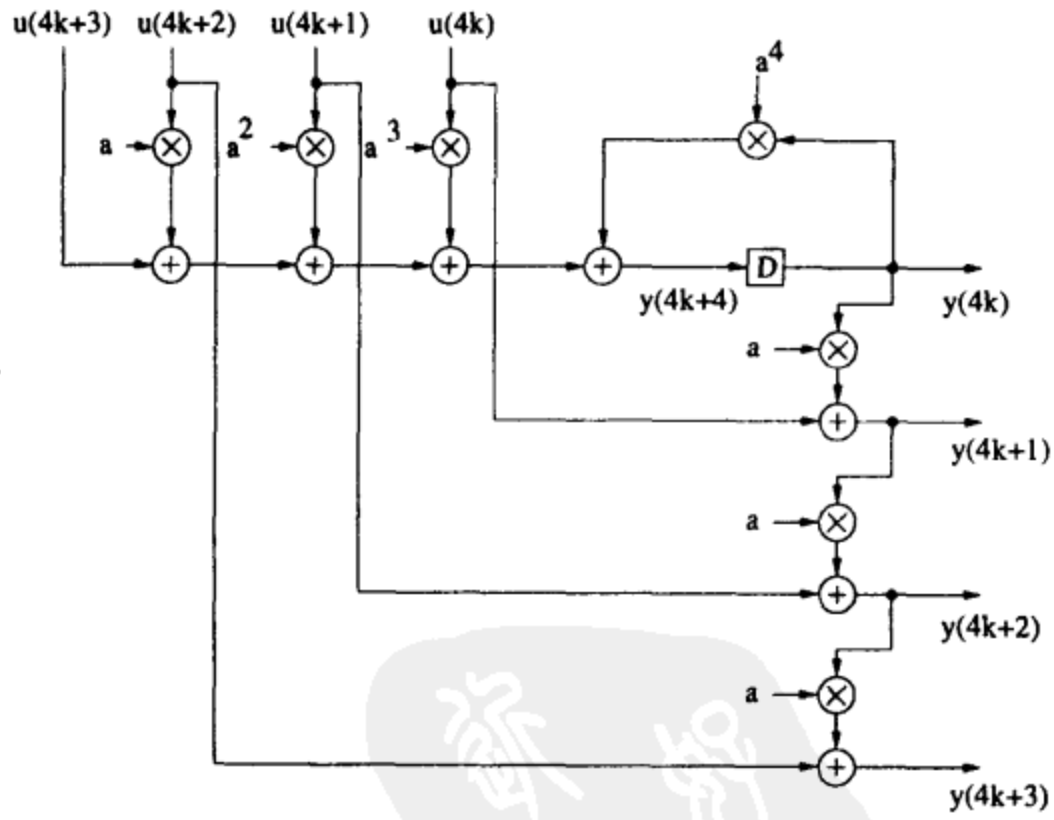


图10-16 $L=4$的增量块滤波器结构

例10.5.3 考虑由公式(10-54)的传输函数描述的二阶IIR滤波器。

$$H(z) = \frac{(1+z^{-1})^2}{1-\frac{5}{4}z^{-1}+\frac{3}{8}z^{-2}} \tag{10-54}$$

这个滤波器的零极点位置如图10-17所示。从中得到一个3并行的IIR滤波器，在每一个时钟周期里，处理3个输入并能得到3个输出。

因为滤波器的阶数是2，所以需要独立地更新两个输出，第三个输出可以使用以上两个已经更新的输出，在反馈环路外用增量技术计算出来。假设使用环路更新操作来计算 $y(3k)$ 和 $y(3k+1)$，使用增量技术来计算 $y(3k+2)$。从传输函数中，能得到

$$y(n)=\frac{5}{4}y(n-1)-\frac{3}{8}y(n-2)+f(n) \tag{10-55}$$

其中，$f(n)=u(n)+2u(n-1)+u(n-2)$。3并行系统的环路更新操作如图10-18所示，其中使用 $y(3k)$ 和 $y(3k+1)$来计算 $y(3k+3)$ 和 $y(3k+4)$。如果能实现用 $y(n)$ 和 $y(n+1)$来计算 $y(n+3)$，那么就能实现用 $y(3k)$ 和 $y(3k+1)$ 来计算 $y(3k+3)$。类似地，如果能用 $y(n)$ 和 $y(n+1)$ 来表示 $y(n+4)$，就能用 $y(3k)$ 和 $y(3k+1)$ 来计算 $y(3k+4)$，如图10-19所示。这些状态更新操作与$M=2$和$M=3$时的聚类超前操作相对应。2级和3级的聚类超前方程如下所示：

342

$$\begin{aligned}y(n)=&\ \tfrac{5}{4}y(n-1)-\tfrac{3}{8}y(n-2)+f(n)\\=&\ \tfrac{5}{4}[\tfrac{5}{4}y(n-2)-\tfrac{3}{8}y(n-3)+f(n-1)]-\tfrac{3}{8}y(n-2)+f(n)\\=&\ \tfrac{19}{16}[\tfrac{5}{4}y(n-3)-\tfrac{3}{8}y(n-4)+f(n-2)]-\tfrac{15}{32}y(n-3)\\&+\tfrac{5}{4}f(n-1)+f(n)\end{aligned} \tag{10-56}$$

在公式(10-56)中代入 $n=3k+3$ 和 $n=3k+4$，能得到如下的两个环路更新方程：

$$y(3k+3)=\frac{19}{16}y(3k+1)-\frac{15}{32}y(3k)+\frac{5}{4}f(3k+2)+f(3k+3) \tag{10-57}$$

$$y(3k+4)=\frac{65}{64}y(3k+1)-\frac{57}{128}y(3k)+\frac{19}{16}f(3k+2)+\frac{5}{4}f(3k+3)+f(3k+4) \tag{10-58}$$

输出 $y(3k+2)$ 可以按下式增量计算得到：

$$y(3k+2)=\frac{5}{4}y(3k+1)-\frac{3}{8}y(3k)+f(3k+2) \tag{10-59}$$

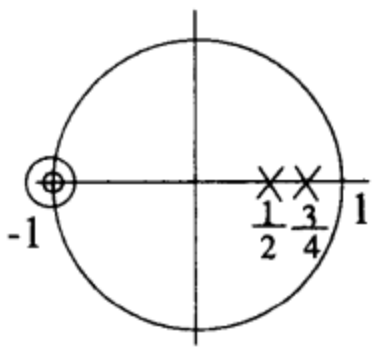


图10-17 传输函数的零极点图

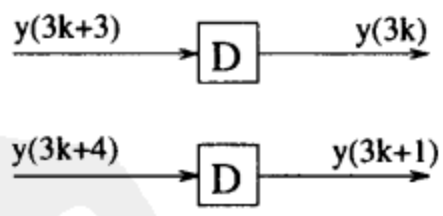


图10-18 块数是3时的环路更新操作

343

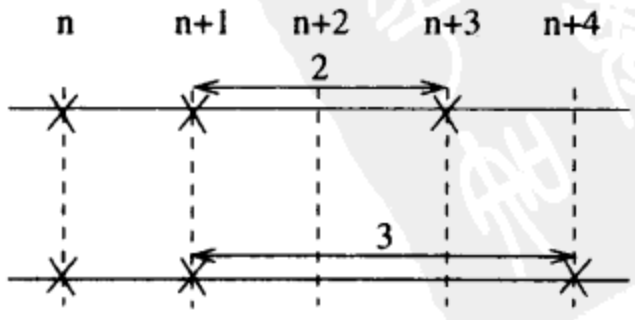


图10-19 递归输出之间的关系

块结构如图10-20所示。 ■

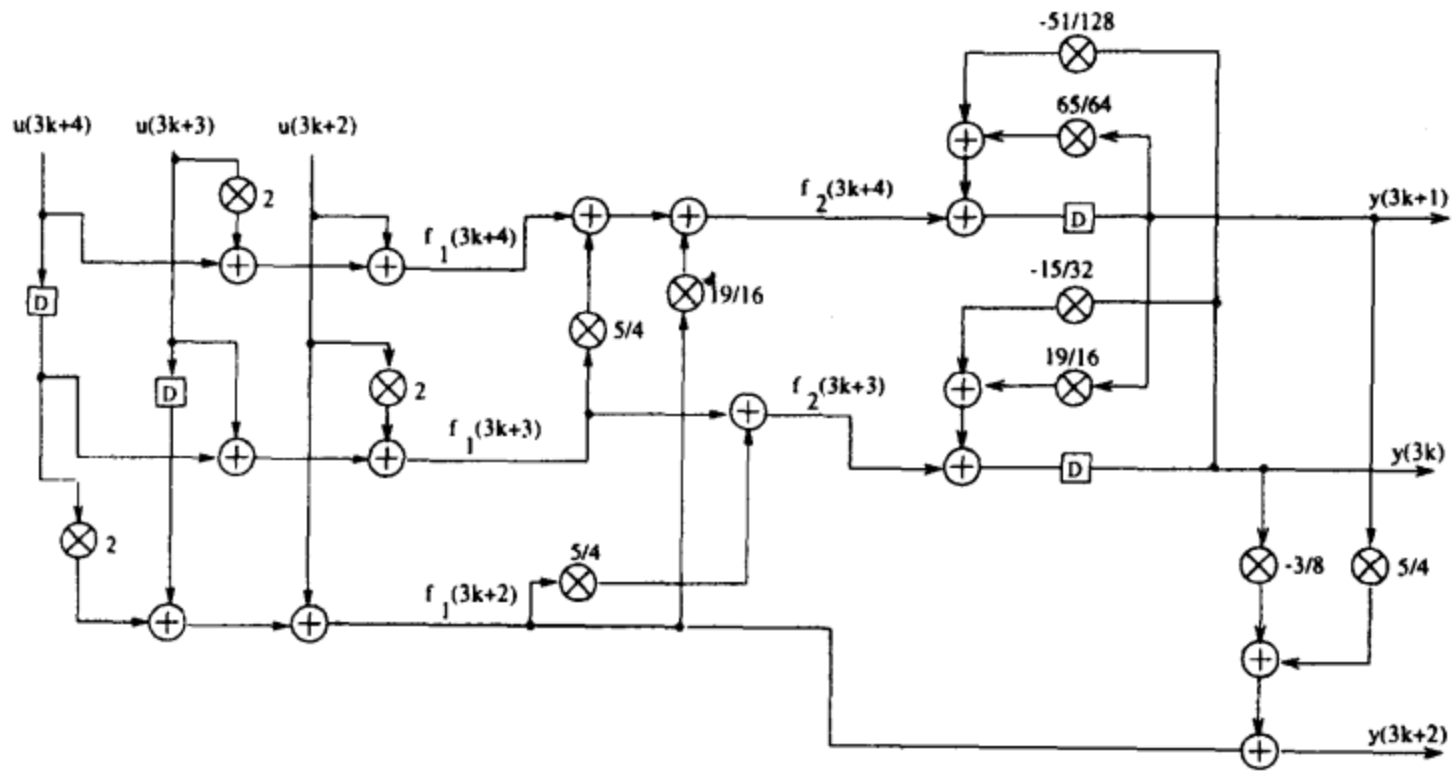


图10-20 二阶IIR滤波器 ($L=3$) 的块结构

注释

- 原顺序系统在 $\frac{1}{2}$、$\frac{3}{4}$ 有两个极点。现在考虑新的并行系统的极点位置。把上述两个状态更新方程用矩阵形式表示:

$$\begin{pmatrix} y(3k+3) \\ y(3k+4) \end{pmatrix} = \begin{pmatrix} -\frac{15}{32} & \frac{19}{16} \\ -\frac{57}{128} & \frac{65}{64} \end{pmatrix} \begin{pmatrix} y(3k) \\ y(3k+1) \end{pmatrix} + \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \tag{10-60}$$

把系统矩阵记做A。A的特征值是 $\left(\frac{1}{2}\right)^3$、$\left(\frac{3}{4}\right)^3$，它们是新的并行系统的极点。这样，并行系统就更加稳定。需要注意的是并行系统和原系统有同样数目的极点。 344

- 对于二阶IIR滤波器(N=2)而言，总共有 $3L + [(L-2) + (L-1)] + 4 + 2(L-2) = 7L-3$ 个乘法操作，其中第一项来自分子，第二项来自环路更新操作，第三项是环路乘法，第四项来自增量计算。很明显，乘法复杂度和块数L是成线性关系的。可以使用快速并行滤波器结构和对增量计算结果输出的子结构进行共享来进一步减少乘法复杂度（参看习题17和习题18）。

一般地，对N阶IIR滤波器来说，它的L级增量并行处理结构可以分两步得到。第一步，先使用由聚类超前技术得到的环路更新方程独立地计算前N个输出采样 $y(Lk)$, $y(Lk+1)$, …, $y(Lk+N-1)$; 第二步，利用得到的前N个输出采样，增量计算剩下的 $L-N$ 个采样 $y(Lk+N)$, …, $y(Lk+L-1)$。注意当 $L<N$ 时，增量计算技术就不适用了。

并行滤波器还可以用来减少多速率系统中乘法运算的个数。在包含有抽样器的系统中，只有一个输出需要被计算出来。要实现这个目的，可以使用增量块滤波器并保持第一个输出不变，或者使用离散超前技术来更新所需要的输出。哪一种方法更好决定于滤波器的阶数和

抽样的多少（参看习题19）。类似地，在包含扩展器的系统中，在零输入的情况下，就不必进行乘法运算。

10.6 组合了流水线和并行处理的IIR滤波器

IIR滤波器中，把流水线和并行处理组合起来，可以使采样率提高 $L \times M$ 倍或者在同样的采样率下降低功耗，其中L是块处理的个数，M是流水线的级数。

例10.6.1 考虑一阶IIR滤波器，它的传输函数是

$$H(z) = \frac{1}{1 - az^{-1}} \tag{10-61}$$

我们想要的滤波器将有4级流水线和3个块处理，即$M=4$，$L=3$。

因为滤波器的阶数是1，所以只需要1个环路更新操作。其余的3个输出可以增量计算出来。同时因为$M=4$，环路中必须包含4个延时单元，如图10-21所示。因为$L=3$，每一个延时单元代表了一个块延时，即3个采样延时。所以 $y(3k+12)$ 需要用 $y(3k)$ 和输入来表示（参看图10-21）。要达到上述目的，可以使用如下的超前算法：

345

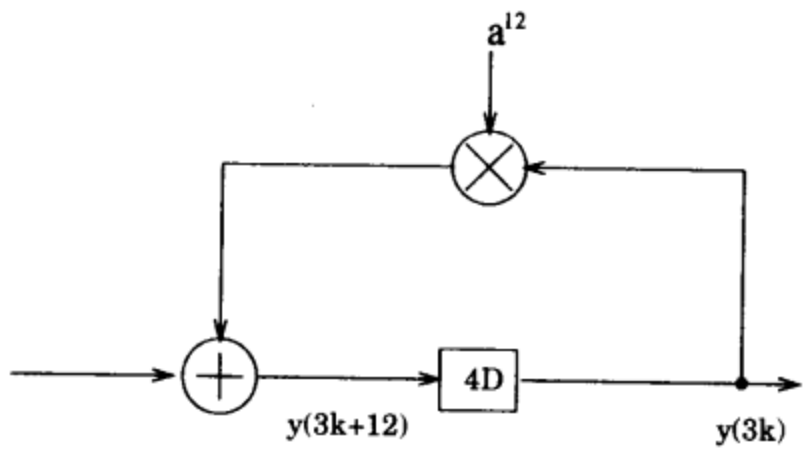


图10-21 流水线增量块系统的环路更新

$$\begin{aligned} y(n) &= ay(n-1) + u(n) \\ &= a^2y(n-2) + au(n-1) + u(n) \\ &= \cdots \\ &= a^{12}y(n-12) + a^{11}u(n-11) + a^{10}u(n-10) + \cdots + u(n) \end{aligned} \tag{10-62}$$

代入 $n=3k+12$，得到

$$\begin{aligned} y(3k+12) = \; & a^{12}y(3k) \\ & +a^{11}u(3k+1) + a^{10}u(3k+2) + a^9u(3k+3) \\ & +a^8u(3k+4) + a^7u(3k+5) + a^6u(3k+6) \\ & +a^5u(3k+7) + a^4u(3k+8) + a^3u(3k+9) \\ & +a^2u(3k+10) + au(3k+11) + u(3k+12) \\ = \; & a^{12}y(3k) + a^6f_2(3k+6) + a^3f_1(3k+9) + f_1(3k+12) \end{aligned} \tag{10-63}$$

其中

$$\begin{aligned} f_1(3k+12) &= a^2u(3k+10) + au(3k+11) + u(3k+12) \\ f_2(3k+12) &= a^3f_1(3k+9) + f_1(3k+12) \end{aligned} \tag{10-64}$$

所得到的并行滤波器结构如图10-22所示。 ■

注释

这个并行流水线滤波器有4个极点：a^3, $-a^3$, ja^3, $-ja^3$。因为流水线级数是4，滤波器阶数是1，所以新系统中共有被相同的角度分开的极点4个。因为块数是3，我们希望各个极点到原点的距离是$|a^3|$。

在流水线阶段，使用了分解技术。乘法复杂度（假设流水线级数M是2的幂）可以按下式计算：$(L-1)+\log_2 M+1+(L-1)=2L-1+\log_2 M$。对$L$而言，它是线性关系的；对$M$而言是对数关系的，对$L$和$M$而言是相加的关系。 346

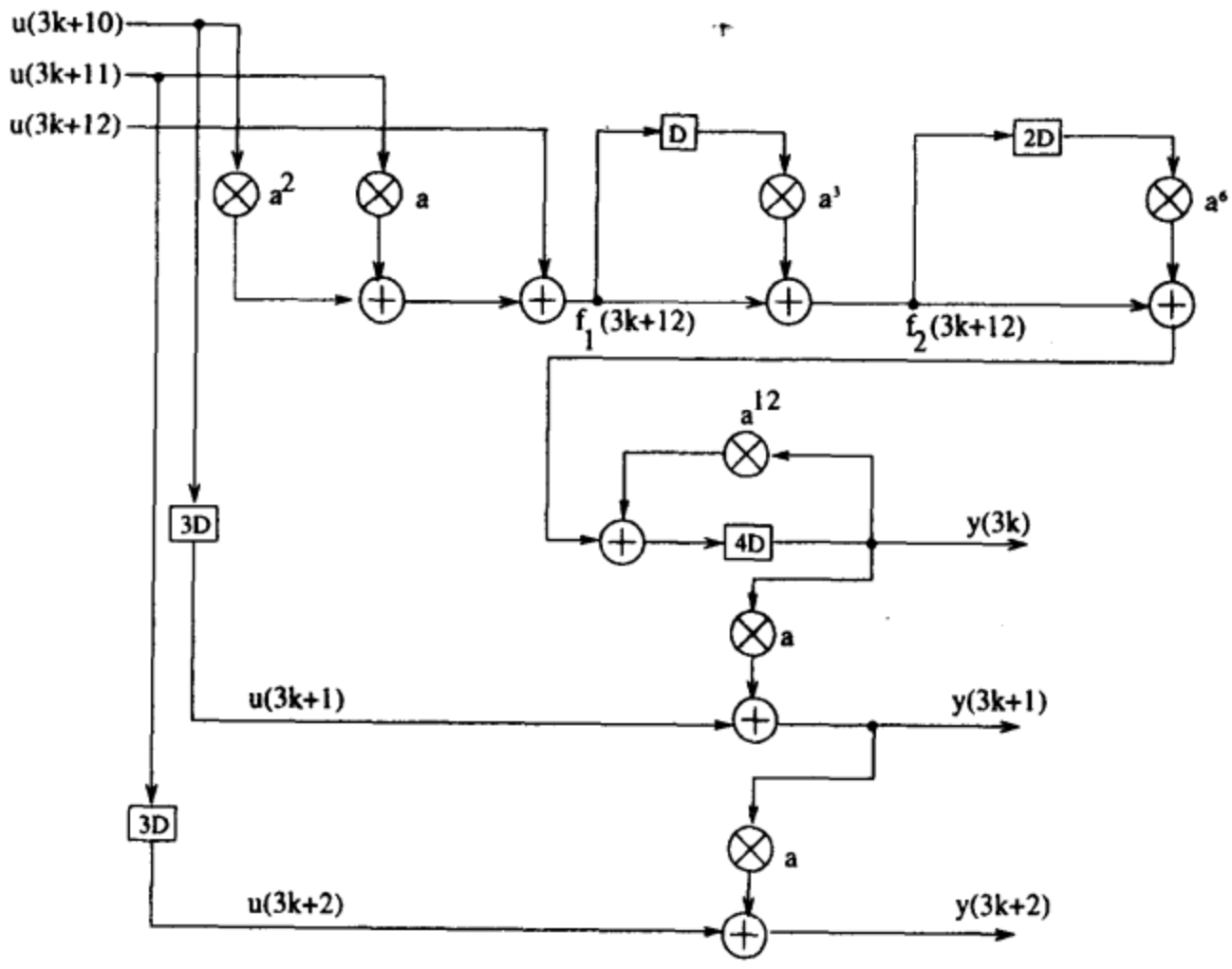


图10-22 流水线块结构的滤波器

例10.6.2 再次考虑例10.5.3中的二阶滤波器。这次的目的是设计一个$L=3$，$M=2$的流水线增量块系统。这个系统的环路更新操作如图10-23所示。可以使用和聚类超前类似的方法，用$y(3k)$和$y(3k+1)$来更新$y(3k+6)$和$y(3k+7)$。然后，通过下标替换，可以得到最后的系统方程。假设系统更新矩阵是A。因为原系统的极点是$\frac{1}{2}$、$\frac{3}{4}$，A的特征值可以计算出来是$\left(\frac{1}{2}\right)^6$、$\left(\frac{3}{4}\right)^6$。新并行流水线二阶滤波器的极点是$A$的特征值的平方根，即$\left(\frac{1}{2}\right)^3$、$-\left(\frac{1}{2}\right)^3$、$\left(\frac{3}{4}\right)^3$、$-\left(\frac{3}{4}\right)^3$。 ■

注释

一般地，按照下面给出的步骤可以用一种系统的方法来计算新并行流水线系统中极点的位置。

(1) 使用LM级超前写出环路更新方程，M和L分别代表流水线和并行处理的级数。

(2) 写出并行流水线滤波器的状态空间表达式，其中状态矩阵A是$N \times N$维，N是滤波器的阶数。

347 (3) 计算矩阵A的特征值λ_i，$1 \leq i \leq N$。

(4) 新并行流水线系统的NM个极点和A的特征值的M次方根 $\lambda_i^{\frac{1}{M}}$ 相对应。

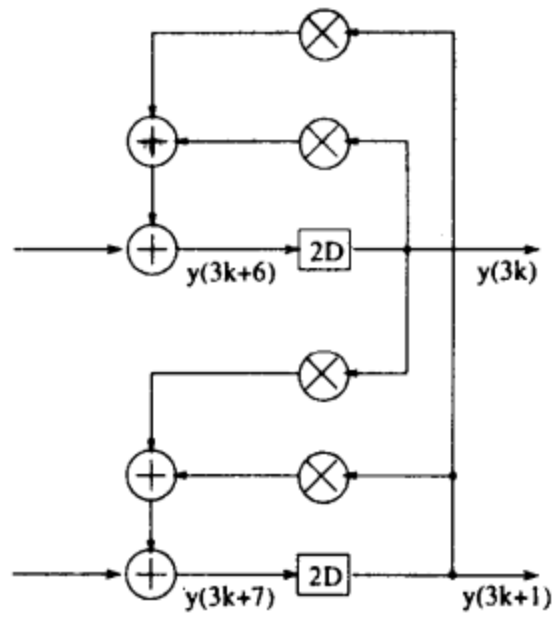


图10-23 流水线块系统的环路更新

10.7 利用流水线和并行处理的低功耗IIR滤波器设计

正如第3章所说的，流水线和并行处理可以通过减小供电电压来降低系统的功耗。

例10.7.1 考虑下面4阶Chebyshev低通滤波器[13]的一种低功耗实现方式

$$H(z) = \frac{0.001836(1+z^{-1})^4}{(1-1.5548z^{-1}+0.6493z^{-2})(1-1.4996z^{-1}+0.8482z^{-2})}$$

假设乘法器的电容起主要作用，加法器的电容可以忽略。假设原系统的供电电压是5 V，CMOS阈值电压是1 V。通过基于2的幂次分解的离散超前技术，可以用 $N(z)/D(z)$ 来表示4级流水线的传输函数，其中

$$\begin{aligned} N(z) &= 0.001836(1+z^{-1})^4 \\ &\quad \times (1+1.5548z^{-1}+0.6493z^{-2})(1+1.4996z^{-1}+0.8482z^{-2}) \\ &\quad \times (1+1.1188z^{-2}+0.4216z^{-4})(1+0.5524z^{-2}+0.7194z^{-4}) \\ D(z) &= (1-0.4085z^{-4}+0.1777z^{-8})(1+1.1337z^{-4}+0.5175z^{-8}) \end{aligned}$$

使用流水线可以在不改变采样速度的情况下降低功耗。因为流水线系统有比原系统更短的关键路径，在和原系统相同的时钟周期里，进行充电和放电的电容数量要少。所以，流水线系统的供电电压可以降低到 βV_0，其中 $\beta<1$，V_0 是原系统的供电电压。没有流水线结构的系统的传输延时是 348

$$T_{pd} = \frac{C_{charge} \times 5}{\kappa(5-1)^2}$$

在4级流水线系统中，传输延时 T_{pd} 是

$$T_{pd} = \frac{C_{charge}}{4}\frac{5\beta}{\kappa(5\beta-1)^2}$$

让上述的两个传输延时 T_{pd} 相等，得到 β=0.476。所以，流水线系统的供电电压可以降到2.38 V。注意原滤波器和流水线滤波器的时钟周期都是 T_{pd}。采样周期也是 T_{pd}。所以，原系统的功耗是

$$P_{seq} = \frac{C_{total}^{(seq)} \times 5^2}{T_{pd}} = m_{seq}C_M 5^2 f_s$$

$m_{seq}=5$ 代表原系统中乘法运算的个数，C_M 代表一个乘法器的电容。流水线系统的功耗是

$$P_{pip} = \frac{C_{total}^{(pip)} \times 2.38^2}{T_{pd}} = m_{pip}C_M(2.38)^2 f_s$$

$m_{pip}=13$ 是4级流水线系统中乘法器的个数。那么，

$$Ratio = \frac{P_{pip}}{P_{seq}} = \left(\frac{13}{5}\right)\left(\frac{2.38}{5}\right)^2 = 0.5891 \tag{10-65}$$

所以，4级流水线系统的功耗是没有流水线结构的系统的58.91%。 ■

例10.7.2 考虑例10.5.3中的二阶IIR滤波器。输入输出关系是

$$y(n) = \frac{5}{4}y(n-1) - \frac{3}{8}y(n-2) + u(n) + 2u(n-1) + u(n-2) \tag{10-66}$$

滤波器结构见图10-24。考虑在例10.5.3中得到的3并行滤波器。现在应用并行处理来降低功耗。假设并行滤波器和顺序滤波器的采样速度保持相同。因为在3并行系统中，每个时钟周期产生3个输出，为了保持相同的采样率，并行系统的时钟周期应该是采样周期的3倍，或者是顺序滤波器的时钟周期的3倍。顺序滤波器的关键路径是1个乘加运算时间。为了在并行系统中保持相同的关键路径，一些流水线锁存器被放在图10-25中用虚线标出的前馈割集处。这样，并行滤波器的关键路径是一个乘法两个加法运算时间。假设不论电容还是计算时间都是乘法器占主要因素。忽略加法器和锁存器的电容和运算时间因素。这说明并行滤波器要花费3倍多的时间来对关键路径上同样数目的电容（1个乘法器的C_M）进行冲放电，这反过来又说明3并行IIR滤波器的功耗可以降低到 βV_0，其中 $\beta<1$，V_0=5.0 V是原顺序滤波器的供电电压。假设阈值电压 V_t 是0.6 V。顺序滤波器的传输延时或时钟周期是 349

$$T_{seq} = \frac{C_M V_0}{k(V_0-V_t)^2} \tag{10-67}$$

并行滤波器的传输延时或时钟周期是

$$T_{par} = \frac{C_M \beta V_0}{k(\beta V_0-V_t)^2} \tag{10-68}$$

通过解方程 $T_{par}=3T_{seq}$，即

$$3\cdot\frac{V_0}{(V_0-V_t)^2} = \frac{\beta V_0}{(\beta V_0-V_t)^2} \tag{10-69}$$

得到 β=0.4673。所以，3并行滤波器的供电电压可以降低到 βV_0=2.3365 V。顺序系统的总功耗是：

$$P_{seq} = 3C_M V_0^2 f_s \tag{10-70}$$

3并行系统的总共功耗是：

$$P_{par} = 12C_M(\beta V_0)^2 \frac{f_s}{3} = 4\beta^2 C_M V_0^2 f_s \tag{10-71}$$

然后，得到：

$$Ratio = \frac{P_{par}}{P_{seq}} = \frac{4\beta^2}{3} = 29.116\% \tag{10-72}$$

所以，3并行IIR滤波器的功耗是原顺序滤波器的 29.116%。 ■

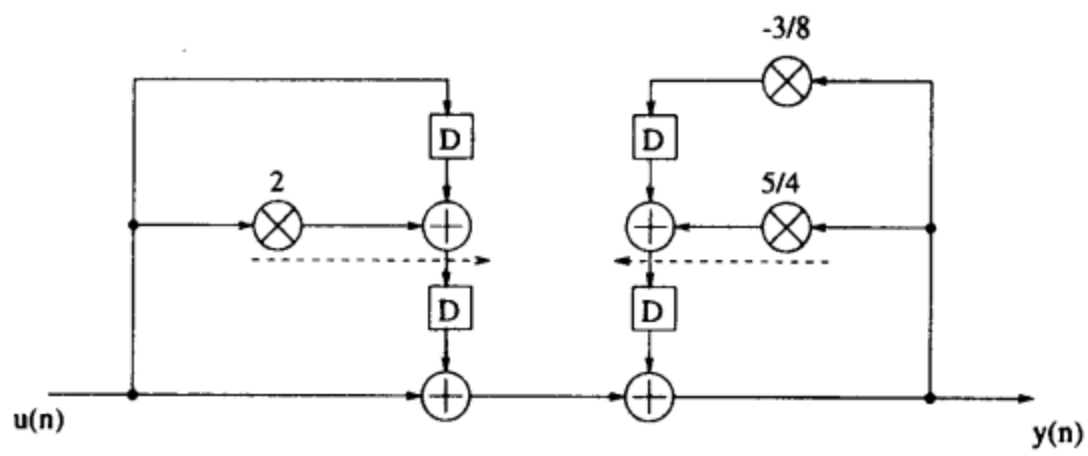


图10-24 二阶IIR滤波器的结构

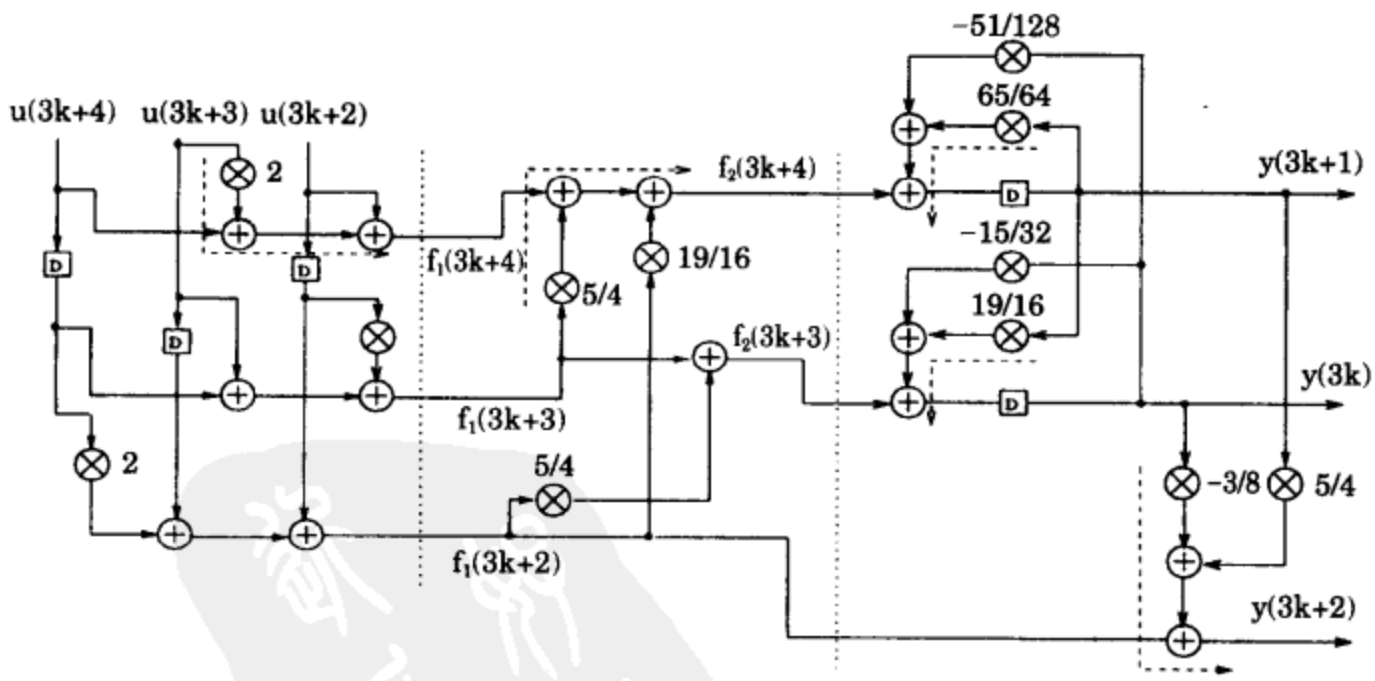


图10-25 流水线3并行二阶滤波器的结构

注释

例10.7.1介绍了流水线是怎样被用来降低IIR滤波器的功耗的。例10.7.2介绍了怎样利用并行处理来降低功耗。这两种技术也可以结合起来得到更快或者更低功耗的IIR滤波器结构。

10.8 流水线自适应数字滤波器

自适应数字滤波器因为有长的反馈环路而不同于流水线滤波器。可以用超前计算等技术在这些滤波器中实现流水线，但是因为硬件的大幅增长而使得所得到的结构不适合于用集成电路实现[14]。超前计算在频率修整滤波器（比如IIR数字滤波器）中保持了精确的输入输出映射关系，但是这种精确的输入输出映射关系在自适应滤波器中并不是必须的，因为系数在收敛之前都是变化的。但是，对自适应滤波器的失调误差和自适应时间常数的分析很重要。失调误差表征的是采用自适应滤波器所得到的结果和最优结果之间的误差。自适应时间常数表征的是收敛的速度；时间常数越小，收敛得越快。

350 ~ 351

在这一节，使用弛豫超前变换技术在自适应滤波器中实现了流水线，而基本上没有增加硬件负担，代价是自适应性能有很小的下降。弛豫超前变换建立在对超前变换的某种近似的基础上。这里介绍三种弛豫超前技术，包括乘积近似、和近似、延时近似。不同类型的弛豫超前会有不同形式的流水线拓扑结构。所以，与超前变换不同，弛豫超前变换会得到一系列的流水线拓扑结构，每种结构都会有不同的自适应特性；而超前变换有精确的输入输出映射关系并且变换是惟一的。另外，在弛豫超前中所用的近似在滤波器收敛后或者在收敛的最后阶段通常来说是合理的，但是在最初的收敛阶段这种近似是不合理的。所以，这些流水线自适应滤波器的自适应特性应该用随机技术来分析。

这一节介绍弛豫超前变换和它在LMS自适应滤波器、随机梯度自适应格型滤波器中实现流水线时的应用。

10.8.1 弛豫超前变换

考虑一阶时变递归方程

$$y(n+1) = a(n)y(n) + u(n) \tag{10-73}$$

它的时变系数是 $a(n)$。使用超前变换，$y(n+M)$ 可以用 $y(n)$ 表示：

$$\begin{aligned} y(n+M) &= \prod_{i=0}^{M-1} a(n+M-1-i)y(n) \\ &+ \sum_{i=1}^{M-1}\left[\prod_{j=0}^{i-1} a(n+M-1-j)\right]u(n+M-1-i) + u(n+M-1) \end{aligned} \tag{10-74}$$

$M=4$时公式(10-74)的信号流图如图10-26所示。一般地，超前变换会在递归环路中产生$M-1$个附加的延时单元，这些延时单元可以用来把环路中的乘加操作用流水线实现。例如，图10-26所示结构的迭代周期是 $(T_m + T_a)/4$。注意这种变换并不改变输入输出特性。输入输出特性不改变的代价是公式(10-74)的右侧中第二项所带来的冗余超前变换结构，它和流水线级数M是有关的。这种冗余结构的代价是很昂贵的。但是，在某种特定的条件下，可以把近似表达式代入，来简化公式(10-74)中右侧的各项。根据不同的应用，可以使用不同类型的近似。这被称为弛豫超前变换。

352

在自适应滤波中，我们更关心它的随机特性而不是它的输入输出映射关系。这样，就能以很小的硬件代价应用弛豫超前技术在自适应滤波器中实现流水线，但是要以牺牲某种程度的自适应特性为代价。三种形式的近似可以用来在自适应滤波器中实现流水线，包括乘积近似、和近似、延时近似。不同的近似会有不同形式的流水线拓扑结构。所以，使用弛豫超前技术可以得到一系列的流水线自适应滤波器。

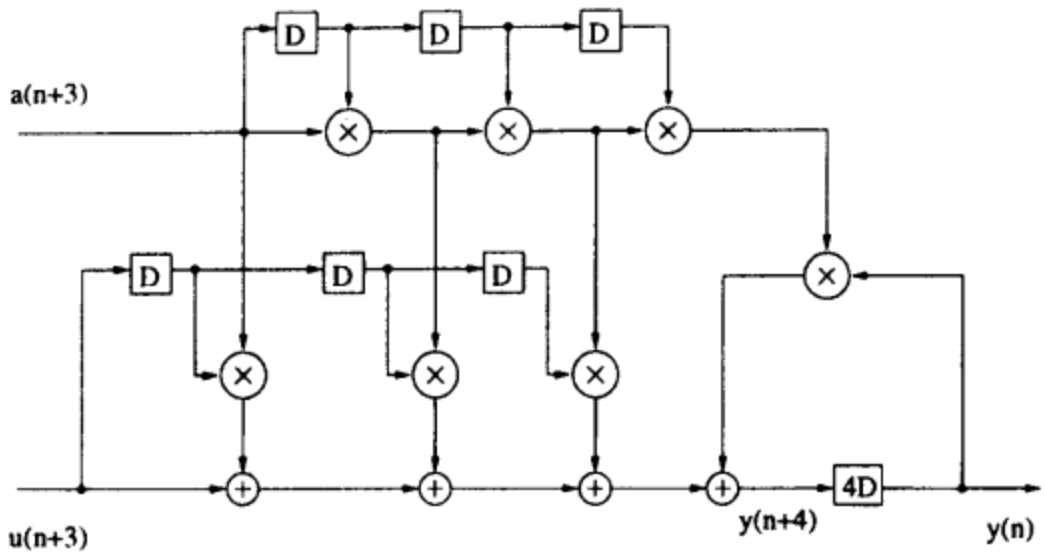


图10-26 一阶4超前流水线递归的信号流图

10.8.1.1 乘积近似

如果 $a(n)$ 的幅值接近于1，那么 $a(n)$ 就能被 $(1-\varepsilon(n))$ 代替，$\varepsilon(n)$ 接近于0。那么有如下的近似

$$\begin{aligned}\prod_{i=0}^{M-1} a(n+i) &\approx a(n+M-1)^M = (1-\varepsilon(n+M-1))^M \\ &= 1 - M\varepsilon(n+M-1) = 1 - M(1-a(n+M-1))\end{aligned} \tag{10-75}$$

更进一步，如果 $u(n+i)$ 接近于0，那么 $[\prod_{j=0}^{i-1} a(n+M-1-j)]\, u(n+M-1-i)$ 能用 $u(n+M-1-i)$ 近似。所以，公式(10-74)能被近似表示成

$$\begin{aligned}y(n+M) &= (1-M(1-a(n+M-1)))y(n) \\ &\quad + \sum_{i=0}^{M-1} u(n+M-1-i)\end{aligned} \tag{10-76}$$

353

所以，公式(10-76)是对公式(10-73)应用乘积近似后得到的M级弛豫超前的结果。对公式(10-73)中的一阶递归应用4级乘积近似流水线后的信号流图见图10-27。因为假设 $a(n)$ 近似等于1，那么 $\prod_{j=0}^{M-1} a(n+M-1-j)$ 也能用 $a(n+M-1)$ 来近似。可以得到公式(10-74)的另一种近似表示

$$\begin{aligned}y(n+M) &= a(n+M-1)y(n) \\ &\quad + \sum_{i=0}^{M-1} u(n+M-1-i)\end{aligned} \tag{10-77}$$

10.8.1.2 和近似

如果在M周期中输入 $u(n)$ 变化很慢，那么 $\prod_{j=0}^{M-1} a(n+M-1-j)$ 能被近似成 $Mu(n)$。这样可以得到:

$$y(n+M) = \prod_{i=0}^{M-1} a(n+M-1-i)y(n) + Mu(n) \tag{10-78}$$

如果 $u(n)$ 也接近于0，那么 $Mu(n)$ 可以用 $u(n)$ 来近似，得到

$$y(n+M)=\prod_{i=0}^{M-1}a(n+M-1-i)y(n)+u(n) \tag{10-79}$$

这样，公式(10-78)和公式(10-79)就是对公式(10-73)应用和近似后得到的M级弛豫超前的结果。对一阶递归应用4级和近似流水线后的信号流图如图10-28所示。

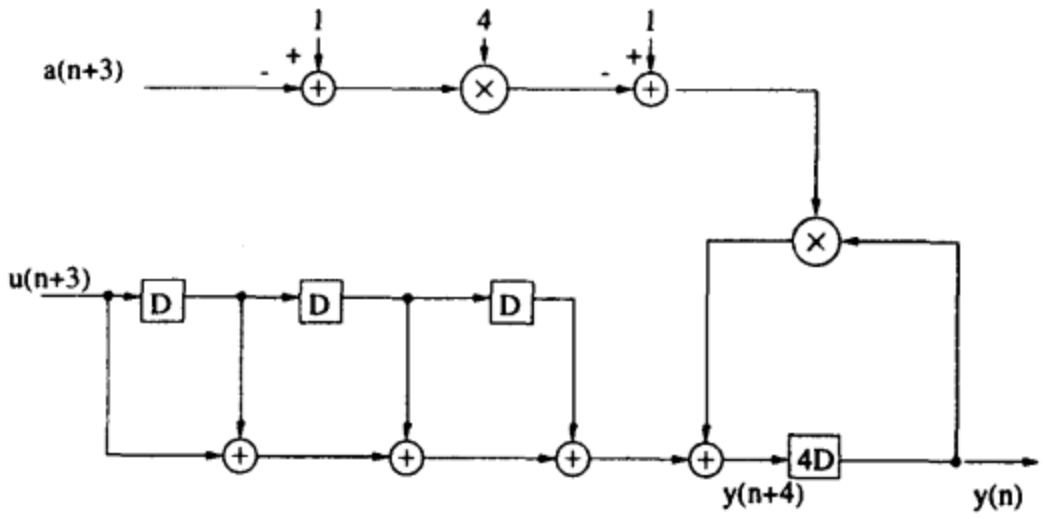


图10-27 4超前乘积近似一阶流水线递归

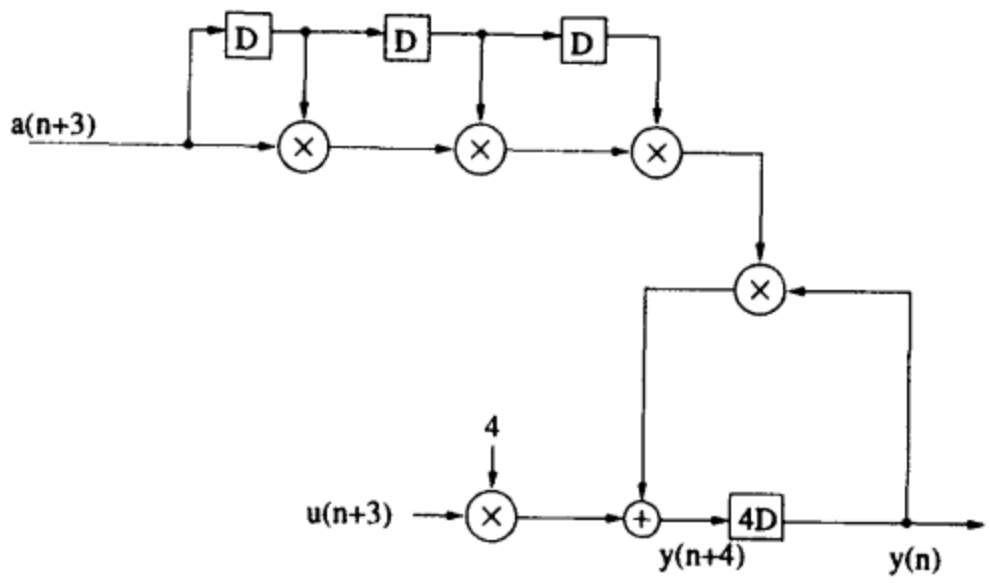


图10-28 一阶4级和弛豫超前流水线递归

10.8.1.3 延时近似

考虑递归表达式

$$y(n)=y(n-1)+a(n)u(n) \tag{10-80}$$

它的M级超前流水线表达式:

$$y(n)=y(n-M)+\sum_{i=0}^{M-1}a(n-i)u(n-i) \tag{10-81}$$

延时近似使用公式(10-81)中延时后的输入 $u(n-M')$ 和延时后的系数 $a(n-M')$。这种近似基于乘积 $a(n)\,u(n)$ 在 M' 个采样中基本不变这一假设[15]。一般地，这种假设对固定的或缓慢变化的乘积 $a(n)\,u(n)$ 是一种合理的近似。这样，公式(10-81)近似表示为:

$$y(n)=y(n-M)+\sum_{i=0}^{M-1}a(n-M'-i)u(n-M'-i) \tag{10-82}$$

这些就是弛豫超前所使用的技术。这些近似可以单独使用，也可以组合起来得到一系列丰富的结构。（这和超前技术形成鲜明的对比，使用超前技术在最终的流水线结构和原结构之间有一一对应关系。）每一种结构都有不同的收敛特性，它取决于所做近似的性质和这些近似在应用中的有效性。因为在输入输出特性已经被修改的情况下，平均输出保持不变，所以弛豫超前是一种随机性质的变换技术。

10.8.2 流水线LMS自适应滤波器

354~355

LMS算法可以描述为

$$e(n) = d(n) - \hat{d}(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{U}(n) \tag{10-83}$$

$$\mathbf{W}(n) = \mathbf{W}(n-1) + \mu e(n)\mathbf{U}(n) \tag{10-84}$$

在第1章中有这种自适应算法的公式推导和框图（参看1.2.4.1节）。在LMS结构中有两个递归环路：权重更新环路和误差反馈环路。为了得到流水线结构的自适应滤波器，从LMS等式开始，并尝试直接应用超前技术。可以得到：

$$\begin{aligned}\mathbf{W}(n) &= \mathbf{W}(n-1) + \mu e(n)\mathbf{U}(n) \\ &= \mathbf{W}(n-M_2) + \sum_{i=0}^{M_2-1} \mu e(n-i)\mathbf{U}(n-i)\end{aligned} \tag{10-85}$$

其中 M_2 是权重更新环路中延时的个数。这种变换在权重更新环路中产生 M_2 个锁存器，它们可以通过时序变换对加法操作实现所需要的流水线。但是，为了精确的应用超前技术，$e(n)$ 需要表示成 $\mathbf{W}(n-M_2)$ 的函数，并代入公式(10-85)，这将导致所得到的方程很复杂进而大大增加硬件负担。这种开销可以通过应用弛豫超前来降低。

延时近似在误差反馈环路（EFP）中引入 M_1 个延时。在公式(10-85)中应用延时近似，可以得到如下的方程：

$$\mathbf{W}(n) = \mathbf{W}(n-M_2) + \mu \sum_{i=0}^{M_2-1} e(n-M_1-i)\mathbf{U}(n-M_1-i) \tag{10-86}$$

这种求和的有效性建立在假设梯度估计 $e(n)\,\mathbf{U}(n)$ 在 M_1 个采样中不变的基础上。公式(10-86)的硬件开销是 $N(M_2-1)$ 个加法器，并且还可以应用和近似，去掉 M_2' 个项来进一步减少加法器的数目，其中 $M_2' < M_2$。这样得到

$$\mathbf{W}(n) = \mathbf{W}(n-M_2) + \sum_{i=0}^{M_2'-1} \mu e(n-M_1-i)\mathbf{U}(n-M_1-i) \tag{10-87}$$

对缓慢变化的乘积 $e(n)\,\mathbf{U}(n)$ 这种近似是可行的。所以，$e(n)$ 可以写成

$$\begin{aligned}e(n) &= d(n) - \mathbf{W}^T(n-1)\mathbf{U}(n) \\ &= d(n) - [\mathbf{W}^T(n-M_2-1) \\ &\quad + \mu \sum_{i=0}^{M_2'-1} e(n-M_1-i-1)\mathbf{U}(n-M_1-i-1)]\mathbf{U}(n)\end{aligned} \tag{10-88}$$

假设μ足够小，并用 $\mathbf{W}(n-M_2)$ 代替 $\mathbf{W}(n-M_2-1)$，公式(10-88)可以近似写成 356

$$e(n) = d(n) - \mathbf{W}^T(n-M_2)\mathbf{U}(n) \tag{10-89}$$

公式(10-87)和公式(10-89)完整地描述了流水线LMS滤波器（PIPLMS），它的硬件开销是 $N(M_2-1)$ 个加法器[16]。图10-29是这种PIPLMS结构的系统框图。流水线LMS滤波器还可以用来设计流水线自适应决策反馈均衡器[17]。

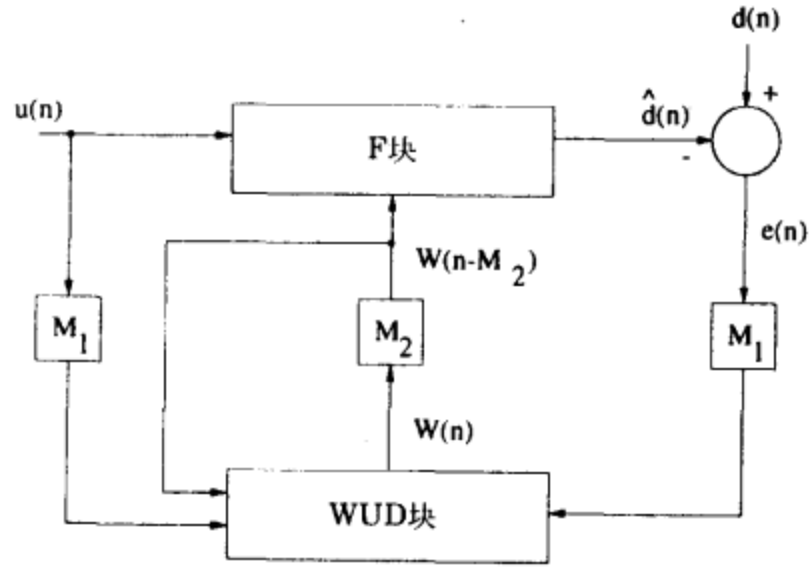


图10-29 PIPLMS结构的系统框图

10.8.3 用流水线实现随机梯度格型结构

10.8.3.1 流水线随机梯度格型结构（PIPSGLA）

可以用如下的等式来完整地描述随机梯度自适应算法：

$$\begin{aligned} k_m(n+1) = \ & [1-\beta_m(e_f^2(n|m-1)+e_b^2(n-1|m-1))]k_m(n) \\ & +2\beta_m e_f(n|m-1)e_b(n-1|m-1) \end{aligned} \tag{10-90}$$

$$\begin{aligned} S(n+1|m-1) = \ & (1-\beta)S(n|m-1) \\ & +e_f^2(n|m-1)+e_b^2(n-1|m-1) \end{aligned} \tag{10-91}$$

$$\beta_m = \frac{1}{S(n|m-1)} \tag{10-92}$$

$$e_f(n|m) = e_f(n|m-1) - k_m(n)e_b(n-1|m-1) \tag{10-93}$$ 357

$$e_b(n|m) = e_b(n-1|m-1) - k_m(n)e_f(n|m-1) \tag{10-94}$$

公式(10-90)~(10-92)和公式(10-93)~(10-94)分别叫做自适应方程和阶更新方程。随机梯度格型（或网格）结构（SGLA）的框图见第1章图1-4。有关详细的推导，读者可以参看第1.2.4.2节。SGLA包含两个递归环路，分别计算 k_m 和 S，计算时间是 T_m+T_a。所以，SGLA的流水线出现在两个步骤中。第一步包含级间流水线，延时单元放置在沿网格各级的前馈割集之间。第二步是环路流水线，应用弛豫超前把自适应滤波器中的递归环路实现流水线。因为网格的各个级是以一种非递归的方式连接在一起的，因此级间流水线是可忽略的并且不会

改变输入输出特性。所以，忽略这一步，对SGLA中的这两个递归环路应用弛豫超前可以得到PIPSGLA结构。

可以用从公式(10-79)~(10-91)近似得到M级弛豫超前的公式，

$$\begin{aligned} S(n+M|m-1) &= (1-\beta)^M S(n|m-1) \\ &\quad +e_f^2(n|m-1)+e_b^2(n|m-1) \end{aligned} \tag{10-95}$$

该公式是用来递归计算输入能量的，其中 $(1-\beta)^M$ 可以预先计算出来。类似地，用和近似对公式(10-78)和公式(10-90)应用M级弛豫超前可以得到

$$\begin{aligned} k_m(n+M) = &\ [1-\beta_m(n)(e_f^2(n|m-1)+e_b^2(n-1|m-1))]^M k_m(n) \\ &+2M\beta_m(n)e_f(n|m-1)e_f(n-1|m-1) \end{aligned} \tag{10-96}$$

可以证明，第一项中方括号内的部分在收敛的最后阶段是接近于1的。所以公式(10-75)的乘积近似可以应用到公式(10-96)，得到

$$\begin{aligned} k_m(n+M) = &\ [1-M\beta_m(n)(e_f^2(n|m-1)+e_b^2(n-1|m-1))]k_m(n) \\ &+2M\beta_m(n)e_f(n|m-1)e_f(n-1|m-1) \end{aligned} \tag{10-97}$$

假设第m级网格的输入是 $e_f(n\,|\,m-1)$ 和 $e_b(n\,|\,m-1)$，下面的等式描述了被称作PIPSGLA1的第一种PIPSGLA结构的时序更新方程。

PIPSGLA1:

$$\begin{aligned} k_m(n+M) = &\ [1-M\beta_m(n)(e_f^2(n|m-1)+e_b^2(n-1|m-1))]k_m(n) \\ &+2M\beta_m(n)e_f(n|m-1)e_b(n-1|m-1) \end{aligned} \tag{10-98}$$

358

$$\begin{aligned} S(n+M|m-1) = &\ (1-\beta)^M S(n|m-1) \\ &+e_f^2(n|m-1)+e_b^2(n-1|m-1) \end{aligned} \tag{10-99}$$

图10-30是PIPSGLA1的结构[18]。

另一种流水线结构PIPSGLA2可以简单通过在公式(10-90)和公式(10-91)的递归环路中引入M个锁存器得到。参见公式(10-90)和公式(10-91)，它们都应用由公式(10-79)定义的和近似以及公式(10-77)定义的乘积近似。PIPSGLA2的两个时序更新方程如下:

PIPSGLA2:

$$\begin{aligned} k_m(n+M) = &\ [1-\beta_m(n)(e_f^2(n|m-1)+e_b^2(n-1|m-1))]k_m(n) \\ &+2\beta_m(n)e_f(n|m-1)e_b(n-1|m-1) \end{aligned} \tag{10-100}$$

$$\begin{aligned} S(n+M|m-1) = &\ (1-\beta)S(n|m-1) \\ &+e_f^2(n|m-1)+e_b^2(n-1|m-1) \end{aligned} \tag{10-101}$$

10.8.3.2 PIPSGLA的收敛性分析

前面提到过，对由弛豫超前技术得到的流水线结构需要进行收敛性分析。这一节简要分析PIPSGLA1和PIPSGLA2的收敛性。假设一个网格结构的级的输入是固定的，我们对它进行分析。同时还假设在时刻 n_1 的反射系数 $k_m(n_1)$ 是独立于 $e_f(n\,|\,m-1)$ 和 $e_b(n\,|\,m-1)$ 的，对所有 $n<n_1$ 都成立。这是独立性假设。为了求高阶统计期望值，假设 $e_f(n\,|\,m-1)$ 和 $e_b(n\,|\,m-1)$

都服从联合高斯分布。基于以上假设，用下列的指标分析、比较各种结构的自适应特性:

- 保持收敛/稳定的 β 的范围
- 收敛率，τ
- 失调误差

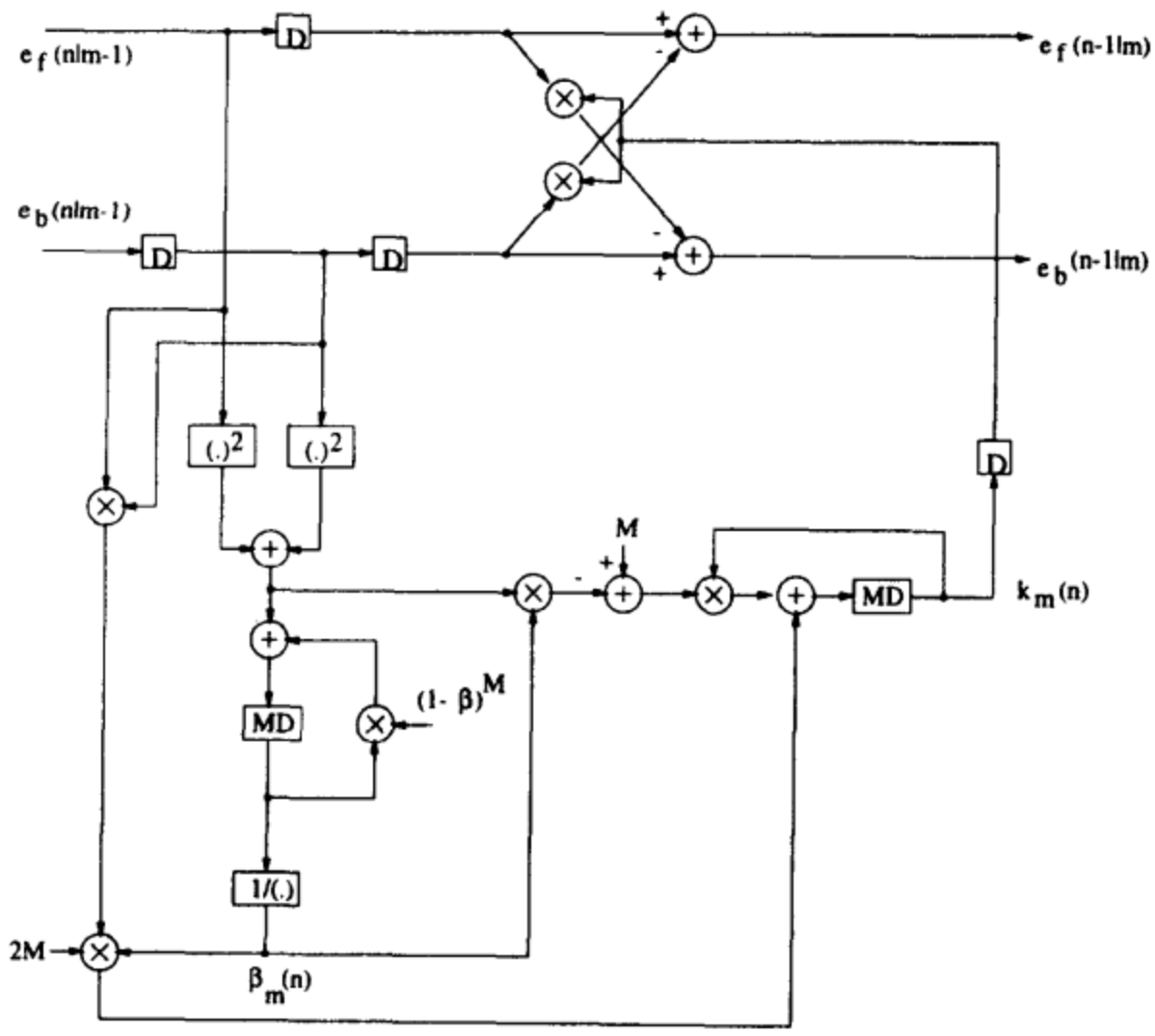


图10-30 PIPSGLA1结构

下面推导PIPSGLA1中这些指标的表达式。SGLA和PIPSGLA2中的表达式可以类似地推导出来。注意由于我们的假设，我们只是想得到对各种结构的一个比较分析，而不是要得出一种它们性能的绝对的评价标准。

保持收敛/稳定的 β 的范围 求PIPSGLA1中公式(10-98)的数学期望，得到

$$E[k_m(n+M)] = [1 - M\frac{E(e_f^2+e_b^2)}{E(S)}]E(k_m) + \frac{2ME(e_f e_b)}{E(S)} \tag{10-102}$$

其中，$E[\cdot]$表示求期望运算。为了找到使误差最小的最优值 $k_{m,\,opt}$，考虑前向和后向误差更新方程，

$$\begin{aligned} e_f(n|m) &= e_f(n|m-1) - k_m(n)e_b(n-1|m-1) \\ e_b(n|m) &= e_b(n-1|m-1) - k_m(n)e_f(n|m-1) \end{aligned} \tag{10-103}$$

从这往后，为了简单起见，去掉了时间序号和级的序号。注意左侧的表达式对应级m，而右侧的表达式对应级$m-1$。把公式(10-103)中的两个等式平方后相加，得到

$$e_f^2 + e_b^2 = e_f^2 + e_b^2 - 4e_f e_b k_m + k_m^2(e_f^2 + e_b^2) \tag{10-104}$$

现在，可以计算 $E\left[e_f^2(n|m)+e_b^2(n|m)\right]$ 对 $k_m(n)$ 的梯度。让这个梯度等于0，可以得到 k_m 的最优值（记做 $k_{m,opt}$）：

359 ~ 360

$$\begin{aligned}\frac{\partial E(e_f^2+e_b^2)}{\partial k_m} &= -4E(e_f e_b)+2k_m E[e_f^2+e_b^2]=0 \\ \Rightarrow \qquad & k_{m,opt}=\frac{2E(e_f e_b)}{E(e_f^2+e_b^2)}\end{aligned} \tag{10-105}$$

求公式(10-99)的期望，得到

$$\begin{aligned}E[S] &= (1-\beta)^M E[S]+E(e_f^2+e_b^2) \\ \frac{E(e_f^2+e_b^2)}{E(S)} &= 1-(1-\beta)^M \\ &= (1-(1-M\beta)) \quad \text{如果 } \beta \text{ 很小} \\ &= M\beta=\gamma.\end{aligned} \tag{10-106}$$

把上式两边同时乘以 $k_{m,opt}$，得到

$$\begin{aligned}\frac{E(e_f^2+e_b^2)}{E(S)}k_{m,opt} &= \gamma k_{m,opt} \\ &= \frac{2E(e_f e_b)}{E(S)}.\end{aligned} \tag{10-107}$$

在公式(10-102)中代入公式(10-106)和公式(10-107)，得到

$$\begin{aligned}E[k_m(n+M)] &= (1-M\gamma)E(k_m)+M\gamma k_{m,opt} \\ E[k_m(n+M)-k_{m,opt}] &= (1-M\gamma)E(k_m)+(M\gamma-1)k_{m,opt} \\ &= (1-M\gamma)E[k_m(n)-k_{m,opt}].\end{aligned} \tag{10-108}$$

使 $v(n)=E[k_m(n)-k_{m,opt}]$，重写公式(10-108)

$$v(n+M) = (1-M\gamma)v(n). \tag{10-109}$$

代入 $n=0$ 和 $n=M$，

$$\begin{aligned}v(M) &= (1-M\gamma)v(0) \quad \text{当 } n=0 \\ v(2M) &= (1-M\gamma)^2 v(0) \quad \text{当 } n=M\end{aligned} \tag{10-110}$$

重复迭代，得到

$$v(kM) = (1-M\gamma)^k v(0), \quad \text{当 } n=kM \tag{10-111}$$

要使公式(10-111)的左侧收敛到0，充分条件是

361

$$\begin{aligned}|1-M\gamma| &< 1 \\ -1 < 1-M\gamma &< 1 \\ -2 < -M\gamma \quad \text{且} \quad -M\gamma &< 0 \\ 0 < M\gamma &< 2 \\ 0 < M^2\beta &< 2 \\ 0 < \beta &< \frac{2}{M^2}\end{aligned} \tag{10-112}$$

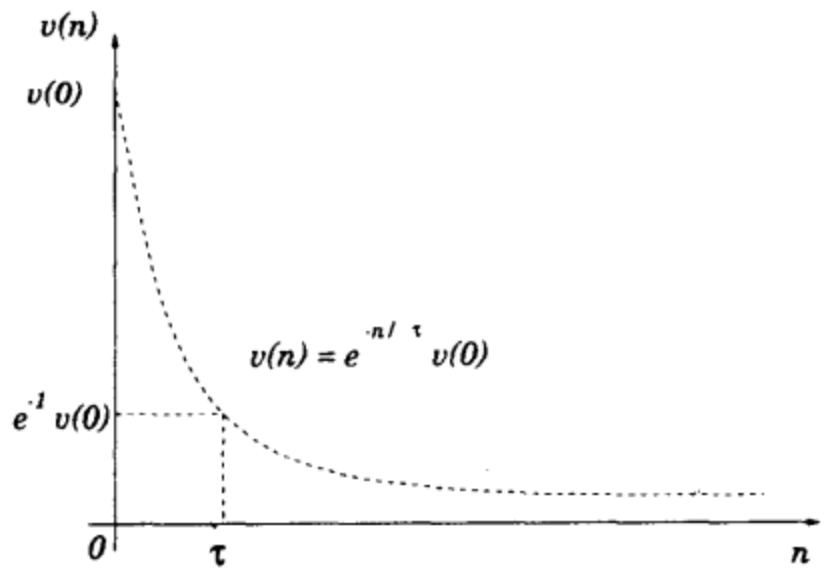


图10-31 指数衰减的离散序列

收敛时间常数 在公式(10-111)中代入 $n=kM$，得到

$$\begin{aligned} v(n) &= (1-M\gamma)^{n/M}v(0) \\ &= [(1-M\gamma)^{1/M}]^n v(0) \end{aligned} \tag{10-113}$$

考虑图10-31所示的指数衰减曲线。当 $n=\tau$ 时，$v(\tau)=e^{-1}v(0)$。所以，使公式(10-113)的右侧等于以时间常数 τ 衰减的指数函数，得到

$$\begin{aligned} \frac{1}{\mathrm{e}}v(0) &= [(1-M\gamma)^{1/M}]^{\tau}v(0) \\ \ln\frac{1}{\mathrm{e}} &= \ln[(1-M\gamma)^{1/M}]^{\tau} \\ \tau &= \frac{-\ln \mathrm{e}}{\ln(1-M\gamma)^{1/M}} \\ &= \frac{-1}{\ln(1-M\gamma)^{1/M}}. \end{aligned} \tag{10-114}$$

使用近似

$$(1-M\gamma)^{1/M} \approx 1-(\frac{1}{M})M\gamma \approx 1-\gamma \tag{10-115}$$ 362

和

$$\ln(1-x) \approx -x-\frac{x^2}{2}-\frac{x^3}{3}-\cdots \approx -x \tag{10-116}$$

公式(10-114)可以简化成

$$\begin{aligned} \tau &= \frac{-1}{\ln(1-M\gamma)^{1/M}} \approx \frac{-1}{-\gamma} \\ &= \frac{1}{\gamma} = \frac{1}{M\beta} \end{aligned} \tag{10-117}$$

所以，当M增加时，收敛的速度也加快。

失调误差 一个自适应算法的自适应精确度是用它的失调误差定义的，失调误差定义如下:

$$\mathcal{M} = \frac{E(J(\infty)) - J_{min}}{J_{min}} = \frac{E(J_{ex}(\infty))}{J_{min}} \tag{10-118}$$

其中 $J(n)$ 是在时刻 n 的均方误差，定义如下:

$$J(n) = e_f^2(n|m) + e_b^2(n|m) \tag{10-119}$$

$E(J(n))$ 是它的平均值，J_{ex} 表示剩余误差。J_{min} 指最小均方误差，如果反射系数 $k_m(n)$ 等于它的最优值 $k_{m,opt}$，就可以求出 J_{min}。

为了得到SGLA、PIPSGLA1和PIPSGLA2的失调误差的近似表达式，要做某种简化后的近似。特别地，使用到了本节开始时所做的独立性假设和联合高斯分布假设。另外，让

$$\sigma_e^2 = E[e_f^2(n|m-1)] = E[e_b^2(n|m-1)] \tag{10-120}$$

表示输入信号的功率。接下来，先推导 $E(J(n))$ 或 $E(J_{ex}(n))$ 的表达式，然后推导一些中间表达式，最后推导PIPSGLA1的失调误差的表达式。

我们知道，如果 X_1、X_2、X_3 和 X_4 是服从共同的高斯分布的变量，那么

$$\begin{aligned} E[X_1X_2X_3X_4] &= E[X_1X_2]E[X_3X_4] \\ &\quad + E[X_1X_3]E[X_2X_4] + E[X_1X_4]E[X_2X_3] \end{aligned} \tag{10-121}$$

使用公式(10-122)和公式(10-105)中 $k_{m,opt}$ 的表达式，非常容易得到下面的公式:

363

$$E[e_f(n|m-1)e_b(n-1|m-1)] = k_{m,opt}\sigma_e^2 \tag{10-122}$$

$$E[e_f^2(n|m-1)e_b^2(n-1|m-1)] = (1+2k_{m,opt}^2)\sigma_e^4 \tag{10-123}$$

$$E[e_f^3(n|m-1)e_b(n-1|m-1) + e_f(n|m-1)e_b^3(n-1|m-1)] = 6k_{m,opt}\sigma_e^4 \tag{10-124}$$

$$E[(e_f^2(n|m-1) + e_b^2(n-1|m-1))^2] = (8+4k_{m,opt}^2)\sigma_e^4 \tag{10-125}$$

从公式(10-106)可知

$$\frac{E[e_f^2 + e_b^2]}{E[S]} = \gamma \tag{10-126}$$

由公式(10-105)

$$k_{m,opt} = \frac{2E[e_fe_b]}{E[e_f^2 + e_b^2]} \tag{10-127}$$

所以，可以得到

$$\frac{E[e_fe_b]}{E[S]} = \frac{\gamma}{2}k_{m,opt} \tag{10-128}$$

最后，需要推导出 $E[S^2(n|m-1)]$的表达式。把公式(10-99)进行平方，并求其期望，得到

$$E[S^2] = (1-\beta)^{2M}E[S^2] + 2(1-\beta)^M E[S(e_f^2 + e_b^2)] + E[(e_f^2 + e_b^2)^2] \tag{10-129}$$

公式(10-99)两边同时乘以 $S(n\,|\,m-1)$，并求其期望，得到

$$E[S^2] = (1-\beta)^M E[S^2] + E[S(e_f^2 + e_b^2)] \tag{10-130}$$

即

$$E[S(e_f^2 + e_b^2)] = \quad [1-(1-\beta)^M]E[S^2] \approx \gamma E[S^2] \tag{10-131}$$

在公式(10-129)中代入公式(10-131)和公式(10-125)，可以得到想要的表达式

$$E[S^2] = \frac{(8+4k_{m,opt}^2)\sigma_e^4}{\gamma^2} \tag{10-132}$$

到目前为止，我们已经得到了所有的中间表达式。下面推导 $E[J(n)]$ 的表达式。注意

$$\begin{aligned} J(n) &= e_f^2(n|m) + e_b^2(n|m) \\ &= (e_f^2(n|m-1) + e_b^2(n-1|m-1))(1+k_m^2(n)) \\ &\quad -4k_m(n)e_f(n|m-1)e_b(n-1|m-1) \end{aligned} \tag{10-133}$$

364

所以

$$\begin{aligned} E[J(n)] &= E[e_f^2(n|m-1) + e_b^2(n-1|m-1)](1+E[k_m^2(n)]) \\ &\quad -2E[k_m(n)]\cdot 2E[e_f(n|m-1)e_b(n-1|m-1)] \\ &= (1+E[k_m^2(n)] - 2E[k_m(n)]k_{m,opt}) \\ &\quad \cdot E[e_f^2(n|m-1) + e_b^2(n-1|m-1)] \\ &= (1+(E[k_m(n)]-k_{m,opt})^2 - k_{m,opt}^2 + E[k_m^2(n)] - E^2[k_m(n)]) \\ &\quad \cdot E[e_f^2(n|m-1) + e_b^2(n-1|m-1)] \\ &= (1+v^2(n) - k_{m,opt}^2 + var(k_m(n))) \\ &\quad \cdot E[e_f^2(n|m-1) + e_b^2(n-1|m-1)] \end{aligned} \tag{10-134}$$

考虑当 $n\to\infty$ 时的极限（$v^2(n)$趋向于0），得到

$$\begin{aligned} E[J(\infty)] &= (1-k_{m,opt}^2 + var(k_m(\infty)))\cdot 2\sigma_e^2 \\ &= (1-k_{m,opt}^2)\cdot 2\sigma_e^2 + var(k_m(\infty))\cdot 2\sigma_e^2 \\ &= J_{min} + E[J_{ex}(\infty)] \end{aligned} \tag{10-135}$$

其中 J_{min} 表示最小误差，J_{ex} 表示剩余误差。所以，网格结构的一个级的失调误差 $\mathcal{M}$ 可以表示成

$$\mathcal{M} = \frac{E[J_{ex}(\infty)]}{J_{min}} = \frac{var(k_m(\infty))}{(1-k_{m,opt}^2)} \tag{10-136}$$

所以，剩下的工作就是计算反射系数的稳态变化 $var(k_m(\infty))$。公式(10-98)两边同时减去 $k_{m,opt}$ 并求平方，得到

$$\begin{aligned} (k_m(n+M)-k_{m,opt})^2 &= (1-\frac{M(e_f^2+e_b^2)}{S})^2 k_m^2(n) + \frac{4M^2e_f^2e_b^2}{S^2} \\ &\quad + k_{m,opt}^2 + \frac{4Me_fe_b}{S}(1-\frac{M(e_f^2+e_b^2)}{S})k_m(n) \\ &\quad -2k_{m,opt}(1-\frac{M(e_f^2+e_b^2)}{S})k_m(n) - \frac{4Me_fe_b}{S}k_{m,opt} \end{aligned} \tag{10-137}$$

在下面的叙述中，所有的期望在 $n\to\infty$ 时都是收敛的。注意当 $n\to\infty$ 时，$E[k_m(n)]=k_{m,opt}$。求公式(10-137)的期望，并使用公式(10-122)~(10-132)进行化简，得到

365

$$\begin{aligned} var(k_m(\infty)) &= E[(k_m(n+M)-k_{m,opt})^2] \\ &= (1-2M\gamma+M^2\gamma^2)E[k_m^2(n)]-(1-2M\gamma)k_{m,opt}^2 \\ &\quad +\frac{M^2\gamma^2(1-4k_{m,opt}^2)}{2+k_{m,opt}^2} \\ &= (1-2M\gamma+M^2\gamma^2)(E[k_m^2(n)]-k_{m,opt}^2) \\ &\quad M^2\gamma^2k_{m,opt}^2+\frac{M^2\gamma^2(1-4k_{m,opt}^2)}{2+k_{m,opt}^2} \end{aligned} \tag{10-138}$$

注意

$$var(k_m(\infty))=E[(k_m(n+M)-k_{m,opt})^2]=E[k_m^2(n)]-k_{m,opt}^2 \tag{10-139}$$

这样有

$$var(k_m(\infty)) = \frac{M\gamma(1-k_{m,opt}^2)^2}{(2-M\gamma)(2+k_{m,opt}^2)}. \tag{10-140}$$

PIPSGLA1的失调误差由公式(10-141)给出

$$\mathcal{M} = \frac{M\gamma(1-k_{m,opt}^2)}{(2-M\gamma)(2+k_{m,opt}^2)}. \tag{10-141}$$

表10-1 不同SGLA结构失调误差的比较

	τ	失调误差
SGLA	$\frac{1}{\beta}$	$\frac{\beta(1-k_{m,opt}^2)}{(2-\beta)(2+k_{m,opt}^2)}$
PIPSGLA1	$\frac{1}{M\beta}$	$\frac{M^2\beta(1-k_{m,opt})}{(2-M^2\beta)(2+k_{m,opt}^2)}$
PIPSGLA2	$\frac{M}{\beta}$	$\frac{\beta(1-k_{m,opt}^2)}{(2-\beta)(2+k_{m,opt}^2)}$

表10-1给出了不同结构收敛性的比较。为了得到更精确的比较，假设 β 比1小的多，并用2来近似 $(2-\beta)$ 和 $(2-M^2\beta)$。这样得出表10-2中新的比较结果，从中可以清楚地看到M级流水线格型自适应滤波器的失调误差和时间常数都减小了M倍。另一方面，弛豫超前流水线格型滤波器提供了失调误差和收敛时间常数的折中方案。不同类型的近似技术在这两个参数间提供不同的折中方案。

表10-2 不同SGLA结构失调误差的比较

	τ	失调误差	$\tau\times$失调误差
SGLA	$\frac{1}{\beta}$	$\frac{\beta}{2}\frac{1-k_{m,opt}^2}{2+k_{m,opt}^2}$	$\frac{1}{2}\frac{1-k_{m,opt}^2}{2+k_{m,opt}^2}$
PIPSGLA1	$\frac{1}{M\beta}$	$\frac{M^2\beta}{2}\frac{1-k_{m,opt}^2}{2+k_{m,opt}^2}$	$\frac{M}{2}\frac{1-k_{m,opt}^2}{2+k_{m,opt}^2}$
PIPSGLA2	$\frac{M}{\beta}$	$\frac{\beta}{2}\frac{1-k_{m,opt}^2}{2+k_{m,opt}^2}$	$\frac{M}{2}\frac{1-k_{m,opt}^2}{2+k_{m,opt}^2}$

10.9 结论

在这一章里，我们应用聚类超前和离散超前流水线的方法在一阶和高阶IIR数字滤波器中实现了流水线。这一章还讲述了并行IIR滤波器的设计。我们还使用了增量块处理技术来设计只有很小面积开销的并行结构高阶IIR滤波器。如何在格型递归数字滤波器中实现流水线参看第12章。应用流水线和并行处理可以提高采样率或在同样的速度下降低供电电压以减少功耗。超前技术可以用于动态规划问题[19]（参看习题25）、Viterbi 解码器[20]和量化器环路[21]的并行实现。使用包括和近似、乘积近似、延时近似等不同形式的弛豫超前技术得出了LMS和随机梯度格型自适应数字滤波器的一系列流水线拓扑结构。[22]介绍了如何在其他一些类型的自适应数字滤波器中实现流水线。[23]讲述了如何使用伸缩正切旋转（scaled tangent rotation、STAR）在递归最小平方（RLS）自适应滤波器中实现流水线；[24]讲述了使用超前湮没重排序（annihilation-reordering look-ahead）的方法在RLS自适应滤波器中实现流水线。

10.10 习题

1. 考虑直接形式3阶IIR滤波器

$$H(z)=\frac{0.5+0.25z^{-1}}{1-\frac{3}{2}z^{-1}+\frac{11}{16}z^{-2}-\frac{3}{32}z^{-3}} \tag{10-142}$$

使用聚类超前流水线在每一个乘加操作中实现 (a)2级流水线；(b)4级流水线。两种情况下，新系统的极点是什么？这些流水线滤波器稳定吗？

2. 在习题1的IIR滤波器中应用离散超前技术使每个乘加操作实现2级流水线。画出完整的结构，并在适当的位置放置锁存器来实现流水线（需要把乘法器分解为两部分）。这个系统的极点是什么？ 366～367

3. 考虑二阶IIR滤波器，传输函数

$$H(z)=\frac{0.13021(1+z^{-1})}{1-\frac{13}{24}z^{-1}+\frac{1}{16}z^{-2}} \tag{10-143}$$

使用离散超前变换得出一个3级流水线传输函数。指出这个流水线滤波器的零点和极点。

4. 考虑IIR数字滤波器的传输函数

$$H(z)=\frac{1}{1-\frac{4}{3}z^{-1}+\frac{5}{12}z^{-2}} \tag{10-144}$$

使用(a)聚类超前(b)离散超前和分解方法得出一个等效的4级流水线传输函数。

5. 考虑二阶全极点递归滤波器

$$H(z)=\frac{1}{1-0.6z^{-1}+0.25z^{-2}} \tag{10-145}$$

这个滤波器需要应用离散超前和分解技术实现9级流水线。使用3 × 3的分解推导出传输函数。使用极点-零点图清楚地画出每一部分的零点和滤波器的极点。

（提示：$\frac{1}{1-a}=\frac{(1+a+a^2)(1+a^3+a^6)}{1-a^9}$）

6. 应用离散超前和分解技术把习题5中的滤波器实现6级流水线。推导实现流水线后的传输函数和对应于3×2分解、2×3分解的结构。画出每一部分零点的位置。
7. 二阶IIR数字滤波器的传输函数是

$$H(z)=\frac{1}{1-2r\cos\theta z^{-1}+r^2z^{-2}} \tag{10-146}$$

可以用下面的两个传输函数实现：

$$\begin{aligned} H_1(z) &= \underbrace{\frac{1+2r\cos\theta z^{-1}+r^2z^{-2}}{1-2r^2\cos 2\theta z^{-2}+r^4z^{-4}}}_{\text{(2-级流水线)}} \\ H_2(z) &= \underbrace{\frac{(1+2r\cos\theta z^{-1}+r^2z^{-2})(1+2r^2\cos 2\theta z^{-2}+r^4z^{-4}}{1-2r^4\cos 4\theta z^{-4}+r^8z^{-8}}}_{\text{(4-级流水线)}} \end{aligned} \tag{10-147}$$

368

我们的目的是验证在采样周期是一个已知常数的情况下，哪一种实现方式面积乘以功率的积（$area\times power$）更小？$H_1(z)$ 和 $H_2(z)$ 的结构都可以在低电压下工作。假设用5 V电压给 $H(z)$ 供电，阈值电压是0.6 V。假设加法器所需要的面积和它的电容相对于乘法器的面积和电容来说都可以忽略。计算 $H_1(z)$ 和 $H_2(z)$ 对 $H(z)$ 归一化的 $area\times power^2$ 积。哪一种实现方式的 $area\times power^2$ 比较小？

8. 考虑传输函数是 $1/(1-az^{-1})$ 的单极点IIR数字滤波器的实现。这个滤波器要求的采样周期是乘加运算时间的1/3，可以用三级流水线或3个块运算来实现。哪一种实现方式的功耗更低？
9. 同习题8，只是条件改为采样周期是乘加时间的四分之一。
10. 传输函数是 $1/(1-az^{-1})$ 的单极点IIR滤波器需要以顺序实现方式速度的15倍实现。可有多种方式，比如15级流水线、15级块运算、3级流水线、5级块运算或5级流水线、3级块运算。假设顺序系统和并行系统都工作在相同的采样率下。当不考虑功耗时，哪一种可能的实现方式有最小的面积开销？当不考虑面积时，哪一种可能的实现方式有最小的功耗？哪一种有最小的 $area\times power$ 积？
11. 考虑传输函数

$$H(z)=\frac{(1+z^{-1})^2}{1-\frac{3}{2\sqrt{2}}z^{-1}+\frac{9}{16}z^{-2}} \tag{10-148}$$

极点位于 $3/4e^{\pm j\pi/4}$。求出等效的4级流水线IIR滤波器。求出等效的4并行IIR滤波器。

12. 为习题4中的滤波器设计块大小为4的块滤波器结构。用系统的方法求出系统的极点。
13. 考虑直接形式三阶IIR滤波器

$$H(z)=\frac{1}{1-\frac{3}{2}z^{-1}+\frac{11}{16}z^{-2}-\frac{3}{32}z^{-3}} \tag{10-149}$$

画出当块大小为2时，块处理结构。采用状态机的方法计算块滤波器的极点。

14. 当块大小是4时，重复计算习题13。

15. 4阶IIR数字滤波器的极点位于 0.6、$0.3 \pm j0.4$ 和 -0.8。应用离散超前流水线和/或块处理技术得到的下列滤波器的极点在哪里？

(a) 3级流水线

(b) 4级并行处理

(c) 2级流水线，3级并行处理

(d) 4级流水线，2级并行处理

16. 计算习题12中的4并行IIR滤波器相对于顺序实现方式降低的功耗。原系统中的初始供电电压是3.3 V，阈值电压是0.45 V。假设块数是变化的，使 $area \times power$ 最小的块数是多少？假设供电电压的下限是1.0 V。

17. 考虑根据如下传输函数设计一个3并行IIR滤波器：

$$\frac{Y(z)}{U(z)} = H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} \tag{10-150}$$

让 $f(n) = b_0 u(n) + b_1 u(n-1) + b_2 u(n-2)$。这样，需要计算 $f(3k)$、$f(3k+1)$ 和 $f(3k+2)$。基于第9章介绍的快速并行FIR滤波方法，可以用更少的乘法器计算出这3个输出。用15个乘法器设计这种结构。计算所得到的结构中加法器的个数。

18. 用6并行的方法实现习题17中的二阶IIR数字滤波器，所需要的乘法操作的个数最少是多少？使用快速并行FIR滤波器和累计输出结果强度递减的方法，可以减少乘法操作的个数。解释怎样设计使用这些最小数目乘法操作的滤波器。不必完整地设计出这个滤波器。（提示：使用31个乘法器进行设计。）

19. 用最少的乘法操作实现图10-32中的系统。

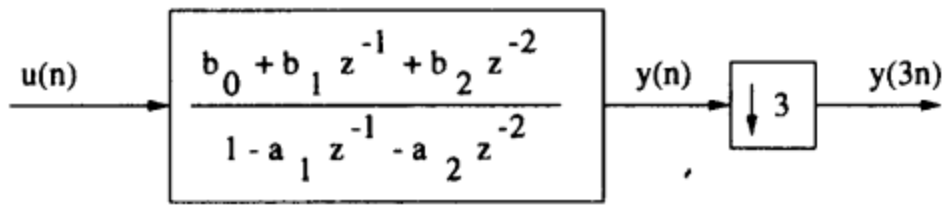


图10-32 习题19的图

20. 图10-33是判断系统类型的题目，考虑一个线性时不变离散时间装置，它的“未知”系统单位采样响应是$h(n)$=[0.3208，0.1535，0.2133，0.1964，0.1859，0.1694，0.1579，0.1457，0.1367，0.1281]。输入 $u(n)$ 是一个独立同分布（IID）随机序列，它取 +1或 −1的概率相等。应用LMS算法找到未知的滤波器系数。在以下的情况中，作为时间的函数，以dB为单位画出平方后的误差输出：

(a) 正常LMS，10抽头，权重更新系数 μ=0.1。迭代计算1000次。

(b) 流水线LMS滤波器，(i) M_1=10的延时近似，(ii) M_1=10，M_2=5的和，延时近似。使用和(a)中相同的 μ 值，迭代5000次。

(c) 为了验证（M_2'−1）个加法器的效果，使用 M_1=20、M_2=6 和 μ=0.003 并比较 M_2'=1 和 M_2'=5 时的结果。

(d) 定义

$$R = E[\mathbf{U}(n)\mathbf{U}^T(n)] \tag{10-151}$$

$$M_1 = KM_2 \tag{10-152}$$

$$\lambda_{rms}^2 = \frac{\sum_{i=1}^{N}\lambda_i^2}{N} \tag{10-153}$$

$$\lambda_{av} = \frac{\sum_{i=1} N\lambda_i}{N} = \sigma^2 \tag{10-154}$$

$$\alpha = \frac{\lambda_{rms}^2}{\lambda_{av}^2} \tag{10-155}$$

$$v = \frac{E[(u(n))^4]}{(E[(u(n))^2])^2} \tag{10-156}$$

$$P = N + v - 1 \tag{10-157}$$

$$b = \mu\sigma^2 \tag{10-158}$$

并考虑PIPLMS滤波器收敛时 μ 的上下限

$$0 < \mu < \frac{\alpha P + 2K - \sqrt{(\alpha P + 2K)^2 - 8K(K+1)}}{2K(K+1)\sigma^2} \tag{10-159}$$

使用上述公式计算滤波器收敛时 μ 的取值范围。设 M_1=20，M_2=M_2'=5，μ 取 0.01、0.05、0.1进行计算。

图10-33 习题20中判断系统类型的示意图

21. 证明表10-1中SSGLA和PIPSGLA2结构的失调误差。

22. 考虑下面随机梯度格型滤波器的2个状态更新方程。通过在原格型滤波器中应用不同的和、乘积近似得到了这两个方程。对每一个表达式，得出稳定性对 β 的限制，并得出时间常数和失调参数。在这些方程中，去掉了级序号m。其他格型滤波器方程保持和顺序滤波器中相同。

$$\begin{aligned} k(n+M) = \quad & [1-\beta(n)(e_f^2(n)+e_b^2(n-1))]k(n) \\ & +2\beta(n)e_f(n)e_b(n-1) \\ S(n+M) = \quad & (1-\beta)^M S(n) + M(e_f^2(n)+e_b^2(n-1)) \end{aligned} \tag{10-160}$$

23. 用下式重复计算习题22。

$$\begin{aligned} k(n+M) = \quad & [1-\beta(n)(e_f^2(n)+e_b^2(n-1))]k(n) \\ & +2\beta(n)e_f(n)e_b(n-1) \\ S(n+M) = \quad & (1-\beta)^M S(n) + (e_f^2(n)+e_b^2(n-1)) \end{aligned} \tag{10-161}$$

24. 用下式重复计算习题22。

$$\begin{aligned} k(n+M) = & \quad [1-\beta(n)(e_f^2(n)+e_b^2(n-1))]k(n) \\ & \quad +2\beta(n)e_f(n)e_b(n-1) \\ S(n+M) = & \quad (1-\beta)^{\sqrt{M}}S(n)+(e_f^2(n)+e_b^2(n-1)) \end{aligned} \tag{10-162}$$

25. 考虑一个2状态动态规划计算（参看第1.2.8节）

$$x_i(n+1) = \max_j[x_j(n)+a_{ji}(n)],\ i,j=1,2 \tag{10-163}$$

它的迭代上下限受加法、比较与选择（addition-comparison-selection，ACS）操作的限制。 370 ~ 372

(a) 使用超前变换技术来得到等效的实现方式，其中ACS操作有4级流水线。使用分解技术来降低硬件复杂度的开销。

(b) 使用超前变换设计它的4并行实现方式。

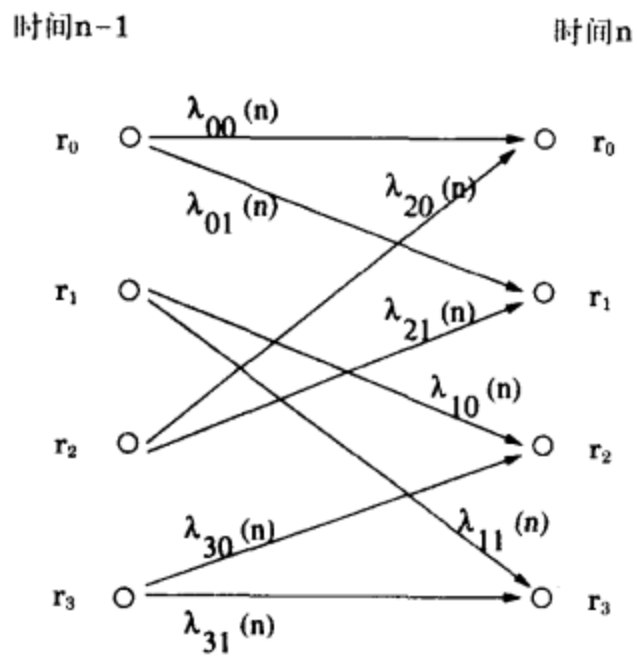


图10-34 习题26的格状图

26. 考虑一个4状态、编码率为1/2的Viterbi解码器（参看第1.2.8节），图10-34是它的格状图。格状图中的每一个结点代表一个ACS操作。例如，最顶端的结点（对应于状态0），进行如下运算:

$$r_0(n) = \min[r_0(n-1)+\lambda_{00}(n), r_2(n-1)+\lambda_{20}(n)] \tag{10-164}$$

这个结构迭代的上下限受一个ACS操作的吞吐量的限制。

(a) 使用超前变换设计Viterbi解码器，在ACS单元中实现2级流水线。

(b) 重复(a)在ACS单元中实现4级流水线。 373

参考文献

1. S. Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proc. IEEE*, vol. 72, pp. 867–884, July 1984.

2. C. E. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," in *3rd Caltech Conference on VLSI*, (Pasadena, CA), pp. 87–116, March 1983.

3. D. A. Schwartz and T. P. Barnwell, III, "A graph theoretic technique for the generation of systolic implementations for shift invariant flow graphs," in *Proc of ICASSP-84*, (San Diego), Mar. 1984.

4. K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters–part I: pipelining using scattered look-ahead and decomposition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, pp. 1099–1117, July 1989.

5. K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters–part II: pipelined incremental block filtering," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, pp. 1118–1134, July 1989.

6. H. H. Loomis, Jr. and B. Sinha, "High speed recursive digital filter realization," *Circuits, Systems, Signal Process.*, vol. 3, no. 3, pp. 267–294, 1984.

7. M. A. Soderstrand, A. E. de la Serna, and H. H. Loomis, Jr., "New approach to clustered look-ahead pipelined IIR digital filters," *IEEE Trans. on Circuits and Systems, Part II: Analog and Digital Signal Processing* , vol. 42, pp. 269–274, April 1995.

8. A. G. Deczky, "Synthesis of recursive digital filters using the minimum p-error criterion," *IEEE Trans. on Audio and Electroacoustics*, vol. AU-20, pp. 257–263, Oct. 1972.

9. R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Comput. Journal*, vol. 6, pp. 163–168, 1963.

10. M. A. Richards, "Application of Deczky's program for recursive filter design to the design of recursive decimators," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, pp. 811–814, Oct. 1982.

11. H. G. Martinez and T. W. Parks, "Design of recursive digital filters with optimum magnitude and attenuation poles on the unit circle," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-26, pp. 150–157, April 1978.

12. H. G. Martinez and T. W. Parks, "A class of infinite-duration impulse response digital filters for sampling rate reduction," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-27, pp. 154–162, April 1979.

13. A. V. Oppenheim and R. W. Schafer, *Discrete Time Signal Processing*. Prentice-Hall, 1989.

14. T. Meng and D. G. Messerschmitt, "Arbitrarily high sampling rate adaptive filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 35, no. 4, pp. 455–470, 1987.

374

15. G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 9, pp. 1397–1405, 1989.

16. N. R. Shanbhag and K. K. Parhi, "Relaxed look-ahead pipelined LMS adaptive filters and their application to ADPCM coder," *IEEE Trans. on Circuits*

and Systems, Part II: Analog and Digital Signal Processing, vol. 40, no. 12, pp. 753–766, 1993.

17. N. R. Shanbhag and K. K. Parhi, "Pipelined adaptive DFE architectures using relaxed look-ahead," *IEEE Trans. on Signal Processing*, vol. 43, no. 6, pp. 1368–1385, 1995.
18. N. R. Shanbhag and K. K. Parhi, "A pipelined adaptive lattice filter architecture," *IEEE Transactions on Signal Processing*, vol. 41, no. 5, pp. 1925–1939, May 1993.
19. K. K. Parhi, "High speed VLSI architectures for Huffman and Viterbi decoders," *IEEE Trans. on Circuits and Systems, Part II: Analog and Digital Signal Processing*, vol. 39, no. 6, pp. 385–391, 1992.
20. G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: Breaking the ACS-bottleneck," *IEEE Trans. on Communications*, vol. 37, no. 8, pp. 785–789, 1989.
21. K. K. Parhi, "Pipelining in algorithms with quantizer loops," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 7, pp. 745–754, 1991.
22. N. R. Shanbhag and K. K. Parhi, *Pipelined Adaptive Digital Filters*. Kluwer, 1994.
23. K. J. Raghunath, and K. K. Parhi, "Pipelined RLS adaptive filtering using scaled tangent rotations (STAR)," *IEEE Trans. on Signal Processing*, vol. 44, no. 10, pp. 2591–2604, Oct. 1996.
24. J. Ma, E. F. Deprettere and K. K. Parhi, "Pipelined CORDIC based QRD-RLS adaptive filtering using matrix look-ahead," *Proc. of 1997 IEEE Workshop on Signal Processing Systems: Design and Implementation*, (Leicester, U.K.), pp. 131-140, Nov. 1997.

第11章 缩放噪声与舍入噪声

11.1 引言

用数字系统实现一个数字滤波器传输函数时，必然要对系统中的信号和系数进行量化(quantization)，从而会导致总的输入-输出特性不理想。任何一种实现都会存在两种基本类型的量化效应[1],[2]。第一种来自对传输函数中参数（系数）的量化。对参数进行量化的结果是最终所实现的传输函数不同于理想的传输函数。于是，在定点VLSI实现或使用定点可编程DSP软件实现数字滤波器时，对其中有限字长行为的研究就显得特别重要。

第二种类型的量化效应源自信号的舍入。一个数字滤波器中的内部信号总是要被量化的，从而引起计算输出（computed output）产生误差。这种量化显然是一种非线性现象，并且可进一步细分为两种情况，即有限循环振荡（limit-cycle oscillation）和舍入噪声(roundoff noise)。有限循环振荡可以定义为滤波器输出中不希望的周期分量（undesirable periodic components)，它是由于量化过程本身是一种非线性操作引起的。注意，当反馈路径中存在非线性操作时，振荡总是可能发生的。相反，舍入噪声对滤波器输出的影响总是以一种随机扰动的方式存在，可以利用适当的噪声分析模型和线性系统理论进行分析。

除参数量化和信号舍入外，内部溢出必须防止，因为它们会产生更大的误差。这将通过在实现中适当地进行缩放来完成。缩放可以使滤波器内部变量的取值限制在与硬件相适应的范围。

当需要计算与滤波器内部结构相关的量时，一个线性滤波器的状态变量描述是极其有用的。这种表示能够提供一种对各种结构进行研究的数学公式，而用其他任何描述却是不可行的。由于已有简单的计算算法，一旦数字滤波器得以用状态变量方式描述，其内部每个节点(node）处的功耗以及输出中的舍入噪声都可以很容易地计算出来。本章所提出的这些技术对FIR数字滤波器也是有用的。

本章组织结构如下：第11.2节介绍缩放噪声和舍入噪声的概念。第11.3节讨论状态变量描述、状态协方差矩阵**K**和输出协方差矩阵**W**。第11.4节介绍缩放变换和舍入噪声的计算，其递归流水数字滤波器中的应用在第11.5节中讨论。第11.6节讨论矩阵**K**和**W**的快速计算[3]。第11.7节讨论降速、重定时和流水线变换，以便将非状态变量转换为状态变量。

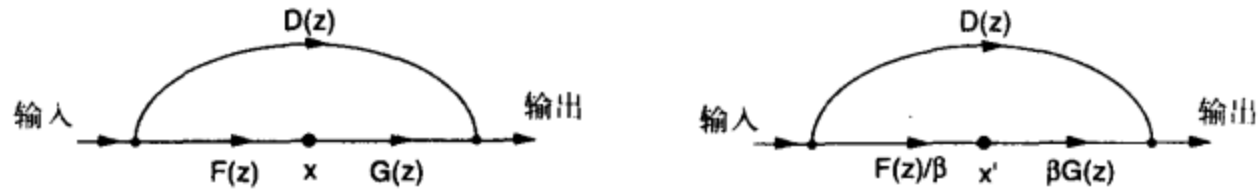
11.2 缩放噪声和舍入噪声

11.2.1 缩放操作

缩放（Scaling）是在保证传输函数从输入到输出不变的条件下，重新调整一定的内部增益参数，以便限制内部信号的范围能够适应硬件的一种处理。

图11-1a给出未对节点 x 进行缩放处理的滤波器，其传输函数为

$$H(z) = D(z) + F(z)G(z) \tag{11-1}$$



a) x点未作缩放的滤波器　　b) x' 点处进行缩放后的滤波器

图 11-1

为了对节点 x 进行缩放，我们用某个数 β 去除 $F(z)$，同时用它去乘 $G(z)$，如图11-1b所示。虽然这种操作不会改变传输函数，但是节点 x 处的信号水平已经发生变化。可以选择缩放参数 β 满足任何特殊缩放规则，例如

$$l_1 \text{ 缩放: } \quad \beta = \sum_{i=0}^{\infty} |f(i)| \tag{11-2}$$

$$l_2 \text{ 缩放: } \quad \beta = \delta\sqrt{\sum_{i=0}^{\infty} f^2(i)} \tag{11-3}$$

这里 $f(i)$ 是从输入到节点 x 的单位样值响应，而参数 δ 可解释为当输入是单位方差（unit-variance）白噪声时，节点 x 处的标准偏差数，可表示在寄存器中。如果输入满足 $|u(n)| \leqslant 1$，则

$$|x(n)| = \left|\sum_{i=0}^{\infty} f(i)u(n-i)\right| \leqslant \sum_{i=0}^{\infty} |f(i)| \tag{11-4}$$

方程（11-4）指出了 x 允许变化的真实界限以及通过公式(11-2)中的 l_1 缩放，溢出是完全可以避免的。这是一种最强的缩放策略。

很多情况下，输入可以假定是白噪声。这样假定时虽然不能准确计算节点 x 处的真实界限，但是能够计算出节点 x 处的方差。对单位方差白噪声输入，

$$\mathrm{E}\left[x^2(n)\right] = \sum_{i=0}^{\infty} f^2(i) \tag{11-5}$$

由于大部分输入信号可以假定是白噪声，所以通常使用 l_2 缩放。此外，公式(11-5)能够容易被计算出。由于公式(11-5)表示的是方差（不是一个严格的界限），所以存在溢出的可能，而这可以通过增加公式（11-3）中 δ 的值来消除。对大的 δ 值，可以保证缩放内部变量后溢出不再发生。然而，这里有个溢出噪声和舍入噪声之间的折中问题，因为 δ 的增加会使输出的SNR（信噪比）变坏。

378 ~ 379

11.2.2 舍入噪声

如果两个W比特字长的定点分数相乘，其积是一个2W-1比特长的数。这个积通过舍入或截断最终必须量化为一个W比特字长的数。例如，考虑图11-2给出的一阶IIR滤波器。假定输入字长W=8比特。如果乘法器的系数字长也是8比特，则为了做到输出的完全精确必须在每次迭代的输出中增加8比特的字长。这显然是不可行的。解决的办法是对输出进行舍弃（或截断）

以保留最左面的8比特。

这种量化的后果就是引入了舍入噪声 $e(n)$。为了数学上的方便，一个带有舍入操作的系统可以抽象为一个带有外部误差输入的、具有无限精度的系统。前面例子中（图11-2）对乘加操作的输出进行舍入后的等价模型如图11-3所示。

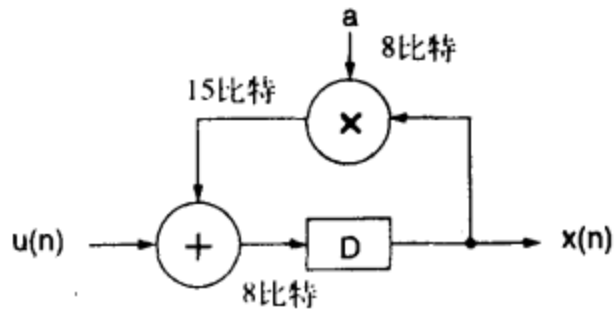


图11-2 一阶IIR滤波器（W=8）

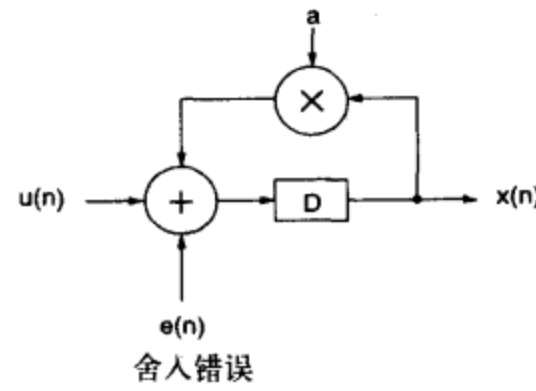


图11-3 舍入错误模型

380

尽管舍入操作不是一种线性操作，但是它对输出的影响在以下关于 $e(n)$ 的假定之下可以按照线性系统理论加以分析：

(1) $e(n)$ 是均匀分布的白噪声。

(2) $e(n)$ 是一个广义静态（wide-sense stationary）随机过程，也就是说，$e(n)$ 的均值和方差都独立于时间标记 n。

(3) $e(n)$ 与所有其他信号都不相关，例如输入信号和其他噪声信号等等。

设输出字长为W比特，则舍入错误 $e(n)$ 可表示为

$$\frac{-2^{-(W-1)}}{2} \leqslant e(n) \leqslant \frac{2^{-(W-1)}}{2} \tag{11-6}$$

由于已经假定误差在公式（11-6）给出的区间上服从均匀分布，所以相应的概率分布如图11-4所示，其中Δ表示区间的长度（即 $2^{-(W-1)}$）。

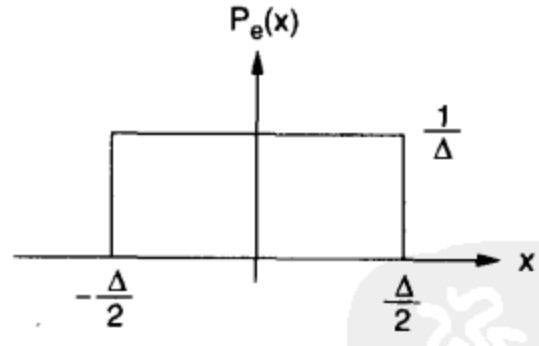


图11-4 错误概率分布

现计算这个错误函数的均值 $E[e(n)]$ 和方差 $E[e^2(n)]$。

$$E[e(n)] = \int_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} xP_e(x)dx = \frac{1}{\Delta}\frac{x^2}{2}\bigg|_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} = 0 \tag{11-7}$$

注意，由于均值为零，方差就简化为 $E[e^2(n)]$。

$$E[e^2(n)] = \int_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} x^2P_e(x)dx = \frac{1}{\Delta}\frac{x^3}{3}\bigg|_{\frac{-\Delta}{2}}^{\frac{\Delta}{2}} = \frac{\Delta^2}{12} = \frac{2^{-2W}}{3} \tag{11-8}$$

381

换句话说，公式（11-8）可写为

$$\sigma_e^2 = \frac{2^{-2W}}{3} \tag{11-9}$$

这里 σ_e^2 是W比特字长下有限精度系统中舍入错误的方差。因为这一方差与 2^{-2W} 成正比，所以每增加1比特的字长，错误就可以减小4倍。分析舍入噪声的目的是要确定它对输出信号的影响。如果输出的噪声方差与输出信号水平相比不能忽略，就应该增加字长或应该使用其他低噪声结构。因此，需要计算输出中的SNR，而不只是计算输出中的噪声增益。在噪声分析中，将使用一种称为双长度（double-length）累加器模型，亦即对$2W-1$比特的乘积做完加法后再对其进行舍入。另外要注意，乘法器是舍入噪声的根源。

11.3 数字滤波器的状态变量描述

考虑图11-5所示的一个N阶数字滤波器的信号流图（SFG）。可以通过以下的递归矩阵形式表示这个滤波器，

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{b}u(n) \tag{11-10}$$

$$y(n) = \mathbf{c}^T\mathbf{x}(n) + du(n) \tag{11-11}$$

其中，黑体字母表示矢量或矩阵。上述表示中，$\mathbf{x}$是状态，u是滤波器的输入，y是输出；$\mathbf{x}$、$\mathbf{b}$和$\mathbf{c}$是 $N\times 1$ 的列；$\mathbf{A}$是 $N\times N$ 矩阵；d、u和y是标量。令 $f_i(n)$ 是从输入 $u(n)$ 到状态 $x_i(n)$ 的单位样值响应，令 $g_i(n)$ 是从状态 $x_i(n)$ 到输出 $y_i(n)$ 的单位样值响应。为避免内部溢出，缩放乘法器的输入是必要的。图11-5中，由于信号 $x(n)$ 是乘法器的输入，所以需要计算 $f(n)$ 以便缩放。相反地，为获得输出中的噪声方差，确定从噪声源 $e(n)$ 到 $y(n)$ 的单位样值响应也是必要的。因此，$g(n)$ 表示不考虑延时情况下噪声传输函数的单位样值响应，它对输出噪声方差没有影响。

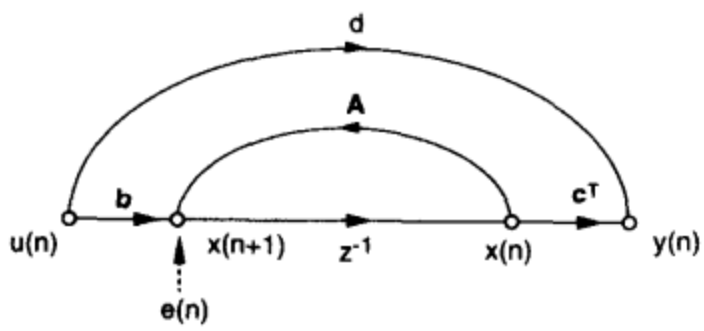


图11-5 IIR滤波器的信号流图

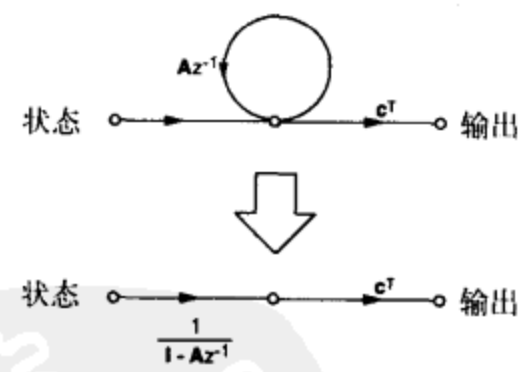


图11-6 $g(n)$ 的信号流图

从图11-5的SFG，可以获得如下表达式，

$$\frac{\mathbf{X}(z)}{U(z)} = \frac{\mathbf{b}z^{-1}}{\mathbf{I} - z^{-1}\mathbf{A}} \tag{11-12}$$

进一步表示 $f(n)$ 的 z 变换 $F(z)$ 为，

$$\mathbf{F}(z) = \frac{\mathbf{X}(z)}{U(z)} = (\mathbf{I} + \mathbf{A}z^{-1} + \mathbf{A}^2z^{-2} + \cdots)\mathbf{b}z^{-1} \tag{11-13}$$

$$\Rightarrow \mathbf{f}(n) = \mathbf{A}^{n-1}\mathbf{b}, \quad n \geqslant 1 \tag{11-14}$$

用 $\delta(n)$ 代替 $u(n)$ 后利用递归关系，

$$\mathbf{f}(n+1) = \mathbf{A}\mathbf{f}(n) + \mathbf{b}\delta(n) \tag{11-15}$$

在初始条件 $f(0)=0$ 之下完全可以计算出 $f(n)$。类似地，对于从状态 $x(n)$ 到输出 $y(n)$ 之间的单位样值响应 $g(n)$，也能在条件 $u(n)=0$ 下计算出。与之对应的SFG如图 11-6所示，它表示了以下的传输函数 $G(z)$，

$$\mathbf{G}(z) = \frac{\mathbf{c}^T}{\mathbf{I} - \mathbf{A}z^{-1}} \tag{11-16}$$

382 ~ 383

$$\Rightarrow \mathbf{g}(n) = \mathbf{c}^T\mathbf{A}^n \quad n \geqslant 0 \tag{11-17}$$

考虑如下定义的协方差矩阵$\mathbf{K}$，

$$\mathbf{K} = E[\mathbf{x}(n)\mathbf{x}^T(n)] \tag{11-18}$$

由于$\mathbf{X}$为 $N \times 1$，所以$\mathbf{K}$是一个 $N \times N$ 矩阵。令 $x_i^{'}s$ 表示$\mathbf{X}$的各元素，即，

$$\mathbf{x}(n) = [x_1(n)\ x_2(n)\ \cdots\ x_N(n)]^T \tag{11-19}$$

为理解矩阵$\mathbf{K}$的含义，现考虑 K_{11}，即矩阵$\mathbf{K}$的对角线上的第一个元素。注意 K_{11} 就是 $E\left[x_1^2(n)\right]$，它仅仅是输入白噪声在状态 $x_1(n)$ 处错误信号的能量。这表明，从某种意义上说，矩阵$\mathbf{K}$是各个状态点错误信号能量的一种度量。继续我们的讨论，试着用能够反映滤波器错误特性的某种方式来表示$\mathbf{K}$。注意公式（11-18）中的 $x(n)$ 可以写为输入 $u(n)$ 和单位样值响应 $f(n)$ 的卷积。进一步用公式（11-14）代替 $f(n)$ 可以得到，

$$\mathbf{x}(n) = \mathbf{f}(n) * u(n) = \sum_{l=0}^{\infty} \mathbf{A}^l\mathbf{b}u(n-l-1) \tag{11-20}$$

因此，

$$\begin{aligned}
\mathbf{K} &= E\left[\sum_{l=0}^{\infty} \mathbf{A}^l\mathbf{b}u(n-l-1) \sum_{m=0}^{\infty} u(n-m-1)(\mathbf{A}^m\mathbf{b})^T\right] \\
&= E\left[\sum_{l=0}^{\infty}\sum_{m=0}^{\infty} \mathbf{A}^l\mathbf{b}u(n-l-1)u(n-m-1)(\mathbf{A}^m\mathbf{b})^T\right] \\
&= \sum_{l=0}^{\infty}\sum_{m=0}^{\infty} \mathbf{A}^l\mathbf{b}E\left[u(n-l-1)u(n-m-1)\right](\mathbf{A}^m\mathbf{b})^T
\end{aligned} \tag{11-21}$$

如果假定输入 $u(n)$ 是均值为零的单位方差白噪声，则，

$$E[u^2(n)] = 1 \tag{11-22}$$

$$E[u(n)u(n-k)] = 0 \quad 当\ k \neq 0\ 时 \tag{11-23}$$

将公式（11-22）和公式（11-23）代入公式（11-21），得到，

$$\begin{aligned}
\mathbf{K} &= \sum_{l=0}^{\infty}\sum_{m=0}^{\infty} \mathbf{A}^l\mathbf{b}\delta_{lm}(\mathbf{A}^m\mathbf{b})^T \\
&= \sum_{l=0}^{\infty} \mathbf{f}(l)\mathbf{f}^T(l)
\end{aligned}$$

384

$$
\begin{aligned}
&= \sum_{l=0}^{\infty} \mathbf{A}^l\mathbf{b}(\mathbf{A}^l\mathbf{b})^T \\
&= \mathbf{b}\mathbf{b}^T + \sum_{l=1}^{\infty} \mathbf{A}^l\mathbf{b}(\mathbf{A}^l\mathbf{b})^T \\
&= \mathbf{b}\mathbf{b}^T + \sum_{k=0}^{\infty} \mathbf{A}^{k+1}\mathbf{b}(\mathbf{A}^{k+1}\mathbf{b})^T \\
&= \mathbf{b}\mathbf{b}^T + \sum_{k=0}^{\infty} \mathbf{A}[\mathbf{A}^k\mathbf{b}(\mathbf{A}^k\mathbf{b})^T]\mathbf{A}^T \\
&= \mathbf{b}\mathbf{b}^T + \mathbf{A}\left[\sum_{k=0}^{\infty} \mathbf{A}^k\mathbf{b}(\mathbf{A}^k\mathbf{b})^T\right]\mathbf{A}^T \qquad (11\text{-}24) \\
\Rightarrow \mathbf{K} &= \mathbf{b}\mathbf{b}^T + \mathbf{A}\mathbf{K}\mathbf{A}^T \qquad (11\text{-}25)
\end{aligned}
$$

方程（11-25）又称为李亚普诺夫（Lyapunov）方程。注意，如果对某个状态 x_i，$E\left[x_i^2\right]$有比其他状态高的值，则 x_i 需要占用更多的比特位。但是这样就会在不同的状态处有着不同的字长，这将导致额外的硬件开销和一种不规则的设计。如果在某种情况下能够确保所有点都具有相同的功率（亦即对所有的 i，$E\left[x_i^2\right]$相同），则可以给所有的点赋予相同的字长。这一点可以通过缩放来达到。

如果滤波器所有的内部变量不相关并且在白噪声输入下都具有单位方差，则该滤波器结构称为是正交的[3]。换句话说，一个滤波器结构如果满足以下条件则是正交的:

$$\mathbf{K} = \mathbf{I} = \mathbf{A}\mathbf{A}^T + \mathbf{b}\mathbf{b}^T \qquad (11\text{-}26)$$

正交滤波器结构具有以下优点:

- 缩放规则能够被自动满足。
- 舍入噪声低并且在频率变换下保持不变。
- 不可能有溢出振荡。

类似地、定义输出协方差矩阵$\mathbf{W}$如下:

$$\mathbf{W} = \sum_{n=0}^{\infty} \mathbf{g}^T(n)\mathbf{g}(n) = \sum_{n=0}^{\infty} (\mathbf{c}^T\mathbf{A}^n)^T\mathbf{c}^T\mathbf{A}^n \qquad (11\text{-}27)$$

类似于前面的推导过程，可以证明

385

$$\mathbf{W} = \mathbf{A}^T\mathbf{W}\mathbf{A} + \mathbf{c}\mathbf{c}^T \qquad (11\text{-}28)$$

11.4 缩放噪声和舍入噪声的计算

11.4.1 缩放操作

如前所述，给系统中的所有变量指定相同的字长，只有在所有的状态具有相同的能量时才行。实现这一目标的方法称为缩放(scaling)。状态被缩放矩阵$\mathbf{T}$的逆矩阵左乘。若记缩放后的状态为 X_s，则可得，

$$\mathbf{x}_s(n) = \mathbf{T}^{-1}\mathbf{x}(n) \;\Rightarrow\; \mathbf{x}(n) = \mathbf{T}\mathbf{x}_s(n) \tag{11-29}$$

将公式（11-29）中的 X 代入状态更新方程（the state update equation）（11-10）并对 X_s 进行求解可得，

$$\begin{aligned} \mathbf{T}\mathbf{x}_s(n+1) &= \mathbf{A}\mathbf{T}\mathbf{x}_s(n) + \mathbf{b}u(n) && (11\text{-}30)\\ \Rightarrow\; \mathbf{x}_s(n+1) &= \mathbf{T}^{-1}\mathbf{A}\mathbf{T}\mathbf{x}_s(n) + \mathbf{T}^{-1}\mathbf{b}u(n) && (11\text{-}31)\\ \Rightarrow\; \mathbf{x}_s(n+1) &= \mathbf{A}_s\mathbf{x}_s(n) + \mathbf{b}_s u(n) && (11\text{-}32) \end{aligned}$$

这里，$\mathbf{A}_s = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$ 和 $\mathbf{b}_s = \mathbf{T}^{-1}\mathbf{b}$。

类似地，输出方程（11-11）可推导如下，

$$\begin{aligned} y(n) &= \mathbf{c}^T\mathbf{T}\mathbf{x}_s(n) + du(n)\\ &= \mathbf{c}_s^T\mathbf{x}_s(n) + d_s u(n)\\ \Rightarrow\; \mathbf{c}_s^T &= \mathbf{c}^T\mathbf{T} \;\text{ 和 }\; d_s = d && (11\text{-}33) \end{aligned}$$

对缩放后的矩阵**K**可推导如下

$$\begin{aligned} \mathbf{K}_s &= E[\mathbf{x}_s\mathbf{x}_s^T]\\ &= E[\mathbf{T}^{-1}\mathbf{x}\mathbf{x}^T(\mathbf{T}^{-1})^T]\\ &= \mathbf{T}^{-1}E[\mathbf{x}\mathbf{x}^T](\mathbf{T}^{-1})^T\\ \Rightarrow\; \mathbf{K}_s &= \mathbf{T}^{-1}\mathbf{K}(\mathbf{T}^{-1})^T && (11\text{-}34) \end{aligned}$$

由于希望所有的状态都具有相同的能量，所以选择变换矩阵**T**使缩放后的系统矩阵 $\mathbf{K}_s$ 的主对角线上的元素都为1。进一步，假定**T**是对角矩阵，即，

$$\mathbf{T} = diag\,[t_{11}\; t_{22}\; \cdots\; t_{NN}] \tag{11-35}$$

$$\Rightarrow\; \mathbf{T}^{-1} = diag\,[\frac{1}{t_{11}}\; \frac{1}{t_{22}}\; \cdots\; \frac{1}{t_{NN}}] = (\mathbf{T}^{-1})^T \tag{11-36}$$

由公式（11-34）可见，矩阵 $\mathbf{K}_s$ 的对角线上元素可以用矩阵**K**表示，并且假如让 $(K_s)_{ii}$ 等于1，则可以得到缩放矩阵**T**: 386

$$(K_s)_{ii} = \frac{K_{ii}}{t_{ii}^2} = 1 \tag{11-37}$$

$$\Rightarrow\; t_{ii} = \sqrt{K_{ii}} \tag{11-38}$$

总之，通过选取矩阵**T**的对角线上第i个元素为矩阵**K**的对角线上第i个元素的平方根，就能保证所有状态都有相同的单位能量。

例 11.4.1 考虑图11-7所示的一个未经缩放的二级滤波器。其状态变量矩阵为

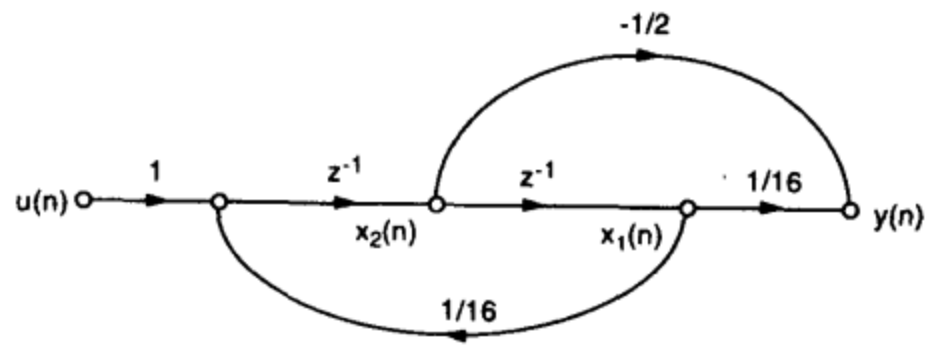
$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 0 & 1\\ \frac{1}{16} & 0 \end{bmatrix}\\ \mathbf{b} &= \begin{bmatrix} 0\\ 1 \end{bmatrix}\\ \mathbf{c} &= \begin{bmatrix} \frac{1}{16}\\ -\frac{1}{2} \end{bmatrix}\\ d &= 0 \end{aligned}$$

利用公式（11-25），对状态协方差矩阵**K**可计算如下

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{1}{16} & 0 \end{bmatrix} \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} 0 & \frac{1}{16} \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} K_{22} & \frac{1}{16}K_{21} \\ \frac{1}{16}K_{12} & \frac{1}{256}K_{11}+1 \end{bmatrix}$$

因此

$$K_{11} = K_{22} = \frac{256}{255}, \quad K_{12} = K_{21} = 0$$



387

图11-7 一个未缩放二级滤波器的SFG

对 $\delta=1$ 时的 l_2 缩放，变换矩阵为

$$\mathbf{T} = \begin{bmatrix} \frac{16}{\sqrt{255}} & 0 \\ 0 & \frac{16}{\sqrt{255}} \end{bmatrix}$$

因此缩放后的滤波器可描述为

$$\mathbf{A}_s = \mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \begin{bmatrix} 0 & 1 \\ \frac{1}{16} & 0 \end{bmatrix}$$

$$\mathbf{b}_s = \mathbf{T}^{-1}\mathbf{b} = \begin{bmatrix} 0 \\ \frac{\sqrt{255}}{16} \end{bmatrix}$$

$$\mathbf{c}_s = \mathbf{T}^{T}\mathbf{c} = \begin{bmatrix} \frac{1}{\sqrt{255}} \\ -\frac{8}{\sqrt{255}} \end{bmatrix}$$

$$d_s = 0$$

其信号流图如图11-8所示。注意，缩放后的滤波器的状态协方差矩阵是

$$\mathbf{K}_s = \begin{bmatrix} \frac{\sqrt{255}}{16} & 0 \\ 0 & \frac{\sqrt{255}}{16} \end{bmatrix} \begin{bmatrix} \frac{256}{255} & 0 \\ 0 & \frac{256}{255} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{255}}{16} & 0 \\ 0 & \frac{\sqrt{255}}{16} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

■

11.4.2 舍入噪声

令 $e_i(n)$ 是状态 x_i 处的舍入错误。则由这一错误所引起的输出舍入噪声 $y_i(n)$ 可表示为

输入 $e_i(n)$ 和状态到输出的单位样值响应 $g_i(n)$ 之间的卷积:

$$y_i(n) = e_i(n) * g_i(n) = \sum_{l=0}^{\infty} e_i(l) g_i(n-l) \tag{11-39}$$

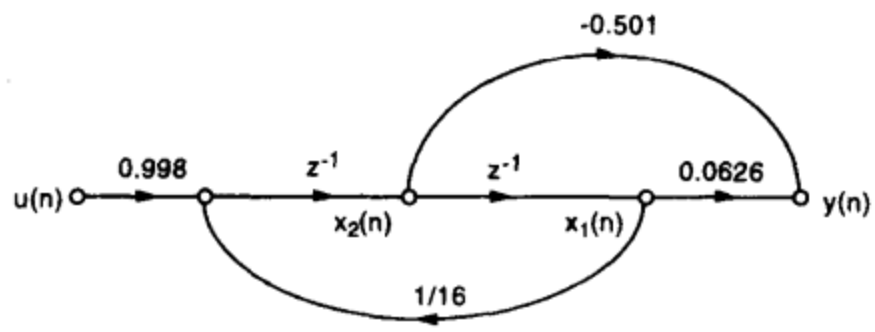


图11-8 一个二阶缩放滤波器的SFG

考虑 $y_i(n)$ 的均值和方差。由于 $e_i(n)$ 是均值为零的白噪声，所以 $y_i(n)$ 的均值也是零。

$$E[y_i(n)] = 0 \tag{11-40}$$

$$\begin{aligned} E[y_i^2(n)] &= E\left[\sum_l e_i(l) g_i(n-l) \sum_m e_i(m) g_i(n-m)\right] \\ &= \sum_l \sum_m g_i(n-l) E[e_i(l) e_i(m)] g_i(n-m) \\ &= \sum_l \sum_m g_i(n-l) \sigma_e^2 \delta_{lm} g_i(n-m) \\ &= \sigma_e^2 \sum_l g_i^2(n-l) = \sigma_e^2 \sum_n g_i^2(n) \end{aligned} \tag{11-41}$$

这里 σ_e^2 是 $e_i(n)$ 的方差。将**W**展开并将其写为矩阵形式，则可以看到所有对角线上的元素都具有形式 $\sum_n g_i^2(n)$。

$$\begin{aligned} \mathbf{W} &= \sum_n \mathbf{g}^T(n)\mathbf{g}(n) \\ &= \sum_n \begin{bmatrix} g_1(n) \\ g_2(n) \\ \vdots \\ g_N(n) \end{bmatrix} [g_1(n)\ g_2(n)\ \cdots\ g_N(n)] \end{aligned} \tag{11-42}$$

$$= \begin{bmatrix} \sum_n g_1^2(n) & \sum_n g_1(n)g_2(n) & \cdots & \sum_n g_1(n)g_N(n) \\ \sum_n g_2(n)g_1(n) & \sum_n g_2^2(n) & \cdots & \sum_n g_2(n)g_N(n) \\ \vdots & \vdots & \vdots & \vdots \\ \sum_n g_N(n)g_1(n) & \sum_n g_N(n)g_2(n) & \cdots & \sum_n g_N^2(n) \end{bmatrix} \tag{11-43}$$

利用公式（11-41），输出的总舍入噪声可用**W**的迹表示为:

$$\text{总舍入噪声} = \sigma_e^2 \sum_{i=1}^{N} \sum_n g_i^2(n) = \sigma_e^2 \sum_{i=1}^{N} W_{ii} = \sigma_e^2\ \mathrm{Trace}(\mathbf{W}) \tag{11-44}$$

388 ~ 389

虽然公式（11-44）给出的表示式对所有情况都有效，但是读者仍应注意，假如在某节点处没有舍入操作，则在计算噪声能量时对应于该节点的 W_{ii} 项不应包括在其中。

从公式（11-44）也能计算缩放系统的总舍入噪声，简单地说，这一噪声就是缩放矩阵**W**的迹。亦即，

$$\text{总舍入噪声（缩放系统）} = \sigma_e^2 \ \mathrm{Trace}(\mathbf{W}_s) \tag{11-45}$$

在公式（11-27）中，用缩放参数代替滤波器参数后，可以证明

$$\mathbf{W}_s = \mathbf{T}^T\mathbf{W}\mathbf{T} \tag{11-46}$$

此外，对对角矩阵**T**有，

$$\mathrm{Trace}(\mathbf{W}_s) = \sum_{i=1}^{N}(W_s)_{ii} = \sum_{i=1}^{N} t_{ii}^2 W_{ii} \tag{11-47}$$

由于 t_{ii} 定义为 $\sqrt{K_{ii}}$ ，公式（11-47）又可写为

$$\begin{aligned} \mathrm{Trace}(\mathbf{W}_s) &= \sum_{i=1}^{N}(K_{ii}W_{ii}) \\ \Rightarrow \text{总舍入噪声（缩放系统）} &= \sigma_e^2 \sum_{i=1}^{N}(K_{ii}W_{ii}) \end{aligned} \tag{11-48}$$

缩放系统的舍入噪声可由 K_{ii} 和 W_{ii} 计算得出。

例11.4.2 为获得图11-8所示 l_2 缩放滤波器的输出舍入噪声，利用（11-28）式可对**W**计算如下

$$\begin{aligned} \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} &= \begin{bmatrix} 0 & \frac{1}{16} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ \frac{1}{16} & 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{255} & -\frac{8}{255} \\ -\frac{8}{255} & \frac{64}{255} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{256}W_{22} + \frac{1}{255} & \frac{1}{16}W_{21} - \frac{8}{255} \\ \frac{1}{16}W_{12} - \frac{8}{255} & W_{11} + \frac{64}{255} \end{bmatrix} \end{aligned}$$

因此

$$\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} 0.0049 & -0.0332 \\ -0.0332 & 0.2559 \end{bmatrix}$$

该缩放滤波器总的舍入噪声是

390

$$(W_{11} + W_{22})\sigma_e^2 = 0.2608\sigma_e^2$$

对图11-7所示未经缩放的滤波器，

$$\begin{aligned} \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} &= \begin{bmatrix} 0 & \frac{1}{16} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ \frac{1}{16} & 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{256} & -\frac{1}{32} \\ -\frac{1}{32} & \frac{1}{4} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{256}W_{22} + \frac{1}{256} & \frac{1}{16}W_{21} - \frac{1}{32} \\ \frac{1}{16}W_{12} - \frac{1}{32} & W_{11} + \frac{1}{4} \end{bmatrix} \end{aligned}$$

因此

$$\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} 0.0049 & -0.0333 \\ -0.0333 & 0.2549 \end{bmatrix}$$

该未缩放滤波器总的舍入噪声是

$$(W_{11} + W_{22})\sigma_e^2 = 0.2598\sigma_e^2$$

注意，缩放滤波器具有更大的舍入噪声，亦即，

$$\frac{\text{舍入噪声（非缩放）}}{\text{舍入噪声（缩放）}} = \frac{0.2598}{0.2608}$$

这一结果也可以从比较未缩放滤波器结构和缩放滤波器结构得出。缩放滤波器中，通过给输入乘以0.998以避免溢出发生（参见图11-8）。因此，为保证两个滤波器的传输函数是相同的，缩放滤波器的输出路径的增益应该是未缩放滤波器输出路径的增益的1/0.998倍。所以，缩放滤波器的舍入噪声是未缩放滤波器的$1/0.998^2$倍。

上述情况说明了溢出噪声和舍入噪声之间的折中。缩放越严格越能降低溢出的可能，但也增加了舍入噪声效应。

另外注意，对以上未缩放滤波器，

$$K_{11}W_{11} + K_{22}W_{22} = \frac{256}{255}(0.0049 + 0.2549) = 0.2608$$

它等于缩放滤波器中的 $W_{11} + W_{22}$。因此公式（11-48）得到了验证。 ■

11.5 流水线IIR滤波器中的舍入噪声

本节讨论一阶和二阶直接型流水线IIR滤波器（Pipelined 1st-order and 2nd-order direct-form IIR filters）的舍入噪声特性[4]，[5]。缩放流水线的舍入噪声情况也将给予考虑。 391

11.5.1 一阶IIR滤波器

考虑有一个极点（Pole）的IIR滤波器的传输函数

$$H(z) = \frac{bcz^{-1}}{1 - az^{-1}} + d \tag{11-49}$$

我们感兴趣的是这个滤波器在有或没有 l_2 缩放情况下不同级流水线（亦即不同M值）的总舍入噪声。该滤波器的信号流图如图11-9所示。

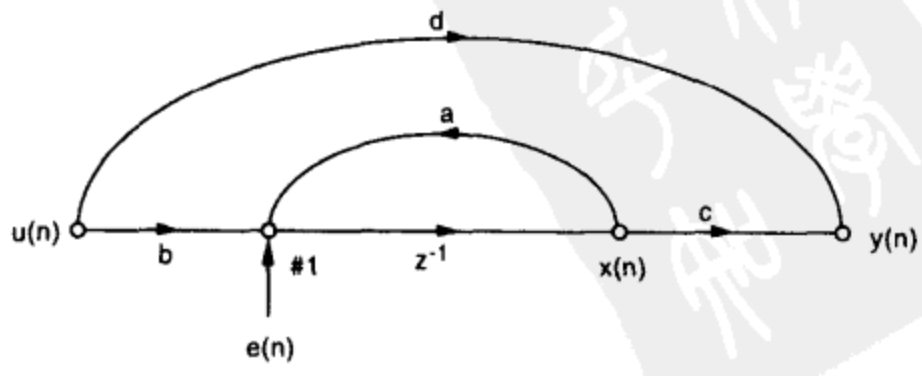


图11-9 一阶非流水IIR滤波器的信号流图

M=1 非缩放系统

从图11-9能够看出，该滤波器仅有一次舍入操作（在节点1处），因此这里只有一个错误源 $e(n)$ 和一个状态。如公式（11-44）所指出的，舍入噪声是 $\sigma_e^2\sum_n g^2(n)$，其中 $g(n)$ 是由这一状态到输出 $y(n)$ 的单位样值响应。从图11-9的SFG可以得到从舍入输入到输出的传输函数是:

$$G(z)=\frac{cz^{-1}}{1-az^{-1}}=cz^{-1}(1+az^{-1}+a^2z^{-2}+\cdots)$$

$$\Rightarrow G(z)=cz^{-1}+caz^{-2}+ca^2z^{-3}+\cdots \tag{11-50}$$

$$\Rightarrow \text{时间序列}\quad g(n)=\{0,\ c,\ ac,\ a^2c,\ \cdots\} \tag{11-51}$$

舍入噪声由下式计算得出

$$\text{舍入噪声}=\sigma_e^2\sum_n g^2(n)=\sigma_e^2\frac{c^2}{1-a^2} \tag{11-52}$$

M=1 缩放系统

为计算缩放系统的舍入噪声，首先需要计算**K**。因为**K**由 $\sum_n f^2(n)$ 定义，所以需要计算 $f(n)$。其中 $f(n)$ 是从输入到状态节点的单位样值响应。与传输函数 $F(z)$ 对应的SFG如图11-10所示，并由此可以写出 $F(z)$ 的表达式以及能够得到时间序列 $f(n)$。最后，从**K**和 t 能得到所有的缩放参数。

392

$$F(z)=\frac{b}{1-az^{-1}}=b(1+az^{-1}+a^2z^{-2}+\cdots) \tag{11-53}$$

$$\Rightarrow f(n)=\{b,\ ab,\ a^2b,\cdots\} \tag{11-54}$$

$$\Rightarrow K=\sum_n f^2(n)=\frac{b^2}{1-a^2} \tag{11-55}$$

$$\Rightarrow t=\sqrt{K}=\frac{b}{\sqrt{1-a^2}} \tag{11-56}$$

因此，

$$\begin{aligned}
a_s &= \mathbf{T}^{-1}a\mathbf{T}=\frac{1}{t}at=a\\
b_s &= \mathbf{T}^{-1}b=\frac{\sqrt{1-a^2}}{b}b=\sqrt{1-a^2}\\
c_s &= \mathbf{T}^{T}c=\frac{bc}{\sqrt{1-a^2}}\\
d_s &= d
\end{aligned} \tag{11-57}$$

读者可以验证缩放系统中的 $K_s=b_s^2/(1-a_s^2)=1$。正如公式（11-52），用缩放参数 a_s、b_s、c_s 表示的舍入噪声是

$$\text{舍入噪声}=\sigma_e^2\sum_n g^2(n)=\sigma_e^2\frac{c_s^2}{1-a_s^2}=\sigma_e^2\frac{b^2c^2}{(1-a^2)^2} \tag{11-58}$$

除此之外，舍入噪声也可以通过计算原始系统中的**KW**来获得，公式如下:

$$\text{舍入噪声} = \sigma_e^2(KW) = \sigma_e^2 \frac{b^2}{(1-a^2)} \frac{c^2}{(1-a^2)} \tag{11-59}$$

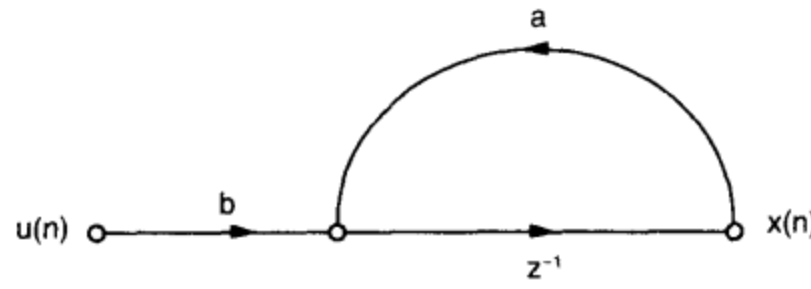


图11-10 $f(n)$ 的信号流图

393

M=2 情况

这种情况下的传输函数是:

$$H(z) = \frac{(bcz^{-1})(1+az^{-1})}{1-a^2z^{-2}} + d \tag{11-60}$$

图11-11给出与 $z^{-1}H(z)$ 对应的SFG。为分析舍入噪声，首先计算该滤波器的K。注意，这里仍然只有一次舍入操作，从而只有一个状态。确定 $F(z)$ 的SFG如图11-12所示。

$$F(z) = \frac{b+abz^{-1}}{1-a^2z^{-2}} = \frac{b}{1-az^{-1}} \tag{11-61}$$

$$\Rightarrow K = \frac{b^2}{1-a^2} \tag{11-62}$$

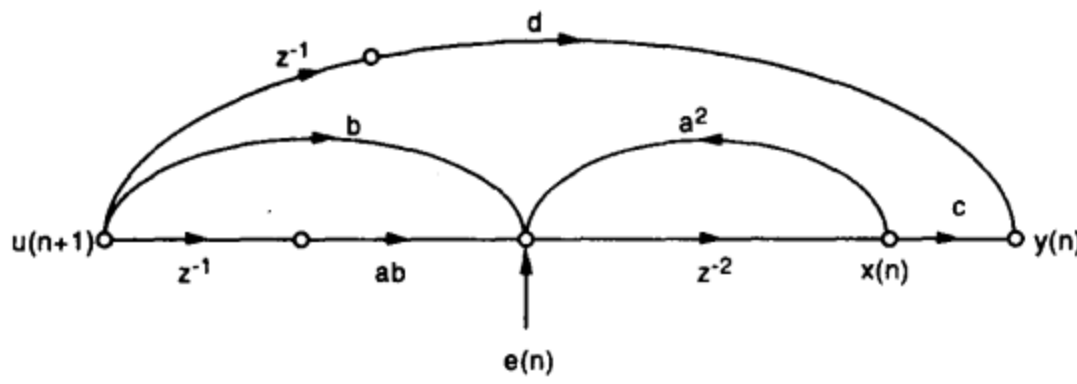


图11-11 二级流水IIR滤波器的信号流图

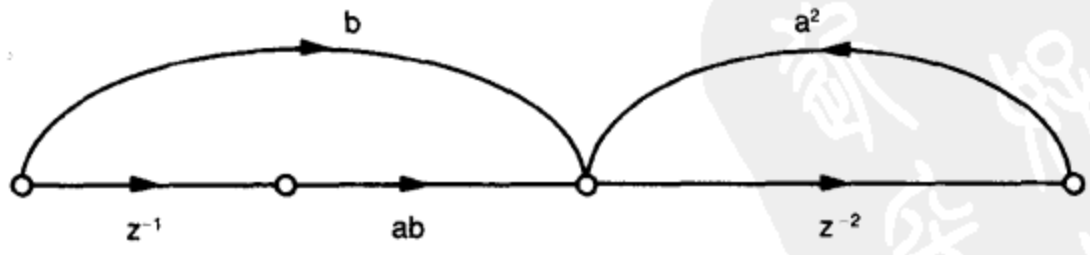


图11-12 $f(n)$ 的信号流图

类似地，由 $G(z)$ 可计算出W。确定 $G(z)$ 的SFG如图11-13所示。

$$G(z) = \frac{z^{-2}c}{1-a^2z^{-2}} = cz^{-2}(1+a^2z^{-2}+a^4z^{-4}+\cdots) \tag{11-63}$$

394

$$\Rightarrow g(n) = \{0,\ 0,\ c,\ 0,\ a^2c,\ 0,\ a^4c,\ \cdots\} \tag{11-64}$$

$$\Rightarrow W = \sum_n g^2(n) = \frac{c^2}{1-a^4} \tag{11-65}$$

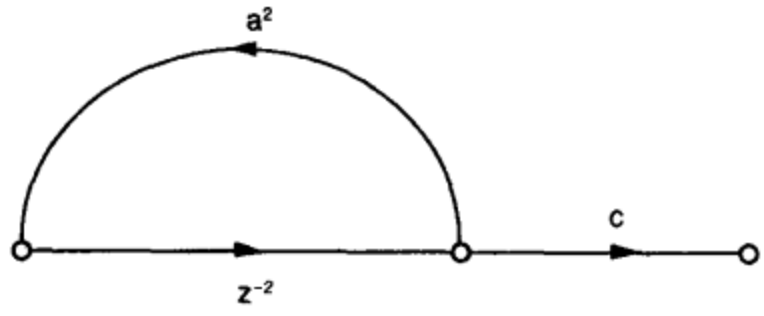


图11-13 $g(n)$ 的信号流图

由于这时**K**和**W**都是已知的，所以既可以计算缩放系统的舍入噪声也可以计算非缩放系统的舍入噪声。

$$\text{舍入噪声（非缩放）} = \sigma_e^2\ \text{Trace}\ (W) = \sigma_e^2 \frac{c^2}{1-a^4} \tag{11-66}$$

$$\text{舍入噪声（缩放）} = \sigma_e^2 \sum_i K_{ii}W_{ii} = \sigma_e^2 \frac{b^2c^2}{(1-a^2)(1-a^4)} \tag{11-67}$$

M=4 情况

4级流水的传输函数 $H(z)$ 如下:

$$H(z) = \frac{(bcz^{-1})(1+az^{-1})(1+a^2z^{-2})}{1-a^4z^{-4}} + d \tag{11-68}$$

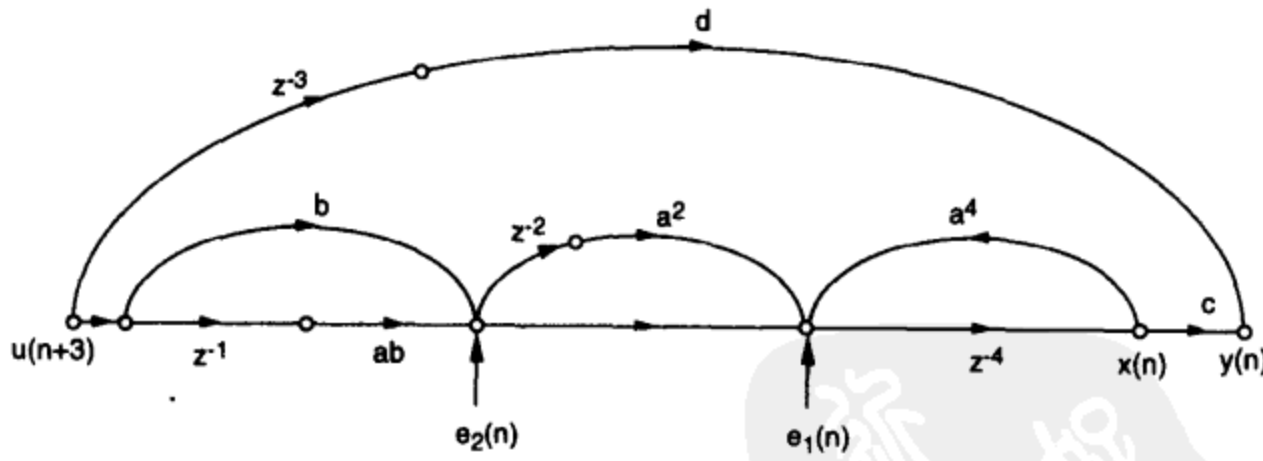


图11-14 4级流水IIR滤波器的信号流图

395 图11-14所示的SFG实现了 $z^{-3}H(z)$。注意，现在有两个噪声源 $e_1(n)$ 和 $e_2(n)$，这是因为正在做两个舍入操作。因此，**K**和**W**都是 2×2 矩阵并且需要计算**K**和**W**对角线上的所有元素。

K_{11}:

$$\begin{aligned} F_1(z) &= \frac{(b+abz^{-1})(1+a^2z^{-2})}{1-a^4z^{-4}} = \frac{b}{1-az^{-1}} \\ &= b(1+az^{-1}+a^2z^{-2}+\cdots) \\ \Rightarrow f_1(n) &= \{b,\ ab,\ a^2b,\ \cdots\} \end{aligned} \tag{11-69}$$

$$\Rightarrow K_{11} = \sum_{n=0}^{\infty} f_1^2(n) = \frac{b^2}{1-a^2} \tag{11-70}$$

K_{22}:

$$\begin{aligned} F_2(z) &= b + abz^{-1} \\ \Rightarrow f_2(n) &= \{b,\ ab,\ 0,\ 0,\ \cdots\} \end{aligned} \tag{11-71}$$

$$\Rightarrow K_{22} = \sum_{n=0}^{\infty} f_2^2(n) = b^2(1+a^2) \tag{11-72}$$

W_{11}:

$$\begin{aligned} G_1(z) &= \frac{cz^{-4}}{1-a^4z^{-4}} = cz^{-4}(1+a^4z^{-4}+a^8z^{-8}+\cdots) \\ \Rightarrow g_1(n) &= \{0,\ 0,\ 0,\ 0,\ c,\ 0,\ 0,\ 0,\ a^4c,\ \cdots\} \end{aligned} \tag{11-73}$$

$$\Rightarrow W_{11} = \sum_{n=0}^{\infty} g_1^2(n) = \frac{c^2}{1-a^8} \tag{11-74}$$

W_{22}:

$$\begin{aligned} G_2(z) &= \frac{cz^{-4}(1+a^2z^{-2})}{1-a^4z^{-4}} = \frac{cz^{-4}}{1-a^2z^{-2}} \\ &= cz^{-4}(1+a^2z^{-2}+a^4z^{-4}+\cdots) \\ \Rightarrow g_2(n) &= \{0,\ 0,\ 0,\ 0,\ c,\ 0,\ a^2c,\ \cdots\} \end{aligned} \tag{11-75}$$

$$\Rightarrow W_{22} = \sum_{n=0}^{\infty} g_2^2(n) = \frac{c^2}{1-a^4} \tag{11-76}$$

现在，计算系统舍入噪声的所有参数都已具备。

$$\text{舍入噪声（非缩放）} = \sigma_e^2 \sum_{i=1}^{2} W_{ii} = \sigma_e^2 \frac{c^2(2+a^4)}{1-a^8} \tag{11-77}$$

396

表11-1 两种特殊情况下的舍入噪声（$\sigma_e^2=1$）

情 况	未缩放	缩 放
$a\to 0$	$c^2\log_2 M$	$b^2c^2\log_2 M$
$a=1-\varepsilon$	$\frac{c^2(M-1)}{2M\varepsilon}$	$\frac{b^2c^2}{4M\varepsilon^2}$

表11-2 作为M和ε函数时的舍入噪声（$\sigma_e^2=1$）

	未 缩 放			缩 放		
M	ε=0.1	ε=0.2	ε=0.3	ε=0.1	ε=0.2	ε=0.3
1	5.00	2.50	1.67	25.0	6.25	2.78
2	2.50	1.25	0.83	12.5	3.12	1.39
4	3.75	1.87	1.25	6.25	1.56	0.69

$$\text{舍入噪声（缩放）} = \sigma_e^2 \sum_{i=1}^{2} K_{ii} W_{ii} = \sigma_e^2 \frac{b^2 c^2 (2 - a^8)}{(1 - a^2)(1 - a^8)} \tag{11-78}$$

事实上，对M是2的任意M次幂时的舍入噪声，都可以写出它的一般表达式。缩放系统和非缩放系统舍入噪声值由下述公式给出:

$$\text{舍入噪声（非缩放）} = \sigma_e^2 c^2 \sum_{j=1}^{\log_2 M} \frac{1}{1 - a^{2^{j+1}}} \tag{11-79}$$

$$\text{舍入噪声（缩放）} = \sigma_e^2 b^2 c^2 \frac{\log_2 M - (log_2 M - 1) a^{2M}}{(1 - a^2)(1 - a^{2M})} \tag{11-80}$$

这里M是2的某次幂并且$M>1$。

如果公式（11-79）和公式（11-80）中的a非常接近1或者0，则需分别给予考虑。当$a \to 0$时，公式（11-79）和公式（11-80）中的各个分母都变成1，这时缩放系统和非缩放系统的舍入噪声分别是 $\sigma_e^2 b^2 c^2 \log_2 M$ 和 $\sigma_e^2 c^2 \log_2 M$ 。为分析 $a \to 1$ 时的情况，假定 $a=1-\varepsilon$，这里$\varepsilon<<1$。用这一式子替换公式（11-79）和公式（11-80）中的a，可得舍入噪声表达式，如表11-1所示。

为进一步说明这种情况，考虑 $b=c=1$ 和 $a=1-\varepsilon$ 时的情况。表11-2给出了这种情况下不同M和不同ε所对应的舍入噪声值。这里最重要的是要注意不能把缩放和降低舍入噪声相混淆。缩放是给系统中每个状态分配相同字长的一种方法。表11-2清楚地表明了这一点。注意，在某些情况下缩放使总舍入噪声增加了。进一步，当极点接近1时，缩放滤波器的舍入噪声随着M的减小线性地减少，而这种情况下未缩放滤波器的舍入噪声却大致不变。

397

为确定流水线滤波器的字长或SNR，必须计算该滤波器的输出信号功率。当一个滤波器通过先行离散技术(scattered look-ahead technique)进行流水时，该滤波器输出信号的功率等同于非流水滤波器的功率。由公式（11-49），非流水滤波器的输出信号功率可利用Parseval等式得到

$$\sum_n h^2(n) = \frac{1}{2\pi j} \oint_C H(z) H(z^{-1}) z^{-1} dz \tag{11-81}$$

$$= \frac{b^2 c^2}{1 - a^2} + d^2 \tag{11-82}$$

这里C是一个沿逆时针方向的环线，而利用柯西(Cauchy)留数定理[1]，[3]容易计算出公式（11-81）中的环线积分值。

为方便讨论，可令公式（11-82）中的d为零。这是因为在公式（11-49）中，d未引起内部舍入操作（见图11-9以及注意公式（11-79）和公式（11-80）与d无关）。这样，对非流水滤波器（M=1）可以计算出它的噪信比是:

$$\text{NSR（非缩放）} = \frac{\sigma_e^2}{b^2} \tag{11-83}$$

$$\text{NSR（缩放）} = \frac{\sigma_e^2}{1 - a^2} \tag{11-84}$$

对$M>1$，

$$\text{NSR（非缩放）} = \sigma_e^2 \frac{1-a^2}{b^2} \sum_{j=1}^{\log_2 M} \frac{1}{1-a^{2j+1}} \tag{11-85}$$

$$\text{NSR（缩放）} = \sigma_e^2 \frac{\log_2 M - (log_2 M - 1)a^{2M}}{1-a^{2M}} \tag{11-86}$$

11.5.2 二阶IIR滤波器

考虑下面的二阶IIR滤波器

$$H(z) = \frac{1}{1-2r\cos\theta z^{-1} + r^2 z^{-2}} \tag{11-87}$$

该滤波器的SFG如图11-15所示。我们感兴趣的是这一滤波器在 l_2 缩放之下进行不同各级流水时的总舍入噪声。未缩放时总舍入噪声的计算留给读者。 398

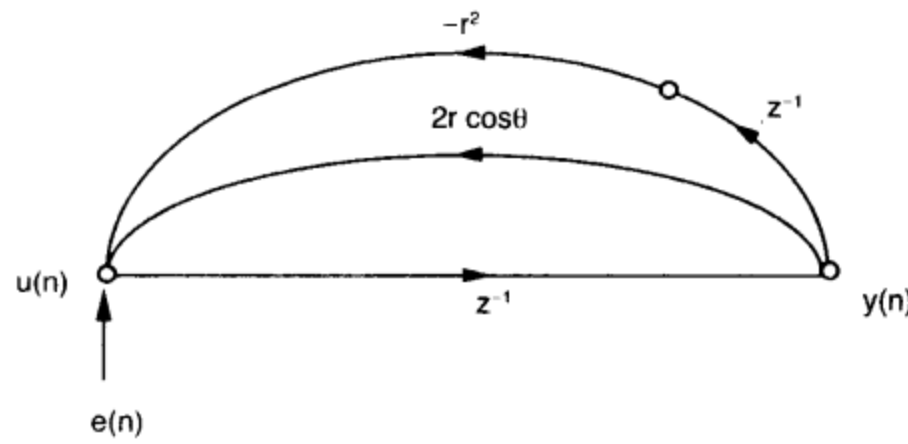


图11-15 二阶非流水IIR滤波器的信号流图

M=1 情况

正如图11-15所示，这种情况下该滤波器只进行一次舍入操作。因此，只有一个错误源 $e(n)$。由图11-15，

$$F(z) = \frac{1}{1-2r\cos\theta z^{-1} + r^2 z^{-2}} \tag{11-88}$$

由此，利用Parseval公式，

$$\sum_n f^2(n) = \frac{1}{2\pi j} \oint_C \frac{zdz}{(z-re^{j\theta})(z-re^{-j\theta})(1-re^{j\theta}z)(1-re^{-j\theta}z)} \tag{11-89}$$

由柯西（Cauchy）留数定理，公式（11-89）可表示为

$$\begin{aligned} K &= \sum_n f^2(n) \\ &= \frac{re^{j\theta}}{(re^{j\theta}-re^{-j\theta})(1-r^2e^{j2\theta})(1-r^2)} + \\ &\quad \frac{re^{-j\theta}}{(re^{-j\theta}-re^{j\theta})(1-r^2e^{-j2\theta})(1-r^2)} \\ &= \frac{1+r^2}{1-r^2}\frac{1}{1-2r^2\cos 2\theta + r^4} \end{aligned} \tag{11-90}$$

由于$K=M$（从图11-15可见），可得2阶非流水系统的舍入噪声是

$$\begin{aligned}\text{舍入噪声} &= \sigma_e^2(KW)\\ &= \sigma_e^2K^2\\ &= \sigma_e^2\left(\frac{1+r^2}{1-r^2}\right)^2\frac{1}{(1-2r^2\cos 2\theta+r^4)^2}\end{aligned} \tag{11-91}$$

399

从公式（11-91）可以看出，极点位置和舍入噪声之间的关系是

$$\begin{aligned}r\to 0 &: \quad \text{舍入噪声} \to \sigma_e^2\\ r\to 1 &: \quad \text{舍入噪声} \to \frac{\sigma_e^2}{(4\epsilon\sin^2\theta)^2}\end{aligned} \tag{11-92}$$

这里 $\varepsilon\to 0$。

M=2 情况

2级流水的传输函数是:

$$H(z)=\frac{1+2r\cos\theta z^{-1}+r^2z^{-2}}{1-2r^2\cos 2\theta z^{-2}+r^4z^{-4}} \tag{11-93}$$

相应的信号流图如图11-16所示。图11-16中只有一个错误源 $e(n)$。因此，$F(z)$ 可计算如下

$$\begin{aligned}F(z) &= \frac{1+2r\cos\theta z^{-1}+r^2z^{-2}}{1-2r^2\cos 2\theta z^{-2}+r^4z^{-4}}\\ &= \frac{1}{1-2r\cos\theta z^{-1}+r^2z^{-2}}\end{aligned} \tag{11-94}$$

注意，公式（11-94）中的 $F(z)$ 和公式（11-88）中的 $F(z)$ 是相同的。所以K由公式（11-90）给出。

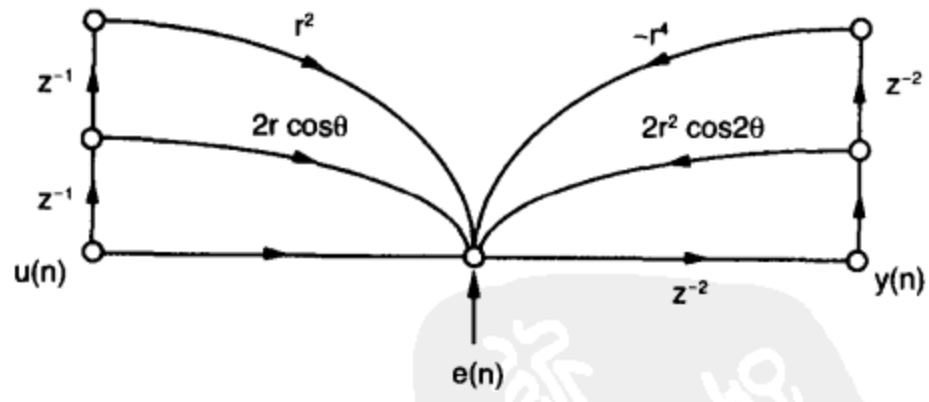


图11-16 2级流水IIR滤波器的信号流图

取图11-16中的输入为0，则可计算 $G(z)$ 如下

$$G(z)=\frac{1}{1-2r^2\cos 2\theta z^{-2}+r^4z^{-4}} \tag{11-95}$$

从而

400

$$W=\frac{1+r^4}{1-r^4}\frac{1}{1-2r^4\cos 4\theta+r^8} \tag{11-96}$$

因此，

$$
\begin{aligned}
\text{舍入噪声} &= \sigma_e^2(KW) \\
&= \sigma_e^2 \frac{(1+r^2)(1+r^4)}{(1-r^2)(1-r^4)} \times \\
&\quad \frac{1}{(1-2r^2\cos 2\theta + r^4)(1-2r^4\cos 4\theta + r^8)}
\end{aligned}
\tag{11-97}
$$

从公式（11-97）可以看出极点位置与舍入噪声之间的关系如下:

$$
\begin{aligned}
r \to 0 \quad &: \quad \text{舍入噪声} \quad \to \sigma_e^2 \\
r \to 1 \quad &: \quad \text{舍入噪声} \quad \to \frac{\sigma_e^2}{(4\epsilon \sin^2\theta)(8\epsilon \sin^2 2\theta)}
\end{aligned}
\tag{11-98}
$$

M=4 情况

进行4级流水后，传输函数 $H(z)$ 变成了

$$
H(z) = \frac{(1+2r\cos\theta z^{-1} + r^2 z^{-2})(1+2r^2\cos 2\theta z^{-2} + r^4 z^{-4})}{1-2r^4\cos 4\theta z^{-4} + r^8 z^{-8}}
\tag{11-99}
$$

该滤波器的SFG如图11-17所示。注意，这时存在两个噪声源 $e_1(n)$ 和 $e_2(n)$。利用如前的推导过程，可得如下结论:

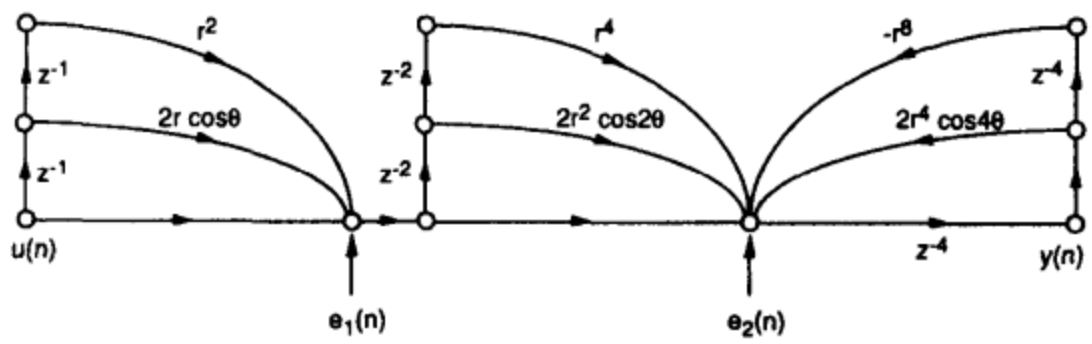


图11-17 4级流水IIR滤波器的信号流图

$e_1(n)$:

$$
\begin{aligned}
r \to 0 \quad &: \quad K_{11}W_{11} \to 1 \\
r \to 1 \quad &: \quad K_{11}W_{11} \to \frac{2+4\cos^2\theta}{8\epsilon \sin^2 2\theta}
\end{aligned}
\tag{11-100}
$$

$e_2(n)$:

$$
\begin{aligned}
r \to 0 \quad &: \quad K_{22}W_{22} \to 1 \\
r \to 1 \quad &: \quad K_{22}W_{22} \to \frac{1}{(4\epsilon \sin^2\theta)(16\epsilon \sin^2 4\theta)}
\end{aligned}
\tag{11-101}
$$

401

从公式（11-100）和公式（11-101）可得总舍入噪声是

$$
\begin{aligned}
r \to 0 \quad &: \quad \sigma_e^2 \sum_i K_{ii}W_{ii} \to 2\sigma_e^2 \\
r \to 1 \quad &: \quad \sigma_e^2 \sum_i K_{ii}W_{ii} \to \frac{\sigma_e^2}{(4\epsilon \sin^2\theta)(16\epsilon \sin^2 4\theta)}
\end{aligned}
\tag{11-102}
$$

一般地，当M是2的方幂时，M级流水滤波器的舍入噪声可表示为

$$
\begin{aligned}
r \to 0 \quad &: \quad \sigma_e^2 \sum_i K_{ii} W_{ii} \to \sigma_e^2 \log_2 M \\
r \to 1 \quad &: \quad \sigma_e^2 \sum_i K_{ii} W_{ii} \to \frac{\sigma_e^2}{(4\epsilon \sin^2 \theta)(4M\epsilon \sin^2 M\theta)}
\end{aligned}
\tag{11-103}
$$

为计算NSR，还需要计算输出信号的功率，它由公式（11-90）给出（注意当$M=1$时$H(z)=F(z)$）。由公式（11-90），在输出节点，

$$
\begin{aligned}
r \to 0 \quad &: \quad \text{信号功率} \quad \to 1 \\
r \to 1 \quad &: \quad \text{信号功率} \quad \to \frac{1}{4\epsilon \sin^2 \theta}
\end{aligned}
\tag{11-104}
$$

因此，

$$
\begin{aligned}
r \to 0 \quad &: \quad \text{NSR} \to \log_2 M \\
r \to 1 \quad &: \quad \text{NSR} \to \frac{1}{4M\epsilon \sin^2 M\theta}
\end{aligned}
\tag{11-105}
$$

当极点靠近原点，NSR变得与M和θ无关。然而，当极点接近单位圆时NSR将迅速增加。当极点同时满足下面两个条件时，舍入噪声的影响将是非常严重的：

$$
\begin{aligned}
r \quad &\to \quad 1 \\
\theta \quad &\to \quad \frac{n\pi}{M}, \quad n = 0, \pm 1, \cdots, \pm M
\end{aligned}
\tag{11-106}
$$

注意公式（11-105）中关于NSR的表达式在 $\theta=n\pi/M$，$n=0, \pm 1, \ldots, \pm M$ 时是不成立的。这种情况下舍入噪声的计算作为练习留给读者（见习题11）。

402

11.6 舍入噪声的状态变量描述计算

Parseval关系式和柯西留数定理对处理数字滤波器中的信号功率和舍入噪声是有用的。然而，将Parseval关系式应用于复杂结构的滤波器（例如见图11-18中所示的3阶规格化缩放格型滤波器）时却是不容易的。

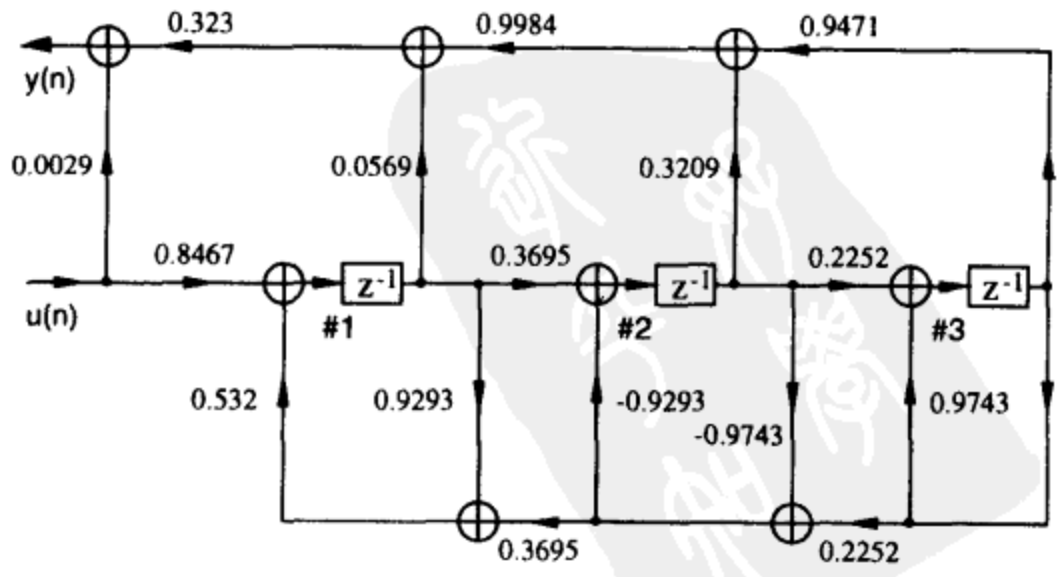


图11-18 一个3阶规格化缩放格型滤波器

不过，借助于一些可用的简单算法[3]，一旦一个数字滤波器得以用状态变量形式描述，该滤波器内部节点处的功率和该滤波器的输出舍入噪声就可以很容易地被计算出。利用公式(11-24)，应用下面的算法[3]可以有效地计算出**K**。

计算**K**的算法（**K**算法）

- 初始化:

  $\mathbf{F} \leftarrow \mathbf{A}$,

  $\mathbf{K} \leftarrow \mathbf{b}\mathbf{b}^T$。

- 循环:

  $\mathbf{K} \leftarrow \mathbf{F}\mathbf{K}\mathbf{F}^T + \mathbf{K}$,

  $\mathbf{F} \leftarrow \mathbf{F}^2$。

- 继续，直到 $\mathbf{F}=0$。

第一次循环迭代之后，

$$\begin{aligned} \mathbf{K} &= \mathbf{A}\mathbf{b}\mathbf{b}^T\mathbf{A}^T + \mathbf{b}\mathbf{b}^T \\ \mathbf{F} &= \mathbf{A}^2 \end{aligned} \tag{11-107}$$

403

第二次循环迭代之后，

$$\begin{aligned} \mathbf{K} &= \mathbf{A}^3\mathbf{b}\mathbf{b}^T(\mathbf{A}^3)^T + \mathbf{A}^2\mathbf{b}\mathbf{b}^T(\mathbf{A}^2)^T + \mathbf{A}\mathbf{b}\mathbf{b}^T\mathbf{A}^T + \mathbf{b}\mathbf{b}^T \\ \mathbf{F} &= \mathbf{A}^4 \end{aligned} \tag{11-108}$$

由此可见，每执行一次迭代，公式（11-24）中求和的项数将翻一番。由于矩阵**A**的特征值都是传输函数的极点，所以只要滤波器是稳定的上述算法就是收敛的（由线性代数理论可知，当且仅当矩阵**A**的所有特征值都小于1时，在 i 趋向无穷大时幂矩阵 $\mathbf{A}^i$ 趋于零矩阵）。注意，在计算**K**的算法中，只要在初始化步骤中用 $\mathbf{A}^T$ 代替 **A**，用c代替 b，则该算法也能计算**W**。

计算**W**的算法（**W**算法）

- 初始化:

  $\mathbf{F} \leftarrow \mathbf{A}^T$,

  $\mathbf{W} \leftarrow \mathbf{c}\mathbf{c}^T$。

- 循环:

  $\mathbf{W} \leftarrow \mathbf{F}\mathbf{W}\mathbf{F}^T + \mathbf{W}$,

  $\mathbf{F} \leftarrow \mathbf{F}^2$。

- 继续，直到 $\mathbf{F} = 0$。

例11.6.1 考虑如图11-18所示的规格化缩放格型滤波器，假定我们现在要计算节点1、2和3处的信号功率。由于有三个状态（1—3），所以矩阵 **A**、**b**、**c** 和 d 的维数分别是 3×3, 3×1, 3×1 和 1×1。由图11-18，状态方程可写为

$$\begin{aligned} x_1(n+1) &= 0.4944x_1(n) - 0.1915x_2(n) + 0.0443x_3(n) + 0.8467u(n) \\ x_2(n+1) &= 0.3695x_1(n) + 0.9054x_2(n) - 0.2093x_3(n) \\ x_3(n+1) &= 0.2252x_2(n) + 0.9743x_3(n) \\ y(n) &= 0.0184x_1(n) + 0.1035x_2(n) + 0.3054x_3(n) + 0.0029u(n) \end{aligned}$$

由这些方程，可以直接得到矩阵 **A**、**b**、**c** 和 d。将这些矩阵代入计算**K**的算法中，可得

$$\mathbf{K}=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

因为 $\mathbf{K}_{11}=\mathbf{K}_{22}=\mathbf{K}_{33}=1$，因此对节点1—3没有必要进行缩放。进一步，由于矩阵**K**的非主对角线上的元素都是零，这说明节点1—3处的信号互相正交。

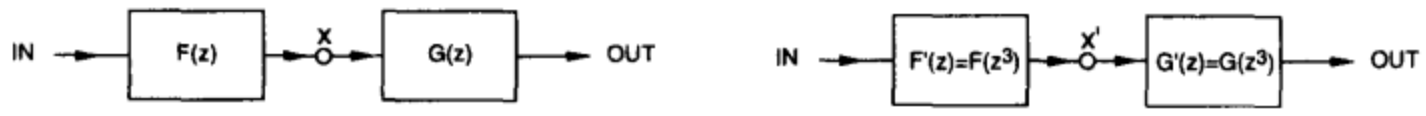
404

通过运用**W**算法可以得到 $W_{11}=0.1455$，$W_{22}=0.2952$ 和 $W_{33}=0.3096$。 ■

例11.6.1说明，只要一个数字滤波器得以用状态变量描述，则再计算它的信号功率或舍入噪声时都会是直截了当的。然而，对没有与单位延迟分支相连的节点，要计算这些点处的信号功率和舍入噪声却是困难的，因为这些节点不出现在状态变量描述中。例如，对图11-18中最顶端和最底端边上的节点，其信号功率就不能直接计算出。

11.7 降速、重定时和流水线

很多有用的实现都包含不与单位延迟分支相连的节点。这些节点或变量因而不出现在状态变量描述中，因此缩放和噪声计算方法都不能直接应用。为克服这一困难，可以使用SRP（即降速、重定时/流水线）变换技术（见第4和第3章）作为一个预处理步骤。



a) 传输函数为 $H(z)=F(z)\,G(z)$ 的一个滤波器　　b) 进行3降速变换后所得滤波器 $H'(z)=F(z^3)\,G(z^3)$

图 11-19

考虑图11-19b中所示的滤波器。这是一个图11-19a中所示滤波器通过降速变换（M=3）后得到的滤波器。通过3次降速变换，图11-19a中的每一个 z 变量都变成了 z^3。因此，变换后所得滤波器的传输函数 $H'(z)$ 和原滤波器的传输函数 $H(z)$ 之间的关系是

$$H'(z)=F'(z)G'(z)=F(z^3)G(z^3)=H(z^3) \tag{11-109}$$

所以，假如定义图11-19a中从输入到内部节点 x 的单位样值响应为

405

$$f(n)=\{f(0),\ f(1),\ f(2),\cdots\} \tag{11-110}$$

则对于图11-19b中从输入到内部节点 x' 处的单位样值响应是

$$f'(n)=\{f(0),\ 0,\ 0,\ f(1),\ 0,\ 0,\ f(2),\ 0,\ 0,\cdots\} \tag{11-111}$$

这样，

$$K'_{xx}=\sum_n (f'(n))^2=\sum_n f(n)^2=K_{xx} \tag{11-112}$$

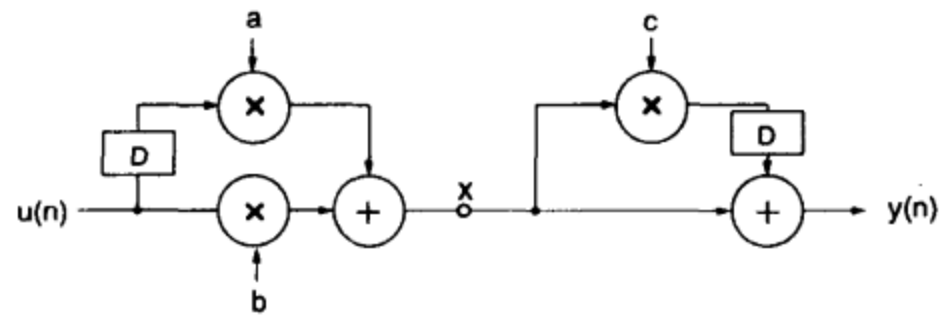
类似地，可以证明

$$W'_{xx}=W_{xx} \tag{11-113}$$

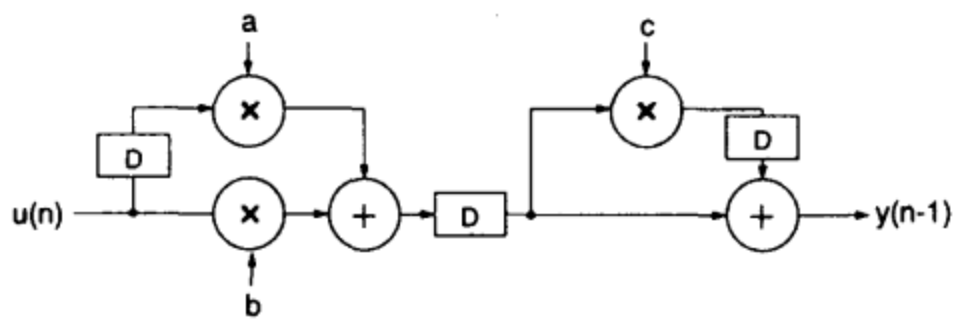
这一分析表明，降速变换不改变有限字长特性。

考虑图11-20a所示的滤波器，在前馈路径上它有一个非状态变量节点 x。很明显，非状态变量节点是不能通过降速变换转化为状态变量节点的。然而，由于 x 处于前馈路径，如图

11-20b所示，可以在适当的割集位置插入一个延时。这种流水线操作将这个非状态变量节点转化成了状态变量节点。该流水线滤波器的输出序列除相差一个时钟延时外等于原滤波器的输出序列。这意味着该流水线滤波器承载了与原滤波器完全一样的溢出可能和舍入噪声效应。因此，流水线显然不改变有限字长特性。



a) 前馈路径有一非状态变量的滤波器



b) 通过流水将非状态变量转换成了状态变量

图 11-20

在一个线性阵列，如果各模块之间的所有左向边或右向边都至少带有一个延时，则割集定位过程可应用于将一些延迟或一个延时的一部分转换为相反方向的边。(见第4章) [6]，[7]。基于这一特性和前面的观察，可将SRP变换技术概括如下: 406

(1) 将一个M因子的降速变换应用于一个线性阵列，相当于用 z^M 去替换 z。此外，还可以将流水线技术应用于适当的位置。

(2) 将额外的延迟散布于适当的位置使得将非状态变量节点转化为状态变量节点。

(3) 应用状态变量描述的缩放和噪声计算方法。

例11.7.1 考虑如图11-21所示的滤波器。除多了5个延时外这一滤波器与图11-18所示的三阶规格化缩放格型滤波器完全相同。利用一个带有重定时变换或割集变换的2次减速变换，可以得到图11-21的信号流图。注意，因为该滤波器中的每一个节点都与一个单位延迟分支相连，所以利用状态变量描述可以计算出每个内部各节点处的信号功率或舍入噪声。因为有8个状态(1~8)，所以矩阵 **A**、**b**、**c** 和 d 的维数分别是 8×8, 8×1, 8×1 和 1×1。由图11-21，可写出状态方程为

$$
\begin{aligned}
x_1(n+1) &= 0.532x_7(n)+0.8467u(n)\\
x_2(n+1) &= 0.3695x_1(n)-0.9293x_8(n)\\
x_3(n+1) &= 0.2252x_2(n)+0.9743x_4(n)\\
x_4(n+1) &= x_3(n)\\
x_5(n+1) &= 0.0569x_1(n)+0.9984x_6(n)\\
x_6(n+1) &= 0.3209x_2(n)+0.9471x_4(n)
\end{aligned}
$$

$$
\begin{aligned}
x_7(n+1) &= 0.9293x_1(n) + 0.3695x_8(n) \\
x_8(n+1) &= -0.9743x_2(n) + 0.2252x_4(n) \\
y(n) &= 0.323x_5(n) + 0.0029u(n)
\end{aligned}
$$

从上述这些方程，可以直接得到矩阵 **A**、**b**、**c** 和 d。注意，尽管矩阵的维数已经增加，但是矩阵**A**的每一行仍最多只有两个非零元素。进一步，图11-21中的每一参数都是未做任何计算地出现在上述方程的（在例11.6.1中需要了一些计算）。事实上通过简单的观察，矩阵 **A**、**b**、**c** 和 d 就可以直接从图11-21中得出。利用**K**算法可得 $\{K_{ii} = 1, i=1, 2, \ldots, 8\}$，这意味着内部的每一节点都得到了完美的缩放。类似地，可以得到 W_{11}~W_{88} 是 {0.1455, 0.2952, 0.3096, 0.3096, 0.1043, 0.104, 0.0412, 0.1912}。因此总的舍入噪声是 $1.191\,\sigma_e^2$。注意，这里没有与节点4或状态 $x_4(n)$ 有关的舍入操作。因此，在舍入噪声的计算中 $Trace(\mathbf{W})$ 中不包含 W_{44}。 ■

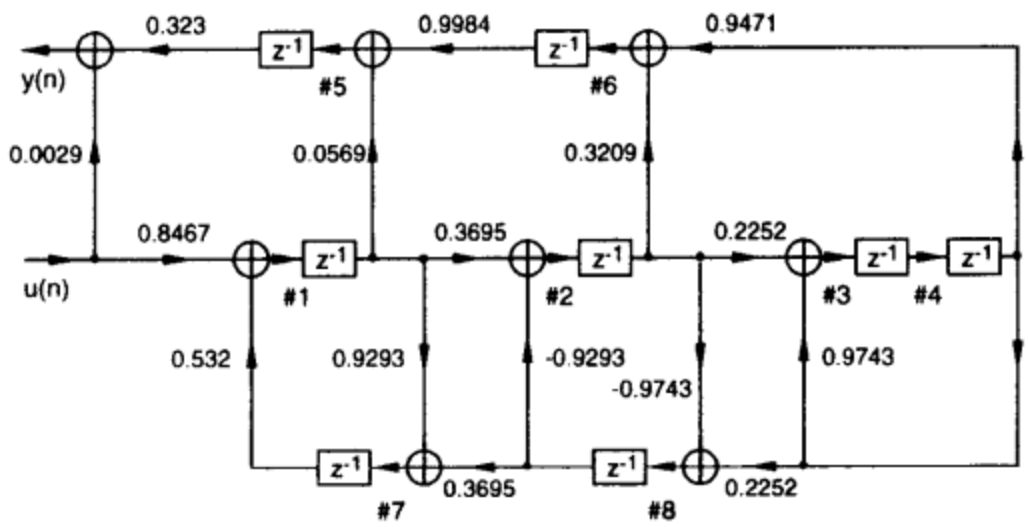


图11-21 图11-18中3阶缩放正规格型滤波器的一个变换形式

例11.7.2 考虑如图11-22所示的8抽头低通数据广播FIR滤波器，其频率响应如图11-23所示。该滤波器内部每个节点处的信号功率都可以通过状态变量描述计算。这里通过在输出信号 $y(n)$ 之前增加一个延时器引入了一个额外的状态变量。这一结构通过状态变量描述如下：

$$
\begin{aligned}
x_1(n+1) &= h_7u(n) \\
x_2(n+1) &= x_1(n) + h_6u(n) \\
x_3(n+1) &= x_2(n) + h_5u(n) \\
x_4(n+1) &= x_3(n) + h_4u(n) \\
x_5(n+1) &= x_4(n) + h_3u(n) \\
x_6(n+1) &= x_5(n) + h_2u(n) \\
x_7(n+1) &= x_6(n) + h_1u(n) \\
x_8(n+1) &= x_7(n) + h_0u(n) \\
y(n-1) &= x_8(n)
\end{aligned}
$$

从这些方程可以直接得到矩阵 **A**、**b**、**c** 和 d，且矩阵 **A**、**b**、**c** 和 d 的维数分别是 8×8, 8×1, 8×1 和 1×1。利用**K**算法和**W**算法，可得

$$\mathbf{K}=\begin{bmatrix} 0.407 & 0.322 & 0.147 & -0.002 & -0.054 & -0.034 & -0.003 & 0.007 \\ 0.322 & 0.401 & 0.324 & 0.164 & 0.029 & -0.023 & -0.017 & -0.002 \\ 0.147 & 0.324 & 0.401 & 0.320 & 0.1560 & 0.021 & -0.027 & -0.017 \\ -0.002 & 0.164 & 0.320 & 0.356 & 0.237 & 0.073 & -0.024 & -0.031 \\ -0.054 & 0.029 & 0.156 & 0.237 & 0.204 & 0.085 & -0.009 & -0.031 \\ -0.034 & -0.023 & 0.021 & 0.073 & 0.085 & 0.052 & 0.003 & -0.017 \\ -0.003 & -0.017 & -0.027 & -0.024 & -0.009 & 0.003 & 0.007 & -0.002 \\ 0.007 & -0.002 & -0.017 & -0.031 & -0.031 & -0.017 & -0.002 & 0.007 \end{bmatrix}$$

和$\mathbf{W}$，$\mathbf{W}$是一个8×8阶的单位阵 $\mathbf{I}_{8\times 8}$。因而该滤波器的舍入噪声是 $8\sigma_e^2$。矩阵$\mathbf{K}$可以用来获得如图11-24所示的缩放结构。这一缩放结构的总舍入噪声是

$$\sum_{i=1}^{8} K_{ii}W_{ii}=\sum_{i=1}^{8} K_{ii}=1.835\sigma_e^2$$

这一缩放的FIR滤波器的舍入噪声当然也可以通过计算缩放系统的$\mathbf{W}$矩阵的迹来获得。■

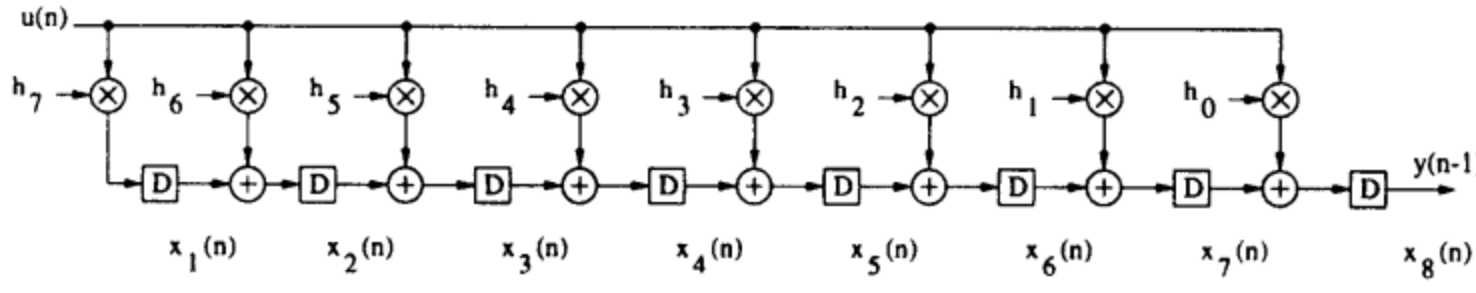


图11-22 例11.7.2中的8抽头FIR滤波器

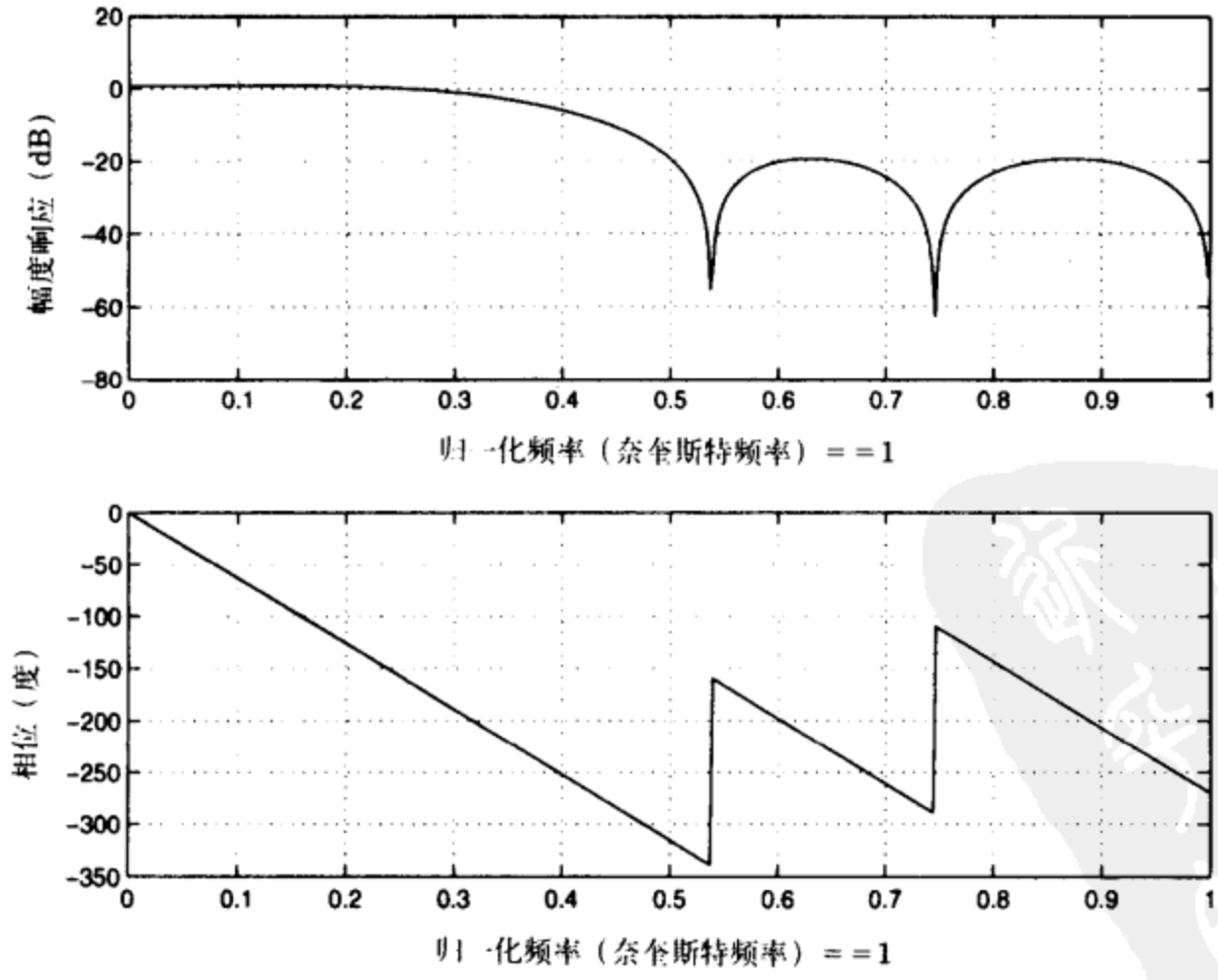


图11-23 例11.7.2中8抽头FIR滤波器的频率响应

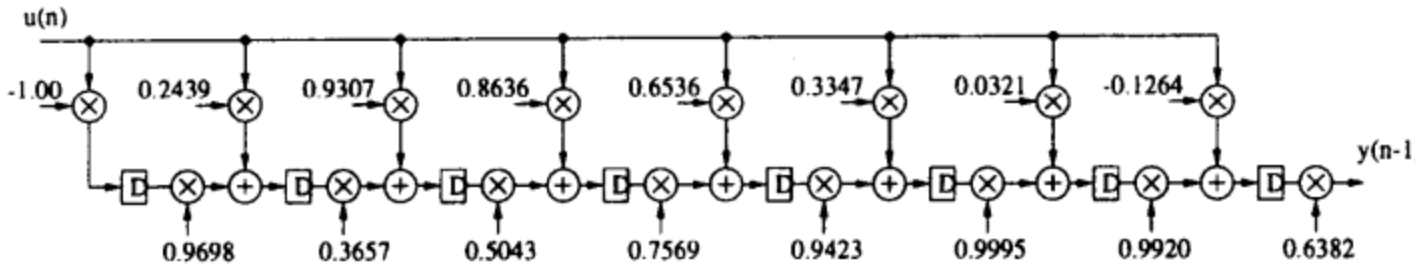


图11-24 例11.7.2中的缩放FIR滤波器

11.8 结论

基于IIR滤波器和FIR滤波器，本章我们讨论了用状态变量描述缩放噪声和舍入噪声的方法。在很多数字滤波器中，所有的舍入噪声节点是不能模拟为状态变量节点的。对这些滤波器，可以使用流水线和/或M降速变换以及重定时等进行预处理，使每一个舍入噪声节点的输出能够通过一个状态变量描述。此外，本章还提出了计算状态和输出协方差矩阵的快速算法。这些矩阵可用于缩放和对舍入噪声的计算。

11.9 习题

1. 考虑如图11-25所示的SFG，其对应的全极点传输函数是

$$H(z)=\frac{z^{-1}}{1-az^{-1}}$$

这里，为了稳定使得 $-1<a<1$。

(a) 计算输出舍入噪声。证明噪声方差 σ_e^2 通过反馈部分后被放大了，亦即证明，虽然 $e(n)$ 可能很小，但是它对 $y(n)$ 的影响可能很大。

407 ~ 410

(b) 计算(i) $a=0.1$，(ii) $a=0.5$，(iii) $a=0.998$时的舍入噪声。讨论舍入噪声和极点位置之间的关系。

(c) 对窄带滤波器，$|a|$通常非常接近1。例如，$a=0.998$是不常见的。对这样的滤波器，噪声方差 σ_e^2 将被极大地放大（那种情况下，像格型滤波器和波型数字滤波器这种低噪声/敏感滤波结构需要专门考虑）。从概念上解释为什么对这种窄带滤波器，极点位置非常靠近单位圆。

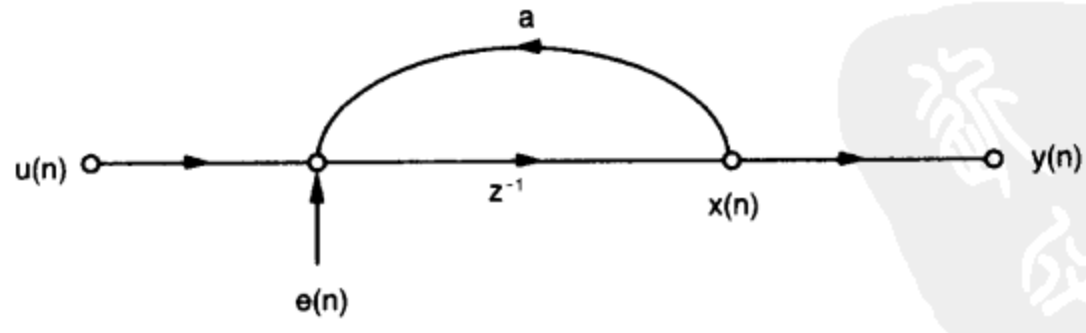


图11-25 一个全极IIR滤波器的信号流图

2. 假定图11-25中的输入信号 $u(n)$ 是一个方差为 σ_u^2，均值为零的广义静态随机白噪声过程。

(a) 计算与输入信号 $u(n)$ 对应的输出信号 $y(n)$ 的方差。

(b) 证明NSR（信噪比）是

$$\frac{N}{S}=\frac{\sigma_e^2}{\sigma_u^2}$$

(c) 由(b)，NSR显然独立于极点位置。问是什么原因导致了这样一个假的结论？提示：当a->1时，输入信号与输出信号之间的信号增益非常大。这样 $y(n)$ 可能会溢出。

3. 图11-25中的全极点滤波器可以利用 l_1 缩放法进行缩放，如图11-26所示。

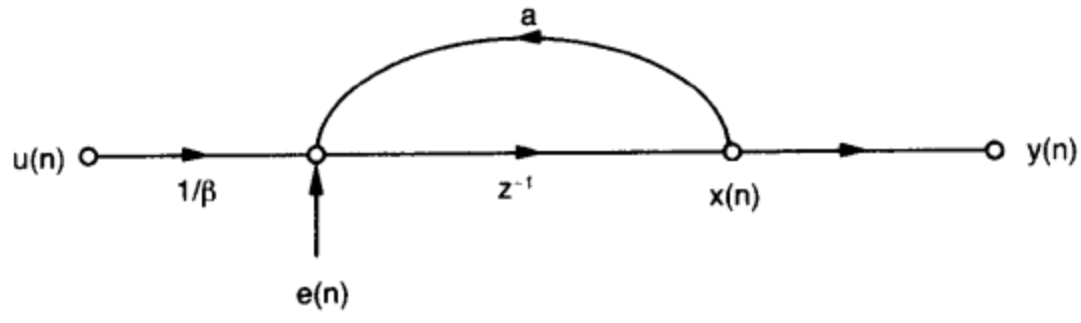


图11-26 一个缩放全极IIR滤波器的信号流图

假定输入 $u(n)$ 是一个具有均匀概率密度函数且 $|u(n)|\leqslant 1$ 的广义静态随机白噪声过程。 411

(a) 证明 $\beta>1$。（为避免用到较高成本的乘法器 $1/\beta$，可用一个最接近的2的方幂代替它或将其合并到滤波器前一级的乘法器中）。

(b) 计算输入方差。

(c) 计算输出信号的方差 σ_y^2，即无噪声情况下，$y(n)$ 的方差。

(d) 证明输出NSR是

$$\frac{N}{S}=\frac{3\sigma_e^2}{(1-|a|)^2}$$

注意，对一个给定的 a，降低这一比值的惟一方法是通过增加字长来降低 σ_e^2。

4. 对一个N阶IIR滤波器的标准状态变量表示，证明：

(a) $\mathbf{W}=\sum_{n=0}^{\infty}(\mathbf{c}^T\mathbf{A}^n)^T\mathbf{c}^T\mathbf{A}^n$

(b) $\mathbf{W}=\mathbf{A}^T\ \mathbf{W}\mathbf{A}+\mathbf{c}\mathbf{c}^T$

(c) 对一个缩放系统，$\mathbf{W}_s=\mathbf{T}^T\mathbf{W}\mathbf{T}$。

5. 证明每执行W算法中的一次迭代，公式（11-27）的求和式中所包括的项数将翻一番。

6. 考虑图11-27中与下列传输函数所对应的SFG（信号流图）

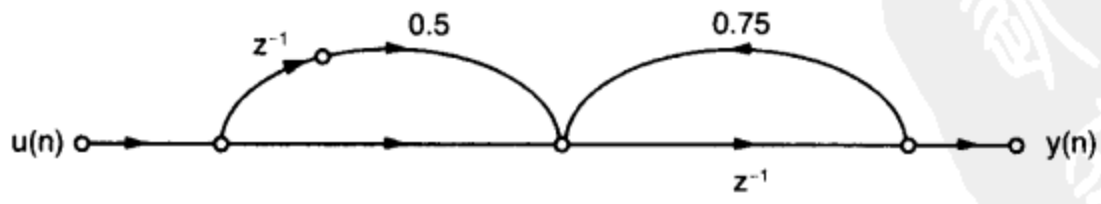


图11-27 习题6中 $H(z)$ 的信号流图

$$H(z)=\frac{z^{-1}+0.5z^{-2}}{1-0.75z^{-1}}$$

(a) 计算它的舍入噪声。

(b) 这一结构的2级流水实现可以通过如图11-28所示的两种方式实现。计算这两种结

构的舍入噪声。

(c) 重做缩放结构的问题(a)和问题(b)，这些缩放结构不是图11－28中(a)和(b) 派生的缩放结构。

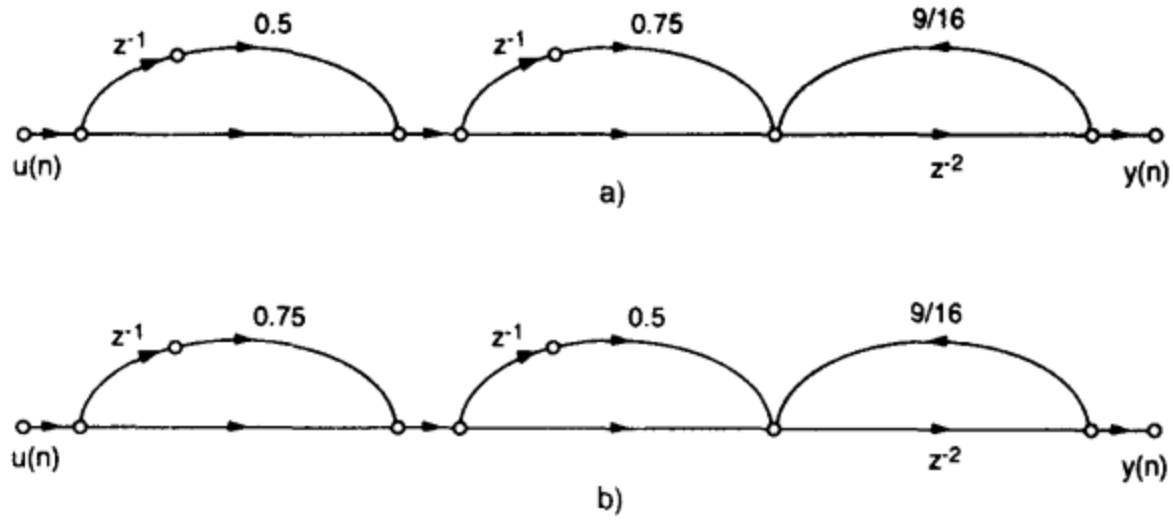


图11-28 习题6中滤波器的一个2级流水实现

7. 考虑如图11-29所示一个2阶滤波器的两种不同实现。

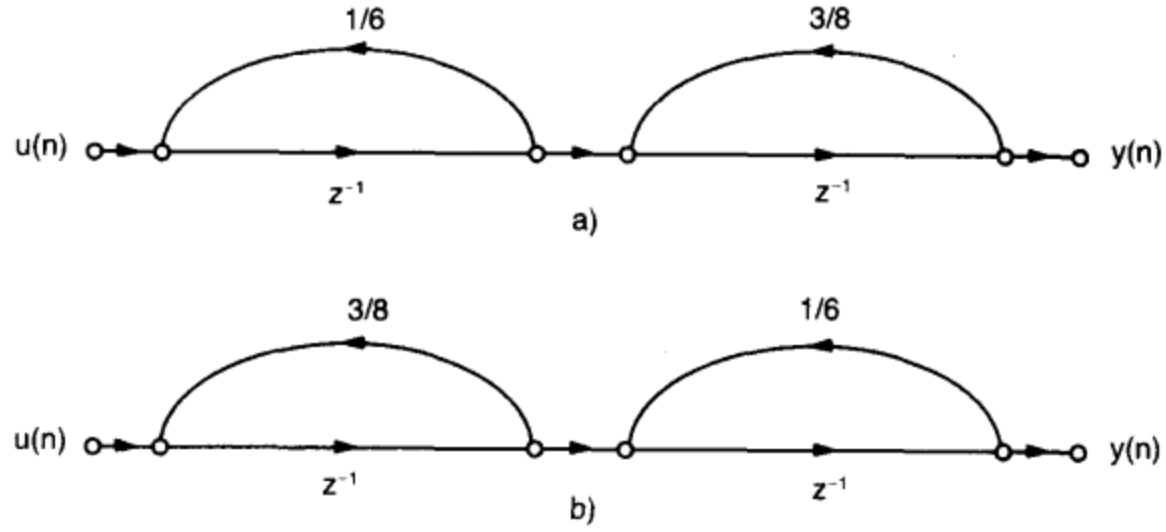


图11-29 习题7中一个2阶滤波器的两种实现

(a) 确定这两种实现在 l_2 缩放之下的滤波器结构。

412～413

(b) 计算(a)中所得两种缩放滤波器的NSR，并做比较。

8. 考虑图11-30所示的2阶滤波器。

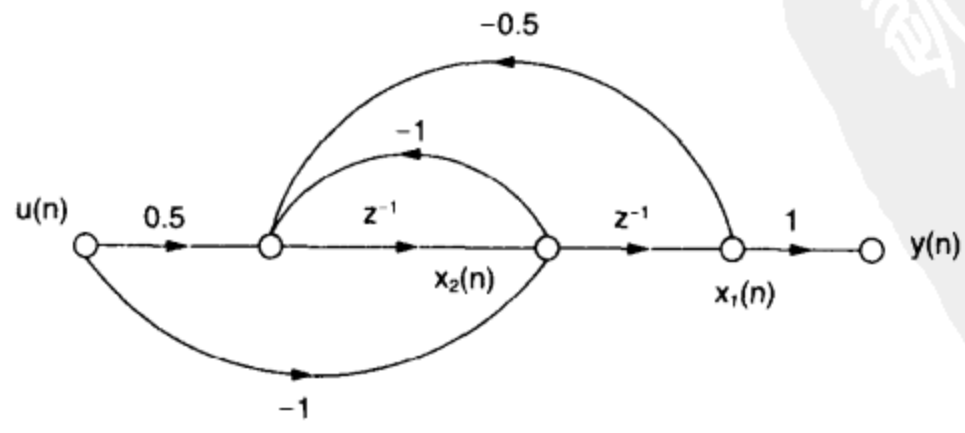


图11-30 一个直接形式的2阶滤波器

(a) 计算其传输函数。

(b) 通过求解（11-25）计算**K**。

(c) 对 l_2 缩放，选取一个对角变换矩阵**T**使 $t_i = \sqrt{K_{ii}}$ 。

(d) 画出该缩放滤波器的SFG。对这一缩放结构，通过计算矩阵**W**求输出舍入噪声。

9. 考虑图11-30所示2阶滤波器的一种2级流水实现。

(a) 应用先行离散变换求图11-30所示滤波器的一个可进行2级流水的传输函数。画出相应于该传输函数的SFG。

(b) 通过(i) **K**算法和(ii) 求解方程（11-25）的方法计算**K**。

(c) 对 l_2 缩放，选取一个对角变换矩阵**T**使 $t_i = \sqrt{K_{ii}}$ 。

(d) 画出该缩放滤波器的一个SFG。对该滤波器结构，通过计算矩阵**W**求其输出的舍入噪声。

10. 利用**K**算法计算图11-21所示滤波器结构的矩阵**K**和**W**。

11. 对式子（11-105）中关于NSR的表示式，当 $\theta=n\pi/M$、$n=0, \pm 1, \ldots, \pm M$ 时是不成立的。试给出 $M=4$ 和 $\theta=\pi/4$ 时NSR的一个可以成立的表示式。

12. 考虑用 z^{-2} 代替图11-30中的 z^{-1} 后所得到的图11-31中的SFG。

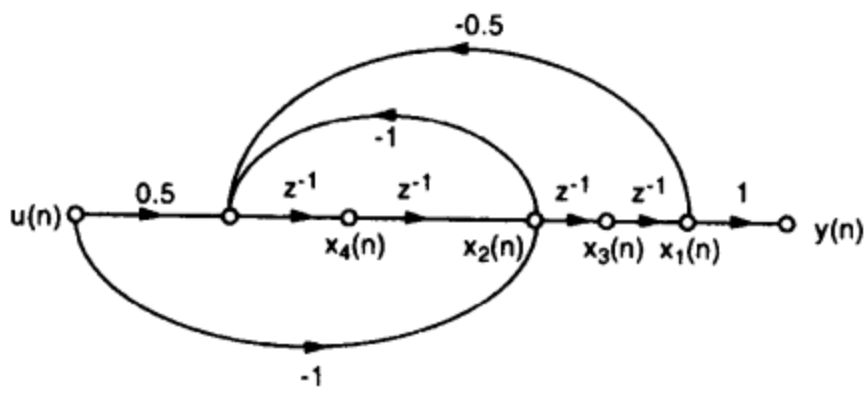


图11-31 用 z^{-2} 代替图11-30中的 z^{-1} 之后所得到的滤波器结构

(a) 证明 $K_{11} = K_{33}$、$K_{22} = K_{44}$。

(b) 求 $F_1(z)$ 和 $F_2(z)$。

(c) 用图11-30中的 $F_1(z)$ 和 $F_2(z)$ 表示图11-31中的 $F_1(z)$ 和 $F_2(z)$。

(d) 证明:

图11-30中的 K_{11}=图11-31中的 K_{11},

图11-30中的 K_{22}=图11-31中的 K_{22}。 414

13. 推导图11-30中的**W**与图11-31中**W**之间的关系。

14. 证明对图11-20中所示的原滤波器和其流水线滤波器，在计算**K**矩阵和**W**矩阵时表现了相同的有限字长特性。这里假定 $a=2$、$b=3$ 和 $c=4$。

15. 考虑如图11-32所示的滤波器结构。注意该滤波器有一个非状态变量节点。

(a) 修改图11-32中的滤波器使获得的新结构中其每一个节点都是状态变量节点且有限字长特性同原滤波器。

(b) 求修改后的滤波器的状态变量矩阵 **A**、**b**、**c** 和 d。

(c) 计算该滤波器的**K**并证明 $K=I$，从而说明该滤波器是一个正交滤波器。

(d) 验证等式 $\mathbf{AA}^T + \mathbf{bb}^T = \mathbf{I}$。

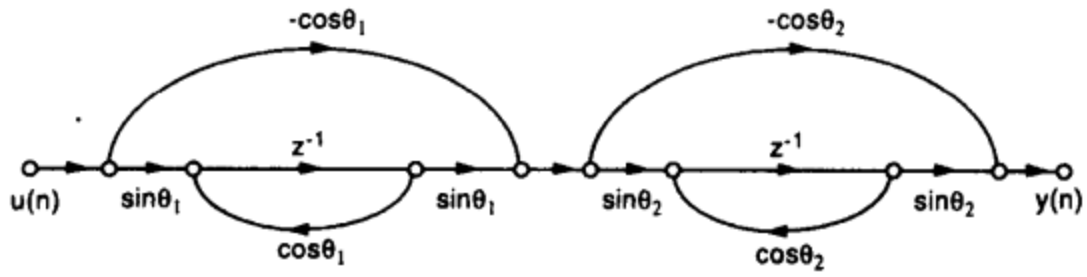
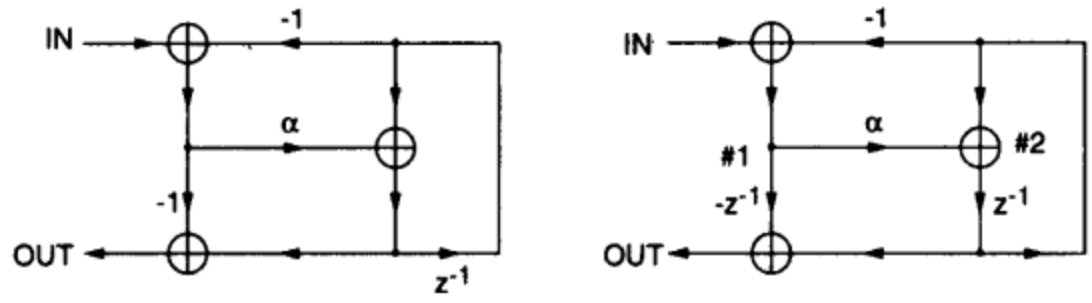


图11-32 一个具有非状态变量的正交滤波器结构

16. 考虑如图11-33所示的1阶滤波器。



a) 一个1阶滤波器　　b) 经过SRP变换后的滤波器

图 11-33

(a) 求如图11-33a所示滤波器的传输函数。输出信号功率是多少?

(b) 假定该1阶滤波器的极点位置在 $z=r$ 处。证明 $\alpha=1-r$。

(c) 证明图11-33b中的滤波器可以通过SRP变换从图11-33a获得。

(d) 计算图11-33b所示滤波器的**K**和**W**。推导极点位置 r 和该缩放滤波器NSR之间的关系。(注意，只有节点2产生舍入噪声)。

17. 考虑例11.7.2中对称8抽头低通滤波器在以下系数

$$h_0 = h_7 = -0.0807$$
$$h_1 = h_6 = 0.0203$$
$$h_2 = h_5 = 0.2118$$
$$h_3 = h_4 = 0.3897$$

和使用如图11-34所示4个乘法器之下的两种实现。假定舍入运算只在乘法或乘-加节点发生，仅有加法的操作没有舍入运算。

(a) 计算这两种结构中的舍入噪声。

(b) 用状态变量形式表示这两种结构。通过选择合适的割集，重定时如图11-34b中所示的结构使得所有舍入节点处的输出都能用状态变量模拟。利用**K**算法和**W**算法计算矩阵**K**和**W**，并给出这两种结构的缩放版本。

(c) 应用从(b)中所计算出的矩阵**K**和**W**，计算这两种缩放结构的舍入噪声。

(d) 计算(b)中所给结构的**W**矩阵，并由此计算这些结构的舍入噪声。

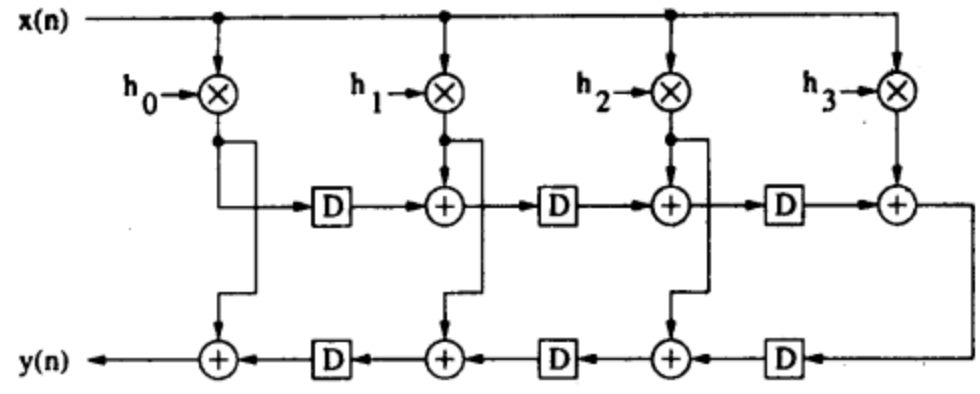
(e) 验证由(c)和(d)所得到的结果是相同的。

415 ~ 416

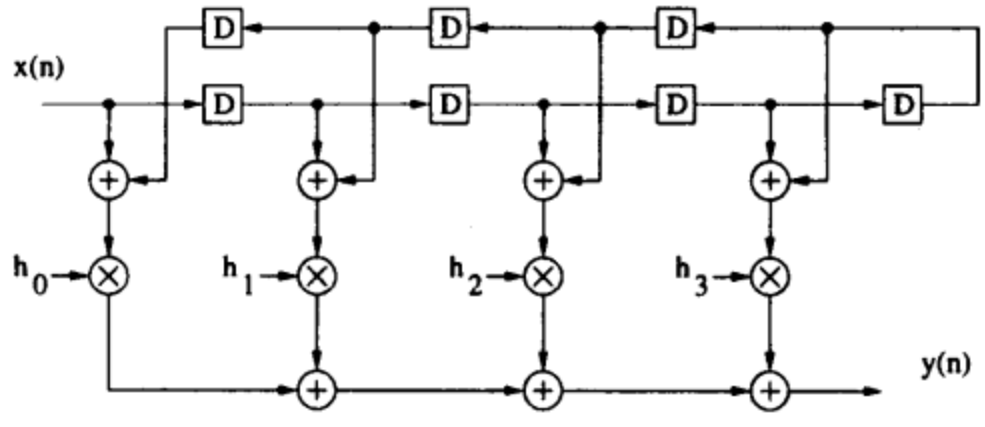
18. 考虑应用如图11-35所示结构在系数

$$h_0 = h_7 = -0.0807$$
$$h_1 = h_6 = 0.0203$$
$$h_2 = h_5 = 0.2118$$
$$h_3 = h_4 = 0.3897$$

之下所实现的8抽头低通滤波器。假定舍入运算是在1次乘-加操作之后或是在1次乘法操作紧跟2次加法操作之后进行。



a)



b)

图11-34 习题17中FIR滤波器的两种实现

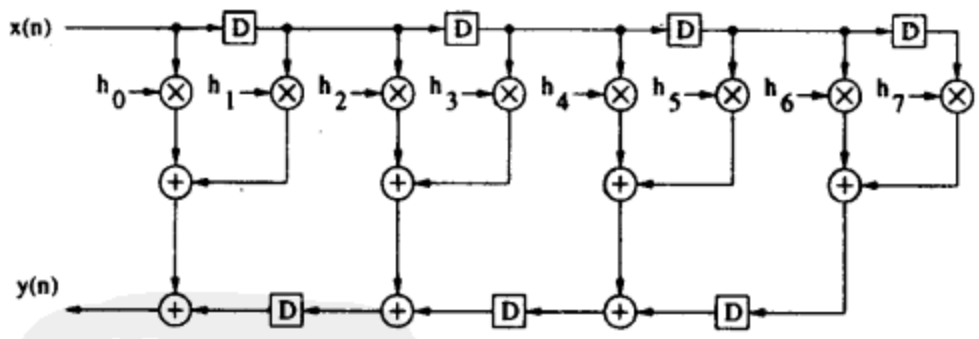


图11-35 习题18中两个FIR滤波器实现

(a) 计算该结构的舍入噪声。

(b) 应用**K**算法和**W**算法计算**K**和**W**。给出这一结构的缩放版本。计算该滤波器缩放版本的舍入噪声。

(c) 计算该缩放滤波器的**K**矩阵和**W**矩阵。

19. 考虑由以下系数所描述的8抽头低通FIR滤波器的一种低噪声2并行度实现:

$$h_0 = h_7 = -0.0807$$
$$h_1 = h_6 = 0.0203$$
$$h_2 = h_5 = 0.2118$$
$$h_3 = h_4 = 0.3897$$

使用 l_2 缩放，4种可能的并行结构如图11-36所示。当 l_2 缩放应用于这些缩放结构时，哪一种结构具有最小平均舍入噪声？这里平均舍入噪声被定义为缩放结构中两个输出舍入噪声之和的一半。利用K算法和W算法计算所有这4种结构缩放版本的平均舍入噪声。这里假定舍入只在乘-加操作中发生，加法器不产生舍入错误。此外，假定在并行结构中使用数据广播FIR子滤波器结构。

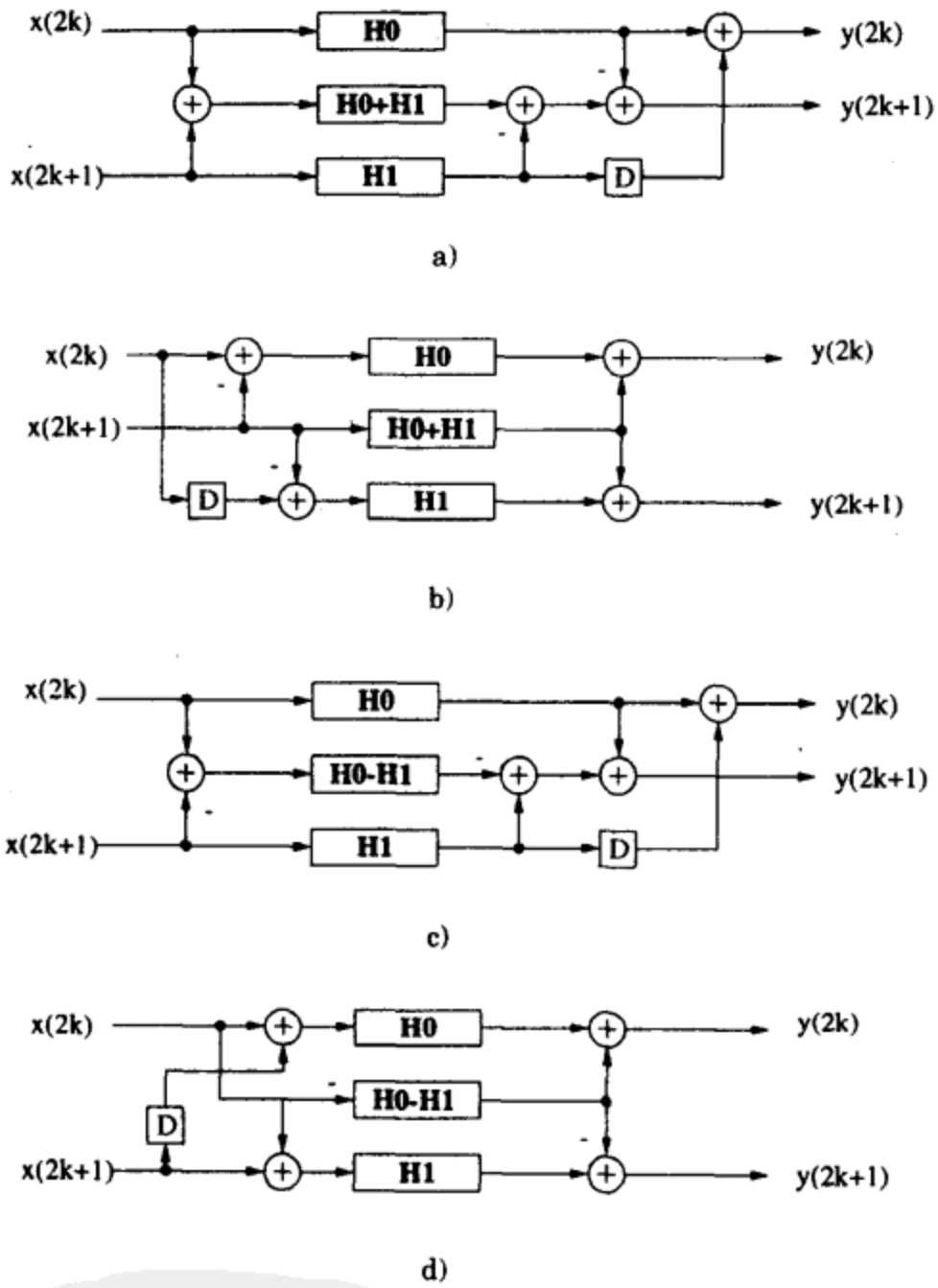


图11-36 习题19中的四种并行FIR实现

参考文献

1. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Prentice Hall, 1975.
2. P. P. Vaidyanathan, "Low-noise and low-sensitivity digital filters," in *Handbook of Digital Signal Processing* (D. F. Elliott, ed.), pp. 359–479, Academic Press, 1987.
3. R. A. Roberts and C. T. Mullis, *Digital Signal Processing*. Addison-Wesley, 1987.

4. K. H. Chang and W. G. Bliss, "Finite word-length effects of pipelined recursive digital filters," *IEEE Trans. on Signal Processing*, vol. 42, no. 8, pp. 1983–1995, Aug. 1994.

5. J. G. Chung and K. K. Parhi, *Pipelined Lattice and Wave Digital Recursive Filters*. Kluwer, 1996.

6. S. Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proc. IEEE*, vol. 72, pp. 867–884, July 1984.

7. C. E. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," in *3rd Caltech Conference on VLSI*, (Pasadena, CA), pp. 87–116, March 1983.

417 ~ 419

第12章 格型数字滤波器结构

12.1 引言

本章介绍格型数字滤波器的设计。这里给出4种格型数字滤波器：基本、单乘法器、归一化和可缩放归一化。虽然本章的重点在IIR格型滤波器的设计，但所用技术同样适用于FIR格型滤波器。FIR格型滤波器设计将只在正文的基本格型滤波器部分涉及。

最简单的IIR数字滤波器结构是直接式结构，其分子和分母的系数直接作为乘法器系数使用。不过这种结构有很高的灵敏度，因为多项式的根对系数很敏感，所以给定的传输函数的极点和零点对量化的乘法器系数也很敏感[1]。对于像低通、高通和选通之类的归一化滤波器，极点一般较多分布在靠近波段边缘的拐角里。随着靠近的极点数目增大，这种结构的灵敏度会下降。为了避免这个灵敏度问题，可以用1阶项和2阶项的和或者积实现该传输函数，即采用并行结构或级联结构。但对于复杂的小角度极点共轭问题（例如窄波段高强度转换滤波器），即使使用2阶项方法仍需要解决高灵敏度问题。

尽管如此，由于给定传输函数的分母的充分解析，格型数字滤波器有很好的数字特性。除此以外，格型滤波器还用一些规则的模块级联实现，所以这些滤波器很容易用VLSI实现。 421

格型结构是由自回归信号模型原理推出的[2]。但很多格型结构的数字特性的内涵是通过用Schur算法来推导格型结构才获得的。本章将给出多种应用Schur算法的格型数字滤波器的设计和流水线技术，以及一种利用多项式展开算法(polynomial expansion algorithm)和转置定理（transposition theorem）计算舍入噪声的方法。

12.2 Schur算法

Schur算法最初用于验证幂级数在单位圆内是否解析且有界[3]。如果一个N阶多项式$\Phi_N(z)$的所有零点都在单位圆内，Schur算法可以产生$N+1$个多项式$\{\Phi_i(z), i=N, N-1, \ldots, 0\}$。Schur算法最重要性质之一是这$N+1$个多项式两两正规直交，可作为直交基函数展开任何$N$阶多项式。Schur算法的这种正规直交性目前已经用于各种格型滤波器综合。

为了说明Schur算法的正规直交性，这里给出一个基于滤波器结构内部节点幂计算的内积公式。虽然内积公式的定义中包含复分析，考虑到实用性，在内积求值过程中不需要进行复分析。

12.2.1 Schur多项式的计算

Schur多项式是指在单位圆边界或外部没有零点的多项式。所以，一个稳定的IIR滤波器的分母属于Schur多项式。定义一个零点都在单位圆内部的实多项式如下：

$$D_N(z) = \sum_{i=0}^{N} d_i z^i \tag{12-1}$$

再初始化N阶Schur多项式 $\Phi_N(z)$ 如下:

$$\Phi_N(z) = D_N(z) \tag{12-2}$$

其中:

422

$$\Phi_N(z) = \sum_{i=0}^{N} \phi_i z^i \tag{12-3}$$

根据 $\Phi_N(z)$，构造多项式 $\Phi_{N-1}(z)$ 如下:

$$\begin{aligned} \Phi_{N-1}(z) &= \frac{z^{-1}\{\phi_N \Phi_N(z) - \phi_0 \Phi_N^*(z)\}}{\sqrt{\phi_N^2 - \phi_0^2}} \\ &= \frac{z^{-1}\{\Phi_N(z) - k_N \Phi_N^*(z)\}}{\sqrt{1 - k_N^2}} \end{aligned} \tag{12-4}$$

其中 $k_N = \Phi_0 / \Phi_N$，而 $\Phi_N^*(z)$ 是 $\Phi_N(z)$ 的逆多项式，定义如下:

$$\Phi_N^*(z) = z^N \Phi_N(z^{-1}) \tag{12-5}$$

请注意这里 $\Phi_{N-1}(z)$ 的次数比 $\Phi_N(z)$ 少1。利用由 $\Phi_N(z)$ 生成 $\Phi_{N-1}(z)$ 的相同方法可以依次由 $\Phi_{N-i+1}(z)$ 生成所有的 $\Phi_{N-i}(z)$。因为 $\Phi_N(z)$ 是Schur多项式，这要求在多项式组 $\{\Phi_N(z), \Phi_{N-1}(z), \ldots, \Phi_1(z)\}$ 中每个多项式常数项的绝对值小于 z 最高次幂系数的绝对值。

可以看出 $\Phi_{N-1}(z)$ 的由低到高 z 的各次幂的系数是N个2×2矩阵行列式值的 $1/\sqrt{\Phi_N^2 - \Phi_0^2}$ 倍，这些矩阵由下面给出的矩阵的第一列依次和其后每一列组成:

$$\begin{bmatrix} \phi_N & \phi_{N-1} & \phi_{N-2} & \cdots & \phi_2 & \phi_1 & \phi_0 \\ \phi_0 & \phi_1 & \phi_2 & \cdots & \phi_{N-2} & \phi_{N-1} & \phi_N \end{bmatrix} \tag{12-6}$$

公式（12-6）所示的Schur算法的矩阵表示对流水线操作的格型滤波器很有用处，这将在12.8节中介绍。

例12.2.1 分析如下的3阶多项式:

$$D_3(z) = 345.1z^3 - 819.3z^2 + 665.8z - 183.6$$

应用Schur算法可以得到:

$$\begin{aligned} \Phi_3(z) &= 345.1z^3 - 819.3z^2 + 665.8z - 183.6 \\ \Phi_2(z) &= 292.2072z^2 - 549.2662z + 271.5337 \\ \Phi_1(z) &= 107.956z - 105.1841 \\ \Phi_0(z) &= 24.3064 \end{aligned}$$

对于多项式组 $\{\Phi_3(z), \Phi_2(z), \Phi_1(z)\}$ 的每一个多项式，z 的最高次幂的系数都大于常数。因此，得出 $D_3(z)$ 是Schur多项式的结论。 ■

423

12.2.2 Schur多项式的正规直交性

分析如图12-1所示的滤波器结构，它实现了一个表示因果关系稳定的实传输函数（a real, causal, stable transfer function）$H(z) = N_N(z) / D_N(z)$。如果多项式和节点 p 相关，

且输入的节点分别为 $P(z)$ 和 $D_N(z)$，则从输入端到节点 p 的传输函数为 $P(z)/D_N(z)$。如果输入信号是单位功率的模拟未知随机信号，节点 p 的平均功率为:

$$\begin{aligned} P_p &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\left|\frac{P(e^{j\omega})}{D_N(e^{j\omega})}\right|^2 \mathrm{d}w \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\frac{P(e^{j\omega})P(e^{-j\omega})}{D_N(e^{j\omega})D_N(e^{-j\omega})}\mathrm{d}w \end{aligned} \tag{12-7}$$

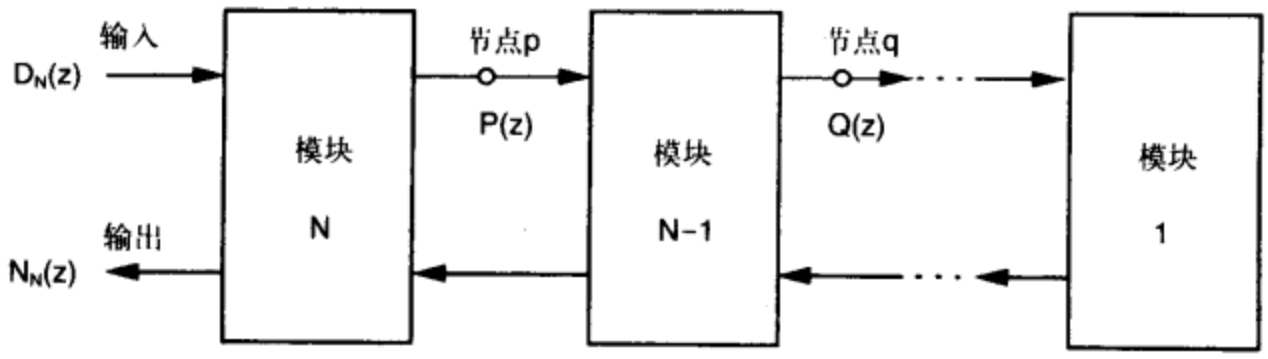


图12-1 实现 $H(z)=N_N(z)/D_N(z)$ 的滤波器结构

对于单位圆，也就是 $z=\mathrm{e}^{jw}$ 有:

$$\mathrm{d}z = je^{j\omega}\mathrm{d}\omega \tag{12-8}$$

则平均功率可在 z 域计算如下:

$$P_p = \frac{1}{2\pi j}\oint_C \frac{P(z)P(z^{-1})z^{-1}}{D_N(z)D_N(z^{-1})}\mathrm{d}z \tag{12-9}$$

其中C为一个逆时针回路。请注意这里可将该回路作为单位圆，因为 $D_N(z)$ 的所有零点均在此单位圆内部。应用公式（12-9）的平均功率，定义图12-1中2个内部多项式 $P(z)$ 和 $Q(z)$ 的内积为:

$$\langle P(z),Q(z)\rangle = \frac{1}{2\pi j}\oint_C \frac{P(z)Q(z^{-1})z^{-1}}{D_N(z)D_N(z^{-1})}\mathrm{d}z \tag{12-10}$$

现在公式（12-9）中的平均功率可这样计算: $P_p=\langle P(z),P(z)\rangle$。为了证明公式（12-10）的内积定义正确，必须证明其满足下面3个性质: 424

(1) 共轭对称性:

$$\langle P(z),Q(z)\rangle = \langle Q(z),P(z)\rangle^*$$

其中（*）表示复共轭操作。本章中，$P(z)$ 和 $Q(z)$ 假设为仅包含实系数，所以有:

$$\langle P(z),Q(z)\rangle = \langle Q(z),P(z)\rangle$$

(2) 线性: 对于任意实常数 α、β、γ 有:

$$\langle \alpha P(z),\beta Q(z)+\gamma R(z)\rangle = \alpha\beta\langle P(z),Q(z)\rangle + \alpha\gamma\langle P(z),R(z)\rangle.$$

(3) 正定性:

$$\langle P(z),P(z)\rangle \geqslant 0$$

其中当 $P(z)=0$ 时取等号。

很容易验证公式（12-10）所示的内积满足这3个性质。内积的可交换性，即 $\langle P(z),Q(z)\rangle=\langle Q(z),P(z)\rangle$ 证明如下:

$$\begin{aligned}\langle P(z),Q(z)\rangle &= \frac{1}{2\pi j}\oint_{C_0}\frac{P(z)Q(z^{-1})z^{-1}}{D_N(z)D_N(z^{-1})}\mathrm{d}z && (12\text{-}11)\\ &= \frac{-1}{2\pi j}\oint_{C_1}\frac{P(t^{-1})Q(t)t}{D_N(t)D_N(t^{-1})t^2}\mathrm{d}t \quad (t=z^{-1}) && (12\text{-}12)\\ &= \frac{1}{2\pi j}\oint_{C_0}\frac{Q(t)P(t^{-1})t^{-1}}{D_N(t)D_N(t^{-1})}\mathrm{d}t && (12\text{-}13)\\ &= \langle Q(z),P(z)\rangle && (12\text{-}14)\end{aligned}$$

在前面的等式中，C_0 表示一个包含原点的逆时针回路，C_1 表示顺时针回路。

回路积分常用柯西余数定理很容易计算出来[1]，[4]:

$$\frac{1}{2\pi j}\oint_C X(z)\mathrm{d}z=\sum[C\text{内极点处}X(z)\text{的数}] \tag{12-15}$$

任意有理函数 $X(z)$ 可以表示如下:

$$X(z)=\frac{\Theta(z)}{(z-d_0)^s} \tag{12-16}$$

425 其中 $X(z)$ 在 $z=d_0$ 点有 s 个极点且 $\Theta(z)$ 在 $z=d_0$ 点没有极点。然后，给出 $X(z)$ 在 $z=d_0$ 点的微分:

$$\mathrm{Res}[X(z)\text{ at }z=d_0]=\frac{1}{(s-1)!}\left[\frac{d^{s-1}\Theta(z)}{dz^{s-1}}\right]_{z=d_0} \tag{12-17}$$

举例如下，如果 $X(z)=z^{-i}$,

$$\frac{1}{2\pi j}\oint_C z^{-i}\mathrm{d}z=\begin{cases}1 & \text{如果 } i=1,\\ 0 & \text{其他情况。}\end{cases} \tag{12-18}$$

然后令 $D_N(z)=\Phi_N(z)$:

$$\begin{aligned}\langle\Phi_N(z),z^i\rangle &= \frac{1}{2\pi j}\oint_C\frac{\Phi_N(z)z^{-i}}{\Phi_N(z)\Phi_N(z^{-1})z}\mathrm{d}z\\ &= \frac{1}{2\pi j}\oint_C\frac{z^{N-i-1}}{\Phi_N^*(z)}\mathrm{d}z\\ &= \begin{cases}1/\phi_N & \text{当 } i=N\\ 0 & \text{当 } i<N-1\end{cases} && (12\text{-}19)\end{aligned}$$

请注意 $\Phi_N^*(z)=z^N\Phi_N(z^{-1})$ 指逆Schur多项式。在公式（12-19）的推导中，应用了 $\Phi_N^*(z)$ 的所有根在单位圆外部的结论。类似地，应用公式（12-3）和公式（12-5）:

$$\begin{aligned}\langle\Phi_N^*(z),z^i\rangle &= \langle z^i,\Phi_N^*(z)\rangle\\ &= \frac{1}{2\pi j}\oint_C\frac{z^iz^{-N}\Phi_N(z)}{\Phi_N(z)\Phi_N(z^{-1})z}\mathrm{d}z\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2\pi j}\oint_C \frac{z^{i-1}}{\Phi_N^*(z)}\mathrm{d}z \\
&= \begin{cases} 1/\phi_N & \text{当 } i=0 \\ 0 & \text{当 } i \geqslant 1 \end{cases}
\end{aligned} \tag{12-20}
$$

应用公式（12-19）和公式（12-20），得到以下两个有用的关系式:

$$\langle \Phi_N(z), P(z)\rangle = p_N/\phi_N \tag{12-21}$$

$$\langle \Phi_N^*(z), P(z)\rangle = p_0/\phi_N \tag{12-22}$$

其中 p_N 是 N 阶多项式 $P(z)$ 的 z^N 项的系数，p_0 是 $P(z)$ 的常数项。简单应用公式（12-21）和（12-22），可以得到以下关系式:

$$\langle \Phi_N(z), \Phi_N(z)\rangle = \phi_N/\phi_N = 1 \tag{12-23}$$

$$\langle \Phi_N^*(z), \Phi_N^*(z)\rangle = \phi_N/\phi_N = 1 \tag{12-24}$$

$$\langle \Phi_N(z), \Phi_N^*(z)\rangle = \phi_0/\phi_N \tag{12-25}$$

426

根据公式（12-21）和公式（12-22），由公式（12-10）很容易得到以下两个有用的等式:

$$\langle z^{-j}\Phi_N(z), z^i\rangle = \langle \Phi_N(z), z^{i+j}\rangle \tag{12-26}$$

$$\langle z^{-j}\Phi_N^*(z), z^i\rangle = \langle \Phi_N^*(z), z^{i+j}\rangle \tag{12-27}$$

根据内积的性质，可以看出Schur多项式满足如下的正规直交条件:

$$\langle \Phi_i(z), \Phi_j(z)\rangle = \begin{cases} 1, & i=j \\ 0, & i\neq j \end{cases} \tag{12-28}$$

其中 $0 \leqslant i, j \leqslant N$。（公式（12-28）的详细证明可见附录D）。

和Schur多项式相反，逆Schur多项式不互相直交。如例题:

$$\langle \Phi_N^*(z), \Phi_i^*(z)\rangle = \Phi_i^*(0)/\phi_N \neq 0, \quad \text{当 } 0 \leqslant i \leqslant N-1 \tag{12-29}$$

因为 $\Phi^*_i(z)$ 的常数一般非零。不过，在满足特定条件时，逆Schur多项式也可以作为正规直交基函数。分析内积:

$$
\begin{aligned}
\langle \Phi_i^*(z), z^{i-j}\Phi_j^*(z)\rangle &= \langle z^{j-i}\Phi_j^*(z^{-1}), \Phi_i^*(z^{-1})\rangle \quad \text{（由于共轭对称性）} \\
&= \langle z^j\Phi_j^*(z^{-1}), z^i\Phi_i^*(z^{-1})\rangle \\
&= \langle \Phi_j(z), \Phi_i(z)\rangle \\
&= 0
\end{aligned} \tag{12-30}
$$

其中 $0 \leqslant j < i \leqslant N$。公式（12-30）的关系称为*逆Schur多项式的 $(i-j)$ 阶正规直交性*。所以，多项式 $\{\Phi_N^*(z),\ z\Phi_{N-1}^*(z),\ z^2\Phi_{N-2}^*(z),\ \ldots, z^N\Phi_0(z)\}$ 组成一组正规直交基。逆Schur多项式的 $(i-j)$ 阶正规直交性在讨论各种格型滤波器结构时将会用到。

12.2.3 多项式展开算法

多项式组 $\{\Phi_N(z),\ \Phi_{N-1}(z),\ \ldots,\Phi_0(z)\}$ 中的Schur多项式两两正规直交，因此可以作为正规直交基函数。所以，任意一个N阶多项式 $N_N(z)$ 可以用这些多项式展开为:

$$N_N(z) = \sum_{i=0}^{N} c_i \Phi_i(z) \tag{12-31}$$

427 其中的 c_i，i由$i = 0$到N，称为展开系数。这些扩展系数可用如下的多项式展开算法算出[5]:

多项式展开算法

- 对于任意m阶的多项式 $N_m(z)$ $(0 < m \leqslant N)$，初始化 $Q(z) = N_m(z)$，且对于 $m < i \leqslant N$，令 $c_i = 0$;
- 对于 $i = m, m-1, \ldots, 0$,
  (a) 根据公式 $c_i = Q^*(0) / \Phi_i^*(0)$ 计算 c_i;
  (b) 根据公式 $Q(z) = Q(z) - c_i \Phi_i(z)$ 更新 $Q(z)$。

在上面的算法中，$Q^*(z)$ 和 $\Phi_i^*(z)$ 分别是 $Q(z)$ 和 $\Phi_i(z)$ 的逆多项式。因此 $Q^*(0)$ 和 $\Phi_i^*(0)$ 是分别由 $Q(z)$ 和 $\Phi_i(z)$ 产生的系数。

例12.2.2 考虑展开如下的3阶多项式:

$$N_3(z) = z^3 + 3z^2 + 3z + 1$$

应用例12.2.1得到的Schur多项式 $\{\Phi_3(z), \ldots, \Phi_0(z)\}$。展开所需的每一步总结如下:

(1) 初始化 $Q(z) = N_3(z) = z^3 + 3z^2 + 3z+1$,
然后令 $c_3 = 1/345.1 = 0.0029$;

(2) $Q(z)$ 更新得到 $Q(z) = 5.3741z^2 + 1.0707z + 1.532$,
然后令 $c_2 = 5.3741/292.2072 = 0.0184$;

(3) $Q(z)$ 更新得到 $Q(z) = 11.1725z - 3.4619$,
然后令 $c_1 = 11.1725/107.956 = 0.1035$;

(4) $Q(z)$ 更新得到 $Q(z) = 7.4238$,
然后令 $c_0 = 7.4238/24.3064 = 0.3054$。 ■

在格型滤波器中，分母应用Schur算法来分析；分子部分则用以Schur算法对分母处理的结果为正规直交基函数的多项式展开算法进行分析。

12.2.4 应用Schur算法的功率计算

仍然考虑图12-1所示的滤波器结构。如果输入信号是单位功率的模拟随机未知信号，那么对公式（12-10）应用内积定义，节点 p 的平均功率可以写成:

428

$$P_p = \langle P(z), P(z) \rangle \tag{12-32}$$

因为分母 $D_N(z)$ 为Schur多项式，$P(z)$ 可用多项式展开算法表示如下:

$$P(z) = \sum_i c_i \Phi_i(z) \tag{12-33}$$

将公式（12-33）代入公式（12-32）:

$$P_p = \left\langle \sum_i c_i \Phi_i(z), \sum_i c_i \Phi_i(z) \right\rangle \tag{12-34}$$

然后，由于内积的线性以及Schur算法的正规直交性，公式（12-34）化简为:

$$P_p = \sum_i c_i^2 \tag{12-35}$$

因此内部节点或者与 $P(z)$ 相关的输出节点的平均信号功率可简单地通过计算 $P(z)$ 展开系数的平方和得出。

例12.2.3 分析如下的滤波器输出信号功率的计算:

$$H(z) = \frac{z^3 + 3z^2 + 3z + 1}{345.1z^3 - 819.3z^2 + 665.8z - 183.6}$$

根据例12.2.2，用多项式展开算法计算出的展开系数为{0.0029，0.0184，0.1035，0.3054}。然后对展开系数的平方求和得到，输出信号功率为0.1043。 ■

12.3 基本数字格型滤波器

数字格型滤波器由规则模块构成。如果滤波器为N阶，则需要N个模块。实现某个传输函数的格型滤波器可以看作有下列性质的滤波器：分母多项式输入滤波器，在输出节点可以得到分子多项式。这个分析过程本质上基于多项式次数缩减，也就是说每经过一个模块，这些多项式的次数将递减一。经过N个模块之后，多项式为0次，即常数。通过匹配这些常数，就可用适当的乘法器简单地把它们连接起来。

本节将给出两种略有不同的基本格型滤波器结构。第一种来自Schur多项式的应用，第二种来自逆Schur多项式的应用。 429

12.3.1 基本格型滤波器的导出

应用次数缩减过程得到Schur多项式:

$$\Phi_{i-1}(z) = \frac{z^{-1}\{\Phi_i(z) - k_i\Phi_i^*(z)\}}{s_i} \tag{12-36}$$

其中 s_i 是任意非零缩放因数且:

$$k_i = \Phi_i(0)/\Phi_i^*(0) \tag{12-37}$$

如果选取:

$$s_i = \sqrt{1 - k_i^2} \tag{12-38}$$

则Schur多项式 $\{\Phi_N(z), \Phi_{N-1}(z), \ldots, \Phi_0(z)\}$ 两两直交，可以用于设计归一化或可缩放归一化格型滤波器。另一方面，基本格型滤波器设计时选择 s_i 如下:

$$s_i = 1 - k_i^2 \tag{12-39}$$

本章中，用公式（12-39）选择的第 i 阶Schur多项式用 $\psi_i(z)$ 表示，而用公式（12-38）选择的则用 $\Phi_i(z)$ 表示。

考虑一个N阶IIR传输函数 $H_N(z) = N_N(z)/D_N(z)$。现在，初始化N阶Schur多项式 $\psi_N(z)$ 为 $\psi_N(z) = D_N(z)$，其中:

$$\Psi_N(z)=\sum_{i=0}^{N}\psi_i z^i \tag{12-40}$$

根据 $\psi_N(z)$，用次数缩减过程构成多项式 $\psi_{N-1}(z)$ 为:

$$\Psi_{N-1}(z)=\frac{z^{-1}\{\Psi_N(z)-k_N\Psi_N^*(z)\}}{1-k_N^2} \tag{12-41}$$

同时，$\psi^*_{N-1}(z)$，即 $\psi_{N-1}(z)$ 的逆多项式定义如下:

$$\Psi_{N-1}^*(z)=z^{N-1}\Psi_{N-1}(z^{-1})=\frac{\Psi_N^*(z)-k_N\Psi_N(z)}{1-k_N^2} \tag{12-42}$$

根据公式（12-42），

$$\Psi_N^*(z)=(1-k_N^2)\Psi_{N-1}^*(z)+k_N\Psi_N(z) \tag{12-43}$$

应用公式（12-41）和公式（12-43），$\psi_{N-1}(z)$ 可表示为 $\psi_N(z)$ 和 $\psi^*_{N-1}(z)$ 的函数:

430

$$\Psi_{N-1}(z)=z^{-1}\{\Psi_N(z)-k_N\Psi_{N-1}^*(z)\} \tag{12-44}$$

同时，根据公式（12-43）和公式（12-44）

$$\begin{aligned}\Psi_N^*(z) &= \Psi_{N-1}^*(z)+k_N\{\Psi_N(z)-k_N\Psi_{N-1}^*(z)\}\\ &= \Psi_{N-1}^*(z)+k_N z\Psi_{N-1}(z)\end{aligned} \tag{12-45}$$

公式（12-44）和公式（12-45）的一种实现如图12-2a所示。通过以 $\psi_{i-1}(z)$ 替换 $\psi_i(z)$ 重复进行前面的过程，对于 $i=N, N-1, \ldots, 1$，分母 $D_N(z)$ 可分解，如图12-2所示。这种结构可用来实现传输函数为 $\psi_0(z)/\psi_N(z)$ 的全极点滤波器或是传输函数为 $\psi^*_N(z)/\psi_N(z)$ 的全通滤波器。

根据公式（12-36)~(12-39），$\psi_i(z)$ 和 $\Phi_i(z)$ 关系如下:

$$\begin{aligned}\Psi_N(z) &= \Phi_N(z)\\ \Psi_i(z) &= \frac{\Phi_i(z)}{\sqrt{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}} \quad 0\leqslant i<N\end{aligned} \tag{12-46}$$

请注意 $\psi_i(z)$ 和 $\Phi_i(z)$ 只相差一个缩放因数。因此，由 $\psi_i(z)$ 计算出来的所有参数 k 和由 $\Phi_i(z)$ 计算出来的结果相等:

$$k_i=\Psi_i(0)/\Psi_i^*(0)=\Phi_i(0)/\Phi_i^*(0) \tag{12-47}$$

同时，由公式（12-46）很容易看出 $\{\psi_N(z), \psi_{N-1}(z), \ldots, \psi_0(z)\}$ 直交但非正规直交，因为:

$$\begin{aligned}\langle\Psi_i(z),\Psi_i(z)\rangle &= \langle\Psi_i^*(z),\Psi_i^*(z)\rangle,\\ &= \frac{1}{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}\end{aligned} \tag{12-48}$$

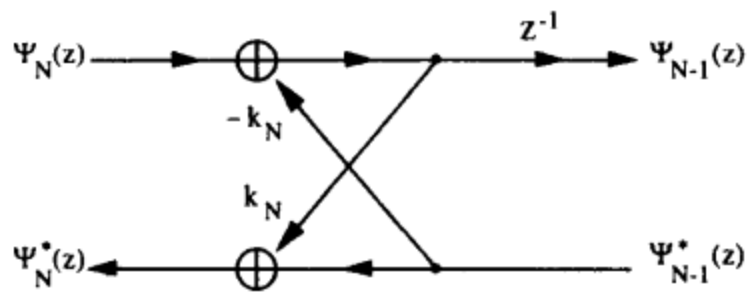
根据公式（12-48），可看出 $\langle\psi_i(z), \psi_i(z)\rangle$ 随着 i 减小而增大，因为所有参数 k 都小于1。当多数参数 k 近似为1时，图12-2b所示的节点功率差将很大，为了避免在临界节点溢出，需要对输入收缩较大的倍数。这样的话，舍入噪声影响将会显著增加。不过，在归一化格型

滤波器中并不需要进行输入缩放，因为反馈部分的所有节点都具有相同的功率，这在本章后面将可看到。

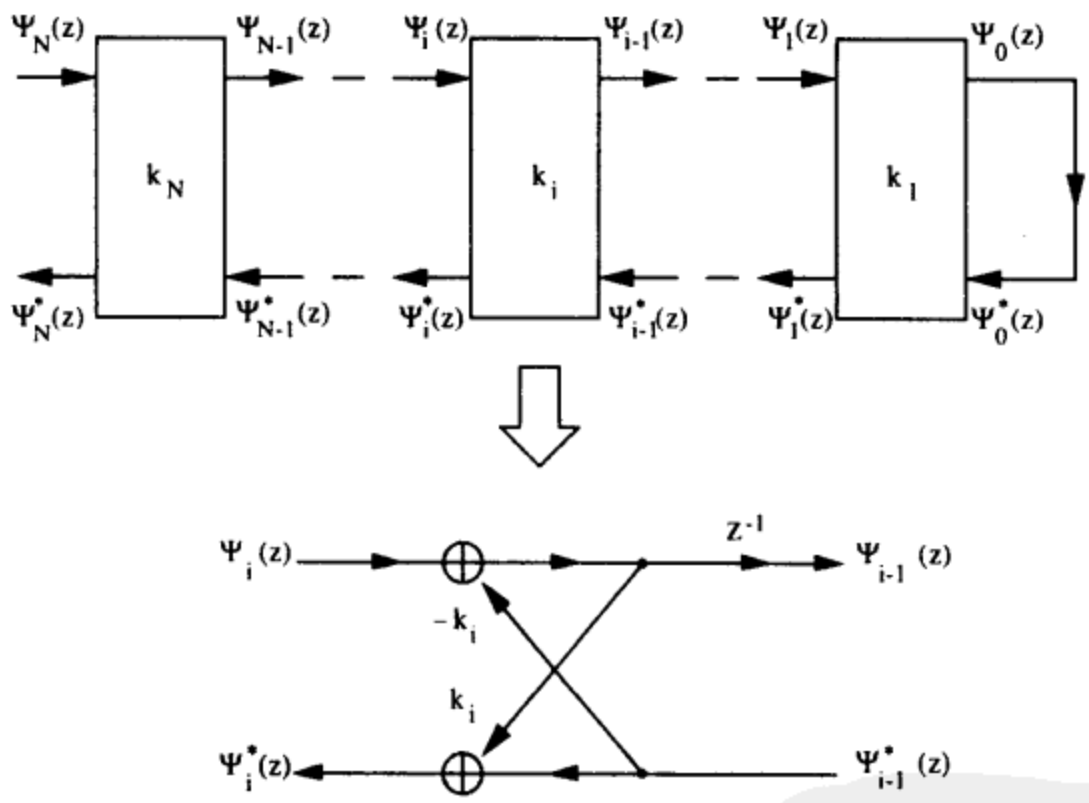
因为图12-2b上面一行的多项式 $\{\psi_N(z), \psi_{N-1}(z), \ldots, \psi_0(z)\}$ 是直交的，可以以它们为基函数，用多项式展开算法展开分子 $N_N(z)$ 为:

$$N_N(z) = \sum_{i=0}^{N} c_i \Psi_i(z) \tag{12-49}$$

431



a) 公式（12-44）和公式（12-45）的实现



b) 分母 $D_N(z)$ 的实现

图 12-2

432

通过把这些逐轮获取的参数 $\{c_N, c_{N-1}, \ldots, c_0\}$ 和图12-2b的全极点部分联系起来，可以得到如图12-3所示的基本格型滤波器结构。

例12.3.1 分析3阶巴特沃斯低通滤波器:

$$H(z) = \frac{(1+z)^3}{345.1z^3 - 819.3z^2 + 665.8z - 183.6}$$

根据公式（12-40）和公式（12-41），

$$
\begin{aligned}
\Psi_3(z) &= 345.1z^3 - 819.3z^2 + 665.8z - 183.6 \\
\Psi_2(z) &= 345.1z^2 - 648.6896z + 320.6843 \\
\Psi_1(z) &= 345.1z - 336.2392 \\
\Psi_0(z) &= 345.1
\end{aligned}
$$

所以求得参数 k 如下:

$$k_3 = -0.532,\ k_2 = 0.9293,\ \text{及}\ \ k_1 = -0.9743$$

由公式(12-49)，使用 $\psi_i(z)$ 求得的展开系数为:

$$c_3 = 0.0029,\ c_2 = 0.0156,\ c_1 = 0.0324,\ \text{及}\ \ c_0 = 0.0215$$

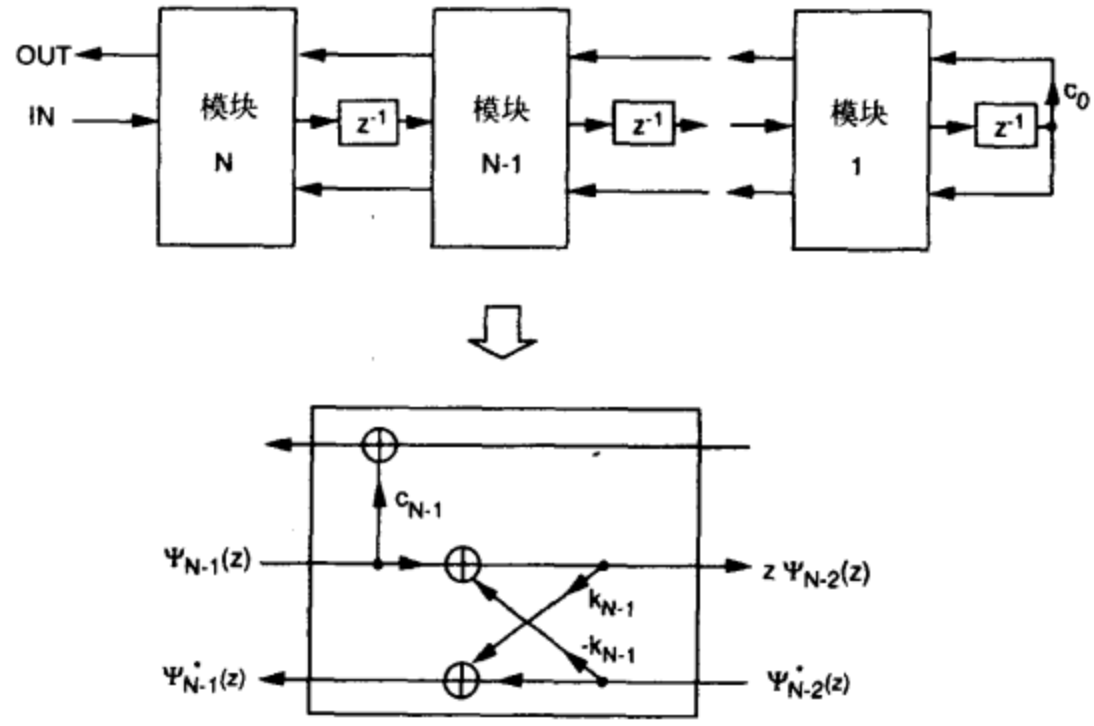


图12-3 一个 N 阶基本格型滤波器的结构

433 图12-4给出了应用Schur多项式分解的基本格型滤波器。

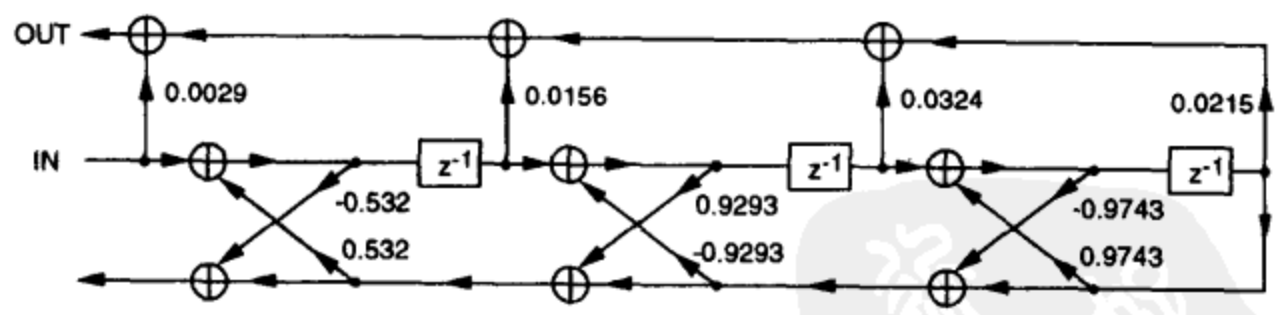


图12-4 应用Schur多项式设计的3阶基本格型滤波器

12.3.2 基本格型滤波器的逆Schur多项式导出

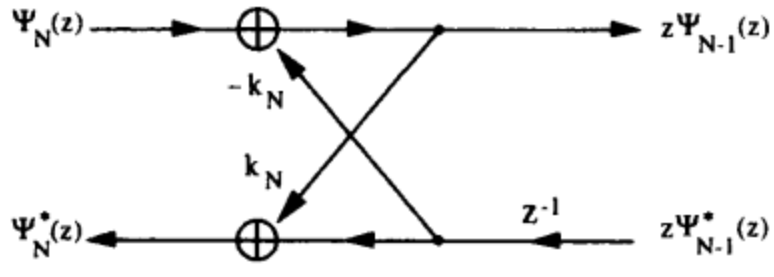
图12-2a中，可将上升沿的延时移到下降沿而不改变输入输出关系。请注意在图12-5a的右边是多项式 $z\psi_{N-1}(z)$ 和 $z\psi^*_{N-1}(z)$ 而不是 $\psi_{N-1}(z)$ 和 $\psi^*_{N-1}(z)$，因为延时由上升改为下降沿。根据逆Schur多项式的 $(i-j)$ 阶正规直交化，$\{\Phi^*_N(z),\ z\Phi^*_{N-1}(z),\ z^2\Phi^*_{N-2}(z),\ \ldots,\ z^N\Phi^*_0(z)\}$ 正规直交。因此由图12-5b下面一行的每个节点得到的多项式 $\{\psi^*_N(z),\ z\psi^*_{N-1}(z),\ z^2\psi^*_{N-2}(z),\ \ldots,\ z^N\psi^*_0(z)\}$ 直交，且这些多项式可作为基展开分子:

$$N_N(z)=\sum_{i=0}^{N}c_i z^{N-i}\Psi_i^*(z) \tag{12-50}$$

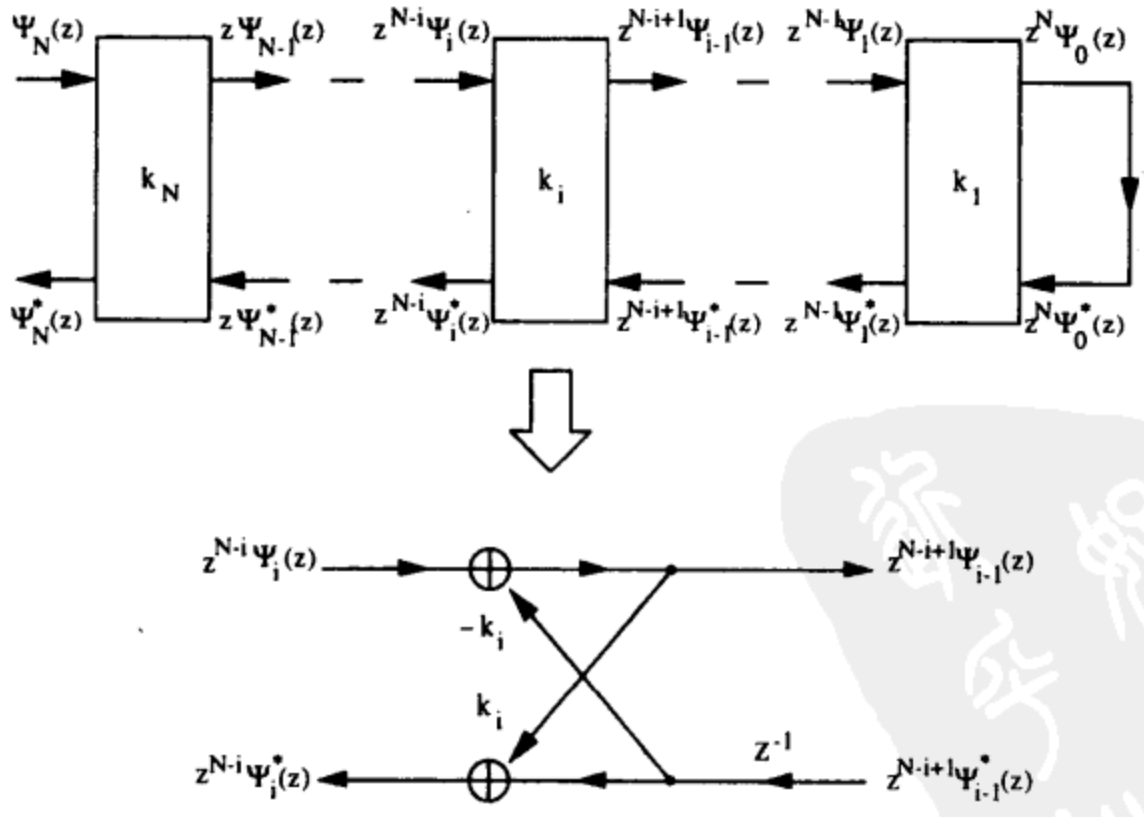
在公式（12-50）中，请注意对于 $i=N$ 到0，每一个多项式 $z^{N-i}\psi_i^*(z)$ 的阶为N，所以不能直接使用多项式展开算法。不过，根据公式（12-50）有

$$\begin{aligned} N_N^*(z) &= z^N\left\{\sum_{i=0}^{N}c_i z^{-(N-i)}\Psi_i^*(z^{-1})\right\} \\ &= \sum_{i=0}^{N}c_i\Psi_i(z) \end{aligned} \tag{12-51}$$

显然多项式展开算法可用于公式（12-51）。因此，公式（12-50）的展开系数与应用 $\{\psi_N^*(z),\psi_{N-1}^*(z),\psi_{N-2}^*(z),\ldots,\psi_0^*(z)\}$ 扩展分子的逆多项式得到的展开系数相同。前面的过程得到的基本格型滤波器结构见图12-6。



a) 公式（12-44）和公式（12-45）下降沿延时的实现



b) 分子 $D_N(z)$ 的实现

图 12-5

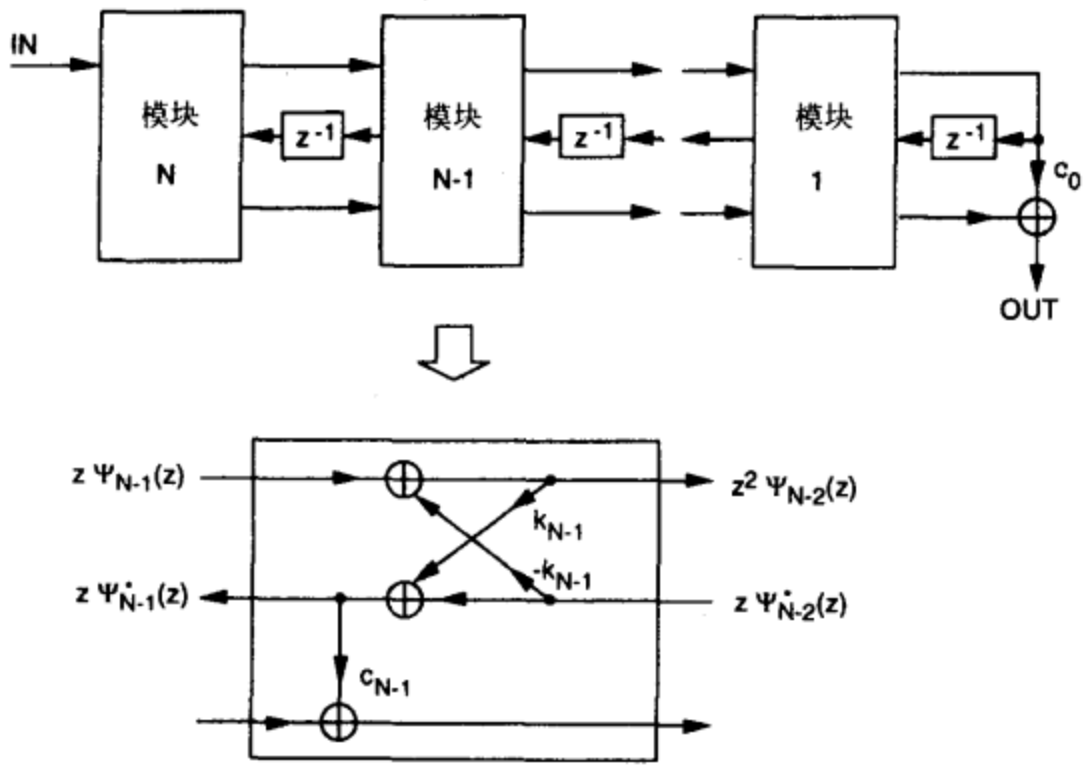


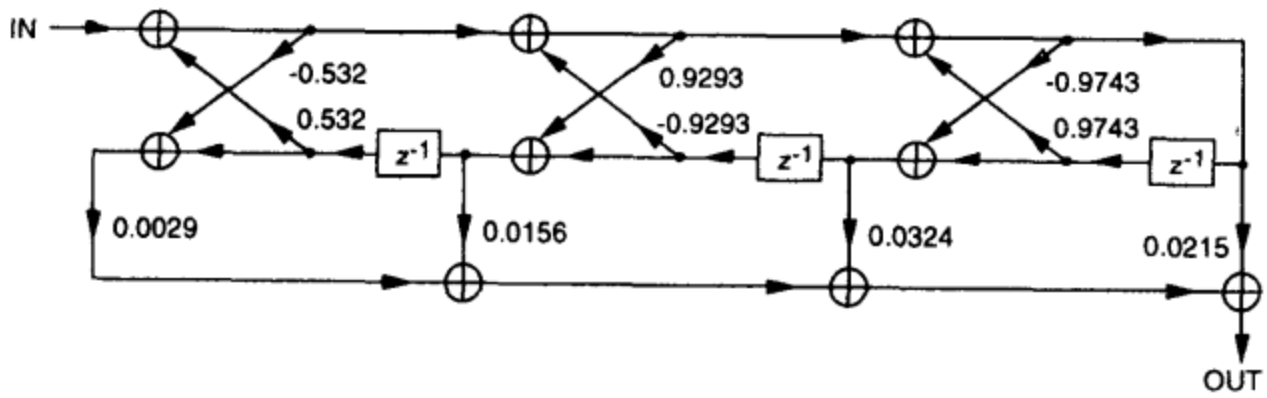
图12-6 应用逆Schur多项式得到的 N 轮基本格型滤波器结构

例12.3.2 重新分析例12.3.1给出的3轮巴特沃斯低通滤波器。

根据公式（12-51），应用逆Schur多项式（$\psi_i^*(z)$ 之类）计算得展开系数为：

$$c_3 = 0.0029,\ c_2 = 0.0156,\ c_1 = 0.0324,\quad 及\ c_0 = 0.0215$$

注意本例求得的展开系数和例12.3.1用分子对称得到的系数相同，也就是 $N(z) = N^*(z)$。图12-7给出了应用逆Schur多项式分解的基本格型滤波器。 ■



434 ~ 436

图12-7 应用逆Schur多项式设计的3阶基本格型滤波器

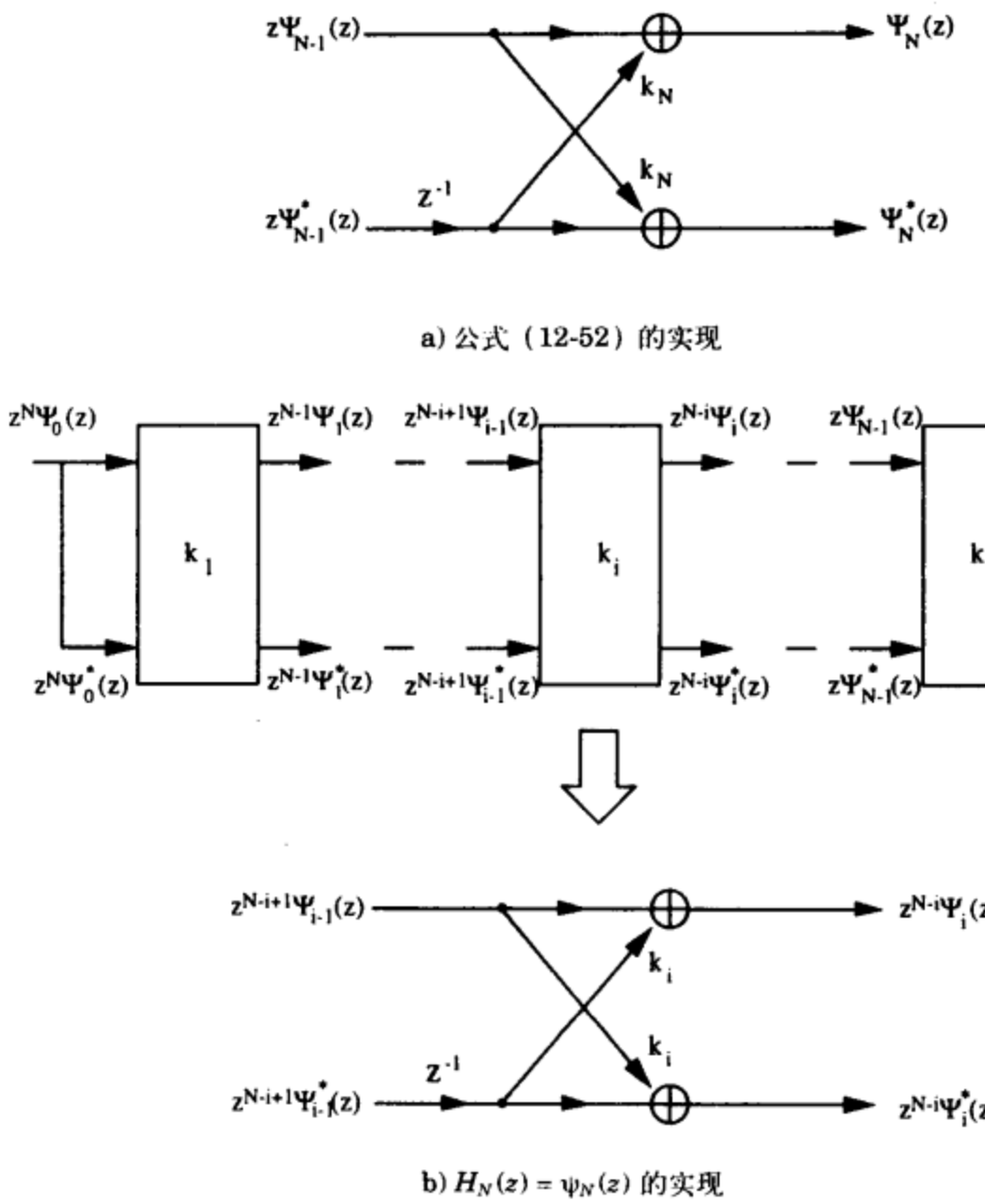
12.3.3 FIR格型滤波器的导出

本节将分析对应于 N 阶FIR传输函数 $H_N(z) = \psi_N(z)$ 的FIR格型滤波器结构。（假定 $\psi_N(z)$ 的 z 最高次幂系数为单位1）。根据公式（12-44）和公式（12-45），

$$\begin{aligned} \Psi_N(z) &= z\Psi_{N-1}(z) + k_N\Psi_{N-1}^*(z) \\ \Psi_N^*(z) &= \Psi_{N-1}^*(z) + k_N z\Psi_{N-1}(z) \end{aligned} \tag{12-52}$$

公式（12-52）的实现见图12-8a。通过对 $i = N, N-1, \ldots, 1$ 以 $\psi_{i-1}(z)$ 替代 $\psi_i(z)$ 重复使用上面的处理过程，$H_N(z)$ 综合后如图12-8b所示。请注意图12-8b中的FIR格型滤波器有 $2N$ 个乘法器，尽管它的单位抽样响应只有 N 个不同的系数。因此，这种结构效率不如直接式

FIR滤波器。但是，这种结构有一些实现自适应滤波器时需要的良好特性（见1.2节），并且是自回归信号模拟理论的中心环节。



a) 公式（12-52）的实现

b) $H_N(z) = \psi_N(z)$ 的实现

图 12-8

例12.3.3 分析如下3阶FIR滤波器传输函数：

$$H(z) = z^3 - 2.3741z^2 + 1.9293z - 0.532$$

这是由例12.3.1的3阶巴特沃斯滤波器的分母除 z 的最高次幂的系数而单位化得到的。所以，

$$\begin{aligned}
\Psi_3(z) &= z^3 - 2.3741z^2 + 1.9293z - 0.532 \\
\Psi_2(z) &= z^2 - 1.8797z + 0.9293 \\
\Psi_1(z) &= z - 0.9743 \\
\Psi_0(z) &= 1
\end{aligned}$$

参数 k 为：

$$k_3 = -0.532,\ k_2 = 0.9293 \quad 及\ k_1 = -0.9743$$

图12-9给出了 $H(z)$ 的FIR格型滤波器实现。 ■

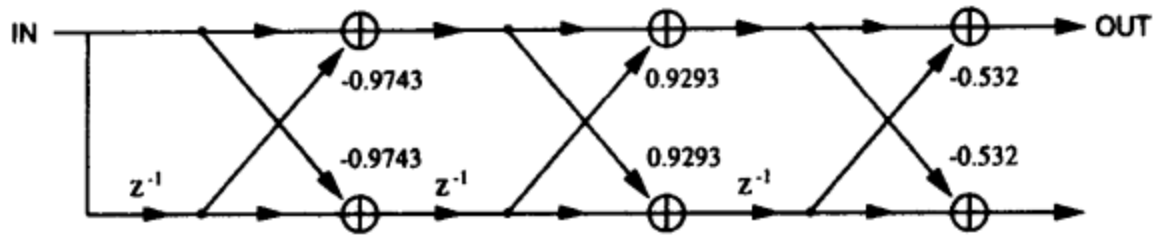


图12-9 例12.3.3的3阶FIR格型滤波器

12.4 单乘法器格型滤波器的导出

公式（12-36）中选取 $s_i = 1-\varepsilon_i k_i$，Schur算法表示如下:

437 ~ 438

$$\Lambda_{i-1}(z) = \frac{z^{-1}\{\Lambda_i(z) - k_i\Lambda_i^*(z)\}}{1-\epsilon_i k_i} \tag{12-53}$$

其中 ε_i 称为标志量，为1或-1。考虑一个N阶IIR传输函数 $H_N(z) = N_N(z) / D_N(z)$。然后，初始化 N 阶Schur多项式 $\Lambda_N(z)$ 为 $\Lambda_N(z) = D_N(z)$。根据 $\Lambda_N(z)$，用Schur多项式构成多项式 $\Lambda_{N-1}(z)$:

$$\Lambda_{N-1}(z) = \frac{z^{-1}\{\Lambda_N(z) - k_N\Lambda_N^*(z)\}}{1-\epsilon_N k_N} \tag{12-54}$$

其中

$$k_i = \Lambda_i(0)/\Lambda_i^*(0) \tag{12-55}$$

令 $i = N$,

$$\Lambda_{N-1}^*(z) = \frac{\Lambda_N^*(z) - k_N\Lambda_N(z)}{1-\epsilon_N k_N} \tag{12-56}$$

根据公式（12-56），

$$\Lambda_N^*(z) = k_N\Lambda_N(z) + (1-\epsilon_N k_N)\Lambda_{N-1}^*(z) \tag{12-57}$$

应用公式（12-54）和公式（12-57），

$$\Lambda_{N-1}(z) = z^{-1}\{(1+\epsilon_N k_N)\Lambda_N(z) - k_N\Lambda_{N-1}^*(z)\} \tag{12-58}$$

公式（12-57）和公式（12-58）的实现见图12-10a。对 $i=N, N-1, \ldots, 1$ 以 $\Lambda_{i-1}(z)$ 替代 $\Lambda_i(z)$ 重复应用前面的过程，分母 $D_N(z)$ 可分解如图12-10b所示。
$\Lambda_i(z)$ 和 $\Phi_i(z)$ 的关系如下:

$$\begin{aligned} \Lambda_N(z) &= \Phi_N(z) \\ \Lambda_i(z) &= \Phi_i(z)\sqrt{\frac{(1+\epsilon_N k_N)(1+\epsilon_{N-1}k_{N-1})\cdots(1+\epsilon_{i+1}k_{i+1})}{(1-\epsilon_N k_N)(1-\epsilon_{N-1}k_{N-1})\cdots(1-\epsilon_{i+1}k_{i+1})}} \end{aligned} \tag{12-59}$$

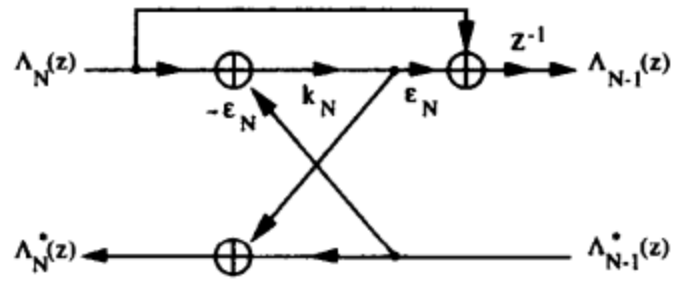
其中 $0 \le i < N$。请注意由 $\Lambda_i(z)$ 算出的参数 k 和由 $\Phi_i(z)$ 算出的参数 k 相同，因为:

$$k_i = \Lambda_i(0)/\Lambda_i^*(0) = \Phi_i(0)/\Phi_i^*(0) \tag{12-60}$$

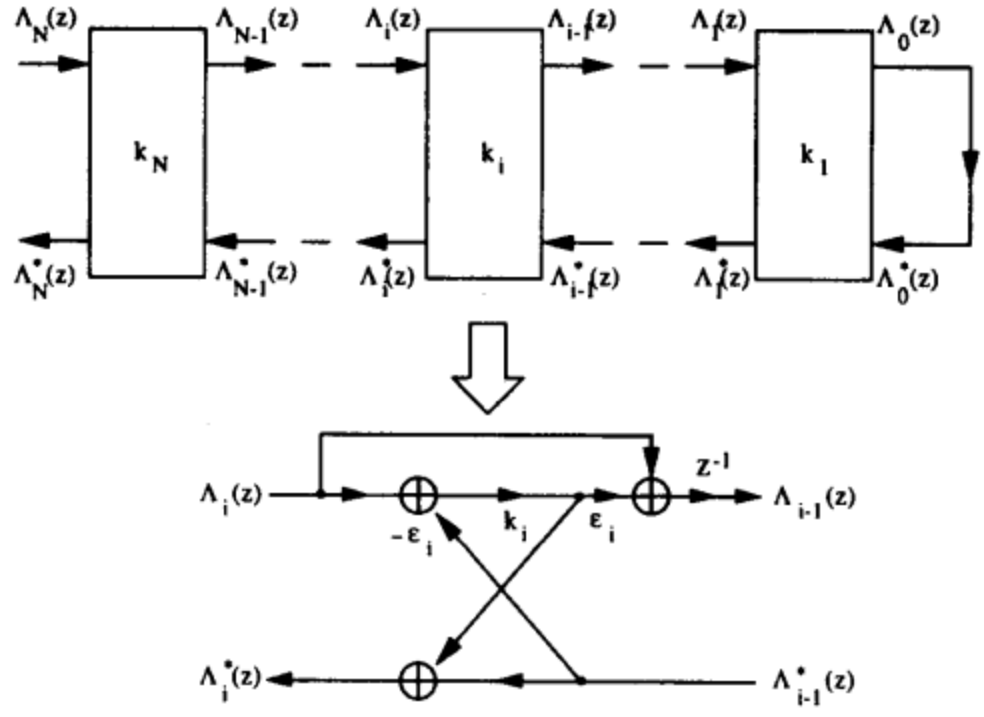
同时，根据公式（12-59）很容易看出 $\{\Lambda_N(z), \Lambda_{N-1}(z), \ldots, \Lambda_0(z)\}$ 直交而非正规直交，因为:

$$\begin{aligned}\langle \Lambda_i(z), \Lambda_i(z)\rangle &= \langle \Lambda_i^*(z), \Lambda_i^*(z)\rangle \\ &= \frac{(1+\epsilon_N k_N)(1+\epsilon_{N-1}k_{N-1})\cdots(1+\epsilon_{i+1}k_{i+1})}{(1-\epsilon_N k_N)(1-\epsilon_{N-1}k_{N-1})\cdots(1-\epsilon_{i+1}k_{i+1})}\end{aligned} \tag{12-61}$$

根据（12-61），可看出 $\langle \Lambda_i(z), \Lambda_i(z)\rangle$ 的值可适当选择标志量来调整。所以，单乘法器格型滤波器不需要严格的输入比例，并且比基本格型滤波器有更好的舍入噪声性质。 439



a) 公式（12-57）和公式（12-58）的实现



b) 分母 $D_N(z)$ 的实现

图 12-10 440

标志量选择的一个优化标准是要求和最大数目的参数 k 相关的节点具有最大的功率。然后通过递归选择标志量满足其他节点的功率在不超过最大和情况下尽可能大[6]。这个过程将在例12.4.1中解释。

因为图12-10b上面一行的多项式 $\{\Lambda_N(z), \Lambda_{N-1}(z), \ldots, \Lambda_1(z), \Lambda_0(z)\}$ 直交，可以以他们为直交基用多项式展开算法展开分子 $N_N(z)$ 为:

$$N_N(z) = \sum_{i=0}^{N} c_i \Lambda_i(z) \tag{12-62}$$

通过将这些逐轮获取的参数 $\{c_N, c_{N-1}, \ldots, c_0\}$ 和图12-10b的全极点部分联系起来，可得到如图12-11所示的单乘法器格型滤波器结构。

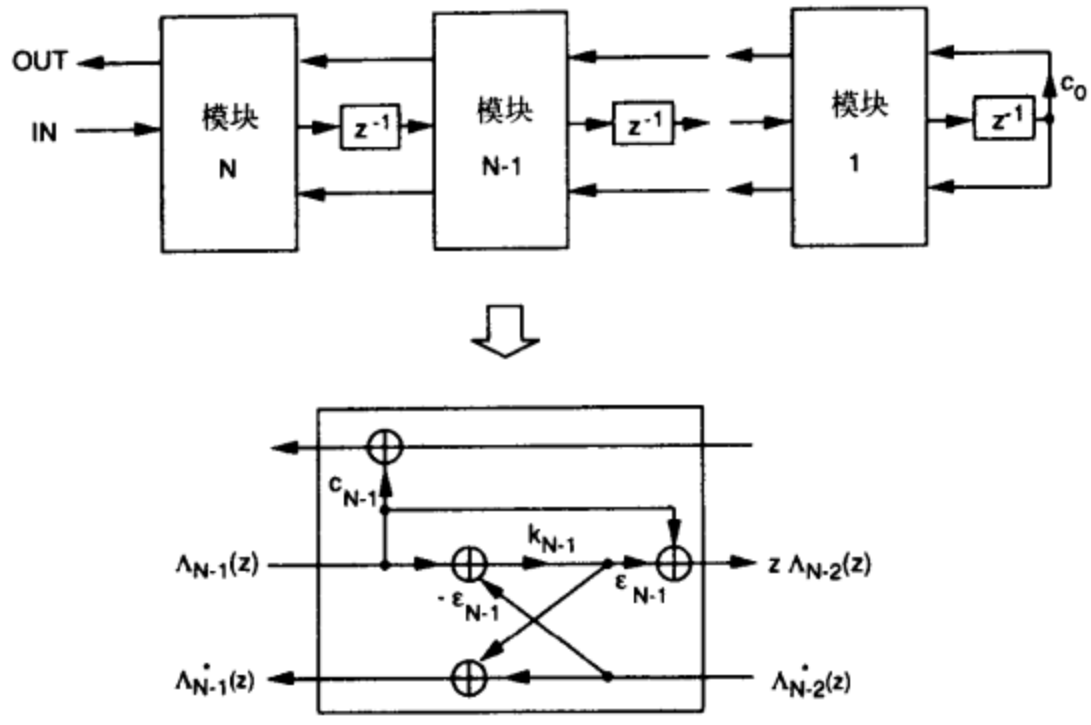


图12-11 一个N阶单乘法器格型滤波器的结构

通过和前一节相同的处理，图12-10b的上升沿延时可以转移到下降沿。然后，那些逐轮获取的参数可这样求得:

$$N_N^*(z) = \sum_{i=0}^{N} c_i \Lambda_i(z) \tag{12-63}$$

441 图12-12给出了用逆Schur多项式得到的单乘法器格型滤波器结构。

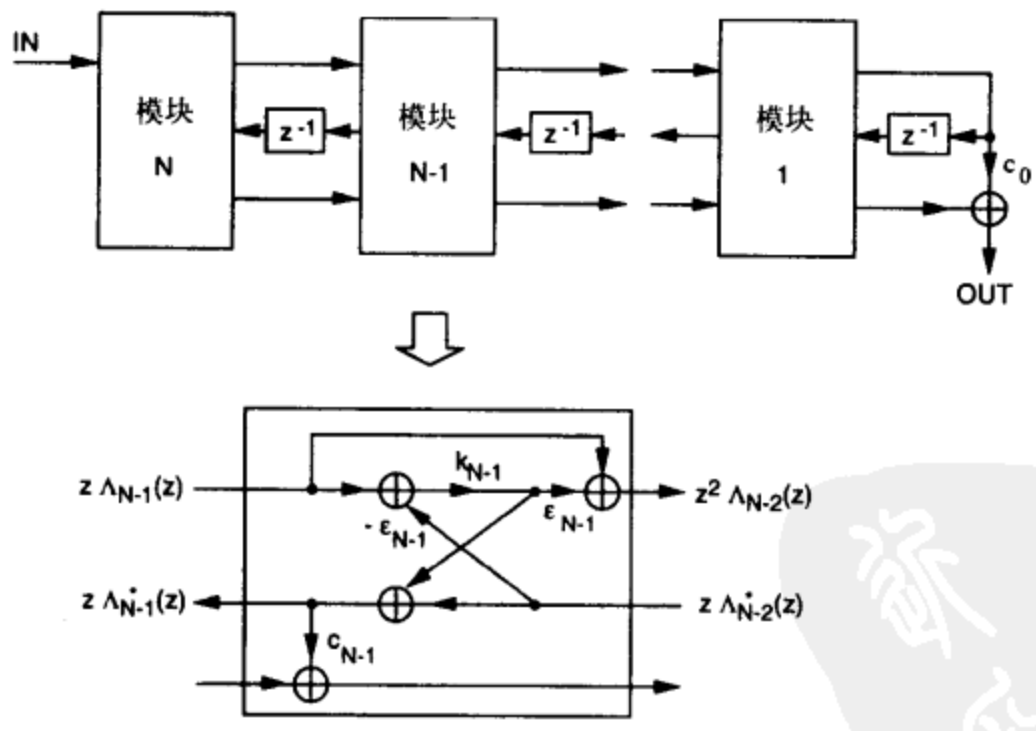


图12-12 应用逆Schur多项式得到的N阶单乘法器格型滤波器结构

例12.4.1 根据例12.3.1的3阶巴特沃斯滤波器，计算下列Schur多项式:

$$\begin{aligned}
\Phi_3(z) &= 345.1z^3 - 819.3z^2 + 665.8z - 183.6 \\
\Phi_2(z) &= 292.2072z^2 - 549.2662z + 271.5337 \\
\Phi_1(z) &= 107.956z - 105.1841 \\
\Phi_0(z) &= 24.3064
\end{aligned}$$

根据公式（12-60），求得参数 k 为：

$$k_3 = -0.532,\ k_2 = 0.9293,\ k_1 = -0.9743$$

应用公式（12-61），求得每个内部多项式为：

$$\begin{aligned}
\langle \Lambda_3(z), \Lambda_3(z)\rangle &= 1 \\
\langle \Lambda_2(z), \Lambda_2(z)\rangle &= \frac{(1+\epsilon_3 k_3)}{(1-\epsilon_3 k_3)} \\
\langle \Lambda_1(z), \Lambda_1(z)\rangle &= \frac{(1+\epsilon_3 k_3)(1+\epsilon_2 k_2)}{(1-\epsilon_3 k_3)(1-\epsilon_2 k_2)} \\
\langle \Lambda_0(z), \Lambda_0(z)\rangle &= \frac{(1+\epsilon_3 k_3)(1+\epsilon_2 k_2)(1+\epsilon_1 k_1)}{(1-\epsilon_3 k_3)(1-\epsilon_2 k_2)(1-\epsilon_1 k_1)}
\end{aligned}$$

442

因为 k_1 是最大的数，要求 $\langle \Lambda_0(z), \Lambda_0(z)\rangle$ 取最大值。同时，还要让其他节点的功率在不超过 $\langle \Lambda_0(z), \Lambda_0(z)\rangle$ 情况下尽可能大。为了确定 ε_1，分析如下比例：

$$\frac{\langle \Lambda_1(z), \Lambda_1(z)\rangle}{\langle \Lambda_0(z), \Lambda_0(z)\rangle} = \frac{1-\epsilon_1 k_1}{1+\epsilon_1 k_1}$$

它要不能大于1，且满足：

$$\epsilon_1 = sign(k_1) = -1$$

现在，$\langle \Lambda_1(z), \Lambda_1(z)\rangle$ / $\langle \Lambda_0(z), \Lambda_0(z)\rangle = 0.013$。下一步是要满足如下条件：

$$\frac{\langle \Lambda_2(z), \Lambda_2(z)\rangle}{\langle \Lambda_0(z), \Lambda_0(z)\rangle} = \frac{1-\epsilon_2 k_2}{1+\epsilon_2 k_2}(0.013) < 1$$

这个值要在不大于1条件下尽可能大。所以，选择：

$$\epsilon_2 = -sign(k_2) = -1$$

可以得到：

$$\frac{(1-\epsilon_2 k_2)(1-\epsilon_1 k_1)}{(1+\epsilon_2 k_2)(1+\epsilon_1 k_1)} = 0.3548$$

最后，根据下面的条件：

$$\frac{\langle \Lambda_3(z), \Lambda_3(z)\rangle}{\langle \Lambda_0(z), \Lambda_0(z)\rangle} = \frac{1-\epsilon_3 k_3}{1+\epsilon_3 k_3}(0.3548) < 1$$

ε_3 选择为：

$$\epsilon_3 = sign(k_3) = -1$$

将 ε_1、ε_2、ε_3 代入公式（12-59），得到：

$$\begin{aligned}
\Lambda_3(z) &= 345.1z^3 - 819.3z^2 + 665.8z - 183.6 \\
\Lambda_2(z) &= 528.7z^2 - 993.8053z + 491.2947 \\
\Lambda_1(z) &= 37.4053z - 36.448 \\
\Lambda_0(z) &= 73.8501
\end{aligned}$$

根据公式（12-63）和多项式展开算法，应用逆Schur多项式（$\Lambda_i^*(z)$一类）的分子的展开系数为：

$$c_3 = 0.0029,\ c_2 = 0.0102,\ c_1 = 0.2987,\ c_0 = 0.1005$$

443 图12-13给出了用逆Schur多项式分解的单乘法器格型滤波器。 ■

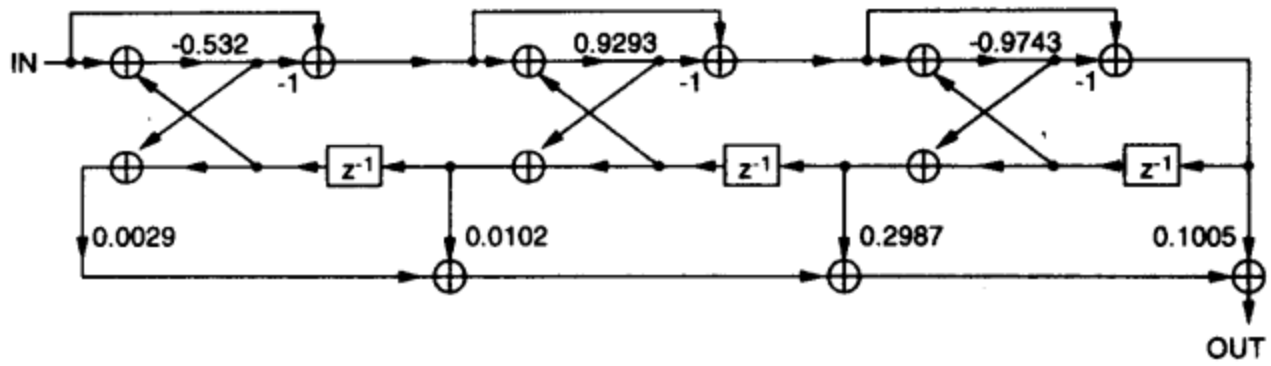


图12-13 例12.4.1的3阶单乘法器格型滤波器

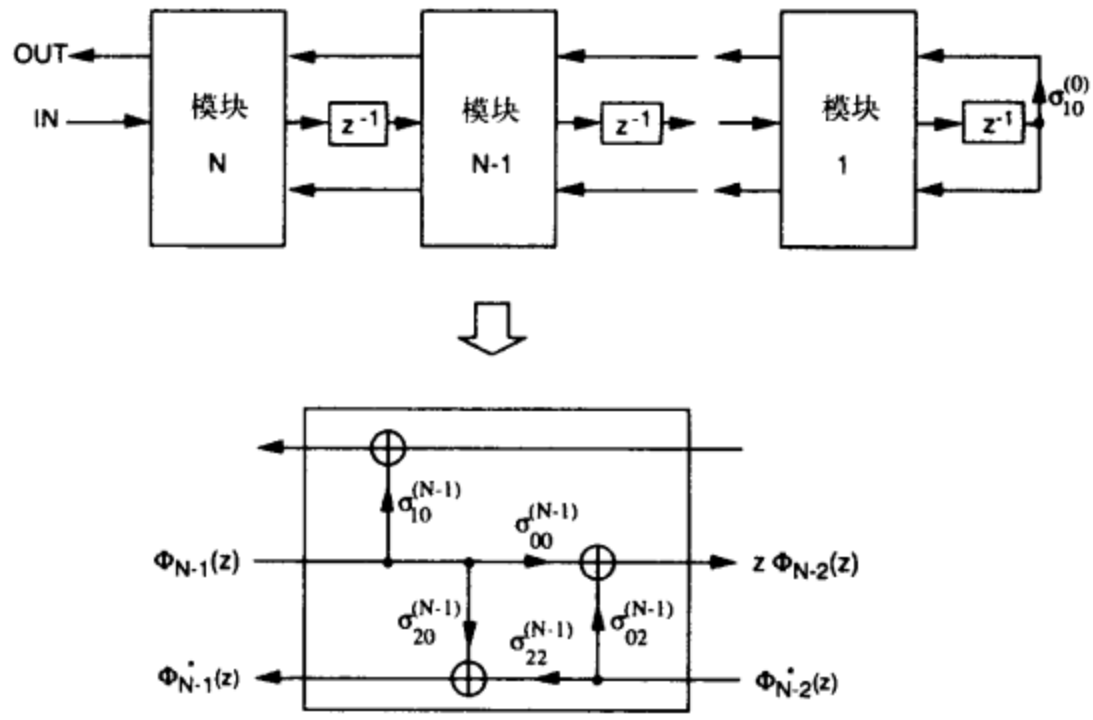


图12-14 N阶归一化格型滤波器结构

12.5 归一化格型滤波器的导出

图12-14给出了一个N阶归一化格型滤波器结构。考虑一个N阶IIR传输函数 $H_N(z) = N_N(z)/D_N(z)$。再初始化N阶Schur多项式 $\Phi_N(z)$ 为 $\Phi_N(z) = D_N(z)$，其中：

$$\Phi_N(z) = \sum_{i=0}^{N} \phi_i z^i \tag{12-64}$$

根据 $\Phi_N(z)$，应用Schur算法构成多项式 $\Phi_{N-1}(z)$ 为：

444

$$\Phi_{N-1}(z) = \frac{z^{-1}\{\Phi_N(z) - k_N \Phi_N^*(z)\}}{\sqrt{1-k_N^2}} \tag{12-65}$$

其中

$$k_i = \Phi_i(0)/\Phi_i^*(0) \tag{12-66}$$

令 $i = N$，同样，$\Phi_{N-1}(z)$ 的逆多项式 $\Phi^*_{N-1}(z)$ 为:

$$\Phi^*_{N-1}(z) = z^{N-1}\Phi_{N-1}(z^{-1}) = \frac{\Phi^*_N(z) - k_N\Phi_N(z)}{\sqrt{1-k_N^2}} \tag{12-67}$$

根据公式（12-67），

$$\Phi^*_N(z) = \sqrt{1-k_N^2}\Phi^*_{N-1}(z) + k_N\Phi_N(z) \tag{12-68}$$

应用公式（12-65）和公式（12-68），$\Phi_{N-1}(z)$ 可以表示为 $\Phi_N(z)$ 和 $\Phi^*_{N-1}(z)$ 的函数:

$$\Phi_{N-1}(z) = z^{-1}\{\sqrt{1-k_N^2}\Phi_N(z) - k_N\Phi^*_{N-1}(z)\} \tag{12-69}$$

图12-15a给出了公式（12-68）和公式（12-69）的实现，它将和图12-14的归一化格型滤波器的模块N的反馈部分相同，如果满足:

$$\sigma_{20}^{(N)} = -\sigma_{02}^{(N)} = k_N,\ \ \sigma_{00}^{(N)} = \sigma_{22}^{(N)} = \sqrt{1-k_N^2} \tag{12-70}$$

通过对 $i=N, N-1, \ldots, 1$ 以 $\Phi_{i-1}(z)$ 替代 $\Phi_i(z)$ 重复应用上面的过程，可将分母 $D_N(z)$ 分解，如图12-15b所示。对于图12-15b的模块 i:

$$\sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = k_i,\ \sigma_{00}^{(i)} = \sigma_{22}^{(i)} = \sqrt{1-k_i^2} \tag{12-71}$$

请注意图12-15b中的每个模块组成一个Givens旋转，它能看出良好的数字属性。同时，在图12-15b中，上面一行的多项式是 $\{\Phi_N(z), \Phi_{N-1}(z), \ldots, \Phi_0(z)\}$，可以用来作为正规直交基函数展开任意$N$阶多项式。如果用多项式展开算法将分子 $N_N(z)$ 按下式展开:

$$N_N(z) = \sum_{i=0}^{N} c_i\Phi_i(z) \tag{12-72}$$

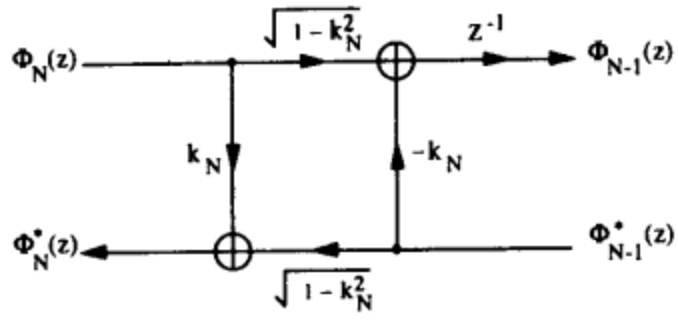
然后可选择下式将分子分解，如图12-14所示:

$$\sigma_{10}^{(i)} = c_i \tag{12-73}$$

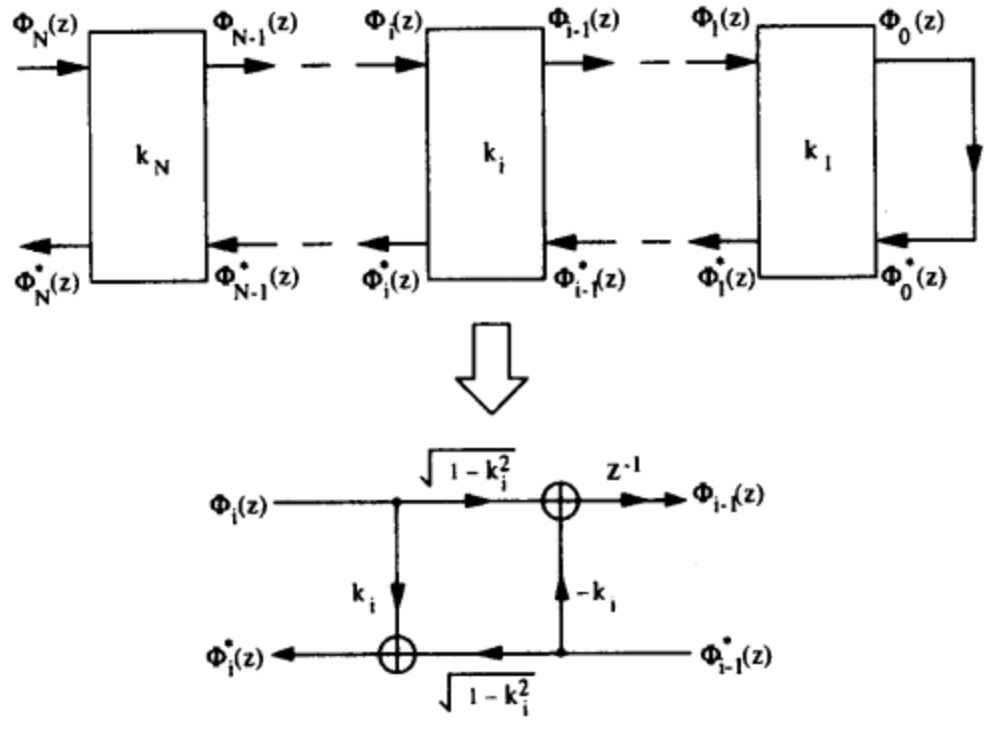
所以，按公式（12-71）和公式（12-73）选择滤波器参数，如果滤波器输入为 $D_N(z)$，可由滤波器输出得到分子 $N_N(z)$。这表示如图12-14所示的归一化格型滤波器实现了 $H_N(z) = N_N(z) / D_N(z)$。请注意反馈路径的每一个节点，也就是图12-15b的每一个节点都为单位功率，因为: 445

$$\langle\Phi_i(z), \Phi_i(z)\rangle = \langle\Phi_i^*(z), \Phi_i^*(z)\rangle = 1 \tag{12-74}$$

则该状态协方差矩阵$\mathbf{K}$，对于全极点归一化格型滤波器应为恒等矩阵，即$\mathbf{K} = \mathbf{I}$，且这个结构为正规直交滤波器。但是，极点-零点归一化格型滤波器不是正规直交滤波器，因为状态的方差对应分子的部分不是单位量。最终通过与前几节相同的处理，可用逆Schur多项式推导出归一化格型滤波器。



a) 公式(12-68)和公式(12-69)的实现



b) 分母 $D_N(z)$ 的实现

图 12-15

例12.5.1 分析用归一化格型滤波器实现例12.3.1给出的传输函数。应用公式（12-47）和例12.3.1的结果，归一化格型滤波器的参数k为：

$$k_3 = -0.532,\ k_2 = 0.9293,\ k_1 = -0.9743$$

应用公式（12-72），应用Schur多项式的分子的展开系数为：

$$c_3 = 0.0029,\ c_2 = 0.0184,\ c_1 = 0.1035,\ c_0 = 0.3054$$

然后将归一化格型滤波器分解如图12-16所示。 ■

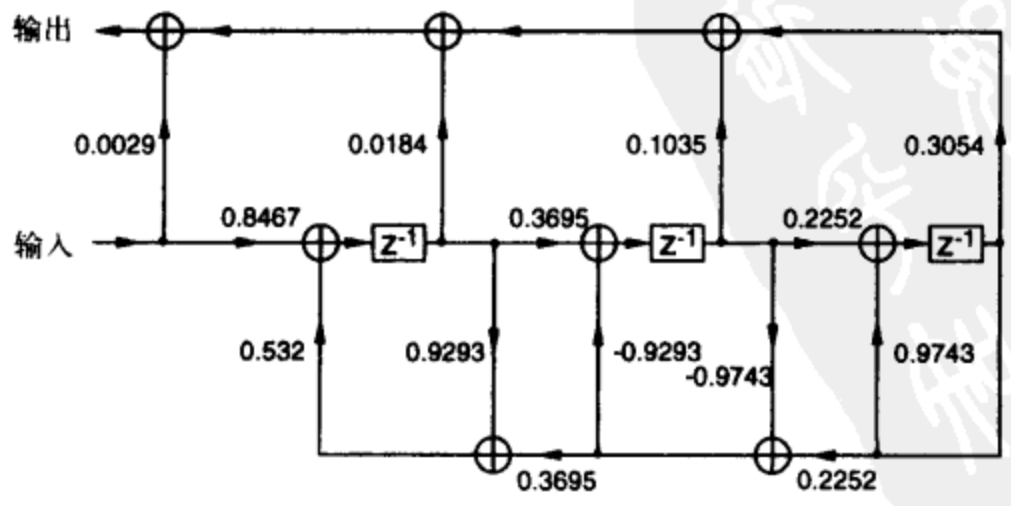


图12-16 例12.5.1的3阶归一化格型滤波器

12.6 可缩放归一化格型滤波器的导出

对3阶归一化格型滤波器应用$M = 2$的降速定时调整/流水线（SRP）变换，可以得到如图12-17所示的滤波器结构。图12-17中，每个延时成分用 x_i 表示，i从1到8；$\sigma_{10}^{(i)}$用 c_i 表示，i从1到3。根据公式（12-74），对于i由3到8的每一个 $x_i(n)$ 都为单位功率。对于i从3到8，状态协方差矩阵的各项 $K_{ii} = 1$，其中 K_{ii} 为K的第i个对角线元素。现在，为了缩放滤波器，只要计算 $x_1(n)$ 和 $x_2(n)$ 的功率即可，这可由公式（11-25）求出。如果是归一化格型滤波器，可根据Schur多项式的正规直交性更快求出 K_{11} 和 K_{22}。根据图12-17所示，x_1 状态的多项式为: 446 ~ 447

$$\Phi^{(x_1)}(z) = c_2\Phi_2(z) + c_1\Phi_1(z) + c_0\Phi_0(z) \tag{12-75}$$

所以，应用Schur多项式的正规直交性，

$$\begin{aligned} K_{11} &= \langle \Phi^{(x_1)}(z), \Phi^{(x_1)}(z) \rangle \\ &= c_2^2 + c_1^2 + c_0^2 \end{aligned} \tag{12-76}$$

同样，

$$K_{22} = c_1^2 + c_0^2 \tag{12-77}$$

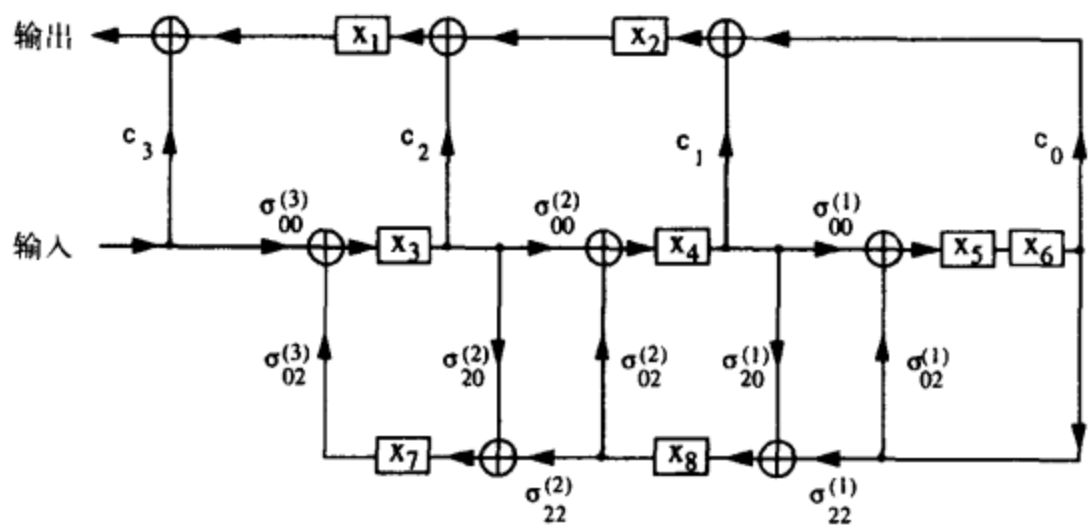


图12-17 用$M = 2$采样和定时调整过的3阶归一化格型滤波器

一般来说，对模块 i 的上面一行的节点求和，可得到功率为 $\sum_{j=0}^{i} c_j^2$。为了让每个内部节点有单位功率，3阶归一化格型滤波器的对角变换矩阵 $\mathbf{T}$ 的各元素可用公式（11-38）求得:

$$\begin{aligned} t_1 &= \sqrt{c_2^2 + c_1^2 + c_0^2} \\ t_2 &= \sqrt{c_1^2 + c_0^2} \\ t_i &= 1,\ i = 3 \text{ 到 } 8 \end{aligned} \tag{12-78}$$

448

根据变换法则 $x' = T^{-1}x$，可以发现 $x_i' = x_i / t_i$。因此为了缩放该系统，所有和状态 x_i 输入端有关的系数都要乘 $1/t_i$，而所有和状态 x_i 输出端有关的系数都要乘以 t_i（见图12-18）。

图12-18 应用变换法则 $x' = T^{-1}x$ 的变换

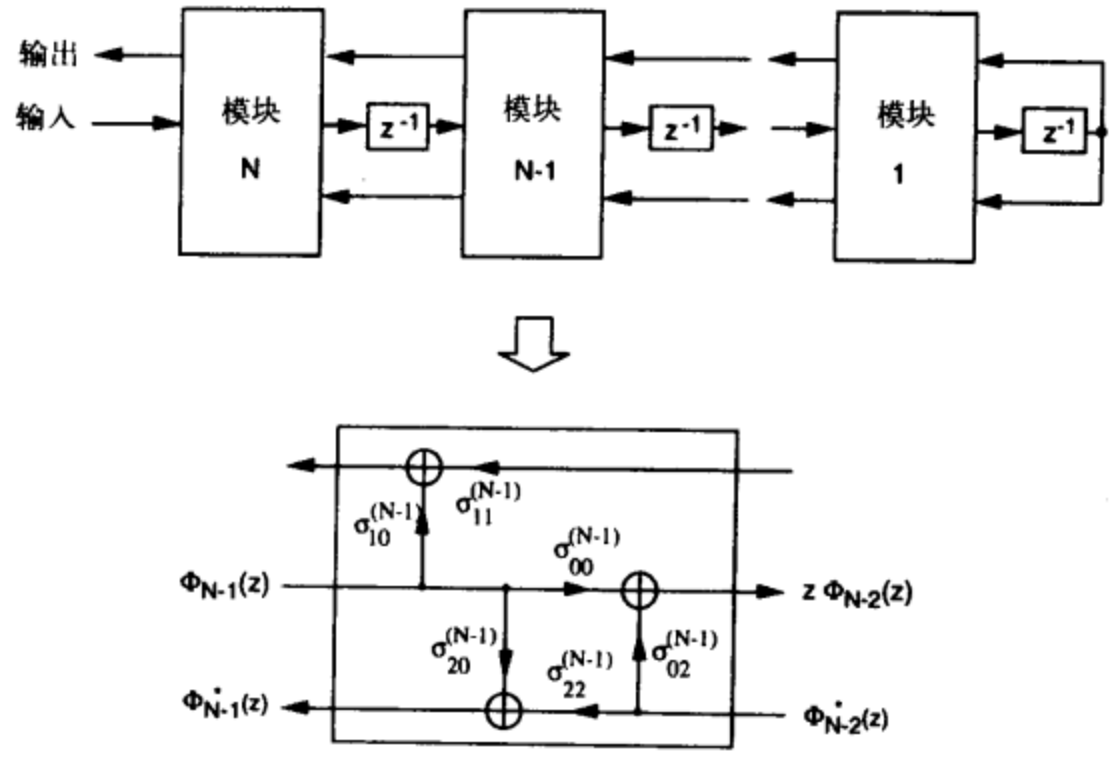


图12-19 N阶可缩放归一化格型滤波器结构

将上面的结果应用到一般的 N 阶归一化格型滤波器，可以得到如图12-19所示的 N 阶可缩放归一化格型滤波器结构，其中：

$$
\begin{aligned}
&\text{模块 } N: \quad \sigma_{10}^{(N)} = c_N,\ \sigma_{11}^{(N)} = \sqrt{\sum_{j=0}^{N-1} c_j^2} \\
&\text{模块 } i\ (i = N-1 \text{ 到 } 1): \quad \sigma_{10}^{(i)} = \frac{c_i}{\sqrt{\sum_{j=0}^{i} c_j^2}},\ \sigma_{11}^{(i)} = \frac{\sqrt{\sum_{j=0}^{i-1} c_j^2}}{\sqrt{\sum_{j=0}^{i} c_j^2}} \\
&\text{任一模块 } i\ (i = N \text{ 到 } 1): \quad \sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = k_i \\
&\qquad\qquad\qquad\qquad\qquad\qquad \sigma_{00}^{(i)} = \sigma_{22}^{(i)} = \sqrt{1 - k_i^2}
\end{aligned}
\tag{12-79}
$$

449 上面的等式描述了可缩放归一化格型滤波器。因此相对于[5]中所用的局部优化方案，这里根据缩放思想推导出这种格型滤波器[7]。

在公式（12-79）中，参数 k_i 和 c_i 的计算方法和归一化格型滤波器的方法相同。可以看到归一化和可缩放归一化格型滤波器的反馈部分 $\left\{\sigma_{20}^{(i)}, \sigma_{02}^{(j)}, \sigma_{00}^{(i)}, \sigma_{22}^{(j)}\right\}$ 也相同。

对于有限字长运算，要求在临界节点不溢出的情况下所有量值尽可能大，从而让独立计算中的关键数字也尽可能大[8]。另外，如果内部节点都有相同的功率，每次计算要用同样个数的关键数字实现（按均值计算）。如果选取乘 l_2 让每个内部节点不超过单位量，那么滤波器的输入通常按反馈路径节点中的最大值缩减。这样缩减将减少输入的动态范围，增加输出信号的舍入误差。

对可缩放归一化格型滤波器最佳的缩放能令每个内部节点都为单位功率。同时，由于最大值为单位量，输入不需要缩放。这样的话，可缩放归一化格型滤波器将会有很好的有限字长特性。另外，这种滤波器还有一些利于VLSI设计的性质，例如规则化、模块化、连线区域化以及流水线操作。而且，每个模块的Givens循环很容易用坐标旋转数字计算机（CORDIC）

算法实现[9]，[10]。

当滤波器极点不聚合时，可缩放归一化格型滤波器和平行/级联结构滤波器的舍入噪声相差很小。不过，对于聚合的极点，归一化或可缩放归一化格型滤波器有远优于常规平行/级联结构滤波器的舍入噪声特征。[11]中用36位PDP-10计算机模拟一个聚合极点的选通滤波器，归一化格型滤波器比平行结构滤波器优越大概10位。因为可缩放归一化格型滤波器的舍入噪声小于归一化格型滤波器[5]，可以推断在相同的规格下，可缩放归一化格型滤波器比平行结构滤波器至少少用10位。

可缩放归一化格型滤波器比常规平行/级联结构滤波器需要更多的乘法器。但是，如上一段所讨论，在不同滤波器结构之间只比较乘法器数目是没有意义的。因为乘法器的硬件复杂度正比于字长的平方，较多乘法器但较小字长的滤波器有时可能比较少乘法器却较大字长的滤波器需要更少的硬件实现。因此，数字滤波器的硬件复杂度要严格根据频率规格确定。

可缩放归一化格型滤波器的设计过程总结如下: 450

(1) 根据分母 $D_N(z)$ 计算Schur多项式 $\{\Phi_N(z), \Phi_{N-1}(z), \ldots, \Phi_0(z)\}$;

(2) 对于 $i = N, N-1, \ldots, 1$，求出 k 参数 $k_i = \Phi_i(0) / \Phi^*_i(0)$;

(3) 用多项式展开算法展开分子 $N_N(z)$ 为 $N_N(z) = \sum_{i=0}^{N} c_i \Phi_i(z)$;

(4) 由公式（12-79），用步骤2和3的参数分解可缩放归一化格型滤波器。

例12.6.1 分析用可缩放归一化格型滤波器实现例12.3.1给出的传输函数。应用公式（12-79）和例12.5.1得到的分子扩展系数，可缩放归一化格型滤波器可分解如图12-20所示。 ■

图12-15a中，上升沿延时可以移到下降沿而不改变图12-21a所示的输入输出关系。请注意图12-21a右边的多项式为 $z\Phi_{N-1}(z)$ 和 $z\Phi^*_{N-1}(z)$ 而非 $\Phi_{N-1}(z)$ 和 $\Phi^*_{N-1}(z)$。图12-21b给出了分母位下降沿延时的实现。根据逆Schur多项式的 $(i-j)$ 阶正规直交化，由图12-21b下面一行的每个节点得到的多项式是正规直交的，可作为正规直交基展开分子为:

$$N_N(z) = \sum_{i=0}^{N} c_i z^{N-i} \Phi_i^*(z) \tag{12-80}$$

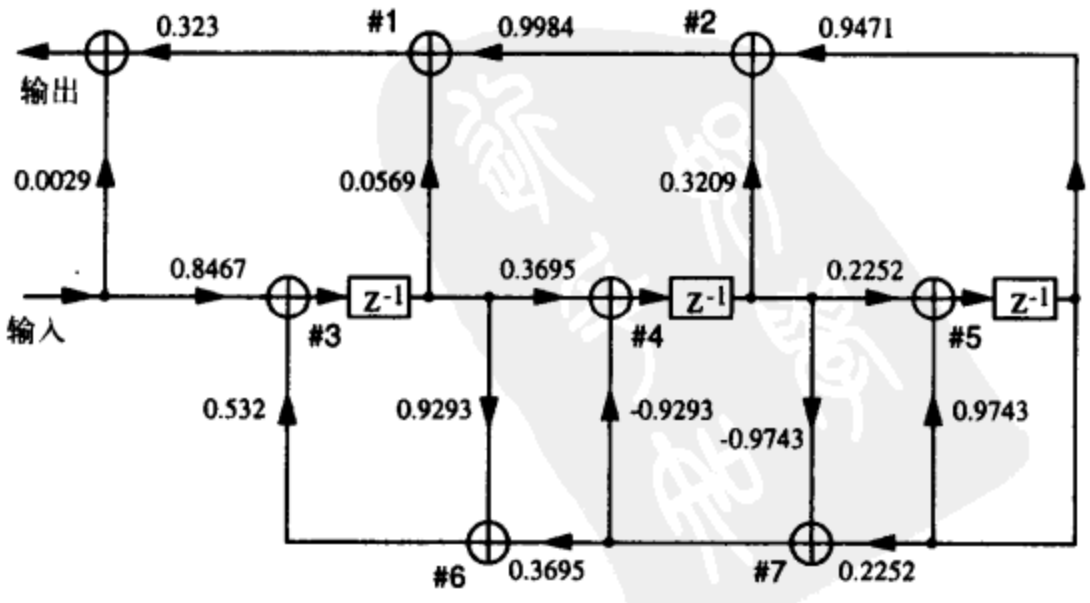
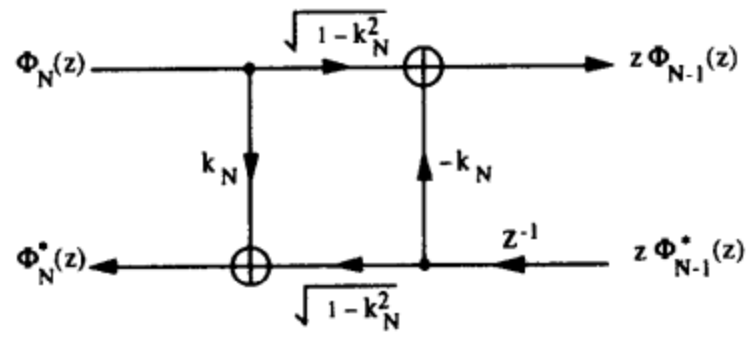
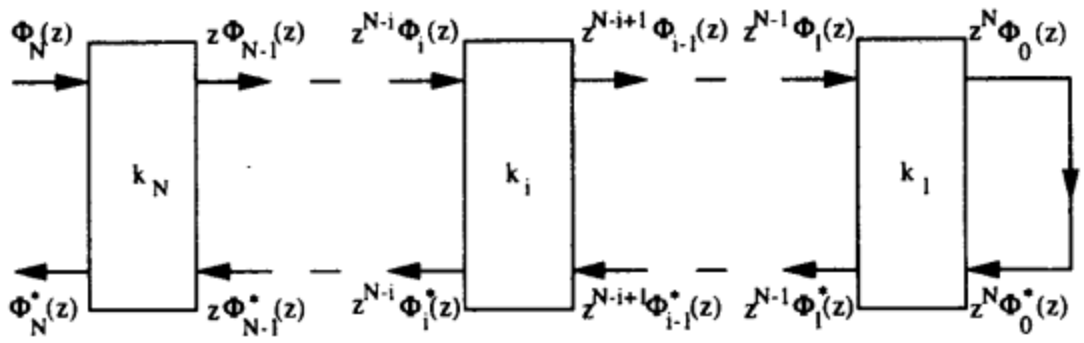
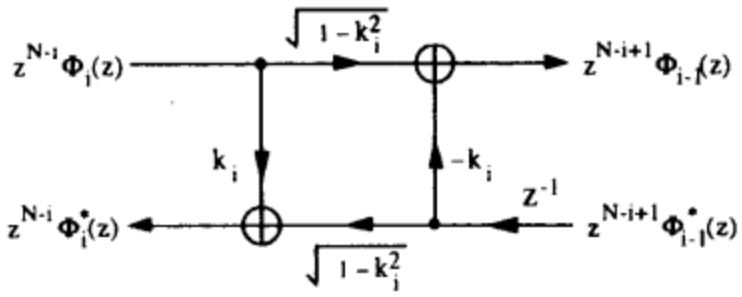


图12-20 例12.6.1的3阶可缩放归一化格型滤波器 451



a) 令公式(12-68)和公式(12-69)下降沿延时的实现





b) 分母 $D_N(z)$ 的实现

452

图 12-21

在公式（12-80）中，请注意对于 $i=N$ 到0的每个多项式 $z^{N-i}\Phi_i^*(z)$ 的阶都为N，所以不能直接应用多项式展开算法。但是，根据公式（12-80），

$$
\begin{aligned}
N_N^*(z) &= z^N\left\{\sum_{i=0}^{N} c_i z^{-(N-i)}\Phi_i^*(z^{-1})\right\} \\
&= \sum_{i=0}^{N} c_i\Phi_i(z)
\end{aligned}
\tag{12-81}
$$

显然多项式展开算法可对公式（12-81）应用。所以，公式（12-80）的展开系数可用 $\{\Phi_N(z), \Phi_{N-1}(z), \ldots, \Phi_0(z)\}$ 展开分子的逆多项式求得。如图12-22给出了逆可缩放归一化格型滤波器（用逆Schur多项式得到逐轮获取的参数的可缩放归一化格型滤波器结构）。在图12-22中，参数 $\left\{\sigma_{00}^{(i)}, \sigma_{20}^{(i)}, \sigma_{02}^{(i)}, \sigma_{22}^{(i)}\right\}$ 和公式（12-79）中相同。通过和上一节同样的处理过程，这些逐轮获取的参数可令每个内部节点为单位功率。求得的逐轮获取的参数为:

$$
\begin{aligned}
&\text{模块 } N \quad : \quad \sigma_{10}^{(N)} = 1,\ \sigma_{11}^{(N)} = 0 \\
&\text{模块 } i\ (i = N-1 \text{ 到 } 1) \quad : \quad \sigma_{10}^{(i)} = \frac{c_i}{\sqrt{\sum_{j=i}^{N} c_j^2}},\ \sigma_{11}^{(i)} = \frac{\sqrt{\sum_{j=i+1}^{N} c_j^2}}{\sqrt{\sum_{j=i}^{N} c_j^2}} \\
&\text{其他} \quad : \quad \sigma_{10}^{(0)} = c_0,\ \sigma_{11}^{(0)} = \sqrt{\sum_{j=1}^{N} c_j^2}
\end{aligned}
\tag{12-82}
$$

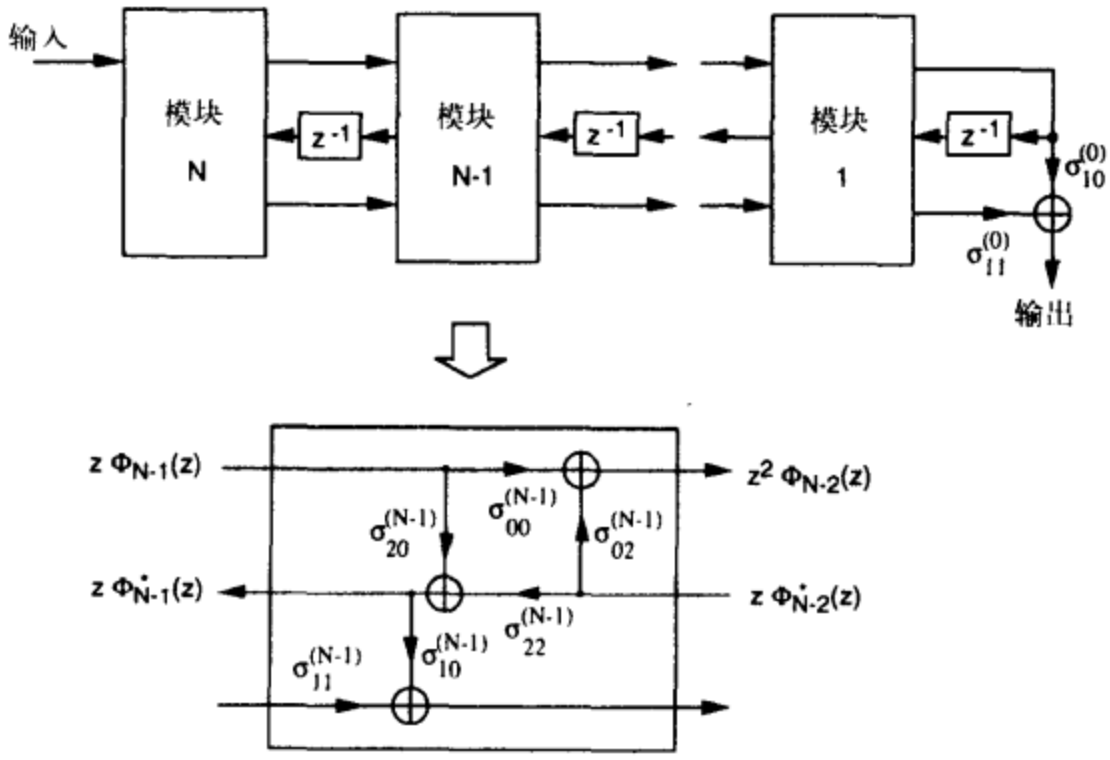


图12-22 N阶逆可缩放归一化格型滤波器的结构

453

例12.6.2 例12.3.1的传输函数用逆可缩放归一化格型滤波器实现，如图12-23所示。注意图12-23的逐轮获取参数和图12-20相同，因为 $N^*(z) = N(z)$。 ■

12.7 格型滤波器中的舍入噪声计算

除了计算基于状态变量描述理论的 **K** 和 **W** 矩阵，用[5]中的方法也很容易算出格型滤波器的舍入噪声。本节将给出这个基于Schur算法和转置定理方法的解释[1]，[12]。同时还将比较可缩放归一化和逆可缩放归一化格型滤波器的舍入噪声特性。

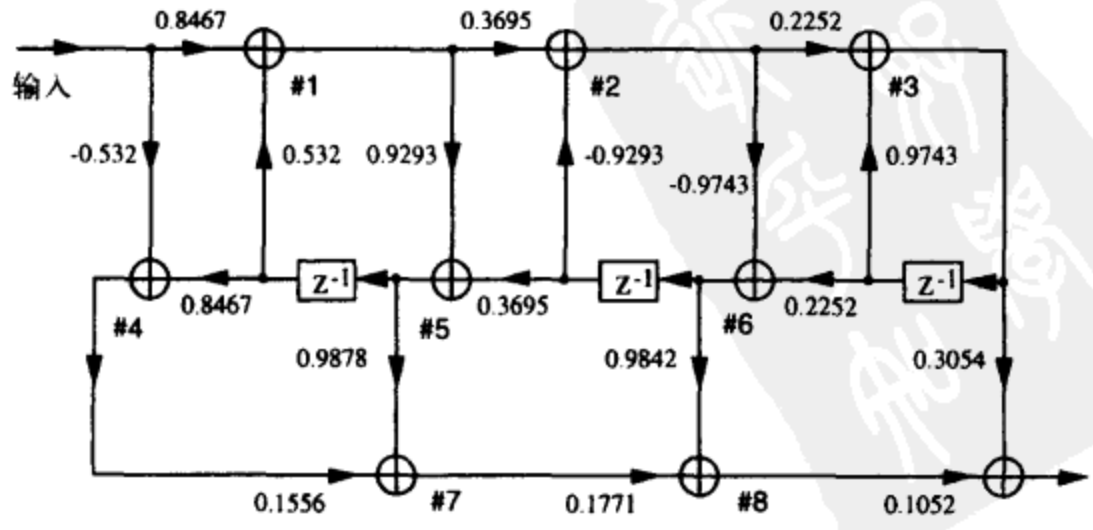


图12-23 例12.6.2的3阶逆可缩放归一化格型滤波器

12.7.1 应用转置定理的舍入噪声计算

为了计算输出舍入噪声，就需要从内部节点到输出节点的传输函数，这可通过转置定理找到。根据转置定理，转置图由输入到内部节点的传输函数也是初始图中相应的内部节点到输出节点的传输函数。图12-24给出了一个可缩放归一化格型滤波器的转置图模块。(为了方便标记，省略上标。同时，应用公式（12-79），使用如下标记：$k=\sigma_{20}=-\sigma_{02}$，$k_c=\sqrt{1-k^2}=\sigma_{00}=\sigma_{22}$。)

454

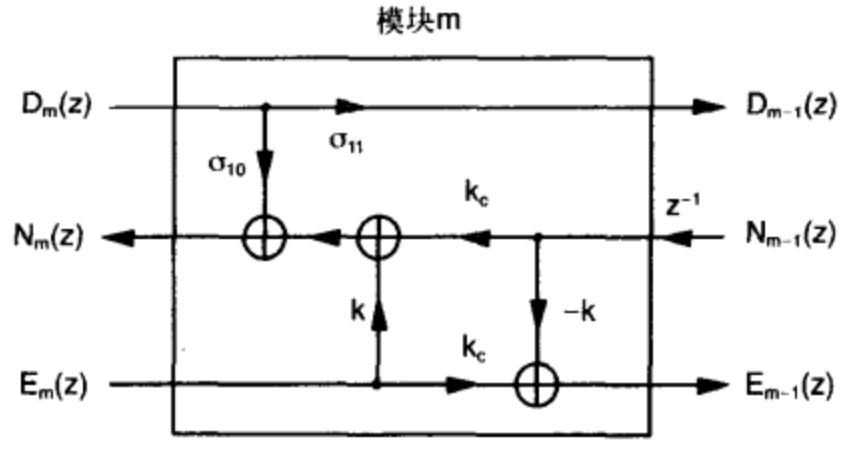


图12-24 可缩放归一化格型滤波器的转置图的模块m

根据转置图，可推导出如下等式：

$$\begin{aligned} N_m(z) &= \sigma_{10}D_m(z)+kE_m(z)+k_cz^{-1}N_{m-1}(z) \\ D_{m-1}(z) &= \sigma_{11}D_m(z) \\ E_{m-1}(z) &= k_cE_m(z)-kz^{-1}N_{m-1}(z) \end{aligned} \tag{12-83}$$

简单的代数操作之后，可将 $Z^{-1}N_{m-1}(z)$、$D_{m-1}(z)$ 和 $E_{m-1}(z)$ 用项 $N_m(z)$、$D_m(z)$ 和 $E_m(z)$ 表示为：

$$\begin{bmatrix} z^{-1}N_{m-1}(z) \\ D_{m-1}(z) \\ E_{m-1}(z) \end{bmatrix} = \frac{1}{k_c}\begin{bmatrix} 1 & -\sigma_{10} & -k \\ 0 & k_c\sigma_{11} & 0 \\ -k & k\sigma_{10} & k_c^2+k^2 \end{bmatrix}\begin{bmatrix} N_m(z) \\ D_m(z) \\ E_m(z) \end{bmatrix} \tag{12-84}$$

只要将 σ_{11} 替换为1，这个关系可直接应用归一化格型滤波器。通过类似的处理，可推导出该关系的基本和单乘法器格型滤波器形式。

考虑一个实现N阶传输函数 $H(z)=N_N(z)/D_N(z)$ 的格型滤波器。如果滤波器的输入为 $D_N(z)$，那么输出应为 $N_N(z)$。所以，对于模块N，根据传输函数可确定 $N_N(z)$ 和 $D_N(z)$。同时，注意对于模块N有 $E_N(z)$ 为零。然后模块N右边的多项式用上面的矩阵乘法求出。同理，求出对于 $m=N, N-1, \ldots, 1$ 的每个模块 m 的多项式。当得到和内部节点相关的多项式，输出舍入噪声即可用12.2.4节讨论的方法求出。

例12.7.1 再分析例12.5.1讨论的归一化格型滤波器。图12-25给出了图12-16的归一化格型滤波器的转置图。从转置图的输入到内部节点1-9的传输函数也是图12-16的初始图中从相应的内部节点到输出的传输函数。和转置图的节点1-9相关的多项式为：

455

$$\begin{aligned} 1 &: \ 6.3469z^3+1.2645z^2+1.8093z \\ 2,5,8 &: \ 345.1z^3-819.3z^2+665.8z-183.6 \\ 3 &: \ 3.3767z^2+0.6727z+0.9626 \end{aligned}$$

$$
\begin{aligned}
4 &: \ 35.7147z^3 - 29.9385z^2 + 6.7185z \\
6 &: \ -31.9404z^2 + 28.0689z - 5.8876 \\
7 &: \ 105.4016z^3 - 154.7295z^2 + 58.914z \\
9 &: \ 95.5039z^2 - 144.4369z - 56.0758
\end{aligned}
$$

对每个多项式应用多项式展开算法，得到内部节点1-9对应的输出舍入噪声为：

{0.1455，1，0.0412，0.2952，1，0.1912，0.3096，1，0.2163}

把节点1-7的噪声求和，得到图12-16的归一化格型滤波器总输出舍入噪声为2.9827。（注意节点9不产生舍入噪声，见图12-16，因为用了倍长累加器模块，节点8的舍入噪声求值同节点5）。 ■

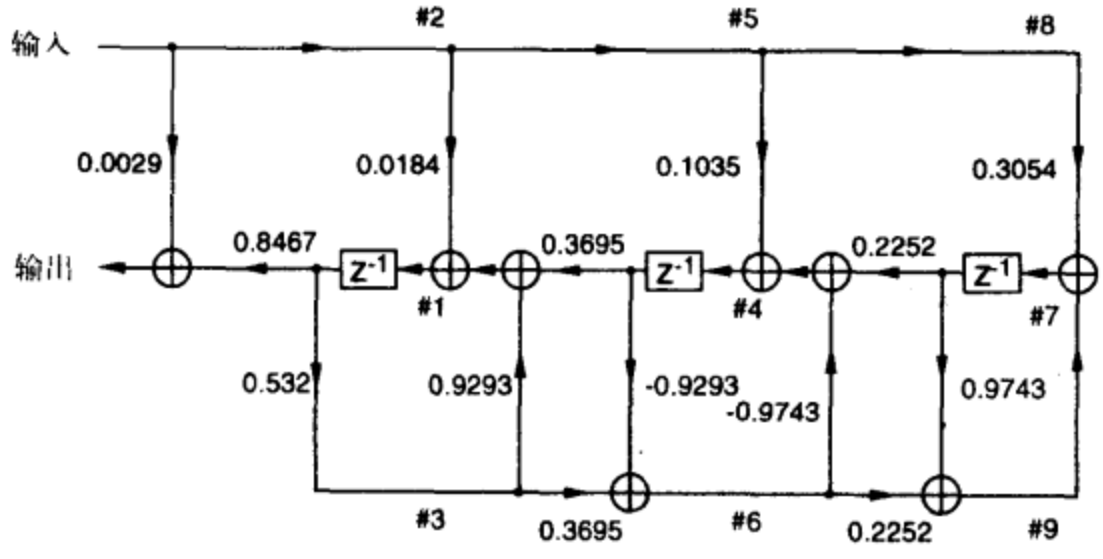


图12-25 图12-16的归一化格型滤波器的转置图

12.7.2 舍入噪声比较

所有稳定的传输函数都可用可缩放归一化格型滤波器或者逆可缩放归一化格型滤波器实现。除了这些结构，还可线性组合Schur多项式和逆Schur多项式计算逐轮获取参数。这种情况下，逐轮获取参数同时和Schur多项式方面及逆Schur多项式方面有关。因此，有必要检验这些结构各自的舍入噪声特性。 456

在[13]中，可见最好的舍入噪声特性在逐轮获取参数和Schur多项式方面相关时达到。考虑图12-21b中没有逐轮获取参数的纯格型部分。令 $S_{ij}(z)$ 为从状态 x_j 到 x_i 的传输函数，其中 x_j 为模块 j 的延时。然后，在第 j 个节点注入的噪声对第 i 个节点的效果表示为如下形式：

$$\|S_{ij}\|^2 = \frac{1}{2\pi}\int_0^{2\pi} |S_{ij}(e^{j\omega})|^2 \mathrm{d}\omega \tag{12-85}$$

下一步，考虑只将第 i 个延时由上升移到下降沿的同样结构。这个新结构从 x_j 到 x_i 的传输函数表示为 $T_{ij}(z)$。然后，应用状态变量描述，由[13]推导出如下恒等式：

$$\|S_{ij}\|^2 - \|T_{ij}\|^2 = \begin{cases} 1, & 1 \leqslant j \leqslant i-1 \\ 0, & i \leqslant j \leqslant n \end{cases} \tag{12-86}$$

公式（12-86）应用如下：在包含两端的纯格型部分添加任意的割集。对于割集的右边，和割集的上升沿有关的变量将比和下降沿有关的变量少受状态节点产生的舍入噪声的影响。

如果噪声源在割集的左边，则对上升和下降变量效果相同。这表示减少由Schur多项式方面产生的逐轮获取参数可改良舍入的误差。

例12.7.2 根据图12-20的可缩放归一化格型滤波器，求得节点1-7的输出舍入噪声为{0.1043, 0.104, 0.1455, 0.2952, 0.3096, 0.0412, 0.1912}。（参考例12.6.1）。通过对每个节点的噪声求和，可得到可缩放归一化格型滤波器的总输出舍入噪声为1.191。因为输出信号可表示为 $\sum_{i=0}^{3} c_i\Phi_i(z)$，输出信号功率为 $\sum_{i=0}^{3} c_i^2 = 0.10434$。因此，滤波器信噪比（NSR）为1.191/0.10434 = 11.4146。

这个可缩放归一化格型滤波器所用的传输函数再应用图12-23的逆Schur多项式实现。（参考例12.6.2。）根据节点1-8得到的输出舍入噪声为{0.1432，0.19，0.2057，0.0933，0.1428，0.19，0.104，0.1043}。通过对每个节点的噪声求和，逆可缩放归一化格型滤波器的总输出舍入噪声为1.1734。因为输出信号噪声为0.10434，逆可缩放归一化格型滤波器的信噪比为1.1734/0.10434 = 11.246。 ■

例12.7.2中，逆可缩放归一化格型滤波器的信噪比几乎和可缩放归一化格型滤波器相同。下面的例子将清楚地证明公式（12-86）的正确性。

457

例12.7.3 分析一个3阶椭圆低通滤波器：

$$H(z) = \frac{0.3296(z^3 + 1.61165z^2 + 1.61165z + 1)}{z^3 + 0.0522z^2 + 0.7139z - 0.0445}$$

参数 k 和逐轮获取参数计算如下：

$$\begin{array}{lll} k_3 = -0.0445, & k_2 = 0.71766, & k_1 = -0.049 \\ \sigma_{10}^{(3)} = 0.3296, & \sigma_{10}^{(2)} = 0.8141 & \sigma_{10}^{(1)} = 0.9894 \\ \sigma_{11}^{(3)} = 0.632, & \sigma_{11}^{(2)} = 0.5807, & \sigma_{11}^{(1)} = 0.145 \end{array}$$

输出信号功率为0.508。可缩放归一化格型滤波器的信噪比为2.863，而逆可缩放归一化格型滤波器的信噪比为5.441。因此，可缩放归一化格型滤波器的信噪比只有逆可缩放归一化格型滤波器的52.8%。 ■

12.8 格型IIR数字滤波器的流水线

虽然格型滤波器可通过割集区域化进程完成流水线，这些流水线滤波器的最大样本率受到反馈回路计算的限制。例如，在归一化格型滤波器中，割集区域化进程（见第4章）可用来传输图12-26中每个由向右的端点到向左的端点的延时的一半。半延时可用时间重组来实现。例如，若用1个时钟周期表示半延时，可以每两个时钟输入一个样本同时每两个时钟产生一次输出样本。这种结构的最大样本率局限于反馈回路的计算，包括2次乘法和2次加法。

应用Schur多项式，可以看到如果传输函数的分母是离散预测格式，那么传输函数满足格型数字滤波器的流水性质[14]，[15]。但离散预测技术的缺点是引入补偿零点。这些零点增加了实现数字滤波器的乘法操作并且导致了混合实现的零点-极点补偿不精确。为了避免补偿零点的缺点，可用使用受限滤波器设计方案设计的传输函数流水线。一种方案是用降抽样滤波器设计的滤波器设计方式[16]~[19]。另外一种同样使用降抽样滤波器设计的方案可见[20]，[21]。

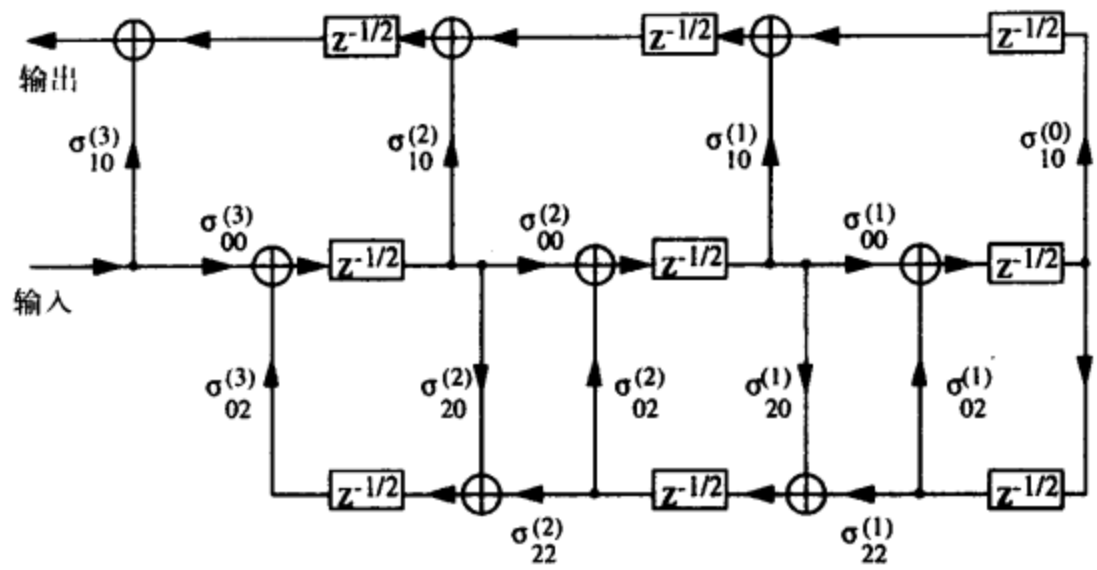


图12-26 传输半延时的3阶归一化格型滤波器

12.8.1 Schur算法的流水线性质

一个 i 阶实Schur多项式表示为:

$$\Phi_i(z) = \sum_{m=0}^{i} \phi_m^{(i)} z^m \tag{12-87}$$

为了求出基本、单乘法器、归一化和可缩放归一化格型滤波器的参数组 k，需要用Schur算法根据 $\Phi_N(z) = D_N(z)$ 计算Schur多项式 $\{\Phi_{N-1}(z), \Phi_{N-2}(z), \ldots, \Phi_0(z)\}$。$\Phi_{i-1}(z)$ 按 z 指数增大顺序的幂系数是 i 个2×2矩阵行列式的 $\left\{\left(\phi_i^{(i)}\right)^2 - \left(\phi_0^{(i)}\right)^2\right\}^{-\frac{1}{2}}$ 倍，这些矩阵由下面矩阵的第一列和后面的每一列组成:

$$\begin{bmatrix} \phi_i^{(i)} & \phi_{i-1}^{(i)} & \phi_{i-2}^{(i)} & \cdots & \phi_2^{(i)} & \phi_1^{(i)} & \phi_0^{(i)} \\ \phi_0^{(i)} & \phi_1^{(i)} & \phi_2^{(i)} & \cdots & \phi_{i-2}^{(i)} & \phi_{i-1}^{(i)} & \phi_i^{(i)} \end{bmatrix} \tag{12-88}$$

对于每个2×2子矩阵，如果某列全部由零元素组成，相应的行列式为零。应用行列式的这个性质，可得到格型滤波器流水线操作一个有用的性质。

Schur算法的零系数性质

如果 $\Phi_N(z)$ 是一个N阶Schur多项式，且在每两个最近次数的非零系数之间有 j 个连续零系数，那么 $(N-i)$ 阶多项式 $\Phi_{N-i}(z)$ 也在每两个最近次数的非零系数之间有 j 个连续零系数，也就是对于 $i=1$ 到 $N-j-1$ 都为零位，而在 $i=N-j$ 到 N 只有一位非零系数。

例12.8.1 分析多项式:

$$D_8(z) = z^8 + 0.63459z^4 + 0.10067$$

458 ~ 459

注意 $D_8(z)$ 在每两个非零系数 $(j=3)$ 之间有3位连续零系数。应用Schur算法，求得Schur多项式为:

$$\begin{aligned} \Phi_8(z) &= z^8 + 0.63459z^4 + 0.10067 \\ \Phi_7(z) &= 0.99492z^7 + 0.57362z^3 \end{aligned}$$

$$
\begin{aligned}
\Phi_6(z) &= 0.99492z^6 + 0.57362z^2 \\
\Phi_5(z) &= 0.99492z^5 + 0.57362z \\
\Phi_4(z) &= 0.99429z^4 + 0.57362 \\
\Phi_3(z) &= 0.81291z^3 \\
\Phi_2(z) &= 0.81291z^2 \\
\Phi_1(z) &= 0.81291z \\
\Phi_0(z) &= 0.81291
\end{aligned}
$$

多项式 $\Phi_8(z)$、$\Phi_7(z)$、$\Phi_6(z)$、$\Phi_5(z)$ 和 $\Phi_4(z)$ 在两个最近次数的非零系数之间有连续3位零系数，而其他多项式只有一位非零系数。 ■

考虑 $\Phi_N(z)$，它有一位非零常数项且在每两个最近次数的非零系数之间有 j 位连续零系数。现在，和 $\Phi_N(z)$ 相比，$\Phi_{N-1}(z)$ 的次数减少1，但是 $\Phi_{N-1}(z)$ 在两个最近次数的非零系数之间应有 j 位连续零系数。因此，$\Phi_{N-1}(z)$ 在 $j>1$ 时应包含零常数项。这个结论可以推广如下。

Schur多项式的零常数项性质:

如果 $\Phi_N(z)$ 是 N 阶Schur多项式，有一位非零项且在两个最近次数的非零系数之间有 j 位连续零系数，则如果 i 不是 $(j+1)$的倍数，多项式 $\Phi_{N-i}(z)$ 含有零常数项。

例12.8.2 分析例12.8.1的Schur多项式。

$\Phi_8(z)$ 常数项非零且对应 $j=3$。则含有非零常数项的多项式为:

(1) $\Phi_{8-0.4}(z)=\Phi_8(z)$, (2) $\Phi_{8-1.4}(z)=\Phi_4(z)$; (3) $\Phi_{8-2.4}(z)=\Phi_0(z)$。 ■

12.8.2 基本格型滤波器的流水线

2阶基本格型滤波器结构如图12-27所示。请注意逐轮获取参数 $\{\{c_2, c_1, c_0\}$ 组成前馈部分，而反馈部分只用参数组 k 描述。这种结构的迭代边界受限于2次乘法和3次加法操作的时间。在图12-27中，如果模块1的参数 k（也就是 k_1）为零，则去掉模块1的反馈回路。然后将模块1的延时在模块2中共享，从而反馈回路中有2个延时，这将导致基本格型滤波器的2级流水线。

460

同样，如果一个N阶基本格型滤波器连续 $(M-1)$ 个参数 k 为零，则去掉这些模块的反馈回路，从而每个回路含有M个延时，导致基本格型滤波器的M级流水线。

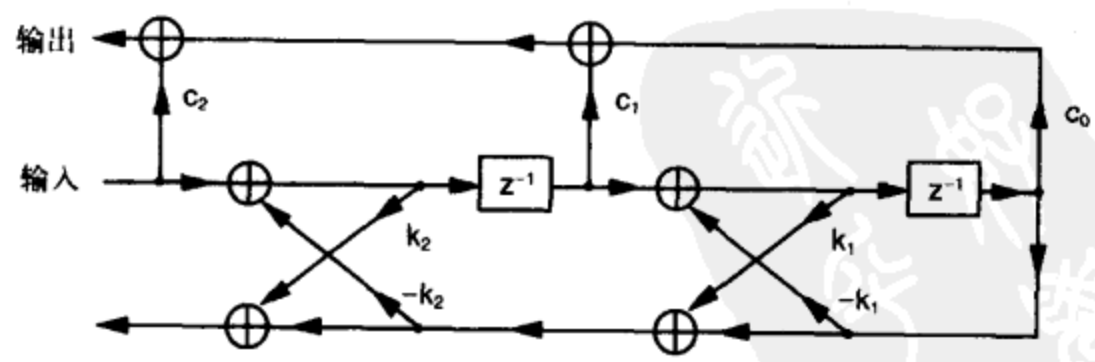


图12-27 2阶基本格型滤波器

N阶格型滤波器的参数组 k 计算公式为:

$$k_i = \frac{\Psi_i(0)}{\Psi_i^*(0)} = \frac{\Phi_i(0)}{\Phi_i^*(0)}, \quad i = N, N-1, \cdots, 1 \tag{12-89}$$

所以，为了让模块 i 的参数 k 为零，$\Phi_i(z)$ 的常数项要为零。根据Schur多项式的零常数性质，如果 $D_N(z)$ 有非零常数项且每两个最近次数的非零系数间有 $(M-1)$ 位连续零系数，则只有 $\{\Phi_N(z), \Phi_{N-M}(z), \Phi_{N-2M}(z), \ldots, \Phi_0(z)\}$ 含有非零常数项。因此，有 $(M-1)$ 位连续零参数 k从而每个回路含 M 个延时，这将导致基本格型滤波器的 M 级流水线[14]。

每个回路的 M 个延时可再分配到合适的位置来进行回路的乘-加操作流水。同时，请注意在前馈部分的乘-加操作可在任意级别流水，只要在合适的前馈割集部分添加自锁电路即可。因为在每个回路乘-加操作的次数仍是常数，M 级的流水线滤波器可按比非流水线滤波器快 M 倍的速度计时。

例12.8.3 分析如下传输函数：

$$H(z)=\frac{z^4+3.9z^3+5.7z^2+3.7z+0.9}{z^4-0.17z^2-0.5184}$$

注意分母在每两个最近次数的非零系数之间有一位零系数。根据分母，参数组 k 求得为： 461

$$k_4=-0.518,\ k_3=0,\ k_2=-0.353,\ k_1=0$$

求出的系数组 k 清楚地说明该传输函数可用2级流水线的基本格型滤波器实现。图12-28给出应用逆Schur多项式设计的2级流水线基本格型滤波器。在非流水线滤波器中，每个回路只有一个延时。但是，2级流水线滤波器每个回路有2个延时，可按比非流水线滤波器快2倍的速度计时。 ■

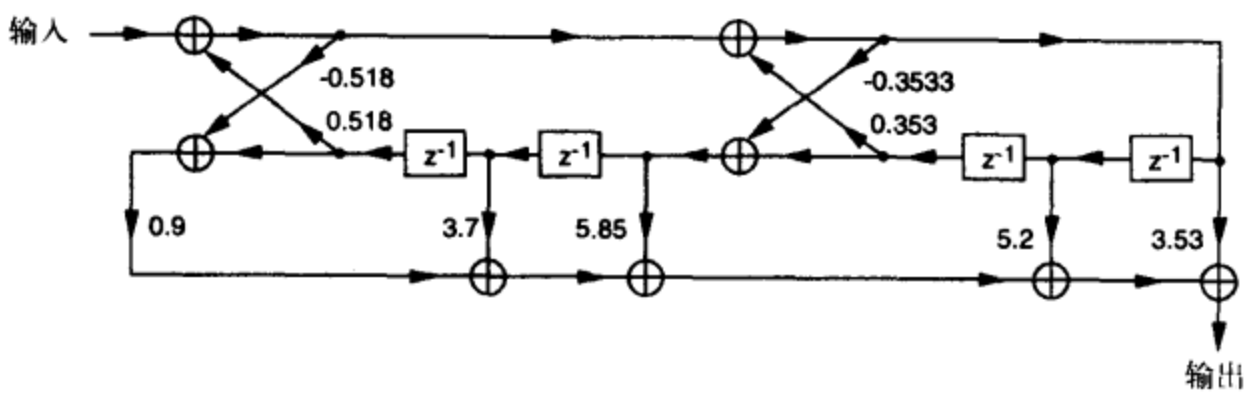


图12-28 例12.8.3的2级流水线基本格型滤波器

12.8.3 单乘法器格型滤波器的流水线

图12-29给出了一个2阶单乘法器的结构。请注意逐轮获取参数 $\{c_2, c_1, c_0\}$ 组成了前馈部分，反馈部分仅由参数组 k 和标志量 $\{\varepsilon_2, \varepsilon_1, \varepsilon_i=\pm 1\}$ 描述。该结构的迭代边界（见第2章）受限于2次乘法和4次加法操作。由图12-29可见，模块 i 的反馈回路当模块 i 的参数 k 为零时将去掉。因此，通过去掉 $(M-1)$ 个连续的参数 k，单乘法器格型滤波器可用 M 级流水线。

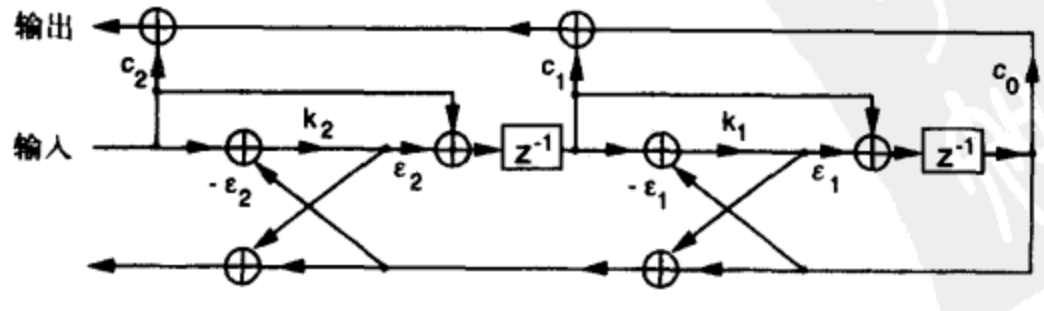


图12-29 2阶单乘法器格型滤波器

462 N 阶单乘法器格型滤波器的 k 参数计算公式为:

$$k_i = \frac{\Lambda_i(0)}{\Lambda_i^*(0)}, \quad i = N,\ N-1,\ \cdots,\ 1 \tag{12-90}$$

请注意 $\Lambda_i(z)$ 和 $\Phi_i(z)$ 只相差一个缩放因数。也就是说，根据 $\Lambda_i(z)$ 求得的 k 参数等于根据 $\Phi_i(z)$ 求得的 k 参数。因此，显然单乘法器格型滤波器的反馈部分可用和基本格型滤波器同样的方法进行流水线操作。而前馈部分的乘-加操作也可进行任意级别的流水，只要在合适的前馈割集位置添加自锁电路即可。

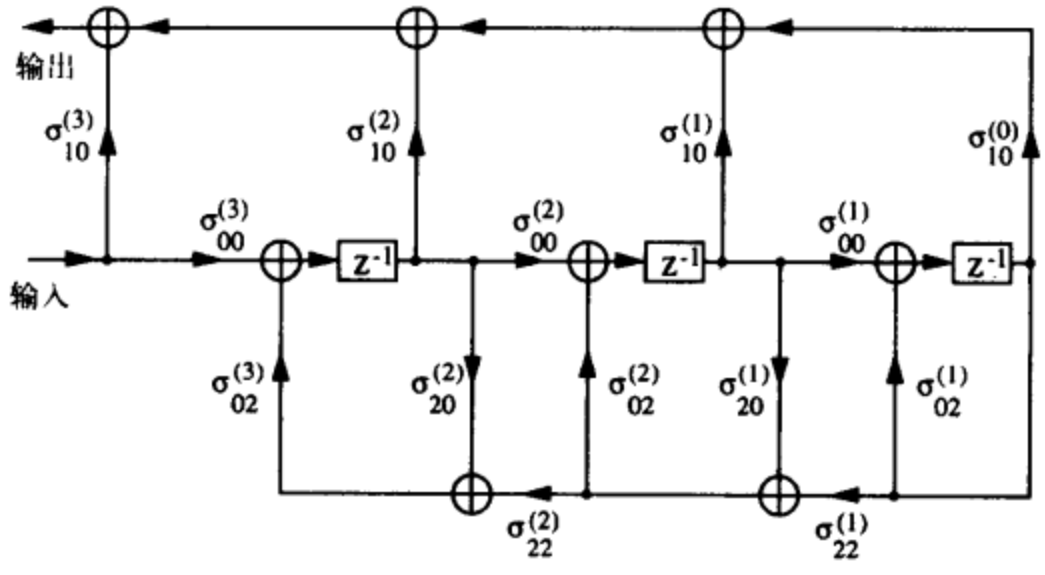


图12-30 3阶归一化格型滤波器

12.8.4 归一化格型滤波器的流水线

图12-30给出了一个3阶归一化格型滤波器结构。请注意逐轮获取参数 $\left\{\sigma_{10}^{(3)},\ \sigma_{10}^{(2)},\ \sigma_{10}^{(1)},\ \sigma_{10}^{(0)}\right\}$ 组成前馈部分，反馈部分则由参数 $\left\{\sigma_{00}^{(i)},\ \sigma_{02}^{(i)},\ \sigma_{20}^{(i)},\ \sigma_{22}^{(i)}\right\}$ 描述，这里 i=1 到3。回忆一下公式(12-71)，

$$\sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = k_i,\ \sigma_{00}^{(i)} = \sigma_{22}^{(i)} = \sqrt{1-k_i^2} \tag{12-91}$$

所以，前馈部分只由参数 k 描述。如果模块 i 的参数 k 为零，则 $\sigma_{20}^{(i)} = -\sigma_{02}^{(i)} = 0$，且 $\sigma_{00}^{(i)} = \sigma_{22}^{(i)} = 1$。所以，模块 i 的反馈回路可去掉，而模块的延时和邻近模块共享。同理，如果连续 $(M-1)$ 个
463 模块的参数 k 都为零，则这些反馈回路都可以去掉，从而每个回路有 M 个延时，即导致 M 级流水线归一化格型滤波器。因为模块 i 的参数 k 是根据 $\Phi_i(z)$ 求出的，归一化格型滤波器也可用和基本格型滤波器同样的方法完成流水线。

12.8.5 可缩放归一化格型滤波器的流水线

可缩放归一化格型滤波器的反馈部分和归一化格型滤波器完全相同。因此，如果分母在两个最近次数的非零系数间有 $(M-1)$ 位连续零系数，则反馈部分可使用 M 级流水线。而前馈部分的乘-加操作可用任意级流水线，只要在合适的前馈割集位置添加自锁电路即可。

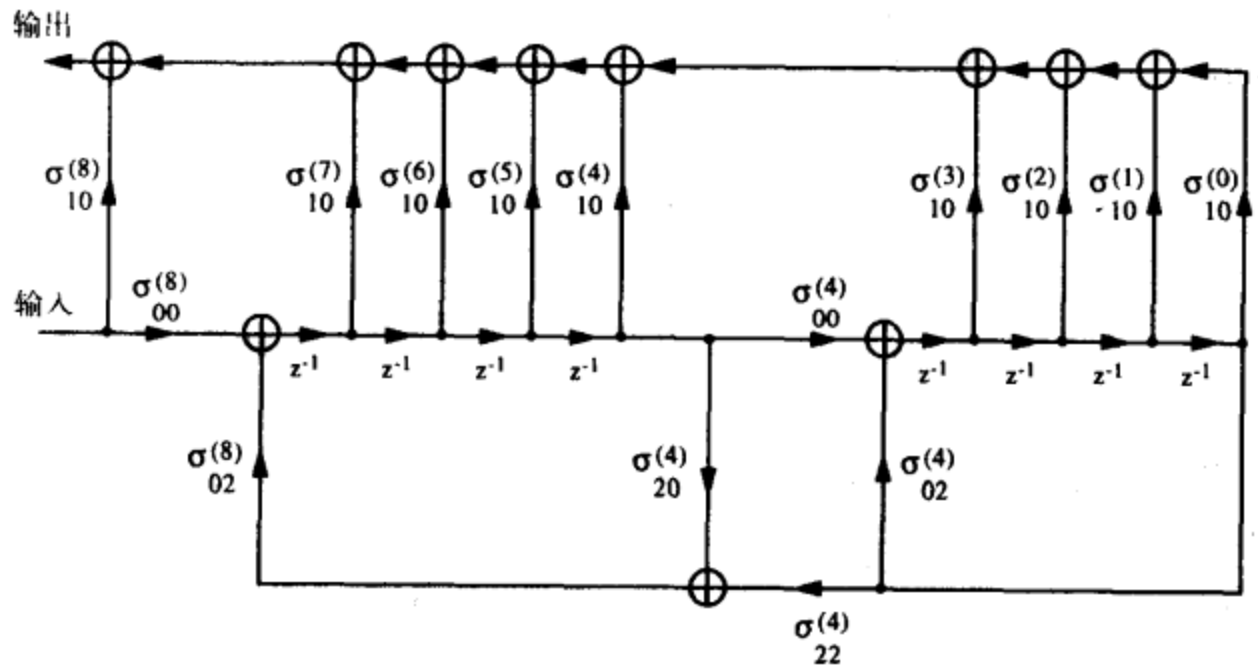
12.8.6 重定时流水线的格型滤波器

根据前面的讨论，显然如果分母在每两个最近次数的非零系数间有 $(M-1)$ 位零系数，则基本、单乘法器、归一化和可缩放归一化格型滤波器都可使用 M 级流水线。每个回路的 M 个延时可再分配到合适的位置来完成回路乘-加操作的流水操作。

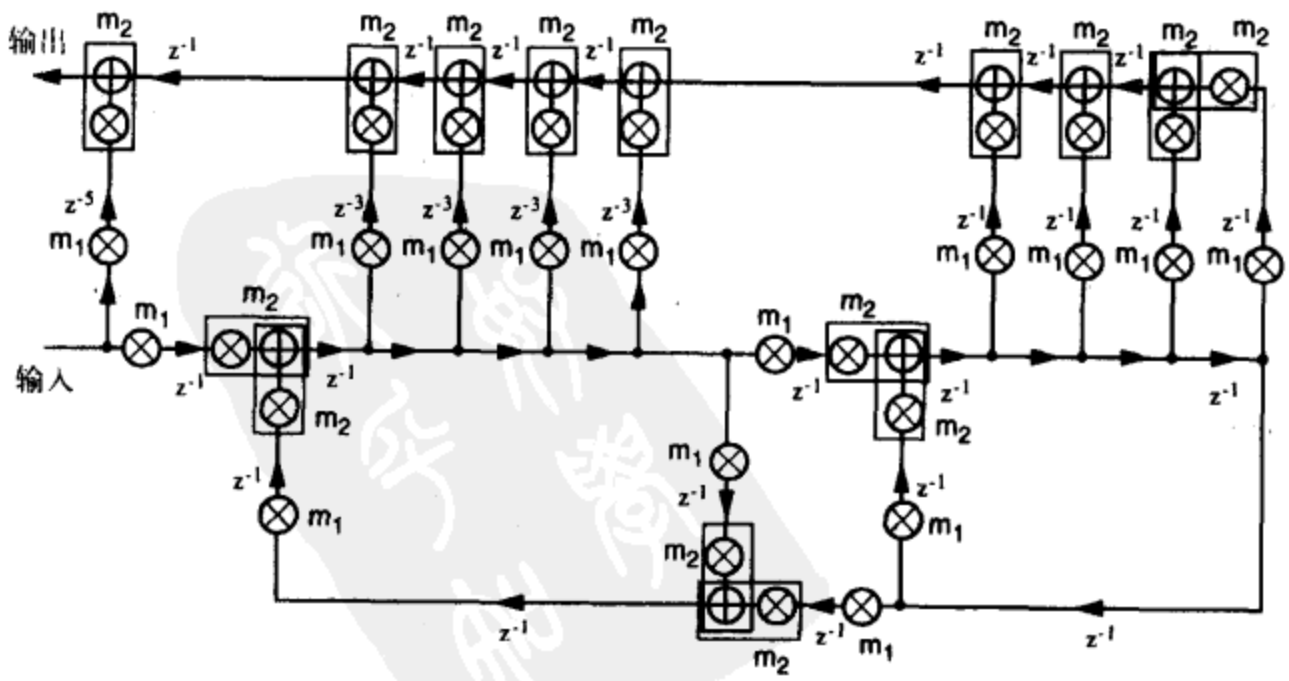
举个例子，考虑图12-31a所示的一个4级归一化格型滤波器的流水线。因为每个反馈回路有4个延时，时钟周期 T_{clk} 范围为:

$$T_{clk} \geqslant \frac{2(\text{乘-加})}{4} = \frac{\text{乘-加}}{2} \tag{12-92}$$

因此图12-31中的乘-加操作可分为 m_1 和 m_2 两部分，使得：计算时间 (m_1) = 计算时间 (m_2) = 计算时间（乘-加）/ 2。图12-31b给出用重定时（见第4章）得到的电路，由于非流水线电路的时钟周期不超过2次乘-加操作，这里的电路操作可比非流水线电路快4倍。



a) 重定时前的4级归一化格型滤波器流水线



b) 重定时后的4级归一化格型滤波器流水线

图 12-31

计算时间 (m_1) = 计算时间 (m_2) = 计算时间（乘&加）/ 2

12.9 流水线格型滤波器的设计例子

464 ~ 465

为了对基本、单乘法器、归一化和可缩放归一化格型滤波器用 M 级流水线，传输函数的分母要在每两个最近次数的非零系数之间有 $(M-1)$ 位连续零系数。可流水线传输函数可通过对非流水线滤波器传输函数应用离散预测理论得到。为了避免补偿零点会在有限字长实现中增加硬件或零点-极点补偿不精确的缺点，可流水线的传输函数也可直接根据滤波器波谱设计，前提为分母是关于 z^M 而不是 z 的多项式。

在本节中，将给出两个流水线格型滤波器的例子。这里的可流水线传输函数应用改进Deczky方法得到[19]。

例12.9.1 分析一个4段可流水线低通滤波器，要求导通频带：0~0.2π (0.5 dB) 且阻断频带：0.3π~π(20 dB)。应用改进Deczky方法得到的传输函数为：

$$
\begin{aligned}
N(z) &= 0.09742(1-1.16548z^{-1}+z^{-2})(1-0.29069z^{-1}+z^{-2}) \\
&\quad (1+1.19469z^{-1}+z^{-2})(1+1.75885z^{-1}+z^{-2}) \\
D(z) &= 1+0.63459z^{-4}+0.10067z^{-8}
\end{aligned}
$$

注意 $D(z)$ 可进一步分解为 $D(z) = (1+0.31729z^{-4})^2$。所有零点都在阻断频带的单位周期内，且极点等效半径为0.75，等效角度差0.5π。传输函数的极点-零点图如图12-32a所示。注意例12.8.1中 $D(z)$ 是作为Schur多项式的。因为Schur多项式 $\phi_m(z)$ 在 m=1 到3和5到7时没有常数项，即对于 m=1 到3和5到7的 $k^{(m)} = 0$。所以，每个回路有4个延时。一个可缩放归一化格型滤波器的传输函数的实现如图12-32b所示。当每个回路乘-加操作的次数和非流水线滤波器相同时，每个回路的延时数增加到4。因此，4级流水线滤波器可通过在合适的位置再分配延时达到原来操作4倍的速度。 ■

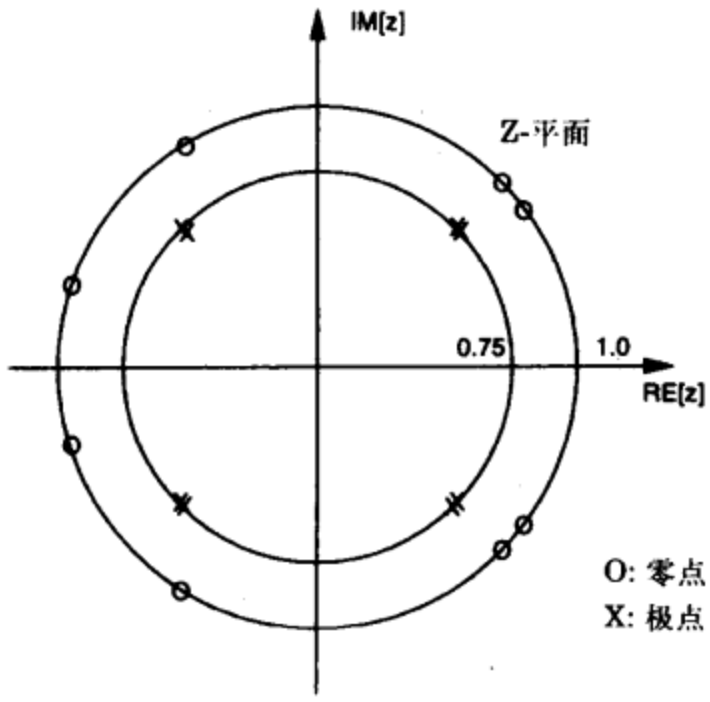
例12.9.2 分析如下低通滤波器：

导通频带：0~0.3π（0.2 dB）且阻断频带：0.36π~π（40 dB）。这个规格可用一个6阶椭圆滤波器完成。对于 M=1、2和4的可流水线传输函数，通过改进Deczky方法可以得到。因为每个 M 的分子次数大于分母，分子可表示为 $N_f(z)\,N_i(z)$，其中 $N_f(z)$ 用一个FIR滤波器实现而 $N_i(z)\,/\,D(z)$ 通过一个可缩放归一化格型滤波器实现。这里的实现中，FIR滤波器输出连接到IIR格型滤波器的输入。

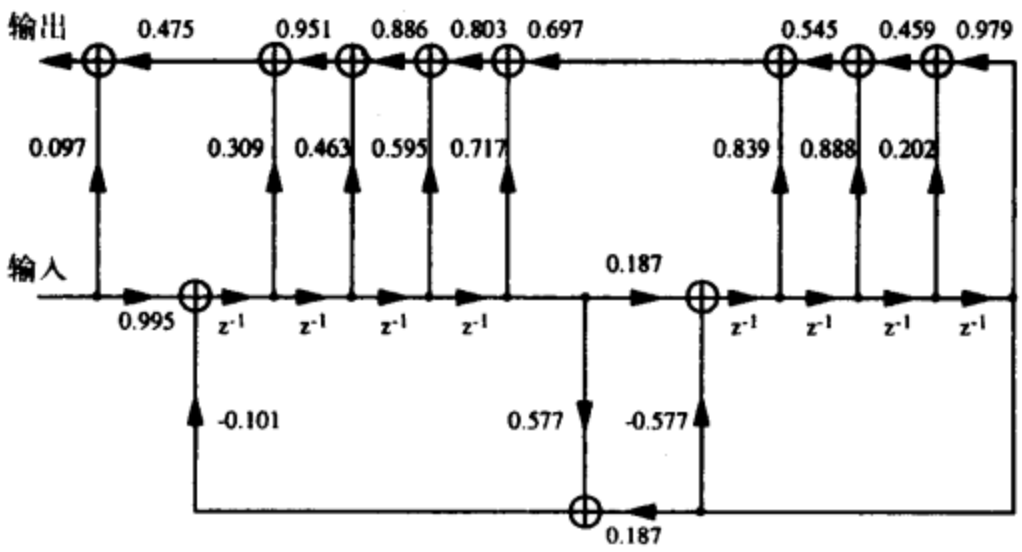
M = 1情况：

$$
\begin{aligned}
N_f(z) &= 0.11+0.3624z^{-1}+0.5972z^{-2}+0.5972z^{-3}+0.3624z^{-4}+ \\
&\quad 0.11z^{-5} \\
N_i(z) &= 0.1282-0.1739z^{-1}+0.3117z^{-2}-0.1739z^{-3}+0.1282z^{-4} \\
D(z) &= 1-2.0874z-1+2.5698z^{-2}-1.533z^{-3}+0.5227z^{-4}
\end{aligned}
$$

466



a) 例12.9.1的传输函数的极点-零点图



b) 例12.9.1的可缩放归一化格型滤波器的4段流水线结构

图 12-32

M = 2 情况:

$$
\begin{aligned}
N_f(z) &= 0.4539 + 0.5311z^{-1} + 0.1543z^{-2} + 0.5311z^{-3} + 0.4539z^{-4} \\
N_i(z) &= 0.0434 + 0.1042z^{-1} + 0.223z^{-2} + 0.2864z^{-3} + 0.3448z^{-4} + \\
&\quad 0.2864z^{-5} + 0.223z^{-6} + 0.1042z^{-7} + 0.0434z^{-8} \\
D(z) &= 1 + 0.7802z^{-2} + 1.2014z^{-4} + 0.3109z^{-6} + 0.2418z^{-8}
\end{aligned}
$$

M = 4 情况:

$$
\begin{aligned}
N_f(z) &= 0.1436 + 0.4896z^{-1} + 0.6923z^{-2} + 0.4896z^{-3} + 0.1436z^{-4} \\
N_i(z) &= 0.1365 + 0.0232z^{-1} + 0.2879z^{-2} + 0.0557z^{-3} + 0.3954z^{-4} + \\
&\quad 0.0443z^{-5} + 0.4058z^{-6} + 0.0472z^{-7} + 0.3867z^{-8} + 0.0532z^{-9} + \\
&\quad 0.2952z^{-10} + 0.0229z^{-11} + 0.1717z^{-12} + 0.0106z^{-13} + \\
&\quad 0.0528z^{-14} + 0.0056z^{-15} + 0.0329z^{-16} \\
D(z) &= 1 + 1.794z^{-4} + 1.4404z^{-8} + 0.483z^{-12} + 0.0582z^{-16}
\end{aligned}
$$

由于每个 $N_f(z)$ 的系数的对称性，FIR部分可基于直接式结构有效地实现。另外，为了保证可缩放归一化格型滤波器的单位内部功率的性质，FIR部分的所有内部节点缩放到单位功率。对6阶椭圆滤波器应用离散预测变换得到受限滤波器和流水线椭圆滤波器的乘/加操作次数和舍入噪声数值，二者的比较见表12-1。舍入噪声虽然依赖于极点位置略有不同，但硬件复杂度随着 M 增大将由受限设计方案大规模降低。 ■

表12-1 M = 1、2和4时可缩放归一化格型滤波器的乘/加操作次数以及舍入噪声

	受限滤波器			流水线椭圆滤波器		
M	乘法	加法	噪声	乘法	加法	噪声
1	31	15	2.2850	34	17	3.6477
2	37	18	2.8899	46	23	2.7247
4	53	26	1.0527	70	35	2.3608

467 ~ 468

12.10 低功耗CMOS格型IIR滤波器

如第3章所解释，流水线的使用可在相同抽样速度下的功耗缩减可以达到很夸张的程度。在下面的例子中可以看到流水线可用于可缩放归一化格型滤波器的低功耗实现。

例12.10.1 本例分析满足例12.9.2规格的可缩放归一化格型滤波器的低功耗实现。假设乘法器电容占优势且加法器电容可忽略。同时，假设初始系统（$M = 1$）电源电压为5 V 且CMOS开启电压为0.5 V。那么初始系统的传播延迟为：

$$T_o = \frac{C_l \times 5}{\kappa(5-0.5)^2} \tag{12-93}$$

下面，考虑4段（$M = 4$）流水线系统。对于低功耗实现，保持时钟速度和初始系统相同。那么，对于流水线系统，

$$T_o = \frac{1}{4}\frac{C_l V_p}{\kappa(V_p-0.5)^2} \tag{12-94}$$

根据公式（12-93）和公式（12-94），$V_p = 1.879$ V。因此，流水线系统的电源电压可降低到1.879 V。

初始系统的功耗为：

$$P_o = \frac{C_l V_o^2}{T_o} \tag{12-95}$$

另一方面，流水线系统的功耗为：

$$P_p = \frac{m_P}{m_o}\frac{C_l V_p^2}{T_o} \tag{12-96}$$

其中 m_p 和 m_0 分别是4段流水线系统和初始系统的乘法器个数（参考表12-1）。则有，

$$\text{功率节省} = \frac{P_p}{P_o} = \left(\frac{m_P}{m_o}\right)\left(\frac{V_p}{V_o}\right)^2 = 0.2414 \tag{12-97}$$

因此，4段流水线系统的功耗只有初始系统的24.14%。注意这里的功耗减少是以面积增加70%为代价的。 ■ 469

12.11 结论

本章给出了4种格型滤波器的设计，包括基本、单乘法器、归一化和可缩放归一化，可以看出离散预测流水线传输函数经常导致流水线格型IIR数字滤波器；还给出了一种应用多项式扩展原理和转置定理计算格型数字滤波器的舍入噪声的方法。格型滤波器设计程序也可用于IIR和FIR数字滤波器设计。

12.12 习题

1. 分析正规直交Schur多项式的基 $\{\Phi_i(z)\}$。从定义开始，证明下式:
   (a) $\langle \Phi_{N-1}(z), \Phi_{N-1}(z) \rangle = 1$
   (b) $\langle \Phi_N(z), \Phi_{N-1}(z) \rangle = 0$
   (c) $\langle \Phi_N(z), \Phi_{N-2}(z) \rangle = 0$
2. 证明下列内积恒等式:
   (a) $\langle \Phi_N^*(z), \Phi_{N-1}^*(z) \rangle = \sqrt{1-k_N^2}$
   (b) $\langle \Phi_N^*(z), \Phi_{N-2}^*(z) \rangle = \sqrt{(1-k_N^2)(1-k_{N-1}^2)}$
3. 分析如下传输函数:

$$H(z) = \frac{1+3z^{-1}+3z^{-2}+z^{-3}}{1-0.9z^{-1}+0.64z^{-2}-0.576z^{-3}}$$

   (a) 根据分母应用Schur算法得出正规直交基函数;
   (b) 用(a)得到的正规直交多项式展开分子;
   (c) 计算当输入是未知广义固定的随机产生的单位功率时的输出信号功率;
   (d) 求出一个缩放常数 a，令缩放传输函数 $aH(z)$ 输出信号为单位功率。
4. 分析如下传输函数:

$$H(z) = \frac{1+5/12z^{-1}+1/24z^{-2}}{1-9/8z^{-1}+5/16z^{-2}}$$ 470

   (a) 导出两个 $H(z)$ 的基本格型滤波器结构，其中分子用Schur多项式和逆Schur多项式展开;
   (b) 应用离散预测将 $H(z)$ 变换为2级流水线格式。分别推导两个基本格型滤波器为2级流水线滤波器;
   (c) 分别针对(a)和(b)设计，计算这样4个格型滤波器的舍入噪声值:
      i 基于状态变量描述的W矩阵方法;
      ii 多项式展开方式。
   (d) 假设2级流水线格型滤波器和初始非流水线格型滤波器将在相同取样速度下应用，

流水线结构能减少多少功耗？功率计算时，假设初始顺序系统的电源电压为5 V且CMOS开启电压为0.5 V。假设电容和关键路径属于乘法器的部分占统治地位而加法器部分可忽略不计。

5. 对于第3题的 $H(z)$ 求基本、单乘法器、归一化和可缩放归一化格型滤波器结构。
6. 对第3题的 $H(z)$ 用离散预测变换求2级流水线传输函数。对于流水线传输函数，求基本、单乘法器、归一化和可缩放归一化格型滤波器结构。
7. 对下列2阶传输函数求单乘法器格型滤波器结构:

$$H(z)=\frac{(1+z^{-1})^2}{1-3/(2\sqrt{2})z^{-1}+9/16z^{-2}}$$

这个滤波器必须根据逆Schur多项式方式展开分子进行设计。

8. 分析如下传输函数:

$$H(z)=\frac{1/2+1/4z^{-1}+z^{-2}}{1+1/4z^{-1}+1/2z^{-2}}$$

(a) 求Schur多项式 $\Phi_2(z)$、$\Phi_1(z)$ 和 $\Phi_0(z)$;

(b) 用(a)得到的Schur多项式展开分子。再由展开系数，用归一化格型滤波器实现 $H(z)$;

471

(c) 说明(b)的结构效率不高。也就是说明不用逐轮获取参数也可得到正确的输出;

(d) 改进(b)得到的结构实现传输函数:

$$H(z)=\frac{1}{1+1/4z^{-1}+1/2z^{-2}}$$

9. 对如下传输函数推导基本、单乘法器、归一化和可缩放归一化格型滤波器:

$$H(z)=\frac{1+3z^{-1}+3z^{-2}+z^{-3}}{1-\frac{3}{2}z^{-1}+\frac{11}{16}z^{-2}-\frac{3}{32}z^{-3}}$$

10. 对第9题的 $H(z)$ 应用离散预测变换求2级可流水传输函数。对于流水线传输函数，推导基本、单乘法器、归一化和可缩放归一化格型滤波器结构。
11. 对于第9题的 $H(z)$ 和第10题的流水线传输函数，推导逆可缩放归一化格型滤波器结构。
12. 对于第9题的可缩放归一化格型滤波器和第11题的逆可缩放归一化格型滤波器，分别计算信噪比。比较两个信噪比。
13. 求第3题的传输函数的逆归一化格型滤波器实现。
14. 对如下的2段可流水传输函数:

$$H(z)=\frac{1}{1-2r^2\cos 2\theta z^{-2}+r^4z^{-4}} \tag{12-98}$$

(a) 推导可缩放归一化格型滤波器结构;

(b) 计算输出信号功率;

(c) 计算信噪比;

(d) 将(c)求得的信噪比作为 r 的函数做图。

15. 设计一个极点在 $z = -0.9$ 和 $z = 0.9e^{\pm j\pi/4}$ 的3阶全通传输函数。应用归一化格型滤波器结构实现该全通传输函数。这里实现的归一化格型滤波器应该不含有逐轮获取参数。

16. 一个实用传输函数家族，包括巴特沃斯、切雪比夫和椭圆滤波器，可表示为:

$$H(z) = \frac{1}{2}\{A_0(z) + A_1(z)\}$$

472

其中 $A_0(z)$ 和 $A_1(z)$ 为稳定的单位量级全通滤波器。对于一个奇数阶巴特沃斯、切雪比夫和椭圆滤波器，全通的分解可用如下极点交错性质有效地完成[22]。极点交错性质：令 $H(z)$ 的极点为 z_0，z_1，…，z_{N-1} 且极角为 θ_0，θ_1，…，θ_{N-1}，其中 $\theta_i < \theta_{i+1}$。然后，$A_0(z)$ 的极点由 z_{2k} 给出，而 $A_1(z)$ 的极点由 z_{2k+1} 给出。根据这些极点，$A_0(z)$ 和 $A_1(z)$ 即可求出。

考虑一个3阶椭圆低通滤波器:

$$H(z) = \frac{0.3296(1 + 1.61165z^{-1} + 1.61165z^{-2} + z^{-3})}{1 + 0.0522z^{-1} + 0.7139z^{-2} - 0.0445z^{-3}}$$

(a) 用极点交错性质将 $H(z)$ 分解为2个全通函数;

(b) 用单乘法器格型滤波器实现分解的传输函数。这样的实现需要多少乘法器和加法器?

17. 考虑图12-33a所示的窄波段低通规格。因为一个典型的窄波段滤波器的极点分布靠近单位圆，这个滤波器会有大的舍入噪声，从而实现时需要大的字长。所以，这里不用图12-33a的规格，而设计一个双重展宽IIR滤波器 $G_2(z)$ 来代替。图12-33b给出了 $G_2(z^2)$ 的数值响应，其中 π 附近的导通频带是多余的，如图12-33c所示可用一个FIR滤波器 $I_1(z)$ 和 $G_2(z^2)$ 级联来屏蔽。$I_1(z)$ 由于它的宽传输带宽很容易进行设计，有效响应由 $G_2(z^2)$ 和 $I_1(z)$ 级联得到。注意这里实现的滤波器 $G_2(z^2)$ $I_1(z)$ 是2段可流水的，因为FIR滤波器 $I_1(z)$ 可以任意级别进行流水。假设图12-33a的低通滤波器规格如下: $\omega_p = 0.0625\pi$, $\omega_s = 0.075\pi$，导通纹波 = 0.5 dB 且阻断衰减 = 50 dB。本题中，所有IIR滤波器用椭圆滤波器设计方法设计且用单乘法器格型滤波器结构实现; FIR滤波器则用McClellan-Parks算法设计且用直接式结构实现。

(a) 设计一个满足图12-33a的IIR滤波器。画出滤波器的信号流图（SFG）并计算信噪比;

(b) 设计双重展宽IIR滤波器 $G_2(z)$。设计一个满足图12-33c规格的FIR滤波器。画出数值响应和级联系统 $G_2(z^2)$ $I_1(z)$ 的数据流图。计算信噪比并和(a)中的信噪比比较。

(c) 设计一个4重展宽滤波器 $G_4(z)$。$G_4(z^4)$ 的数值响应可见12-33d。假设多余的导通频带用2个FIR滤波器 $I_1(z)$ $I_2(z^2)$ 的级联屏蔽，其中 $I_1(z)$ 规格如图12-33c所示。画出级联系统 $G_4(z^4)$ $I_1(z)$ $I_2(z^2)$ 的数值响应。计算级联系统的信噪比并和(a)及(b)的信噪比做比较。 473

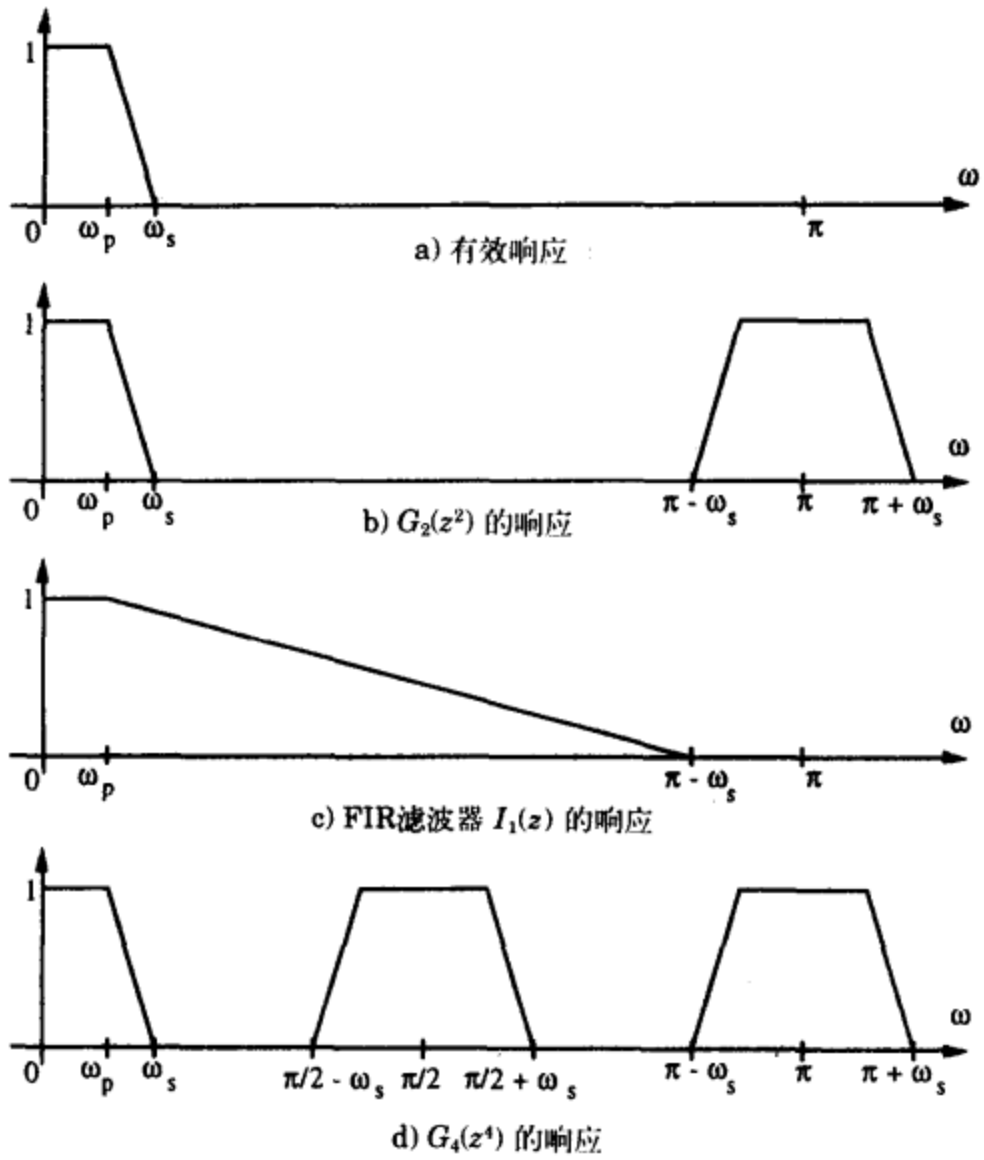


图 12-33

参考文献

1. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing.* Prentice Hall, 1975.

2. J. D. Markel and A. H. Gray, Jr., "On autocorrelation equations as applied to speech analysis," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. AU-21, pp. 69–79, April 1973.

3. J. Schur, "Ueber Potenzreihen die im Innern des Einheitslreises beschrankt sind," in *J. Reine Angewandte Mathematik*, vol. 147, pp. 205–232, (Berlin, Germany), 1917.

474

4. R. A. Roberts and C. T. Mullis, *Digital Signal Processing.* Addison-Wesley, 1987.

5. S. M. Lei and K. Yao, "A class of systolizable IIR digital filters and its design for proper scaling and minimum output roundoff noise," *IEEE Trans. on Circuits and Systems*, pp. 1217–1230, Oct. 1990.

6. A. H. Gray, Jr., and J. D. Markel, "Digital lattice and ladder filter synthesis," *IEEE Trans. on Audio and Electroacoustics*, vol. AU-21, pp. 491–500, Dec. 1973.

7. J. G. Chung and K. K. Parhi, "Scaled normalized lattice digital filter structures," *IEEE Transactions on Circuits and Systems, Part II: Analog and Digital Signal Processing*, pp. 278–282, April 1995.

8. A. H. Gray, Jr., and J. D. Markel, "A normalized digital filter structure," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-23, pp. 268–277, June 1975.

9. J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. on Electronic Computers*, vol. EC-8, pp. 330–334, Sept. 1959.

10. P. Dewilde, E. Deprettere, and R. Nouta, "Parallel and pipelined VLSI implementation of signal processing algorithms," in *VLSI and Modern Signal Processing* (S. Kung, H. Whitehouse, and T. Kailath, eds.), Prentice Hall, 1984.

11. J. D. Markel and A. H. Gray, Jr., "Roundoff noise characteristics of a class of orthogonal polynomial structures," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-23, pp. 473–486, Oct. 1975.

12. S. J. Mason and H. J. Zimmermann, *Electronic Circuits, Signals and Systems*. Wiley, 1960.

13. H. Kimura and T. Osada, "Canonical pipelining of lattice filters," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, pp. 878–887, June 1987.

14. J. G. Chung and K. K. Parhi, "Pipelining of lattice IIR digital filters," *IEEE Trans. on Signal Processing*, vol. 42, pp. 751–761, April 1994.

15. J. G. Chung and K. K. Parhi, *Pipelined Lattice and Wave Digital Recursive Filters*. Kluwer, 1996.

16. A. G. Deczky, "Synthesis of recursive digital filters using the minimum p-error criterion," *IEEE Trans. Audio Electroacoustics*, vol. AU-20, pp. 257–263, Oct. 1972.

17. A. G. Deczky, "Program for minimum-p synthesis of recursive digital filters," in *Programs for Digital Signal Processing*, IEEE Press, 1979.

18. R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Comput. Journal*, vol. 6, pp. 163–168, 1963.

19. M. A. Richards, "Application of Deczky's program for recursive filter design to the design of recursive decimators," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, pp. 811–814, Oct. 1982.

20. H. G. Martinez and T. W. Parks, "Design of recursive digital filters with optimum magnitude and attenuation poles on the unit circle," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-26, pp. 150–157, April 1978. 475

21. H. G. Martinez and T. W. Parks, "A class of infinite-duration impulse response digital filters for sampling rate reduction," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-27, pp. 154–162, April 1979.

22. L. Gazsi, "Explicit formulas for lattice wave digital filters," *IEEE Trans. on Circuits and Systems*, vol. CAS-32, pp. 68–88, Jan. 1985. 476

第13章 位级运算架构

13.1 引言

本章给出了在DSP算法中经常碰到的加法和乘法的位级架构设计，涉及到位并行、位串行和数位串行三种实现类型。*位并行*（bit-parallel）系统每个时钟周期处理输入样点的一个字，适合于高速应用。*位串行*（bit-serial）系统每个时钟周期处理输入样点的一位。这些系统可以用基于整数线性规划的调度方法来进行综合（见附录7）。位串行系统节省面积，适合于低速应用[1]~[3]。位串行运算适用于中等复杂度和低等到中等数据速率的数据流算法实现，而位并行操作则可以用于低复杂度、高数据速率的数据流算法实现。*数位串行*（digit-serial）系统[4]，[5]每个时钟周期处理多位（称为*数位大小*，digit-size），最适合于需要中等采样速率、而面积和功耗要求苛刻的应用。

本章讨论位并行和位串行乘法器的设计、位串行数字滤波器的设计和实现，以及关于矢量乘矢量的位级实现方案。其中，主要强调的是在位级水平将算法映射到运算架构时基于设计方法学的架构设计。我们假定读者已具备了计算机运算的基本知识，比如数的表示、相加和相乘算法等这些在许多VLSI设计和计算机运算方面的书中均有介绍的内容[6]~[8]。本章中，假设所有的数都采用定点二的补码表示法来表示。一个 W 位的数 A 表示为：

$$A = a_{W-1}.a_{W-2}\cdots a_1 a_0 \tag{13-1}$$

其中位 a_i 为0或1，$0 \le i \le W-1$，并且最高位（msb）a_{W-1} 为*符号位*（sign bit），为0时表示正数，为1时表示负数。该数的值在区间 $[-1, 1-2^{-W+1}]$ 中，并由下式给出：

$$A = -a_{W-1} + \sum_{i=1}^{W-1} a_{W-1-i}2^{-i} \tag{13-2}$$

二的补码运算的主要优点是可以不考虑中间运算溢出而能够得到正确的最终结果。因此，在一个加法或减法运算链中，只要知道了最终结果在区间 $[-(1-2^{-W+1}), 1-2^{-W+1}]$ 中，即使中间运算导致溢出，最终结果依然正确。对于位串行实现，我们考虑固定字长的乘法器，其中所有的信号均为常数 W 位。虽然 $W \times W$ 位的乘法会得到 $(2W-1)$ 位的积，但我们只保留最高的 W 位。

本章组织如下：第13.2节讨论并行乘法器的设计，包括串行进位阵列乘法、进位保留阵列乘法、Baugh-Wooley乘法和Booth重编码乘法。两种高效的数字滤波器布局策略，*交错布局规则*（interleaved floor-plan）和*位平面*（bit-plane）在第13.3节中讨论。利用脉动映射和Horner法则的位串行乘法器的设计在第13.4节中给出。第13.5节给出了具有常系数的位级流水线位串行FIR和IIR滤波器的设计方法学。该小节还讨论了与位串行实现中的延迟控制和同步相关的问题。第13.6节给出了利用正规符号数（CSD）表示法的低成本、高速度常数乘法器的设计。第13.7节给出了基于分布式运算方法的矢量-矢量乘法方案。

13.2 并行乘法器

本节考虑在时间 $O(W)$ 内利用规则结构能够完成二的补码乘法运算和能够完成符号扩展乘法（推导自Horner法则）与Baugh-Wooley乘法的乘法器，介绍了两种规则性的实现类型，即串行进位阵列和进位保留阵列。

478 可以证明乘法的时间复杂度不可能小于 $O(\log_2 W)$。二叉树（binary-tree）和Wallace树（Wallace-tree）乘法器可以达到这个极值（见附录E）[10]，但它们对应的结构极不规则。

令被乘数和乘数分别为 A 和 B:

$$A = a_{W-1}.a_{w-2}\cdots a_1a_0 = -a_{W-1} + \sum_{i=1}^{W-1} a_{W-1-i}2^{-i}$$

$$B = b_{W-1}.b_{w-2}\cdots b_1b_0 = -b_{W-1} + \sum_{i=1}^{W-1} b_{W-1-i}2^{-i}$$

其中 $A > -1, B < 1$，并且 A 和 B 不同时为-1。它们的积 $P = A \times B$ 由下式给定:

$$P = -p_{2W-2} + \sum_{i=1}^{2W-2} p_{2W-2-i}2^{-i} \tag{13-3}$$

其中小数点在最高位 p_{2W-2} 的右边。既然两个在区间 $[-1, 1)$ 中数的积仍然在该区间中，结果中 p_{2W-2} 以左的更高位可以忽略。

在固定字长乘法中，舍去积 P 的低 $W-1$ 位，则积可以表示为 $X \Leftarrow P = A \times B$，其中

$$X = -x_{W-1} + \sum_{i=1}^{W-1} x_{W-1-i}2^{-i} \tag{13-4}$$

积 X 在考虑固定字长乘法时使用，而积 P 在考虑全精度积时使用。

13.2.1 具有符号扩展的并行乘法

利用Horner法则，A 和 B 相乘可以写为

$$\begin{aligned} P &= A \times (-b_{W-1} + \sum_{i=1}^{W-1} b_{W-1-i}2^{-i}) \\ &= -A \cdot b_{W-1} + [A \cdot b_{W-2} + [A \cdot b_{W-3} + [\cdots + [A \cdot b_1 \\ &\quad + A \cdot b_0 2^{-1}]2^{-1}]\cdots]2^{-1}]2^{-1} \end{aligned} \tag{13-5}$$

479 其中 2^{-1} 表示缩放（scaling）运算。在二的补码系统中，求一个数的相反数等价于先取它关于一的补，再将结果的最低位（lsb）加一，具体如下:

$$\begin{aligned} -A &= a_{W-1} - \sum_{i=1}^{W-1} a_{W-1-i}2^{-i} \\ &= a_{W-1} + \sum_{i=1}^{W-1} (1 - a_{W-1-i})2^{-i} - \sum_{i=1}^{W-1} 2^{-i} \end{aligned} \tag{13-6}$$

$$
\begin{aligned}
&= a_{W-1} + \sum_{i=1}^{W-1}(1-a_{W-1-i})2^{-i} - 1 + 2^{-W+1} \\
&= -(1-a_{W-1}) + \sum_{i=1}^{W-1}(1-a_{W-1-i})2^{-i} + 2^{-W+1} \\
&= -\bar{a}_{W-1} + \sum_{i=1}^{W-1}\bar{a}_{W-1-i}2^{-i} + 2^{-W+1}
\end{aligned}
$$

其中 $(1-a_i)$ 是 a_i 关于一的补码，记作 $\bar{a}_i$，$0 \le i \le W-1$。因此，在4位乘法中项 $-A.\ b_{W-1}$ 可以写为

$$
-A \times b_3 = -\bar{a}_3 b_3 + \bar{a}_2 b_3 2^{-1} + \bar{a}_1 b_3 2^{-2} + \bar{a}_0 b_3 2^{-3} + b_3 2^{-3}
$$

利用图13-1所示的表可以完成乘法运算。在图13-1中，由于其中的量具有负的权重，所以加法不能直接完成。*符号扩展*（sign extension）可以用来解决这个问题。例如，

$$
\begin{aligned}
A &= -a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3} \\
&= -a_3 2 + a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3} \\
&= -a_3 2^2 + a_3 2 + a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}
\end{aligned}
$$

描述了将 A 的一位和两位符号扩展，而它的值不变。在二的补码表示法中，一般利用具有符号扩展的右移位来完成缩放运算 2^{-i}。例如，将 A 缩放 2^{-1} 倍可以按如下方式完成:

$$
\begin{aligned}
A \cdot 2^{-1} &= (-a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3})2^{-1} \\
&= -a_3 2^{-1} + a_2 2^{-2} + a_1 2^{-3} + a_0 2^{-4} \\
&= -a_3 + a_3 2^{-1} + a_2 2^{-2} + a_1 2^{-3} + a_0 2^{-4} \\
&\Rightarrow -a_3 + a_3 2^{-1} + a_2 2^{-2} + a_1 2^{-3},
\end{aligned}
\tag{13-7}
$$

其中所有位均右移一位，被乘数 A 被扩展的符号位 a_3 被保留在最高位而 a_0 被舍去。缩放后的数仍用4位字长表示。

			a_3	a_2	a_1	a_0
			b_3	b_2	b_1	b_0
			$-a_3b_0$	a_2b_0	a_1b_0	a_0b_0
		$-a_3b_1$	a_2b_1	a_1b_1	a_0b_1	
	$-a_3b_2$	a_2b_2	a_1b_2	a_0b_2		
$-\bar{a}_3b_3$	$\bar{a}_2b_3$	$\bar{a}_1b_3$	$\bar{a}_0b_3$			
			b_3			
p_6	p_5	p_4	p_3	p_2	p_1	p_0

图13-1 位级阵列乘法的图表形式

现在我们可以在位级水平描述具有符号扩展的乘法。4 × 4位的乘法由图13-2中的表格形式给出，其中 pp_j^i 为部分积 PP^i 的第 j 位。部分积右移后，舍去方框中的最低位以保持固定的字长。因为部分积是逐步累加的，每一步中符号扩展一位就可以对齐部分积的符号位。

然而，每一步符号扩展一位并不能消除溢出（overflow）。为了得到正确的积，将 W 位被乘数的最高位 a_{W-1} 定义为被乘数的*警戒*（guard）位，它的值与 a_{W-2} 相等。换句话说，这

些乘法器的被乘数必须在 [-1/2, 1/2) 中。否则，也可以使用每一步符号扩展两位的方法以得到正确的积。

			$a_3.$	a_2	a_1	a_0
			$b_3.$	b_2	b_1	b_0
		a_3b_0	$\leftarrow a_3b_0$	a_2b_0	a_1b_0	$\boxed{a_0b_0}$
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	
	pp_3^1	$\leftarrow pp_3^1$	pp_2^1	pp_1^1	$\boxed{pp_0^1}$	
	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
pp_3^2	$\leftarrow pp_3^2$	pp_2^2	pp_1^2	$\boxed{pp_0^2}$		
$\bar{a}_3b_3$	$\bar{a}_2b_3$	$\bar{a}_1b_3$	$\bar{a}_0b_3$			
			b_3			
$x_3.$	x_2	x_1	x_0			

图13-2 位级二的补码阵列乘法的图表形式

13.2.1.1 并行串行进位阵列乘法器

480~481

在串行进位阵列乘法器中，一个加法器的进位输出被传到同一行中其左边的加法器。这种乘法运算的位级依赖图（DG）由图13-3给出。

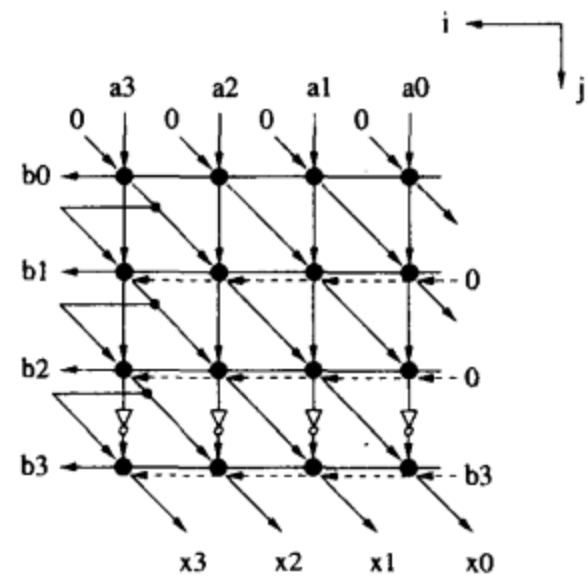


图13-3 4×4位串行进位阵列乘法的依赖图

由图13-3的依赖图可以直接推导出并行阵列乘法器，如图13-4所示。在图13-4所示的割集处放置延迟元件形成流水线，可以将乘法器的关键路径从2W减小到W。利用二维流水线可以获得位级流水线（图13-4中没有示出）。

13.2.1.2 并行进位保留阵列乘法器

在串行进位乘法器中，每次加法均需要传递进位。进位传递限制了乘法的速度。在涉及多次加法的情况下，比如乘法中的部分积累加，并不严格地需要在每个周期中传递进位。相反地，可以先保留（save）两个操作数相加产生的进位，经过适当对齐后再与下一个操作数相加。这就引出了进位保留（carry-save）加法的概念。在进位保留阵列乘法器中，进位输出被保留到下一行使用。用部分和（partial sum）和部分进位（partial carry）代替部分积，它们被保留并传递到下一行。进位保留加法的优点是在同一行不同位上的加法相互独立从而可以并行完成，这从根本上提高了每个周期的加法速度，从而提高了乘法速度。除了最后一步以外，其他加法均可以使用进位保留加法，因为最后一步中只将部分和和部分进位相加，而再不用加上被乘数的倍数。部分和和部分进位的相加用矢量合并加法器（vector merging adder）（VMA）完成。位级进位保留乘法的依赖图由图13-5示出。根据该依赖图可以设计出并行进位保留阵列乘法器，如图13-6所示。图中虚线所示的割集处放置延迟元件可以形成流水线。

482

VMA可以用如图13-7a所示的4×4进位保留阵列实现，该实现只包含半加器，结构更规则，更易于形成流水线，但是需要更多的半加器；或者用如图13-7b所示的4位串行进位加法

器。如果不需要位级流水线，可以将串行进位和进位保留原理综合起来，得到如图13-7c所示的结构，其中在同一行中的进位输出至多传递一位，并且与进位保留结构类似，每个串行进位加法器产生的进位被传递到下一行。这种组合的矢量合并结构所需的加法器数目介于进位保留和串行进位矢量合并结构之间。

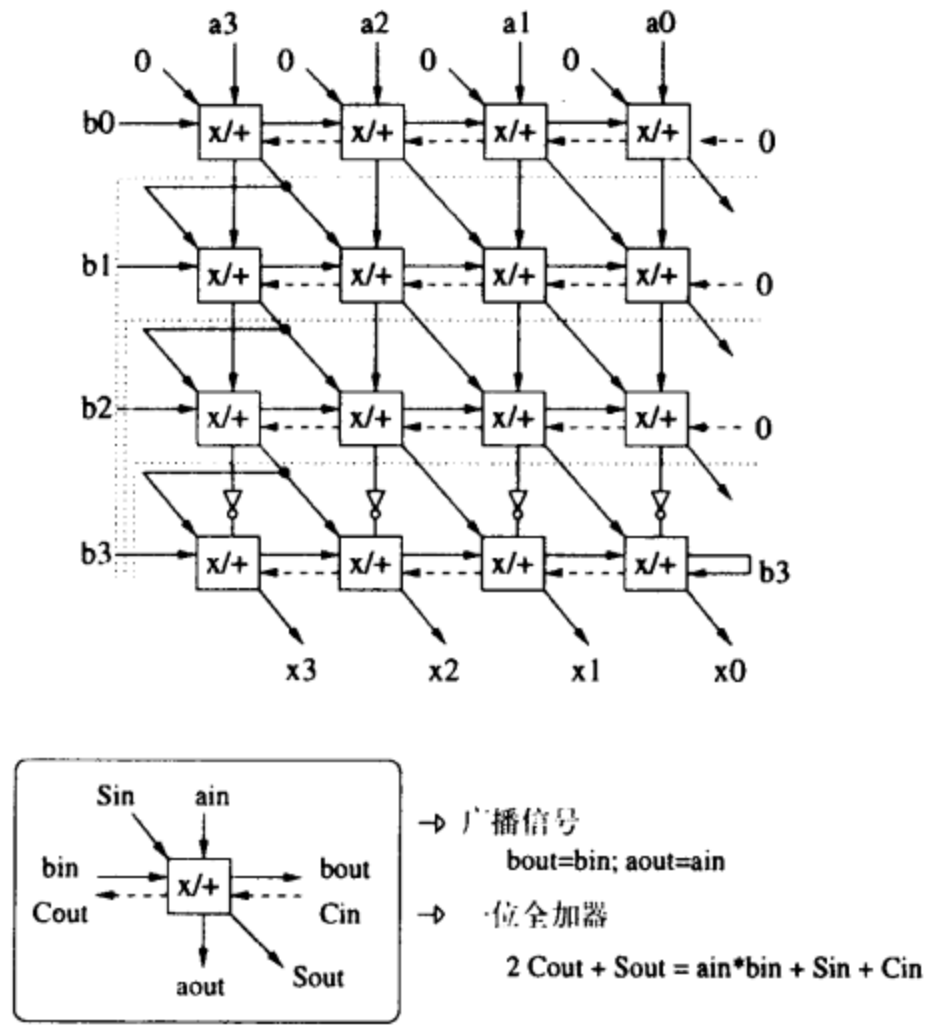


图13-4 并行串行进位阵列乘法器

483

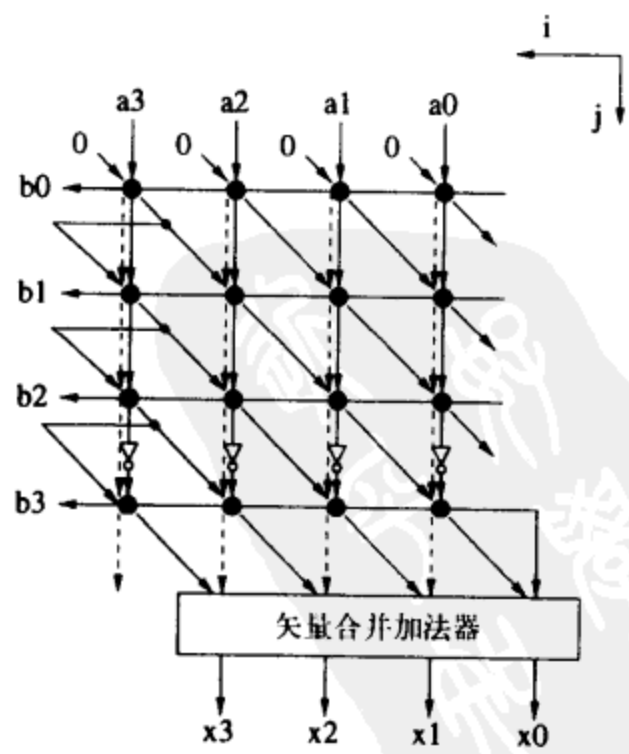


图13-5 4×4位进位保留阵列乘法的依赖图

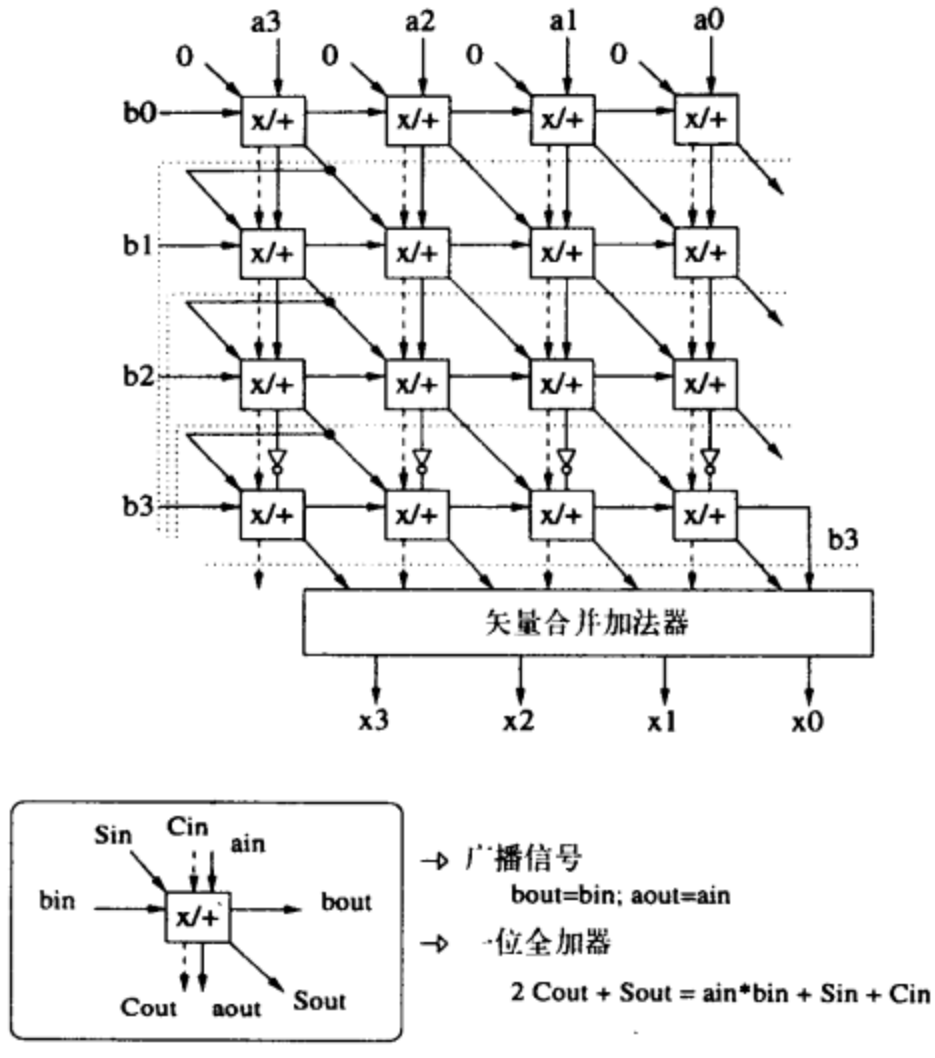
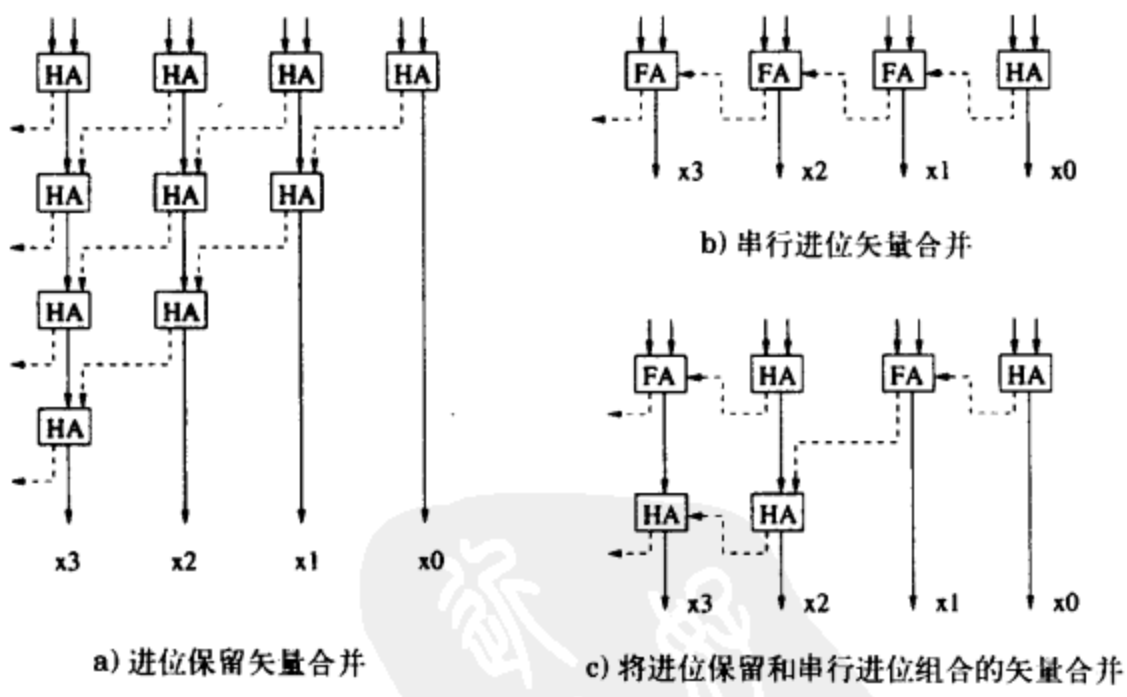


图13-6 并行进位保留阵列乘法器



a) 进位保留矢量合并

b) 串行进位矢量合并

c) 将进位保留和串行进位组合的矢量合并

图13-7 4×4位并行进位保留乘法的矢量合并

13.2.2 Baugh-Wooley乘法器

二的补码乘法的难点在于对被乘数和乘数的符号的处理。Baugh-Wooley乘法算法提供了一种解决这个问题的有效方法[9]。图13-8所示的乘法表以4×4为例，图解了该乘法算法，其中所有的位均具有正的权重。该算法的推导留作本章后面的习题（见习题4）。

			a_3	a_2	a_1	a_0
			b_3	b_2	b_1	b_0
			$\overline{a_3b_0}$	a_2b_0	a_1b_0	a_0b_0
		$\overline{a_3b_1}$	a_2b_1	a_1b_1	a_0b_1	
	$\overline{a_3b_2}$	a_2b_2	a_1b_2	a_0b_2		
a_3b_3	$\overline{a_2b_3}$	$\overline{a_1b_3}$	$\overline{a_0b_3}$			
		1				
x_3	x_2	x_1	x_0			

图13-8 位级Baugh-Wooley乘法的图表形式

图13-8中的Baugh-Wooley乘法表可以用串行进位阵列或者进位保留阵列实现。进位保留Baugh-Wooley乘法器由图13-9示出，其中 $P = p_6 \cdot p_5\, p_4\, p_3\, p_2\, p_1\, p_0$ 是全精度积。注意两个在区间（-1, 1）中的4位数的积仍然在该区间中。既然小数点在位 p_6 的右边，p_6 以左的更高位可以忽略。截断后的结果表示为 $X = x_3 \cdot x_2\, x_1\, x_0$。

13.2.3 改进的Booth重编码并行乘法器

484 ~ 486

乘法运算涉及两个基本运算：部分积的产生和部分积的累加。因此，有两条提高乘法速度的途径：减少部分积的数目或者提高累加的速度。进位保留加法可以用来提高部分积累加的速度。为了减少部分积的数目，一种比较直接的方法是每次检查乘数的两位或更多位。部分积的数目的减少可以减少乘法运算的等待时间。但是，这种方法需要产生一些倍数 A、$2A$、$3A$等等，其中 A 是被乘数。

用改进的Booth重编码方法可以减少部分积的数目。这个算法基于这样的事实：一组连续的零或一需要产生更少的部分积。对于一组连续的零就不需要产生新的部分积。对于乘数中的 m 个连续的一，即，

$$\begin{aligned}\cdots 0\{11\cdots 1\}0\cdots &= \cdots 1\{00\cdots 0\}0\cdots - \cdots 0\{00\cdots 1\}0\cdots \\ &= \cdots 1\{00\cdots \bar{1}\}0\cdots\end{aligned} \tag{13-8}$$

如果使用有符号数位（signed-digit）表示法，那么只需要产生两个部分积，而不是 m 个部分积。所以，在这种乘法方案中，首先将乘数的各个位重编码（recode）为含有更少的非零数位的有符号数位表示法；然后利用重编码后的乘数数位产生并累加部分积。

在改进的Booth重编码算法中，使用的有符号数位集合为 {-2, -1, 0, 1, 2}，称为5级Booth重编码。在这种重编码算法中，以 b_{2i-1} 为参考位，乘数的位 b_{2i+1} 和 b_{2i} 被重编码为有符号数位 b_i'。例如，一个8位数可以按如下步骤重编码：

$$\begin{array}{ccccccccc} b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & b_{-1} \\ & \underbrace{\qquad\quad} & & \underbrace{\qquad\quad} & & \underbrace{\qquad\quad} & & \underbrace{\qquad\quad} & \\ & b_3' & & b_2' & & b_1' & & b_0' & \end{array} \tag{13-9}$$

其中 $b_{-1} = 0$ 作为参考位添加在乘数的最低位之后。由 b_{2i+1}、b_{2i} 和 b_{2i-1} 产生 b_i' 的改进的Booth重编码算法由表13-1示出。

表13-1 基4改进的Booth重编码算法

b_{2i+1}	b_{2i}	b_{2i-1}	b'_i	操作	说 明
0	0	0	0	+0	一串零
0	0	1	1	+A	一的结束
0	1	0	1	+A	一个一
0	1	1	2	+2A	一的结束
1	0	0	-2	-2A	一的开始
1	0	1	-1	-A	一个零
1	1	0	-1	-A	一的开始
1	1	1	0	-0	一串一

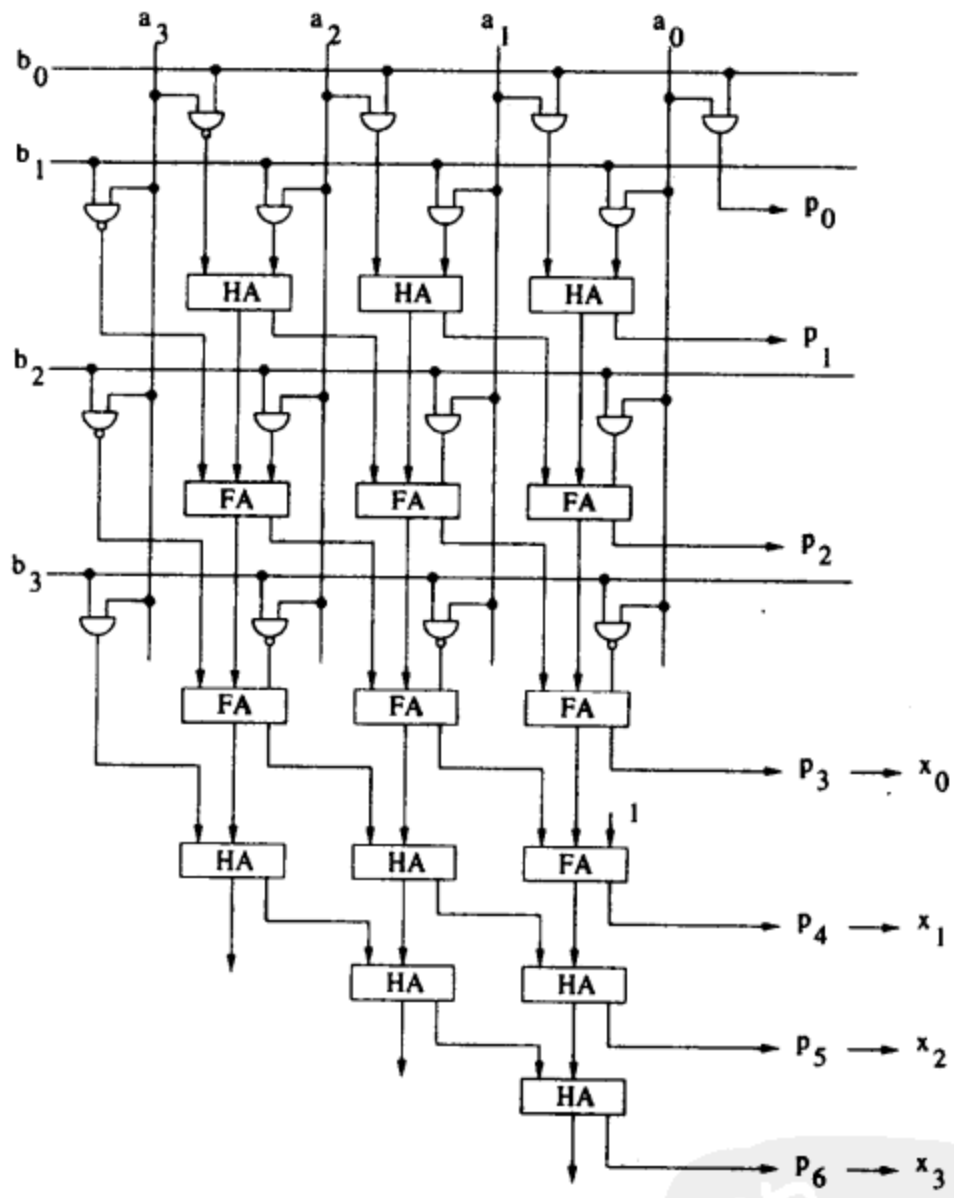


图13-9 4×4位Baugh-Wooley进位保留乘法器

一种描述重编码运算的简单方法是计算

$$b'_i = -2b_{2i+1} + b_{2i} + b_{2i-1} \tag{13-10}$$

读者可以验证

$$-b_7 + \sum_{i=1}^{7} b_{7-i}2^{-i} = \sum_{i=0}^{3} b'_{3-i}2^{-(2i+1)} \tag{13-11}$$

487~488

Booth重编码乘法由三部分组成，对乘数位的重编码（控制）电路、部分积的产生和累加。使用5级改进的Booth重编码的快速乘法器的设计留作本章后面的习题（见习题11）。常规的

Booth重编码电路由于输入到输出的路径不平衡而产生竞争（race）问题，从而导致较高的功耗。[11]中给出了一种低功耗Booth重编码电路，它利用冗余重编码方案减少毛刺和功耗。习题12探讨了低功耗Booth重编码电路的设计。

13.3 交织布局规则与基于位平面的数字滤波器

本节涉及并行FIR数字滤波器的高效实现和布局技术。这些技术同样适用于IIR滤波器的实现。

考虑常系数FIR滤波运算

$$y(n) = x(n) + f \cdot x(n-1) + g \cdot x(n-2) \tag{13-12}$$

其中 $x(n)$ 为输入信号、f 和 g 是滤波系数。假设信号和系数的字长分别为6位和4位，并且均用二的补码表示。

下面给出高速滤波的交织（interleaved）方法[12]。交织方法的主要思路是同时进行与 f 和 g 相关的部分积的计算和累加，从而大大减小布线复杂度。另一个优点是由于只在最后一步才进行截断，从而得到更高的精度。基于Baugh-Wooley算法，利用交织方法计算输出 $y(n)$ 的乘法表由图13-10示出，它是将两个并行的Baugh-Wooley乘法交织而得的。注意由于输出只需要6位的结果，只需考虑 $x(n)$ 的高4位。

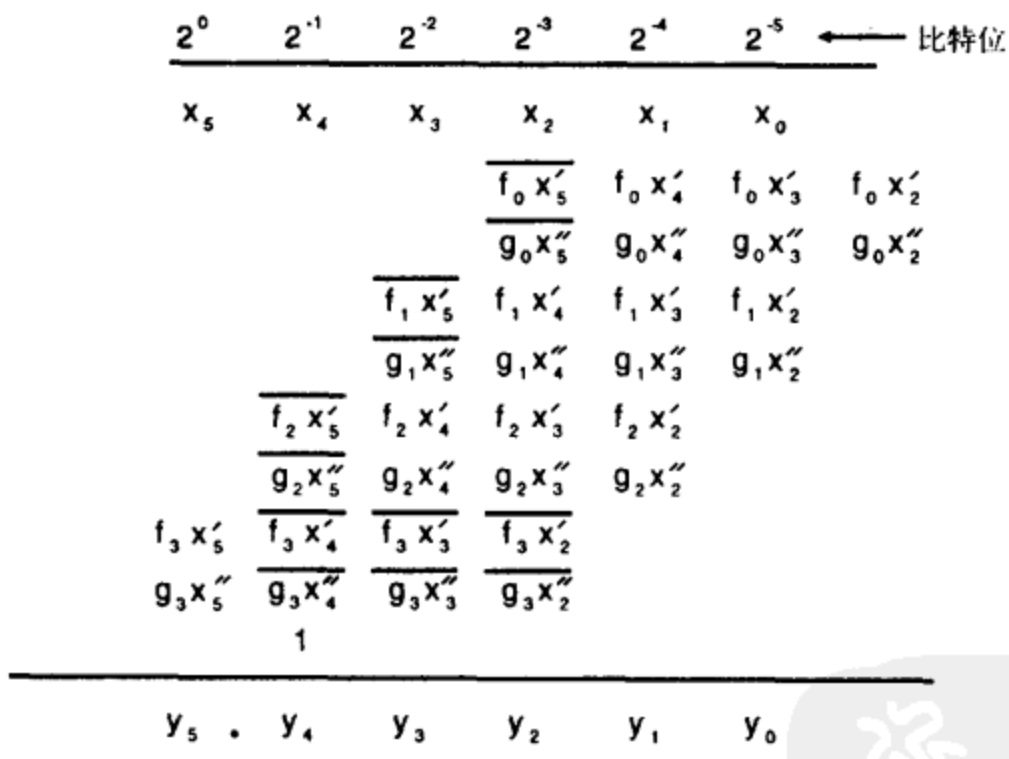


图13-10 用于图解交织方法的乘法表，
其中 x_i 表示 $x_i(n)$，x'_i 表示 $x_i(n-1)$，x''_i 表示 $x_i(n-2)$

最终的交织架构由图13-11示出，其中矢量合并部分是串行进位和进位保留实现的组合，在符合该架构可以形成两位级的流水线的限制下允许串行进位运算（流水线锁存器没有在图13-11中示出）。所以，进位在传递过两个加法器后就被保留并传递到下一行的加法器。

如果交织架构中不同系数的部分积在不同行中计算，所得到的架构称为位平面（bit-plane）架构[13]，由图13-12示出，在阵列中每行加法器后面均加了一行触发器，以得到较高的处理速度。为了允许交织运算，和与进位信号方向交替。如果数字滤波器实现需要 p 位

级流水线，那么位平面架构中的每个位平面可以用一个改进的位平面（modified bit-plane）代替，其中每个位用连续的 p 位代替。设 $p=2$，可以根据图13-13中的乘法表设计改进的位平面架构。

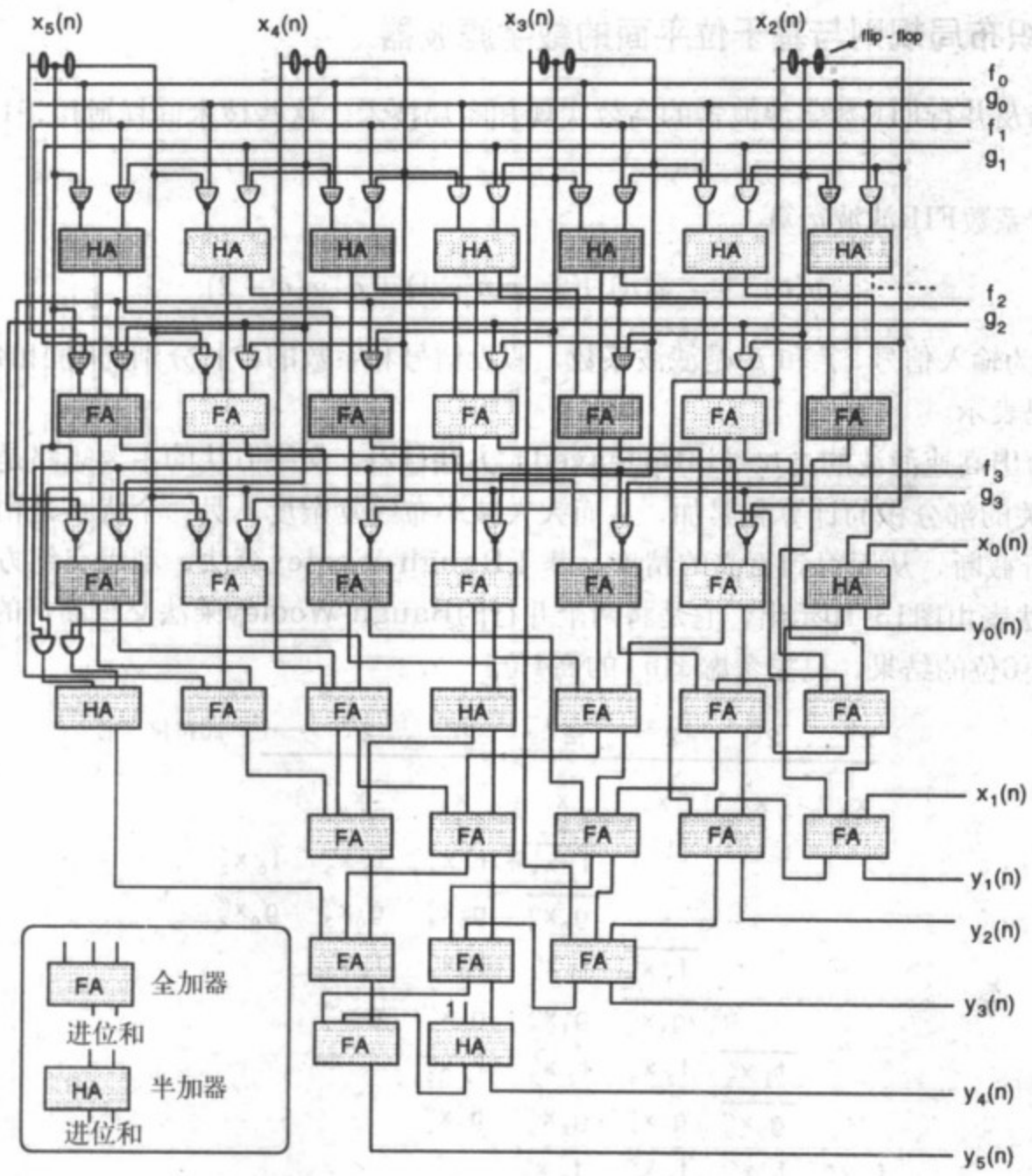


图13-11 图13-10中的乘法表的架构

13.4 位串行乘法器

本节将利用Horner法则推导Lyon位串行乘法器[14]，然后利用脉动映射技术推导其他一些位串行乘法器。

13.4.1 利用Horner法则的Lyon位串行乘法器的设计

公式（13-5）中的乘法法则可以用来推导位串行乘法器。4×4位位串行乘法的架构由图13-14a示出，其中 $\boxed{2^{-1}}$ 是位串行零迟滞缩放算子，它的功能如图13-14b所示。对于位串行零迟滞系统，第一个输出位必须在第一个输入 a_0 进入系统的时钟周期内产生。对于缩放算子，第一个输出位 a_1 必须在第一个输入 a_0 进入算子的时刻产生。由于此时输入 a_1 还没有进入系统，所以缩放算子是非因果或超前运算，不可能用硬件实现。

489 ~ 491

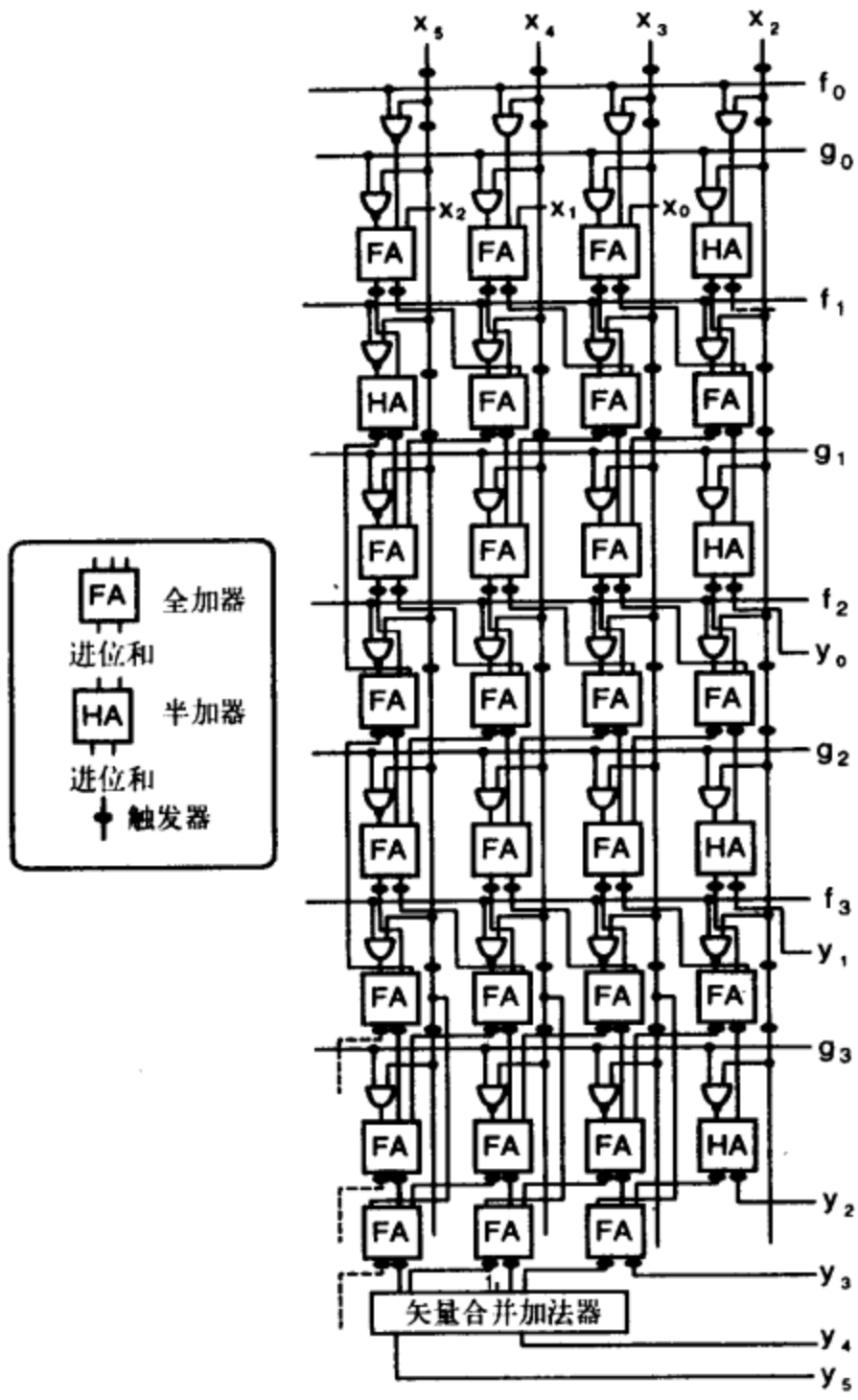
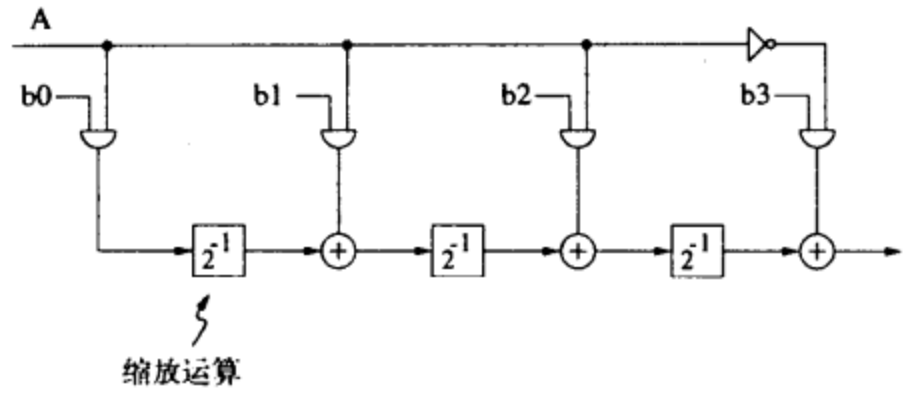


图13 -12 图13-10中的乘法表的位平面架构

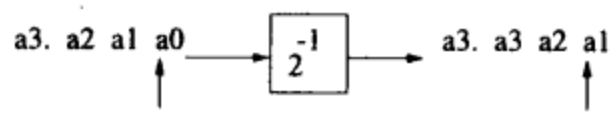
492

2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	← 比特位
x_5	x_4	x_3	x_2	x_1	x_0	
			$\overline{f_0 x'_5}$	$f_0 x'_4$	$f_0 x'_3$	$f_0 x'_2$
		$\overline{f_1 x'_5}$	$f_1 x'_4$	$f_1 x'_3$	$f_1 x'_2$	
			$\overline{g_0 x''_5}$	$g_0 x''_4$	$g_0 x''_3$	$g_0 x''_2$
		$\overline{g_1 x''_5}$	$g_1 x''_4$	$g_1 x''_3$	$g_1 x''_2$	
	$\overline{f_2 x'_5}$	$f_2 x'_4$	$f_2 x'_3$	$f_2 x'_2$		
$f_3 x'_5$	$\overline{f_3 x'_4}$	$\overline{f_3 x'_3}$	$\overline{f_3 x'_2}$			
	$\overline{g_2 x''_5}$	$g_2 x''_4$	$g_2 x''_3$	$g_2 x''_2$		
$g_3 x''_5$	$\overline{g_3 x''_4}$	$\overline{g_3 x''_3}$	$\overline{g_3 x''_2}$			
	1					
y_5 .	y_4	y_3	y_2	y_1	y_0	

图13-13 改进的位平面架构的乘法表



a) 推导自Horner法则的位串行二的补码乘法，由于缩放运算的存在该架构不可实现



b) 缩放算子是零迟滞超前运算

493

图 13-14

可以使用流水线技术在图13-15中的前馈割集处放置延迟元件，实现图13-14a中的架构。一个延迟元件和一个零延迟缩放算子的组合可以用硬件实现。在图13-15中，开关元件S0、S1和S2分别代替了相应位置上的缩放算子和延迟元件组合。通过调度阵列乘法中的运算，可以得到这些开关元件的开关时刻，如图13-16所示，图中每个运算右边方框中的数字为进行该运算的时刻。注意这些开关舍去了部分积的最低位，同时扩展上一个部分积的符号。所以，缩放算子由于重复了最高位，也被称为*位重复器*（bit-repeater）。乘数的各位在时刻0输入到相应的AND门；被乘数的各位从最低位开始串行输入。4位的积的最低位在时刻3输出。位串行二的补码乘法器的完整的示意图由图13-17示出。这种乘法器也被称为Lyon*乘法器*

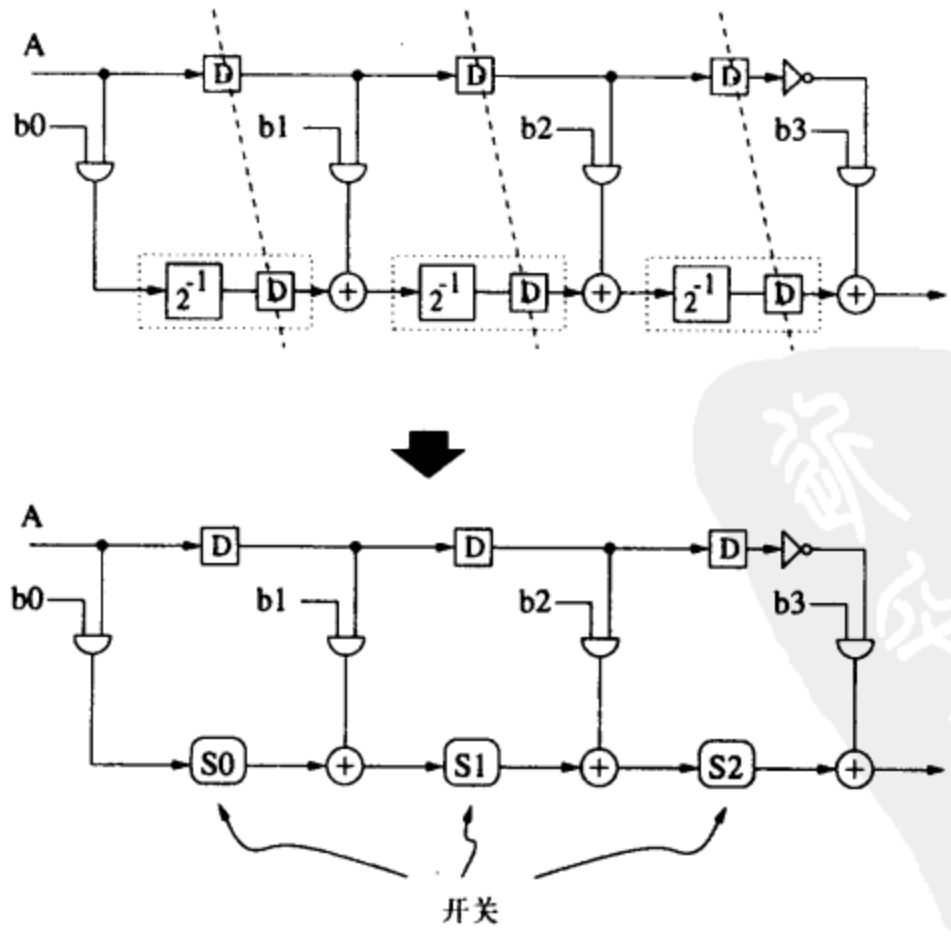


图13-15 可实现的位串行二的补码乘法器的推导

(Lyon's multiplier) [14]。该系统的等待时间为3个时钟周期。如果忽略AND和多路操作的时间，该位串行乘法器的关键路径的计算时间等于 $3T_a$，其中 T_a 为一位加法的时间。一般地，一个 $W \times W$ 位位串行乘法器的等待时间等于 $W-1$ 个时钟周期，且周期时间 $T_{clk} \geqslant (W-1)T_a$。流水线技术可以用来缩短关键路径，从而减小时钟周期。例如，在图13-17中的前馈割集处（用虚线示出）放置延迟元件，那么该4×4位串行乘法器的关键路径就减为一个一位加法，但代价是等待时间的增大。利用结合律和时序变换，而无需引入额外的流水线锁存器，可以将 W 位Lyon乘法器的关键路径缩短为 T_a [15]（见习题18）。

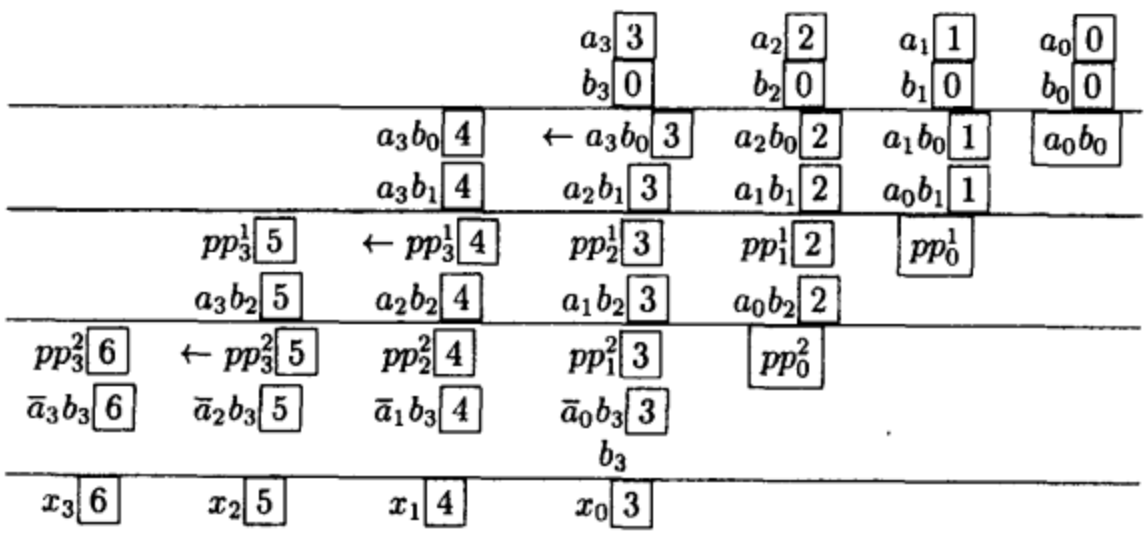


图13-16 位串行二的补码阵列乘法中的运算的调度时刻

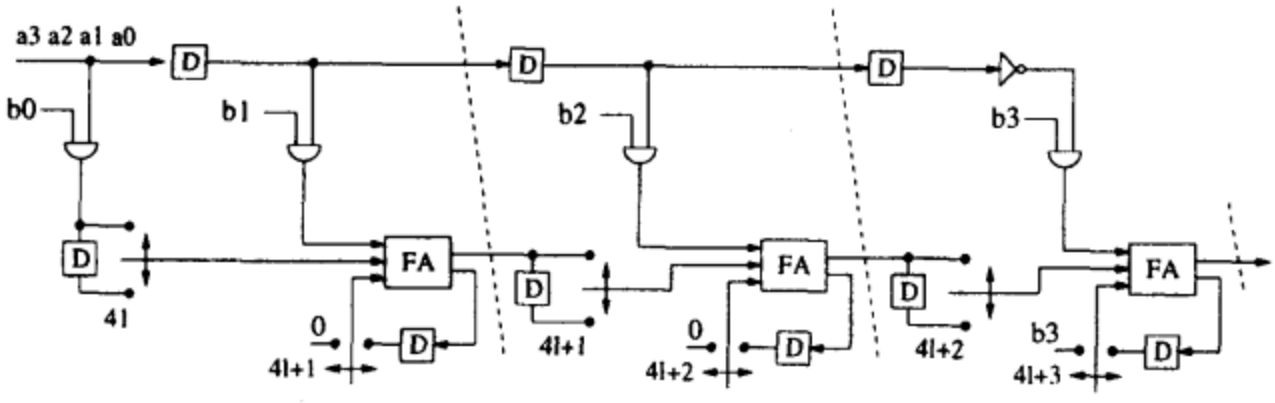


图13-17 Lyon位串行二的补码乘法器

13.4.2 利用脉动映射的位串行乘法器的设计

位串行乘法器可以利用从二维DG到一维阵列的脉动映射方法（见第7章）进行设计。

例13.4.1 本例考虑根据图13-3中的串行进位乘法的DG，利用脉动映射设计Lyon位串行乘法器。使用投影矢量 $\mathbf{d}^T = [1 \quad 0]$，调度矢量 $\mathbf{s}^T = [1 \quad 1]$，处理器空间矢量 $\mathbf{p}^T = [0 \quad 1]$，我们有如下的边映射：

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
a (0, 1)	*1*	*1*
b (1, 0)	*0*	*1*
carry (1, 0)	*0*	*1*
x (−1, 1)	*1*	*0*

利用附加的符号扩展电路可以得到图13-17中的位串行乘法器。同样地，可以根据串行进位乘法的DG，利用脉动映射技术直接设计具有位级流水的Lyon乘法器（见习题10）。

例13.4.2 考虑图13-3中的串行进位阵列乘法的DG。使用投影矢量、处理器空间矢量和调度矢量

$$\mathbf{d}=\begin{pmatrix}0\\1\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}1 & 0\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}0 & 1\end{pmatrix} \tag{13-13}$$

我们有如下的边映射:

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
a (0, 1)	*0*	*1*
b (1, 0)	*1*	*0*
carry (1, 0)	*1*	*0*
x (−1, 1)	−1	*1*

由此得到的位串行串行进位乘法器由图13-18示出，为了清楚起见，省略了图中多路器的开关时刻。 ■

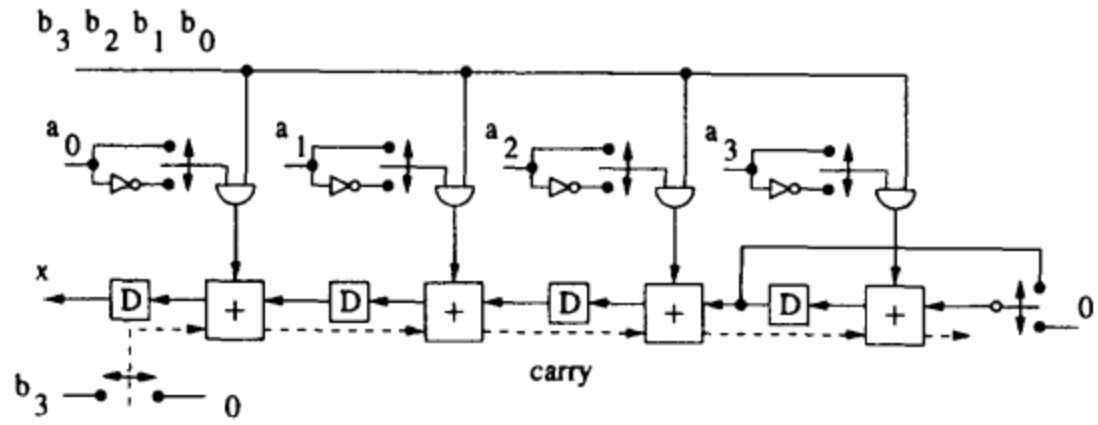


图13-18 例13.4.2中的位串行串行进位乘法器

494 ~ 496

例13.4.3 考虑图13-5中的进位保留阵列乘法的DG。假设矢量合并部分使用串行进位加法器。使用投影矢量、处理器空间矢量和调度矢量

$$\mathbf{d}=\begin{pmatrix}1\\0\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}0 & 1\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}1 & 1\end{pmatrix} \tag{13-14}$$

我们有如下的边映射:

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
a (0, 1)	*1*	*1*
carry (0, 1)	*1*	*1*
b (1, 0)	*0*	*1*
x (−1, 1)	*1*	*0*

由此得到的位串行进位保留乘法器由图13-19示出。为了清楚起见，省略了图中多路器的控制信号。 ■

例13.4.4 考虑图13-8所示的Baugh-Wooley乘法的乘法表。假设使用进位保留加法累加部分积，矢量合并部分使用串行进位加法。该运算的DG由图13-20示出。
使用如下的投影矢量、处理器空间矢量和调度矢量

$$\mathbf{d}=\begin{pmatrix}0\\1\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}1 & 0\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}0 & 1\end{pmatrix} \tag{13-15}$$

进行投影得到如下的边映射：

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
a (0, 1)	0	1
carry (0, 1)	0	1
b (1, 0)	1	0
x (1, 1)	1	1
carry-vm (−1, 0)	−1	0

其中*carry-vm*表示矢量合并部分的进位输出。由此得到的位串行进位保留Baugh-Wooley乘法器由图13-21示出，其中单元 C 在某些时钟周期完成反相运算，以产生正确的部分积，为了简明起见省略了这些控制细节。注意在该设计中，节点 (i, j) 处的计算在时刻 j 执行，$j = 0, 1, 2, 3, 4$。所以，结果位 x_0 在时刻 $5l+3$ 产生，x_3、x_2 和 x_1 在时刻 $5l+4$ 并行产生。一个并行到串行转换器被用来转换这些位，使得这些结果位分别在时刻 $5l$、$5l+1$、$5l+2$、$5l+3$ 和 $5l+4$ 串行输出。在时刻 $5l$，即开始一次新的乘法时，控制信号 $T_0 = 1$，否则 $T_0 = 0$。因此该架构的等待时间等于5个周期，每5个周期能够产生一个输出字。但是，该设计除了本身硬件利用效率低的缺点以外，其主要缺点是关键路径比较长，并且与字长成比例（在图13-21用虚线示出）。

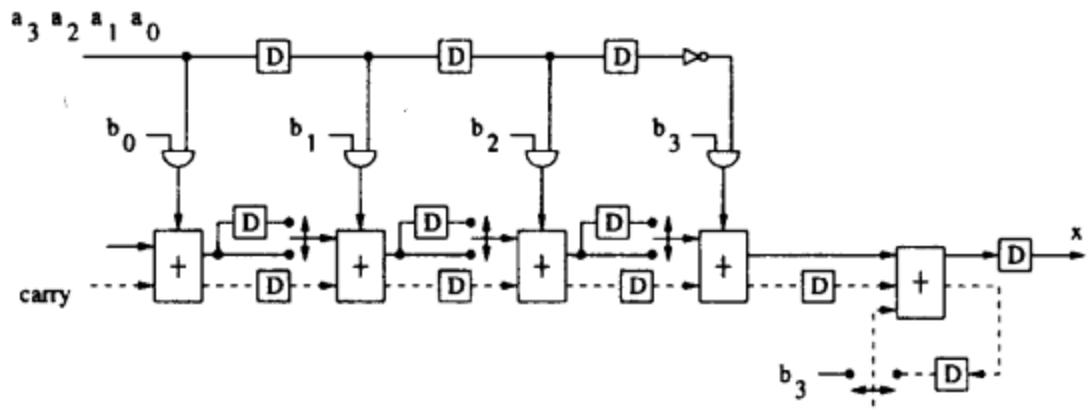


图13-19 例13.4.3中的位串行进位保留乘法器

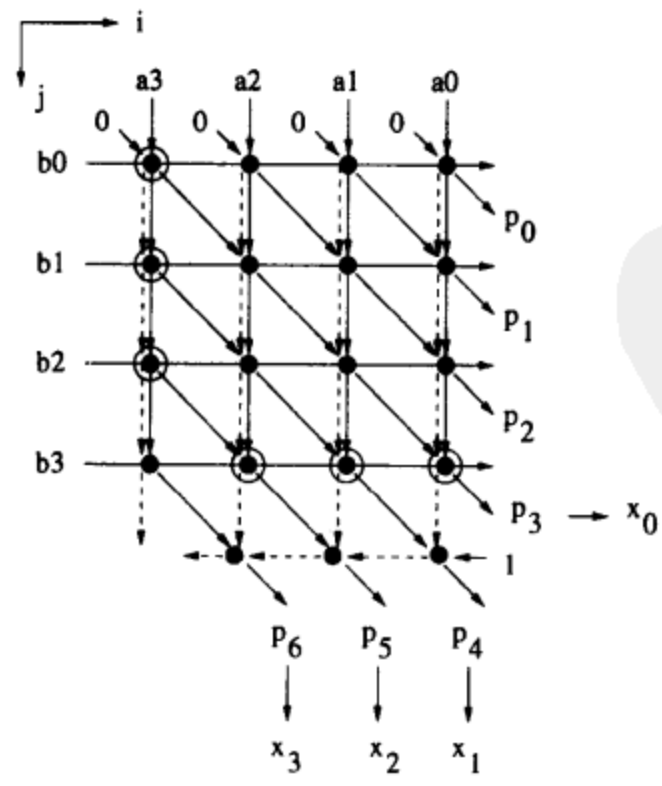


图13-20 具有串行进位矢量合并的进位保留Baugh-Wooley乘法的依赖图

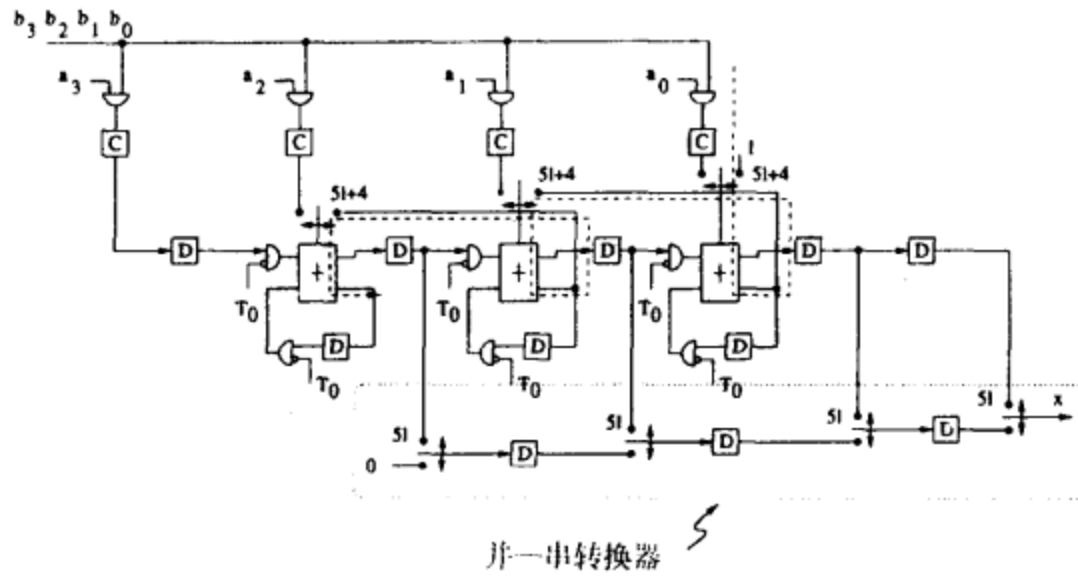


图13-21 例13.4.4中的位串行Baugh-Wooley乘法器1

根据图13-22所示的改进的DG，使用与（13-15）相同的投影矢量、处理器空间矢量和调度矢量，可以得到另一个设计。在该设计中，将进位保留阵列和矢量合并部分作为两个单独的平面进行脉动映射，矢量合并部分中的串行进位加法用一个单独的加法器完成，结果位串行产生。该位串行设计由图13-23示出。它的等待时间等于4个时钟周期，每4个时钟周期能够产生一个输出。注意该设计的关键路径限制为一个二进制加法器的延迟。 ■

13.5 位串行滤波器的设计与实现

FIR和IIR滤波器是DSP中使用两种最基本的结构。本节考虑具有固定的常系数的位串行FIR和IIR数字滤波器的设计，其中与常系数相乘的乘法被分解成位串行移位和加法来实现。

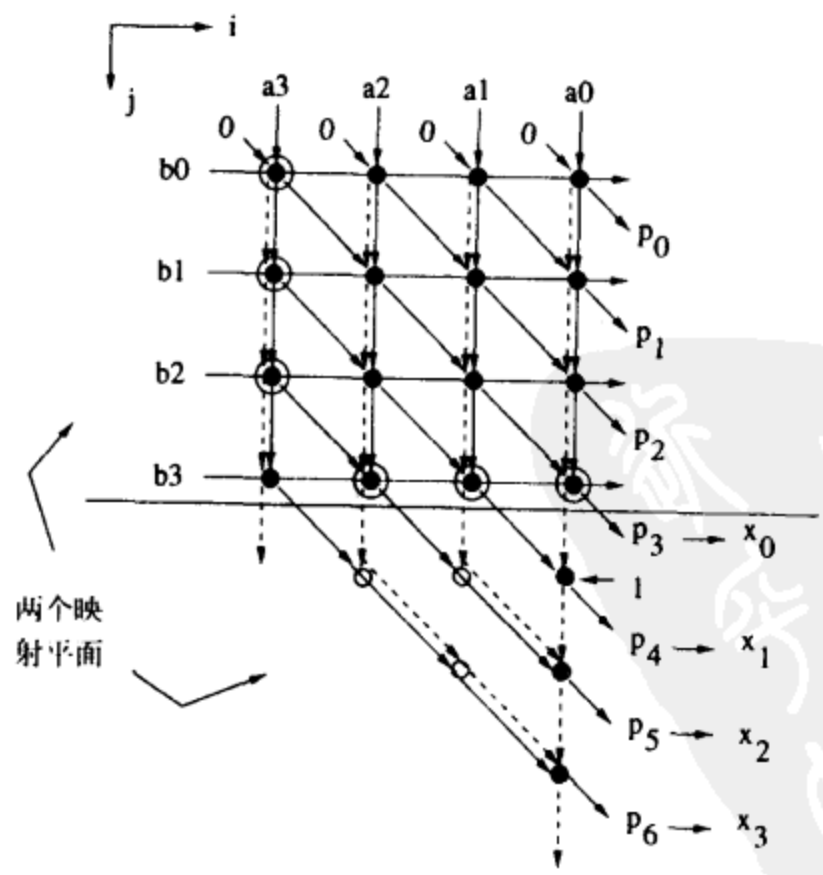


图13-22 例13.4.4中的位串行Baugh-Wooley乘法器DG

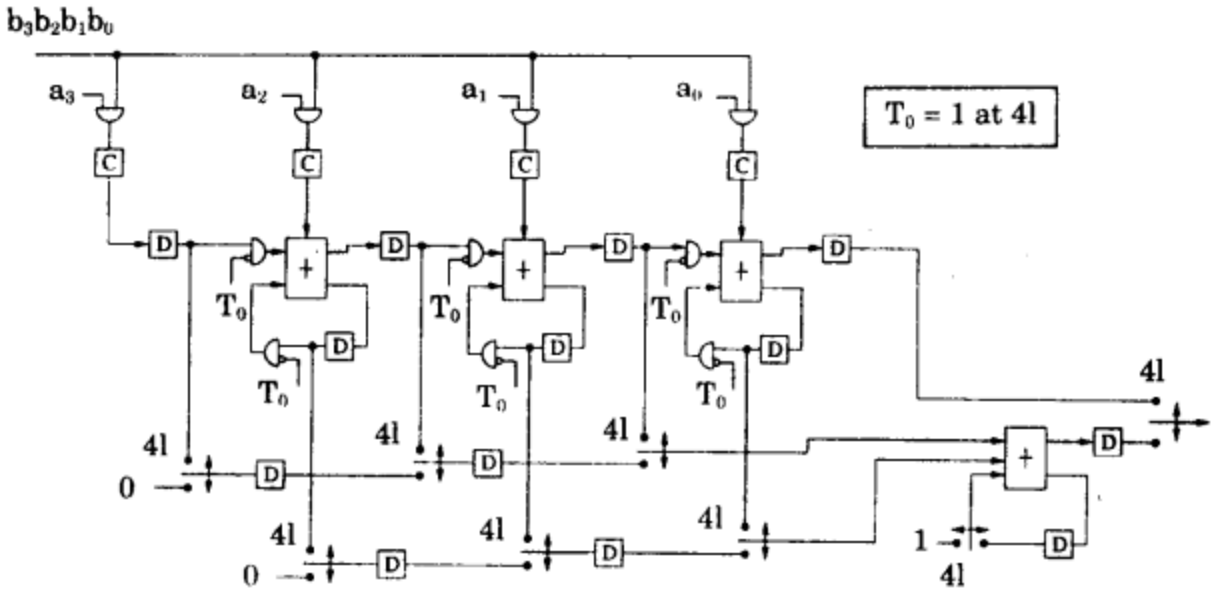


图13-23 例13.4.4中的位串行Baugh-Wooley乘法器2

13.5.1 位串行FIR滤波器

考虑FIR滤波器

$$y(n) = -\frac{7}{8}x(n) + \frac{1}{2}x(n-1) \tag{13-16}$$

的实现，其中信号字长为8。公式（13-16）可以用移位和加法的形式改写成下式:

$$y(n) = -x(n) + x(n)2^{-3} + x(n-1)2^{-1} \tag{13-17}$$

基于移位和加法的FIR滤波器的字级信号流图由图13-24a示出。由于非因果缩放算子的存在，该设计不可实现。图13-24a中的虚线所示的流水线割集可以用来延迟超前的缩放算子。通过在割集处放置延迟元件，用开关代替被延迟的缩放算子，可以推导出可实现的位级流水线位串行FIR滤波器，如图13-24b所示。注意对于字长为 W 的信号，一个字级延迟等价于 W 个位级延迟。所以，图13-24a中的1个字延迟在图13-24b中用8个位延迟代替。开关时刻可以通过调度位级运算得出。 497 ~ 500

13.5.2 位串行IIR滤波器

考虑IIR滤波器

$$y(n) = -\frac{7}{8}y(n-1) + \frac{1}{2}y(n-2) + x(n) \tag{13-18}$$

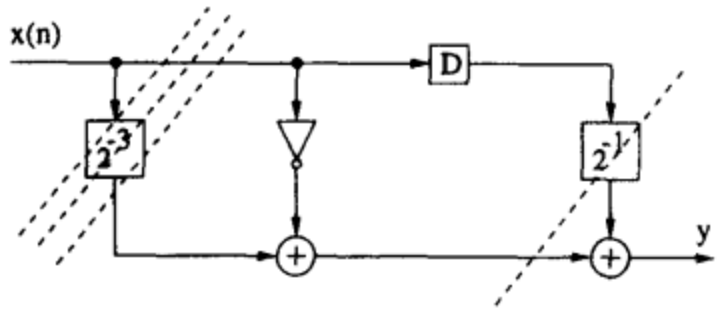
的实现，其中信号字长为8。该IIR滤波器方程可以改写成

$$\begin{aligned} w(n) &= -\frac{7}{8}y(n-1) + \frac{1}{2}y(n-2) \\ y(n) &= w(n) + x(n) \end{aligned}$$

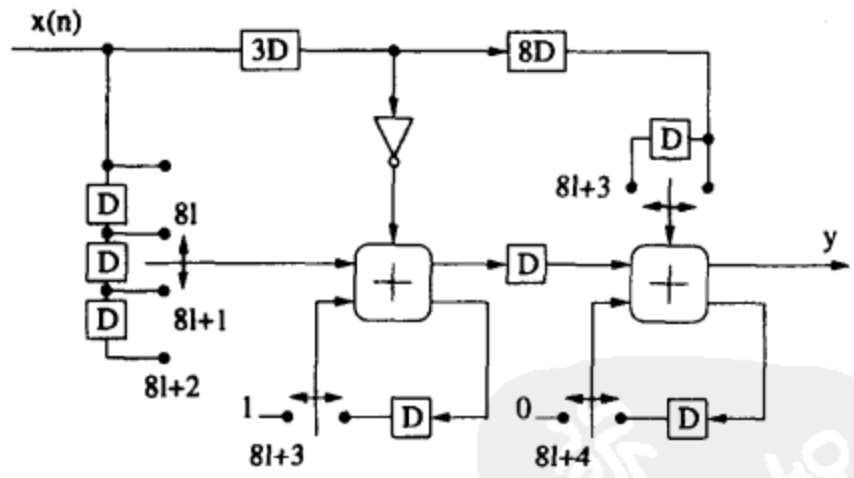
该式可以用一个FIR部分（由 $y(n-1)$ 计算 $w(n)$）和一个加法、一个反馈环路实现，如图13-25所示。位级流水线串行IIR滤波器架构可以根据下述步骤推导得出。

首先需要推导出FIR部分的位级流水线位串行实现。（对于本例，这一步已经在前一小节里完成了。）第二步是中间设计步骤。在这一步中，输入信号 $x(n)$ 与位串行FIR部分的输出 $w(n)$

相加。结果信号 $y(n)$ 与信号 $y(n-1)$ 相连。在这条边上的延迟元件数目待定，在图13-26a中用 ?D 标出。所得的结构由图13-26a示出，含有两个用虚线示出的环路。这样就完成了该位串行系统中的所有互连。但是，对于含有环路的系统，为了确保同步和功能的正确性，环路上的延迟元件总数必须与原信号流图（SFG）中的数目相一致。在本例中，位串行环路上的延迟元件数目应等于 $W \times N_D$，其中 W 为信号字长，N_D 为原字级SFG的（字级）延迟元件数目。使字级环路上的延迟元件数目与位串行架构中的数目匹配被称为*环路延迟同步*（loop delay synchronization）。对于本例，原来内环含有一个字延迟，外环含有两个字延迟。因此，在所得的架构中，内环和外环应该分别含有8个和16个延迟元件。为了计算位串行架构的总延迟，所考虑的路径在开关元件中应该是*延迟元件最多的支路*。对于本例，中间架构的内环和外环分别含有6个和14个延迟元件。所以，应该在内环和外环的公共部分插入两个*同步延迟*，最终的架构由图13-26b示出。在输入端插入5个输入同步延迟元件，这也被称为*补偿*（shimming）延迟或*歪斜*（skewing）延迟。该位级流水线位串行IIR滤波器的关键路径含有一个一位加法器，吞吐量为每8个时钟周期一个输出字。注意中间位级流水线架构的环路也可能含有多于 $W \times N_D$ 个位级延迟元件。这种情况则需要增加字长。



a) 具有缩放算子的滤波器架构



b) 可实现的位级流水线架构

图13-24 位级流水线串行FIR滤波器，其中的常系数乘法用移位和加法实现

注意，如果没有两个环路同步延迟（阴影部分），该架构可以正确处理字长为6的信号，6就是图13-26b中的位级流水线位串行架构的*最小字长*（minimum wordlength）。（如果信号字长为6，外环上的 8D 延迟应改为 6D 延迟。）位级流水线位串行系统的最小可实现字长决定了它的最大吞吐量。在本例中，吞吐量为每6个时钟周期1个输出（字）。

图13-26b中的IIR滤波器架构是根据图13-27a所示的信号流图推导出的。图13-27a的信

号流图中的一乘二加（内环）时间的环路迭代周期限定了它的最小字长为6。利用结合律（associativity）得到如图13-27b所示的结构，可以将该迭代边界（见第2章）减小为一乘加。图13-27b中的结构的位串行实现由图13-28示出，其最小可实现字长为5。

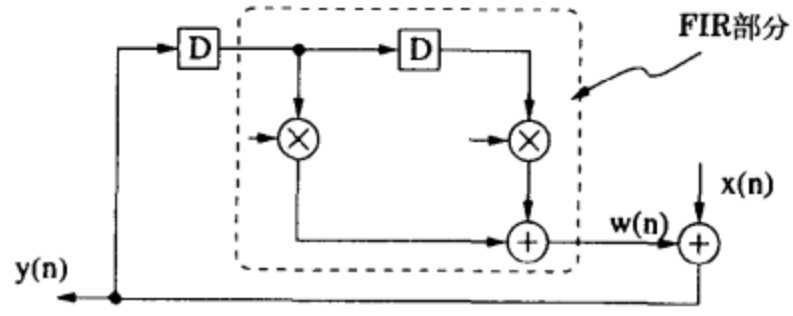
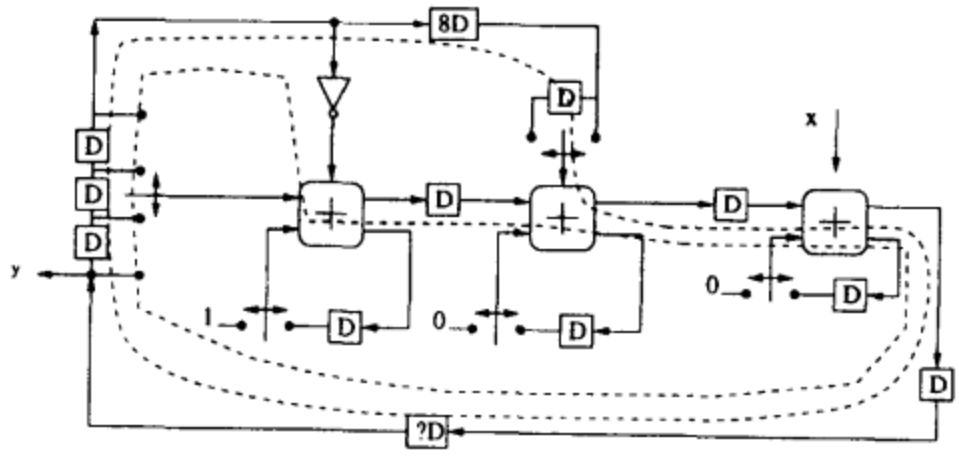
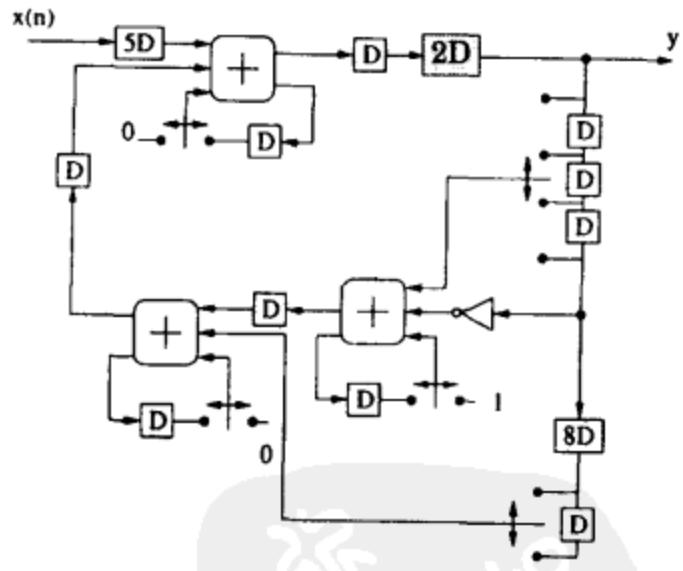


图13-25 本图中IIR滤波器可以用一个FIR部分获得



a) 没有同步延迟元件的位级流水线位串行架构



b) 位串行IIR滤波器。注意该实现的最小可实现字长为6

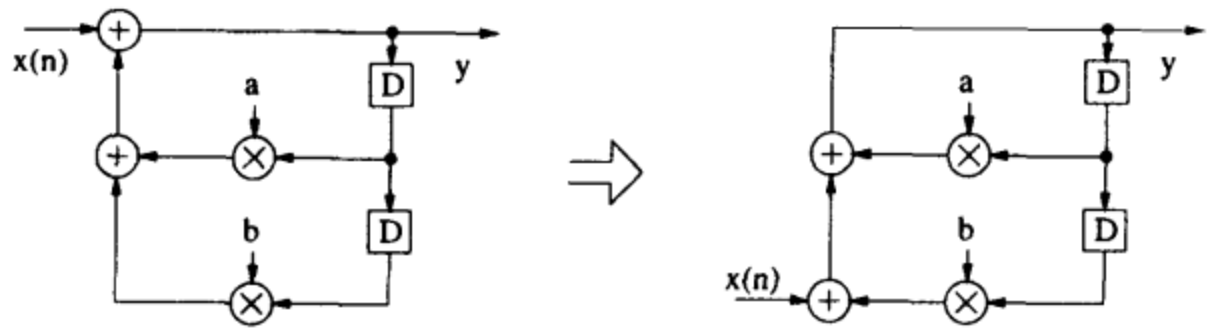
图13-26 位级流水线串行IIR滤波器的设计

例13.5.1 考虑方程：

$$x(n) = -\frac{7}{32}x(n-1) + \frac{3}{4}x(n-2) + u(n) \tag{13-19}$$

的实现。设信号字长为8（即 x 和 u 的字长为8位）。假设系数用二的补码格式，-7/32=-1+1/2+1/4+1/32编码为1.11001，3/4=1/2+1/4 编码为0.11。

基于图13-27b的框图的位串行架构由图13-29示出，其中符号 $(8l+i)$ 表示时刻 $\{8l+i\}$，7/8表示周期为8的一个周期中剩余的7个时间片。注意内环的环路延迟元件数目为8，外环为16（计算环路延迟元件数目时开关中的延迟元件数目应取各支路的最大值），这与字级架构是一致的。 ■



a) b)

501 ~ 504

图13-27 运用结合变换可以将IIR滤波器的环路迭代边界由一乘二加减小为一乘加

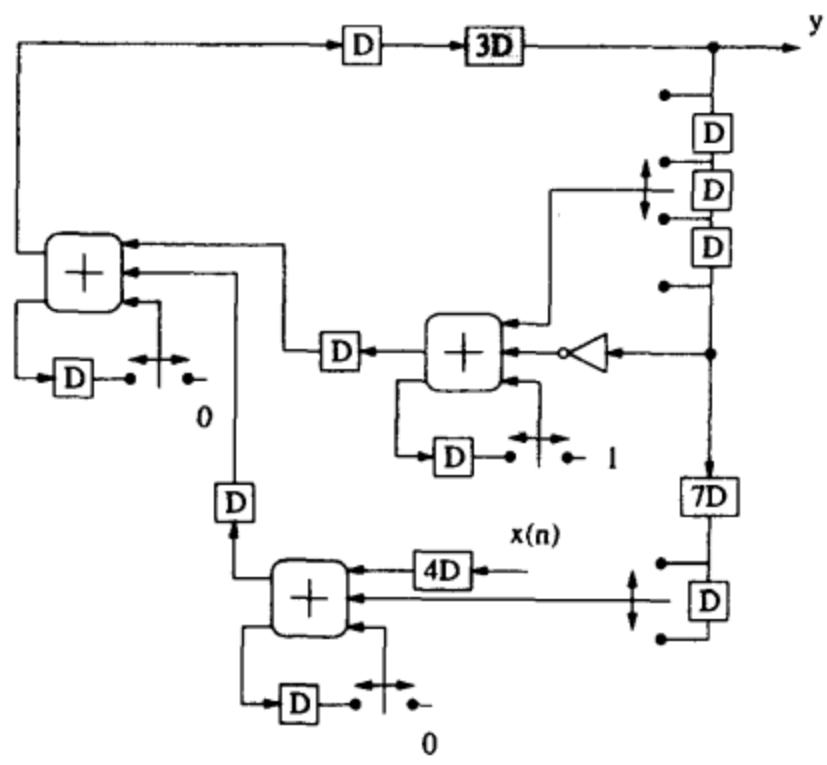


图13-28 经过结合变换的位串行IIR滤波器。该实现的最小可实现字长为5

13.6 正则符号数运算

一次常系数乘法所需的加法次数等于该常系数中非零位的数目减一。为了进一步减小面积和功耗，可以将常系数编码使得其中的非零位最少，这一点可以利用正则有符号数(canonic signed digit)（CSD）表示法实现。使用若干个有符号的二的幂(SPT)项的FIR数字滤波器的量化在附录G中论述。

13.6.1 CSD表示法

考虑数 $A = a_{W-1}a_{W-2}\cdots a_1a_0$，其中每个 a_i $(W-1 \geq i \geq 0)$ 在集合 $\{-1, 0, 1\}$ 中。二的补码表示法可以看成一种特例，其中位 a_{W-1} 等于0或者-1，而 $a_i = 0$ 或1，对于 $0 \leq i \leq W-2$。现

在乘法 $A \times X$，其中 X 为一变量，可以表示为

$$A \times X = \sum_{i=0}^{W-1} a_{W-1-i} X 2^{-i}$$

505

其中所需的加法或减法次数等于 A 中非零位的数目减一。CSD表示法是另一种特例，其中不存在连续的两个非零位 a_i。

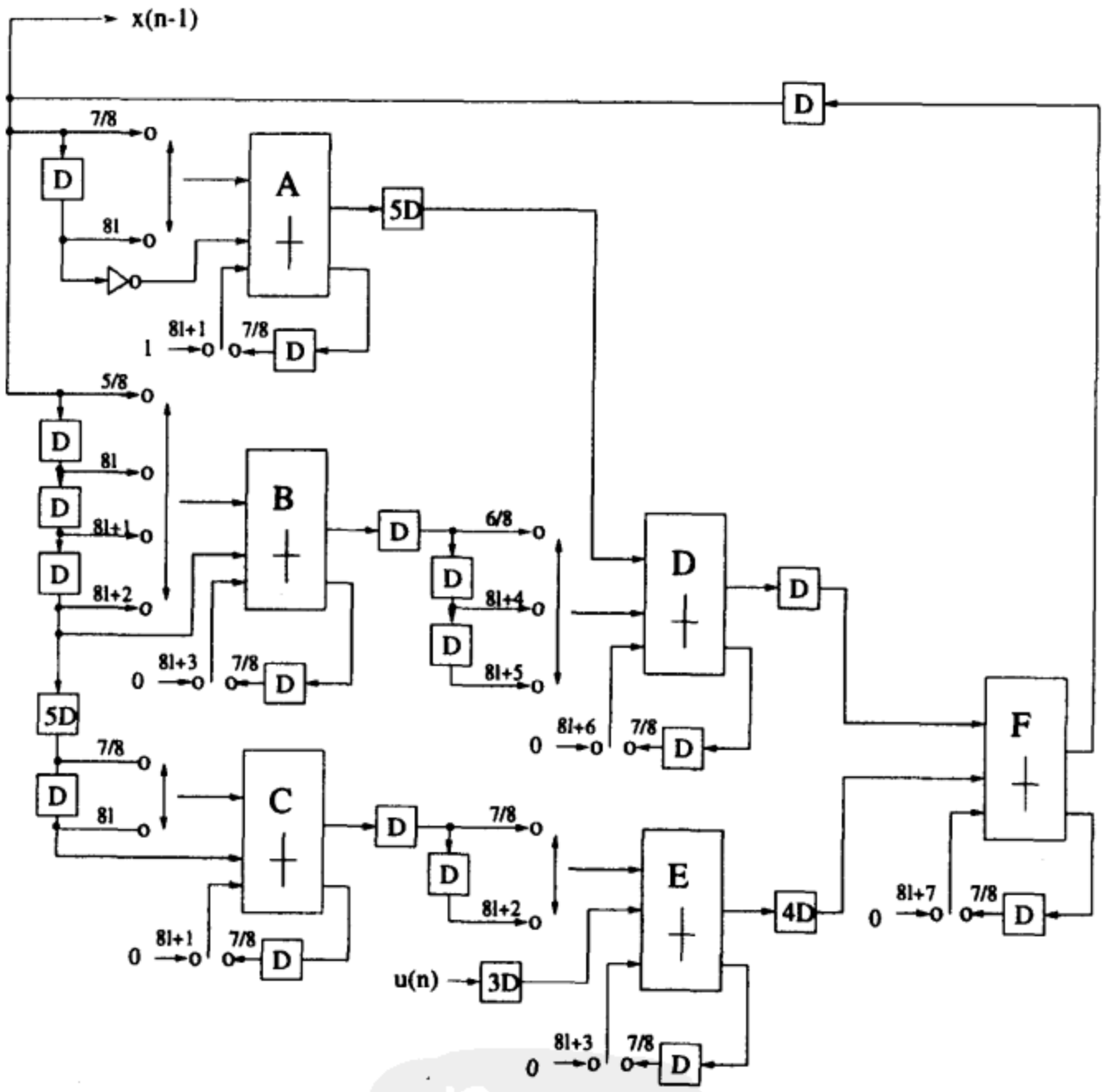


图13-29 使用二的补码运算的位串行架构

[16]讨论了CSD数表示法的性质，总结如下。

- CSD数中不存在连续两个非零位。
- 一个数的CSD表示中含有的非零位是所有表示中最少的，因而被称为是正则的(canonic)。
- 一个数的CSD表示是惟一的。
- CSD数覆盖区间 (−4/3, 4/3)，其中最有价值的是区间 [−1, 1)。
- 在区间 [−1, 1) 中的W位CSD数中，平均每个数的非零位数为 $W/3+1/9+O(2^{-W})$。所以，CSD数中的非零位数平均比二的补码数中的非零位数少33%。

506

[16]中介绍了一种计算W位数的CSD表示的算法。记数A的二的补码表示为$A=\hat{a}_{W-1}\cdot\hat{a}_{W-2}\cdots\hat{a}_1\hat{a}_0$，CSD表示为$A=a_{W-1}\cdot a_{W-2}\cdots a_1a_0$。下面的迭代算法说明了转换的方法。

$$
\begin{aligned}
&\hat{a}_{-1} = 0\\
&\gamma_{-1} = 0\\
&\hat{a}_W = \hat{a}_{W-1}\\
&\text{for } (i = 0 \text{ to } W-1)\\
&\quad\{\\
&\quad\theta_i = \hat{a}_i \oplus \hat{a}_{i-1}\\
&\quad\gamma_i = \overline{\gamma_{i-1}}\,\theta_i\\
&\quad a_i = (1-2\hat{a}_{i+1})\gamma_i\\
&\quad\}
\end{aligned}
$$

其中，符号⊕表示异或，上划线表示求补。

例13.6.1 输入为数1.01110011。其CSD表示的计算如下：

i	W	$W-1$				$\cdots$				0	-1
$\hat{a}_i$	1	1	0	1	1	1	0	0	1	1	0
θ_i		1	1	0	0	1	0	1	0	1	
γ_i		0	1	0	0	1	0	1	0	1	0
$1-2\hat{a}_{i+1}$		-1	-1	1	-1	-1	-1	1	1	-1	
a_i		0	-1	0	0	-1	0	1	0	-1	

■

13.6.2 CSD乘法

如前所述，常数乘法可以通过根据常乘数的非零位加上或减去相应的部分积完成。用CSD编码的乘数中含有的非零位最少，所以所需的加法或减法也最少。使用CSD表示法，乘法$x\times0.10100\bar{1}0010\bar{1}00\bar{1}$可以按图13-30计算，其中部分积按线性排列的方式累加。但是，该实现在计算时间和精度方面均不佳。本节介绍基于Horner法则的乘法和树高度减小技术以解决这些问题。注意这两种技术也可应用于二的补码运算。

507

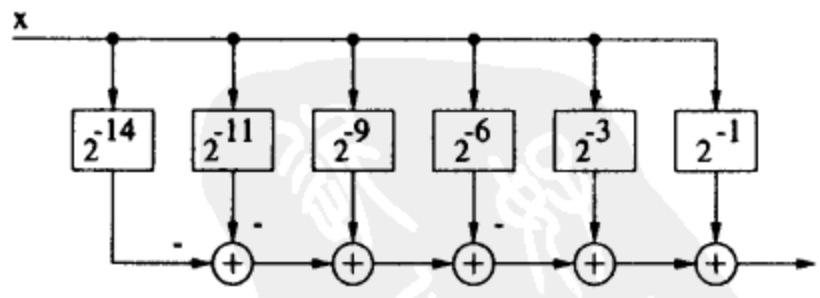


图13-30 使用线性排列的加法器的CSD乘法器

13.6.2.1 利用Horner法则提高精度

在固定字长乘法器中，部分和和最终积的多余位将被截断，累加两个部分积所需的二进制加法器数目等于字长W。这就产生了截断误差，其值不但取决于被乘数和乘数，而且取决于部分积累加的排列方式。

例如，考虑加法 $x \cdot 2^{-5} + x \cdot 2^{-3}$，设变量 x 的字长为8。可以按图13-31a来计算该加法，其中的两个部分积为 $x \cdot 2^{-5}$ 和 $x \cdot 2^{-3}$；累计计算只涉及方框中的那些位，更低的位被截断。也可以按 $(x \cdot 2^{-2} + x)2^{-3}$ 来计算，其中的两个部分积为 $x \cdot 2^{-2}$ 和 x，如图13-31b所示。在这种情况下，所需的加法器数目不变；但是，与图13-31a中的计算相比，累加计算涉及更多的位，截断误差也更小。因此，通过推迟这两个部分积中公共的缩放运算，可以提高最终结果的精度。但是，由于考虑到动态范围，某些缩放运算需要在加法之前进行。例如，为了使操作数 x 的动态范围达到 $[-1, 1)$，$x \cdot 2^{-5} + x \cdot 2^{-3}$ 应该改写成 $(x \cdot 2^{-3} + x \cdot 2^{-1})2^{-2}$。在这个表示式中加法器的输出不可能溢出。注意为了使 $(x \cdot 2^{-2} + x)$ 不溢出，x 的动态范围为 $(-0.8, 0.8)$。我们不再进一步考虑动态范围的问题。

类似地，图13-30中 $x \times 0.10100\bar{1}0010\bar{1}00\bar{1}$ 的计算可以重新组合成

$$
\begin{aligned}
& x \times 0.10100\bar{1}0010\bar{1}00\bar{1} \\
= \; & x \cdot (2^{-1} + 2^{-3} - 2^{-6} + 2^{-9} - 2^{-11} - 2^{-14}) \\
= \; & 2^{-1}(x + 2^{-2}(x + 2^{-3}(-x + 2^{-3}(x + 2^{-2}(-x - 2^{-3}x))))) \\
= \; & 2^{-1}(x + 2^{-2}(x + 2^{-3}(-x + 2^{-3}(x - 2^{-2}(x + 2^{-3}x)))))
\end{aligned}
$$

从最内层括号开始计算，并推迟同一括号中的两个部分积中公共的缩放运算，如图13-32所示。这种利用Horner法则的扩展可以减小截断误差，提高乘法的精度。 508

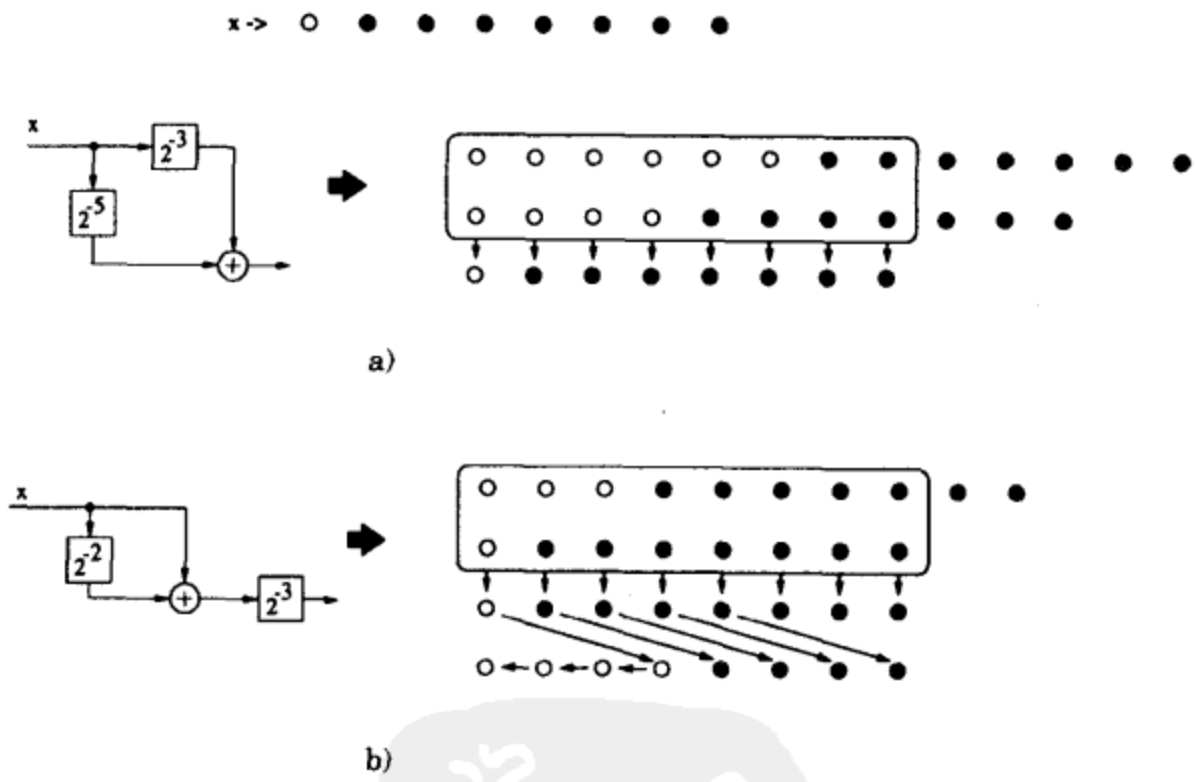


图13-31 利用Horner法则进行部分积累加以减小截断误差

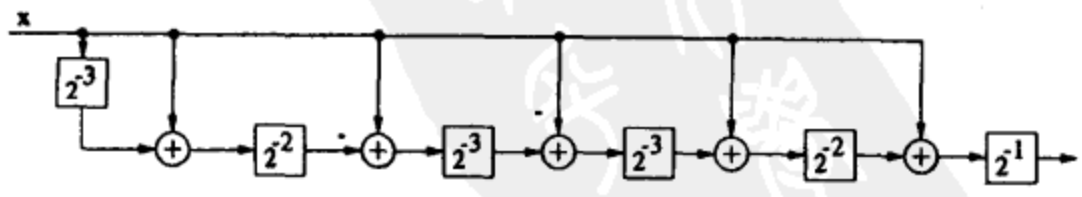
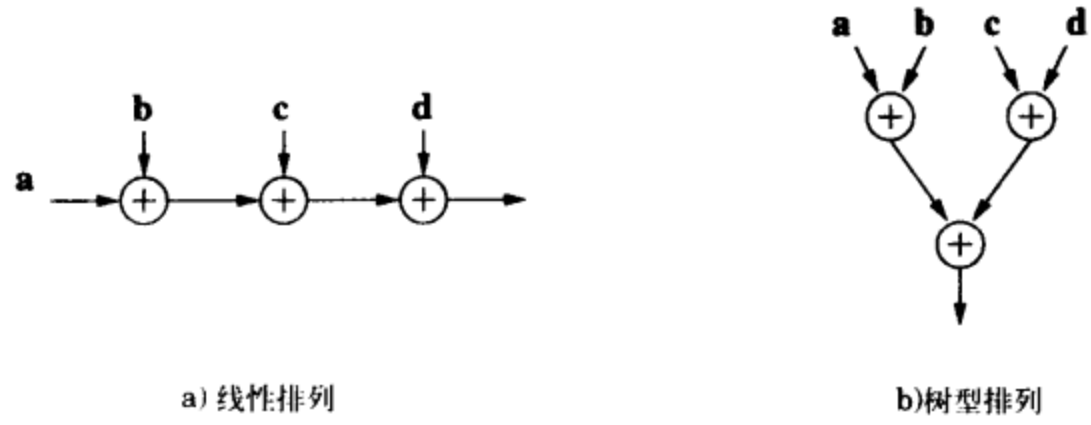


图13-32 利用Horner法则重新排列图13-30中的CSD乘法的部分积累加以减小截断误差

注意为了防止溢出，加法的每个操作数在相加之前至少需要一次缩放运算。如第13.2节所述，这里我们在变量 x 中设定一个警戒位。这样就不必在加法之前进行缩放运算。

13.6.2.2 利用树高度减小减小等待时间

N个值的累加可以排列成线性或树型结构的方式，对于 $N=4$ 如图13-33所示。在相同的硬件需求下，树型排列可以将等待时间从 $(N-1)T_a$ 减小至 $\lceil \log_2 N \rceil T_a$，其中 T_a 表示一次加法运算的计算时间。



a) 线性排列　　b)树型排列

图 13-33

图13-30和图13-32中的CSD乘法的部分积是按线性排列的方式累加的。可以通过使用加法器树减小它们的等待时间。在本例中，按如下方式完成乘法

$$
\begin{aligned}
& x \times 0.10100\bar{1}0010\bar{1}00\bar{1} \\
= \; & x \cdot (2^{-1} + 2^{-3} - 2^{-6} + 2^{-9} - 2^{-11} - 2^{-14}) \\
= \; & 2^{-1}((x + 2^{-2}x) + 2^{-5}((-x + 2^{-3}x) - 2^{-5}(x + 2^{-3}x)))
\end{aligned}
$$

如图13-34所示。

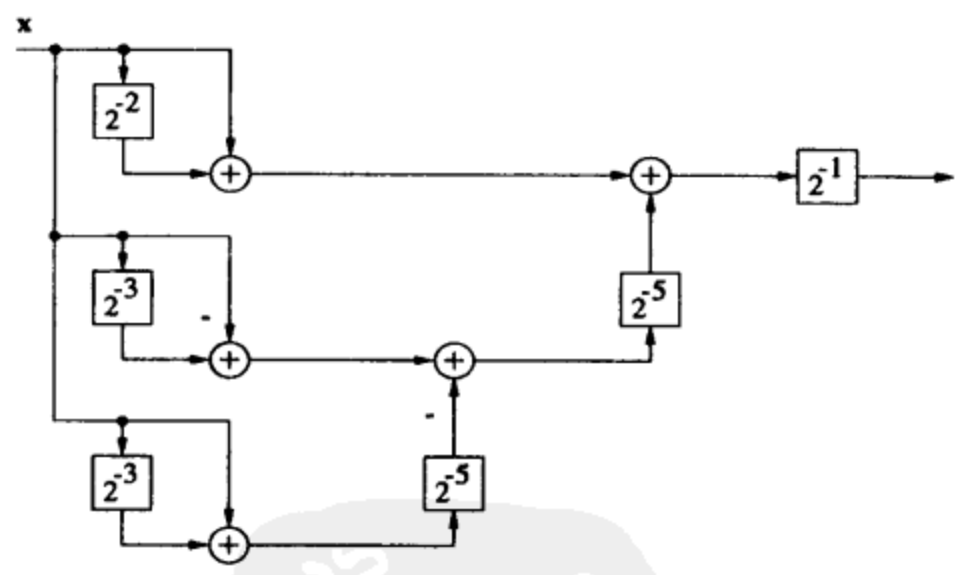


图13-34 CSD乘法的部分积累加的树型和Horner法则的组合

综上所述，对于固定字长乘法，利用Horner法则和树高度减小变换可以减小截断误差和计算等待时间。对于全精度乘法，也可以使用这两种方法减少所需的加法器数目。

509 ~ 510

例13.6.2 在本例中，利用正则符号数表示法重新设计例13.5.1中的IIR滤波器。这样按CSD格式，$-7/32=-1/4+1/32$ 编码为 $0.0\bar{1}001$；$3/4=1-1/4$编码为 $1.0\bar{1}$。使用CSD表示法的位串行架构由图13-35示出。注意加法器数目从6减至4。另外，由于内环乘法的等待时间此时为6个时间单位，最小可实现字长减至 6+1=7 位，而使用二的补码表示法为8位。 ■

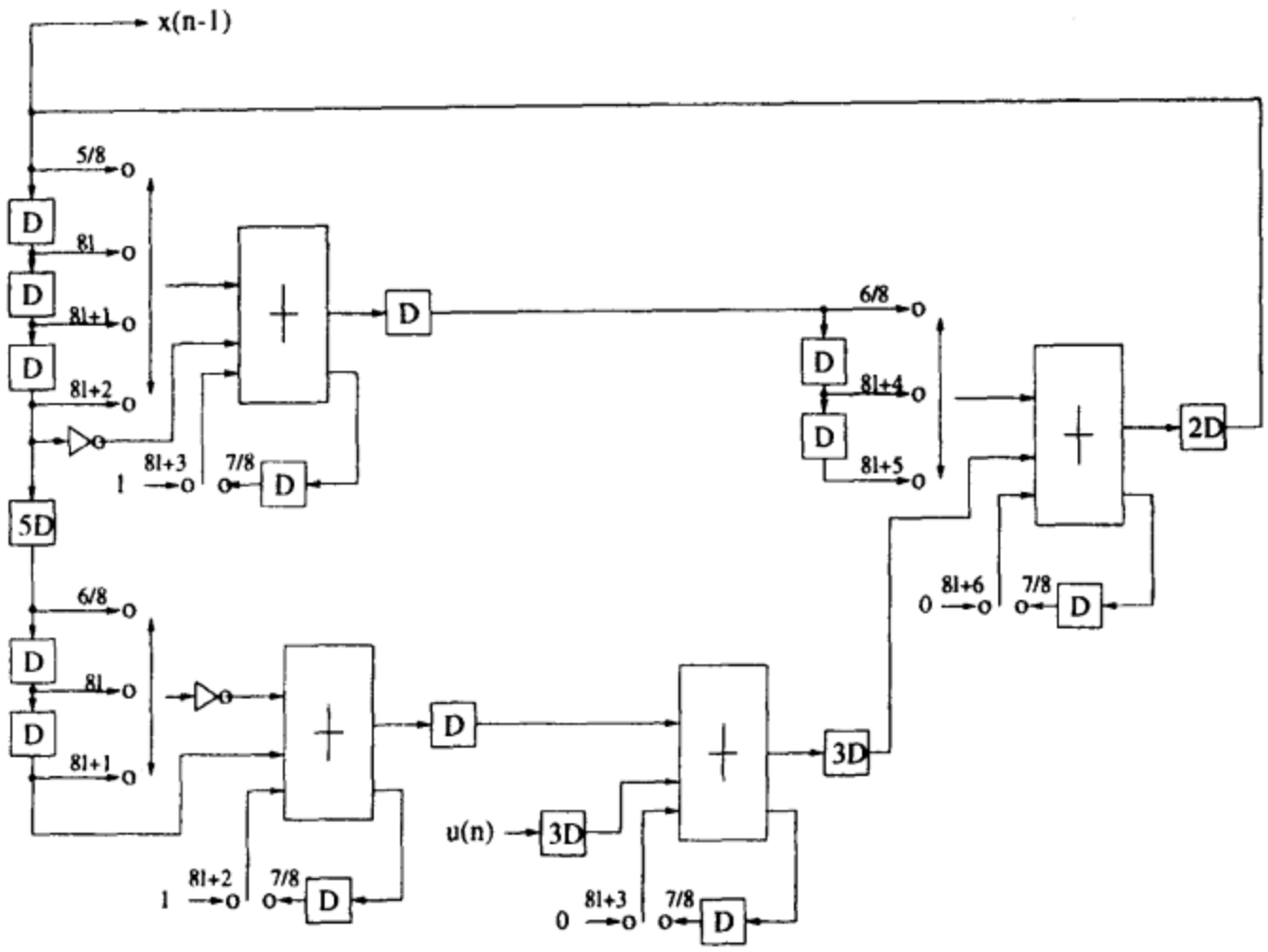


图13-35 使用CSD运算的位串行架构

13.7 分布式运算

分布式运算（distributed arithmetic）被用来设计矢量-矢量乘法的位级架构。在分布式运算中，矢量中的各个字均用一个二进制数表示，乘法被重新排序和混合，使得运算“分布”于整个结构中。分布式运算常用于在视频压缩系统中常用的卷积运算和离散余弦（DCT）的实现[17]、[18]。

13.7.1 传统的分布式运算

考虑两个长度为 N 的矢量 C 和 X 的内积:

$$Y = \sum_{i=0}^{N-1} c_i x_i \tag{13-20}$$

511

其中 $\{c_i\}$ 为 M 位常数，$\{x_i\}$ 用 W 位二的补码数编码

$$x_i = -x_{i,W-1} + \sum_{j=1}^{W-1} x_{i,W-1-j} 2^{-j} \tag{13-21}$$

将（13-21）代入（13-20），

$$Y = \sum_{i=0}^{N-1} c_i(-x_{i,W-1} + \sum_{j=1}^{W-1} x_{i,W-1-j} 2^{-j}) \tag{13-22}$$

$$= -\sum_{i=0}^{N-1} c_i x_{i,W-1} + \sum_{j=1}^{W-1}\left(\sum_{i=0}^{N-1} c_i x_{i,W-1-j}\right)2^{-j}$$

定义

$$C_{W-1-j} = \sum_{i=0}^{N-1} c_i x_{i,W-1-j}\ (j \neq 0),\ C_{W-1} = -\sum_{i=0}^{N-1} c_i x_{i,W-1} \tag{13-23}$$

那么，

$$Y = \sum_{j=0}^{W-1} C_{W-1-j} 2^{-j} \tag{13-24}$$

所以，通过交换对 i 和 j 求和的次序，原来（13-20）的乘法现在分布（distributed）成为另一种计算形式[19]，[20]。

由于项 C_j 取决于值 $x_{i,j}$，且只有 2^N 个可能值，那么可以预先计算出这些值并保存在只读存储器（ROM）中。N位的输入集合 $(x_{0j}, x_{1j}, \cdots, x_{N-1,j})$ 被用作地址获得相应的 C_j 值。这W个中间值在W个时钟周期中累加起来，得到一个Y值。这就是矢量乘法的乘数无关实现。表13-2所示的是 N=4 时ROM的内容。图13-36所示的是计算两个长度为N的矢量的内积的典型架构。其中的移位累加器是位并行进位传播加法器，用来将ROM的内容与前一个累加值相加。反相器和MUX用来将ROM的输出反相以计算 C_{W-1}，控制信号 S 当 j=W−1 时为1否则为0。从 j=0 到 j=W−1 重复进行这个计算，W个时钟周期后则按位并行的格式得到结果。这种方法就是位串行分布式运算。对于某些实时应用，如用于数字电视系统中的视频压缩的DCT实现，传统的位串行分布式运算实现的速度可能不够快。这些系统的速度可以通过数位串行分布式运算[21]，[22]来提高，这种运算在一个时钟周期处理一个含有多个位的数位。例如，如果使用 J 个ROM，一个时钟周期处理 J 个连续的位，那么 W / J 个时钟周期处理完输入的字。使用一个多输入移位累加器将 J 个ROM的内容与前一个累加值相加，按位并行的格式输出。习题24探讨了多输入移位累加器的详细设计。

512

表13-2 ROM的内容（N=4）

$x_{0,j}$	$x_{1,j}$	$x_{2,j}$	$x_{3,j}$	ROM的内容
0	0	0	0	0
0	0	0	1	c_3
0	0	1	0	c_2
0	0	1	1	$c_2 + c_3$
0	1	0	0	c_1
0	1	0	1	$c_1 + c_3$
0	1	1	0	$c_1 + c_2$
0	1	1	1	$c_1 + c_2 + c_3$
1	0	0	0	c_0
1	0	0	1	$c_0 + c_3$
1	0	1	0	$c_0 + c_2$
1	0	1	1	$c_0 + c_2 + c_3$
1	1	0	0	$c_0 + c_1$
1	1	0	1	$c_0 + c_1 + c_3$
1	1	1	0	$c_0 + c_1 + c_2$
1	1	1	1	$c_0 + c_1 + c_2 + c_3$

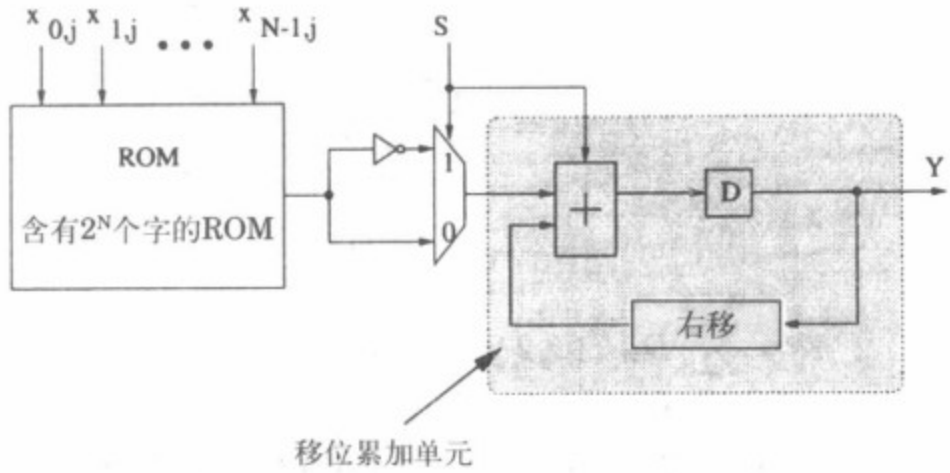


图13-36 利用分布式运算计算两个长度为 N 的矢量的内积的架构

13.7.2 使用偏移二进制编码的分布式运算

本节介绍偏移二进制编码（OBC）[18]，它可以将ROM的大小减小为原来的一半，即 2^{N-1}。将（13-21）改写成:

$$\begin{aligned} x_i &= \frac{1}{2}[x_i - (-x_i)] \\ &= \frac{1}{2}[-(x_{i,W-1} - \overline{x_{i,W-1}}) + \sum_{j=1}^{W-1}(x_{i,W-1-j} - \overline{x_{i,W-1-j}})2^{-j} - 2^{-(W-1)}] \end{aligned} \tag{13-25}$$

其中

$$-x_i = -\overline{x_{i,W-1}} + \sum_{j=1}^{W-1} \overline{x_{i,W-1-j}}2^{-j} + 2^{-(W-1)}$$

定义

$$d_{i,j} = \begin{cases} x_{i,j} - \overline{x_{i,j}}, & 当\ j \neq W-1 \\ -(x_{i,W-1} - \overline{x_{i,W-1}}), & 当\ j = W-1 \end{cases} \tag{13-26}$$

且 $d_{i,j} \in \{-1, +1\}$。公式(13-25)可以改写成:

$$x_i = \frac{1}{2}[\sum_{j=0}^{W-1} d_{i,W-1-j}2^{-j} - 2^{-(W-1)}] \tag{13-27}$$

利用（13-27），（13-20）可以写成

$$\begin{aligned} Y &= \sum_{i=0}^{N-1} \frac{1}{2}c_i[\sum_{j=0}^{W-1} d_{i,W-1-j}2^{-j} - 2^{-(W-1)}] \\ &= \sum_{j=0}^{W-1}(\sum_{i=0}^{N-1} \frac{1}{2}c_i d_{i,W-1-j})2^{-j} - (\frac{1}{2}\sum_{i=0}^{N-1} c_i)2^{-(W-1)} \end{aligned} \tag{13-28}$$

现在定义

$$D_j = \sum_{i=0}^{N-1} \frac{1}{2}c_i d_{i,j}, \quad 当\ 0 \leqslant j \leqslant W-1 \tag{13-29}$$

513 ~ 514

和

$$D_{extra} = -\frac{1}{2}\sum_{i=0}^{N-1} c_i \tag{13-30}$$

所以，我们有

$$Y = \sum_{j=0}^{W-1} D_{W-1-j}2^{-j} + D_{extra}2^{-(W-1)} \tag{13-31}$$

公式（13-29）~（13-31）为OBC方法的特征。表13-3所示的是ROM的内容。

显然，在ROM表中 D_j 值关于第8行和第9行的分界线镜像对称。换句话说，项 D_j 只有 2^{N-1} 个可能值，它们取决于 $x_{i,j}$ 的值。所以ROM的大小可以减小为原来的一半。表13-4所示的是新的ROM表，图13-37所示的是使用以OBC编码的分布式运算计算N输入内积的典型架构。同样地，计算从 x_i 的最低位开始，即 j=0。XOR门用于地址解码，具有常数 D_{extra} 输入的MUX用来为移位累加器提供初始值，ROM后面的MUX在 j=W−1 时用来将ROM的输出反相。需要两个控制信号 S_1 和 S_2，当 j=W−1 时 S_1 为1否则为0，当 j=0 时 S_2 为1否则为0。

表13-3 使用OBC编码的ROM的内容（N=4）

$x_{0,j}$	$x_{1,j}$	$x_{2,j}$	$x_{3,j}$	ROM的内容
0	0	0	0	$-(c_0+c_1+c_2+c_3)/2$
0	0	0	1	$-(c_0+c_1+c_2-c_3)/2$
0	0	1	0	$-(c_0+c_1-c_2+c_3)/2$
0	0	1	1	$-(c_0+c_1-c_2-c_3)/2$
0	1	0	0	$-(c_0-c_1+c_2+c_3)/2$
0	1	0	1	$-(c_0-c_1+c_2-c_3)/2$
0	1	1	0	$-(c_0-c_1-c_2+c_3)/2$
0	1	1	1	$-(c_0-c_1-c_2-c_3)/2$
1	0	0	0	$(c_0-c_1-c_2-c_3)/2$
1	0	0	1	$(c_0-c_1-c_2+c_3)/2$
1	0	1	0	$(c_0-c_1+c_2-c_3)/2$
1	0	1	1	$(c_0-c_1+c_2+c_3)/2$
1	1	0	0	$(c_0+c_1-c_2-c_3)/2$
1	1	0	1	$(c_0+c_1-c_2+c_3)/2$
1	1	1	0	$(c_0+c_1+c_2-c_3)/2$
1	1	1	1	$(c_0+c_1+c_2+c_3)/2$

表13-4 使用OBC编码的（减小大小的）ROM的内容（N=4）

$x_{1,j}$	$x_{2,j}$	$x_{3,j}$	ROM的内容
0	0	0	$-(c_0+c_1+c_2+c_3)/2$
0	0	1	$-(c_0+c_1+c_2-c_3)/2$
0	1	0	$-(c_0+c_1-c_2+c_3)/2$
0	1	1	$-(c_0+c_1-c_2-c_3)/2$
1	0	0	$-(c_0-c_1+c_2+c_3)/2$
1	0	1	$-(c_0-c_1+c_2-c_3)/2$
1	1	0	$-(c_0-c_1-c_2+c_3)/2$
1	1	1	$-(c_0-c_1-c_2-c_3)/2$

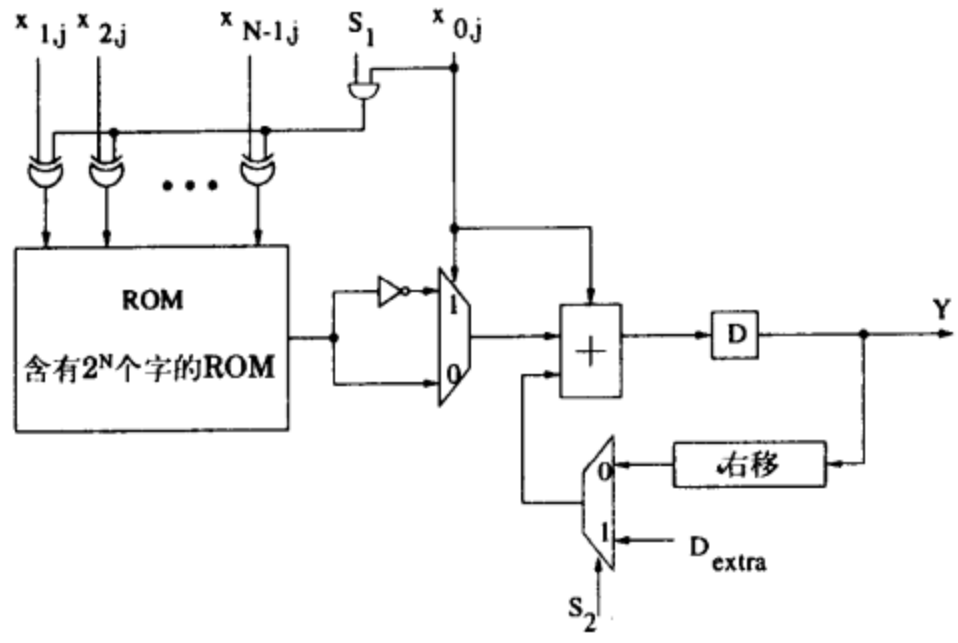


图13-37 使用以OBC编码的分布式运算计算两个长度为N的矢量的内积的架构

515 ~ 516

13.7.3 分布式运算的ROM分解

传统的分布式运算的ROM大小随N指数增长。一般地，ROM访问时间是整个系统速度的瓶颈，尤其是在ROM比较大的时候。所以，减小ROM大小非常重要，并且有很大的实用价值。由于公式（13-24）是线性的，存在一个解可以将ROM的N个地址位分解成 N/K 个K位的组。因此可以将大小为 2^N 的ROM分解成 N/K 个大小为 2^K 的ROM，并使用多输入累加器将这些ROM的输出相加。图13-38所示的是使用具有ROM分解的传统分布式运算计算N输入内积的架构。总的存储大小从 2^N 减小为 $(N/K)2^K$，只随N线性增长。ROM访问时间也随之减小。存储大小的减小是以累加器的运算复杂度为代价的。进位保留运算可以用来实现多输入累加器以使计算时间最小。注意ROM分解技术也可应用于基于OBC编码的分布式运算。

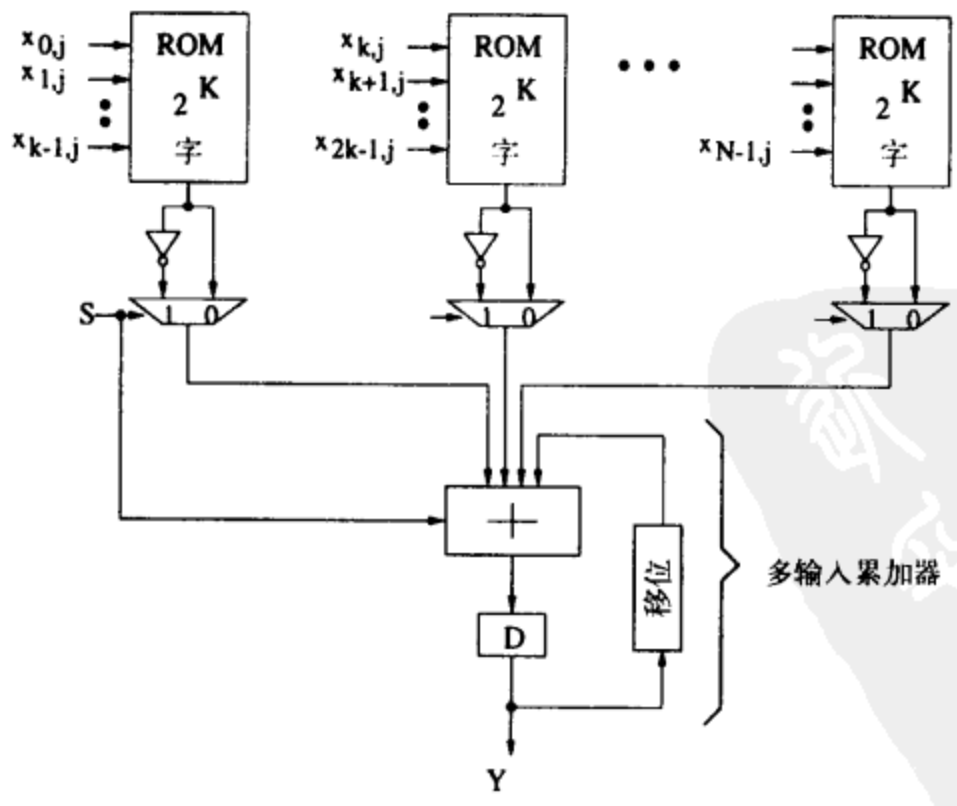


图13-38 使用具有ROM分解的分布式运算计算两个长度为N的矢量的内积的架构

13.8 结论

本章讨论了位级运算架构的设计。介绍了位并行乘法器的设计，包括串行进位阵列、进位保留阵列、Baugh-Wooley和Booth重编码乘法。讨论了数字滤波器的两种高效的布局策略，交织布局规则（interleaved floor-plan）与位平面（bit-plane）。介绍了利用Horner法则和利用脉动映射的位串行乘法器的设计。本章还涉及了具有常系数的位级流水线位串行FIR和IIR滤波器的设计方法学。介绍了适用于低成本的高速常数乘法器的CSD表示法。讨论了利用结合律和树高度减小来减小串行和并行运算中的等待时间。最后给出了基于分布式运算方法的矢量-矢量乘法的位级实现方案。FIR数字滤波器和变换的实现中常用的余数运算[23]不在本书的讨论范围之内。

数位串行架构对于离散小波变换也很有吸引力，小波变换中阶越高的小波可以用越小的数位大小实现[24]。例如，在3阶小波中，数位大小 $W/2$、$W/4$、$W/8$ 可以分别用于1、2、3阶。越高的阶需要越小的计算速率，假设系统使用单一时钟，就可以用越少的硬件实现。传统上，数位串行乘法器是通过折叠位并行乘法器或者展开位串行乘法器进行设计的。这样得到的一些数位串行架构不能形成位级流水线。[25]中介绍的新式单元取代（cell replacement）变换技术可以用来根据位串行乘法器设计低功耗位级流水线数位串行乘法器。

13.9 习题

1. (a) 验证图13-39中的电路实现了一个二进制全加器。

   (b) 图13-39中的加法器电路使用了32个晶体管。推导一个优化架构，使用CMOS传输门，且只需要24个晶体管（提示：尽量共享求和和求进位部分的硬件）。

   (c) 8位串行进位加法器的关键路径上的晶体管有多少个？

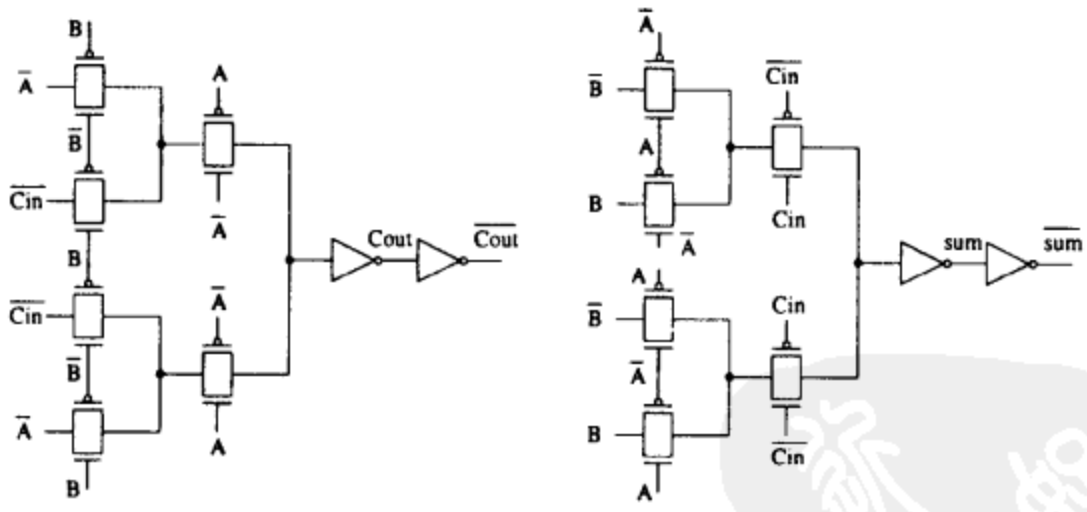


图13-39 基于互补传输门逻辑的CMOS全加器电路图

2. 考虑两个二的补码8位数 X 和 Y 按串行进位方式的加法，如图13-40所示。推导适用于该加法器的由位 x_7、y_7、c_7、c_8 和 s_7 检测溢出的真值表和逻辑电路。（提示：如果 $x_7 \neq y_7$，即 X 和 Y 的符号位不同，那么不可能发生溢出。如果 $x_7 = y_7$，那么当 $x_7 = y_7 \neq s_7$ 时发生溢出。）

3. 设计计算

$$y(n) = \sum_{i=0}^{5} x_i(n) \tag{13-32}$$

的位并行架构，使用

(a) 进位保留运算

(b) 树高度减小技术

并使用半加器、全加器和VMA。设字长为8。计算这些架构的等待时间，不包括VMA的等待时间。

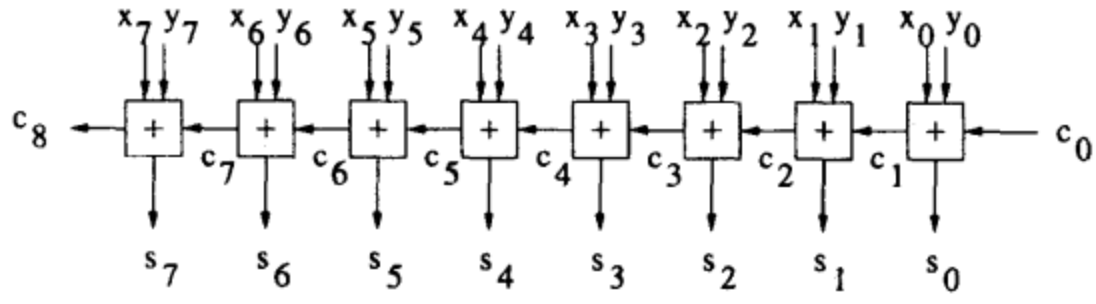


图13-40 习题2中的8位串行进位加法器

4. 本题考虑图13-8所示的Baugh-Wooley乘法算法的推导。考虑4×4位乘法运算 $X=A\times B$，其中 $A=a_3\cdot a_2a_1a_0=-a_3+a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3}$，，$B=b_3\cdot b_2b_1b_0=-b_3+b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3}$。 518~519

A 与 B 相乘等价于计算

$$\begin{aligned} P &= (-a_3+a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})(-b_3+b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3}) \\ &= (a_3b_3+(a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})(b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3})) \\ &\quad -(a_3(b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3})+(a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})b_3) \end{aligned} \tag{13-33}$$

(a) 证明

$$\begin{aligned} &-(a_3(b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3})+(a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})b_3) \\ &= (\overline{a_3b_2}+\overline{a_2b_3})2^{-1}+(\overline{a_3b_1}+\overline{a_1b_3}+1)2^{-2}+(\overline{a_3b_0}+\overline{a_0b_3})2^{-3} \end{aligned} \tag{13-34}$$

其中 $\overline{a}=1-a$，对于 $a=0$ 或1。提示：两个在区间 [-1, 1) 中的数的积仍然在该区间中。因此可以忽略更高位置上的非零位。

(b) 利用公式（13-33）和（13-34）以及结合性质，重新组合乘法中涉及的所有项，推导图13-8所示的Baugh-Wooley乘法表。

5. Baugh-Wooley乘法算法还可以按如下方式实现：

$$\begin{aligned} P &= (-a_3+a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})(-b_3+b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3}) \\ &= a_3b_3+(a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})(b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3}) \\ &\quad +\overline{a_3}+(a_3\overline{b_2}2^{-1}+a_3\overline{b_1}2^{-2}+a_3\overline{b_0}2^{-3})+a_3 2^{-3} \\ &\quad +\overline{b_3}+(\overline{a_2}b_3 2^{-1}+\overline{a_1}b_3 2^{-2}+\overline{a_0}b_3 2^{-3})+b_3 2^{-3} \end{aligned} \tag{13-35}$$

通过将（13-34）改写成：

$$\begin{aligned} &-(a_3(b_2 2^{-1}+b_1 2^{-2}+b_0 2^{-3})+(a_2 2^{-1}+a_1 2^{-2}+a_0 2^{-3})b_3) \\ &= a_3(-b_2 2^{-1}-b_1 2^{-2}-b_0 2^{-3})+b_3(-a_2 2^{-1}-a_1 2^{-2}-a_0 2^{-3}) \end{aligned} \tag{13-36}$$

证明（13-35）中的Baugh-Wooley乘法算法，并推导该算法的乘法表。（提示：利用事实 $\sum_{i=1}^{3}-b_{3-i}2^{-i}=\sum_{i=1}^{3}(1-b_i)2^{-i}-1+2^{-3}$。）

6. 考虑

$$x(n) + fx(n-1) + gx(n-2) \tag{13-37}$$

520 的计算。使用Baugh-Wooley乘法算法和二的补码表示法，画出该计算的交织位并行布局架构，并具有

(a) 位级流水线

(b) 2位级流水线

信号和系数的字长为4。

7. 利用图13-13中的乘法表，用改进的位平面技术设计习题6中的计算的2位级流水线位并行架构。

8. 给出图13-19中的位串行乘法器的详细调度，并验证其正确性。

9. 给出图13-23中的位串行Baugh-Wooley乘法器的详细调度，并验证该架构的正确性。

10. 找出适当的投影矢量、调度矢量和处理器空间矢量，根据图13-3中的DG设计具有位级流水线的Lyon位串式串行进位乘法器。该乘法器应该是图13-17中的乘法器的加上流水线的版本。完成设计的细节，并给出乘法器中所有多路器的开关时刻。

11. 本题与利用Horner法则扩展的字长为12的位串行的基于5级重编码的改进Booth乘法器的完整架构设计有关。使用小数点在最高位（符号位）右边、位数11为最高位而位数0为最低位的记法。

(a) 为了获得重编码后的信号

$$y_i' = -2y_{2i+1} + y_{2i} + y_{2i-1} \tag{13-38}$$

我们需要推导出三个控制信号 A_i、B_i 和 C_i。C_i=1 表示 y_i' 为负，C_i=0 表示 y_i' 非负。B_i=1 表示 $|y_i'|$=2 而 A_i=1 表示 $|y_i'|$=1。画出 A_i、B_i 和 C_i 的卡诺图，并证明

$$\begin{aligned} A_i &= y_{2i} \oplus y_{2i-1} \\ B_i &= (y_{2i+1} \oplus y_{2i})\overline{(y_{2i} \oplus y_{2i-1})} \\ C_i &= y_{2i+1} \end{aligned} \tag{13-39}$$

其中符号⊕表示异或，上划线表示求补。利用 A_i、B_i 和 C_i，设计改进的Booth重编码单元。

521

(b) 画出完整的改进后的Booth重编码位串行乘法器架构，并给出所有的开关时刻。

12. 本题考虑低功耗Booth重编码电路的设计。习题11中的Booth重编码控制电路只包含两个XOR门和一个反相器。但是，控制信号 A_i、B_i 和 C_i 是经过具有不同的传输延迟的不同逻辑产生的，导致部分积产生和累加电路中的毛刺增多，从而带来较高的功耗。通过使用冗余重编码方案，平衡控制电路的输入到输出的各条路径，可以减少毛刺，减小功耗[11]。

考虑乘法 $A \times Y$，记重编码后的乘数位为 y_j'。在新的无竞争（race-free）Booth重编码器中使用4个控制信号：neg、$x1$、$x2$ 和 zp。neg=1 表示 y_j' 为负，neg=0表示 y_j' 非负。$x1$=1表示 $|y_j'|=1$; $x2=\overline{x1}$。zp 用来区分 $|y_j'|=2$ 和 $|y_j'|=0$ 两种情况。新的重编码方案总结于表13-5，其中*表示无关（don't-care）情形。使用这种重编码方案得到的部分积产生器的一个位带由图13-41示出。

表13-5 习题12中的低功耗Booth重编码电路的真值表

y_{2j+1}	y_{2j}	y_{2j-1}	y'_j	neg	$x1$	$x2$	zp
0	0	0	0	0	0	1	1
0	0	1	1	0	1	0	*
0	1	0	1	0	1	0	*
0	1	1	2	0	0	1	0
1	0	0	−2	1	0	1	0
1	0	1	−1	1	1	0	*
1	1	0	−1	1	1	0	*
1	1	1	0	1	0	1	1

(a) 证明图13-41中的电路为Booth乘法产生正确的部分积。

(b) 推导Booth重编码（控制）电路，使得从各输入位，包括 y_{2j+1}、y_{2j}、y_{2j-1}，a_i 和 a_{i-1}，它们到部分积的各位 Ay'_j 路径均相等，并且均含有一个XOR/XNOR门延迟、一个AND门延迟和一个NOR门延迟。zp 信号的无关（don't-care）情形可以用来使逻辑最小化。给出你用来代替这些无关（don't-care）情形的 zp 值。 522

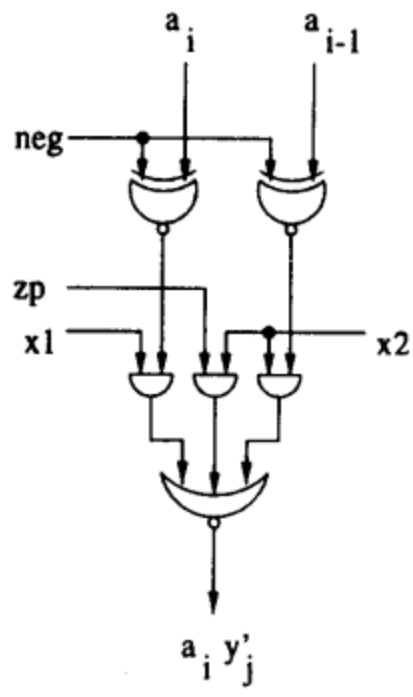


图13-41 习题12中的无竞争Booth乘法器的部分积产生器的一个位带

13. 通过展开图13-17中的Lyon乘法器得出二阶展开串行进位数位串行乘法器。

14. 本题与全极点二阶递归滤波器

$$y(n) = -\frac{7}{8}y(n-1) + \frac{3}{4}y(n-2) + x(n)$$

的定点位串行和数位串行实现有关。设信号字长为8（即 y 和 x 的字长为8位）。假设系数−7/8编码为1.001，3/4编码为0.11。在与系数相乘时只执行与非零位的相乘。

(a) 利用结合律，画出该运算的方框图。你的架构的内环边界限制为一个乘加时间。

(b) 根据(a)中的结构设计功能正确的位级流水线位串行架构。

(c) (b)中的环路迟滞限制了信号 y 和 x 的最小字长。(b)中的系统的最小可实现信号字长是多少？

15. 本题考虑图13-42所示的三阶定系数IIR滤波器的位串行实现。设数据字长为8，滤波器系数字长为4，滤波器系数 $a=-1/4$，$b=1/8$，$c=-3/4$。假设所有的数字均用二的补码表示法表示。

523

(a) 利用结合律，得出一种使得内环边界限制为一个乘加时间的等价结构。

(b) 完成(a)中的结构的位级流水线位串行设计，将设计中的延迟元件的最少化定为次要目标。

16. 本题考虑递归运算

$$y(n)=\frac{3}{16}y(n-1)+\frac{5}{8}y(n-2)+\frac{1}{2}x(n)$$

的位级流水线位串行实现，计算使用2的补码表示法，且信号字长为10。位串行设计必须利用Horner法则来提高精确度。并且必须利用结合律使得字级的内环的环路边界限制为一乘和一加时间。设计信号字长为10的位串行架构。该运算的最小可实现字长是多少？

17. 考虑图13-43中所示的运算

$$y(n)=0.25y(n-1)+x(n) \tag{13-40}$$

的位串行实现。

该电路中使用的字长是多少？对于这个字长，完成缺少的开关时刻。将该位串行结构3阶展开，得出数位大小为3的数位串行实现。

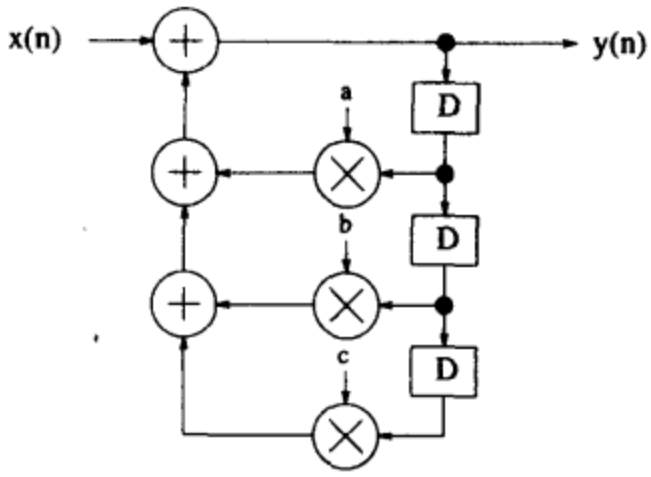


图13-42 习题15的IIR滤波器框图

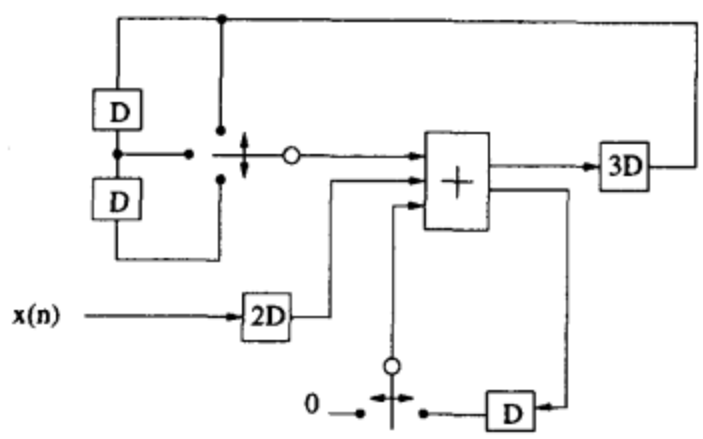


图13-43 习题17中的IIR滤波器的位串行实现

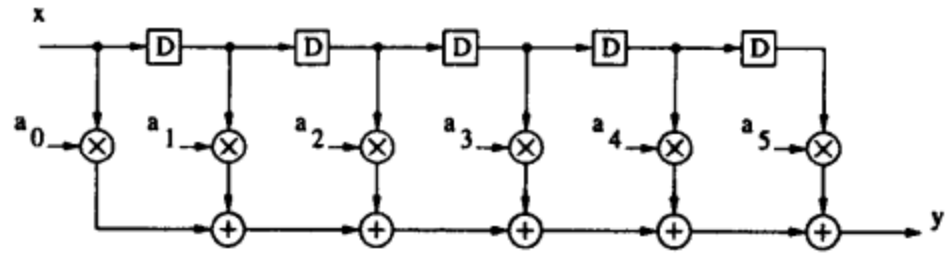
18. 本题考虑利用结合律和重新定时对串行进位Lyon乘法器进行变换，设计低等待时间位串行乘法器。

(a) 说明可以利用结合律和重新定时将图13-44a中的架构变换为图13-44b中的架构。比较这两个架构的关键路径和等待时间。

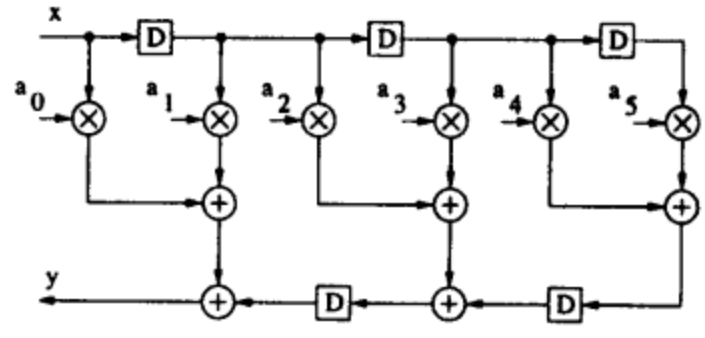
(b) 通过对Lyon串行进位位串式乘法器进行类似于(a)中的变换，或者对Horner法则乘法恒等式的形式重新进行变换，设计低等待时间位串行乘法器。设信号样本和系数的字长为12。给出你的架构中的所有开关时刻。

(c) 只利用(b)中的架构，通过考虑与非零位之间的乘法，设计计算 $y(n)=\alpha x(n)$ 的低等待时间常数位串行乘法器，其中 $\alpha=0.01011100101$。给出该架构中的所有开关时刻。设信号字长为12。

(d) 基于(c)中推导出的位串行乘法器，设计递归运算 $y(n) = ay(n-1) + x(n)$ 的位串行实现，其中 a 与(c)中的 a 相同。设信号字长为12。



a)



b)

图13-44 习题18中的位串行乘法器

19. 计算出下列2的补码数字的CSD表示:

$$c_0 = 0.00010110,\quad c_1 = 0.01001100,\quad c_2 = 0.01100110,$$
$$c_3 = 0.00010110111$$

20. 计算出下列2的补码数字的CSD表示:

$$c_0 = 1.11111111,\quad c_1 = 1.11110010,\quad c_2 = 0.01100110,$$
$$c_3 = 0.01001100111$$

21. 本题考虑IIR滤波器

$$y(n) = a_1 y(n-1) + a_2 y(n-2) + x(n) \tag{13-41}$$

其中 $a_1 = 0.00011011100$，$a_2 = 0.01001111110$。

(a) 将系数 a_1 和 a_2 用CSD表示法表示出来。

(b) 利用CSD表示法和图13-27b中的数据流图，得出信号字长为12的位级流水线位串行IIR滤波器架构。该架构的最小可实现字长是多少?

(c) 应用Horner法则和树高度减小技术，改进(b)中的设计。该设计的最小可实现字长是多少?

22. 扩展OBC的概念，实现使用一个存储 $2^N/4$ 个字的ROM的分布式运算架构。

23. 考虑 N=16，K=4 的ROM分解。计算ROM大小的缩小倍数。使用二输入累加器和进位保留运算，实现该分布式运算架构。

24. 本题考虑数位串行分布式运算架构的设计。设字长为9，利用3位数位串行分布式运

524 ~ 526

算，实现运算

$$Y = \sum_{i=0}^{N-1} c_i x_i \tag{13-42}$$

位 $x_{i,\,3k+l}$ 应该在第 k 个时钟周期 (k = 0, 1, 2)、在第 l 个ROM (l=0, 1, 2) 中进行处理。使用3个ROM和一个多输入移位累加器，设计3位数位串行分布式运算架构。该架构在一个时钟周期中处理3个连续的输入位，在3个时钟周期中处理完所有的输入字。假设ROM的内容用16位表示。使用位并行(i)串行进位和(ii)进位保留加法器，分别设计两种形式的移位累加器。移位累加器的输出用16位表示。串行进位设计的关键路径应该为 $17t_{FA}$，而进位保留设计的关键路径应该为 $3t_{FA}$，其中 t_{FA} 为一个全加器的传输延迟。进位保留设计的末级需要一个额外的进位传播加法器。

(a) 计算两种设计的迟滞，用 t_{FA} 表示。

(b) 哪一个设计更适合于高速度的应用？为什么？

(c) 哪一个设计更适合于低功耗的应用？为什么？

参考文献

1. L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to implementation of digital filters," *IEEE Trans. on Audio Electroacoustics*, vol. 16, pp. 413–421, 1968.
2. P. B. Denyer and D. Renshaw, *VLSI Signal Processing: A Bit-Serial Approach.* Addison-Wesley, 1986.
3. R. Jain et al., "Custom design of a VLSI PCM-FDM transmultiplexor from system specification to circuit layout using a computer aided design system," *IEEE J. Solid State Circuits*, vol. 21, pp. 73–85, Feb. 1986.
4. K. K. Parhi, "A systematic approach for design of digit-serial processing architectures," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 4, pp. 358–375, April 1991.
5. R. I. Hartley and K. K. Parhi, *Digit-Serial Computation.* Kluwer, 1995.
6. N. E. Weste and K. Eshraghian, *Principles of CMOS VLSI design: A Systems Perspective.* Addison-Wesley, 1993.
7. K. Hwang, *Computer Arithmetic: Principles, Architecture and Design.* Wiley, 1979.
8. I. Koren, *Computer Arithmetic Algorithms.* Prentice Hall, 1993.
9. C. R. Baugh and B. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. on Computers*, vol. C-22, no. 12, pp. 1045–1047, Dec. 1973.
10. C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. on Computers*, vol. EC-13, pp. 14–17, Feb. 1964.
11. R. Fried, "Minimizing energy dissipation in high-speed multipliers," in *Proc. of ISLPED-97*, (Monterey, CA), pp. 214–219, Aug. 1997.
12. M. Hatamian and K. K. Parhi, "An 85-MHz fourth-order programmable IIR digital filter chip," *IEEE Journal of Solid State Circuits*, vol. 27, pp. 175–183, Feb. 1992.

527

13. T. G. Noll, "Semi-systolic maximum rate transversal filters with programmable coefficients," in *Proc. of 1986 International Conference on Systolic Arrays*, (Oxford, U.K.), pp. 5–13, July 1986.

14. R. F. Lyon, "Two's complement pipelined multipliers," *IEEE Trans. on Communications*, vol. 24, pp. 418–424, April 1976.

15. D. Ait-Boudaoud, M. K. Ibrahim, and B. R. Hayes-Gill, "Novel cell architecture for bit level systolic arrays multiplication," *IEE Proceedings-E*, vol. 138, Jan. 1991.

16. S. W. Reitwiesner, "Binary arithmetic," *Advances in Computers*, pp. 231–308, 1966.

17. M.-T. Sun, T.-C. Chen, and A.-M. Dottlieb, "VLSI implementation of a 16×16 discrete cosine transform chip," *IEEE Trans. on Circuits and Sytems*, vol. 36, no. 4, pp. 610–617, April 1989.

18. N. Demassieux and F. Jutand, "Orthogonal transforms," in *VLSI Implementations for Image Communications* (P. Pirsch, ed.), pp. 217–250, Elsevier, 1993.

19. A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP-22, no. 6, pp. 456–462, Dec. 1974.

20. C. S. Burrus, "Digital filter structures described by distributed arithmetic," *IEEE Trans. on Circuits and Systems*, Dec. 1977.

21. S. Uramoto et al, "A 100-MHz 2-D discrete cosine transform core processor," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 492–499, April 1992.

22. Y. Katayama, I. Tamitani, A. Taniguchi, and Y. Ooi, "A single-chip MPEG1 audio/video decoder using macrocore and cell-based implementation," in *IEEE VLSI Signal Processing, VIII*, (Osaka, Japan), pp. 431–440, Oct. 1995.

23. F. J. Taylor, "Residue arithmetic: a tutorial with examples," *IEEE Trans. on Computers*, May 1984.

24. K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. on VLSI Systems*, vol. 1, no. 2, pp. 191–202, June 1993.

25. Y.-N. Chang, J. H. Satyanarayana, and K. K. Parhi, "Design and implementation of low-power digit-serial multipliers," in *Proc. 1997 IEEE International Conference on Computer Design (ICCD)*, (Austin, TX), pp. 186–195, Oct. 1997.

愛知県
図書館

第14章 冗余运算

14.1 引言

在普通的非冗余基 r 数字系统中，一个数位可以在 $\{0, 1, \cdots, r-1\}$ 中取值，并且所有的数字都能够以惟一的方式表示。一个基r冗余有符号数位数字系统是基于数位集合 $S \triangleq \{\overline{\beta}, \overline{\beta-1}, \cdots, \overline{1}, 0, 1, \cdots, \alpha\}$的，其中符号 $\overline{x}$ 表示 $-x$，$1 \leq \beta$, $\alpha \leq r-1$，并且数位集合 S 中的值的数目大于 r [1]。最后的这个条件允许任意数字可以用多种有符号格式表示，因此名为冗余（redundant）。一般使用的是对称有符号数位表示法（$\alpha=\beta$），在本章中将对它进行讨论。例如，数位集合 $\{\overline{1}, 0, 1\}$ 就用于基2（$r=2$）冗余数字系统。在这种系统中，数字3可以表示为0011或 $010\overline{1}$，等等。有符号表示法已经在第13章的CSD表示法和Booth重编码部分中用过；但是那些表示法都是惟一的。

冗余有符号数字系统的吸引人之处在于其“无进位”加法的性质。这个性质使得它们在除法和平方根操作的数位串行实现中非常有用，这些运算都是从最高数位（msd）开始计算的。最高数位优先冗余运算也被称为*在线运算*（on-line arithmetic）。在线运算对于递归数字滤波器很有吸引力，因为其无进位性质会得到低等待时间或者高速度[2]。注意2的补码运算不能够用于在线运算。冗余运算也可以用于最低数位（lsd）优先串行运算。但是这样它们就不一定比2的补码运算更加高效。

进位保留运算也导出一种形式的冗余数字表示法。进位保留运算中的“进位-和”数可以取值0、1、2，而在基2表示法的冗余表示法中可取值为-1、0、1。因为进位保留和冗余运算的相似，大多数冗余架构可以改编成进位保留架构。但本章重点是冗余运算。

本章从冗余数字表示法的基本概念开始，着重于冗余数字系统的运算架构的设计。冗余数字表示法，包括最小和最大冗余表示法，在第14.2节中给出。第14.3节给出了基2混合的和有符号数位的加法和减法。最小和最大冗余基4加法在第14.4节中讨论。基2冗余乘法器的设计在第14.5节中考虑。最后，在第14.6节中介绍冗余表示法和2的补码表示法之间的数据格式转换架构，包括最低数位优先和最高数位优先模式。

14.2 冗余数表示

在普通的2的补码表示法中，所有的数字用惟一的方式表示，所以是非冗余的。因此，在普通的加法中，进位可能从最低数位一直传播到最高数位[3]。在冗余表示法中，加法可以在与操作数字长 W 无关的恒定时间内完成。虽然进位的传播局限于向左2或3个位置，但冗余加法却被称为无进位传播，或者简单地称为无进位[4]，[5]。无进位性质使得最高数位优先计算方式或在线运算得以实现[6]~[9]。

对称有符号表示法使用数位集合 $D_{\langle r, \alpha \rangle} = \{\overline{\alpha}, \cdots, \overline{1}, 0, 1, \cdots, \alpha\}$，其中$r$为基，$\alpha$为数位集合中最大数位。一个用使用这种表示法的数字可写成:

$$X_{<r,\alpha>} = x_{W-1} \cdot x_{W-2} x_{W-3} \cdots x_0 = \sum_{i=0}^{W-1} x_{W-1-i} r^i \tag{14-1}$$

数字的符号由最高的非零数位的符号给定。

如果数位集合 $D_{<r,\alpha>}$ 中的数位数目小于r，即 $\alpha < \frac{r-1}{2}$，那么该数位集合是不完全的，因为有的数字无法表示。如果基r是奇数且数位数目等于r，即 $\alpha = \frac{r-1}{2}$，那么该数位集合是完全但非冗余的，即在可能区间内的任何数字只能惟一表示，与普通表示法中的情形相同（注意对于偶数基r，有符号数位集合不可能是完全但非冗余的）。最后，如果数位数目大于r，即 $\alpha > \left\lceil \frac{r}{2} \right\rceil$，那么该数位集合是冗余的。而且，如果 $\alpha = \left\lceil \frac{r}{2} \right\rceil$，那么该数位集合被称为最小冗余（minimally redundant），如果 $\alpha=r-1$，那么它被称为最大冗余（maximally redundant），如果 $\alpha > r-1$，那么它被称为过冗余（over-redundant）[10]，[11]。

530

有符号数位表示法的冗余度用冗余因子（redundancy factor）ρ 来度量，ρ 定义为:

$$\rho = \frac{\alpha}{r-1} \tag{14-2}$$

表14-1是关于不同的有符号数位表示法的冗余性质的总结。注意冗余因子越大，表示法的冗余度越大。例如，如果 $\rho = \frac{1}{2}$，则该数位集合是完全但非冗余的（只有 r 为奇数这才可能）。如果 $\frac{1}{2} < \rho < 1$，则该数位集合是最小冗余。如果 $\rho=1$，则该数位集合是最大冗余。

表14-1 有符号数位表示法的冗余性质

数位集合$D_{<r,\alpha>}$	α	冗余因子ρ
不完全	$< \frac{r-1}{2}$	$< \frac{1}{2}$
完全但非冗余	$= \frac{r-1}{2}$	$= \frac{1}{2}$
冗余	$> \left\lceil \frac{r}{2} \right\rceil$	$> \frac{1}{2}$
最小冗余	$= \left\lceil \frac{r}{2} \right\rceil$	$> \frac{1}{2}$ 且<1
最大冗余	$=r-1$	$=1$
过冗余	$> r-1$	>1

基2有符号数位表示 $D_{<2,1>}$ 还被称为有符号二进制数位（SBD）表示法、冗余有符号数位（RSD）表示法和借位保留（BS）表示法[4]，[12]~[14]。

可以用有符号表示法表示的数字范围是 $[-\rho(r-r^{-W+1}), \rho(r-r^{-W+1})]$。注意该范围取决于字长 W 和冗余因子 ρ。

14.3 无进位基2加法与减法

冗余数字表示法将进位传播局限在几个位中，而且一般与字长W无关。这种无进位传播

的性质使得快速加法成为可能[1]。完成有符号二进制数位加法的算法并不惟一[5]，[12]，[15]。所以它们的逻辑实现也可以各不相同。 531

基2有符号数位数字用两个无符号二进制数字编码，一个为正一个为负，$X=X^+ - X^-$。因此，每个有符号数位用两个位表示，$x_i = x_i^+ - x_i^-$，其中 $x_i^+, x_i^- \in\{0, 1\}$ 且 $x_i \in\{\bar{1}, 0, 1\}$ [4]，[13]。本节考虑基2混合的和有符号数位的加法和减法。在混合操作中，一个输入操作数以及输出操作数是用冗余有符号数位表示法表示的，第二个输入操作数是普通的无符号数字。一个有符号数位加法可以看作是一个混合加法和一个混合减法的串联。

14.3.1 混合基2加法

考虑一个基2有符号数位数字 $X_{<2.1>}$ 和一个无符号普通数字 Y 的加法

$$S_{<2.1>} = X_{<2.1>} + Y \tag{14-3}$$

其中 $X_{<r.a>} = x_{W-1} \cdot x_{W-2}x_{W-3}\cdots x_0$，$Y_{<b.a>} = y_{W-1} \cdot y_{W-2}y_{W-3}\cdots y_0$，且 $x_i \in\{\bar{1},0,1\}$，$y_i \in\{0,1\}$。这个加法可以用两步完成。

第一步是以并行方式在所有位置 i $(0 \le i \le W-1)$ 上进行的。计算出中间和 $p_i = x_i + y_i$，它的值在集合 $\{\bar{1},0,1,2\}$ 中。这个加法表示成

$$x_i + y_i = p_i = 2t_i + u_i \tag{14-4}$$

其中 t_i 是转移数位（transfer digit），值为0或1，记作 t_i^+；u_i 是临时和（interim sum），值为$\bar{1}$或0，记作 $-u_i^-$。最低的转移数位 t_{-1} 被置为零，临时和的最高数位 u_W 同样被置为零。在第二步中，将 t_{i-1}^+ 和 u_i^- 组合成一个数位得到和的数位：

$$s_i = t_{i-1}^+ - u_i^- \tag{14-5}$$

表14-2总结了混合基2加法所涉及的数位集合。表的最后一列对应于数位的二进制码。

表14-2 混合基2加法所涉及的数位集合

数 位	基2数位集合	二进制码
x_i	$\{\bar{1},0,1\}$	$x_i^+ - x_i^-$
y_i	$\{0,1\}$	y_i^+
$p_i = x_i + y_i$	$\{\bar{1},0,1,2\}$	$\cdots$
u_i	$\{\bar{1},0\}$	$-u_i^-$
t_i	$\{0,1\}$	t_i^+
$s_i = u_i + t_{i-1}$	$\{\bar{1},0,1\}$	$s_i^+ - s_i^-$

将表14-2中相应的二进制码代入（14-4），我们得到：

$$x_i^+ - x_i^- + y_i^+ = 2t_i^+ - u_i^- \tag{14-6}$$

这个运算操作可以用一个推广的1型全加器[16]来完成，又被称为加-加-减加法器（PPM）[4]。只要简单地将两条线 t_{i-1}^+ 和 u_i^- 组合成一个数位就得到和的数位，所以，$s_i = s_i^+ - s_i^- = t_{i-1}^+ - u_i^-$。图14-1给出了一个8数位并行混合基2加法器的结构。注意和有9个数位，比加数多一个数位。

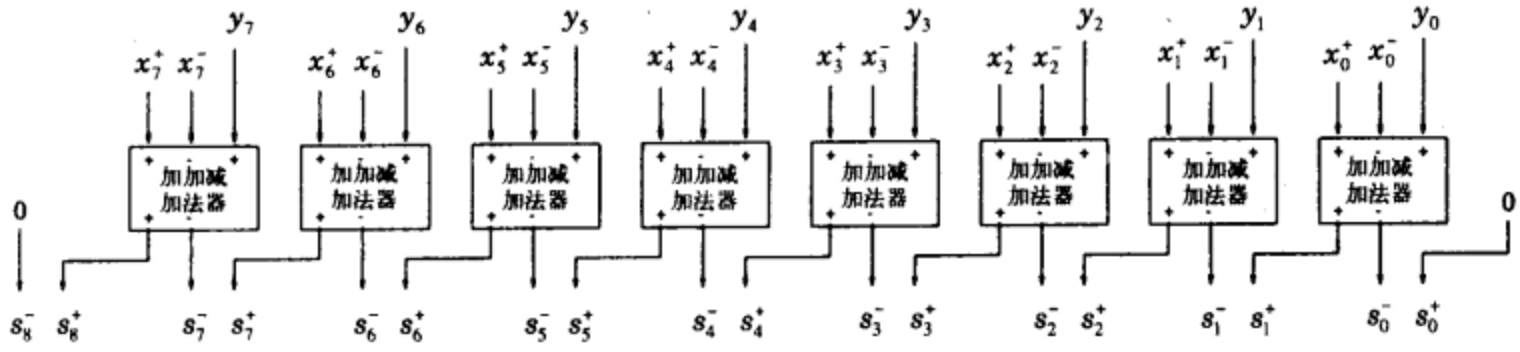
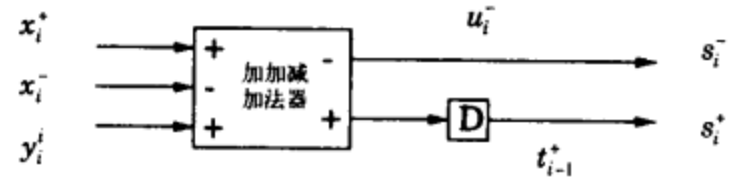
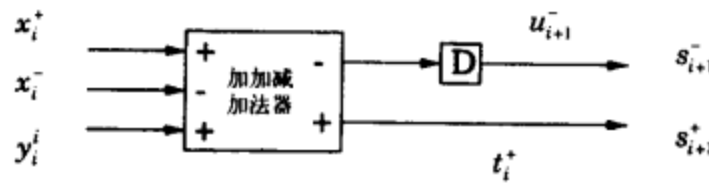


图14-1 8数位混合基2加法器

混合冗余加法也可以用一个PPM加法器串行完成。数位串行混合加法器可以通过将并行加法器按最低数位优先或者最高数位优先的模式折叠（见第6章）而得。最低数位优先和最高数位优先数位串行混合冗余加法器由图14-2中(a)和(b)分别示出。最低数位优先加法器的等待时间为零，而最高数位优先加法器的等待时间为一个时钟周期。对于8数位加法，和有9个数位，因此需要在两个相邻的输入操作数之间插入一个零数位，从而一次加法需要9个时钟周期。并且，这个插入的零数位还在和的最低数位或最高数位的数位上产生零位。



a) 最低数位优先加法器



b) 最高数位优先加法器

图14-2 数位串行混合基2冗余加法器

14.3.2 混合基2减法

在混合减法中，从一个有符号基 r 数字中减去一个无符号普通数字Y。运算 $S_{<2.1>}=X_{<2.1>}-Y$ 可以按照与混合加法相似的方式完成。区别在于这里使用的是负的转移数位 t_i。

在第一步中，数位无关地计算出中间差 $p_i=x_i-y_i$，它的值在集合 $\{\bar{2},\bar{1},0,1\}$ 中，用下式表示:

$$x_i-y_i=p_i=2t_i+u_i \tag{14-7}$$

其中转移数位 t_i 的值为 $\bar{1}$ 或0，记作 $-t_i^-$，临时差 u_i 的值为0或1，记作 u_i^+。在第二步中，将 t_{i-1}^- 和 u_i^+ 组合成一个数位得到和的数位 s_i:

$$s_i=-t_{i-1}^-+u_i^+ \tag{14-8}$$

表14-3列出了混合基2减法所涉及的数位集合。

表14-3 混合基2减法所涉及的数位集合

数 位	基2数位集合	二进制码
x_i	$\{\bar{1},0,1\}$	$x_i^+ - x_i^-$
y_i	$\{0,1\}$	y_i^-
$p_i = x_i - y_i$	$\{\bar{2},\bar{1},0,1\}$	$\cdots$
u_i	$\{0,1\}$	u_i^+
t_i	$\{\bar{1},0\}$	$-t_i^-$
$s_i = u_i + t_{i-1}$	$\{\bar{1},0,1\}$	$s_i^+ - s_i^-$

将表14-3中相应的二进制码代入（14-7），我们得到：

$$x_i^+ - x_i^- - y_i^- = -2t_i^- + u_i^+ \tag{14-9}$$

532 ~ 534

这个运算操作可以用一个推广的2型全加器[16]来完成，又被称为减-减-加加法器（MMP）[4]。

还有，$s_i^+ = u_i^+$ 和 $s_i^- = t_{i-1}^-$。所以，混合基2减法的第二步将这两条线组合在一起就形成了和的数位。图14-3给出了一个8数位并行混合基2减法器的结构。

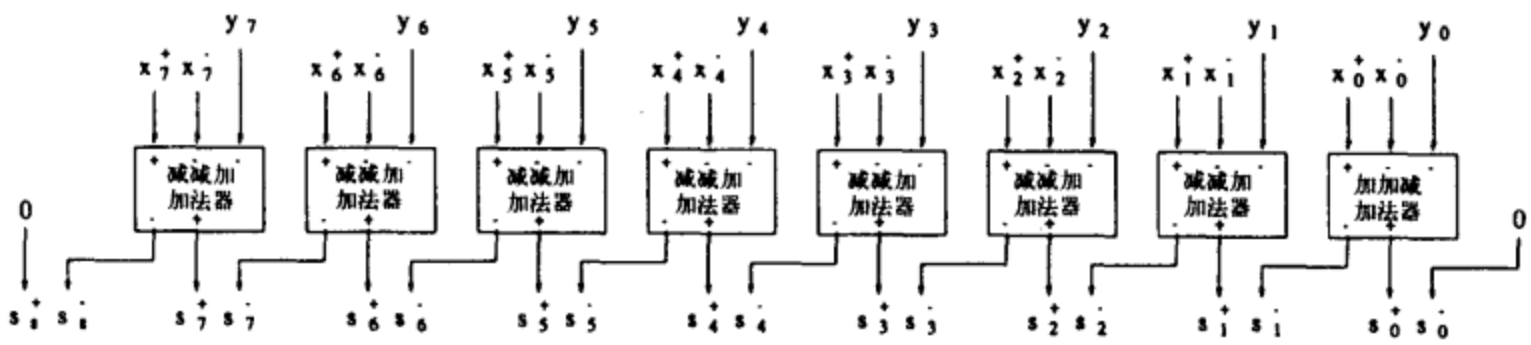


图14-3 8位混合基2减法器

14.3.3 混合基2加法减法器

混合加法器（或减法器）可以将一个有符号数位数字加上（或减去）一个无符号数字。当一个操作数用基 r 的补码表示法表示，对于基2就是2的补码表示法，混合减法可以通过混合加法完成，即被减数加上减数的2的补码并舍去最高数位上的进位输出。所以加法和减法可以用同样的硬件实现。图14-4给出了一种混合基2加法减法器的架构。

注意每个单元都含有一个二到一的多路器用来根据运算类型选择数字Y或者它的2的补码。

14.3.4 有符号二进制数位加法减法器

一个有符号数位数字表示为 $Y_{<r,\alpha>} = Y^+ - Y^-$，其中 Y^+ 和 Y^- 取自数位集合 $\{0, 1, \cdots, \alpha\}$。所以，一个有符号数位数字可以看成两个无符号普通数字相减。因此，有符号数位加法可以通过依次进行一次混合加法和一次混合减法而完成，如下所示：

$$\begin{aligned} S_{<r,\alpha>} &= X_{<r,\alpha>} + Y_{<r,\alpha>} = X_{<r,\alpha>} + Y^+ - Y^- \\ \Rightarrow & \quad S1_{<r,\alpha>} = X_{<r,\alpha>} + Y^+ \\ & \quad S_{<r,\alpha>} = S1_{<r,\alpha>} - Y^- \end{aligned} \tag{14-10}$$

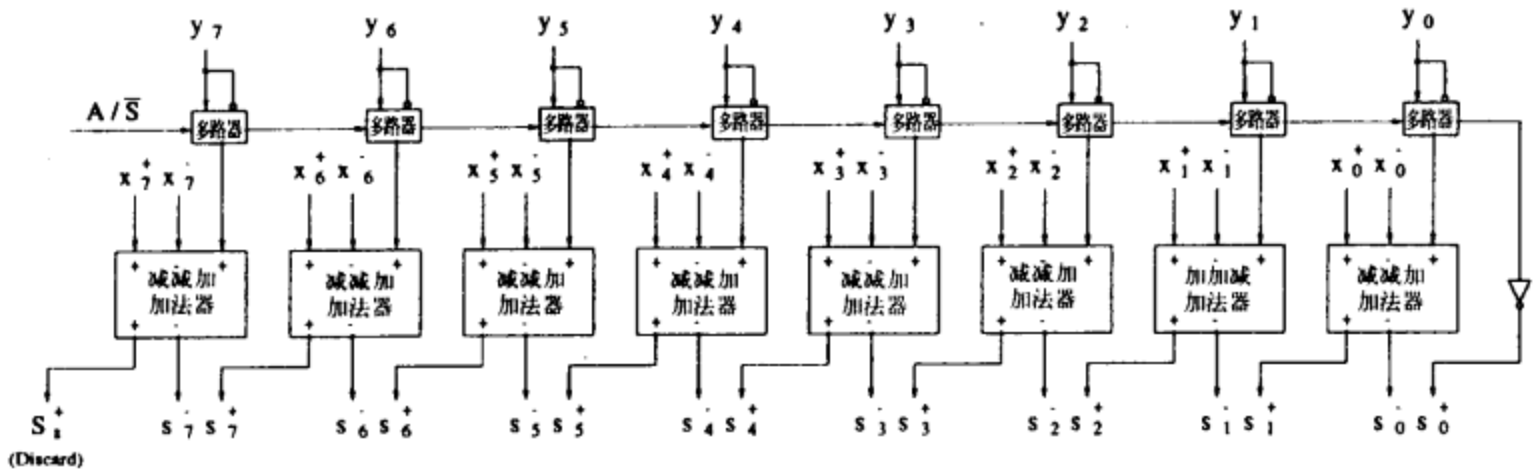
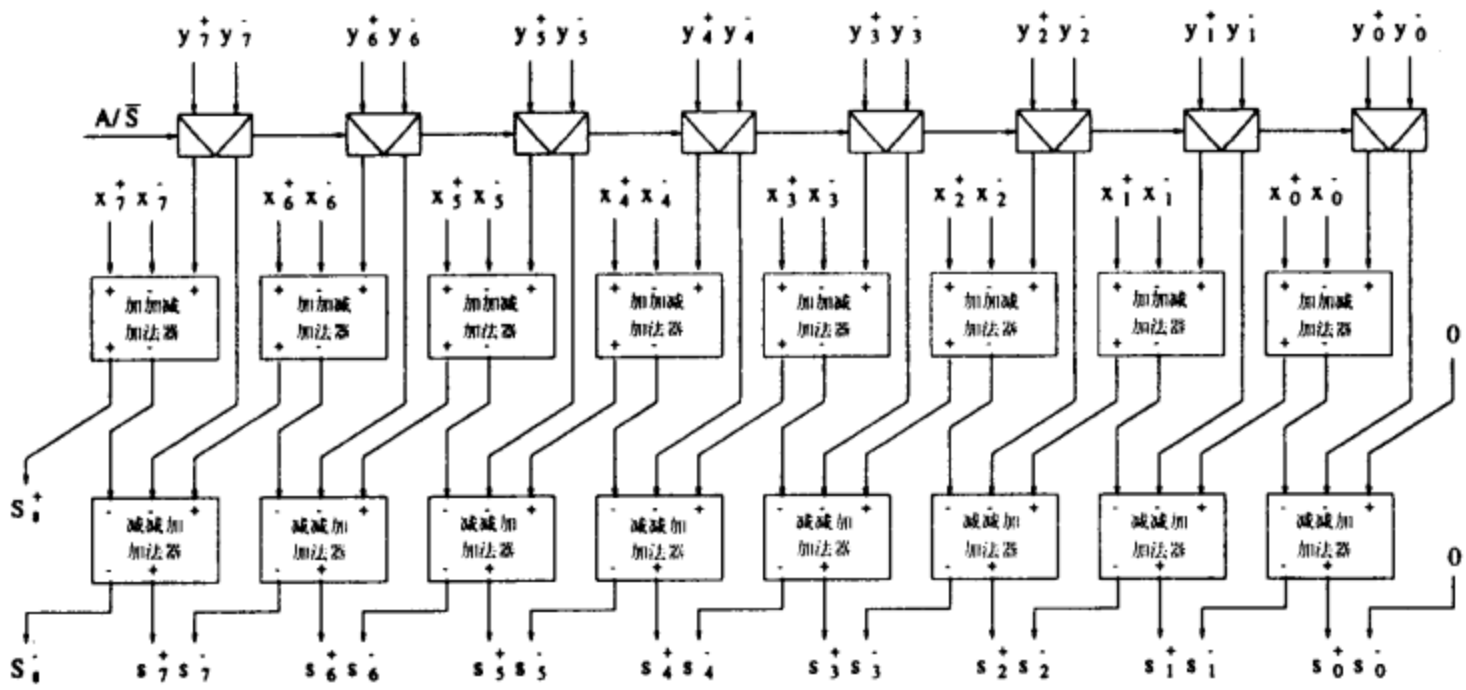
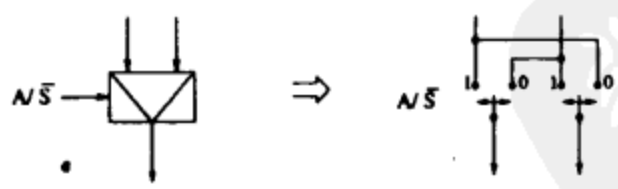


图14-4 混合基2加法减法器（$A/\overline{S}=1$ 为加法，$A/\overline{S}=0$ 为减法）

图14-5以基2的情形图解了这种方法。该图也给出了既可以完成SBD加法又可以完成SBD减法的硬件，只需在每个位上插入一个由合适的加减控制信号 $(A/\overline{S})$ 控制的二到二开关盒子[17]。这是由于 $-Y_{<2.1>}=Y^{-}-Y^{+}$，即只需交换一个SBD数字的每个有符号二进制数位码的正位和负位就可以将它变号。在这个设计中关键路径为两个加法器，对于某些应用可能太慢了。此时可以使用[18]中给出的一种高速SBD加法电路。



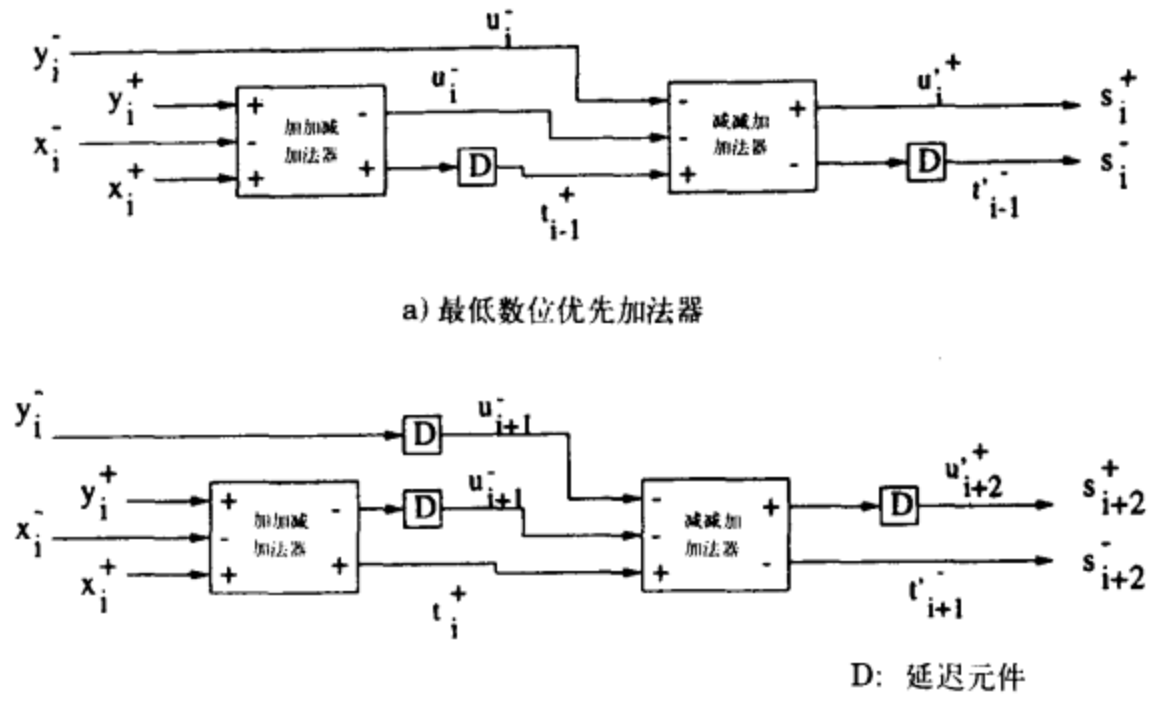
a) 有符号二进制数位加法减法器



b) 开关盒子的定义

图 14-5

数位串行SBD加法器可以通过将数位并行加法器按最低数位优先或者最高数位优先的模式折叠（见第6章）而得，它们的架构由图14-6示出。最低数位优先加法器的等待时间为零，而最高数位优先加法器的等待时间为两个时钟周期。对于8数位加法，其和有9个数位，因此需要在两个相邻的输入操作数之间插入一个零数位，从而一次加法需要9个时钟周期。



a) 最低数位优先加法器

b) 最高数位优先加法器

图14-6 数位串行SBD冗余加法器

很明显，组合的混合加法减法器比有符号二进制数位加法减法器更加高效。混合基2加法减法器的硬件代价是SBD加法减法器的一半。

14.4 混合基4加法

在运算操作的迭代或串行实现中可以引入更高阶的基以减少迭代周期数。基4因为所需的迭代步数只是基2的一半而且实现并不比基2复杂许多，因而很有吸引力。

本节考虑混合基4加法的设计。减法或加法减法的基4架构设计留给读者作为练习。基4冗余有符号数位集合可以是最大冗余（maximally redundant），即基于数位集合 $D_{<4,3>}$，也可以是最小冗余（minimally redundant），即基于数位集合 $D_{<4,2>}$。 535 ~ 537

14.4.1 最大冗余混合基4加法

混合基4加法可以按照与混合基2加法相似的方式完成。表14-4总结了最大冗余混合基4加法（MRHY4A） $S_{<4,3>}=X_{<4,3>}+Y_4$ 所用到的数位集合。表的最后一列列出了赋给数位的二进制码。注意数位 x_i、 $s_i \in D_{<4,3>}$ 以及 $y_i \in D_4$ 的二进制码对应于两个连续的基2数位的带权加法。

混合加法的第一步运算操作计算

$$x_i + y_i = 4t_i + u_i \tag{14-11}$$

将表14-4中相应的二进制码代入（14-11），我们得到：

$$\left(2x_i^{+2} - 2x_i^{-2} + 2y_i^{+2}\right) + \left(x_i^{+} - x_i^{-} + y_i^{+}\right) = 4t_i^{+} + 2u_i^{+2} - 2u_i^{-2} - u_i^{-} \tag{14-12}$$

使用最大冗余混合基4加法单元（MRHY4A）来完成上述的运算。这个单元含有两个PPM加法器，并行完成在括号内的两个加法，位数从6约减至4，它们的权重由（14-12）右端给出。图14-7给出了MRHY4A单元的结构。如前所述，MRHY4A单元就是两个连续的混合基2加法单元。

表14-4 最大冗余混合基4加法所涉及的数位集合

数 位	基4数位集合	二进制码
x_i	$\{\bar{3},\bar{2},\bar{1},0,1,2,3\}$	$2x_i^{+2}-2x_i^{-2}+x_i^{+}-x_i^{-}$
y_i	$\{0,1,2,3\}$	$2y_i^{+2}+y_i^{+}$
$p_i=x_i+y_i$	$\{\bar{3},\bar{2},\bar{1},0,1,2,3,4,5,6\}$	$\cdots$
u_i	$\{\bar{3},\bar{2},\bar{1},0,1,2\}$	$2u_i^{+2}-2u_i^{-2}-u_i^{-}$
t_i	$\{0,1\}$	t_i^{+}
$s_i=u_i+t_{i-1}$	$\{\bar{3},\bar{2},\bar{1},0,1,2,3\}$	$2s_i^{+2}-2s_i^{-2}+s_i^{+}-s_i^{-}$

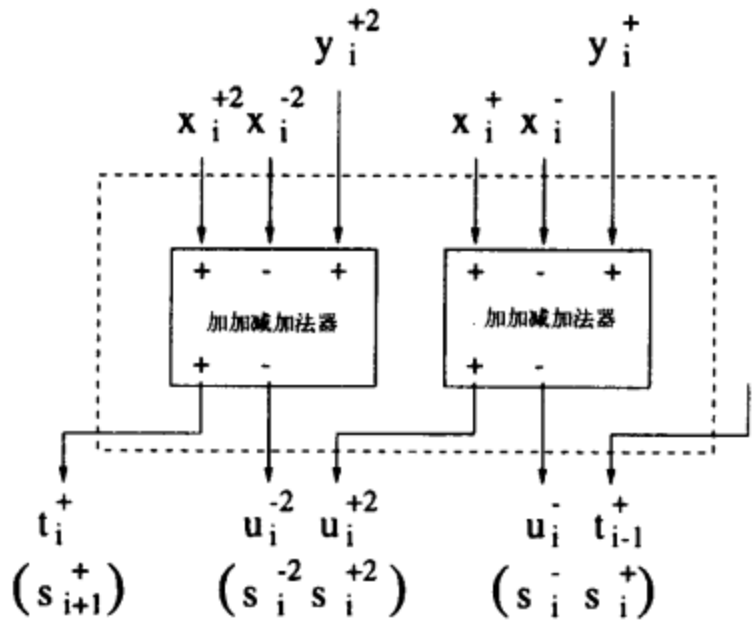


图14-7 最大冗余混合基4加法单元（MRHY4A）

混合加法算法的第二步由 $s_i=u_i+t_{i-1}$ 给定。将 s_i、u_i 和 t_{i-1} 用相应的二进制码（在表14-4中）代替，得到 $s_i^{+2}=u_i^{+2}$，$s_i^{-2}=u_i^{-2}$，$s_i^{+}=t_{i-1}^{+}$ 以及 $s_i^{-}=u_i^{-}$。

混合基4加法器由图14-8示出。从图14-8和图14-1中可以总结出W数位MRHY4A的结构与2W数位混合基2加法器完全相同。

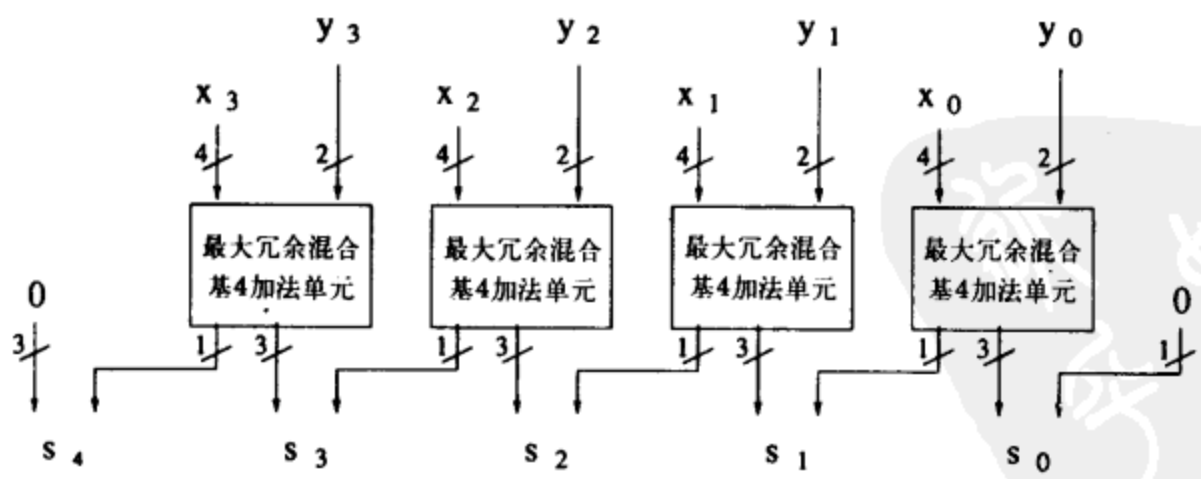


图14-8 4数位MRHY4A

14.4.2 最小冗余混合基4加法

表14-5总结了最小冗余混合基4加法（mrHY4A）$S_{<4.2>}=X_{<4.2>}+Y_4$ 所用到的数位集合。

表的最后一列列出了赋给数位的二进制码。注意数位 x_i 和 $s_i \in D_{<4,2>}$ 的编码用3位表示，即一个权重为2的负位和两个权重为1的正位。

表14-5 最小冗余混合基4加法所涉及的数位集合

数 位	基4数位集合	二进制码
x_i	$\{\bar{2},\bar{1},0,1,2\}$	$-2x_i^{-2}+x_i^{+}+x_i^{++}$
y_i	$\{0,1,2,3\}$	$2y_i^{+2}+y_i^{+}$
$p_i=x_i+y_i$	$\{\bar{2},\bar{1},0,1,2,3,4,5\}$	$\cdots$
u_i	$\{\bar{2},\bar{1},0,1\}$	$-2u_i^{-2}+u_i^{+}$
t_i	$\{0,1\}$	t_i^{+}
$s_i=u_i+t_i$	$\{\bar{2},\bar{1},0,1,2\}$	$-2s_i^{-2}+s_i^{+}+s_i^{++}$

混合加法的第一步运算操作由（14-11）给出。

将表14-5中相应的二进制码代入（14-11），我们得到：

$$\left(-2x_i^{-2}+2y_i^{+2}\right)+\left(x_i^{+}+x_i^{++}+y_i^{+}\right)=4t_i^{+}-2u_i^{-2}+u_i^{+} \tag{14-13}$$

由方程（14-13）得出的mrHY4A单元由图14-9给出，其中一个全加器（FA）将三个权重为1的位（x_i^{+}、x_i^{++}和y_i^{+}）约减为两位（u_i^{+}和中间进位c_i^{+2}），接着一个PPM加法器将三个权重为2的位（x_i^{-2}、y_i^{+2}和c_i^{+2}）约减为两个新的位（t_i^{+}和u_i^{-2}）。注意这个单元含有一个局部的进位传播。

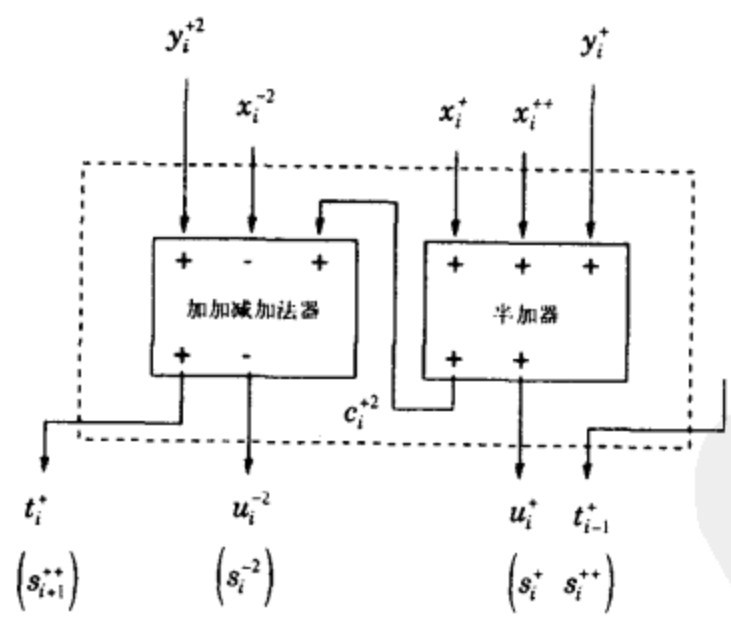


图14-9 最小冗余混合基4加法器单元（mrHY4A）

在混合加法算法的第二步中，利用 $s_i^{-2}=u_i^{-2}$，$s_i^{+}=u_i^{+}$，$s_i^{++}=t_{i-1}^{+}$ 产生和的数位。图14-10给出了4数位mrHY4A的结构。

14.5 基2混合冗余乘法架构

本节考虑基于基2混合冗余数字的串行在线乘法。这些冗余乘法器首先处理输入数据的最

高数位并产生结果的最高数位。从在线运算可以得到低等待时间架构，并且对于如除法和平方根这些本身就是最高数位优先的操作十分有用。

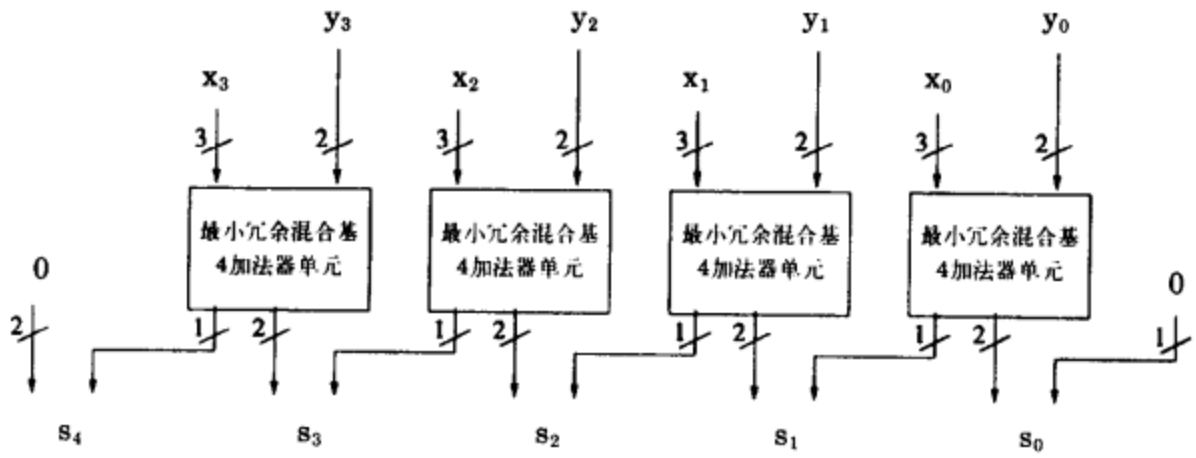


图14-10 4数位mrHY4A

考虑两个数字 $A = a_3 \cdot a_2\, a_1\, a_0$ 和 $B = b_3 \cdot b_2\, b_1\, b_0$ 的乘法，并利用图14-11和图14-12中给出的两个乘法依赖图（DG）。基本处理元件（DG中的实心圆）执行由下式定义的加法操作:

$$a_i * b_j + carry_{in} + sum_{in} = 2 * carry_{out} + sum_{out} \tag{14-14}$$

其中的虚线表示进位（carry），实线表示和（sum）。最后一行的椭圆圈是空操作，只是用来把多个位组合成数位。在图14-11的DG中，进位垂直向下移动，而在图14-12的DG中进位沿对角线移动[19]。

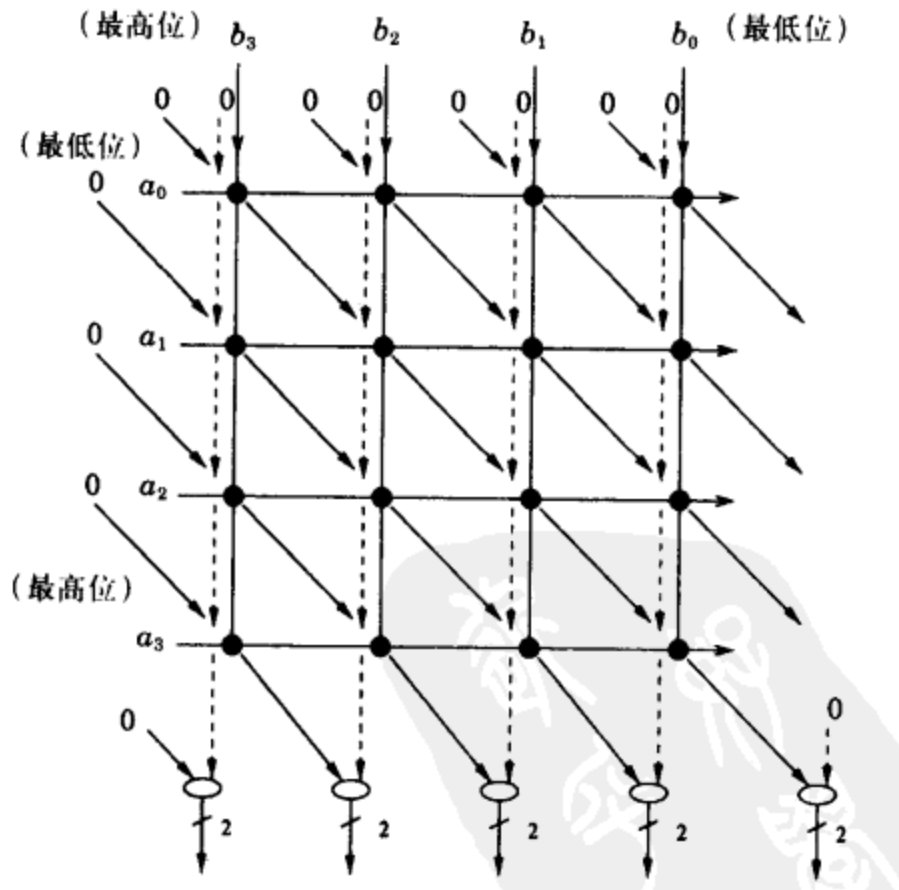


图14-11 垂直进位的乘法DG

利用脉动设计方法学（见第7章）以及下面的投影矢量、处理器空间矢量和调度矢量

$$\mathbf{d} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{p}^T = \begin{pmatrix} 0 & 1 \end{pmatrix}, \quad \mathbf{s}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{14-15}$$

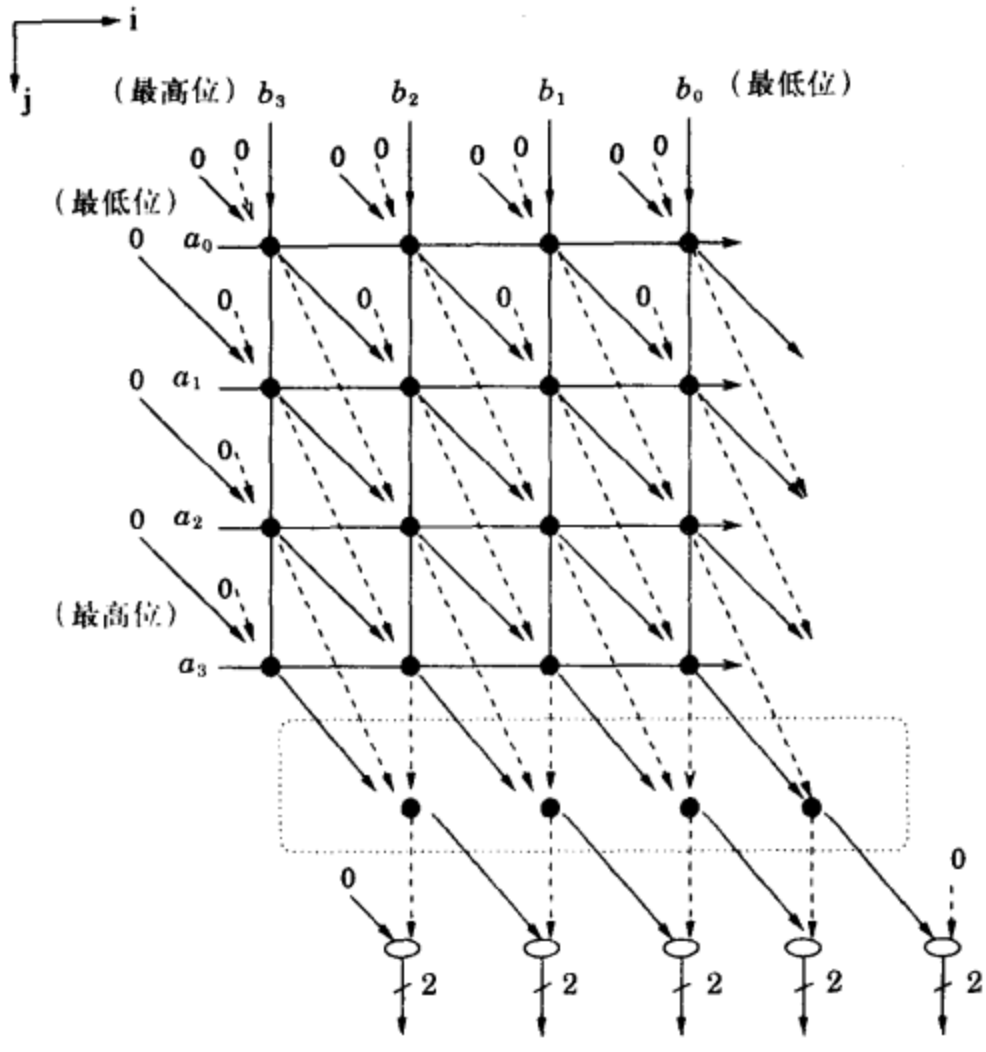


图14-12 对角线进位的乘法DG

对于图14-11中的DG，我们得到下面的边映射：

e	p^Te	s^Te
a (1, 0)	0	1
b (0, 1)	1	0
carry (0, 1)	1	0
sum (1, 1)	1	1

以及对于图14-12中的DG的边映射：

e	p^Te	s^Te
a (1, 0)	0	1
b (0, 1)	1	0
carry (1, 2)	2	1
sum (1, 1)	1	1
carry' (0, 1)	1	0

其中“*carry'*”表示最后两行的垂直进位边。

所得的位串行乘法器分别由图14-13和图14-14示出。它们可以用来完成混合乘法和有符号数位乘法，这取决于被乘数和乘数的位的编码以及（14-14）中的一数位加法操作的定义。本节考虑混合基2冗余乘法，其中被乘数B是基2冗余数字，乘数A是用2的补码表示法表示的。在

这种情况下，和以及进位信号被限制在数位集合 $\{\bar{1},0\}$ 或 $\{0, 1\}$ 中。所以，由更低位上的和以及进位这两个位组合成的数位（DG的最后一行）在集合 $\{\bar{1},0,1\}$ 中，这些数位组合在一起就形成了基2冗余表示的积。下面我们推导处理单元（加法器单元）的详细电路以得到完整的混合基2乘法器。

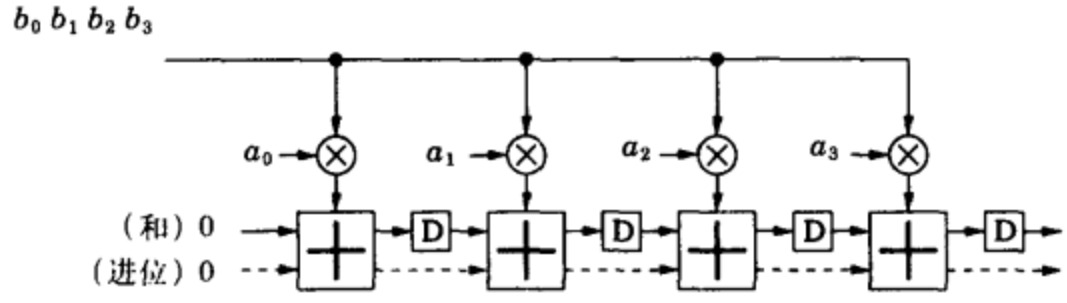


图14-13 由图14-11的DG得到的冗余乘法器架构

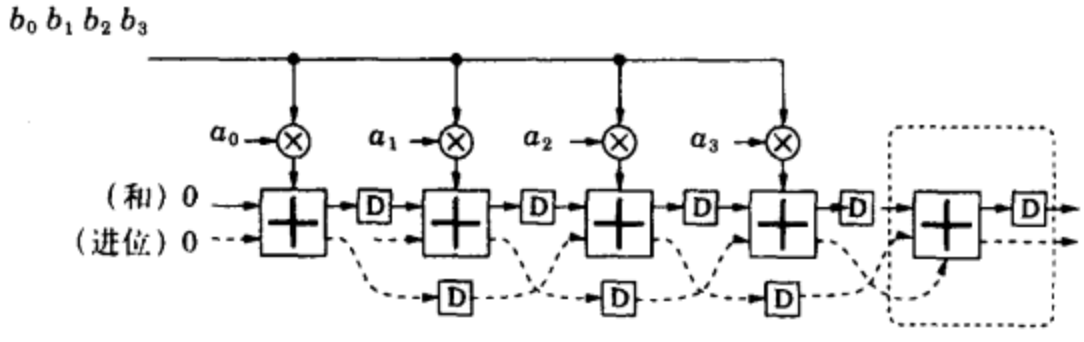


图14-14 由图14-12的DG得到的冗余乘法器架构

基2冗余数字B的每个数位 $b_j = b_j^+ - b_j^-$ 用一个符号位和一个幅度位重新编码，如下所示:

b_j	$\bar{1}$	0	1
sign (b_j)	1	0	0
$\lvert b_j \rvert$	1	0	1

如果输入位 b_j 是正的，那么图14-13或图14-14中对应于系数 a_i $(0 \leqslant i \leqslant 2)$ 的加法器单元可以用一个全加器实现；最后的加法器单元涉及A的最高位，具有负的权重符号位，执行下面的计算:

$$-a_3 * b_j + carry_{in} + sum_{in} = 2 * carry_{out} - sum_{out} \tag{14-16}$$

这个计算可以用一个由一个全加器和两个反相器组成的PPM加法器实现。如果输入位 b_j 是负的，则有

$$\begin{aligned} & -|b_j| * (-a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}) \\ = \quad & |b_j| * (-\bar{a}_3 + \bar{a}_2 2^{-1} + \bar{a}_1 2^{-2} + \bar{a}_0 2^{-3} + 2^{-3}) \\ = \quad & -|b_j|\bar{a}_3 + |b_j|\bar{a}_2 2^{-1} + |b_j|\bar{a}_1 2^{-2} + |b_j|\bar{a}_0 2^{-3} + |b_j| 2^{-3} \end{aligned} \tag{14-17}$$

因此，当 b_j 为负时A的位 a_i $(0 \leqslant i \leqslant 3)$ 必须反相。综合这两种情况，关于加法器单元我们有如下等式:

$$\begin{aligned} & (a_i \oplus sign(b_j)) * |b_j| + carry_{in} + sum_{in} \\ = \quad & 2 * carry_{out} + sum_{out} \quad 当\ 0 \leqslant i \leqslant 2 \\ & -(a_3 \oplus sign(b_j)) * |b_j| + carry_{in} + sum_{in} \\ = \quad & 2 * carry_{out} - sum_{out} \quad 当\ i = 3 \end{aligned} \tag{14-18}$$

这两个混合冗余乘法器的详细电路图由图14-15和图14-16示出。这些架构需要一个警戒数位0来隔离对后续输入数据字的处理（并在最低数位上产生0位）。另外，如图14-15和图14-16所示，当下一个数据字到达输入端时有一个复位（reset）信号用来复位锁存器。注意可以用增加一个控制电路的代价来避免警戒数位。

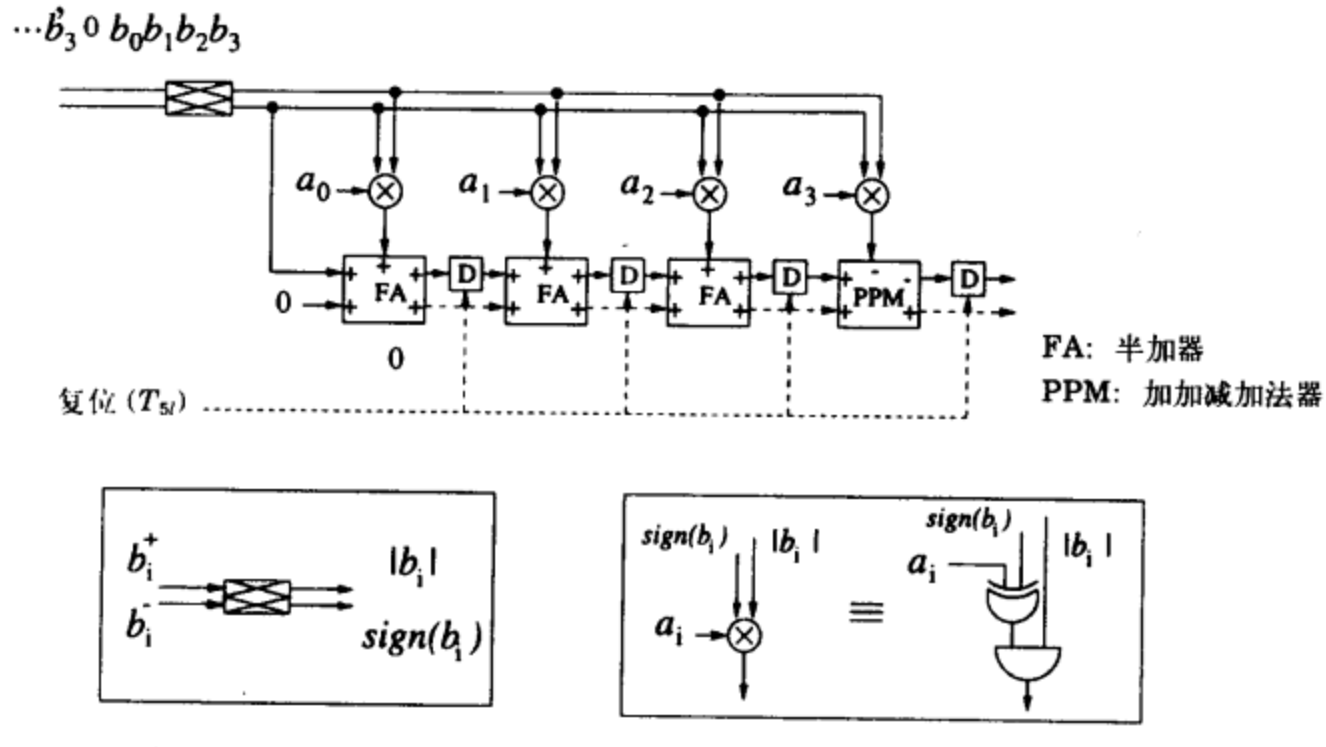


图14-15 由图14-11的DG得到的混合基2冗余乘法器架构

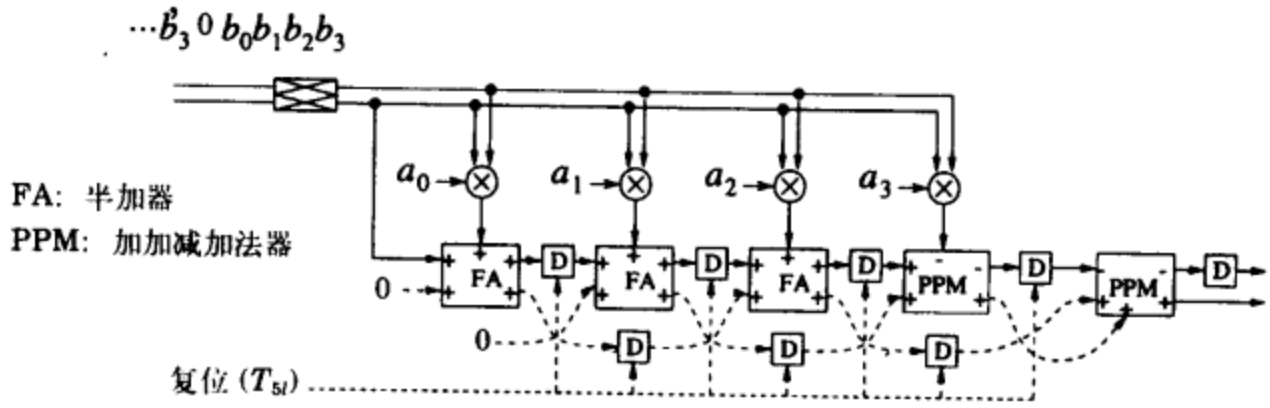


图14-16 由图14-12的DG得到的混合基2冗余乘法器架构

图14-16中的混合乘法器的关键路径为一个加法器延迟，等待时间恒定为3个时钟周期，与字长W无关。按照关键路径和计算等待时间，它是最高效的架构。另外，该乘法器每一级只需要两个延迟元件。值得注意的是位级流水线的2的补码位串行乘法器的每一级至少需要3个延迟元件。

14.6 数据格式转换

本节将给出基2和基4表示法的从非冗余到冗余和从冗余到非冗余转换。

14.6.1 非冗余到冗余转换

538 ~ 545

由于非冗余输入数位集合是冗余输入数位集合的子集，这个转换相对来说比较容易。

14.6.1.1 基2表示法

设非冗余数字X记作 $X = x_3 \cdot x_2\, x_1\, x_0$，相应的冗余表示记作 $Y = y_3 \cdot y_2\, y_1\, y_0$，其中每个数位 y_i 编码为 $y_i^+ - y_i^-$，而每个数位 x_i 表示1或0。注意在2的补码数字系统中位 x_3 的权重

为-1。这个转换由图14-17示出。

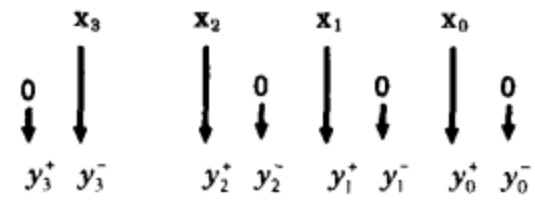


图14-17 字长为4的非冗余到冗余基2转换

14.6.1.2 基4表示法

考虑基4补码数字X到最大冗余数字Y的转换。X的数位 x_i，根据 $x_i = 2x_i^{+2} + x_i^{+}$ 用两条线编码、数位 y_i 编码为 $y_i = 2y_i^{+2} - 2y_i^{-2} + y_i^{+} - y_i^{-}$。令相同权重的信号位系数相等，我们得到图14-18中的转换器，字长为4。注意符号位 x_3 可以取值-3、-2、-1或0，编码为 $x_3 = -2x_3^{-2} - x_3^{-}$。

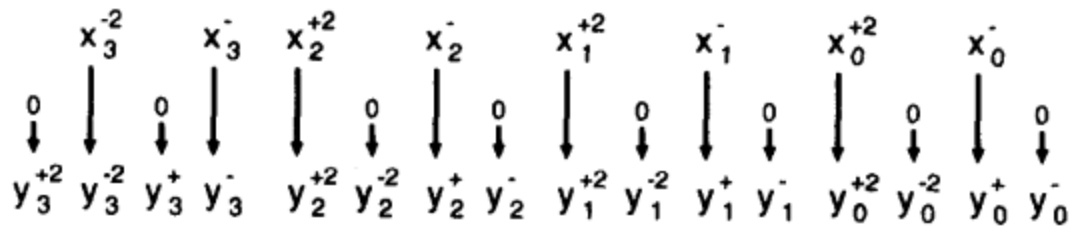


图14-18 字长为4的非冗余到最大冗余基4转换

下面考虑基4补码数字X到最小冗余数字Y的转换，其中Y的数位编码为 $-2y_i^{-2} + y_i^{+} + y_i^{++}$。这个转换并不像非冗余到最大冗余转换那么简单，因为基4补码数字的数位属于集合 {0, 1, 2, 3}，而最小冗余基4表示的数位属于集合 {-2, -1, 0, 1, 2}。所以，一个值为3的数位必须编码为一个转移数位1（值为4）和一个对应的数位-1。类似地，数位2也还原为一个转移数位1和一个对应的数位-2。一般地，如果一个基 r 数字 x 被转换成 $y_{<r,\alpha>}$，那么集合 $[\alpha, r-1]$ 中的数位将使用一个转移数位1和一个相应的数位 $x_i - r$ 编码，其中 x_i 是 x的第 i 个数位。这就等价为下面的变形

546

$$\begin{aligned} 2x_i^{+2} + x_i^{+} &= 4x_i^{+2} - 2x_i^{+2} + x_i^{+} \\ &= y_{i+1}^{++} - 2y_i^{-2} + y_i^{+} \end{aligned} \tag{14-19}$$

转换器可以由下面的式子描述:

$$y_i^{-2} = x_i^{+2}, \quad y_i^{+} = x_i^{+}, \quad y_{i+1}^{++} = x_i^{+2} \tag{14-20}$$

字长为4的转换器由图14-19示出。注意为防止转换时溢出，最高位不允许为-3。

14.7 冗余到非冗余转换器

将冗余数字转换到非冗余数字的问题比较重要且更加有趣。例如，考虑按最高数位优先传输的最大冗余基4格式，其中的数位可以在集合 {-3, -2, -1, 0, 1, 2, 3} 中取值。问题是将这种数字变换成并行非冗余数字。需要强调的是，一般而言在得知最低的冗余数位之前，无法确定任何一个非冗余数位的值。例如，考虑冗余数字 $1000\bar{1}$，它的非冗余基4格式表示是 $(0333)_4$。然而，如果将最低数位由-1改变成1，立即可以看出所得到的非冗余基4格式是 $(10001)_4$，也就是说，将最低数位由-1改变成1导致它前面（更高）的所有数位也被改变。所

以，如果一个冗余数字按最高数位优先扫描并变换成非冗余基4格式，那么在输入的最低数位被处理之前，不可能输出结果的任一数位。

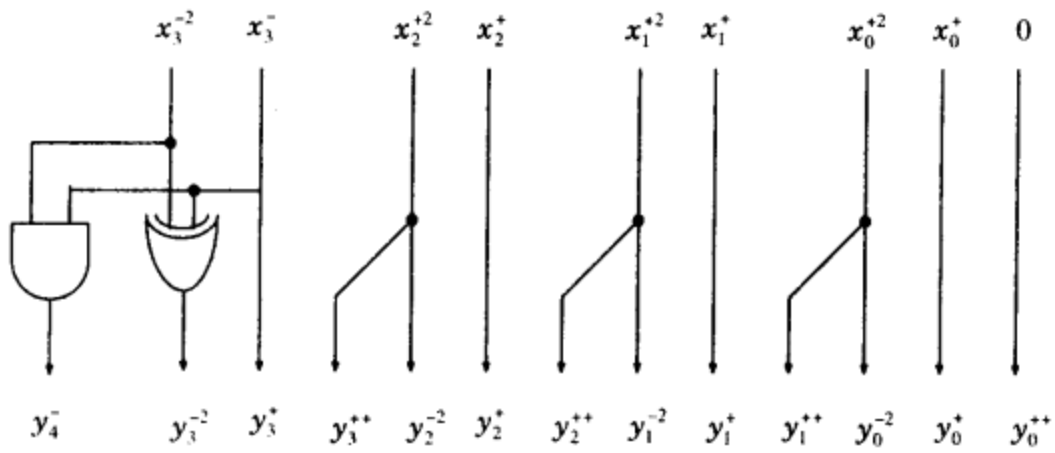


图14-19 字长为4的非冗余到最小冗余基4转换

14.7.1 最低数位优先转换

按最低数位优先模式处理的冗余到非冗余格式转换可以这样进行，将 x^+ 和 x^- 当作两个独立的无符号数字，并将 x^+ 减去 x^-，如下所示:

$$x_i^+ - x_i^- - c_i = -2c_{i+1} + s_i \tag{14-21}$$

其中在每个位上使用一个MMP加法器。

一个简单的最低数位优先冗余格式到非冗余格式转换电路由图14-20示出，字长为4位。注意任一级的进位输出可能是0或 $\bar{1}$ 。图14-20也说明了最低数位优先冗余到二进制转换和二进制加法之间的相似性。

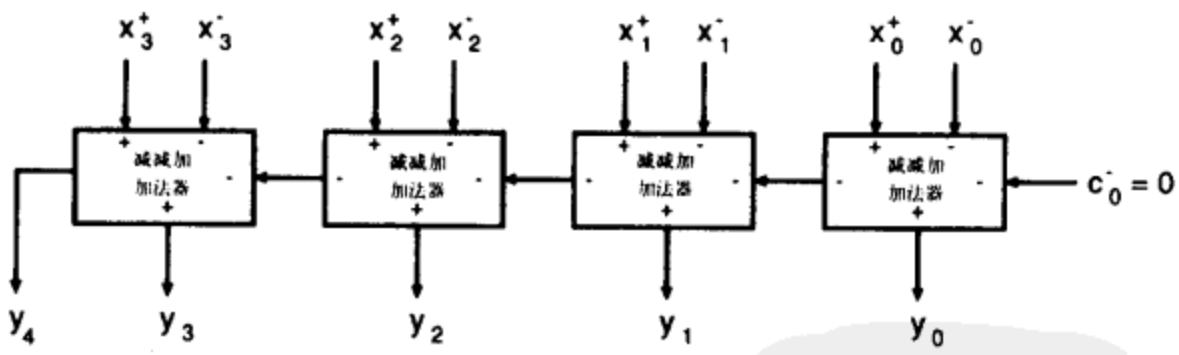


图14-20 按最低数位优先模式工作的基2冗余到非冗余格式转换器

14.7.2 最高数位优先转换

最高数位优先转换是在线运算的根本，在线运算中数位按最高数位优先的方式一次一位串行处理。本节中描述的转换假设输入数位是基2冗余格式。本技术可以被简单地扩展到任意基输入数位。

既然转换器按最高数位优先模式工作，每个数位的进位信号不能推理（a priori）得知。所以，假设进位输入为-1和0分别计算两个可能的输出数位，而最终对应于进位输入0的输出被选作输出结果。表14-6给出了如何根据一个给定的输入数位和进位输入的两个可能值计算输出（注意在方框中的高位是进位输出）。

547 ~ 548

表14-6 基2冗余数字到非冗余（2的补码）数字的转换映射

进位输入	$\bar{1}$	0	1
0	$\boxed{\bar{1}}$ 1	$\boxed{0}$ 0	$\boxed{0}$ 1
$\bar{1}$	$\boxed{\bar{1}}$ 0	$\boxed{\bar{1}}$ 1	$\boxed{0}$ 0

在转换中，假设进位输入等于0和$\bar{1}$分别迭代计算两个部分结果X_i^0和$X_i^{\bar{1}}$（对于$-1 < i < W-1$），初始值为$X_{-1}^0=0$，$X_{-1}^{\bar{1}}=1$。最终结果为X_{W-1}^0。在转换过程中的第i步，根据表14-6中的规则选择X_{i-1}^0或$X_{i-1}^{\bar{1}}$并添加0或1，从而得到X_i^0和$X_i^{\bar{1}}$。这个过程总结在表14-7中，其中*minus*和*plus*当*plus*=1时被用来选择X_{i-1}^0，当*minus*=1时被用来选择$X_{i-1}^{\bar{1}}$。如果输入数位是$\bar{1}$则*minus*是1，如果输入数位是1则*plus*是1。如果输入数位是$\bar{1}$，表14-6的第二列意味着无论进位输入是什么进位输出总等于$\bar{1}$。因此，*minus*置为1且选择$X_{i-1}^{\bar{1}}$。在$X_{i-1}^{\bar{1}}$之后添加位1和0，分别形成X_i^0和$X_i^{\bar{1}}$，对应于进位输入0和$\bar{1}$。如果输入数位是1，表14-6的第四列意味着进位输出总等于0。因此，*plus*置为1且选择X_{i-1}^0。在X_{i-1}^0之后添加位1和0，分别形成X_i^0和$X_i^{\bar{1}}$，对应于进位输入0和$\bar{1}$。对于输入信号为0的情形，表14-6的第三列意味着可以通过在X_{i-1}^0和$X_{i-1}^{\bar{1}}$之后添加0和1，分别得到X_i^0和$X_i^{\bar{1}}$。

表14-7 按最高数位优先模式工作的基2冗余到非冗余数字的转换中输出数位的计算

输入数位	X_i^0	$X_i^{\bar{1}}$		
$\bar{1}$	$X_{i-1}^{\bar{1}}+1\cdot 2^{-i}$	$X_{i-1}^{\bar{1}}+0\cdot 2^{-i}$	1	0
0	$X_{i-1}^0+0\cdot 2^{-i}$	$X_{i-1}^{\bar{1}}+1\cdot 2^{-i}$	0	0
1	$X_{i-1}^0+1\cdot 2^{-i}$	$X_{i-1}^0+0\cdot 2^{-i}$	0	1

549

例14.7.1 利用表14-7，基2数字$1\bar{1}0\bar{1}$可以转换成非冗余表示，转换的要点如表14-8所示。■

表14-8 说明基2冗余数字到非冗余数字的转换的例子

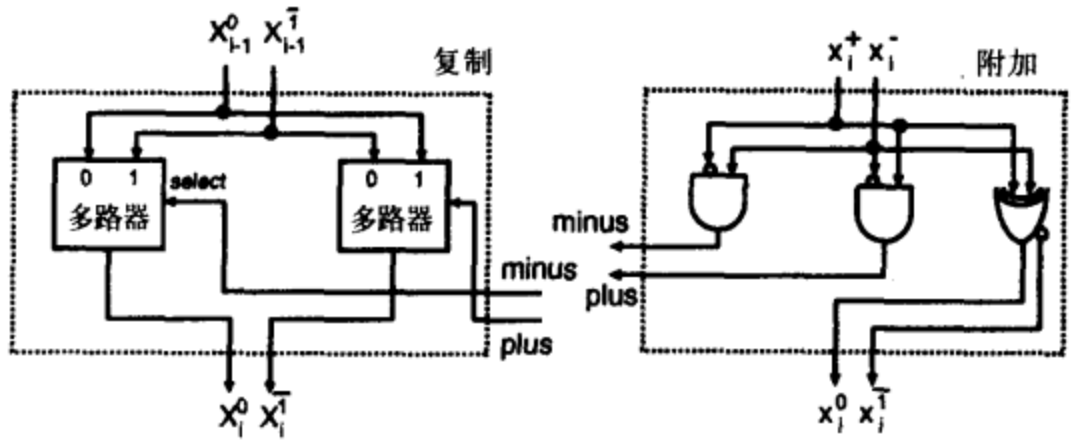
k	输入数位	X_i^0	$X_i^{\bar{1}}$
-	-	0	$\bar{1}$
0	1	0 1	0 0
1	$\bar{1}$	0 0 1	0 0 0
2	0	0 0 1 0	0 0 0 1
3	$\bar{1}$	0 0 0 1 1	0 0 0 1 0
输出 $= X_3^0 = 0\,0\,0\,1\,1 = 3_{10}$			

从例14.7.1和表14-8中可以清楚地看出一个W数位基2冗余数字被转换到$W+1$数位2的补码数字的过程。

为了设计转换器，需要两个基本单元，*COPY*单元和*APPEND*单元[20]。*APPEND*单元根据当前输入的有符号数位的值，产生选择信号*minus*和*plus*，以及两个基于输入数位和进位输入信号的两个可能值的输出位。*COPY*单元根据*minus*和*plus*选择信号的值，选择X_{i-1}^0或$X_{i-1}^{\bar{1}}$。*COPY*和*APPEND*单元的逻辑式可以根据表14-7推导出来，如下所示:

$$
\begin{aligned}
minus &= \overline{x_i^+}\,x_i^- \\
plus &= x_i^+\,\overline{x_i^-} \\
x_i^{\bar{1}} &= \overline{x_i^+ \oplus x_i^-} \\
x_i^0 &= x_i^+ \oplus x_i^-
\end{aligned}
\tag{14-22}
$$

其中 $x_i^{\bar{1}}$ 表示进位输入为 $\bar{1}$ 的输出位（用来更新 $X_i^{\bar{1}}$），x_i^0 表示进位输入为0的输出位（用来更新 X_i^0）。基于这些式子设计*COPY*和*APPEND*单元，分别如图14-21中的(a)和(b)所示。



a) *COPY*单元　　b) *APPEND*单元

图14-21　用于基2冗余数字到非冗余数字的转换的COPY单元和APPEND单元

字长为4数位的按最高数位优先模式工作的基2冗余到非冗余转换器的DG由图14-22（实线部分）示出。为了利用脉动映射（见第7章）获得数位串行最高数位优先转换器，将实线部分的三角形DG扩展为平行四边形（添加上用虚线表示的空单元）。用虚线表示的空单元是退化的*COPY*单元，它只是简单地将输入传递到下一行的输出。使用投影矢量、处理器空间矢

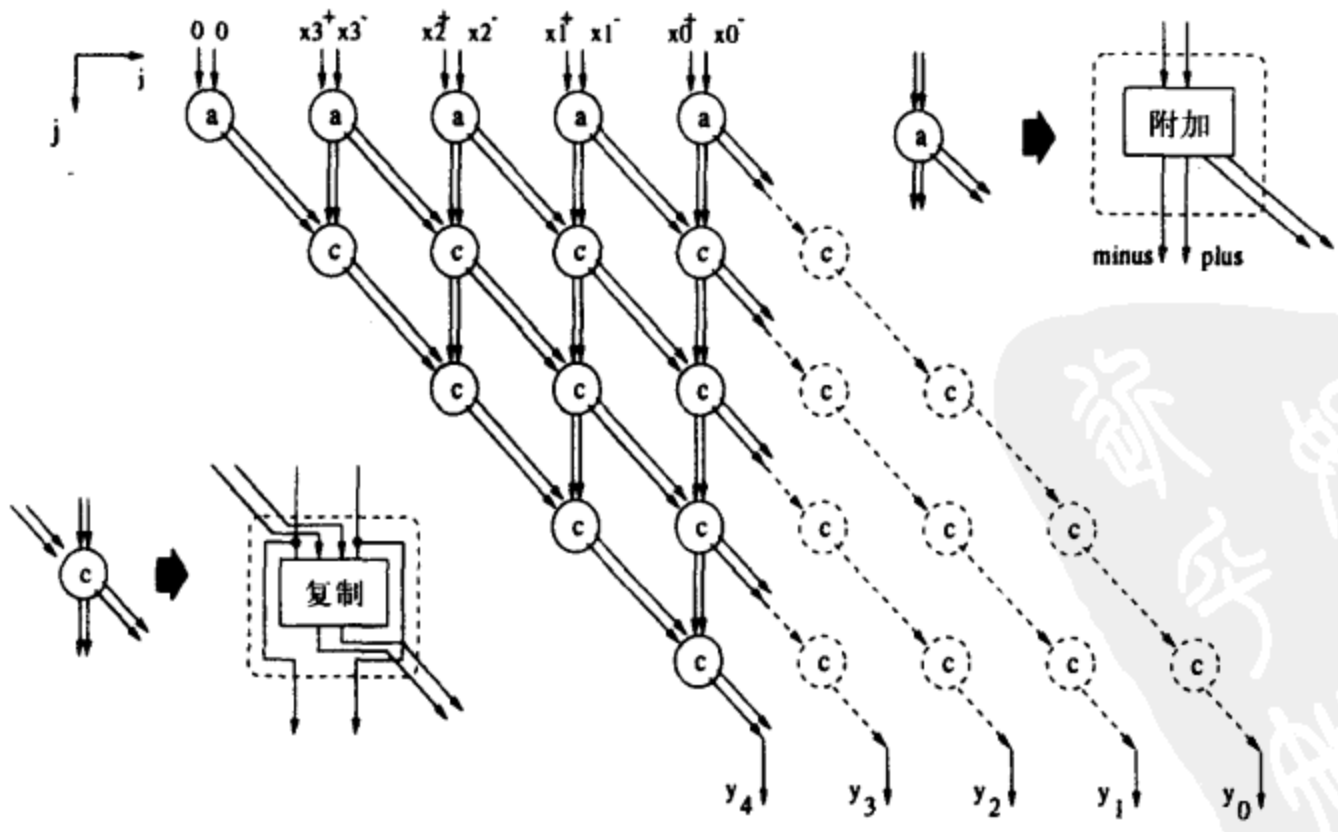


图14-22　字长为4数位的按最低数位优先模式工作的基2冗余到非冗余转换器的依赖图

量和调度矢量

$$\mathbf{d}=\begin{pmatrix}1\\0\end{pmatrix},\quad \mathbf{p}^T=\begin{pmatrix}0 & 1\end{pmatrix},\quad \mathbf{s}^T=\begin{pmatrix}1 & 0\end{pmatrix}$$

可以推导出串行最高数位优先转换器。其方框图由图14-23示出，详细设计由图14-24示出。图14-24中的复位信号在时钟周期 $T=5l$ 时置为1（否则为0），以使得上下两行寄存器的内容相等，均为2的补码数字 X_3^0 的各个位。这就保证了在随后的周期内输出 X_3^0 的各个位，而与随后的*minus*和*plus*值无关。

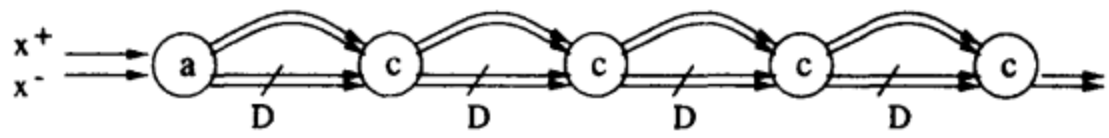


图14-23 最高数位优先冗余到非冗余转换器的方框图

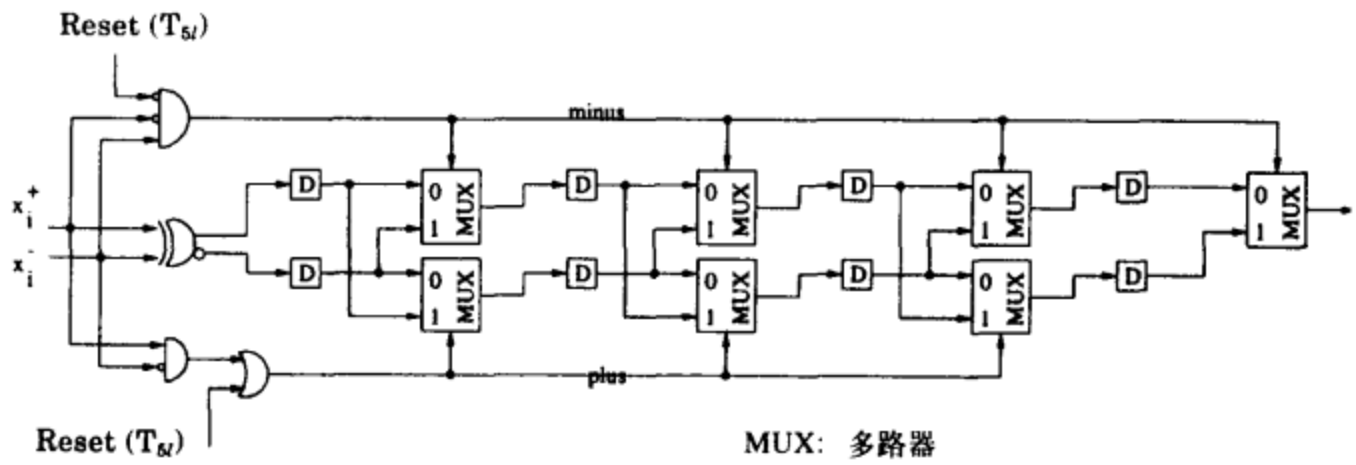


图14-24 最高数位优先冗余到非冗余转换器

14.8 结论

本章给出了冗余数字系统和冗余运算的基本概念，从而得到了无进位加法的特性。本章还研究了混合基2和基4加法减法器，以及有符号数位加法减法器的算法和架构。设计了两个最高数位优先串行混合基2冗余乘法器，该乘法器适合于在线计算。最后给出了数据格式转换器的设计，包括基2的非冗余到冗余转换，以及最低数位优先和最高数位优先模式的冗余到非冗余转换。利用已知的快速进位产生方案的变种，比如进位选择和/或二叉树结构（见附录E），可以设计快速冗余到非冗余转换器。感兴趣的读者可以参考[21]，以获得快速的低功耗冗余到非冗余转换器设计的详细描述以及它们在完成快速二进制加法上的应用。

14.9 习题

1. 考虑进位保留无符号数字X和普通无符号数字Y的混合加法。与第14.3.1节中讨论的混合基2冗余加法类似，加法的第一步是计算

$$s_i = x_i + y_i = 2t_i + u_i \tag{14-23}$$

(a) 令 $x_i = x_i^+ + x_i^{++}$，$s_i = s_i^+ + s_i^{++}$。设计与图14-1类似的字长为8的并行加法器。说明 t_i 和 u_i 的编码。

(b) 设计相应的最高数位优先和最低数位优先串行架构。

2. 图14-5中的加法器计算 $S_{<2,1>}=(X_{<2,1>}+Y^{+})-Y^{-}$。重新设计另一种利用 $S_{<2,1>}=(X_{<2,1>}-Y^{-})+Y^{+}$ 的冗余加法运算架构，并且实现类型为
   (a) 数位并行，
   (b) 最低数位优先数位串行，或
   (c) 最高数位优先数位串行。
3. 设计两个进位保留数字加法的最高数位优先和最低数位优先串行架构。假设进位保留数位 x_i 编码为两位，x_i^{+} 和 x_i^{++}，且 $x_i=x_i^{+}+x_i^{++}$。
4. 设计关键路径为3个全加器的基4最小冗余加法器，假设基4数位 x_i 用3条线编码为 $-2x_i^{-2}+x_i^{+}+x_i^{++}$。所有输入和输出操作数均用最小冗余表示法表示。只使用4个全加器和一些延迟元件，给出相应的最高数位优先和最低数位优先架构。(提示：对于第3级加法器，将 $-a$ 替换为 $-2a+a$。)
5. 设计关键路径为两个全加器的基4最大冗余加法器，假设基4数位 x_i 用4条线表示为 $2x_i^{+2}-2x_i^{-2}+x_i^{+}-x_i^{-}$。只使用4个全加器和一些延迟元件，给出相应的最高数位优先和最低数位优先架构。
6. 写出图14-14所示的混合最高数位优先位串行基2冗余乘法器的位级模拟器，并验证其功能。至少模拟10次乘法运算。假设字长为4。证明为使功能正确必须在输入每个字的最高位之前先输入一个0位。 550~553
7. 设计按最低数位优先模式工作的字长为8数位的基4最小冗余到非冗余转换器。
8. 设计按最低数位优先模式工作的字长为8数位的基4最大冗余到非冗余转换器。
9. 设计字长为8数位的最高数位优先基4最小冗余到非冗余转换器。
10. 设计字长为8数位的最高数位优先基4最大冗余到非冗余转换器。
11. 更快更小的基2冗余到二进制转换器电路可以通过将 $x=x+x^{-}$ 改写为 $x_r^{+}x_r^{-}$ 而设计出来，其中值0表示为00，1和-1分别表示为10和01。在 $(x_r^{+}x_r^{-})$ 表示中，重编码电路需将11改写为00。那么就可以利用11这个无关（don't-care）情形设计更加简单的转换器。
   (a) 说明一般的 $x^{+}x^{-}$ 表示可以改写为 $x_r^{+}x_r^{-}$，其中

$$x_r^{+}=x^{+}\overline{x^{-}},\quad x_r^{-}=\overline{x^{+}}x^{-}$$

   且 $(x_r^{+}x_r^{-})$ 不可能为(11)。
   (b) 只利用 $(x_r^{+}x_r^{-})$ 编码和二输入多路器，可以推导出快速最低数位优先转换器。考虑使图14-20中的每个MMP加法器均利用这种编码，由图14-25示出。证明进位与和输出可以按下面的式子计算

$$c_{out}^{-}=c_{in}^{-}\,\overline{x_r^{+}}+\overline{c_{in}^{-}}\,x_r^{-} \tag{14-24}$$

$$s_{out}^{+}=c_{in}^{-}\,\overline{x_r^{+}}\,\overline{x_r^{-}}+\overline{c_{in}^{-}}\,(x_r^{+}+x_r^{-}) \tag{14-25}$$

   使用两个多路器实现该MMP加法器。如果需要的话可以使用反相器和OR门。
12. 冗余数字的符号由最高的非零数位的符号确定。考虑如习题11中的将(11)改写为(00)的冗余表示法。(本题中的 x^{+} 和 x^{-} 与习题11中 x_r^{+} 和 x_r^{-} 相同。) 在这种表示法

中，x_i^-表示第i个数位的符号，即如果第i个数位为−1则x_i^-为1，如果第i个数位非负，则x_i^-为0。通过检查进位输入为0时冗余到二进制转换器的c_{out}^-的最高位，可以检测冗余数字的符号。s_{out}^+不必计算。

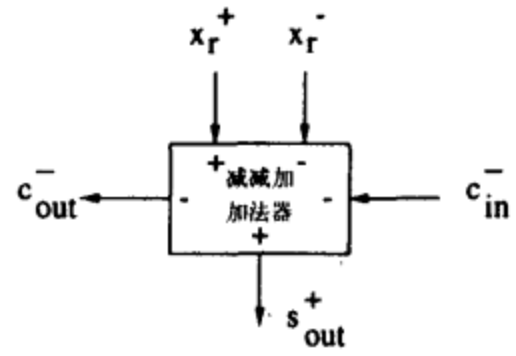


图14-25 习题11的MMP加法器

(a) 设计只使用$(W-1)$个多路器的字长为W的符号检测电路。该电路的等待时间是$(W-1)t_{mux}$，其中t_{mux}是一个多路器的等待时间。

(b) 通过利用二叉树和超前进位思想，设计等待时间为$\log_2 W$的快速符号检测电路。假设W是2的幂。你的设计应该需要$2(W-1)-\log_2 W$个多路器。设计$W=16$的转换器。(提示：将(14-24)改写为$c_{i+1}^- = c_i^- \overline{x_i^+} + \overline{c_i^-} x_i^-$，将该方程迭代一次可以确定2数位数字的符号，迭代两次可以确定4数位数字的符号，等等。)

13. 本题考虑设计基于图14-24中所示最高数位优先冗余到二进制转换器的数位串行最高数位优先2的补码二进制加法器。利用下面的关系式，二进制加法可以用冗余到二进制转换来表示

$$X = A + B = A - (-B) = A - (\overline{B} + 1_{lsb}) = (A - \overline{B}) + (-1)_{lsb}$$

其中$\overline{B}$表示B的一的补码。$(A-\overline{B})$可以认为是一个冗余数字，其中的各个数位$(a_i, \overline{b}_i)$的值为$(a_i - \overline{b}_i)$。所以，最高数位优先加法可以视作将$(A-\overline{B})$转换到一个二进制数字，并保留与进位输入−1对应的最终结果而设计出来。设计字长为8的2的补码数位串行最高数位优先加法器。你的设计的详细程度应该类似于图14-24的详细程度。

参考文献

1. A. Avizienis, "Signed digit number representation for fast parallel arithmetic," *IRE Trans. on Electronic Computers*, vol. EC-10, pp. 389–400, Sept. 1961.

554 ~ 555

2. S. C. Knowles, J. G. McWhirter, R. F. Woods, and J. V. McCanny, "Bit-level systolic architectures for high performance IIR filtering," *Journal of VLSI Signal Processing*, pp. 297–312, 1989.

3. D. S. Phatak and I. Koren, "Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains," *IEEE Trans. on Computers*, vol. 43, no. 8, pp. 880–891, Aug. 1994.

4. A. Guyot, Y. Herreros, and J. Muller, "JANUS, an on-line multiplier/divider for manipulating large numbers," in *Proc. of 9th Symposium on Computer Arithmetic*, pp. 106–111, 1989.

5. N. Takagi, H. Yassura, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Trans. on Computers*, vol. C-34, no. 9, pp. 789–796, Sept. 1985.

6. J. Duprat, Y. Herreros, and J. Muller, "Some results about on-line computation of functions," in *Proc. of 9th Symposium on Computer Arithmetic*, pp. 112–118, 1989.

7. A. Skaf and A. Guyot, "VLSI design of on-line add/multiply algorithms," in *Proc. of ICCD'93*, (Cambridge, Massachusetts), pp. 264–267, Oct. 1993.

8. K. S. Trivedi and M. D. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. on Computers*, vol. C-26, no. 7, pp. 681–687, July 1977.

9. M. J. Irwin and R. M. Owens, "Design issues in digit serial signal processors," in *1989 IEEE International Symposium on Circuits and Systems*, (Portland, OR), pp. 441–444, 1989.

10. M. D. Ercegovac and T. Lang, *Division and Square Root*. Kluwer, 1994.

11. P. Montuschi and L. Ciminiera, "Over-redundant digit sets and the design of digit-by-digit division units," *IEEE Trans. on Computers*, vol. 43, no. 3, pp. 269–279, March 1994.

12. C. Y. Chow and J. E. Robertson, "Logical design of a redundant binary adder," in *Proc. of Symposium on Computer Arithmetic*, pp. 109–115, 1978.

13. H. R. Srinivas and K. K. Parhi, "High-speed VLSI arithmetic processor architectures using hybrid number representation," *Journal of VLSI Signal Processing*, vol. 4, pp. 177–198, April 1992.

14. A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P. Jespers, "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor," *IEEE Journal of Solid State Circuits*, vol. 25, no. 3, pp. 748–756, June 1990.

15. J. M. Muller, *Arithmétiques des Ordinateurs: Opérateurs et fonctions élémentaires*. Masson, 1989.

16. K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*. Wiley, 1979.

17. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.

18. H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko, "A 8.8-ns 54 × 54 multiplier with high speed redundant binary architecture," *IEEE Journal of Solid State Circuits*, no. 6, pp. 773–783, June 1996. 556

19. G. Privat, "A novel class of serial-parallel redundant signed-digit multipliers," in *1990 IEEE International Symposium on Circuits and Systems*, (New Orleans, LA), pp. 2116–2119, May 1990.

20. L. Montalvo and A. Guyot, "Combinational digit-set converters for hybrid radix-4 arithmetic," in *Proc. of IEEE International Conference on Computer Design ICCD'94*, (Cambridge, MA), pp. 498–503, Oct. 1994.

21. K. K. Parhi, "Fast low-power VLSI binary addition," in *Proc. of 1997 IEEE Int. Conf. on Computer Design (ICCD)*, (Austin, Texas), pp. 676–684, Oct. 1997. 557

第15章 数字强度缩减

15.1 引言

算法级的强度缩减可以用来减少加法和乘法的次数（见第9章）。本章描述了旨在减少DSP计算强度的数字变换技术。这类变换，通过用子表达式消除（subexpression elimination）（也称子结构共享（substructure sharing））来重组计算，从而使按照速度、功耗和面积评价的计算性能得到改善。15.2节给出了子表达式消除的基本概念。15.3节描述了子表达式消除如何应用到多常数乘法问题，然后把多常数乘法扩展到诸如线性变换和多项式求值的特殊计算上。15.4节说明了子表达式消除如何应用到数字滤波器。15.5节讨论了加性数字的分割在线性变换上的应用，其中包含了基于行和基于列的分割的详细讨论。这一节中还讨论了乘性数字的分割，并且显示了这种技术如何应用于线性变换，减少硬件实现的功耗或面积。总的来说，强度缩减减少了电路总电容，从而减少了功率损耗。

15.2 子表达式消除

常数乘法能够通过使用专用移位-加乘法器有效地实现。子表达式消除[1]是常数乘法的一种数字变换，它使得面积、功率和速度各指标能够高效硬件实现。子表达式消除应用于一组具有相同乘数的常数乘法，它仅能应用到对一个公共变量操作的常数乘法上。本质上说，子表达式消除是检查常数乘法中移位-加的实现、同时寻找冗余操作的过程。一旦找到这种冗余操作，只需进行一次计算，得到的结果便可在整个常数乘法中共享。

例15.2.1 子表达式消除

进行$a \times x$和$b \times x$这两个操作，其中$a=13$、$b=27$。在二进制表达中，$a=001101$，$b=011011$。检查比较a和b的非零数可以看出，从右边起的第一位和第四位相同，都是1。则可以将乘法$a \times x$和$b \times x$写成如下稍微不同但等价的形式：

$a \times x = 000100 \times x + 001001 \times x$；

$b \times x = 010010 \times x + 001001 \times x = (001001 \times x) << 1 + 001001 \times x$。

显而易见，项$001001 \times x$在$a \times x$和$b \times x$的计算中都要用到。因此，该项只需计算一次。如果在这个特殊的例子中不采用子表达式消除，a的实现需要2次移位和2次乘法，b的实现则需要3次移位和3次加法；而应用子表达式消除，a和b的实现总共只需要3次移位和3次加法。注意到$b \times x=2(a \times x)+x$，再利用得到的结果计算$b \times x$，这样也是需要3次移位和3次加法。■

虽然这是一个相当简单的例子，但已经清楚地说明了子表达式消除变换的基本原理。子表达式消除能以多种格式实现，并且可以应用到大多数使用常数乘法的设计中。本章其余各节将讨论各种子表达式消除形式中的部分形式。

15.3 多常数乘法

子表达式消除可以应用于一组与一个公共变量相乘的常数乘法器。多常数乘法问题

(MCM) [1]决定如何将子表达式消除应用到一组常数乘法器使所需要的移位和加法次数最少。解决这类问题的一个一般框架和算法已经在[1]中提出。这种算法最有吸引力的特点之一就是它针对各种类型的多常数乘法（MCM）问题的通用性和适应性。这种算法使用了一个包含5个基本步骤的迭代匹配（iterative matching）处理。

步骤1：将常数乘法组中的每个常数都表达为二进制格式（例如符号型、无符号型或者二进制补码表示）；

步骤2：确定常数乘法组中所有常数之间按位匹配（非零位）的数；

步骤3：选择最优匹配；

步骤4：消除最优匹配对应的冗余部分，返回剩余的结果以及系数组的冗余；

步骤5：重复以上2至4步，直至再无改进。

虽然还有一些附加的处理步骤可以减少算法的CPU运行时间，这五个步骤却概括了解决MCM问题所需要的基本处理步骤。本节将考虑MCM问题的几个应用，在本节的例题中假定采用无符号型二进制表示。

表15-1 例15.3.1中常数的二进制代码

常数	值	无符号型
a	237	11101101
b	182	10110110
c	93	01011101

例15.3.1 迭代匹配

我们要实现 a、b、c 的常数乘法器，其中 $a=237$，$b=182$，$c=93$。它们分别对应的无符号型二进制表示参见表15-1。检验表15-1可以看出，三个系数之间的匹配字数如下：a和b=>3个匹配位；a和c=>4个匹配位；b和c=>2个匹配位。选定a和c的匹配数，并从a和c中减去这个冗余，再将a和c的余值放回到原常数组。表15-2显示出了第一轮迭代后得到的常数组。重新检验这些常数，得到新的按位匹配：b和a的余值=>2位；b和c的余值=>1位；b和a及c的冗余=>1位。再次选择最优匹配并更新常数组。两个常数之间只有一位的匹配处理不能减少计算实现所需要的移位和加法的次数。由第二轮匹配算法迭代得到的系数组见表15-3。至此，应当清楚，不会有进一步的匹配操作了。而经过上面迭代匹配算法处理得到的a、b、c实现结果如下：a=[01001101+10100000]；b=[00010110+10100000]；c=[01001101+00010000]。项01001101和10100000分别为匹配算法在第一轮和第二轮迭代处理时消去的两个冗余子表达式，这些项只需要计算一次，然后在整个常数乘法器中共享。相对于常规操作共需要14次移位和13次加法，采用子表达式消除处理，只需要9次移位和9次加法即可。 ■

560 ~ 561

表15-2 例15.3.1中经过第一轮迭代处理后更新的常数组

常 数	无符号型
a 的余值	10100000
b	10110110
c 的余值	00010000
a、c 的冗余	01001101

表15-3 例15.3.1中经过第二轮迭代处理后更新的常数组

常 数	无符号型
a 的余值	00000000
b 的余值	00010110
c 的余值	00010000
a、c的冗余	01001101
a 的余值、b 的冗余	10100000

通过这个例题可见，在MCM问题上应用子表达式消除，可以显著减少实现常数乘法的硬件消耗。在常数的字长变大或者常数的个数增多时，这种硬件节约的效果会更加显著。

迭代匹配算法除了已经应用在多常数乘法上之外，还可以扩展应用到其他方面，例如线性变换和多项式求值。

15.3.1 线性变换

子表达式消除还可以应用在线性变换上。子表达式消除应用在常数乘法时，将常数转化为二进制格式，再通过检验确定各个常数之间的公共子表达式。常数的二进制表示中的非零位决定实现该常数乘法器所需要产生和累加的部分积的个数。现在考虑线性变换的一般形式：$\mathbf{y} = \mathbf{T} * \mathbf{x}$，其中$\mathbf{T}$是$m \times n$的矩阵，$\mathbf{y}$是长度为$m$的向量，$\mathbf{x}$是长度为$n$的向量（$m$可以等于$n$）。写成如下的等价形式：

562

$$y_i = \sum_{j=1}^{n} t_{ij} x_j, \quad i = 1, \ldots, m \tag{15-1}$$

当考虑线性变换时，子表达式消除问题包含3个基本步骤。第一步是将计算积 $t_{ij}\ x_j$ 所需要的移位和加法次数最小化，这一步可以利用上文的迭代匹配算法实现。

表15-4 每一列的子表达式

第1列	第2列	第3列	第4列
0101	1000	0010	1001
0010	1011	0111	0100
1100			0010

例15.3.2 分析下面的变换矩阵：

$$\mathbf{T} = \begin{bmatrix} 7 & 8 & 2 & 13 \\ 12 & 11 & 7 & 13 \\ 5 & 8 & 2 & 15 \\ 7 & 11 & 7 & 11 \end{bmatrix} \tag{15-2}$$

变换矩阵的每一列中的常数要乘上一个公共变量。例如变量 x_1 要乘上第一列的常数组[7，12，5，7]。所以迭代匹配算法要应用到每一列常数上，以减少常数乘法实现所需要的移位和加法的次数。表15-4用无符号型二进制格式表示出了实现每列常数乘法所需要的子表达式。例如 $7 * x_1$ 用 $(0101) * x_1 + (0010) * x_1$ 实现。

第二步是用第一步得到的子表达式产生某些特定的积。这些积将在下一步按适当的方式

相加得到 y_i。由表15-4可以得到如下的积：

$$
\begin{aligned}
&p_1 = 0101 * x_1, \quad p_2 = 0010 * x_1, \quad p_3 = 1100 * x_1 \\
&p_4 = 1000 * x_2, \quad p_5 = 1011 * x_2, \quad p_6 = 0010 * x_3, \quad p_7 = 0111 * x_3 \\
&p_8 = 1001 * x_4, \quad p_9 = 0100 * x_4, \quad p_{10} = 0010 * x_4
\end{aligned}
$$

563

再利用这些积表示出 y_i：

$$
\begin{aligned}
y_1 &= p_1 + p_2 + p_4 + p_6 + p_8 + p_9 \\
y_2 &= p_3 + p_5 + p_7 + p_8 + p_9 \\
y_3 &= p_1 + p_4 + p_6 + p_8 + p_9 + p_{10} \\
y_4 &= p_1 + p_2 + p_5 + p_7 + p_8 + p_{10}
\end{aligned}
\tag{15-3}
$$

最后一步共享 y_i 中公共的加法。这一步和本章前面讨论的MCM问题类似。跟部分积共享中部分积用常数乘法器中的非零位位置表示不同，这里共享的是完成上述过程前两步得到的积。用一个k位的字表示 y_i $(1 \leqslant k \leqslant 10)$，字中的每一位表示与第二步中的积对应位的位置。使用这种表示，得到如下结果：

$$
\begin{aligned}
&y_1 = 1101010110, \quad y_2 = 0010101110 \\
&y_3 = 1001010111, \quad y_4 = 1100101101
\end{aligned}
\tag{15-4}
$$

现在，迭代匹配算法可以再次用来减少求得 y_1、y_2、y_3 和 y_4 需要的加法次数。应用迭代匹配算法得到的 y_1、y_2、y_3 和 y_4 表述如下：

$$
\begin{aligned}
y_1 &= p_2 + (p_1 + p_4 + p_6 + p_8 + p_9) \\
y_2 &= p_3 + p_9 + (p_5 + p_7 + p_8) \\
y_3 &= p_{10} + (p_1 + p_4 + p_6 + p_8 + p_9) \\
y_4 &= p_1 + p_2 + p_{10} + (p_5 + p_7 + p_8)
\end{aligned}
\tag{15-5}
$$

因为公共表达式只需计算一次，在这里迭代匹配算法使计算 y_1、y_2、y_3 和 y_4 的加法次数由19次减少到13次。包括计算 p_1 到 p_{10} 的计算在内，产生 y_1、y_2、y_3 和 y_4 的总加法次数由35次减少到20次。 ■

这个例题说明，在实现具有常系数变换矩阵的线性变换时，使用子表达式消除能显著地减少移位和加法操作的次数。这个技术非常强大，因为在信号处理中使用的大多数变换都是现行变换，包括离散余弦变换（DCT）和快速傅里叶变换（FFT）。

564

15.3.2 多项式求值

子表达式消除可以应用于多项式求值以降低计算复杂度。假定要计算如下的多项式：

$$
x^{13} + x^7 + x^4 + x^2 + x \tag{15-6}
$$

如果不考虑冗余，该多项式求值需要22次乘法。然而，如果分析各个指数就能发现可以大大减少这个多项式求值需要乘法的次数。考虑指数的无符号型二进制表示：

$$
1 = 0001, \quad 2 = 0010, \quad 4 = 0100, \quad 7 = 0111, \quad 13 = 1101
$$

可以把 x^7 视作 $x^4 * x^2 * x^1$。通过检查这些指数的二进制表示，可以找出在多项式求值时可以共享的公共子表达式。例如指数7和13共享了子表达式0101。如果对该多项式求值问题应用

子表达式消除，子表达式可以按如下方法计算:

$$x^8 * (x^4 * x) + x^2 * (x^4 * x) + x^4 + x^2 + x \tag{15-7}$$

中间项 x^2、x^4 和 x^8 都只需要一次乘法运算:

$$x^2 = x * x, \quad x^4 = x^2 * x^2, \quad x^8 = x^4 * x^4$$

显然，计算 x^2、x^4 和 x^8 将增加一些冗余操作。但从理论上说，这些运算难度不大，而且只在指数二进制表示的位数总长度增加一位的时候才增加一次计算指数为2的幂的运算。例如，如果多项式的指数用5位二进制字表示时，需要计算 x^2、x^4、x^8 和 x^{16} 作为构成多项式指数的“片段”。从公式（15-7）可看出，（15-6）式只用6次乘法即可计算出来，而如果不在计算过程中消除冗余，原来的计算需要22次乘法。

显而易见，多项式消除可以用于简单多项式求值，还可以把这种技术利用在针对同一个变量的多个多项式求值。分析下面3个多项式等式:

$$\begin{aligned} w(x) &= 11x^5 + 3x^4 + 6x^3 + 5x \\ y(x) &= 13x^5 + 7x^4 + 11x^3 \\ z(x) &= 7x^5 + 15x^4 + 5x^2 + 7x \end{aligned} \tag{15-8}$$

565

对多个多项式应用子表达式消除的过程本质上包含两个步骤。第一步是关于减少求出各次 x 的幂所需要乘法的次数；第二步是关于减少上面得到的幂和常系数相乘时候的移位和加法的次数。这里的第一步和上面提到的单个多项式求值处理过程类似：检查一组多项式，找出所有需要计算的 x 的幂，然后用最少次数的乘法来完成。如公式（15-8）式所示，需要计算 x^2、x^3、x^4 和 x^5。通过在计算中应用子表达式消除，这四个 x 的幂可用四次乘法完成。完成了各次的 x 的幂的计算，就需要进行第二步，也就是最后一步，减少常系数乘法器需要的移位和加法的次数。子表达式消除可以应用到那些和同一个 x 的幂相乘的多个常系数上，例如同和 x^5 相乘的11和13共享了子表达式1001。相比不做简化的27次加法，在对常系数乘法器应用子表达式消除后，整个计算只需要20次加法。

15.4 数字滤波器中的子表达式共享

本节分析使用公共子表达式FIR数字滤波器的实现。本节中使用了整数的正则符号数（CSD）（见第13.6.1节）表示和含一个十进制位的二进制混合补码表示。关于使用特定数目的有符号2的幂项的FIR数字滤波器的量化将在附录G中给予说明。

例15.4.1 本例分析一个简单乘法中的子表达式消除。

$$y = 0.10\bar{1}00010\bar{1} * x$$

它可以这样实现:

$$y = (x >> 1) - (x >> 3) + (x >> 7) - (x >> 9)$$

其中$>>i$表示算术右移i位，效果上等同于乘2^{-i}。或者还可以这样实现这个乘法:

$$\begin{aligned} x2 &= x - x >> 2 \\ y &= (x2 >> 1) + (x2 >> 7) \end{aligned}$$

566 这种方法少了一次加法。 ■

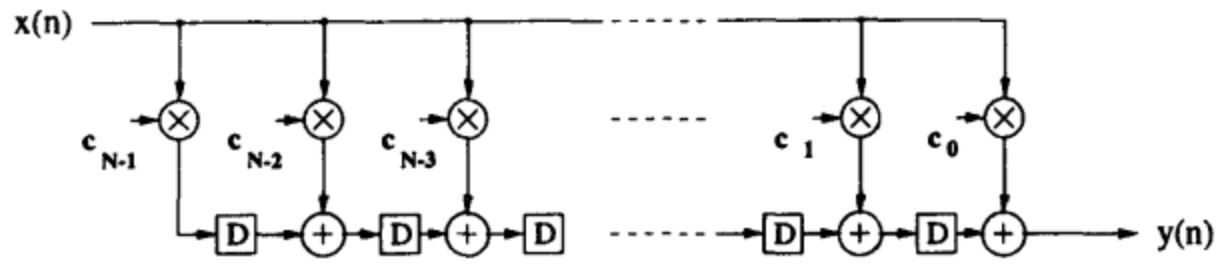


图15-1 常用的直接移位FIR滤波器结构

第15.3节已经讨论过，多个常数乘上一个变量可以应用子表达式消除。这些想法能有效地应用在FIR滤波器上以减少滤波器实现的硬件复杂度。为了实现子表达式消除变换，这样的一个N轮FIR滤波器：

$$y(n) = c_0x(n) + c_1x(n-1) + \cdots + c_{N-1}x(n-N+1) \tag{15-9}$$

必须用移位直接形式结构来描述（见图15-1）。这种结构通常又称数据广播滤波器结构（见1.4.1节）。通过这种结构，一个变量可以同时乘上多个常数。可以使用很多种技术[2]~[4]，将子表达式消除应用到这种数据广播FIR滤波器上。所有这些方法都是类似的，出发点都是通过检查滤波器硬件实现系数，来确定系数中公共的硬件单元。这些公共的硬件单元在各个滤波器系数之间共享。数字滤波器设计中的子表达式消除的基本原则将在本节中通过几个例题来说明。

为了系统地完成子表达式共享，用一个表格（矩阵）$\{x_{ij}\}$ 表示滤波器的操作，其中行索引表示延迟 i 位，列索引表示移位 j 位。也就是说，第 i 行表示数据 $x(n-i)$ 对应的系数 c_i，而第 i 行第0列表示 c_i 的最高位（msb），第 i 行第 $W-1$ 列为 c_i 的最低位（lsb），其中 W 为字长；行和列索引都从0开始；如果使用二进制表示，除标志位以外表格中所有元素都为0或者1，如果用CSD代码则是数位集合 {0，1，-1}。第 i 行第 j 列绝对值为1的非零元素表示 $x(n-i)>>j$。具体做加法还是减法则取决于索引对应元素是+1还是-1。在下面的例题中，用 $x1$ 取代 x，表示初始乘数，而用 $x2$、$x3$ 等表示中间过程第2、3等步所产生的子表达式，$x1[-i]$ 表示 $x(n-i)$，这里的索引 $[-i]$ 表示 i 个采样延迟。

567

例15.4.2 分析如下的常系数FIR滤波器（采用CSD代码形式）：

$$\begin{aligned} y(n) &= 1.000\bar{1}00000 * x(n) + 0.\bar{1}0\bar{1}010010 * x(n-1) \\ &\quad +0.00010000\bar{1} * x(n-2) \end{aligned} \tag{15-10}$$

用表格描述如下：

1				-1					
	-1		-1		1			1	
				1					-1

这个滤波器含有8个非零的数据，常规运算需要作7次加法。但是子表达式 $x1 + x1[-1] >> 1$出现了4次，对不同的量进行移位和延迟，在表中用圆圈出。于是，这个滤波器可以只用4次加法表述：

$$
\begin{aligned}
x2 &= x1 - x1[-1] >> 1 \\
y &= x2 - (x2 >> 4) - (x2[-1] >> 3) + (x2[-1] >> 8)
\end{aligned}
$$

或者使用另外一种计算方案：

$$
\begin{aligned}
x2 &= x1 - (x1 >> 4) - (x1[-1] >> 3) + (x1[-1] >> 8) \\
y &= x2 - (x2[-1] >> 1)
\end{aligned}
$$

■

上面这个例题说明了一个重要的特性，现总结为下面的结论。

结论15.4.1 一个规模为m出现n次的子表达式，等价于一个规模为n出现m次的子表达式。

现在用下面的例题来阐述一个子表达式匹配算法。

例15.4.3 考虑下面的4抽头FIR滤波器。

$$
\begin{aligned}
y(n) = {} & \bar{1}.01010000010 * x(n) + 0.\bar{1}000\bar{1}0\bar{1}0\bar{1}0\bar{1} * x(n-1) \\
& + 0.\bar{1}0010000010 * x(n-2) + 1.0000010\bar{1}000 * x(n-4)
\end{aligned}
\tag{15-11}
$$

其中系数是CSD格式的代码。这个设计的子结构共享过程如下：

(1) 首先从包含FIR滤波器系数的表格开始。表格中每一个绝对值为1的元素就表示对 $x1$ 进行一次加法或者减法的操作。识别出大小为2的“最优”子表达式。最优子表达式是根据优值函数（benefit function）判定的，优值函数中的出现数（number of occurrence）是一个重要因子。 568

-1		1		1						1	
	-1				-1		-1		-1		-1
	-1			1						1	
1						1		-1			

(2) 将各个子表达式的出现数消去，并且在原来子表达式两项的第一项的位置，用一个2或者-2来替代。这里的“第一部分”是表格的元素中依照行优先的顺序判断的。得到的表格中，每一个绝对值为2的输入都表示对$x2$进行一次加法或者减法的操作，对应的就成为2级子表达式。

-1		2		1						2	
					-2		-1				-2
	-2										
						1		-1			

(3) 记录相应子表达式的定义。虽然这里可能需要进行负移位的操作，目前不用管它们。

$$x2 = x1 - x1[-1] >> (-1)$$

(4) 继续寻找更多的子表达式，直到结束。

-1		3								2	
					-3						-2
	-2										
						1		-1			

$$x3 = x2 + x1 >> 2$$

(5) 写出完整的滤波器的定义。

$$\begin{aligned} x2 &= x1 - x1[-1] >> (-1) \\ x3 &= x2 + x1 >> 2 \\ y &= -x1 + x3 >> 2 + x2 >> 10 - x3[-1] >> 5 - x2[-1] >> 11 \\ &\quad -x2[-2] >> 1 + x1[-3] >> 6 - x1[-3] >> 8 \end{aligned}$$

569

(6) 如果某个子表达式的定义包含了负移位操作，就采用下面的方案来修改该变量的定义以及后面的使用。

$$\begin{aligned} x2 &= x1 >> 1 - x1[-1] \\ x3 &= x2 + x1 >> 3 \\ y &= -x1 + x3 >> 1 + x2 >> 9 - x3[-1] >> 4 - x2[-1] >> 10 \\ &\quad -x2[-2] + x1[-3] >> 6 - x1[-3] >> 8 \end{aligned}$$

■

注意在例15.4.3中，每次迭代时的子表达式选取，不仅要根据子表达式出现的次数，还要考虑预期的各种操作减少的数目，包括加法和延迟元件。

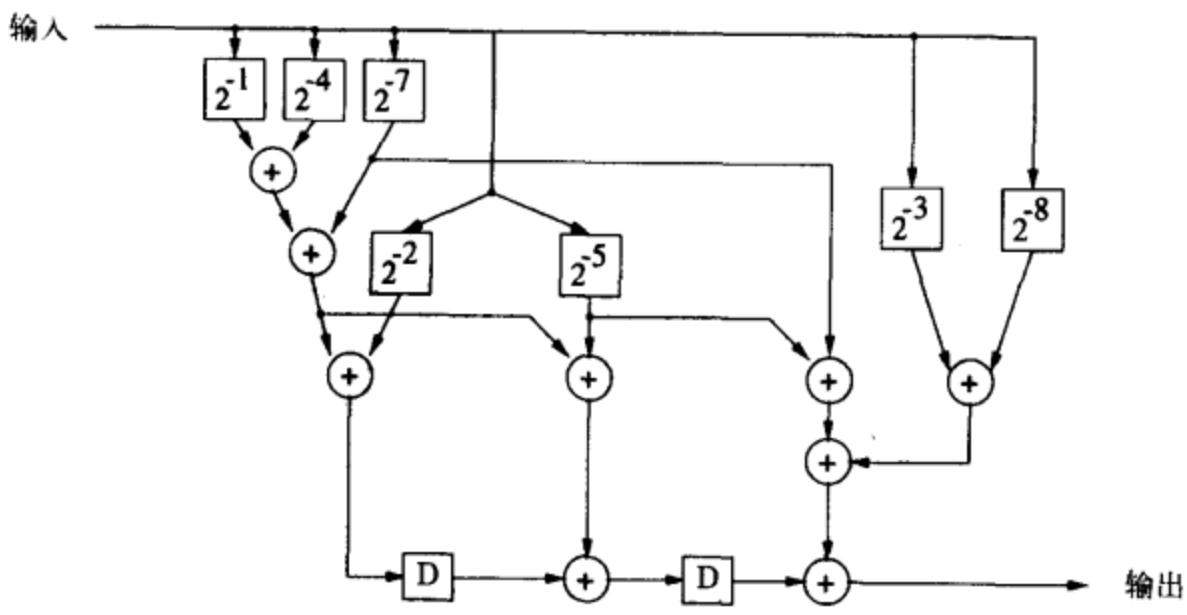


图15-2 例15.4.4中使用了子表达式共享的3阶FIR滤波器

例15.4.4 本例题要对一个3轮滤波器应用子表达式消除，这个滤波器的系数采用补码格式：$c_2 = 0.11010010$, $c_1 = 0.10011010$, $c_0 = 0.00101011$。图15-2给出了应用子表达式消除的该3轮滤波器。使用了子表达式共享后，这个滤波器只需要7次移位和9次加法，如果使用常规的无乘法器实现，则需要12次移位和11次加法。

另外一种基于子表达式共享的结构见图15-3，这种结构需要8次加法。

使用位并行进位保留加法很容易实现图15-2和图15-3中的结构。在位并行设计中，移位器通过合适的硬连线来实现。使用基于Horner规则的实现方式（见第13.6.2.1节）可以改进移位的准确性。然而在位串行设计中，诸如 2^{-1} 这样的移位操作却只能通过在流水线割集的恰当位置上插入延时元件达到同步后才能实现。需要注意的是，由于使用了大量的同步延迟，位串行设计显然是不能让人满意的。

570

■

利用CSD表示，数字滤波器可以用较少的硬件实现。在CSD表示中，两个最常见的子表达式是 $x-x>>2$ 和 $x+x>>2$，统计上看结果也总是如此。只要通过寻找这两个子表达式的发

生，可以很容易达到充分地减少电路面积的效果。对应在滤波器的系数中，每一个表达式 $x-x>>2$ 相应于 $10\bar{1}$ 序列，而表达式 $x+x>>2$ 相应于一个101序列。可以看出，一个W位的CSD数可以渐进地拆分为 $W/18+O(1)$对 $10\bar{1}$、$W/18+O(1)$对101、以及$W/9+O(1)$个孤立的1或者 $\bar{1}$ [3]。若把 $10\bar{1}$ 对、101对或者孤立的非零位称作项，可以看出一个W位的CSD数渐进的、期望的项数为$2(W+1)/9$。相对于非零位总个数为 $(3W+1)/9$，这样的结果意味着节省了33%。而且，利用这些公共子表达式带来的好处还有不会显著地增加布线成本。图15-4示出了使用项共享FIR滤波器的一种可能的线性布局。应用于对该滤波器实现进行变换的这种子表达式方法所依赖的一个事实，就是所有的CSD系数都可以由三种简单子表达式“构成”，即：101、$10\bar{1}$ 和1 [3]。这三种子表达式可以移位、相加，构成滤波器的系数。在图15-4中，$x+x>>2$ 和 $x-x>>2$ 两项预先计算好，然后用 $x+x>>2$、$x-x>>2$ 和 x 这三项沿着单元的行馈送。每个加法器从3个总线获取它的输入。图中的右移算子根据对应的系数值可以表示多位右移和相加。尽管这里给出的3个总线在数据通道之上，但现实中最好的布局规划是让它们从加法单元上面经过。

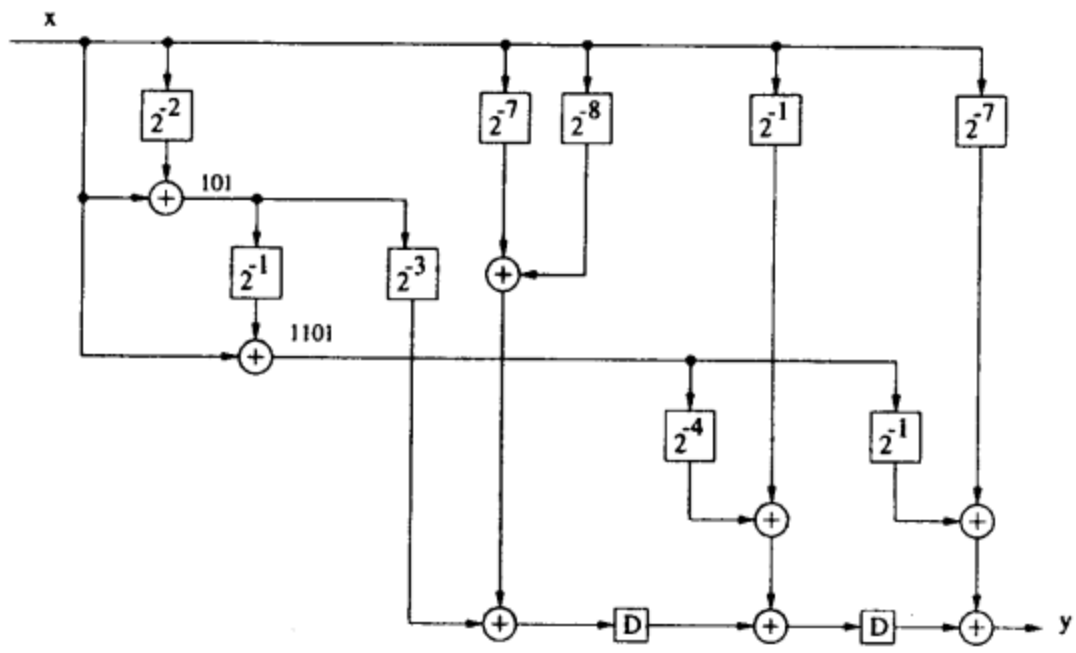


图15-3 例15.4.4中使用了子表达式共享的3阶FIR滤波器

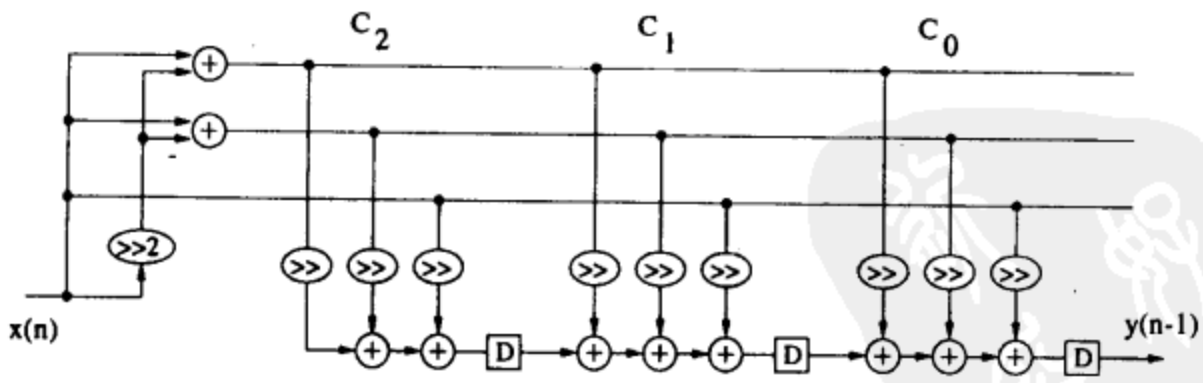


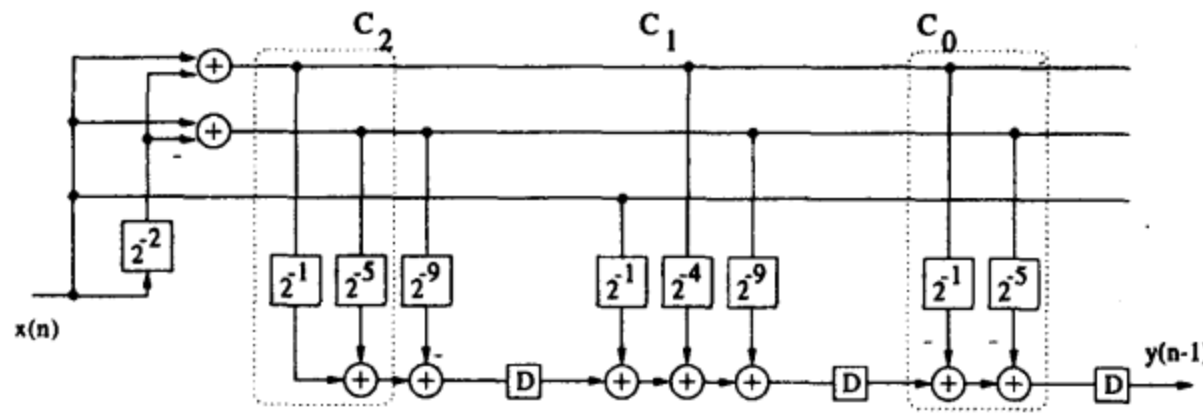
图15-4 一个使用了项共享的FIR滤波器，其中101和 $10\bar{1}$ 这两个在CSD数中最常见的子表达式以及孤立的1在所有的滤波器系数中共享

例15.4.5 本例考虑一个使用了CSD表示中的两个最常见子表达式的3阶FIR滤波器的设计。这个滤波器的12位系数使用CSD二进制格式编码如下：

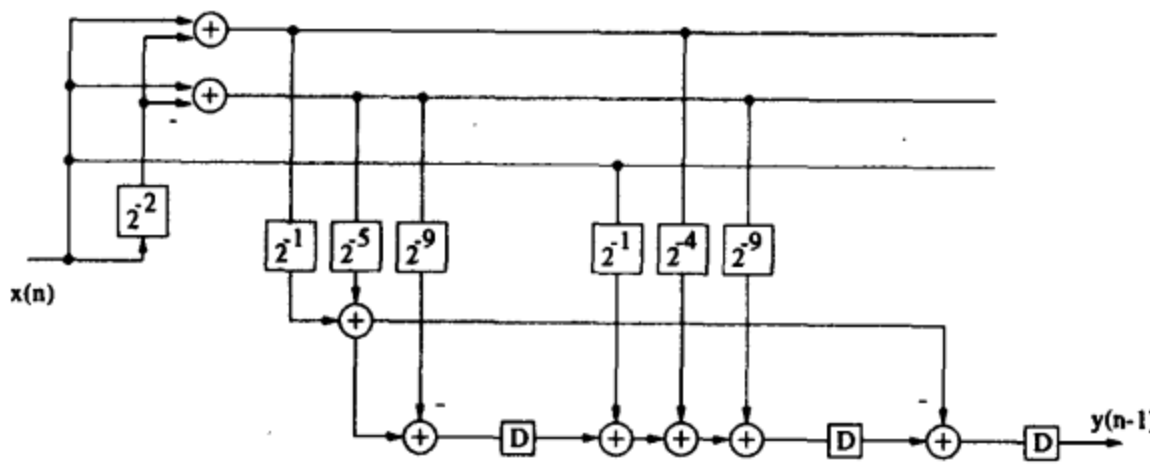
$$c_2 = 0.\underline{101}0\underline{10\bar{1}}0\underline{10\bar{1}},\quad c_1 = 0.100\underline{101}00\underline{10\bar{1}},$$
$$c_0 = 0.\underline{\bar{1}0\bar{1}}0\underline{\bar{1}01}0000$$

其中，子表达式101和$10\bar{1}$已经被标注了下划线。图15-5a给出了利用CSD数的两个最常见子表达式来实现这个滤波器。注意到两个在虚线框中的加法相同，它们是可以共享的，如在图15-5b中所示。结果是，滤波器只需要7次加法和7次移位。而在原来常规的无乘法器实现中，却需要12次加法和12次移位操作。 ■

这一节的例题说明了子结构共享应用在减少FIR滤波器实现的硬件代价时的有效性。而在高阶的FIR滤波器中，子结构共享的可能性加大，使得硬件节约效果更为显著。



a)



b)

图15-5 使用了CSD子表达式共享的3阶FIR滤波器例子

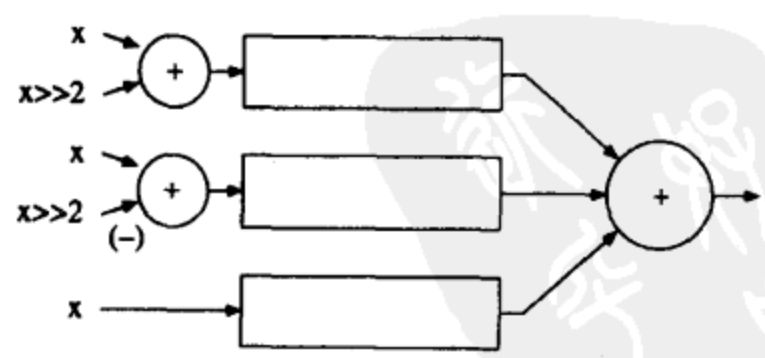


图15-6 对于进位保留的FIR滤波器布局的另一种布局方案

图15-6所示的另一种布局也是可行的，在这里各种类型的项通过各自的数据通道完成关联。然后将3个分离的结果求和得到最终输出。

在这种布局中，如果这3条数据通道可以达成某种程度的平衡，即含有相同数目的项，硬

件实现的效果将达到最佳。这样做的一个好处是布局方便。另外一个好处有关时钟频率。电路的最大时钟频率是由流水线的任一级中加法器的最大数目决定的。为了让一级中加法器最少，需要使在3个数据通道中的加法器数目平衡。但是，前面已经说过，平均算来在数据通道x中项的个数是数据通道 $x + x >> 2$ 或 $x-x >> 2$ 的两倍。这个不等关系可以通过将项 x 替换为项 $x + x >> 2$ 和 $x-x >> 2$ 来解决。这样就不能严格地符合CSD数字表示法。例如，一个包含2个单独的1的序列1001对应项 $x + x >> 3$，我们可以写成 $101\bar{1}$，对应项 $(x + x >> 2) - x >> 3$。这样，其中的一个项 x 就替换为项 $x + x >> 2$ 项。同理，一个序列10001可以写成 $10\bar{1}00+00101$。通过这种方法，两个单独的1就被替换成了两个数对。

15.5 加性和乘性数字拆分

数字拆分[2]，包括加性和乘性，是一种用于减少线性电路的硬件复杂度或者功耗的数字变换技术。加性数字拆分是多常数乘法中子表达式消除的另外一种形式。典型的数字拆分是在无限精度常系数被约简为二进制表示前对常系数进行处理，这种处理导致了一种高层次的强度缩减。加性数字拆分可以按行或者按列进行。

数字拆分是一种运行在优化算法框架内的迭代技术。优化算法通过使用代价函数管理数字拆分算法。在每一步迭代中，在代价函数里能得到最大缩减的数字拆分将被选出，然后重复这样的进程，直到无法再简化。本节将说明数字拆分的基本原则。至于获得最优解法的过程这里不予讨论（见[2]，[5]，[6]）。

在说明数字拆分如何实现之前，首先必须定义一个线性电路的概念。线性电路定义如下:

$$\mathbf{y} = \mathbf{T}\mathbf{x}$$

其中:

$$\mathbf{y} = \begin{bmatrix} y_0(n+1) \\ y_1(n+1) \\ \vdots \\ y_{N-1}(n+1) \end{bmatrix}$$

571 ~ 574

$$\mathbf{T} = \begin{bmatrix} t_{0,0} & t_{0,1} & \cdots & t_{0,M-1} \\ t_{1,0} & t_{1,1} & \cdots & t_{1,M-1} \\ \vdots & \vdots & \ddots & \vdots \\ t_{N-1,0} & t_{N-1,1} & \cdots & t_{N-1,M-1} \end{bmatrix} \tag{15-12}$$

$$\mathbf{x} = \begin{bmatrix} y_0(n) \\ \vdots \\ y_{J-1}(n) \\ x_0(n) \\ \vdots \\ x_{K-1}(n) \end{bmatrix}$$

这里的 $M = J + K$。请注意这里线性电路的定义包含了递归和非递归电路。如果是非递归电路，$y_0(n), \ldots, y_{J-1}(n)$ 将完全包含在 x 中，此时 $M = K$。数字拆分将用于对矩阵T的常系数

进行变换，从而减少线性电路的面积和功率。这个过程相当于对线性电路的数据流图进行变换。但是，线性电路的功能要保持不变。

15.5.1 基于行的加性数字拆分

如题所示，基于行的数字拆分就是对矩阵**T**的行元素进行操作。为了进行基于行的数字拆分，需要先定义一个轴心元素和一个目标元素。假设现在要对矩阵**T**的第i行进行数字拆分，第i行的轴心元素定义为 $t_{i,p}$，而目标元素定义为 $t_{i,q}$。然后用轴心元素 $t_{i,p}$ 把目标元素 $t_{i,q}$ "分割" 为 $t_{i,p}$ 和 δ 两部分 ($t_{i,q} = t_{i,p} + \delta$)，即 $\delta = t_{i,q} - t_{i,p}$。进行完基于行的数字拆分处理后，矩阵**T**将变形为一个新的矩阵**T**'

$$\mathbf{T}' = \begin{bmatrix} t_{0,0} & \cdots & t_{0,p} & \cdots & t_{0,q} & \cdots & t_{0,M-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ t_{i,0} & \cdots & 0 & \cdots & \delta & \cdots & t_{i,M-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ t_{N-1,0} & \cdots & t_{N-1,p} & \cdots & t_{N-1,q} & \cdots & t_{N-1,M-1} \end{bmatrix} \tag{15-13}$$

为了保证**T**'等于**T**，将产生两个新的矩阵。第一个称之为**AG**，它包含的行用适当的方法加上**T**'即得到**T**的行。初始**AG**矩阵为空。每一步基于行的数字拆分迭代之后，加至**T**'第i行以得到**T**第i行所需的行信息将添加到**AG**。显然，在对**T**的第i行进行数字拆分操作之后，为了从**T**'能得到原来的**T**，必须在**T**'的第i行加上行信息 $[0...0\ t_{i,p}\ 0...0\ t_{i,p}\ 0...0]$。第 r 步基于行的数字拆分迭代操作完成后，**AG**将包含各有M个元素的 r 行。

575

第二个称之为**W**的矩阵决定了应该如何将**AG**的各行加到**T**'的各行来得到原来的矩阵**T**。初始的**W**设置为一个$N \times N$阶的单位矩阵。做完一次基于第i行的数字拆分之后，在**W**的右边需要添加一列，这一列包含了N个元素，除了第i个元素置为1以外，所有的输入均置为0。将这个矩阵和**AG**组合，可以建立**T**和**T**'的等式:

$$\mathbf{T} = \mathbf{W} * \begin{bmatrix} \mathbf{T}' \\ \mathbf{AG} \end{bmatrix} \tag{15-14}$$

为了更好地理解基于行的数字拆分是如何操作的，利用下面的线性电路说明这种技术:

$$\begin{bmatrix} y_0(n+1) \\ y_1(n+1) \\ y_2(n+1) \end{bmatrix} = \begin{bmatrix} 0.4 & 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0.2 & 0.9 \\ 0.5 & 0.3 & 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} y_0(n) \\ y_1(n) \\ y_2(n) \\ y_3(n) \end{bmatrix} \tag{15-15}$$

数字拆分将在第一行进行，其中分别将0.4（第一个元素）和0.4（第三个元素）作为轴心元素和目标元素。由此产生的矩阵**T**'、**AG**和**W**如下:

$$\mathbf{T}' = \begin{bmatrix} 0 & 0.7 & 0 & 0.8 \\ 0.3 & 0.6 & 0.2 & 0.9 \\ 0.5 & 0.3 & 0.2 & 0.7 \end{bmatrix}$$

$$\mathbf{AG} = \begin{bmatrix} 0.4 & 0 & 0.4 & 0 \end{bmatrix} \tag{15-16}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

利用这些矩阵，公式（15-14）可以写成:

$$\begin{bmatrix}0.4 & 0.7 & 0.4 & 0.8\\0.3 & 0.6 & 0.2 & 0.9\\0.5 & 0.3 & 0.2 & 0.7\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & 1\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}0 & 0.7 & 0 & 0.8\\0.3 & 0.6 & 0.2 & 0.9\\0.5 & 0.3 & 0.2 & 0.7\\0.4 & 0 & 0.4 & 0\end{bmatrix} \tag{15-17}$$

576

通过基于行的数字拆分处理，输出 $y_0(n+1)$ 的计算方法已经被替换为一种硬件需求较少的等效方式。本来，$y_0(n+1)$ 的计算需要4次乘法和3次加法，计算式如下：$0.4y_0(n) + 0.7y_1(n) + 0.4y_2(n) + 0.8x_0(n)$。变形以后 $y_0(n+1)$ 的计算式是 $0.4(y_0(n) + y_2(n)) + 0.7y_1(n) + 0.8x_0(n)$，此时需要3次乘法和3次加法。现在考虑对**T**'的第3行进行基于行的数字拆分。取0.2作为轴心，0.7作为目标:

$$\mathbf{T}'=\begin{bmatrix}0 & 0.7 & 0 & 0.8\\0.3 & 0.6 & 0.2 & 0.9\\0.5 & 0.3 & 0 & 0.5\end{bmatrix}$$

$$\mathbf{AG}=\begin{bmatrix}0.4 & 0 & 0.4 & 0\\0 & 0 & 0.2 & 0.2\end{bmatrix}$$

$$\mathbf{W}=\begin{bmatrix}1 & 0 & 0 & 1 & 0\\0 & 1 & 0 & 0 & 1\\0 & 0 & 1 & 0 & 0\end{bmatrix}$$

$$\begin{bmatrix}0.4 & 0.7 & 0.4 & 0.8\\0.3 & 0.6 & 0.2 & 0.9\\0.5 & 0.3 & 0.2 & 0.7\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & 1 & 0\\0 & 1 & 0 & 0 & 1\\0 & 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & 0.7 & 0 & 0.8\\0.3 & 0.6 & 0.2 & 0.9\\0.5 & 0.3 & 0 & 0.5\\0.4 & 0 & 0.4 & 0\\0 & 0 & 0.2 & 0.2\end{bmatrix} \tag{15-18}$$

这次数字拆分迭代变换了计算输出 $y_2(n+1)$ 的方法。初始电路中，$y_2(n+1)$ 计算式是 $0.5y_0(n) + 0.3y_1(n) + 0.2y_2(n) + 0.7x_0(n)$，需要4次乘法和3次加法。进行数字拆分处理后的 $y_2(n+1)$ 计算式是 $0.5y_0(n) + 0.3y_1(n) + 0.2(y_2(n) + x_0(n)) + 0.5*x_0(n)$，需要4次乘法和4次加法。看起来变换实际上反而导致硬件复杂度增加，但是注意到原来乘 $x_0(n)$ 的常系数是0.7，这里被替换成了常系数乘数0.5。而一个乘以0.5的操作可以用一次简单的右移来实现。于是，“昂贵”的乘0.7就以多一次加法的代价替换为高效的乘0.5。数字拆分再一次降低线性电路的硬件复杂度。这样的数字拆分过程将一直继续下去，一直到线性电路的硬件复杂度无法再降低为止。

15.5.2 基于列的加性数字拆分

除了对**T**的列进行操作，基于列的数字拆分和基于行的数字拆分本质上是一致的。假设要对**T**的第p列进行基于列的数字拆分，分别取第 p 列的第 i 个元素 $t_{i,p}$ 和第 j 个元素 $t_{j,p}$ 为轴心元素和目标元素。和前面一样，用轴心元素把目标元素“分割”为两部分：$t_{i,p}$ 和 δ ($t_{j,p} = t_{i,p} + \delta$)，即：$\delta = t_{j,p} - t_{i,p}$。然后每次基于列的数字拆分迭代后，通过向**AG**增添具有 [0...0 $t_{i,p}$ 0...0] 格式的行最终组成矩阵**AG**。这一行如果加到矩阵**T**'的第 i 行和第 j 行，可以得到原来的变换矩阵**T**。进行数字拆分操作以前的**AG**为空。同样和基于行的数字拆分类似，矩阵**W**决定如何将**AG**加到**T**'而得到原来的变换矩阵**T**，**W**开始也是$N \times N$阶的单位矩阵。每次数字拆分迭代之后，形如 [0...0 1 0...0 1 0...0] 的一列将被添加到**W**的右边，其中值1分别为第 577

i 个和第 j 个元素。**T'** 和 **T** 的关系仍然如公式（15-14）所示。

下面再一次分析公式（15-15）定义的线性电路:

$$\begin{bmatrix} y_0(n+1) \\ y_1(n+1) \\ y_2(n+1) \end{bmatrix} = \begin{bmatrix} 0.4 & 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0.2 & 0.9 \\ 0.5 & 0.3 & 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} y_0(n) \\ y_1(n) \\ y_2(n) \\ y_3(n) \end{bmatrix} \tag{15-19}$$

我们将对**T**的第三列进行基于列的数字拆分处理，分别取第二个元素.2和第三个元素.2作为轴心以及目标元素。由此产生的矩阵 **T'**、**AG** 和 **W** 如下:

$$\begin{aligned} \mathbf{T}' &= \begin{bmatrix} 0.4 & 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0 & 0.9 \\ 0.5 & 0.3 & 0 & 0.7 \end{bmatrix} \\ \mathbf{AG} &= \begin{bmatrix} 0 & 0 & 0.2 & 0 \end{bmatrix} \\ \mathbf{W} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{aligned} \tag{15-20}$$

利用这些矩阵，公式（15-14）可写成:

$$\begin{bmatrix} 0.4 & 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0.2 & 0.9 \\ 0.5 & 0.3 & 0.2 & 0.7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.4 & 0.7 & 0.4 & 0.8 \\ 0.3 & 0.6 & 0 & 0.9 \\ 0.5 & 0.3 & 0 & 0.7 \\ 0 & 0 & 0.2 & 0 \end{bmatrix} \tag{15-21}$$

通过基于列的数字拆分处理，我们将输出 $y_1(n+1)$ 和 $y_2(n+1)$ 的计算方法替换成一种需要较少硬件的等效方式。本来，$y_1(n+1)$ 和 $y_2(n+1)$ 的计算式分别为: $0.3y_0(n) + 0.6y_1(n) + 0.2y_2(n) + 0.9x_0(n)$, $0.5y_0(n) + 0.3y_1(n) + 0.2y_2(n) + 0.7x_0(n)$; 总共需要8次乘法和6次加法。通过变换，项$0.2y_2(n)$ 只计算一次就在 $y_1(n+1)$ 和 $y_2(n+1)$ 中同时应用，从而节省了一次乘法操作。基于行和基于列的数字拆分的基本区别是: 基于行的数字拆分尝试消除单个输出的计算冗余; 基于列的数字拆分则尝试消除多个输出的计算时发生的冗余。

578

数字拆分通常和使用代价函数控制该数字拆分的优化算法一起进行。在数字拆分迭代时，无论是基于行还是基于列，都是选取使得代价函数得到最大缩减的轴心和目标元素来进行数字拆分。因为线性电路其实就是一种线性变换，数字拆分可以在线性变换时用于降低计算复杂度。

15.5.3 乘性数字拆分

乘性数字拆分[5]，[6]也是一种可以用于线性循环（线性变换），从而降低硬件复杂度或者电路功耗的数字变换技术。类似于加性数字拆分，乘性数字拆分也是通过代价函数管理的迭代技术。基于循环的功率或面积的代价函数，确定了乘性数字拆分算法每次迭代时最理想的拆分操作。

回到公式（15-12），线性变换定义如下:

$$\mathbf{y} = \mathbf{Tx} \tag{15-22}$$

对矩阵T进行乘性数字拆分，以减少线性电路的硬件复杂度或者功耗。借用如下线性代数的结论[5]，可知乘性数字拆分是一种相对简单的变换：

对于如下情况的常数K，积 $T_{i-1}\,T_i$ 保持不变：

(1) T_i 的第 j 行做如下变换（以 $R(i)$ 表示第 i 行，$C(i)$ 表示第 i 列）：

$$R(j) = R(j) - K * R(k)$$

(2) T_{i-1} 的第 k 列做如下变换：

$$C(k) = C(k) + K * C(j)$$

其中 j 和 k 是任意有效的行和列指数。 579

利用这些结论，用乘性数字拆分算法可以将原始矩阵T分解成一系列矩阵的积：$T = T_1 T_2 \ldots T_N$。考虑对矩阵应用乘性数字拆分：

$$\mathbf{T} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix} \tag{15-23}$$

第一步操作必须是将T写成一个单位矩阵和它自身的积：

$$\mathbf{T} = \mathbf{T_1}\mathbf{T_2}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix} \tag{15-24}$$

现在应用上面的结论，在不改变线性电路功能的前提下变换 $\mathbf{T}_1$ 和 $\mathbf{T}_2$。设定数字拆分的系数：$K = 3, j = 1, k = 2$ 。

$$\mathbf{T} = \mathbf{T_1}\mathbf{T_2}$$
$$\begin{bmatrix} 1 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} - 3a_{2,1} & a_{1,2} - 3a_{2,2} & a_{1,3} - 3a_{2,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix} \tag{15-25}$$

注意利用上面结论的第一部分 $R(j) = R(j) - K * R(k)$，对 $\mathbf{T}_2$ 进行变换。利用上面结论的第二部分 $C(k) = C(k) + K * C(j)$，对矩阵 $\mathbf{T}_1$ 进行变换。对 $\mathbf{T}_1$ 和 $\mathbf{T}_2$ 进行第二次变换，系数为 $K = 1, j = 2, k = 3$。

$$\mathbf{T} = \mathbf{T_1}\mathbf{T_2}$$
$$\begin{bmatrix} 1 & 3 & 3 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} - 3a_{2,1} & a_{1,2} - 3a_{2,2} & a_{1,3} - 3a_{2,3} \\ a_{2,1} - a_{3,1} & a_{2,2} - a_{3,2} & a_{2,3} - a_{3,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix} \tag{15-26}$$

在变换进程的任何时候，都可以在矩阵T的左边或右边加上单位矩阵，从而增加可执行的变换次数。

580

再考虑在T的右边加上单位矩阵的情况：

$$\mathbf{T} = \mathbf{T_1T_2T_3} \tag{15-27}$$

$$\begin{bmatrix} 1 & 3 & 3 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1}-3a_{2,1} & a_{1,2}-3a_{2,2} & a_{1,3}-3a_{2,3} \\ a_{2,1}-a_{3,1} & a_{2,2}-a_{3,2} & a_{2,3}-a_{3,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

现在对矩阵 $\mathbf{T}_2$ 和 $\mathbf{T}_3$ 可以进行和对 $\mathbf{T}_1$ 和 $\mathbf{T}_2$ 的变换一样的乘性数字拆分。设定对 $\mathbf{T}_2$ 和 $\mathbf{T}_3$ 变换的系数为: $K = 1, j = 3, k = 1$。

$$\mathbf{T} = \mathbf{T_1T_2T_3} \tag{15-28}$$

$$\begin{bmatrix} 1 & 3 & 3 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1}-3a_{2,1}+a_{1,3}-3a_{2,3} & a_{1,2}-3a_{2,2} & a_{1,3}-3a_{2,3} \\ a_{2,1}-a_{3,1}+a_{2,3}-a_{3,3} & a_{2,2}-a_{3,2} & a_{2,3}-a_{3,3} \\ a_{3,1}+a_{3,3} & a_{3,2} & a_{3,3} \\ a_{4,1}+a_{4,3} & a_{4,2} & a_{4,3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

添加单位矩阵再对线性电路作乘性数字拆分变换，这个过程将一直重复下去，直到从降低功耗或电路的硬件复杂度方面无法再改进为止。典型情况是，在拆分算法的每一次迭代中，使用一种通过试验所有可能的矩阵变换的贪婪算法[6]控制乘性拆分。算法将选择能达到功率或硬件消耗最大缩减的变换。拆分进程将一直进行到无法优化为止。

现在用线性变换说明乘性数字拆分算法:

$$\mathbf{T} = \begin{bmatrix} 0.4 & 0.7 & 0.4 \\ 0.6 & 0.8 & 0.2 \\ 0.1 & 0.3 & 0.2 \end{bmatrix} \tag{15-29}$$

将T写成单位矩阵和自身乘积的等价形式:

581

$$\mathbf{T} = \mathbf{T}_1\mathbf{T}_2$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.4 & 0.7 & 0.4 \\ 0.6 & 0.8 & 0.2 \\ 0.1 & 0.3 & 0.2 \end{bmatrix} \tag{15-30}$$

然后，用 $K = 1, j = 2, k = 3$ 的乘性拆分对矩阵 $\mathbf{T}_1$ 和 $\mathbf{T}_2$ 进行变换:

$$\mathbf{T} = \mathbf{T}_1\mathbf{T}_2$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.4 & 0.7 & 0.4 \\ 0.5 & 0.5 & 0 \\ 0.1 & 0.3 & 0.2 \end{bmatrix} \tag{15-31}$$

$\mathbf{T}_2$ 的第二行由3个复杂的乘法变换为2个简易的乘法（乘.5的操作只需要做一次右移一位）。对 $\mathbf{T}_1$ 和 $\mathbf{T}_2$ 做第二次变换，取 $K = 2, j = 1, k = 3$:

$$\mathbf{T}=\mathbf{T}_1\mathbf{T}_2$$

$$\begin{bmatrix}1&0&2\\0&1&1\\0&0&1\end{bmatrix}\begin{bmatrix}0.2&0.1&0\\0.5&0.5&0\\0.1&0.3&0.2\end{bmatrix}\tag{15-32}$$

这里 $\mathbf{T}_2$ 的第一行由3个复杂的乘法变换为2次复杂的乘法。现在，考虑在 $\mathbf{T}_2$ 的右边加上单位矩阵，以对 $\mathbf{T}_2$ 的列进行变换:

$$\mathbf{T}=\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3$$

$$\begin{bmatrix}1&0&2\\0&1&1\\0&0&1\end{bmatrix}\begin{bmatrix}0.2&0.1&0\\0.5&0.5&0\\0.1&0.3&0.2\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\tag{15-33}$$

用 $K=-1, j=2, k=1$ 变换 $\mathbf{T}_2$ 和 $\mathbf{T}_3$:

$$\mathbf{T}=\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3$$

$$\begin{bmatrix}1&0&2\\0&1&1\\0&0&1\end{bmatrix}\begin{bmatrix}0.1&0.1&0\\0&0.5&0\\-0.2&0.3&0.2\end{bmatrix}\begin{bmatrix}1&0&0\\1&1&0\\0&0&1\end{bmatrix}\tag{15-34}$$

现在，进行最后3次变换:

$$\mathbf{T}=\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3$$

$$\begin{bmatrix}-3&0&2\\-2&1&1\\-2&0&1\end{bmatrix}\begin{bmatrix}0.1&0.1&0\\0&0.5&0\\0&0.5&0.2\end{bmatrix}\begin{bmatrix}1&0&0\\1&1&0\\0&0&1\end{bmatrix}\tag{15-35}$$

582

$$\mathbf{T}=\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3$$

$$\begin{bmatrix}-3&2&2\\-2&2&1\\-2&1&1\end{bmatrix}\begin{bmatrix}0.1&0.1&0\\0&0.5&0\\0&0&0.2\end{bmatrix}\begin{bmatrix}1&0&0\\1&1&0\\0&0&1\end{bmatrix}\tag{15-36}$$

$$\mathbf{T}=\mathbf{T}_1\mathbf{T}_2\mathbf{T}_3$$

$$\begin{bmatrix}-3&2&2\\-2&2&1\\-2&1&1\end{bmatrix}\begin{bmatrix}0&0.1&0\\-0.5&0.5&0\\0&0&0.2\end{bmatrix}\begin{bmatrix}1&0&0\\2&1&0\\0&0&1\end{bmatrix}\tag{15-37}$$

显然，乘性拆分变换降低了原始变换矩阵T的硬件复杂度。原始矩阵需要9次复杂的乘法。通过使用乘性拆分，9次复杂乘法减少为2次复杂乘法以及一些加法和移位操作。

15.6 结论

与算法级的强度缩减类似，数字强度缩减能显著降低硬件复杂度。本章给出了很多基于子表达式消除或者子结构共享的数字强度缩减方法。这些方法可以用于降低固定系数数字滤波器实现的复杂度，例如FIR和IIR数字滤波器、多速滤波器和小波滤波器。此外还发现，子表达式消除技术在编译器中已经得到普遍应用[7]。

15.7 习题

1. 假设乘数a、b、c、d要乘上共同的被乘数x。分别按下面给的各组乘数求出实现 $a\times x$、

$b \times x$、$c \times x$ 和 $d \times x$ 需要的移位和加法的次数（假设使用无符号型二进制表示）:

(a) $a = 54, b = 45, c = 43, d = 21$

(b) $a = 93, b = 59, c = 55, d = 73$

2. 对第1题中的(a)乘数组应用MCM迭代匹配算法。每一轮迭代时，找出数组中所有常数对之间的位级匹配数目（见步骤2）。求出应用该算法后需要多少次移位和加法。节省了多少次移位和加法?

583

3. 对第1题中的(b)乘数组重复第2题。

4. 对使用了如下线性变换矩阵的计算 $\mathbf{y} = \mathbf{T}\mathbf{x}$ 应用子表达式消除:

$$\mathbf{T} = \begin{bmatrix} 13 & 12 & 5 & 3 \\ 11 & 2 & 7 & 11 \\ 6 & 9 & 15 & 7 \\ 6 & 11 & 5 & 15 \end{bmatrix}$$

假设使用无符号型二进制表示。求出:

(a) 每一列的子表达式;

(b) 特定的积表达式;

(c) 二进制格式的 y_1、y_2、y_3 和 y_4，其中各位的位置标志着积表达式的存在与否。对 y_1、y_2、y_3 和 y_4 应用迭代匹配算法，计算应用子表达式消除之后实现T所需要移位和加法的总数。相比没有应用共享的T的实现，节省了多少次移位和加法?

5. 对下面的线性变换矩阵（也就是公式（15-8）所用的）重复第4题:

$$\mathbf{T} = \begin{bmatrix} 11 & 3 & 6 & 0 & 5 \\ 13 & 7 & 11 & 0 & 0 \\ 7 & 15 & 0 & 5 & 0 \end{bmatrix}$$

假设使用无符号型二进制表示。

6. 考虑如下的由YUV色彩格式向RGB色彩格式的转换:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U-128 \\ V-128 \end{bmatrix}$$

图像中的每个像素需要4次乘法和6次加法。所有色彩转换算法的效率有以下两个标准:实现复杂度和转换得的图像的色彩质量[8]。为了降低实现的成本，这个变换矩阵近似等于:

$$\begin{bmatrix} 1 & 0 & 1.5 \\ 1 & -0.375 & -0.75 \\ 1 & 1.5 & 0 \end{bmatrix}$$

其中每一个系数用无符号型二进制表示时，最多包含两位1。利用系数的无符号表示，涉及负系数的计算用减法实现。假设像素字长为8，设计一个应用了子结构共享的色彩格式转换器。请注意U和V减128的操作只需要改变最高位的值即可，而不需要当作一次计算减法操作。

584

7. 用下面的近似变换矩阵重复第6题:

$$\begin{bmatrix} 1 & 0 & 1.375 \\ 1 & -0.34375 & -0.6875 \\ 1 & 1.75 & 0 \end{bmatrix}$$

其中每个系数的二进制表示包含不多于3位的1。

8. 对下面的多项式应用子表达式消除:

(a) $x^{22}+x^{19}+x^{14}+x^{7}+x$;

(b) $x^{29}+x^{23}+x^{20}+x^{15}+x^{9}+x$;

(c) $x^{23}+x^{17}+x^{8}+x^{5}+x^{4}+x$;

分别求出应用子表达式消除前后，这些多项式求值需要的乘法的次数。

9. 考虑这样一个递归求值的实现:

$$y(n)=\frac{37}{64}y(n-2)+\frac{5}{64}y(n-3)+\frac{1}{4}x(n) \tag{15-38}$$

使用二进制补码的定点表示和子结构共享，设计字长为8的位串行实现。

10. 为例15.4.2那种基于子结构共享的3轮FIR滤波器设计一个位串行结构，假设信号字长为8。

11. 本题是有关例15.4.4中FIR滤波器的位串行实现。假设信号字长为8。对于这个例题两个设计的每一个，插入合适的流水线割集以得到可实现的位级流水线位串行结构。在合适的地方插入同步延时单元。比较在这两种结构中包括同步延时在内所需延时元件的数量。注意: 在割集中插入的同步延时是位级延时，而数据流图的延时则是字级延时。求解时不需要给出详细的位串行设计。

12. 分析如下的5轮FIR滤波器:

$$y(n)=c_0x(n)+c_1x(n-1)+c_2x(n-2)+c_1x(n-3)+c_0x(n-4) \tag{15-39}$$ 585

其中:

$$c_0=0.00010110,\quad c_1=0.01001100,\quad c_2=0.01100110$$

用2的补码表示。输出的计算式如下:

$$y(n)=c_0(x(n)+x(n-4))+c_1(x(n-1)+x(n-3))+c_2x(n-2)$$

(a) 用CSD表示系数;

(b) 应用CSD表示和子结构共享，求出字长为9的该滤波器的位串行实现。不需要完成详细设计，也就是说不需要给出开关时刻。在本部分中，设定子结构共享只允许对每个单独的系数使用，即不准在系数之间进行共享;

(c) 利用系数间子结构共享，重复(b)的运算;

(d) 评估(b)和(c)得到的设计。

13. 应用CSD表示以及两个最常见的子表达式和子结构共享，为下面的计算设计位串行结构:

$$y(n)=\frac{83}{128}y(n-2)+\frac{37}{64}y(n-3)-\frac{83}{128}y(n-4)+x(n)$$

828

假设信号和系数字长都为8。

14. 一个长度为15的线性相位FIR滤波器的系数的2的补码二进制表示如下:

$$
\begin{array}{lll}
c_0 = 0.00000010, & c_1 = 0.00001001, & c_2 = 1.11111111 \\
c_3 = 1.11110000, & c_4 = 1.11110010, & c_5 = 0.00010110 \\
c_6 = 0.01001100, & c_7 = 0.01100110, & c_8 = 0.01001100 \\
c_9 = 0.00010110, & c_{10} = 1.11110010, & c_{11} = 1.11110000 \\
c_{12} = 1.11111111, & c_{13} = 0.00001001, & c_{14} = 0.00000010
\end{array}
$$

(a) 用CSD表示所有系数;

(b) 本部分的目标是用CSD系数和子结构共享实现这个FIR滤波器; 用加法、延迟和移位画出该滤波器; 应用基于两个最常见CSD子表达式的子结构共享减少加法的次数; 通过使FIR滤波器的系数对称来进一步降低硬件复杂度。要用最少次数的加法、延迟和移位来实现, 然后算出你的电路加法、延迟和移位操作的次数。

586

15. 本题是关于一个应用Parks-McClellan方法的7轮等波纹线性相位低通FIR滤波器的实现。使用MATLAB中的remez命令设计该滤波器, 其通带边缘为0.2π, 阻带边缘为0.25π。所得的12位滤波器系数如下:

$$
\begin{aligned}
c_0 &= c_6 = 0.0894 = 0.00010110111 \\
c_1 &= c_5 = 0.3004 = 0.01001100111 \\
c_2 &= c_4 = 0.1606 = 0.00101001000 \\
c_3 &= 0.2544 = 0.01000001001
\end{aligned}
\tag{15-40}
$$

由于线性相位的特性, 这个7轮滤波器可以用4次乘法操作完成。请用系数的CSD表示和两个公共子表达式实现该滤波器。还必须利用基于两个公共子表达式的结构的子表达式共享。计算你的实现需要加法器的个数。不用子表达式共享的2的补码实现需要多少个加法器?

16. 考虑一个.8比例的DCT设计, 其中矩阵系数都乘上 $\sqrt{2}$ 。这个比例DCT可以采用公式(9-53)和(9-54), 用20次乘法操作实现(因为 $\sqrt{2}c_4 = 1$)。该DCT可以用如图15-7中的(a)和(b)所示的两个基本单元实现, 其中 $c_i = \cos\frac{i\pi}{16}$ 。

(a) 在含12位小数的14位字长CSD数字系统中表示 $\sqrt{2}c_2$, $\sqrt{2}c_6$, $\sqrt{2}c_1$, $\sqrt{2}c_3$, $\sqrt{2}c_5$, $\sqrt{2}c_7$;

(b) 使用 $x + x >> 2$ 和 $x - x >> 2$ 两个公共子表达式实现图15-7中的(a)和(b)的结构, 要尽量减少加法器的个数;

(c) 计算.8DCT实现加法器和乘法器的总数。不需要画出完整的DCT结构。

17. 应用基于行的加性数字拆分, 试找出第5题矩阵变换的一种低复杂度实现。

18. 应用基于行的加性数字拆分, 找出下式的一种低复杂度实现:

$$
\begin{bmatrix} y_0(n) \\ y_1(n) \\ y_2(n) \end{bmatrix} = \begin{bmatrix} 0.875 & 0.5 & 0.90625 \\ -0.5 & 0.6875 & 1.1875 \\ 0.125 & 0.68625 & 0.625 \end{bmatrix} \begin{bmatrix} x_0(n) \\ x_1(n) \\ x_2(n) \end{bmatrix}
$$

设定所有数字用定点2的补码表示。

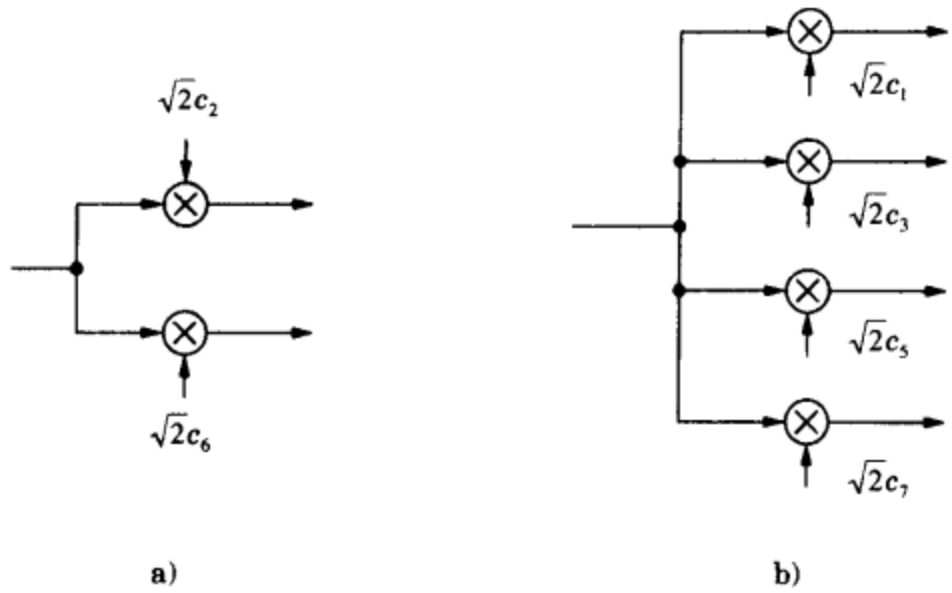


图15-7 第16题的DCT基本单元

19. 应用基于列的加性数字拆分，找出下式的一种低复杂度实现:

$$\begin{bmatrix} y_0(n) \\ y_1(n) \\ y_2(n) \end{bmatrix} = \begin{bmatrix} 0.8125 & 1.1875 & 0.875 \\ 0.25 & 0.4375 & 0.75 \\ 0.9375 & 0.1875 & 0.75 \end{bmatrix} \begin{bmatrix} x_0(n) \\ x_1(n) \\ x_2(n) \end{bmatrix}$$

设定所有数字用定点2的补码表示。

20. 本题考虑对如下的变换应用基于行和列的乘性数字拆分，从而得到低复杂度实现:

$$\begin{bmatrix} y_0(n+1) \\ y_1(n+1) \\ y_2(n+1) \end{bmatrix} = \begin{bmatrix} 1.2 & 1 & -0.6 & 0.6 \\ 0.92 & -1.28 & 0.6 & -0.6 \\ 1 & -1.36 & 0.18 & 0 \end{bmatrix} \times \begin{bmatrix} y_0(n) \\ y_1(n) \\ y_2(n) \\ u(n) \end{bmatrix}$$

用T表示初始的变换矩阵，它需要6次复杂的乘法操作。找出一个3 × 3的矩阵 $\mathbf{T}_1$ 和一个4 × 4的矩阵 $\mathbf{T}_3$，使得 $\mathbf{T} = \mathbf{T}_1\,\mathbf{T}_2\,\mathbf{T}_3$，并且应用 $\mathbf{T}_1\,\mathbf{T}_2\,\mathbf{T}_3$ 只要3次乘法和一些简单的移位操作实现原变换矩阵。

最初，$\mathbf{T}_1$ 和 $\mathbf{T}_3$ 都是单位矩阵，$\mathbf{T}_2$ 等于 $\mathbf{T}$。按如下顺序对 $\mathbf{T}_1$ 进行列变换和对 $\mathbf{T}_3$ 进行行变换，得到只需要3次复杂乘法的 $\mathbf{T}_2$: 587~588

(1) 变换 $\mathbf{T}_1$ 的第1列:

$$C(1) = C(1) - 1 \times C(2) \tag{15-41}$$

(2) 变换 $\mathbf{T}_3$ 的第4行:

$$R(4) = R(4) + 2 \times R(1) \tag{15-42}$$

(3) 变换更新的矩阵 $\mathbf{T}_3$ 的第1行:

$$R(1) = R(1) - 1 \times R(2) \tag{15-43}$$

(4) 变换更新的矩阵 $\mathbf{T}_3$ 的第4行:

$$R(4) = R(4) - 1 \times R(3) \tag{15-44}$$

(5) 变换更新的矩阵 $\mathbf{T}_3$ 的第2行:

$$R(2) = R(2) - \frac{1}{2} \times R(3) \tag{15-45}$$

得到的矩阵 $\mathbf{T}_1$ 和矩阵 $\mathbf{T}_3$ 如下:

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_3 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -0.5 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & -1 & 1 \end{bmatrix} \tag{15-46}$$

请注意积 $\mathbf{T}_1\ \mathbf{T}_2\ \mathbf{T}_3$ 一直等于$\mathbf{T}$不变。每一次对 $\mathbf{T}_1$ 的列操作都要对 $\mathbf{T}_2$ 做相应的行操作; 每一次对 $\mathbf{T}_3$ 的行操作都要对 $\mathbf{T}_2$ 做相应的列操作。写下和上面对 $\mathbf{T}_1$ 和 $\mathbf{T}_3$ 进行的上述列或行线性变换对应的对 $\mathbf{T}_2$ 的行或列变换，并清楚地给出每一步得到的矩阵 $\mathbf{T}_2$。最终得到的矩阵 $\mathbf{T}_2$ 为:

$$\mathbf{T}_2 = \begin{bmatrix} 0 & 1 & 0.5 & 0.6 \\ 0.28 & 0 & 0 & 0 \\ 1 & -0.36 & 0 & 0 \end{bmatrix} \tag{15-47}$$

参考文献

1. M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 2, pp. 151–165, Feb. 1996.

589

2. A. Chatterjee, R. K. Roy, and M. A. d'Abreu, "Greedy hardware optimization for linear digital circuits using number splitting and refactorization," *IEEE Trans. on VLSI Systems*, vol. 1, no. 4, pp. 423–431, Dec. 1993.
3. R. I. Hartley and K. K. Parhi, *Digit-Serial Computation*. Kluwer, 1995.
4. M. Potkonjak, M. B. Srivastava, and A. Chandrakasan, "Efficient substitution of multiple constant multiplications by shifts and additions using iterative pairwise matching," in *DAC-94, Proceedings of the 31st ACM/IEEE Design Automation Conference*, pp. 189–194, 1994.
5. A. Chatterjee and R. K. Roy, "An architectural transformation program for optimization of digital systems by multi-level decomposition," in *30th ACM/IEEE Design Automation Conference*, (Atlanta, GA), pp. 343–348, May 1993.
6. H. Nguyen, A. Chatterjee, and R. K Roy, "Activity measures for fast relative power estimation directed numerical transformation for low power DSP synthesis," *Journal of VLSI Signal Processing Systems*, vol. 18, no. 1, pp. 25–38, Jan. 1998.
7. A. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.
8. B. Gordon, N. Chaddha, and T. Meng, "A low-power multiplierless YUV to RGB converter based on human vision perception," in *Proc. of IEEE Workshop on VLSI Signal Processing*, (San Diego, CA), pp. 408–417, 1994.

590

第16章 同步流水线、波流水线和异步流水线

16.1 引言

信号/图像处理应用对高性能低成本的数字系统的需求极大地推动了高速VLSI设计的进展。传统上，通过将组合逻辑分成许多由全局时钟控制的寄存器/锁存器所分隔的级来实现高吞吐量。根据应用类型，可以使用各种时钟风格，包括两相时钟、真单相时钟等。随着工艺技术的进步，能够集成到单个芯片上的晶体管数目极大地增加了。同时，芯片工作的时钟频率也高速增长。因而，时钟分布和时钟歪斜最小化成为同步系统设计中的主要瓶颈。事实上，时钟歪斜可能增大危险的比例和引起电路故障。因此，一些替换的技术正在研究中，其中最重要的是波流水线[1]~[6]和异步流水线[7]~[18]。

波流水线是一种用于减少数字系统中有效流水线级数而不增加流水线中物理寄存器数目的技术。这是通过在输出（以前的输入数据产生的）得到之前将新的数据施加到组合逻辑的输入来实现的，从而将组合逻辑流水线化。施加新的数据的时间是通过进行详细的系统级和电路级分析来确定的。在系统级，分析应该考虑到时钟周期、故意的时钟歪斜和全局时钟延时。在电路级，要求对组合逻辑电路的最大和最小延时有精确的估计。

异步流水线是一种系统内没有全局同步的技术；系统内的子系统有局部同步，子系统间通过通信协议实现通信。涉及子系统间一些请求/应答握手协议的系统通常叫作自定时系统。因为大多数实际的异步系统是自定时系统，所以在文献中，这些术语表达的是同一个意思。在异步系统中，子系统产生的结果一产生就可以被其他子系统消耗，而不必要等到一个全局时钟的变化。因此，异步系统受限于子系统平均情况下的性能而不是最坏情况下的性能（如在同步系统中）。而且，一个子系统可以很容易地用另一个功能相同而性能不同的子系统代替。然而，在同步系统中，这种替换是不容易做到的，因为时钟周期可能需要重新计算。最后，在一个特定的子系统中，只有当它要处理数据时才产生转换，因此，当它处在空闲状态时没有动态功耗。这使得异步系统对于低功耗设计具有吸引力。然而，握手电路会增加异步电路的功耗。

本章首先对同步流水线和可能用到的各种时钟风格进行详细的讨论。然后，讨论时钟歪斜和时钟分布及其最小化的问题。接着是关于克服时钟歪斜和时钟分布问题的更多的最新技术的一个全面概括，包括波流水线和异步流水线。本章的组织如下：第16.2节简单描述同步流水线和各种可能用到的时钟风格。第16.3节讨论时钟歪斜和时钟分布问题，并给出用于最小化时钟歪斜和时钟分布的一些技术。波流水线的理论背景在第16.4节中进行讨论。关于波流水线级数及其实际意义的细节在第16.5节中讨论。在第16.6节，给出一种针对使用标准工艺互补传输晶体管逻辑（normal process complementary pass transistor logic）（NPCPL）的特定电路的波流水线算法。在第16.7节中，对异步设计风格做一个简单的介绍，

其中，对各种类型的通信协议，包括捆绑数据协议、双轨协议、2相和4相等，都进行了讨论。第16.8节给出了关于设计4相握手电路的信号转换图（STGs）的介绍。第16.9节介绍将STGs应用于设计互连电路。第16.10节给出几个异步电路的设计实例。

592

16.2 同步流水线与时钟风格

考虑如图16-1a所示的一种典型的同步非流水线系统。其中，系统由被边沿触发的寄存器所环绕的单个组合逻辑模块组成。在每个时钟周期开始时，数据由输入寄存器加入到组合逻辑模块。

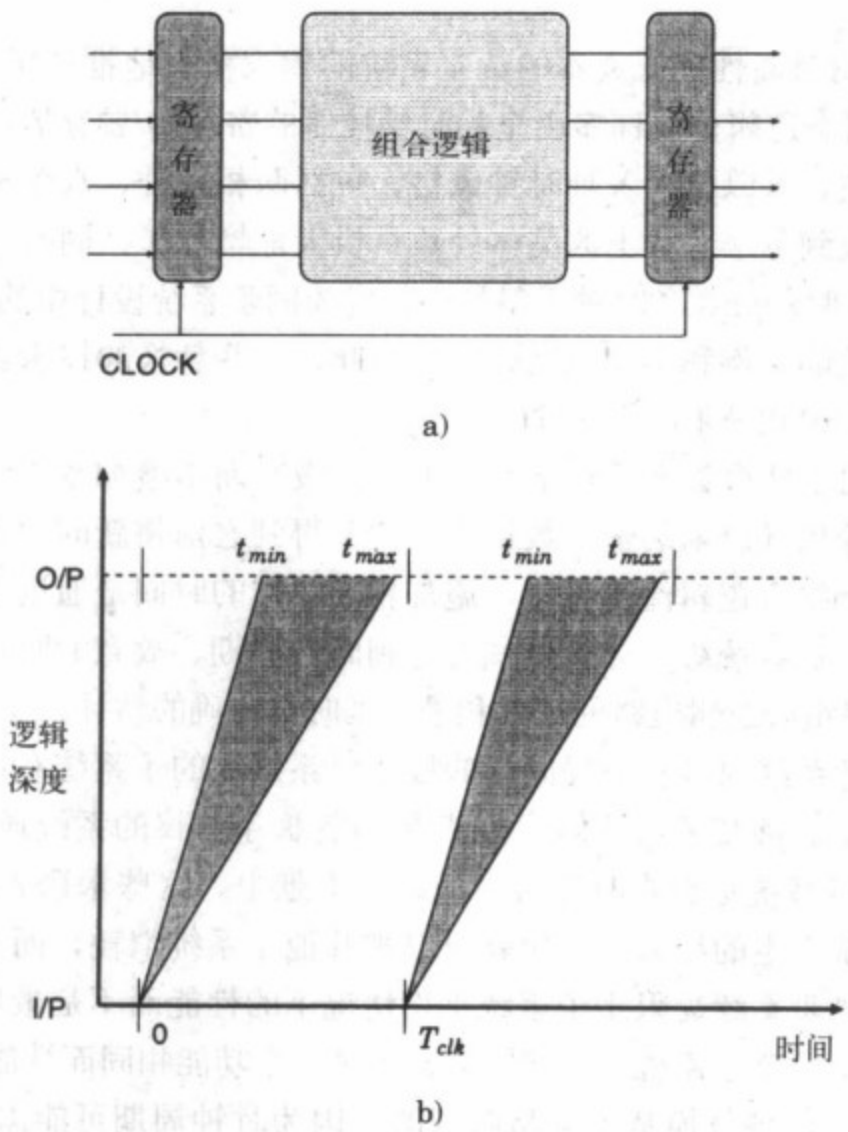


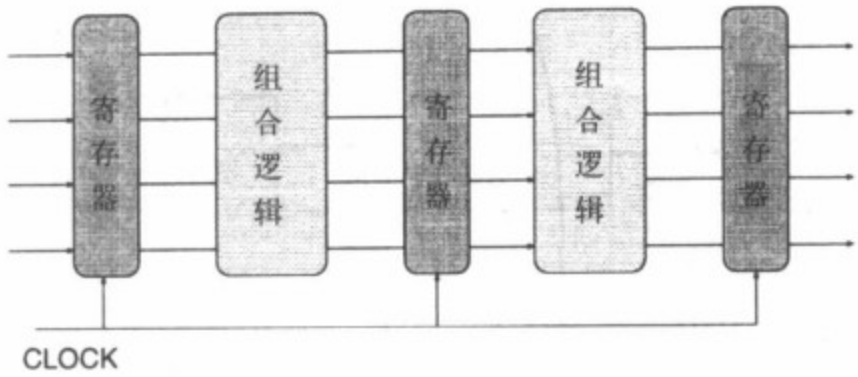
图16-1 a) 同步非流水系统 b) 数据流

由于电路路径长度和其他因素的不同，通过组合逻辑模块的延时可能不同。延时限定于t_{min}和t_{max}之间，如图16-1b所示。其中，阴影区域描述数据通过组合逻辑模块的流程和逻辑的某些部分变化时的时间，而非阴影区域描述逻辑固定时的时间。因此，输出寄存器在最迟的数据到达组合逻辑模块的输出后（考虑到寄存器建立时间）的非阴影区域由时钟将输出数据锁存。同时新的输入数据由时钟锁存到输入寄存器并加入到组合逻辑。因此，最小的时钟周期由输入和输出寄存器间的组合逻辑和传播的最长延时、寄存器的建立时间及任何非故意的时钟歪斜之和来限定。

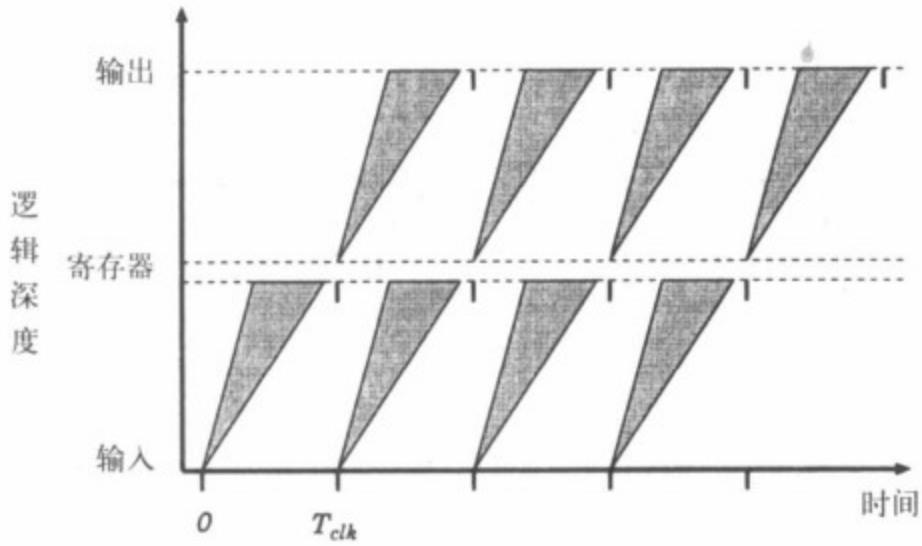
593

传统的缩短系统时钟周期的方法是对组合逻辑进行流水线操作，如图16-2a所示。其中，组合逻辑被分成了两部分，在它们之间插入了一个寄存器。结果，每个运算用两个时钟周期

来传播通过该系统，而每个时钟周期都可以在输出寄存器得到一个运算输出。寄存器的插入减少了组合逻辑模块的最大延时，因而整个系统的时钟周期得以降低。



a) 同步流水系统



b) 数据流

图 16-2

在同步流水线系统中，可以将各种时钟风格用于寄存器的时钟系统。两相时钟系统用于大多数的早期数字系统设计中。这种时钟风格的主要思想就是用两相互不交迭的时钟 ϕ_1 和 ϕ_2，如图16-3所示。时钟 Φ_1 通常称作主时钟，Φ_2 称作从时钟。这两个时钟都满足一个重要特性，就是它们不会同时为高电平。然而，它们可以有有限的非交迭（underlap）时间，如图16-3所示。同步寄存器可以通过主-从配置方式级连2个锁存器来实现，如图16-4所示。其中，第一个锁存器用 Φ_1 作为锁存时钟，第二个锁存器用 Φ_2 作为锁存时钟。非交迭时钟信号保证输入信号D不存在直接通到Q的竞争。时钟信号非交迭的性质可以用相位有效和相位稳定的概念，以一种更正式的方式来表示。 594

如果一个信号 X 在 Φ_i 的下降沿附件是稳定的，则称它在相位 Φ_i 有效（$[V\Phi_i]$），如图16-5a所示。如果一个信号Y在 Φ_i 为高电平的整个期间是稳定的，则称它在相位 Φ_i 稳定（$[S\Phi_i]$），如图16-5b所示。显然，一个在相位 Φ_i 稳定的信号自动满足在相位 Φ_i 有效，因为前者是更严格的限制。如果一个信号X是 $[V\Phi_i]$ 的，则它可以直接连接到用 Φ_i 相位作为时钟的D类型锁存器的D输入端，因为 $[V\Phi_i]$ 特性保证了可以满足建立/保持时间的要求。可以直观地看出，如果用 Φ_1 作为时钟的D锁存器的输入是 $[V\Phi_1]$ 的，则该锁存器的输出自动满足是 $[S\Phi_2]$ 的。$[V\Phi_i]$ 和$[S\Phi_i]$的概念在精确同步系统的设计中是非常有用的。

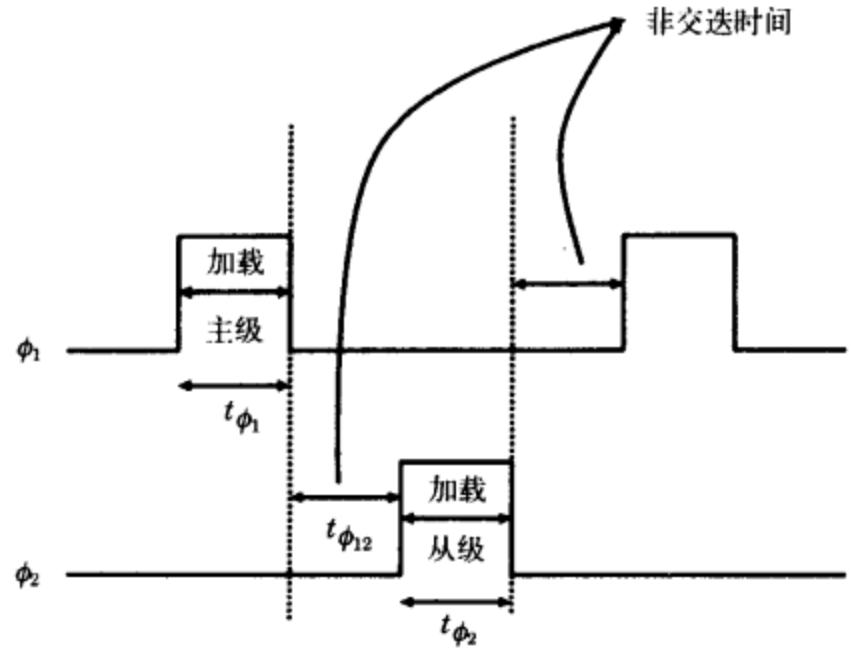


图16-3 两相时钟策略

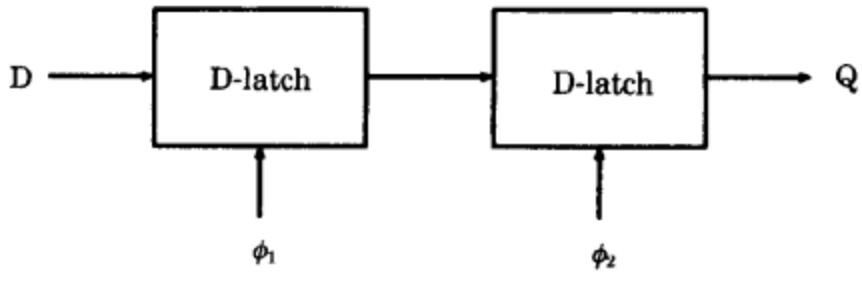
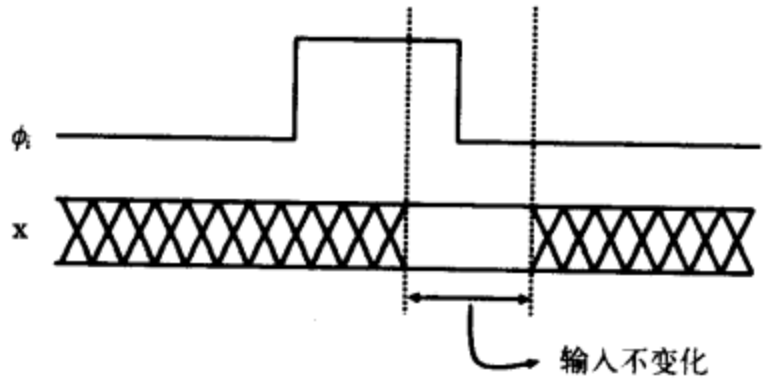
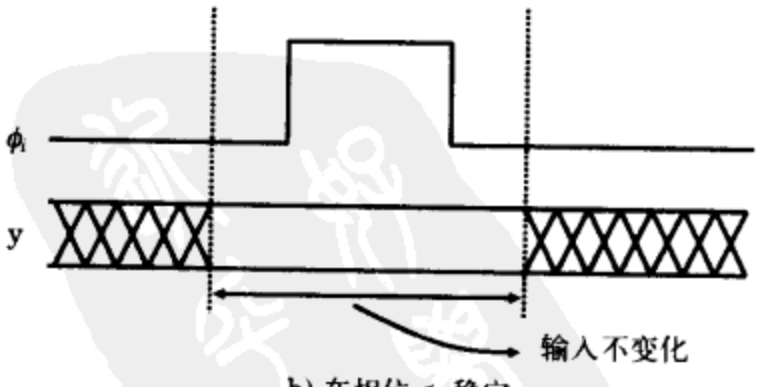


图16-4 一种典型的同步寄存器



a) 在相位 ϕ_i 有效



b) 在相位 ϕ_i 稳定

图 16-5

如图16-4所示的D锁存器可以用静态逻辑或动态逻辑来实现。一种静态锁存器的结构如图16-6所示，它由8个晶体管组成。包括两个传输门分别编号为1和2，一个用作为时钟，另一个用 $\overline{\phi}_1$（ϕ_1的逻辑补）作为时钟。当 ϕ_1 为高时，编号为1的传输门导通，输入信号D通过

两个反相器传输到输出端。当 ϕ_1 为低时，传输门1关断，传输门2导通，因此输出信号Q在锁存器内循环而保持不变。

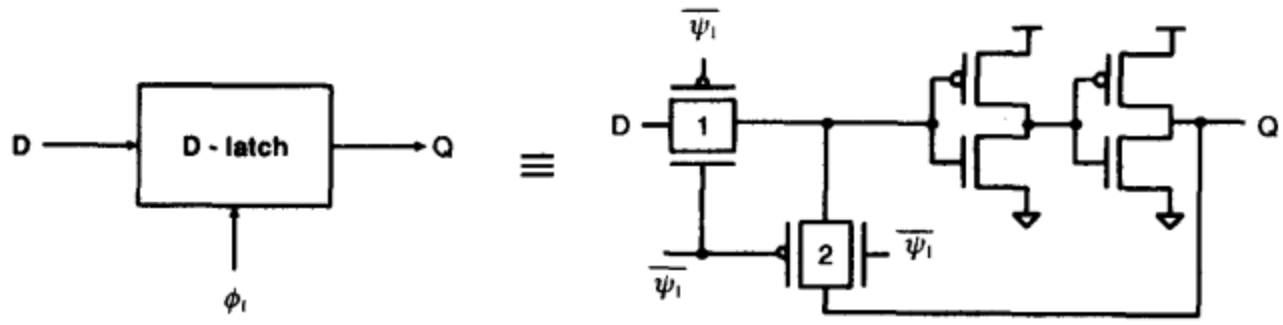


图16-6 静态D锁存器

一种动态D锁存器的结构如图16-7所示，它由6个晶体管组成，可产生Q和$\overline{Q}$。该锁存器与静态锁存器的主要区别是它的输出是不反馈回去的。只有一个用 ϕ_1 作为时钟的传输门位于输入端。当 ϕ_1 为高时，传输门导通，输入信号的真值和补码形式都传输到输出端。当 ϕ_1 为低时，传输门关断，输出端的电容保存D的上一个值。尽管动态锁存器用的晶体管数比静态锁存器少，但是输出电容上的电荷只能保持很短的一段时间，因此输出需要周期地进行刷新。 595～596

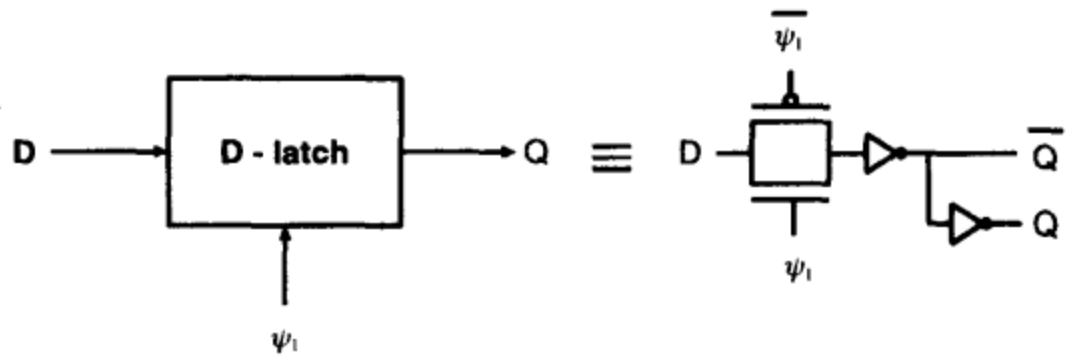


图16-7 动态D锁存器

尽管两相时钟地想法听起来很简单，但设计者必须要确保两相时钟 ϕ_1 和 ϕ_2 永不交迭。这在有些时候是不容易做到的，因为在较大的设计中，时钟线上的延时不能忽略。而且，反相时钟 $\overline{\phi}_1$ 和 $\overline{\phi}_2$ 也要求永远不能同时为低电平。因此，这些时钟不能简单地用它们相应的正时钟求反来设计。一种有效的由单个外部时钟产生正时钟和反时钟的设计如图16-8所示。该电路确保了 ϕ_1 和 ϕ_2 永不同时为高，$\overline{\phi}_1$ 和 $\overline{\phi}_2$ 永不同时为低。假设初始时 ϕ 和 ϕ_2 为1，ϕ_1为0。当 ϕ 发生从1到0的转变时，下面的或非门的输出变为0，因此 ϕ_2 变为0。此时，上面的或非门的输入变为全0，因此它的输出变为1。这使得经过一定的延时后 ϕ_1 变为1。同理，当 ϕ 发生从0到1的转变时，上面的或非门的输出变为0，从而使得 ϕ_1 变为0。这使得下面的或非门的所有输入变为0，因此，经过一定的反相器延时后 ϕ_2 也变为1。这样就用一个外部时钟产生了两个互不交迭的时钟。

随着半导体工艺技术的进步，大量的数字电路将集成到一个芯片上。而且，芯片的很大部分面积是寄存器所占用。从而导致用两相时钟策略的时钟分布变得很困难。因此，最近又开发出了一种称为*真单相时钟*（tspc）的新的时钟风格。这种时钟风格的主要特点是仅用一个时钟 ϕ（甚至连 $\overline{\phi}$ 也不需要用）。这样就得到了一种较简单的时钟分布，潜在地提高了速度。一个tspc正（负）沿触发的触发器可以通过级连一个tspc的p锁存器（n锁存器）和一个tspc的n锁存器（p锁存器）来实现，如图16-9所示。

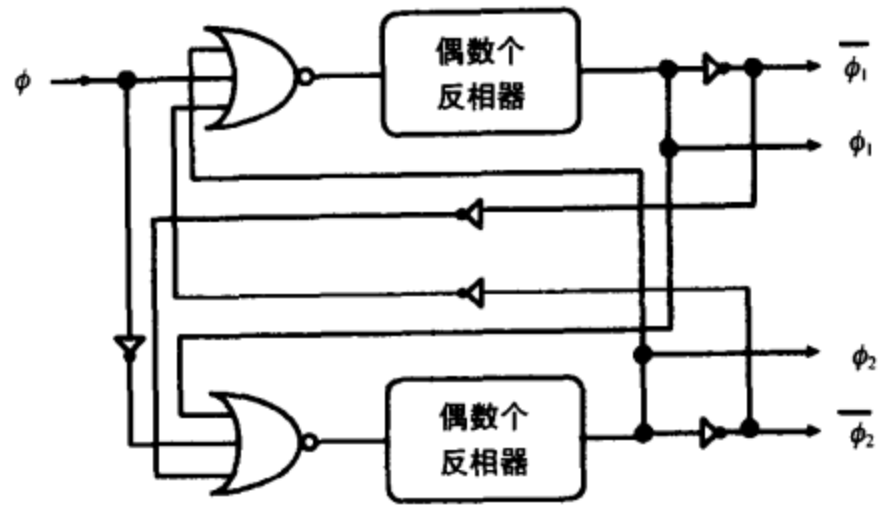


图16-8 两相时钟产生器

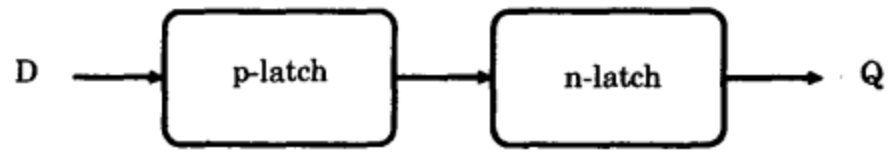


图16-9 正沿触发的tspc触发器

一种tspc的n锁存器如图16-10所示。由图可以清楚地看出，锁存器由6个晶体管和1个单相位时钟信号 ø 所组成。锁存器的操作可以总结成如下两步:

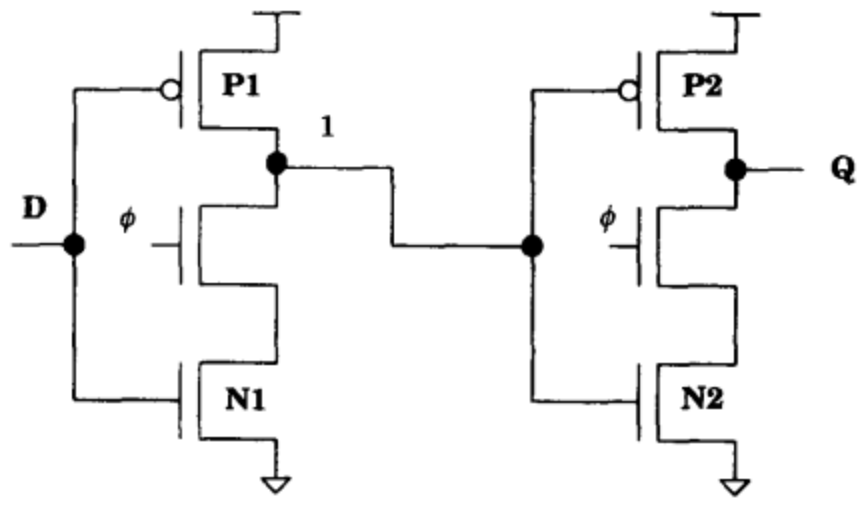


图16-10 一种tspc的n锁存器结构

597 ~ 598

(1) ø = 1: 两个时钟控制的NMOS管都导通。因此，电路的行为就像两个串联的反相器一样，从而锁存器处于穿通状态。于是，输入信号*D*的任何变化都反映到输出信号*Q*。

(2) ø → 0: 这一步分如下两种情况进行分析:

(a) 假设在这个沿之前，输入信号D已经稳定到0。因为ø是0，两个时钟控制的NMOS管截止，没有到GND的通路。然而，由于D为0，P1管是导通的，节点1被拉到Vdd。这又使得P2截止，输出信号Q保持为原来的值，也就是0。这时，假设D变为1，节点1被隔离（因为P1变为关断），它还保持它原来的值Vdd。因此，输出保持不变。

(b) 假设在这个沿之前，输入信号D已经稳定到1。这意味着在这个沿之前，节点1放电到GND，P2管导通，从而输出Q被拉到Vdd。这时，因为ø = 0，节点1被隔离并保持为0，输出Q保持为Vdd。如果D变到0，P1导通，节点1被拉到Vdd。这会

引起P2变为关断，像前面的情况一样，输出Q被隔离并保持其原来的值Vdd。从以上分析中可以清楚地看出，ø=1对应于穿通相位，ø=0对应于隔断相位。

一种tspc的p锁存器的结构如图16-11所示，它由6个晶体管和一个单相位时钟组成。与前面一样，锁存器的功能分如下两步进行分析:

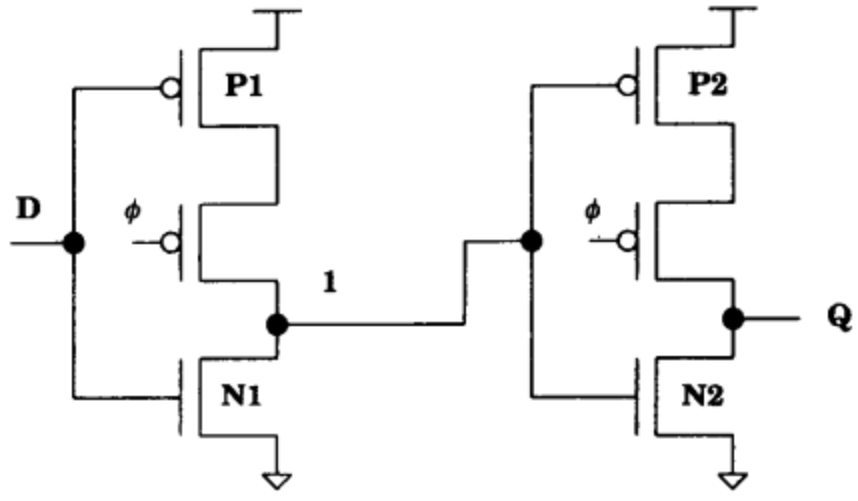


图16-11 一种tspc的p锁存器结构

(1) 假设ø=0。这时，时钟控制的两个PMOS管导通。电路行为与两个串联的反相器一样，输入信号D传输到Q。因此，在这个相位期间，锁存器穿通。 599

(2) 假设ø=1。这又分为如下两种情况:

(a) 假设在这个沿之前，D稳定为0。因为ø→1，两条上拉的路径都被关断。因此，节点1被隔离，其值为1，Q通过N2拉到GND。这时，如果D变为1，N1导通，节点1被放电到GND。虽然这时使得P2导通，但由于它还与一个截止的时钟控制的晶体管串联。因此，Q被隔离，并保持其值为0。

(b) 假设在该时钟沿之前，D稳定为1。这意味着在该时钟沿之前节点1已经被放电到GND，而Q被上拉到Vdd。这时，如果*D*变为0，节点1变成隔离状态而保持为GND，而Q保持其值Vdd。因此，很显然ø=0对应穿通相位而ø=1对应隔断相位。

如果p锁存器和n锁存器级连在一起，tspc寄存器需要12个晶体管。然而，有一种只需要9个晶体管（包括产生$\overline{Q}$）的寄存器的有效实现方法，如图16-12所示。该触发器的操作如下:假设初始时ø=1。这意味着第一级是穿通的，节点1变成等于$\overline{D}$。同时第二级的节点2放电到GND。在第三级，$\overline{Q}$被隔离，存储其原来的值。假设ø变到0，根据D的值，可能出现如下两种情况:

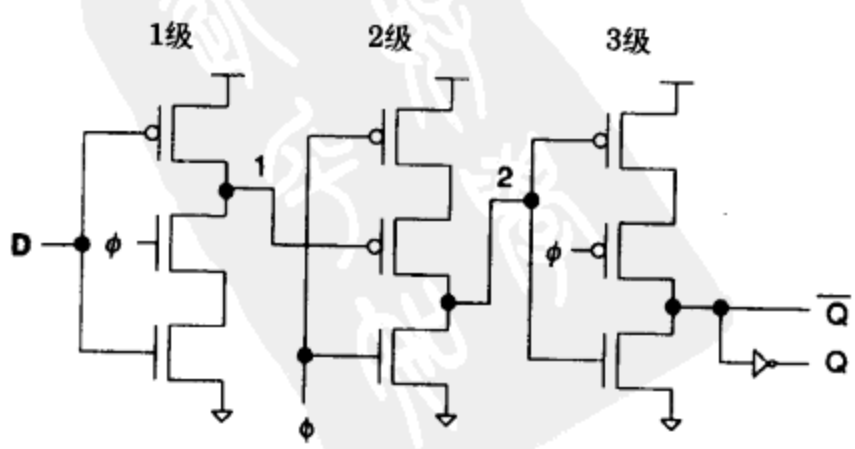


图16-12 9管tspc负沿触发的触发器

(1) 假设ø变为0之前，D为0。ø变为0后，节点1保持在Vdd，而节点2保持为GND。在第三级，上拉路径是导通的，$\overline{Q}$ 被上拉到Vdd。

(2) 假设ø变为0之前，D为1。意味着节点1和节点2将放电到GND，$\overline{Q}$ 将被隔离。这时，一旦ø变为0，第二级的上拉路径导通，节点2被上拉到Vdd。这将使得 $\overline{Q}$ 放电到0。因此，从上述讨论很显然得出所设计的功能是一个负沿触发的触发器。

600

16.3 位级流水线VLSI设计中的时钟歪斜与时钟分布

前一节介绍了可用于流水线系统设计的各种时钟风格。位级流水线设计主要用于高速或低功耗应用，由于用到大量的寄存器，时钟歪斜和时钟分布问题可能很严重。在大系统中，由于时钟线的传输延时，全局时钟到达不同的寄存器所需要的时间会不同。全局时钟到达不同的寄存器的时间的差别叫作时钟歪斜。在本节中，讨论时钟歪斜问题的公式表示和导致时钟歪斜的条件。此外，还讨论用于最小化时钟歪斜的时钟分布技术。

16.3.1 时钟歪斜

在同步系统中，时钟歪斜是一个随着系统的速度和大小而增长的问题。从而，在非常高速的位级流水线系统中，这个问题成了主要的瓶颈。因此在设计位级流水线系统之前，全面地了解时钟歪斜的影响是必要的。此外，时钟歪斜影响系统操作的方式通常依赖于所用时钟风格的类型和数据传输相对于时钟信号的方向。

时钟歪斜问题可借助于图16-13来进行分析[20]。其中，运算逻辑模块（CLB）和寄存器R组成同步系统的1个模块或流水线的1级。根据流水线的深度，每个CLB可以是单个的逻辑门、执行位级操作的全加器或执行字级操作的乘法器。与每个模块相结合，定义如下参数：

t_{sl}，t_{sr}：逻辑单元和寄存器的运算延时，或者叫稳定时间（settling time）。

t_{pl}，t_{pr}：逻辑单元和寄存器的传输延时。

T_{clk}：寄存器时钟周期。

t_{pi}：对应于相邻流水线级之间互连的传输延迟时间。

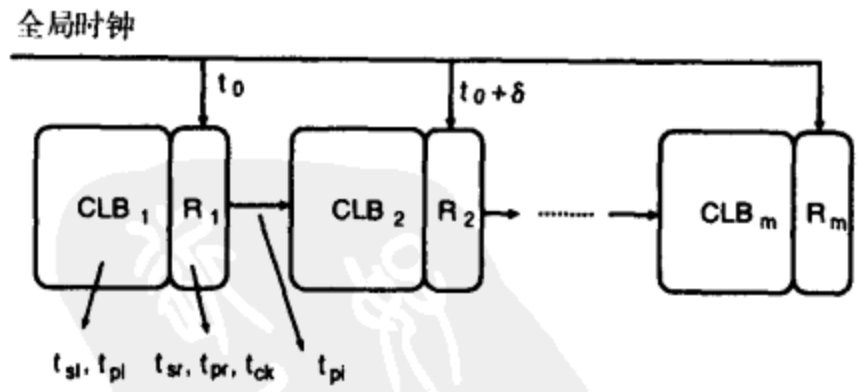


图16-13 时钟歪斜分析的典型流水线系统

传输延时就是当任何一个输入发生变化时，CLB至少有一个输出开始发生转换之前所需要的时间。这个延时通常比稳定时间要短得多，但在时钟歪斜问题上起重要作用。下面讨论两种时钟风格的时钟歪斜问题：边沿触发的单相时钟和两相时钟系统。

16.3.1.1 边沿触发的单相时钟系统

考虑数据在如图16-13所示的模块CLB_1和模块CLB_2之间传输。从图中可以清楚看出，在

时刻 $t = t_0$，CLB_1的输出开始装入寄存器R_1，在时刻 $t = t_0 + \delta$，CLB_2的输出开始装入寄存器R_2，其中，δ表示2个寄存器间的时钟歪斜。此外，在时刻 $t = t_0 + t_{pr} + t_{pi} + t_{pl}$，$R_1$改变的响应一直传输到了$R_2$的输入。然而，$R_2$的装入时间是 $t = t_0 + \delta$，因此，必须满足如下的条件：

$$t_0 + t_{pr} + t_{pi} + t_{pl} > t_0 + \delta \tag{16-1}$$

等效于：

$$\delta < t_{pr} + t_{pi} + t_{pl} \tag{16-2}$$

因此，时钟歪斜必须小于寄存器、互连和逻辑单元的传输延迟时间的总和。通常，δ 可能是正的或负的，取决于时钟的方向与数据流方向的对应关系。因为公式（16-2）中的传输延时总是正的，因此当 δ 为负时，这个条件也应该满足。然而，一个负的 δ 对数字系统的时钟周期 T_{clk} 有很大影响。时钟周期必须足够长，以满足运算的时间。很容易可以算出CLB的运算时间 t_c 如下：

$$t_c = t_{sr} + t_{pi} + t_{sl} \tag{16-3}$$

因此，从图16-14b可以得到负的 δ 对时钟周期的约束表示如下：

$$T_{clk} > t_c + |\delta| = t_{sr} + t_{pi} + t_{sl} + |\delta| \tag{16-4}$$

601
~
602

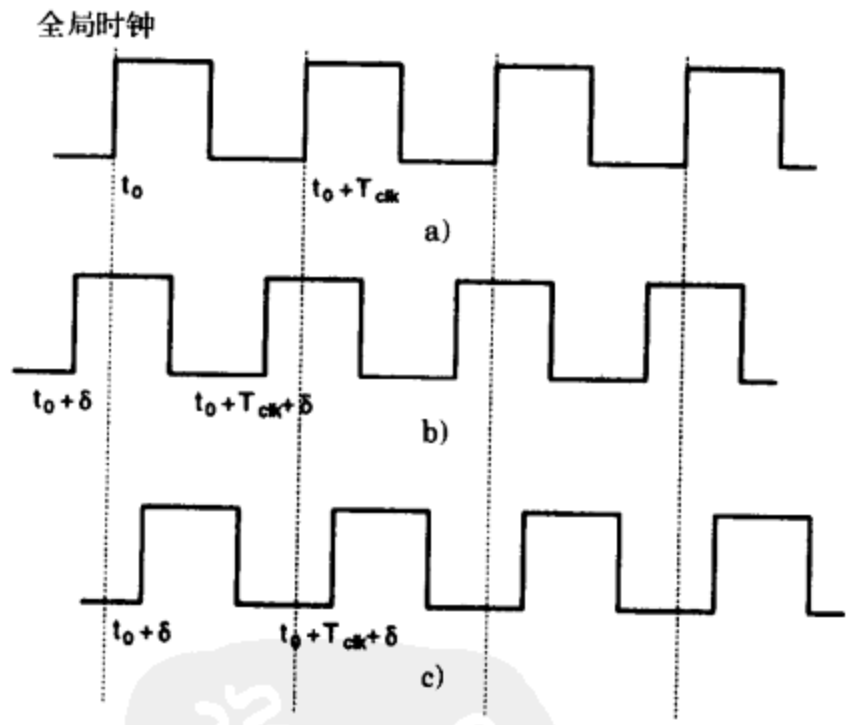


图16-14 a) 寄存器R_1看到的时钟 b) 寄存器R_2看到的 δ 为负值时的时钟 c) 寄存器R_2看到的 δ 为正值时的时钟

类似地，如果 δ 是正的，很显然从图16-14c可以得到对时钟周期的约束变为

$$T_{clk} > t_c - \delta = t_{sr} + t_{pi} + t_{sl} - \delta \tag{16-5}$$

结合公式（16-4）和公式（16-5）可以得到

$$\begin{aligned} T_{clk} &> t_{sr} + t_{pi} + t_{sl} + |\delta|, \quad \delta \leqslant 0 \\ T_{clk} &> t_{sr} + t_{pi} + t_{sl} - \delta, \quad \delta > 0 \end{aligned} \tag{16-6}$$

这意味着，如果时钟歪斜是负的，则公式（16-2）表示的条件总是满足的，假设系统时钟周期足够大，寄存器、逻辑单元和互连的传输延时不会产生不利的影响。然而，如果时钟歪斜是正的，则公式（16-2）的条件必须满足，以防止系统故障。吞吐量可以稍微提高，因为时钟周期可以缩短 δ。另一个有趣的现象是，对于正的 δ，如果公式（16-2）的条件不满足，不管时钟周期是多大，系统将不能正确工作。因此，从前述的讨论可以很清楚知道，如果目标是要最大化吞吐量，则正的歪斜比负的歪斜要好。然而，在通常的同步系统中带有反馈，歪斜既有正的，也有负的。在这种情况下，需要考虑最坏可能的歪斜。

在公式（16-2）中，定义时钟歪斜上限的参数取决于所使用的工艺、CLB的速度和模块之间的互连。对于位级流水线结构，互连传输延时 t_{pi} 通常是很小的，因为数据传输发生在芯片上互相靠近的模块之间。CLB的传输延时 t_{pl} 也很小，因为最大的计算模块通常是一个一位的全加器。如果采用1.25μ的CMOS工艺，$t_{pr}+t_{pi}+t_{pl}$ 的典型值约为1.2纳秒左右。

603

16.3.1.2 两相时钟

考虑图16-3中所示的两相时钟策略，其中 $t_{\phi 1}$ 表示相位1的脉冲宽度，$t_{\phi 2}$ 表示相位2的脉冲宽度，$t_{\phi 12}$ 代表相位1和相位2之间的延迟时间。除了 t_{pr1} 和 t_{sr1} 用于相位1锁存，t_{pr2} 和 t_{sr2} 用于相位2锁存外，其余参数与前一节相同。还假设相位1和相位2的时钟歪斜模式相同。在 $t=t_0$ 时间，CLB_1的输出开始输入寄存器R_1的相位1锁存器，在时间

$$t=t_0+t_{\phi_1}+t_{\phi_{12}}+t_{pr_2}+t_{pi}+t_{pl} \tag{16-7}$$

在R_2的相位1输入可以得到其结果。为了数字系统能正确地工作，这个时间应该大于 $t=t_0+\delta+t_{\phi 1}$。因此，时钟歪斜的条件变为

$$\delta<t_{pr_2}+t_{pi}+t_{pl}+t_{\phi_{12}} \tag{16-8}$$

与单相时钟类似的分析可以得到，时钟周期 T_{clk} 应该满足下面的条件

$$\begin{aligned} T_{clk}&>t_{\phi_1}+t_{\phi_{12}}+t_{sr_2}+t_{pi}+t_{sl}+|\delta|,\quad \delta\leqslant 0\\ T_{clk}&>t_{\phi_1}+t_{\phi_{12}}+t_{sr_2}+t_{pi}+t_{sl}-\delta,\quad \delta>0 \end{aligned} \tag{16-9}$$

而且，还应该有 $t_{\phi 1}>t_{sr1}$ 和 $t_{\phi 2}>t_{sr2}$。两相时钟有趣的一点是，对于正的 δ，总是可以增加 $t_{\phi 12}$ 来满足公式（16-8）的条件，代价是降低了吞吐量。换句话说，不像在边沿触发的单相时钟策略中，在两相时钟方案中，总是可能通过降低时钟频率来防止数字系统失效。参数 $t_{\phi 12}$ 扮演一个可用于控制时钟歪斜的全局参数。

前述的技术可以适用于任何其他时钟策略，来理解位级流水线数字系统操作中的时钟歪斜效应。这些方法已经用于设计非常高速的数字集成电路，包括70-MHz的乘法器[21]和85-MHz的IIR数字滤波器[22]。

604

16.3.2 时钟分布

在同步数字系统中决定时钟歪斜图的因素如下:

- 时钟分布网络中所用互连材料的电阻、电感和电容
- 时钟分布网络的形状
- 芯片或圆片上制造工艺的偏差
- 数字系统中处理单元的数目和每个模块呈现的对时钟分布网络的负载

- 上升、下降时间和时钟信号的频率
- 缓冲策略和时钟缓冲器的使用

为了研究这些影响，可以对时钟分布网络采用分布的RC模型。最广泛研究的最小化时钟歪斜的时钟分布网络是如图16-15所示的H树型网络[23]~[25]。其结构是由一个8×8的大小相同的单元阵列组成。由图中可以清楚看出，由于单元是相同的，它们表现出相同的负载，而且从时钟源到所有单元的路径是相似的，因此时钟歪斜被最小化了。参考文献[26]中表明，对一个$N \times N$的处理单元阵列，与之相关联的时钟脉冲上升时间和时钟歪斜为 $O(N^3)$。因此，随着N的增加，看起来时钟歪斜将急剧增加，从而成为很大的问题。然而，这个问题可以用如图16-16所示的分布的缓冲策略[23]来解决。在H树型网络中引入缓冲器的一个缺点是面积增大了。另一个缺点是设计对工艺偏差的敏感性增加了。不过，这可以通过速度上的提高来弥补。当系统中所有处理单元都相同时H树型网络最有用，时钟可以完全以一种二进制风格分布。另一种适合在片时钟分布可替代H树型网络的是如图16-17所示的2级缓冲策略网络[20]。虽然在理论上这种网络不能提供像理想H树型网络那样的最小化的时钟歪斜特性，但在实际上，它被证明是有用的，因为时钟歪斜最小化的程度依赖于系统能容忍的时钟歪斜的总量。这种技术已经用于设计位级流水线的乘法累加系统[20]。

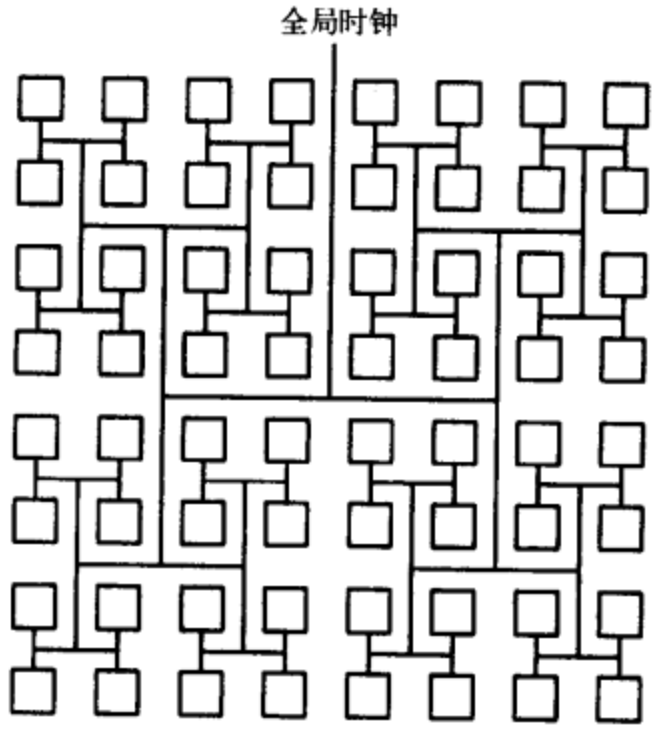


图16-15 8×8单元阵列的H树型分布网络

16.4 波流水线

从上一节可以清楚看出，时钟风格在数字系统的性能方面扮演了很重要的角色。传统方法涉及决定时钟周期的*关键路径*（最长延时的路径）的鉴别。因此，设计者花费了大量的时间试图最小化时钟周期。在数字系统中，通常存在很多延时比关键路径延时小得多的路径。然而，它们还是使用同样时钟周期的时钟，结果这些路径上的逻辑门在时钟周期的大部分时间里保持空闲。这意味着在非关键路径保持空闲的地方有提高时钟速度的空间。*波流水线*[4]的概念就是基于当非关键路径的空闲时间减少时时钟速度可以提高这样的事实。

从图16-1b可以直接看出，输出寄存器的输入在时钟周期的绝大部分时间都是稳定的，意

味着组合逻辑模块没有以它的最大速率操作。那么，有人要问这样一个问题，在当前操作数的结果出现在输出端之前，是否有可能再施加一组新的操作数到组合逻辑的输入端呢？答案是肯定的，事实上时钟周期可以减小到某一特定时钟周期的数据不覆盖前一个时钟周期的数据的程度。这就形成了如图16-18b逻辑深度时序图所示的波流水线的基础，其中，时钟周期被尽可能地减小。现在，制约时钟周期的因素是通过组合逻辑模块最大数据延时（t_{max}）和最小数据延时（t_{min}）的差，连同寄存器建立/保持时间、传输时间和非故意的时钟歪斜。

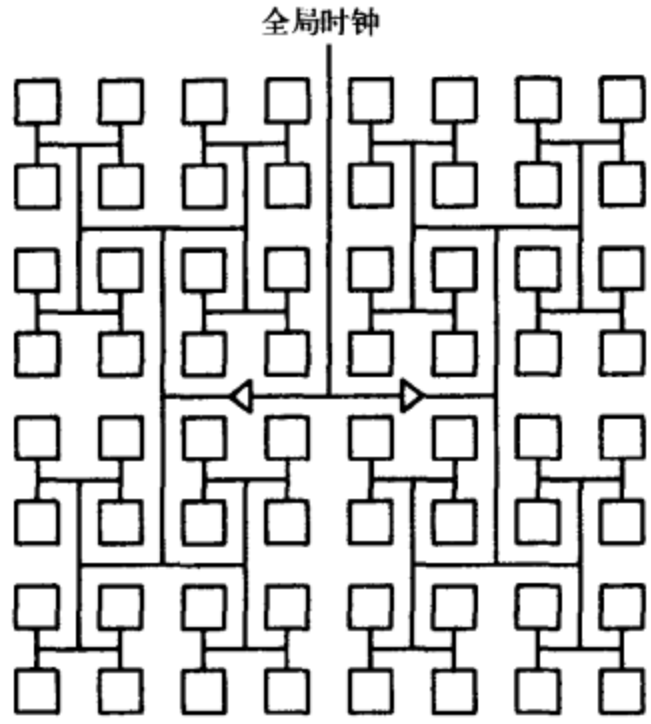


图16-16 带中间缓冲的H树型分布网络

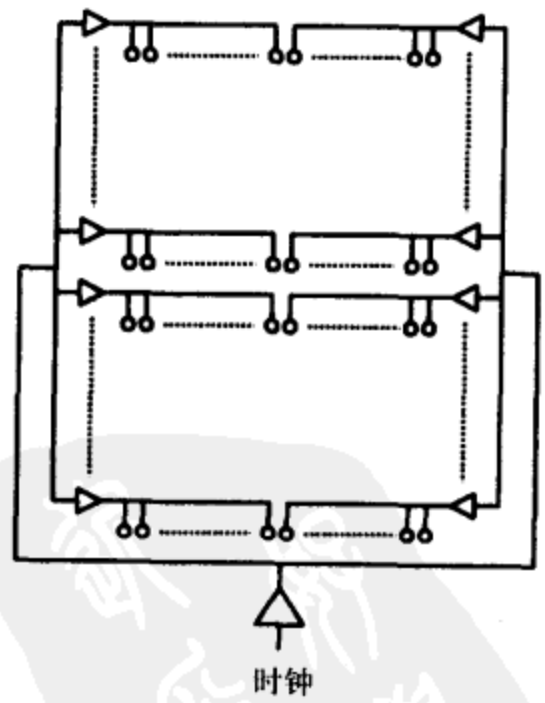
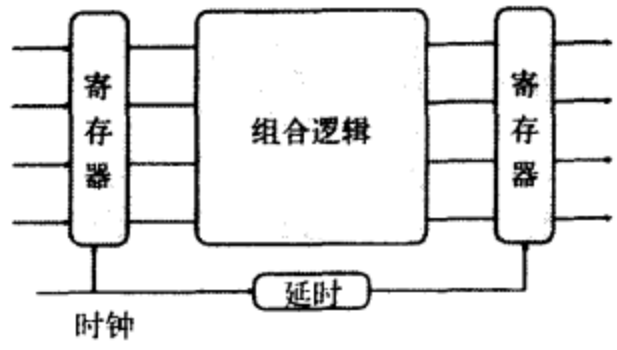


图16-17 两级分布式缓冲策略

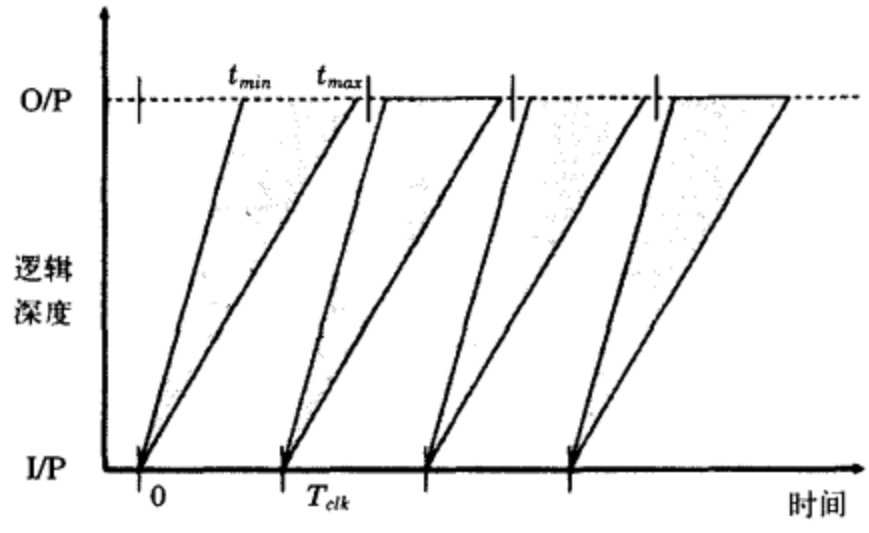
波流水线系统中的时钟周期（T_{clk}）受限于最大和最小组合逻辑延时的差。当然，非理想效应如建立时间和保持时间、输入输出寄存器的时钟歪斜也应该考虑进去。这些效应可以用图16-19b来做最好的说明。其中，t_{min} 和 t_{max} 分别表示最小和最大的组合逻辑延时。建立和保持时间分别用 t_{setup} 和 t_{hold} 表示，而 Δ_i 和 Δ_o 分别表示输入和输出寄存器的内部延时，Δt_e^i 和 Δt_l^i 分别表示输入寄存器最早和最晚可能的时钟歪斜。对输出寄存器，相应的值表示为 Δt_e^o

605 ~ 607

和 Δt_l^o。



a) 波流水线系统



b) 数据流

图 16-18

在图16-19b中，时钟周期选择应该使得输出数据锁存发生在最晚的数据到达输出端后和下一个时钟周期的最早的数据到达之前。输出寄存器在相对于全局时钟的某一个时间 t 锁存设输出数据，则 t 相当于 kT_{clk}，其中 k 叫作全局时钟弛豫[4]。因为输出寄存器本身有一个延时Δ_o，则 t 的条件在数学上可表示为

$$t = kT_{clk} + \Delta_o \tag{16-10}$$

因此，对正确的时钟体系来说，输出数据应该在数据稳定时的时刻 t 被采样。这就意味着必须满足如下的约束:

$$kT_{clk} + \Delta_o - \Delta t_e^o \geqslant \Delta_i + \Delta t_l^i + t_d + t_{max} + t_{setup} \tag{16-11}$$

608

式中 t_d 表示寄存器的延迟时间。而且，为了确保最早的数据正确地锁存，还必须满足下面的约束:

$$\Delta_i + T_{clk} + t_d + t_{min} - t_{hold} - \Delta t_e^i \geqslant kT_{clk} + \Delta_o + \Delta t_l^o \tag{16-12}$$

结合公式（16-11）和（16-12），并重新排列其中的项，得到

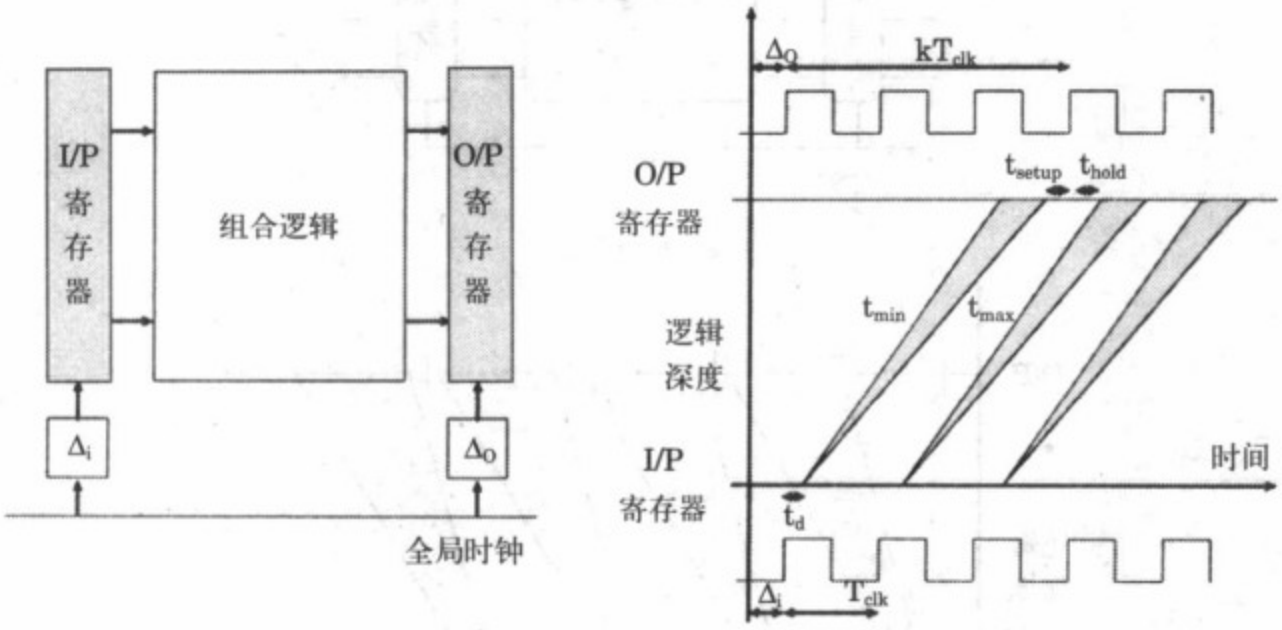
$$\begin{aligned} T_{clk} + t_{min} - t_{hold} - \Delta t_e^i - \Delta t_l^o \geqslant kT_{clk} + \Delta - t_d \geqslant \\ t_{max} + t_{setup} + \Delta t_l^i + \Delta t_e^o \end{aligned} \tag{16-13}$$

其中

$$\Delta = \Delta_o - \Delta_i \tag{16-14}$$

等价于

$$T_{clk} \geqslant (t_{max} - t_{min}) + t_{setup} + t_{hold} + \Delta t_e^i + \Delta t_e^o + \Delta t_l^i + \Delta t_l^o \tag{16-15}$$



a) 波流水线电路　　b) 相应的时序图

图 16-19

从上述表达式可以清楚看出，时钟速率受限于输入输出寄存器间最大和最小延时之差。还说明，非故意时钟歪斜和建立/保持时间直接增加了可能的时钟周期。然而，在传统的流水线中，时钟周期仅受限于最大的运算延时。

609 **例16.4.1** 设 $t_{max}=10$纳秒，$t_{min}=8$纳秒。对传统流水线来说（忽略建立/保持时间），最小可能的时钟周期是$t_{max}=10$纳秒。然而，对波流水线来说，最小可能的时钟周期是$t_{max}-t_{min}=2$纳秒。因此，波流水线工作时，可以比传统流水线快5倍。 ■

对系统中所有内部节点（忽略建立/保持要求）可以相应地推导出类似公式（16-15）的条件

$$T_{clk} \geqslant \left\{ \max_{(i,j)\in C} (t_{max}(i)) - \min_{(i,j)\in C} (t_{min}(i)) \right\} \tag{16-16}$$

其中，C表示组合逻辑模块中所有电路连接的集合。

例16.4.2 考虑图16-20a所示的电路。其中，$G=\{3, 4\}$，$C=\{(1, 4), (2, 3), (3, 4)\}$，$G$表示所有的门输出节点的集合。椭圆形内的项表示 t_{min} 和 t_{max} 的值。

因此，公式（16-16）变为

$$T_{clk} \geqslant max\{t_{max}(1), t_{max}(3)\} - min\{t_{min}(1), t_{min}(3)\} \tag{16-17}$$

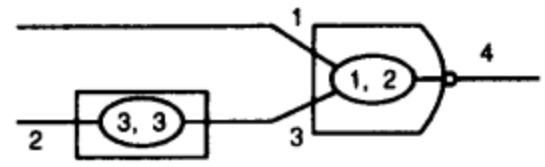
简化后得到

$$T_{clk} \geqslant max\{0, 3\} - min\{0, 3\} = 3 \tag{16-18}$$

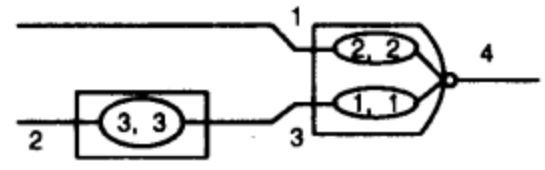
这就意味着为了满足内部节点约束，最小时钟周期必须是3。从图16-20a中还可以看到，从输入到输出所有路径中最小的延时是1，最大的延时是5。因此，输出节点约束是

$$T_{clk} \geqslant \{5-1\} = 4 \tag{16-19}$$

从公式（16-18）和（16-19）可以清楚看出，输出节点约束比内部节点约束更强，因此系统能正确操作所要求的最小时钟周期 $T_{clk}=4$。 ■



a)



b)

图16-20 a)门延时模型电路Ⅰ b)门延时模型电路Ⅱ

例16.4.3 考虑如图16-20b所示的电路。将适当的 t_{min} 和 t_{max} 值代入公式（16-16），得到

$$T_{clk} \geqslant max\{0, 3\} - min\{0, 3\} = 3 \tag{16-20}$$

输出节点约束变为

$$T_{clk} \geqslant \{4-2\} = 2 \tag{16-21}$$

在本例中，内部节点约束强于输出节点约束，因而最小时钟周期为 $T_{clk}=3$。 ■

从公式（16-15）还可以看出，可以通过最小化延时分散或最大延时与最小延时之差 $t_{max}-t_{min}$ 来获得高吞吐量。这意味着，需要平衡通过所有传输路径的延时，使得 $t_{max} \approx t_{min}$。然而，由于下述原因，在大规模电路情况下这是一项艰难的任务：

- 在结构中，由于逻辑深度导致路径偏差
- 上升和下降时间不相等
- 基本架构模块的延时偏差
- 温度和工艺偏差

已经提出了很多方法来抵消这些因素，从粗调/细调到在非关键路径插入延时元件。如果波流水线结构满足下列特征，它就是有效的：

- 对所有的输入组合，逻辑有相同的传输延时和相等的上升/下降时间
- 延时元件对组合逻辑模块的延时仿效得非常完美
- 结构上，在逻辑深度方面有相同的传输路径
- 温度和工艺偏差对结构中所有传输路径有相同的影响

610 ~ 611

16.5 约束空间图与波流水线深度

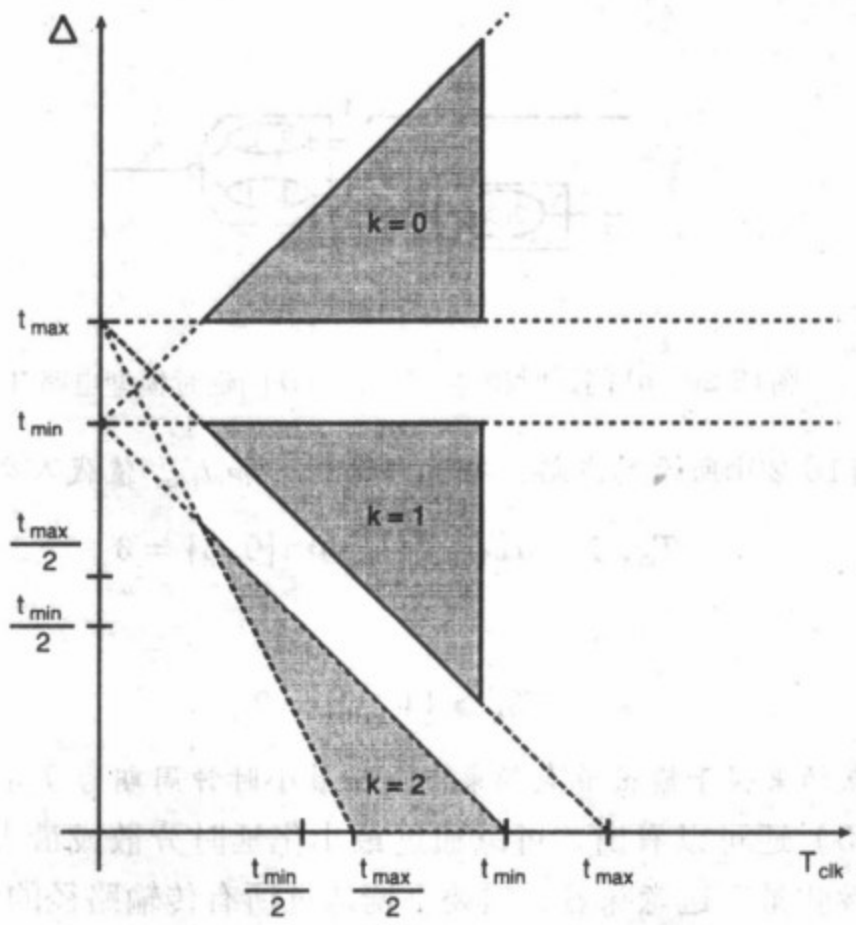
如果忽略建立/保持时间，公式（16-11）和（16-12）所表达的约束可以表示为

$$kT_{clk} + \Delta \geqslant t_{max} \tag{16-22}$$

和

$$T_{clk} + t_{min} \geqslant kT_{clk} + \Delta \tag{16-23}$$

这些公式作为 Δ 和 T_{clk} 的函数的绘图称作时序约束空间图，如图16-21所示。参数 k 称为波流水线深度。从图中可以清楚看出，对固定的 k 值，存在明显的可行操作区域。而且，随着 k 增加，可行区域是减少的。从设计的角度看，一些特殊的 k 和 Δ 值是值得注意的。下面，对这些情形进行分析，同时还研究它们对时钟周期的影响。分析时，使用公式（16-22）和（16-23）的简化约束。



612

图16-21 约束空间图

16.5.1 Δ≤T_{clk}

这些限制迫使设计者在最高的 k 区域操作来获得低的时钟周期。公式（16-22）和（16-23）的简化约束可以等价地表示为

$$kT_{clk} + \Delta \geqslant t_{max} \tag{16-24}$$

和

$$t_{min} \geqslant (k-1)T_{clk} + \Delta \tag{16-25}$$

本节问题的定义导致一个附加的约束

$$T_{clk} \geqslant \Delta \tag{16-26}$$

最后，内部节点的约束导致下面的约束

$$T_{clk} \geqslant t_{internal} \tag{16-27}$$

其中 $t_{internal}$ 表示由于内部节点的约束导致的时钟周期的低限。约束（16-24）和（16-25）合

在一起意味着

$$T_{clk} \geqslant t_{max} - t_{min} \tag{16-28}$$

而约束（16-24）和（16-26）一起意味着

$$T_{clk} \geqslant \frac{t_{max}}{k+1} \tag{16-29}$$

因此，从公式（16-27）、（16-28）和（16-29）可知，对 $\Delta < T_{clk}$ 的单级系统，最小的时钟周期由下面公式给出

$$T_{clk} = max\left(t_{internal}, \frac{t_{max}}{k+1}, t_{max} - t_{min}\right) \tag{16-30}$$

16.5.2 Δ>0

在波流水线系统实际实现中，延时Δ是从输入和输出寄存器的物理延时单位推导出来的，也就是说，$\Delta = \Delta_o - \Delta_i$。这也可以通过移去输入输出寄存器的这些物理延时单位，而以两个寄存器间的单个延时Δ代替来得到。这种系统简化给时钟周期加了一个上限，如下面所述。

4个约束（16-24）、（16-25）、（16-26）和（16-27）的集合变为

$$kT_{clk} + \Delta \geqslant t_{max} \tag{16-31}$$

$$t_{min} \geqslant (k-1)T_{clk} + \Delta \tag{16-32}$$

613

$$\Delta \geqslant 0 \tag{16-33}$$

和

$$T_{clk} \geqslant t_{internal} \tag{16-34}$$

与前面一样，公式（16-31）与（16-32）结合得到如下约束:

$$T_{clk} \geqslant t_{max} - t_{min} \tag{16-35}$$

从公式（16-34）和（16-35），对时钟周期的有效约束由下式给出

$$T_{clk} = max\,(t_{internal}, t_{max} - t_{min}) \tag{16-36}$$

如果附加约束Δ=0加到上述系统，则可以很容易得出对时钟周期的约束减少为:

$$T_{clk} = max\left(t_{internal}, \frac{t_{max}}{k}\right) \tag{16-37}$$

最后，$k=1$，$\Delta=0$的特例就还原到了传统流水线系统的情况，在这种情况下，时钟周期由 t_{max} 制约。

16.6 波流水线系统的实现

在任何波流水线实现中，选择合适的逻辑风格是至关重要的，因为在基本构造模块级传输延时的不匹配可能引起架构失败。使用CMOS工艺是必要的，因为它提供增强的包装密度和降低功耗。然而，静态CMOS不是波流水线的理想选择，因为它的传输延时是高度数据相关的。例如，2输入与非门的上升/下降时间变化可能多达50%，取决于是两个输入信号都为0，

还是只有一个输入信号为0。因此，提出了很多的逻辑风格来设计波流水线结构。本节仅讨论一种叫作NPCPL[2]的这样的风格。

16.6.1 NPCPL

工艺互补传输晶体管逻辑（normal process complementary pass transistor logic, NPCPL）试图利用在一种工艺中只用NMOS传输晶体管逻辑的优点，因而不需要调节传输元件的开启电压。它由以下部分组成:

- 正逻辑和互补逻辑输入
- 两个传输模块，分别产生正逻辑和互补输出变量
- 电平恢复反相器，把在传输模块输出端损失了的电压电平恢复到完全的CMOS电平

614

用于逻辑中的反相器的逻辑阈值可以根据如下公式设定:

$$V_{TH} = \frac{\sqrt{K}(V_{dd} - |V_{Th_p}|) + V_{Th_n}}{1+\sqrt{K}} \tag{16-38}$$

其中 $K=k_p / k_n$，$k_p=k_p'(W/L)_p$，$k_n=k_n'(W/L)_n$。由此得到的值可以作为进一步迭代的起点。细调通过SPICE模拟用实验的方法来实现，而所确定的 W_p / W_n 比率应该使得0→1和1→0的转换延时相等，也就是有相等的上升/下降时间。这个程序是与工艺高度相关的，如果制造工艺改变了，设计程序必须要重做。

在NPCPL设计中，最关键的参数是电平恢复反相器的逻辑阈值电压（V_{TH}），V_{TH} 的增大（降低）会使得NPCPL构造模块的下降时间和转换延时增加（减小），从而影响波流水线电路的性能。通用的NPCPL模块结构如图16-22所示，通过施加合适的输入可以配置成实现不同的布尔函数。这个模块的突出特点是它可以实现所有主要的2输入布尔函数，并具有相同的传输延时。由于它的结构对称，对所有逻辑门提高相同的逻辑深度，因而有相等的延时。而且，它还可以用作延时元件，来模拟其他2输入逻辑门的延时。然而，与静态CMOS相比，NPCPL有噪声容限小的问题。这可以看作是得到相同逻辑深度的折中。

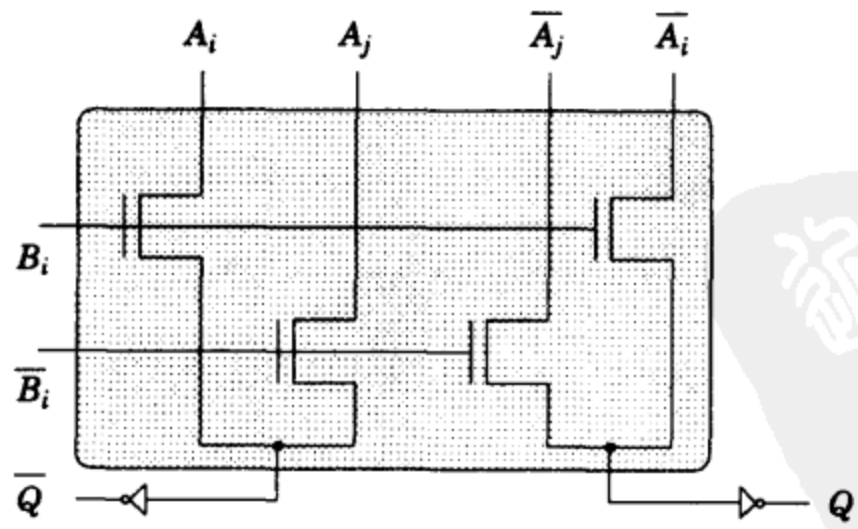


图16-22 NPCPL模块基本结构

16.6.2 给定电路的波流水线算法

615

NPCPL有一个独特的性质，任何电路设计可以看作是NPCPL生成构造模块的组合，生

产构造模块充当通用逻辑元件。电路的延时由最长路径上基本构造模块的数目决定。波流水线的充分必要条件是确保从输入到输出的所有路径有相同的延时。因此，在NPCPL设计用作波流水线前，需要平衡其所有路径的延时。参考文献[2]中引入了CLB的概念来设计波流水线电路。CLB是一个延时为 δ 的NPCPL构造模块，如图16-23所示。其中，输出实现布尔函数

$$out = f(in_1, in_2) \tag{16-39}$$

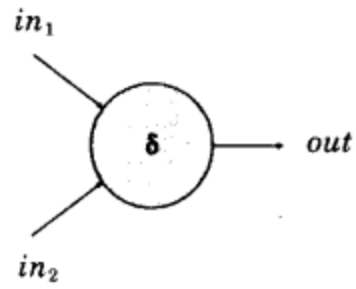


图16-23 CLB的表示

抽象CLB就是当CLB的组合逻辑部分的延时为0，在CLB的输入或输出有一个延时元件D，使得$D=\delta$，如图16-24所示。由此得出，延时元件可以从CLB的输入移到输出，或从CLB的输出移到输入，电路的功能不会有任何改变，这就相当于重定时CLB（参见第4章）。

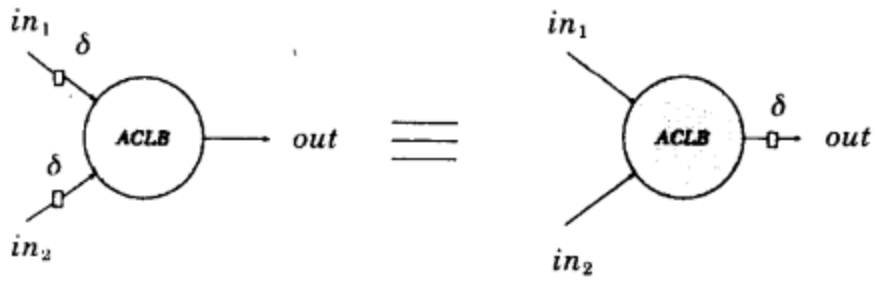


图16-24 CLB的抽象表示

考虑如图16-25所示的任意一个电路需要设计成波流水线。其中 F_1、F_2 和 F_3 分别表示电路需要执行的3种布尔操作。算法的第1步是用2输入函数重新实现电路的布尔函数，如图16-26所示。第2步就是找出电路中从输入到输出的关键路径。这可以通过找出最多数目CLB串联的路径来完成。对图16-26所示的例子，关键路径的长度是5，由 f_1、f_2、f_3、f_4 和 f_5 组成。接下来，物理表示转换成用零延时CLB的抽象表示，如图16-27所示。然后，k个延时元件引入到系统的输入端，每个延时元件延时为 δ，其中 k 是关键路径的长度。然后对电路进行重定时，使得每个CLB后跟着至少1个延时元件，而且，在任何两个延时元件之间，有0或1个CLB，结果得到如图16-28所示的电路。算法的最后一步是合并抽象CLB和它前面的延时，转换成物理CLB，如图16-29所示。由图可以清楚地看出，从输入到输出的每条路径的延时都等于5δ。通常，这种延时平衡技术可以用于任何电路，从而得到相应的波流水线实现。 616

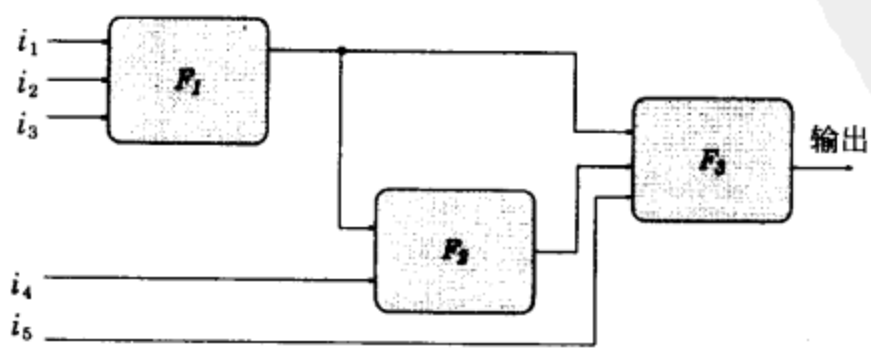


图16-25 用波流水线实现的任意电路

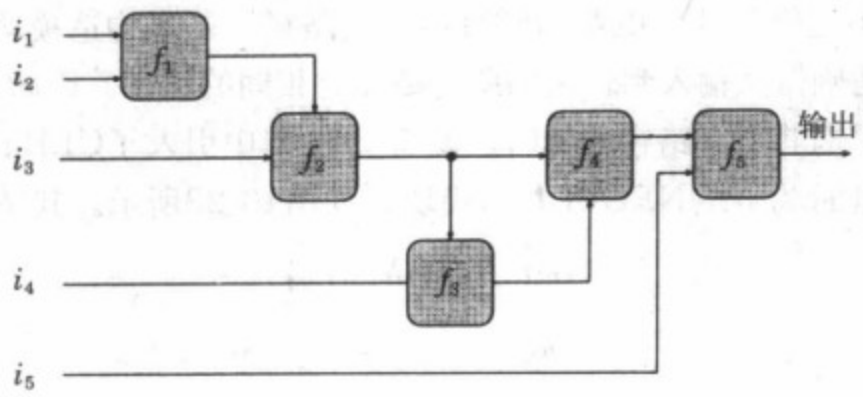


图16-26 用2输入函数实现

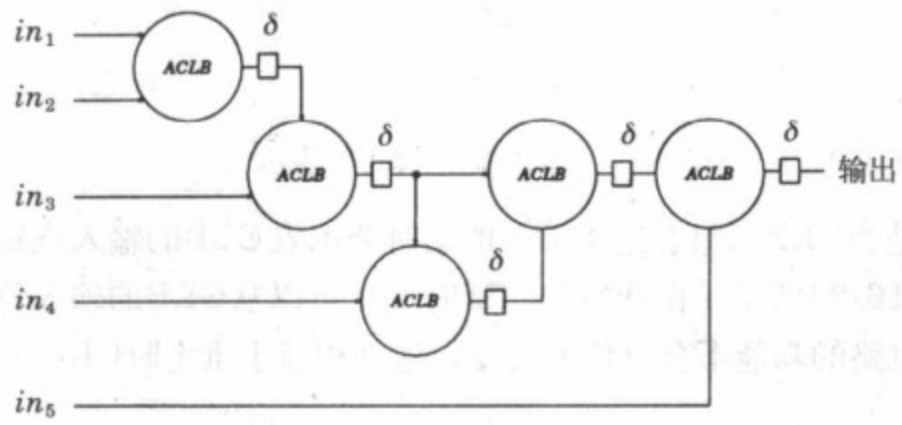


图16-27 用抽象CLB实现

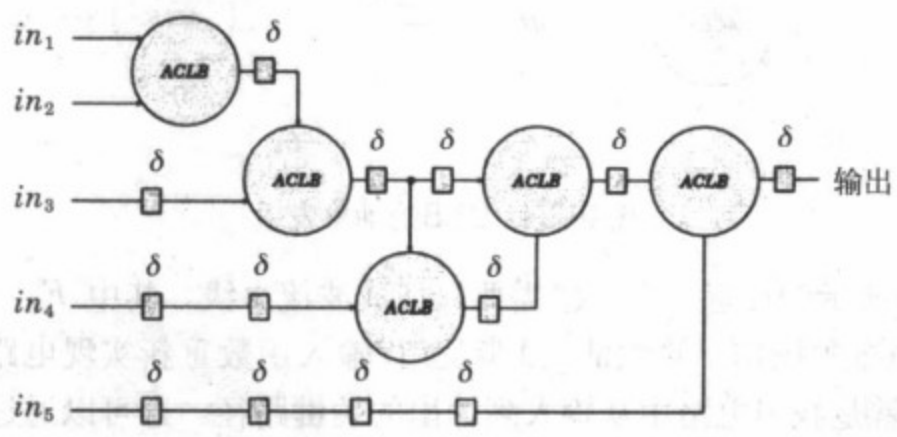


图16-28 抽象表示的重定时版本

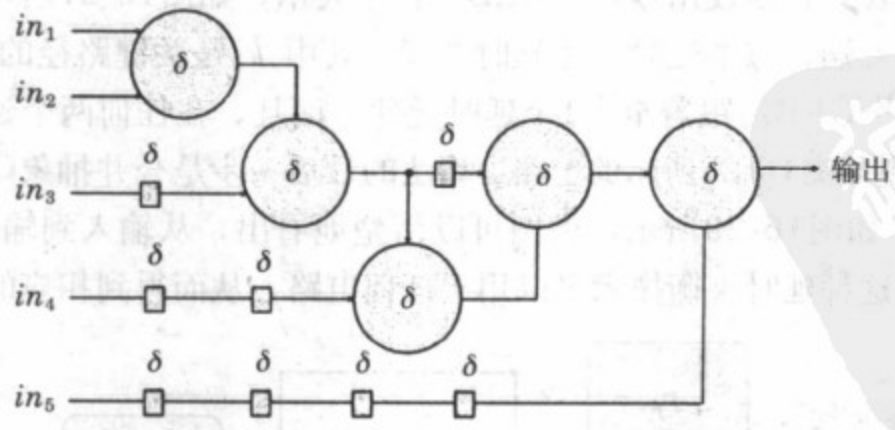


图16-29 抽象表示的重定时版本

波流水线实现具有如下一些有吸引力的性质:

- 不需要流水线锁存器，从而减小了面积，最大限度降低了功耗
- 不需要流水线锁存器，从而简化了时钟分布

- 与相应的同步设计相比，有更高的吞吐量

这种实现的缺点是:

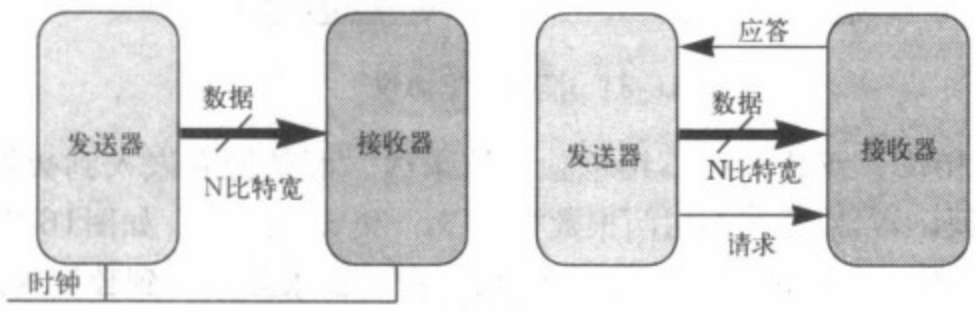
- 数据依赖的延时波动不能完全消除，性能比理想情况有所降低
- 设计需要有晶体管的精确的SPICE模型。因此，当设计要移植到不同的制造工艺时，需要重新设计基本构造模块的静态管尺寸
- 版图对延时的平衡是非常关键的，因此不推荐用自动版图产生器

617 ~ 618

16.7 异步流水线

大多数现在的系统都是基于时钟的，或者说同步的。这些系统由子系统构造而成，其中每个子系统是一个有限状态机。子系统用触发器（寄存器）存储不同的状态，从一个状态变到另一个状态依赖于全局时钟信号。触发器内的状态更新在时钟信号的上升（下降）沿完成。一个典型的同步系统如图16-30a所示。尽管这种方法在数字系统设计中取得很大进展，但也开始面临一些基本的局限性。基于时钟的系统只有在系统的所有部分同时得到时钟时才能正确操作，这只有在时钟线上的延时可以忽略时才能做到。然而，随着工艺的进步，系统趋向于变到越来越大，从而使得时钟线上的延时不再可以被忽略。因此，对许多系统设计者来说，时钟歪斜问题成为了瓶颈。

系统设计者面临的第二个问题就是功耗。从理论上说，功耗应该与系统中作有用工作的各种逻辑门相对应。然而在同步系统中，情况不是这样的。很多逻辑门在做不必要的开关，仅仅是因为它们连接到时钟上了，而不是因为它们要处理新的输入。事实上，最大的逻辑门就是时钟驱动器本身，即使只是系统的一小部分有些有用的事情要做，开关动作也是必需的。于是，同步系统要消耗的功率要多于所必需的。



a) 同步系统　　b) 异步系统

图 16-30

因此，在过去几年中，时钟树或异步系统的设计变得对设计者很有吸引力。异步系统是一种系统内没有全局同步的系统，系统内的子系统只是局部同步，子系统间通过通信协议进行通信。子系统间涉及某些请求/应答类握手协议的系统称作自定时系统。因为大多数实际的异步系统都是自定时系统，这些术语在文献中通常表达的是同一种意思。典型的异步系统如图16-30b所示。

619

在同步系统中，从功耗的角度说，为避免时钟歪斜所付出的代价是非常高的。然而，由于异步系统中没有全局时钟，很好地解决了这个问题。而且，对一个特定子系统来说，只有当数据通过它时，才产生转换，因此当子系统空闲时没有动态功耗。这就使得异步系统对低功耗设计很有吸引力。

异步系统中，一个子系统产生的结果一产生就可以被其他子系统消耗，不需要等待全局时钟的改变。因此，异步系统受限于子系统平均情况下的性能而不是最坏情况下（如同步系统中）的性能。而且，一个子系统可以很容易地用另一个功能相同而性能不同的子系统代替。然而，在同步系统中，这种替换是不容易做到的，因为时钟周期可能需要重新计算。

前述的特征使得异步设计风格对数字系统设计技术发展水平非常有吸引力。很多风格的异步系统流行起来，其中最著名的是捆绑数据协议和双轨协议。在这些系统中，通信协议还可以采用两种形式，如两相和四相协议。

16.7.1 捆绑数据协议和双轨协议

捆绑数据协议使用单条的请求（应答）线和一组数据线（因此称为捆绑），如图16-31所示。请求线用于通知接收器数据捆上的数据的有效性。这就给请求线加了一个固有的约束，称为捆绑约束。根据这个约束，在接收器端，请求线只有在捆绑数据有效后才声明。然而，即使在发送端请求线在数据有效后才声明，任意的连线延时意味着在接收端这个条件可能不再满足。因此，捆绑数据协议并非是延时不敏感的，其中术语延时是指任意的连线延时。因此，在设计捆绑数据系统时，设计者应该确保满足捆绑约束。

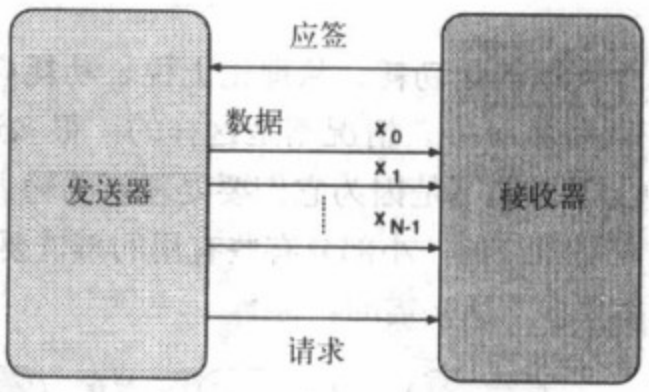


图16-31 捆绑数据协议

与捆绑数据编码相反，双轨编码不用单独的请求线，而是把它嵌入到数据线内。而且，每条数据线 x_i（对 $i \in [1, N]$）现在都用两根数据线 $x_{i,0}$ 和 $x_{i,1}$ 表示，如图16-32所示。$x_{i,0}$ 线上的转换表示传输0，而 $x_{i,1}$ 线上的转换表示传输1。因为请求线嵌入到数据线内，$x_{i,0}$ 或 $x_{i,1}$ 的转换通知接收器数据的有效性。这就使得双轨协议是延时不敏感的，也就是说，它们总能产生功能正确的输出，与连线延时无关。

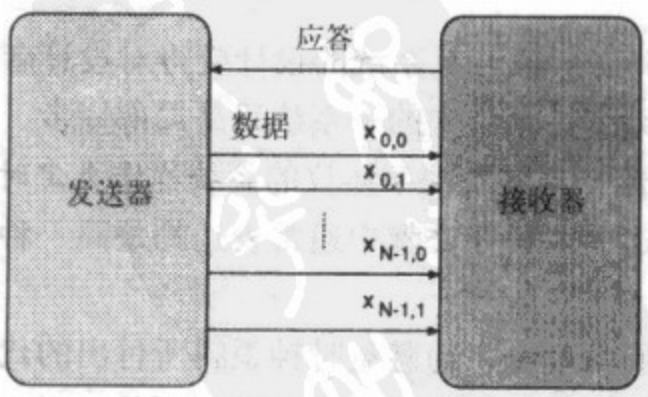


图16-32 双轨协议

选择捆绑数据协议还是双轨数据协议取决于数据捆的特性。例如，如果数据捆相对较大，

捆绑数据协议可能更有效，因为它只需要两个额外的信号，请求和应答。相反，双轨数据协议用两根线表示一个数据位，因此可能需要更多额外的面积。然而，它们的鲁棒性更好，因为对连线延时不敏感。

16.7.2 两相和四相协议

如在前面章节里看到的，请求线用于通知接收器数据的有效性。这可以用两种方法来实现：要么通过请求线上简单的转换（低到高或高到低），要么仅仅是从低到高的转换。前者叫作两相协议，后者叫作四相协议。

用图16-33来说明两相协议，可总结为如下步骤：

(1) 发送器产生数据

(2) 发送器提出请求（用请求线上的一次转换表示）

(3) 接收器接收数据

(4) 接收器发出应答（用应答线上的一次转换表示）

(5) 发送器可以随意禁止数据

620 ~ 621

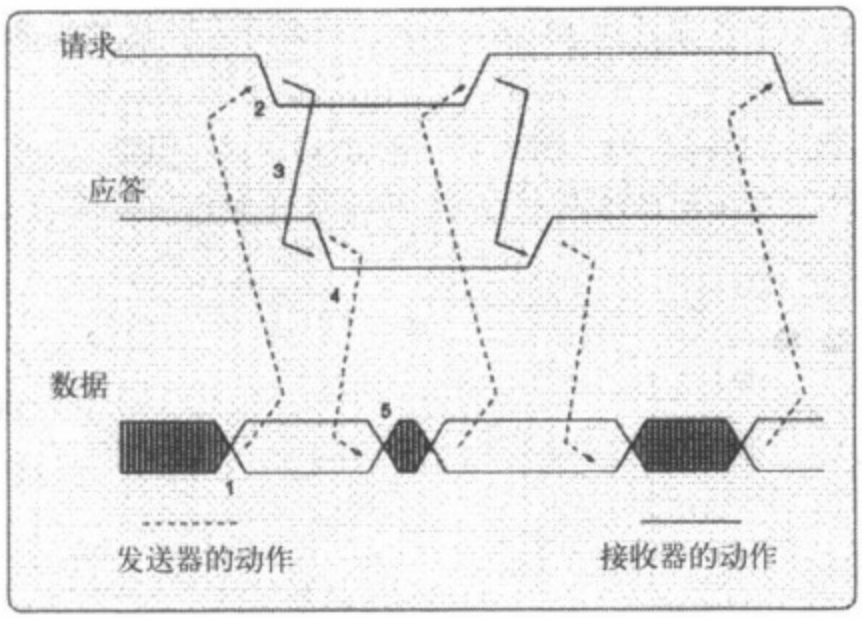


图16-33 两相协议

一旦满足了捆绑约束，也就是遵循了前述的步骤，系统就成为延时不敏感的。一旦设计好了，这些系统就可以放到一个库里，重新用于设计任意复杂的控制结构。

四相协议与两相协议相似，只是它是电平敏感的，即请求线只能通过从低到高的转换来通知接收器数据的有效性。因此，请求线在再次变高之前，必须变回到低，在两个连续的数据值之间引入一个中间的归零相位。四相协议的优点是逻辑处理单元可以简单得多，可以用熟悉的逻辑门。而基于两相协议的设计需要复杂的控制电路来处理转换。

16.8 信号转换图

在本节中，讨论用信号转换图（STG）[14]的异步握手电路规范。在这里，规范中用信号转换而不用信号电平，从而减少映射的复杂性。典型的STG如图16-34所示，其中信号 x 的上升转换用 x^+ 表示，下降转换用 x^- 表示。而且，STG中的弧线代表依赖关系。例如，弧 $z^- \rightarrow y^-$ 的意思是在 y 变低之前 z 必须变低。下面讨论一些与STG相关的术语。 622

活跃性：如果STG中的每一个转换都能最终被使能，就说该STG是活跃的。

例如，如图16-35所示的STG不是活跃的，因为一旦 r^+ 转换被触发，就不能再触发了。

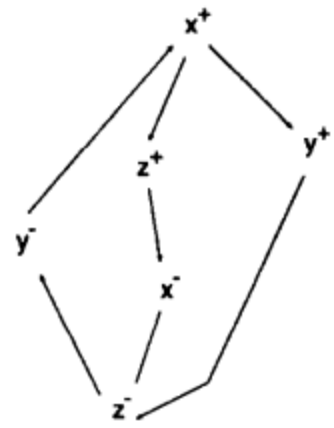


图16-34 典型的STG

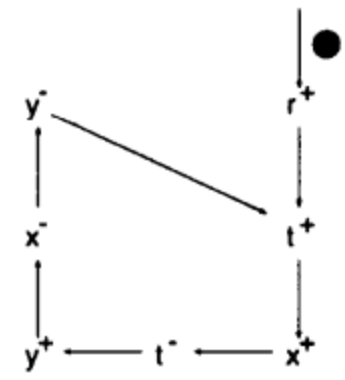


图16-35 违反活跃性的STG

安全性：如果STG中没有地方或弧包含有多于一个记号，就称该STG是安全的。

例如，图16-36所示的STG不是安全的，因为触发序列 $s^+ \to r^- \to s^- \to r^+$ 后，从 r^+ 到 y^+ 的弧有两个记号。

持久性：如果对STG中所有的弧 p^+ $(p^-) \to q^+$，一定存在其他的弧确保 q^+ 在 p^- (p^+) 之前触发，则称该STG是持久的。

图16-37所示的STG不是持久的，因为有一条弧 $x^+ \to y^+$，但是 x^- 可能在 y^+ 之前触发。

一致性：如果各种信号转换严格在+和-之间交替（即已经是低的信号不能再变低，反之亦然），则称STG是一致的。

623

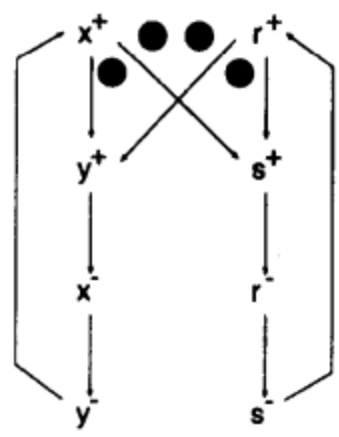


图16-36 违反安全性的STG

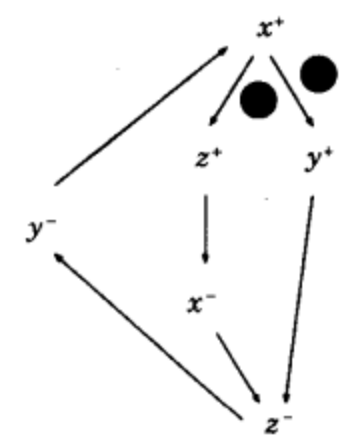


图16-37 违反持久性的STG

对于实现目的来说，这个条件实际上是必需的。如图16-38所示的STG不是一致的，因为可能出现两个连续的 x^+ 而没有 x^- 介于其间，因而它是不可能实现的。

半模块性：如果信号转换一旦被使能，只有触发该转换的信号能禁止它，就称该信号转换是半模块化的。如果电路中每个单个转换都是半模块化的，则称该电路是半模块化的。

例如，如果 a^+ 使能 b^+，半模块性要求只有 b 变高后 a 才可以变低。图16-34所示的STG不是半模块化的，因为在 $x^+ \to y^+$ 操作中的延时可能很长，使得另一个允许的转换序列 $x^+ \to z^+ \to x^-$ 发生在 y 变高之前。这会使得 y 失去变高的机会，因为使能条件 x^+ 不再保持。

在某些广义条件下，在无冒险电路与半模块化信号转换图之间存在等价关系。对图16-34中的图，等价的半模块化图如图16-39所示。这里，增加了一条直接从 y^+ 到 x^- 的弧，防止 x 在 y 变高之前变为低。其他的弧，如从直接从 y^+ 到 z^+ 的弧，虽然也足以使转换半模块化，

但是它们防止 y^+ 和 z^+ 可能同时变化的作用过于强烈。

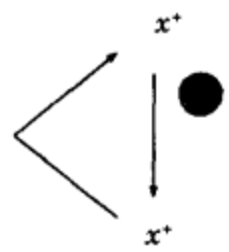


图16-38 违反一致性的STG

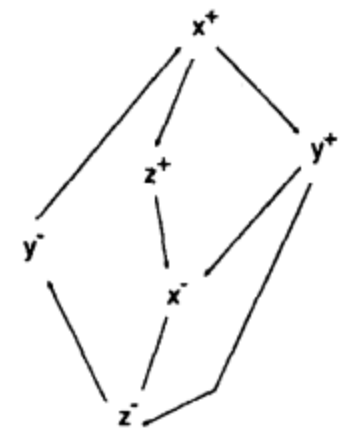


图16-39 图16-34所示图的半模块化STG

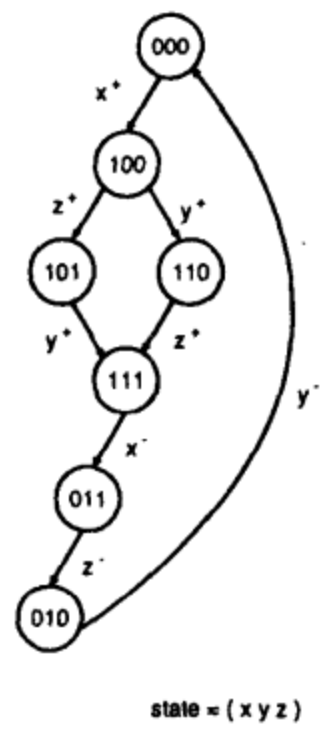


图16-40 图16-39所示STG的状态转换图

为了实现图16-39中所示的STG，第一步是构造如图16-40所示的状态转换图。该图是通过如下方法构造出来的：在相应的STG中，从初始状态开始，然后跟踪所有可能的转换。3个输出的卡诺图分别如图16-41（a）、（b）、（c）所示。所得到的逻辑表达式可以很容易用简单的逻辑门来实现。注意，状态001没有出现，会导致在卡诺图中出现‘X’（不关心）记号。在z的卡诺图中，如果当前状态是100，则 z 可以是0或1。因为 z 可以改变到1，所以放一个1在卡诺图中。 624~625

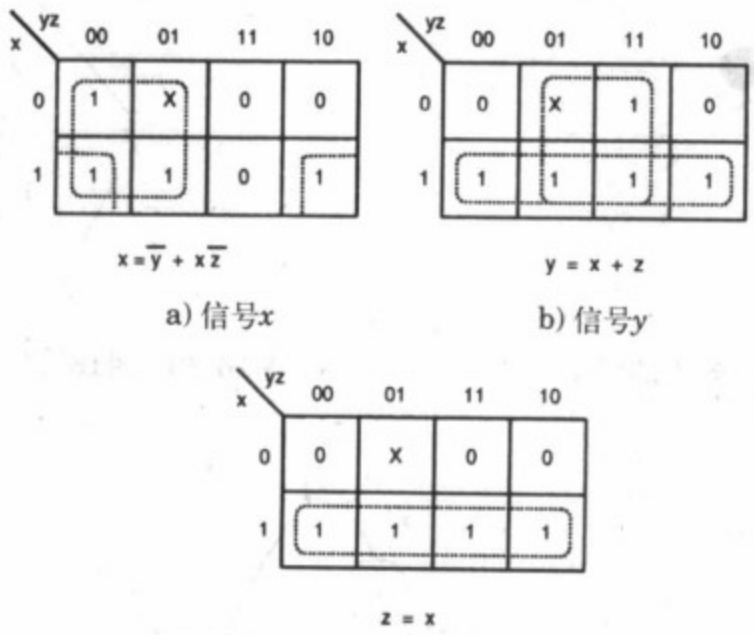
16.9 信号转换图应用于设计互连电路

考虑如图16-42所示的典型的互连电路，其中两个单元 A 和 B 连接在一个流水线里。在握手电路中,当单元B处理当前的数据采样时，单元 A 处理下一个数据采样。因为单元 B 需要的时间可能更长或更短，因此，当 B 完成了它的任务并准备处理下一个数据采样时，需要一个应答信号指示。握手电路的目的就是要确保数据采样正确地传输，防止任何失控（数据重写）状况。

有两种风格的互连电路如图16-43所示[14]，它们在输入与输出信号关系的规格上有区别。

图16-43a的STG的规格是

$$*[R_{in}^{+} \rightarrow R_{out}^{+}] \tag{16-40}$$



a) 信号x　　b) 信号y

c) 信号z

图16-41 输出信号的卡诺图

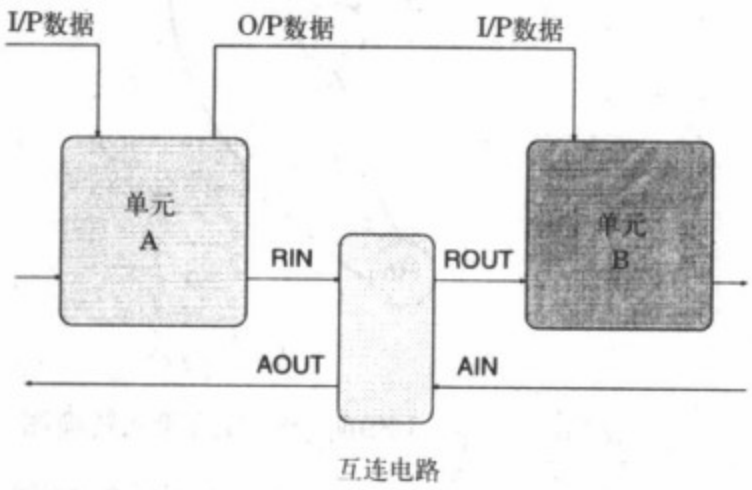
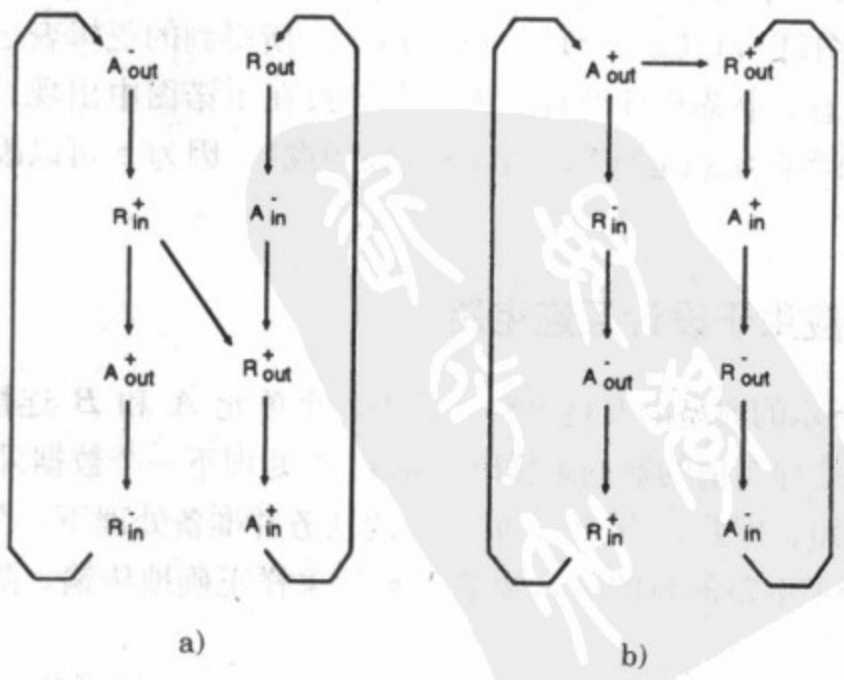


图16-42 互连电路的例子



a)　　b)

图16-43 互连电路的两种不同规格

626

而图16-43b的STG的规格是

$$*[A_{out}^{+} \rightarrow R_{out}^{+}] \tag{16-41}$$

第一个条件意味着，如果输入采样准备好了（R_{in}^{+}），就启动运算（R_{out}^{+}）。在可以接受新的采样前，随后的单元必须是空的（A_{in}^{-}），参见图16-43a所示的四相握手转换 $A_{in}^{-} \rightarrow R_{out}^{+}$。

第二个条件告诉单元A，它的输出数据已经被接受了（A_{out}^{+}），单元B可以开始运算（R_{out}^{+}），而R_{in}可以复位（R_{in}^{-}）。同前面一样，在可以接受新的采样前随后的单元必须是空的（A_{in}^{-}），参见图16-43b所示的四相握手转换 $A_{in}^{-} \rightarrow R_{out}^{+}$。

要满足无冒险的实现，STG必须满足最弱的半模块化条件。这里讨论文献[14]中提出的在一个活跃的STG中推导最弱半模块化条件的算法。算法的第一步是关于非半模块化转换（使能多许可序列的转换）的鉴别。然后，通过增加从非半模块化指向所有转换（其逆转换启动该非半模块化转换）的弧，使得该非半模块化转换满足最弱半模块化条件。随着新的弧增加，多余的弧删除，STG得到更新。这个过程要递归地重复下去，直到STG中的所有转换都是半模块化的。例如，考虑图16-43a所示的STG，其中弧 $R_{in}^{+} \rightarrow R_{out}^{+}$ 表示一非半模块化转换（因为 R_{in}^{+} 有多个许可序列）。根据算法，一条直接从 $R_{out}^{+} \rightarrow R_{in}^{-}$ 的弧要增加到图中，使得从 R_{in}^{+}的转换变为半模块化的。然而，R_{in} 是一个输入信号，物理电路不能在任何输入信号上施加任何约束，它只能在输出信号上施加约束，这反过来会影响目标输入信号。因此，加到图中的第一条弧是直接从 $R_{out}^{+} \rightarrow A_{out}^{+}$，如图16-44a所示，而从 R_{in}^{+} 到 A_{out}^{+} 的多余的弧被删除了。这反过来又导致一非半模块化转换 R_{out}^{+}。因此，要增加一条直接从到 R_{out}^{-} 的弧，又使得A_{out}^{+} 变成非半模块化的。最后，增加一条直接从 R_{out}^{-} 到 A_{out}^{-} 的弧，使得整个STG是半模块化的。图16-43b所示的图的半模块化的STG可以用类似的方法推导出来，所得到的STG如图16-44b所示。最后，如前节所述，卡诺图可以从STG构造出来，相应的实现如图16-45中的(a)和(b)所示。

627

图16-45所示的实现不是真正延时不敏感的，因为使用了SR锁存器，而S和R输入的互斥性不能通过无限的外部门延时保证。例如，考虑图16-45b中产生 A_{out} 的SR锁存器。如果R输入端的反相器门延时大于一个与门加一个SR锁存器的延时，那么S和R输入可能同时变高，导致冒险。然而，因为这种不匹配在实际中不会发生，所以SR锁存器还是采用得比较广泛。

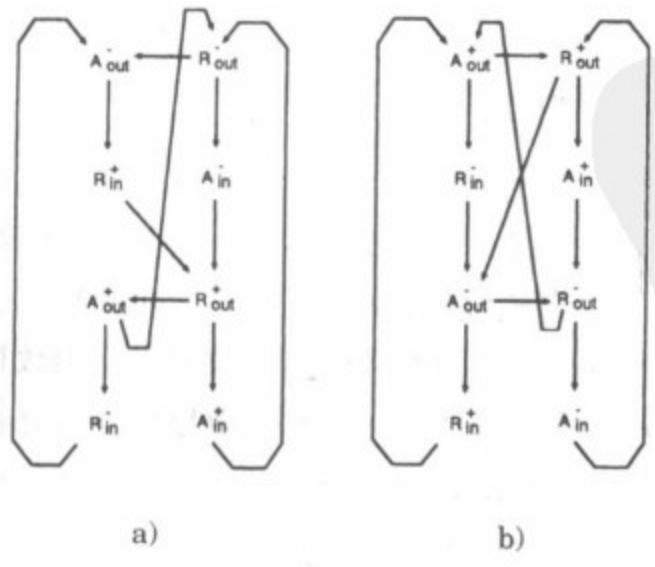


图16-44 图16-43所示图的半模块化STG（最弱条件）

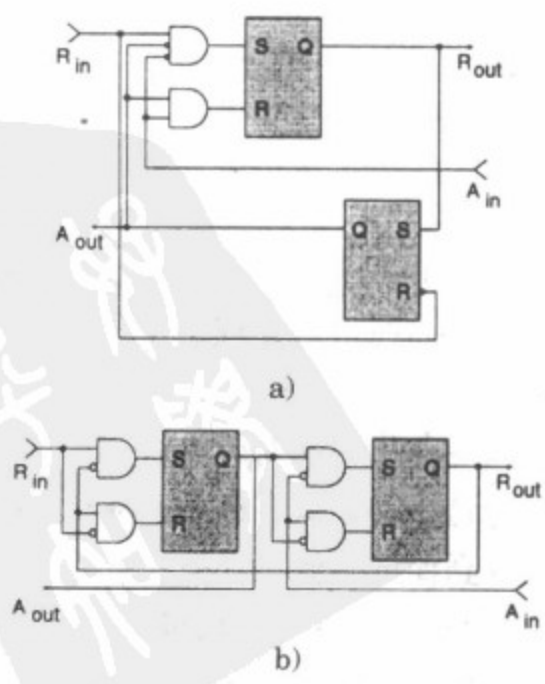


图16-45 从相应STG得出的握手电路的电路图

纯粹的延时不敏感电路还可以用C元件来设计。一个2输入C元件实现布尔函数$C=AB+BC'+AC'$，其中A和B是输入信号，C'是上一个输出信号，C是当前输出信号。C元件有这样的特性，只有当两个输入信号电平相同时输出信号才改变到输入电平，否则维持在上一次的值。也有这样的提法，就是延时不敏感电路只用C元件。C元件的实际实现在下节讨论。因为从半模块化电路得到的SR锁存器排除了任何冒险条件，SR锁存器和带一个反相器的C元件在功能上是等同的。因此，C元件可用于替代SR锁存器，不会有任何功能差别。用C元件替代SR锁存器得到的握手电路分别如图16-46中的(a)和(b)所示。

上述的两种设计的硬件利用效率可以由它们相应的STG推导出来。在活跃的半模块化图中，一旦一个请求被接受，在下一个请求被接受前，每个转换只能经过一次。响应外部请求的延时是图中所有简单回路延时中最长的。运算弛豫时间比握手转换要长得多，因此STG中的运算弧要首先鉴别出来。在图16-44a的STG中，转换环路 $A_{out}^- \to R_{in}^+ \to R_{out}^+ \to A_{in}^+ \to R_{out}^- \to A_{out}^-$ 经过两个运算弧 $A_{out}^- \to R_{in}^+$ 和 $R_{out}^+ \to A_{in}^+$。如果与运算单元相关的弛豫时间相同，则由$^*[R_{in}^+ \to R_{out}^+]$ 指定的握手电路将使得任何时刻都有一半的运算单元处在空闲状态。因此，这种握手策略通常称为不完全握手。然而，在图16-44b中，没有包含两条运算弧的环路，因而运算单元全时工作。因此这种握手策略称为完全握手。

628 ~ 629

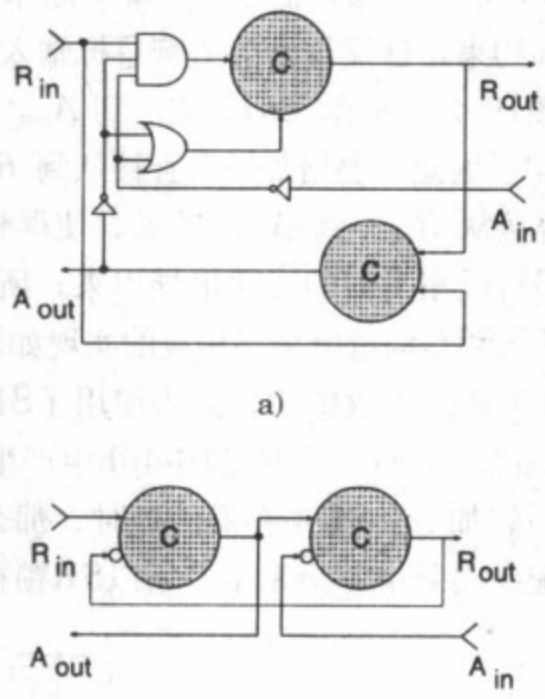


图16-46 使用C元件的握手电路

运算单元和互连模块之间的连接通过请求-应答信号对来规定。寄存器通常当作是互连模块的部分，因为在信号处理意义上它们实现流水线级。例如，图16-46b中，完全握手的第一个C元件控制应答信号A_{out}，而第二个C元件控制到下一个模块的请求信号R_{out}。由此得出，对于用完全握手电路的任何流水线互连，A_{out}是惟一可用于锁存而不破坏输入数据或输出数据的信号。用于检测寄存器锁存的完成检测电路如图16-47所示，其中，延时不敏感性在寄存器锁存操作中得到保证。假设在操作开始时，所有信号初始化为逻辑0。当输入寄存器端数据稳定时，请求线R_{in}上产生一个转换。这使得C元件的输出变高，数据被锁存。然后，寄存器通知第二个C元件，数据已经锁存好了，准备由运算单元去读取。同时，通过A_{out}应答输入端。这时，输入端有空继续进行下一次运算。其间，因为A_{in}已经初始化为逻辑0，第二个C元件的输

630

出为高。然后，进行运算，一旦输出数据准备好，*Done*信号置为高。

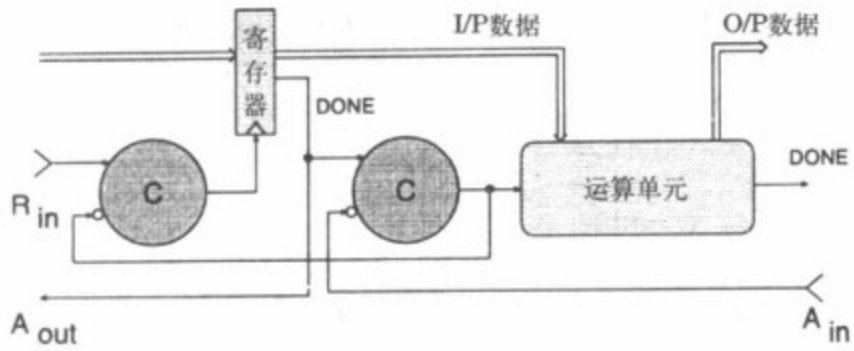


图16-47 用一个寄存器和完全握手的完成检测机制

16.10 运算单元的实现

本节讨论用现场可编程门阵列（FGPA）或全定制VLSI实现异步运算单元的设计技术。

16.10.1 全定制VLSI实现

已经证明，用两个二进制握手信号，*请求和应答*，来实现通常的异步运算单元是充分且必要的。为了提高硬件实现的效率，假设沿着数据通路都可以产生请求和应答信号，则任何组合逻辑电路都可以合并到一个运算模块中。在异步电路设计中普遍采用差分级联电压开关逻辑（DCVSL），因为它提供一种产生完成信号的完美方法。DCVSL模块的通用结构如图16-48所示，其中*request*信号可以看作是来自前一个模块的完成信号。当*request*线变为有效低电平时，两条输出数据线通过PMOS晶体管上拉，而*done*信号被下拉。当*request*线变高，表示前一级的运算已完成，控制/数据信号稳定赋值，两个PMOS管将关断，输入线由NMOS树赋值。NMOS树本身的设计应该使得只有输出数据线（$f/\bar{f}$）中的一条被NMOS树下拉，使得完成信号通过与非门置位为高电平。差分NMOS树可以用现有的逻辑最小化算法来设计，已经证明，与简单的与非门/或非门系列相比，它具有性能上的优点。下面用一个简单的例子来说明DCVSL模块的设计。 631

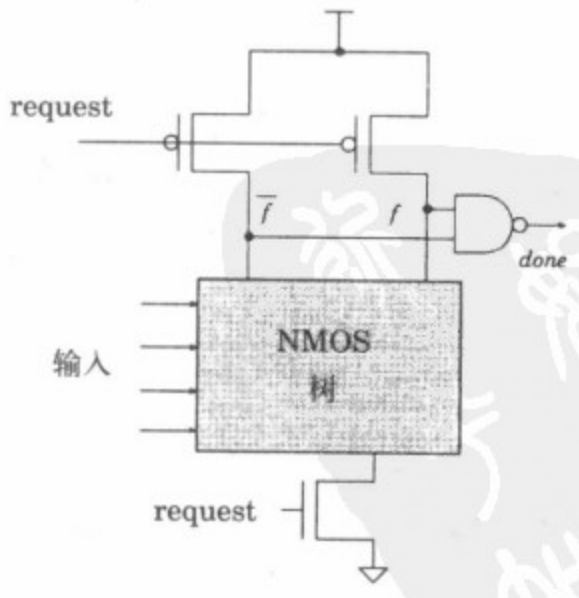


图16-48 用于异步电路设计的DCVSL模块通用结构

考虑如下方程所描述的逻辑函数的实现

$$f = AB + C(A + B) \tag{16-42}$$

用DCVSL实现该函数，需要用如下所示的if-else风格来表示逻辑。

```
if    A = 1
      if B = 1
            f = 1; f̄ = 0
      else
            if C = 1
                  f = 1; f̄ = 0
            else
                  f = 0; f̄ = 1
else
      if B = 1
            if C = 1
                  f = 1; f̄ = 0
            else
                  f = 0; f̄ = 1
      else
            f = 0; f̄ = 1
```

632

这可以转换成如图16-49所示的实现，其中C模块对if和else模块都是一样的，因此由它们两者共用。在DCVSL设计中可能出现电荷共享的问题，应该采取适合的预防措施来避免。例如，节点1和节点2的电容应该做大一些（使用大的预充晶体管实现），如果有必要，可以在节点3、节点4和节点5连接直接的预充静态管。下面讨论一些典型的异步电路例子。

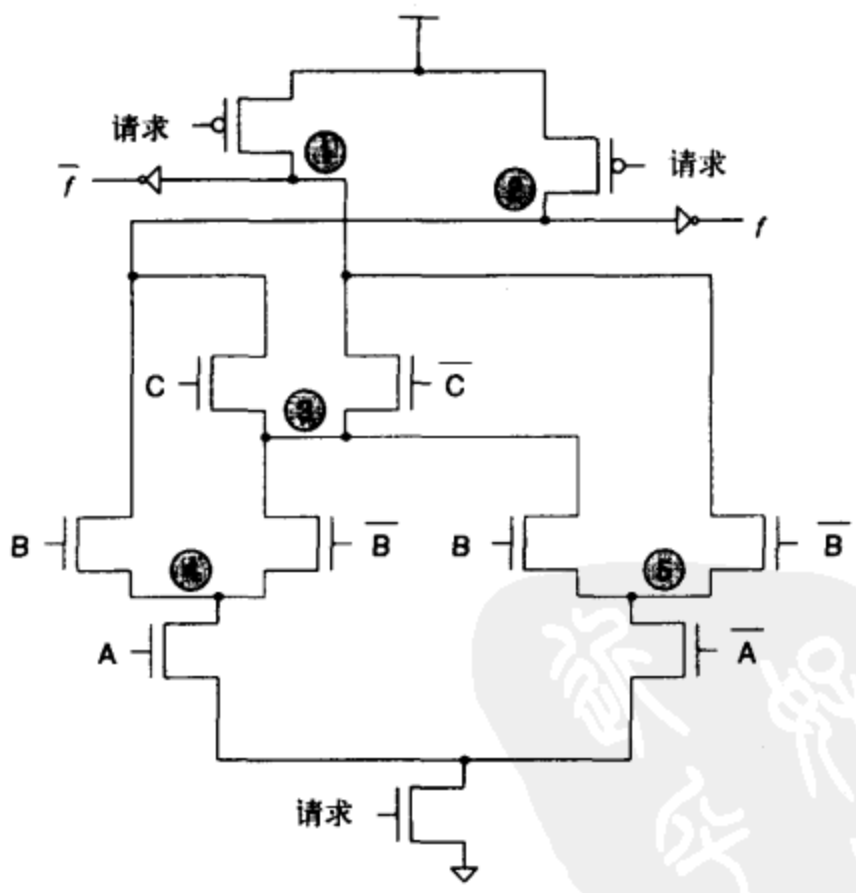


图16-49 逻辑函数$f=AB+C(A+B)$的DCVSL实现

16.10.1.1 Muller C元件

考虑如图16-50所示的Muller C元件电路，其中A和B表示输入信号，C表示输出信号。当A和B都为低时，两个PMOS晶体管P1和P2都导通，节点1上拉到Vdd，从而使得输出变低。

如果A或B一个为高，节点1被隔离，因此保持它原来的值。当A和B都为高时，NMOS管N1和N2都导通，节点1被拉到地，使得输出信号变高。

16.10.1.2 异步2输入多路器

异步2输入多路器如图16-51所示，其中A和B表示两个输入信号，*out*表示输出信号。当*request*信号变为有效的低电平时，两个PMOS上拉管导通，*out*和 $\overline{out}$ 被置为逻辑0。当*request*信号变为有效的高电平时，PMOS管截止，根据选择信号S的值，*out*或 $\overline{out}$ 被拉到地。 633

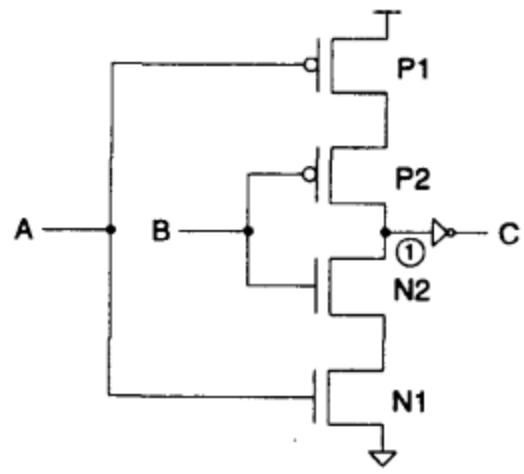


图16-50 Muller C元件的实现

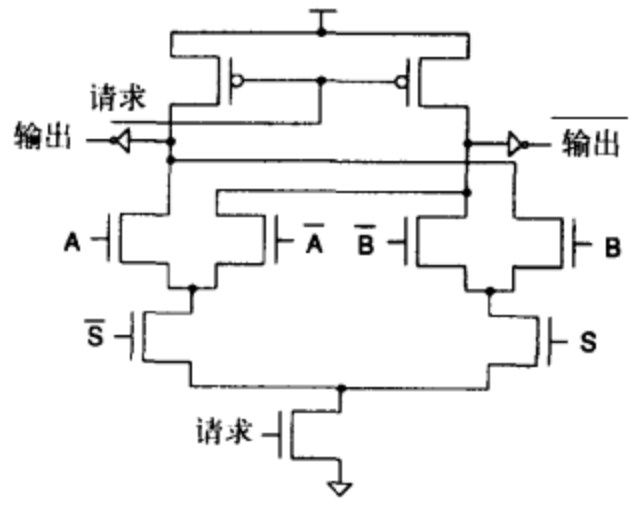


图16-51 异步2输入多路器

16.10.2 FPGA实现

本节讨论用Actel FPGA的FPGA实现。芯片排列非常像传统有布线通道的门阵列，逻辑单元行之间分布着包含各种连线长度的布线通道。这些连线段通过一种叫作反熔丝的两端口电可编程器件相连，当通过它的端口施加适当的电压对它进行编程时，它就不可逆地从高阻变为低阻。逻辑模块通过编程选择的反熔丝连接起来实现预期的功能，从而对整个芯片编程达到预期功能。一次编程后，Actel的FPGA不能再编程改变功能。然而，XILINX系列的FPGA是可以再编程的。本节中，仅讨论用Actel的FPGA进行异步电路的设计。 634

16.10.2.1 C 元件

2输入的C元件是一种由Muller首先提出的逻辑门，当它的两个输入都为高时，它的输出置为高电平，而当两个输入都为低时，输出置为低电平。如果两个输入互为相反的状态，输出保持它的上一个值。因此，C元件可以像一个与门的转换函数使用。只有当C元件的两个输入都发生转换时，C元件的输出才会发生转换，假设两个输入初始化为同样的状态。

C元件的两种不同形式如图16-52a和16-52b所示，第一种没有清0信号（CLR），第二种有清0信号。两种电路都用简单的4选1多路器实现。从图16-52a可以看出，只有当两个输入为相同的状态时，输出才达到相同状态。在图16-52b中，用了两个4选1多路器来结合低电平有效的清0信号。当清0信号为低时，第二个多路器的输出被清0，如果清0信号为高，第一个多路器的输出直接通过去。

16.10.2.2 转换锁存器

转换锁存器是一种响应转换控制信号而捕获和存储数据（D）的电路。在控制信号的每次转换时捕获新的数据。图16-53a显示了一种常锁定的转换锁存器的符号[8]，而图15-53b显示

了它的等效电路图。其中，C和P分别表示捕获和传递控制信号，而D和Q分别表示输入和输出数据信号。该锁存器用使能反相器形成环路来捕获数据信号。初始化后，当两个控制信号为相同电平时，锁存器从输入到输出是不透明的，因为在上部（如果控制信号初始化为逻辑0）或底部（如果控制信号初始化为逻辑1）形成了一个环路。信号C发生第一次转换时，打开一个环路，使得锁存器从输入到输出变成透明的。当P线上发生转换时，又形成了环路，只有当C线上发生另一次转换时才会将回路打开。用4选1多路器的FPGA实现如图6-53c所示，它的操作可以很容易得到验证。

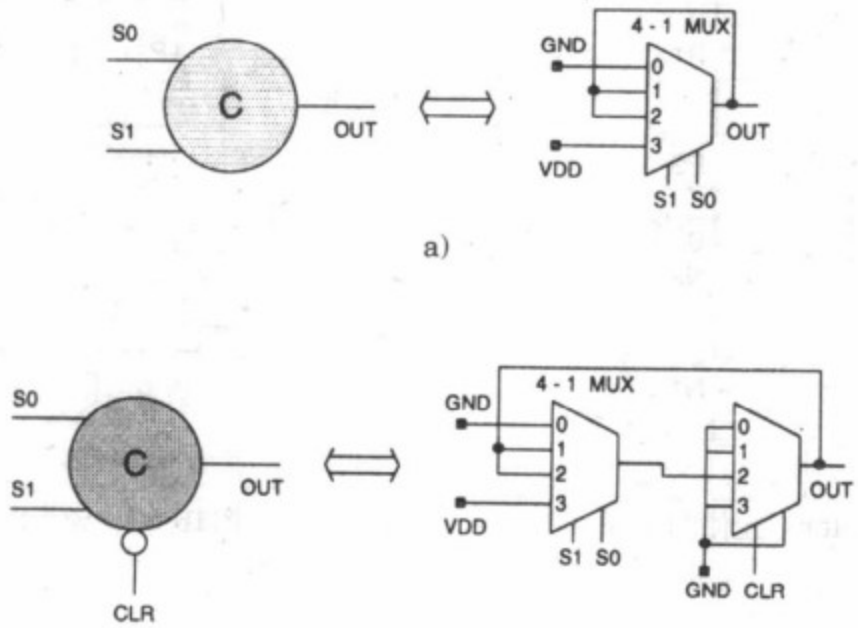
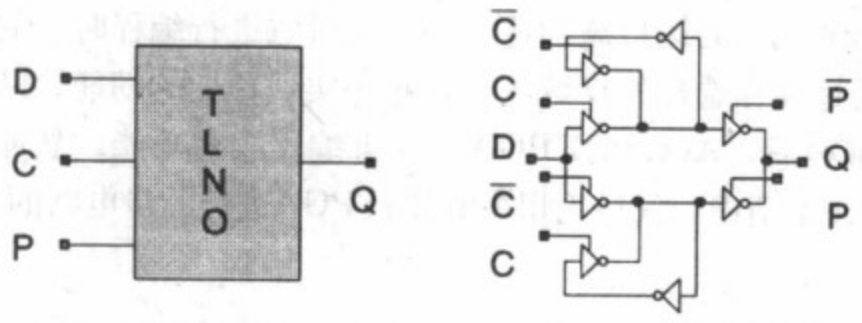
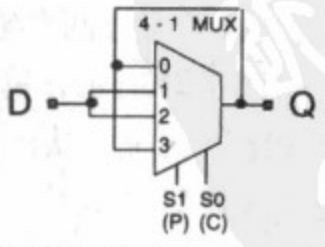


图16-52 两种C元件



a) 转换锁存器（常锁定的）符号　　b) 等效电路图



c) 用Actel FPGA实现

图 16-53

可以很容易设想出上述锁存器的一个变种。锁存器在初始时是透明的，当控制信号发生转换时变为不透明的。这种锁存器称为常通的锁存器，具体细节可参考文献[8]。

16.10.2.3 进位完成加法器模块

加法运算是一种重要的DSP运算，本节讨论加法器模块的设计。加法运算范围可能从串行进位加法到超前进位加法，到进位保留加法等，挑战在于辨别最坏情况数据路径并开发一种满足捆约束的模型。在异步系统中，一种简单处理加法运算的方法是把它看作是一个组合逻辑函数，然后建立一个通用的关于该函数的捆绑模块。例如，对串行进位加法的情形，捆延时必须模拟从较低位到较高位进位传输的情形。如图16-54所示的进位完成加法器（CCA）[7]使用进位完成探测，有一个4相位的控制接口和2个内部进位信号，一个用于表示到下一级的进位（*CIN*/*COUT*），另一个是到下一级的非进位（*DCIN*/*DCOUT*）。这些信号在加法运算开始时初始化为逻辑0。当使能信号（*EN*）变高时，加法器开始计算和与进位信号，接着声称*COUT*或*DCOUT*，表示加法操作完成。*DONE*信号可以很容易有*COUT*和*DCOUT*的逻辑或产生。如果该加法器用于两相环境下，可以用一个2相协议到4相协议的转换器。

635 ~ 636

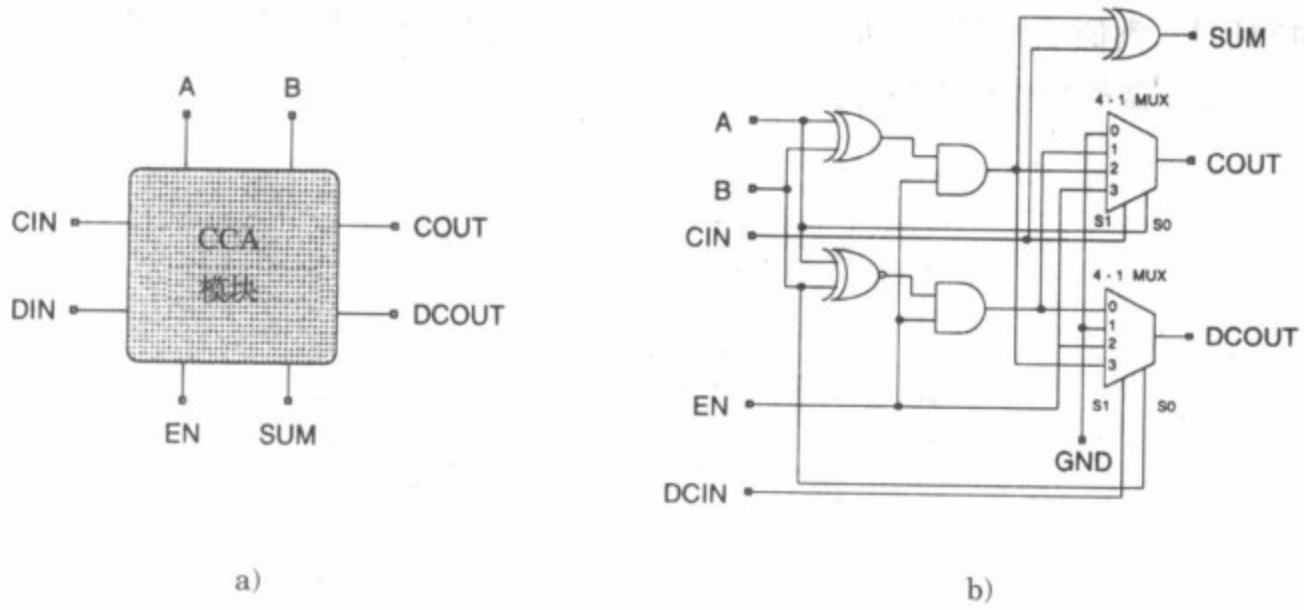


图16-54 a) CCA的符号 b) FPGA实现

通过级连CCA模块得到的4位CCA如图16-55所示，其中每个模块的使能信号连接在一起作为请求线。每个模块的应答信号连接到*C*元件树，得出的信号表示所有加法器位都已经产生了相应的进位。下降的*EN*信号将所有的进位信号拉低，为下一次加法初始化加法器，从而引起*Ack*信号下降，表示每一位都被复位，加法器准备好，可以计算新的结果。

637

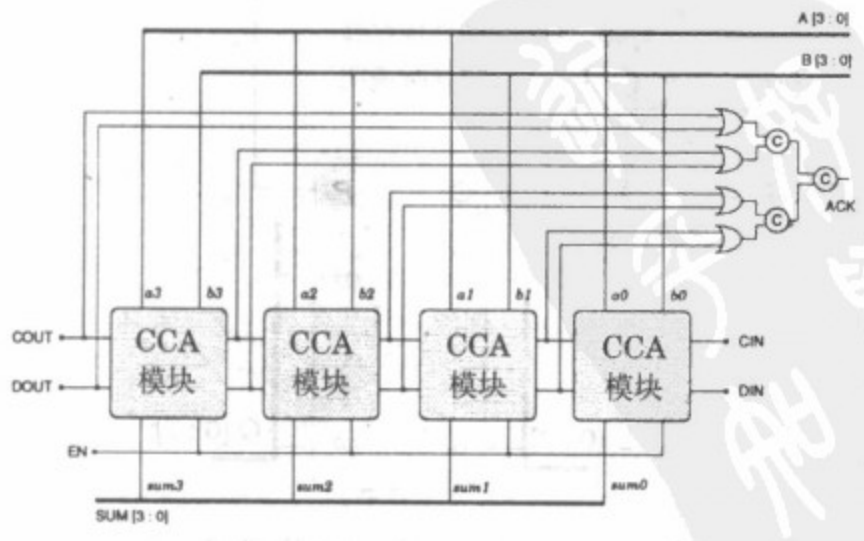


图16-55 4位的CCA

16.10.2.4 先进先出缓冲器

简单的先进先出（FIFO）可以使用转换锁存器和*C*元件构造，如图16-56所示。电路中所有线初始化到低状态后，电路的第一个动作是从通道*A*接受数据。为了完成这一步，通道*A*先把数据放到数据线上，然后在通道*A*的请求线（*REQ*）上产生一个转换。因为所有线都初始化为低，*C*元件的输出被强行置为高。该转换信号通知寄存器锁存其输入端上的数据。然后，寄存器应答通道*A*，数据已经被锁存了，还通知通道*B*，数据已经可以读出了。这时，通道*A*可以通过改变数据线上的值发送更多的数据，并在通道*A*的请求线上产生另一次转换。通道*B*也可以接受当前保持在寄存器中的数据，并通过通道*B*的应答信号线（*ACK*）上的一次转换通知寄存器不再需要保存数据了。只有当这些事件都发生了，也就是，*C*元件的两个输入端都有转换发生，寄存器才算准备好了，可以从通道*A*接受更多的数据。相应的FPGA实现如图16-57所示，其中，寄存器用了8个常通的转换锁存器。可以将1个字的FIFO堆叠起来，构成更深的FIFO，类似于图16-58所示。其中，4个1个字的FIFO级连在一起形成了4个字8位的FIFO。该寄存器的操作很容易得到验证。

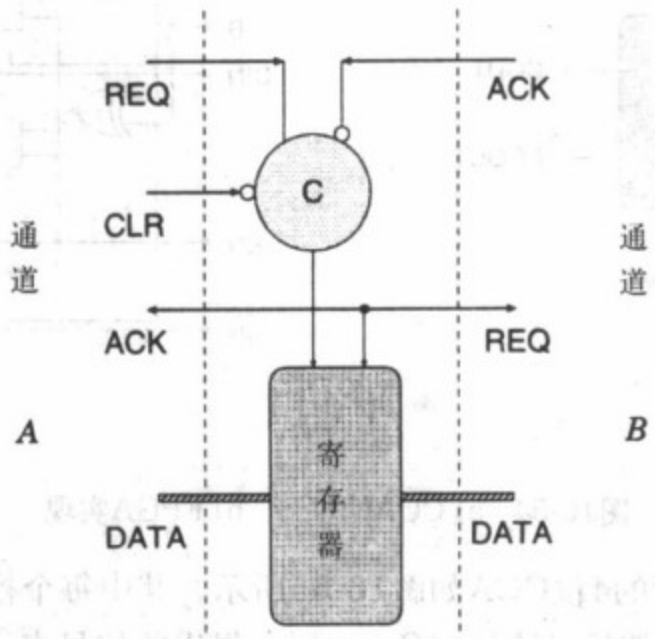


图16-56 先进先出缓冲的一个阶段通道

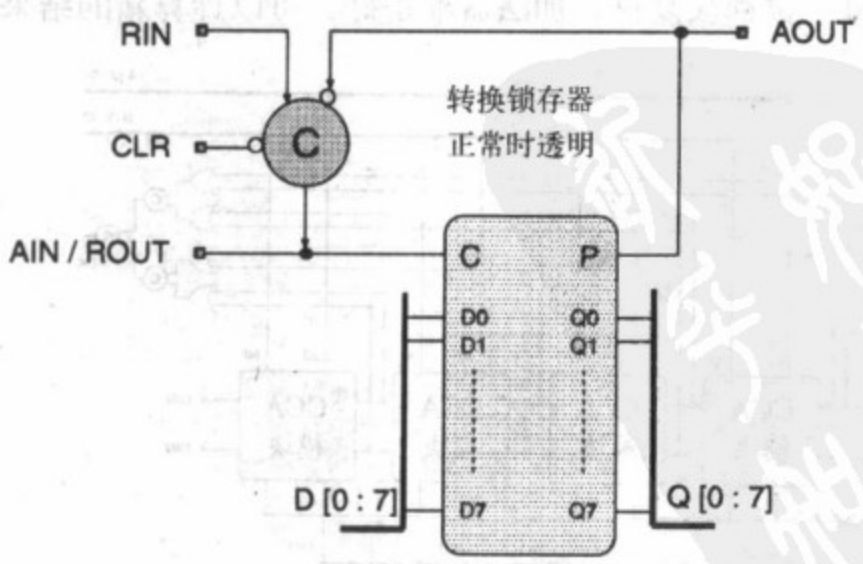


图16-57 一个字的FIFO缓冲器的FPGA实现

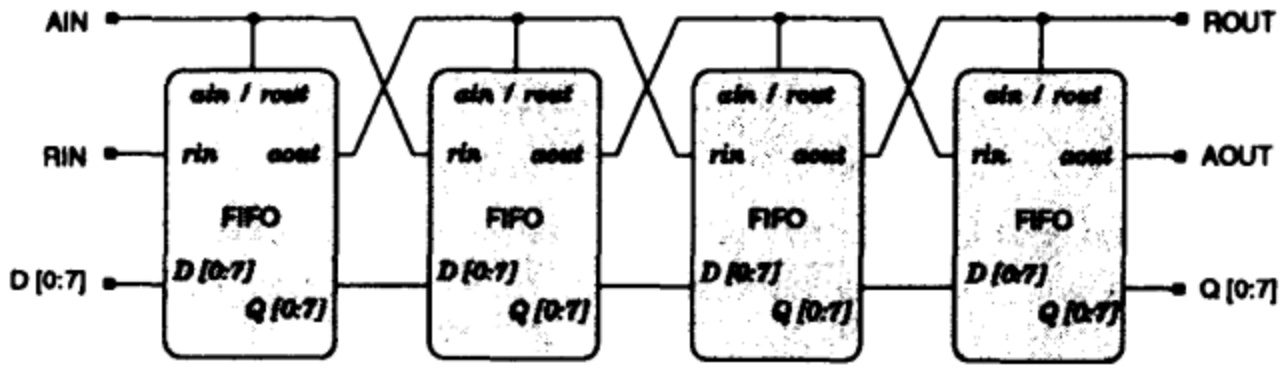


图16-58 4字8位FIFO的FPGA实现

638 ~ 639

16.11 结论

本章对各种流水线风格作了全面的讨论。首先，对同步流水线和各种时钟风格做了详细讨论。随后对时钟歪斜和时钟分布的问题进行了探讨，并介绍了解决这些问题的方法。在余下的章节里，讨论了两种较新的技术，分别是波流水线和异步流水线。波流水线可用于增加流水线级数而不需要用物理寄存器。这是通过在电路级和系统级对设计进行精确的时序分析而实现的。由于不需要用寄存器，从而简化了时钟分布。而且，可以获得比同步系统高得多的吞吐量。另外，还介绍了一种系统地设计波流水线电路的方法。

异步流水线是一种交替的流水线风格，完全不需要使用全局时钟信号，通信用握手信号完成。本章介绍了基于握手信号分类的各种不同类型的异步电路。还介绍了用信号转换图进行4相握手电路的设计。还列举了许多例子说明异步运算单元的设计。

16.12 习题

1. 考虑如图16-59所示的用CLB和锁存器构成的典型的时序系统。假设最左边的锁存器的输入信号是$[V\phi_1]$，该锁存器用ϕ_1作时钟。该时序电路能正确工作吗？解释其原因。节点1~8的信号是什么状态，它们是 $[V\phi_1]$，$[V\phi_2]$，$[S\phi_1]$ 还是 $[S\phi_2]$？说明哪些锁存器的时钟为ϕ_1，哪些锁存器的时钟为ϕ_2。
2. 考虑如图16-60所示的电路，其中输入信号X为$[V\phi_1]$，输入信号Y为$[S\phi_1]$，信号Z作为锁存器的时钟。该锁存器的功能正确吗？如果正确，该电路的主要优点是什么？
3. 画出图16-8所示电路的ϕ_1和ϕ_2的时序波形，并验证两相时钟产生器的操作。
4. 设计一个9个晶体管的、正沿触发的真单相时钟触发器，并说明它的操作。

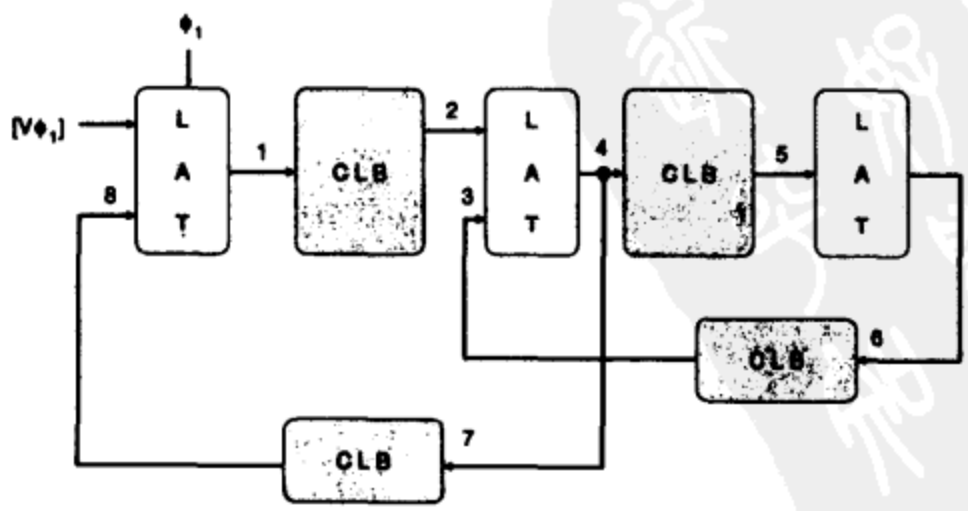


图16-59 典型时序系统

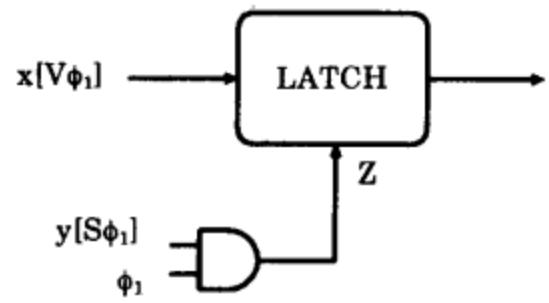


图16-60 习题2的电路

5. 假设位流水线数字系统使用单相电平敏感时钟策略，脉冲宽度 t_ϕ。证明该时钟策略的时钟歪斜 δ 由下面的公式给定。对正的和负的 δ，说明电路的行为。

$$\delta < t_{pr} + t_{pi} + t_{pl} - t_\phi \tag{16-43}$$

6. 画出带和不带中间缓冲的4位 × 4位规则单元阵列的H树型分布网络。
7. 画出图16-61所示电路的等效波流水线实现，使得从输入到输出的每条路径具有相同的延时。
8. 画出4相握手通信系统的时序图，并说明所涉及的各个步骤。
9. 考虑如图16-62所示的STG。说明该STG是否为（a）活跃的（b）安全的（c）一致的（d）半模块化的。如果该STG不是半模块化的，画出它的等价的半模块化的表示。

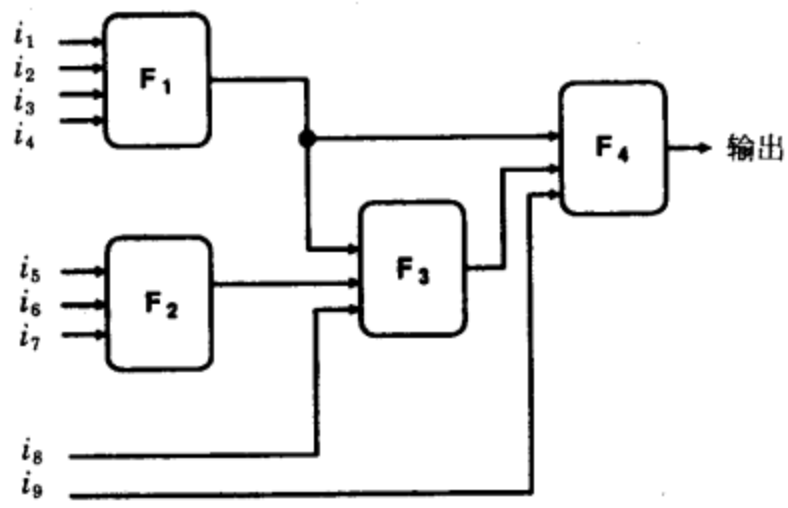


图16-61 习题7的电路

图16-62 习题9的电路

10. 通过画图16-44中（a）和（b）所示STG对应的卡诺图，分别推导出图16-45中（a）和（b）所示的不完全握手和完全握手电路的电路图。（提示：参考[14]）
11. 画出常通的异步转换锁存器的电路图，并说明它的操作。
12. 用进位完成加法器设计一个异步无符号4位 × 4位乘法器。明确标明握手信号的用法。
13. 画出如下方程表示的逻辑函数 f 的DCVSL实现。

640 ~ 642

$$f = AB + D(C + B) + \overline{AD} \tag{16-44}$$

参考文献

1. D. Fan, T. Gray, W. Farlow, T. Hughes, W. Liu, and R. Cavin III, "A CMOS parallel adder using wave pipelining," in *Proc. Brown/MIT Conference Advanced Research in VLSI and Parallel Systems*, pp. 147–164, March 1992.

2. D. Ghosh and S. K. Nandy, "Design and Realization of high-performance wave-pipelined 8×8b multiplier in CMOS technology," *IEEE Trans. VLSI Systems*, vol. 3, no. 1, pp. 36–48, March 1995.

3. C. T. Gray, W. Liu, and R. Cavin III, "Timing constraints for wave-pipelined systems," *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 8, pp. 987–1004, Aug. 1994.

4. C. T. Gray, W. Liu, and R. Cavin III, *Wave Pipelining: Thoery and CMOS Implementation*. Kluwer, 1994.

5. W.-H. Lien and P. Burleson, "Wave-domino logic: theory and applications," *IEEE Trans. Circuits and Systems, Part II–Analog and Digital Signal Processing*, vol. 42, no. 2, pp. 78–90, Feb. 1995.

6. V. Nguyen, W. Liu, C. T. Gray, and R. Cavin III, "A CMOS multiplier using wave-pipelining," in *Proc. Custom Integrated Circuits Conference*, pp. 147–164, May 1993.

7. E. Brunvand, "Using FPGAs to implement self-timed systems," *Journal of VLSI Signal Processing*, vol. 6, pp. 173–190, 1993.

8. E. Brunvand, "Designing self-timed systems using concurrent programs," *Journal of VLSI Signal Processing*, vol. 7, pp. 47–59, Feb. 1994.

9. P. Day and V. Woods, "Investigation into micropipeline latch design styles," *IEEE Trans. on VLSI Systems*, vol. 3, no. 2, pp. 264–272, June 1995.

10. M. E. Dean, D. L. Dill, and M. Horowitz, "Self-timed logic using current-sensing completion detection (CSCD)," *J. of VLSI Signal Processing*, vol. 7, pp. 7–16, 1994.

11. S. Hauck, "Asynchronous design methodologies: an overview," *Proc. IEEE*, vol. 83, no. 1, pp. 67–93, Jan. 1995.

12. G. M. Jacobs and R. W. Brodersen, "A fully asynchronous digital signal processor using self-timed circuits," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 6, pp. 1526–2536, Dec. 1990.

13. A. J. Martin, "Compiling communicating processes into delay insensitive circuits," *Distributed Computing*, vol. 1, pp. 226–234, 1986.

14. T. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Trans. on Computer-Aided Design*, vol. 8, no. 11, pp. 1185–1205, Nov. 1989.

15. P. Patra and D. S. Fussell, "Fully asynchronous, robust, high throughput arithmetic structures," in *Proc. 8th International Conference on VLSI Design*, pp. 141–145, Jan. 1995.

16. J. T. Udding, "A formal model for classifying delay-insensitive circuits and systems," *Distributed Computing*, vol. 1, pp. 197–204, 1986.

17. T. Williams, M. Horowitz, R. L. Alverson, and T. S. Yang, "A self-timed chip for division," in *Proc. of Stanford Conference on Advanced Research in VLSI*, pp. 75–95, March 1987.

18. T. E. Williams and M. A. Horowitz, "A zero-overhead self-timed 160-ns 54-b CMOS divider," *IEEE J. of Solid-State Circuits*, vol. 26, no. 11, pp. 1651–1661, Nov. 1991.

19. J. Yuan and C. Svensson, "High-speed CMOS circuit technique", *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, pp. 62–70, Feb. 1989.

20. M. Hatamian and G. L. Cash, "Parallel bit-level pipelined VLSI designs for high-speed signal processing," *Proc. IEEE*, vol. 75, no. 9, pp. 1192–1202, Sept. 1987.

21. M. Hatamian and G. L. Cash, "A 70-MHz 8-bit×8-bit parallel pipelined multiplier in 2.5-μ CMOS," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 4, pp. 505–513, Aug. 1986.

22. M. Hatamian and K. K. Parhi, "An 85-MHz fourth-order programmable IIR digital filter chip," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 2, pp. 175–183, Feb. 1992.

23. H. B. Bakoglu, J. T. Walker, and J. D. Meindl, "A symmetric clock-distribution tree and optimized high-speed interconnections for reduced clock skew in ulsi and wsi circuits," in *Proc. of IEEE International Conference on Computer Design*, (Rye Brook, NY), pp. 118–122, Oct. 1986.

24. A. L. Fisher and H. T. Kung, "Synchronizing large VLSI processor arrays," *IEEE Trans. on Computers*, vol. C-34, no. 8, pp. 734–740, Aug. 1985.

25. E. G. Friedman and S. Powell, "Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell VLSI," *IEEE Journal of Solid-State Circuits*, vol. SC-21, no. 2, pp. 240–246, April 1986.

26. S. Y. Kung and R. J. Gal-Ezer, "Synchronous versus asynchronous computation in very large scale integrated (VLSI) array processors," in *Proc. of SPIE Symposium*, vol. 341, pp. 53–65, 1982.

644

第17章 低功耗设计

17.1 引言

历史上，数字系统的性能特征是与电路的速度和处理能力密切相关的。例如，在微处理器设计领域，微处理器的性能通常按照MFLOPS（每秒时间内浮点运算的次数）或MIPS（每秒时间内可以执行的指令数）来衡量。数字系统的成本依赖于考虑的实现策略。对于超大规模集成电路（VLSI）系统而言，硅片的面积和成本之间有一个显明的一一对应关系。于是，增加芯片实现的面积将直接导致更高的封装成本，同时降低了成品率，这两个因素提高了产品的成本。此外，系统性能的改进通常是以牺牲硅片面积为代价的。因此，历来VLSI设计者的任务始终是探索面积-时间（area-time,AT）的折中，以及在这些通常互相冲突的目标之间寻求合理的平衡。

近来，研究人员发现仅仅用面积和时间作为评价系统性能的指标是不够的。功耗是另一个非常重要的考虑指标。直到最近，相对于面积和速度指标而言，对功耗的考虑还被放在第二位。但是，近年来这种考虑方法正在开始改变，对功耗指标重要性的考虑逐渐提升到与面积和速度同等重要的高度。许多因素推动了这个趋势的发生。也许最明显的因素就是便携式电子系统的飞速发展。个人数字辅助系统、膝上型计算机和蜂窝电话在消费者中已经取得很大的成功，而且对类似其他便携设备的需求还在迅速上升之中。对于这些便携式电子系统应用，平均功耗已经变成一个最关键的设计指标。例如，用分立器件所搭建的一个便携式多媒体终端，由于没有进行低功耗的优化设计，其消耗的功率是40瓦。用先进的镍-金属-氢化物电池供电，每公斤重可以产生的电能约为65瓦*小时，支持这样的终端运行10小时就需要6公斤重电池，这是无法接受的。甚至用现在最先进的电池技术如锂离子电池，每公斤重可以产生的电能为100瓦*小时，支持这个多媒体终端运行10小时，也需要4公斤锂离子电池。因此，如果没有低功耗设计，当前和未来的便携式电子设备要么需要非常重的电池组，要么电池的寿命非常短。

即使对于非便携式的设备而言，减小功耗也会起到非常关键的作用。例如，进行过性能优化的微处理器，在时钟频率为200MHz~500MHz的情况下，消耗的功率典型值大约为40瓦~80瓦。微处理器的速度正在稳步提高，时钟速率已经达到了1GHz。在这样高的时钟速率下，它们消耗大约300瓦的功率是无法接受的，因为封装和散热设备的成本太高了。因而，除非功耗大大降低，否则因功耗而产生的热量必将限制封装和VLSI系统的性能。

除了成本以外，可靠性也是数字系统设计的一个重要问题。通常大功率系统运行时将产生很大的热量，这些热量会加剧失效机理。实验发现，相对于室温而言，运行系统的温度每升高10℃，系统失效率就提高1倍。此外，最大功率消耗是一个关键的因素，因为它决定了系统的热和电气性能的极限。因此，必须严格控制电子系统的峰值功耗。

所有前面提到的因素引发了本章对低功耗设计技术的讨论。低功耗设计技术的目标指向数字CMOS电路，因为数字CMOS电路广泛应用在工业中的便携式电子系统、DSP结构和微

处理器的设计中。数字CMOS电路的功耗可以表示为:

$$P = I_{standby}V_{dd} + I_{leakage}V_{dd} + I_{sc}V_{dd} + \alpha C_l V_{dd}^2 f_{clk} \tag{17-1}$$

其中,

- 闲置电流 $I_{standby}$ 是从电源 V_{dd} 到地线的连续直流电流。
- 泄露电流 $I_{leakage}$ 主要取决于制造工艺,产生泄露电流的原因主要有以下几条:
  (1) MOS管中源、漏扩散区和衬底之间寄生二极管产生的反偏电流;
  (2) 当栅电压小于MOS管的阈值电压时,由于反型所引起的亚阈值电流。
- 短路电流 I_{sc} 是在输出发生变化时电源到地之间的直流通路所产生的电流。
- 最后一项是电容所产生的功耗。其中α表示开关活动率,是输出端0-1变化的平均数;C_l 是输出节点的负载电容;V_{dd} 是电源电压;f_{clk} 是时钟频率。开关活动率和时钟频率的乘积又称为转移密度[2]。

646

闲置电流所引起的闲置功耗是当NMOS和PMOS连续导通时所产生的。例如,在伪NMOS 反相器中,当一个NMOS晶体管的漏端驱动另外一个处于传输晶体管逻辑中的NMOS晶体管的栅端时;或者当一个CMOS门电路的三态输入端电压泄露到电源电压和地电压之间的某一个电压值时。闲置功耗等于电源电压 V_{dd} 与流过电源电压和地之间的直流电流的乘积。

二极管泄露电流所引起的功耗正比于晶体管源漏扩散区的面积和泄露电流密度。对于1微米的工艺,其典型值是在1皮安的数量级上。对于长沟器件的亚阈值泄露电流而言,其泄露电流随着 $V_{GS} - V_t$ 的减小而指数降低,其中,V_{GS} 为栅端的偏置电压,V_t 为晶体管的阈值电压,该泄露电流可以表示为:

$$I_{st} = ke\frac{(V_{gs} - V_t)}{nV_T}\left(1 - e\frac{V_{ds}}{V_T}\right) \tag{17-2}$$

其中,k是一个工艺参数的函数,V_T为热电势(KT/q)。在正常电源电压和正常阈值电压情况下,该泄露电流是可以忽略的。但是,在电源电压降低和器件的阈值电压下降的情况下,该泄露电流就必须考虑。而且,在短沟道器件中,亚阈值电流与漏端电压指数相关,而不是与 V_{DS} 相关,V_{DS} 就是漏端与源端电压之间的差。

术语静态功耗是指闲置功耗和泄露功耗的总和。数字CMOS电路的泄露电流可以通过选择合适的器件工艺来降低。在设计像伪NMOS电路、NMOS传输晶体管逻辑电路和存储器核电路时,闲置电流就是静态功耗的关键组成部分。

一个逻辑门的短路功耗正比于输入的上升时间 t_r、下降时间 t_f、负载以及逻辑门中晶体管的尺寸。例如,对于一个没有负载的反相器,假设 $t_r = t_f = t_{rf}$,那么短路功耗 P_{sc} 可以表示为[3]:

$$P_{sc} = \frac{\beta}{12}(V_{dd} - 2V_t)^3\frac{t_{rf}}{T_{clk}} \tag{17-3}$$

647

其中,T_{clk} 是输入波形的周期;β是晶体管的增益因子,它是与工艺参数和器件的几何尺寸相关的,可以表示为:

$$\beta = \frac{\mu\epsilon}{t_{ox}}\left(\frac{W}{L}\right) \tag{17-4}$$

其中μ是沟道中载流子的有效表面迁移率，ε是栅绝缘体的介电常数，t_{ox} 是栅绝缘体的厚度，W是沟道宽度，L是沟道长度。因此增益因子β包含了工艺相关因子 $\mu\varepsilon/t_{ox}$，又包含与器件真实版图尺寸相关的几何因子 W/L。当负载为0时，短路电流最大，当负载增加时，短路电流相应地减小。已经有许多不同的技术通过用不同的逼近方法来模拟该短路电流和估计与输入信号的相关性，评价短路电流的功耗。近来，已经获得一个有用的表达式[4]，该表达式给出了短路功耗和设计参数之间的解析关系。在该方法中，用一个等效的短路电容 C_{SC} 来替换短路功耗的定义。如果选择门的尺寸使得输入和输出的上升和下降时间大约相等，那么短路功耗将小于动态功耗的15%。但是，如果希望获得高的性能，用大的逻辑门驱动相对小的负载，同时如果输入信号的上升时间长，那么短路功耗不可忽视。

对于优化设计的CMOS电路而言，容性功耗占整个功耗的绝大部分。术语动态功耗是指短路功耗和容性功耗的总和。如前所说，用等效短路电容的概念，那么动态功耗可以通过把 C_{SC} 加入 C_l 而计算出来。

本章的内容安排情况如下：17.2节阐述了概率理论的理论背景，为本章后续的章节做一个理论铺垫。17.3节描述了工艺按比例缩放对功耗的影响。在该节中说明了随着器件尺寸的下降，电源电压也不得不随之降低，以便保持一个恒定的功率密度。而且，阈值电压也必须随着电源电压而缩放，以便保持一个恒定的电路延迟。17.4节详细分析了影响功耗的不同因数。17.5节给出了目前一些用于降低功耗的技术。17.6节描述了数字CMOS电路功耗的估计技术。

17.2 理论背景

在该小节中，定义了一些简单的概率方法，用于估计数字系统的功耗。设信号 $\hat{x}(t), t\in(-\infty,+\infty)$ 为一个随机过程[5]，该信号在逻辑0和逻辑1之间随机变化。一个逻辑信号 $x(t)$ 可以被看作是随机过程 $\hat{x}(t)$的一个样本，例如，$x(t)$ 是组成 $\hat{x}(t)$的无限多个样本中的一个。在本章中，假设输入过程是严格平稳的[5]，也就是说，它的统计特性是不随时间原点的移动而变化的。 648

假设考虑的数字CMOS电路是同步运行的，例如，它们受一个全局时钟控制。假设 T_{clk} 表示时钟周期，T_{gd} 表示电路中最小的门延迟。为了捕获电路中的毛刺，时钟周期被划分为S个时间段，如图17-1所示。

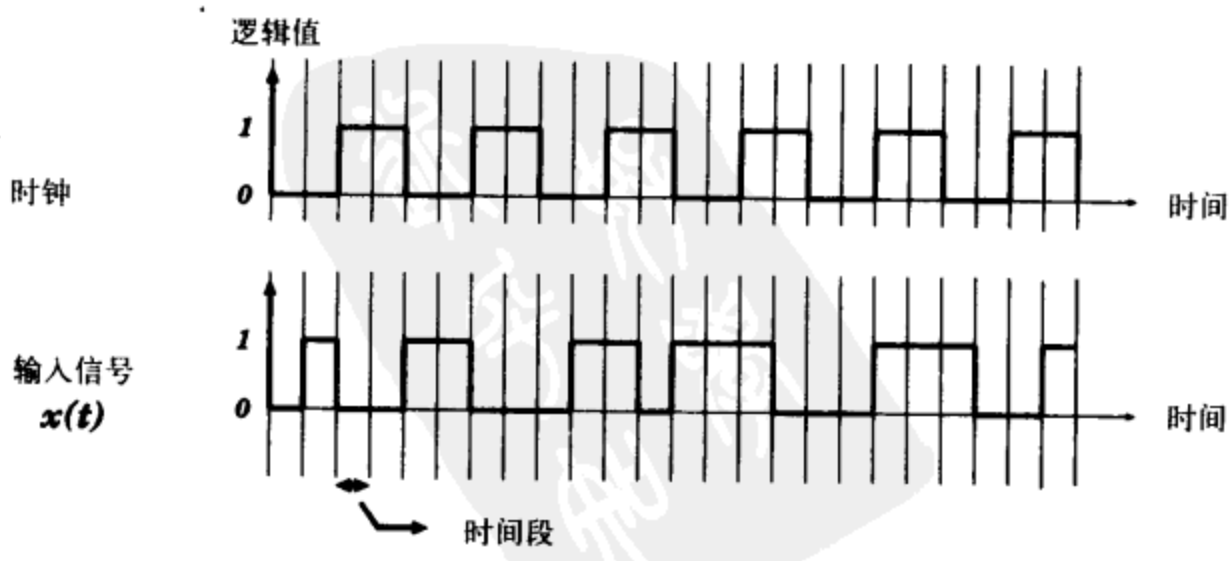


图17-1 时间段的概念

其中，

$$S \triangleq \frac{T_{clk}}{T_{gd}} \tag{17-5}$$

时间段的长度取决于用详细的器件级参数进行SPICE模拟的性能。那么，在一个给定的时间内，信号 x_i 为1的概率定义为:

$$p_{x_i}^1 = \lim_{N\to\infty} \frac{\sum_{n=1}^{N\times S} x_i(n)}{N \times S} \tag{17-6}$$

其中，N表示时钟周期的总数，$x_i(n)$ 是在时间点n和$n+1$之间输入信号 x_i 的值。那么，在一个给定的时间内，信号 x_i 为0的概率定义为:

$$p_{x_i}^0 = 1 - p_{x_i}^1 \tag{17-7}$$

649 假设信号 x_i 发生从0到1的变化。那么，与这个变化过程相关的概率定义为:

$$p_{x_i}^{0\to1} = \lim_{N\to\infty} \frac{\sum_{n=1}^{N\times S} \overline{x_i(n)}x_i(n+1)}{N \times S} \tag{17-8}$$

其他的变化概率可以用相似的方法获得。可以很容易地证明:

$$p_{x_i}^{0\to1} + p_{x_i}^{1\to0} + p_{x_i}^{1\to1} + p_{x_i}^{0\to0} = 1 \tag{17-9}$$

和

$$\begin{aligned} p_{x_i}^{0\to0} + p_{x_i}^{1\to0} &= p_{x_i}^0 \\ p_{x_i}^{0\to1} + p_{x_i}^{1\to1} &= p_{x_i}^1 \end{aligned} \tag{17-10}$$

值得注意的是在前面的方程中，$p_{x_i}^{1\to0}$ 和 $p_{x_i}^{0\to1}$ 是相等的。

从变化概率可以很容易地推导出条件概率，例如

$$p_{x_i}^{1/0} = \frac{p_{x_i}^{0\to1}}{p_{x_i}^{0\to1} + p_{x_i}^{0\to0}} \tag{17-11}$$

表示了在 $x_i(n) = 0$ 的前提下，$x_i(n+1) = 1$ 的概率。

一旦条件概率或变化概率已经知道，信号的特点就可以完全确定。

17.3 按比例缩小与功耗

按比例缩小是影响功耗的一个关键因子。像图17-2所示那样，随着工艺的按比例缩小，电源电压也应该按比例缩小，以便保持相同的功率密度。从图17-2中可以清楚地看出，尽管功率密度没有发生变化，但是，如果电源电压的缩小因子为 $1/k$，功耗的缩小因子就是 $1/k^2$。也可以清楚地看出如果电源电压不按比例进行缩小，功率密度将以立方的关系增加。

参　数	缩放效果	
	缩放 Vdd	恒定 Vdd
器件尺寸	$1/k$	$1/k$
栅厚度 t_{ox}	$1/k$	$1/k$
衬底掺杂	k	k^2
电源电压 V	$1/k$	1
电场 E	1	k
电流 I	$1/k$	k
面积 A	$1/k^2$	$1/k^2$
电容 $C=\varepsilon A/t_{ox}$	$1/k$	$1/k$
门延迟 VC/I	$1/k$	$1/k^2$
功耗 VI	$1/k^2$	k
功率密度 VI/A	1	k^3

图17-2　按比例缩小对功耗的影响

按照一阶方程[6]，电源电压的降低将导致电路延迟 T_d 的提高。

$$T_d = \frac{C_l \times V_{dd}}{\mu C_{ox}(W/L)(V_{dd}-V_t)^2} \tag{17-12}$$

图17-3给出了阈值电压为0.7V的情况下，延迟和电源电压之间的关系。从图中可以清楚地看出，当电源电压为 $V_{dd}=1.5$V 时的电路延迟是电源电压为 $V_{dd}=5$V 时电路延迟的10倍。

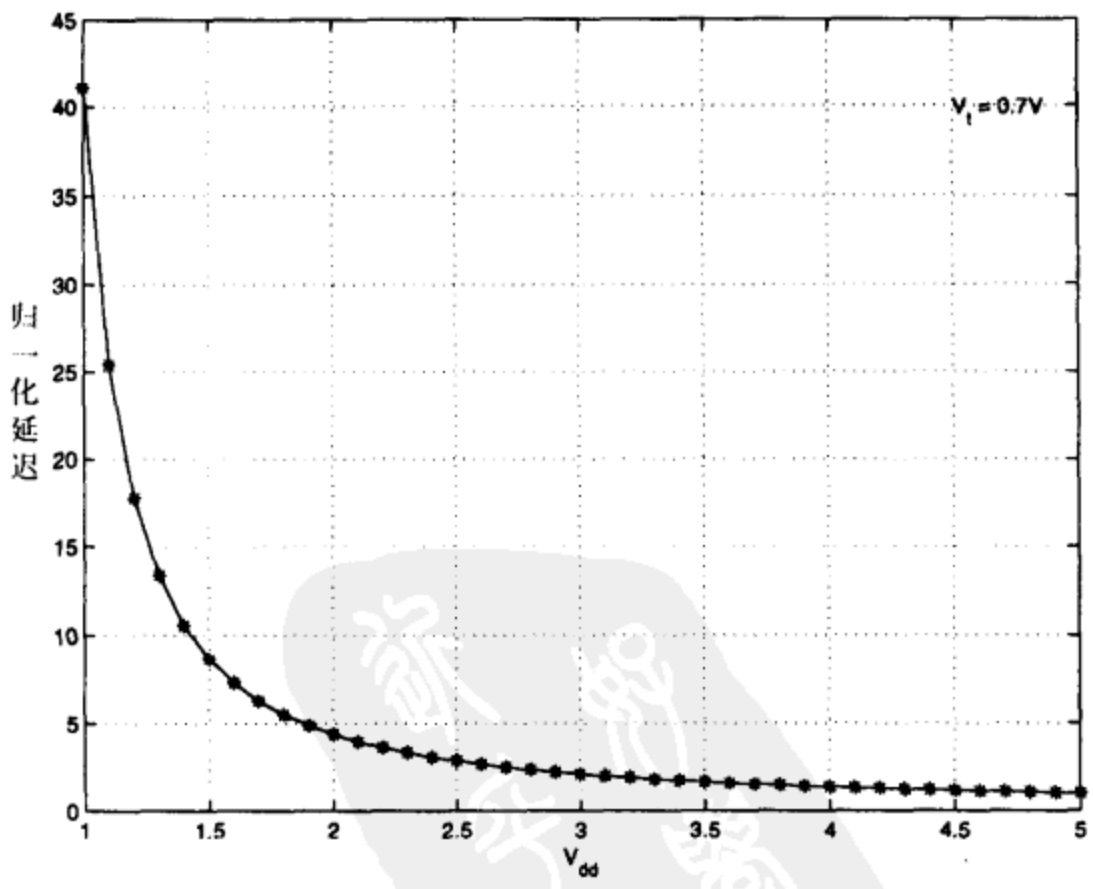


图17-3　电路延迟和电源电压之间的函数关系

一个降低电源电压而不牺牲电路速度的方法是修改器件的阈值电压 V_t。但是，阈值电压 V_t 降低的程度受限于足够的噪声裕度和亚阈值泄漏电流。通常，优化的 V_t 取决于CMOS器件的栅和对泄漏电流的控制。在室温情况下，V_t 减小80mV~100mV，亚阈值电流将增加一

个数量级[1]。作为一个经验规则，器件的关断电流至少应该比导通电流小两个数量级。这就限制了室温情况下运行的CMOS器件的 V_t 应该在0.3V左右。

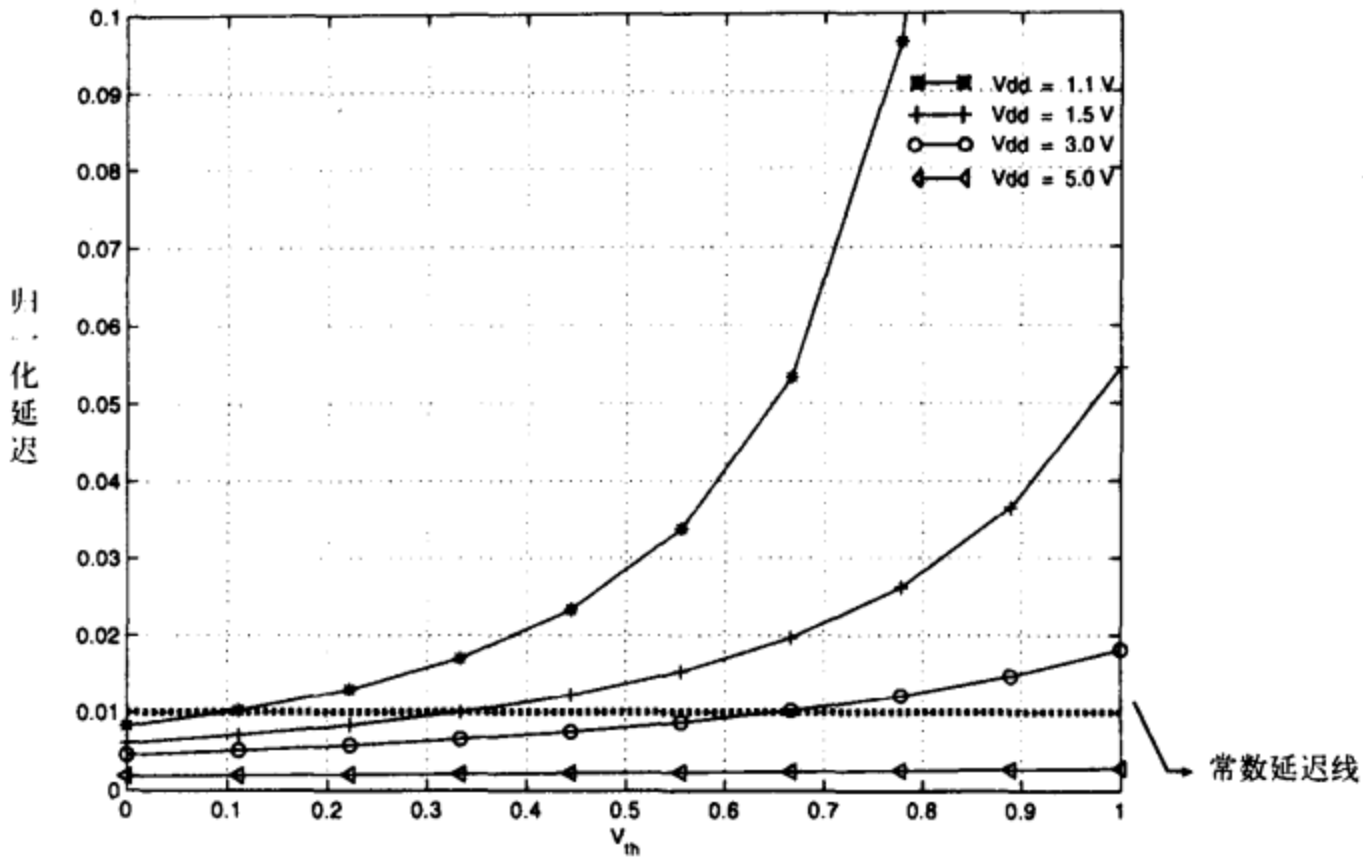


图17-4 对于不同的阈值电压，电路延迟与电源电压之间的函数关系

图17-4给出了在不同电源电压情况下延迟和 V_t 之间的关系。从图中可以清楚地看出，当电源电压 V_{dd} 为1.1V时，对于0.15V的ΔV_t，延迟将变化两倍。因此，限制 V_t 的变化小于0.05V的技术已经研制出来[7]。从图17-4中可以获得的另外一个重要特性是：为了获得常数的电路延迟，电源电压和阈值电压必须同时进行按比例的缩放。例如，从图17-4中可以看出，为了获得0.01的归一化延迟，电源电压和阈值电压必须用以下的组合：V_{dd} = 3V 和 V_t = 0.67V；V_{dd} = 1.5V 和 V_t = 0.33V；V_{dd} = 1.1V 和 V_t = 0.11V。从上面的分析可以看出，电源电压的减小在降低功耗方面起着重要和微妙的作用。

17.4 功耗分析

从17.1节可以知道决定数字CMOS电路功耗的四个关键因素。但是，降低时钟频率不是一个可选择的技术，因为所有的数字系统都有某一最小吞吐量要求。缩放电源电压的门类已经在前面的小节中讨论过了。在本节中对其他两个因素进行简单分析，它们是开关活动率和物理电容，不仅分析它们的重要性，而且分析它们在低功耗挑战性设计中的交互作用。

17.4.1 开关活动率

一个电路的开关活动率是指在一个电路中，一个时钟周期内，不同的输出节点处从0跳变到1的平均数。对于一个普通电路，估计其开关活动率是很困难的，因为，它不仅与电路的输入和电路要实现的逻辑函数密切相关，而且是电路输入之间的时间和空间相关的函数。更进一步，在输出节点信号稳定之前，发生在输出节点的毛刺或寄生信号的变化也影响开关

活动率。

开关活动率与电路输入之间相关性的关系可以通过一个简单例子来说明。如图17-5a所示，一个OR门有两个输入 x 和 y。如果输入信号是独立的，每个信号的概率是1/2，那么开关活动率 α 为3/16，因为在16个可能的输入变化情况下，只有3种可能使得OR门的输出端发生从0到1的变化，参看图17-5b。现在假设OR门的输入仅仅为00模式和11模式，而且出现这两种模式的概率是相等的，那么 α 为1/4，参看图17-5c。此外，假设输入端 x 的信号在每次输入一个0以后，接下来的输入一定是1；而输入端 y 在每次输入一个1以后，接下来的输入一定是0，在这种情况下，α 等于2/9，参看图17-5d。最后，假设 x 的值仅仅在 y 的值变化以后才发生变化，那么，α 等于1/8，参看图17-5e。图17-5c说明了门电路输入之间空间相关的情况，图17-5d说明了门电路输入之间时间相关的情况，图17-5e解释了门电路输入之间时空都相关的情况。

650 ~ 653

a)

x	y	f
0->0	0->0	0->0
0->0	0->1	**0->1☆**
0->0	1->0	1->0
0->0	1->1	1->1
0->1	0->0	**0->1☆**
0->1	0->1	**0->1☆**
0->1	1->0	1->1
0->1	1->1	1->1
1->0	0->0	1->0
1->0	0->1	1->1
1->0	1->0	1->0
1->0	1->1	1->1
1->1	0->0	1->1
1->1	0->1	1->1
1->1	1->0	1->1
1->1	1->1	1->1

b)

x	y	f
0->0	0->0	0->0
0->1	0->1	**0->1☆**
1->0	1->0	1->0
1->1	1->1	1->1

c)

x	y	f
0->1	0->0	**0->1☆**
0->1	0->1	**0->1☆**
0->1	1->0	1->1
1->0	0->0	1->0
1->0	0->1	1->1
1->0	1->0	1->0
1->1	0->0	1->1
1->1	0->1	1->1
1->1	1->0	1->1

d)

x	y	f
0->0	0->0	0->0
0->0	1->1	1->1
0->1	0->1	**0->1☆**
0->1	1->0	1->1
1->0	1->0	1->0
1->0	0->1	1->1
1->1	0->0	1->1
1->1	1->1	1->1

e)

图17-5 举例说明开关活动率与电路输入之间相关性的关系

在输出节点出现毛刺的情况下，将有可能出现附加的从0到1的变化，导致开关活动率提高。人们发现，在数据通路中有很强的运算电路，像乘法器，毛刺的功耗（或称为浪费的功耗）将高达40%。因此，当进行低功耗设计时，必须对所有这些因素进行仔细的考虑。由于毛刺与延迟有复杂的相关性，预测确切的毛刺影响是很困难的。但是，进行估计还是可能的。在本章中，利用时间段的概念来计算出现毛刺时的开关活动率。

引理17.4.1 *假设时钟周期被划分为S个时间段，从0到$S-1$。时间段持续的时间等于库中所有单元的最小门延迟。同时假设输入信号的变化仅仅发生在每一个时钟周期的开始。而且，*

假设电路的输出发生在每一个时钟周期的任何一个时间段 j 的开始，定义为 $f_j(Sn)$，其中 $j \in \{0, S-1\}$，n 表示时间刻度。那么，在出现毛刺的情况下，输出节点 f 的开关活动率[8]可以计算为：

654

$$\alpha_f = \sum_{j=0}^{S-1} \alpha_{f_{j,j+1}} = \sum_{j=0}^{S-1} p_{f_{j+1}} \left(1 - p_{f_j}\right) \left(1 - \rho_{f_{j,j+1}}\right) \tag{17-13}$$

其中，p_{fx} 表示在时间段 x 输出节点 f 为1的概率，同时，

$$\rho_{f_{j,j+1}} = \frac{E\left[f_j(Sn) f_{j+1}(Sn)\right] - p_{f_j} p_{f_{j+1}}}{\sqrt{\left(p_{f_j} - p_{f_j}^2\right)\left(p_{f_{j+1}} - p_{f_{j+1}}^2\right)}} \tag{17-14}$$

在公式（17-14）中，在时间段S时电路的输出和下一个时钟周期在时间段0时电路的输出相同。

证明：

在出现毛刺的情况下，一个节点的开关活动率可以表示为（参看习题4）：

$$\alpha_f = p_f(1 - p_f)(1 - \rho_f) \tag{17-15}$$

其中，p_f 表示输出节点 f 为1的概率，ρ_f 是节点 f 的自相关系数。因为公式（17-15）对于每一个时钟周期是正确的，那么对于任何一个时间段也应该是正确的。因此，公式(17-13)是公式(17-15)的直接扩展。

两个连续时间段的自相关矩阵可以计算为：

$$\mathbf{R}_{f_{j,j+1}} = \begin{bmatrix} E\left[f_j(Sn)^2\right] & E\left[f_j(Sn) f_{j+1}(Sn)\right] \\ E\left[f_{j+1}(Sn) f_j(Sn)\right] & E\left[f_{j+1}(Sn)^2\right] \end{bmatrix} \tag{17-16}$$

均值矢量 $\mu_{f_{j,j+1}}$ 定义为：

$$\mu_{f_{j,j+1}} = \begin{bmatrix} E\left[f_j(Sn)\right] \\ E\left[f_{j+1}(Sn)\right] \end{bmatrix} \tag{17-17}$$

从而，协方差矩阵 $\mathbf{C}_{f_{j,j+1}}$ 可以计算为：

$$\mathbf{C}_{f_{j,j+1}} = \mathbf{R}_{f_{j,j+1}} - \mu_{f_{j,j+1}} \cdot \mu_{f_{j,j+1}}^T \tag{17-18}$$

因此，利用公式(17-16)和公式(17-17)，$\mathbf{C}_{f_{j,j+1}}$ 可以表示为：

655

$$\mathbf{C}_{f_{j,j+1}} = \begin{bmatrix} E\left[f_j(Sn)^2\right] - p_{f_j}^2 & E\left[f_j(Sn) f_{j+1}(Sn)\right] - p_{f_j} p_{f_{j+1}} \\ E\left[f_j(Sn) f_{j,j+1}(Sn)\right] - p_{f_j} p_{f_{j+1}} & E\left[f_{j+1}(Sn)^2\right] - p_{f_{j+1}}^2 \end{bmatrix} \tag{17-19}$$

所以，相关系数可以计算为：

$$\rho_{f_{j,j+1}} = \frac{E\left[f_j(Sn) f_{j+1}(Sn)\right] - p_{f_j} p_{f_{j+1}}}{\sqrt{\left(p_{f_j} - p_{f_j}^2\right)\left(p_{f_{j+1}} - p_{f_{j+1}}^2\right)}} \tag{17-20}$$

■

通过一个简单的例子可以对这个理论进行最好的解释。考虑图17-6所示的一个具有三个

输入的电路。逻辑门里标注的数值定义为延迟，指的是从门的输入到计算输出所延迟的时间段数。这个电路将按照不相关的输入、时间相关的输入、空间相关的输入和时间和空间都相关的输入四种情况进行分析。为了开关活动率的分析，时钟周期被划分为两个时间段。假设输入信号a、b、c为1的概率分别表示为 p_a、p_b 和 p_c。

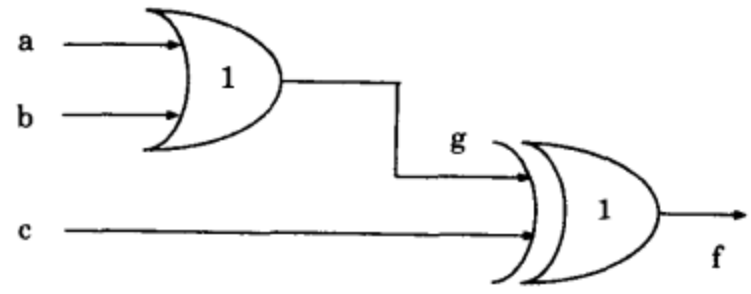


图17-6 一个说明出现毛刺时开关活动率计算的电路

A) 不相关的输入：基于延迟模型，最后的输出可以计算为：

$$\begin{aligned} f_2(2n) = f(2n+2) &= \{1-c(2n)\}\,b(2n) + \\ &\quad \{1-c(2n)\}\,a(2n)\,\{1-b(2n)\} + \\ &\quad c(2n)\,\{1-a(2n)\}\,\{1-b(2n)\} \end{aligned} \tag{17-21}$$

这里，求和项是以它们之间互相没有关系的方式表示。同样，中间的输出可以表示为：

$$g(2n+1) = g(2n+2) = a(2n) + b(2n)\,\{1-a(2n)\} \tag{17-22}$$

输出 $f_1(2n)$ 可以计算为：

$$\begin{aligned} f_1(2n) = f(2n+1) &= g(2n) \oplus c(2n) \\ &= \{1-c(2n)\}\,a(2n-2) + \\ &\quad \{1-c(2n)\}\,b(2n-2)\,\{1-a(2n-2)\} + \\ &\quad c(2n)\,\{1-a(2n-2)\}\,\{1-b(2n-2)\} \end{aligned} \tag{17-23}$$

656

在推导该表达式时，假设输入a、b、c仅仅在每一个时钟周期的开始发生变化，例如，在时间段0，2，4等的开始。利用公式(17-21)，$f_0(2n)$ 可以表示为：

$$\begin{aligned} f_0(2n) = f(2n) &= \{1-c(2n-2)\}\,b(2n-2) + \\ &\quad \{1-c(2n-2)\}\,a(2n-2)\,\{1-b(2n-2)\} + \\ &\quad c(2n-2)\,\{1-a(2n-2)\}\,\{1-b(2n-2)\} \end{aligned} \tag{17-24}$$

用前面的方程式，期望值可以计算为：

$$\begin{aligned} E\left[f_0(2n)f_1(2n)\right] &= (1-p_c)^2 p_a p_b + (1-p_c)^2 p_a (1-p_b) + \\ &\quad (1-p_c)^2 p_b (1-p_a) + p_c^2\,(1-p_a)\,(1-p_b) \\ &= \frac{1}{4} \\ E\left[f_1(2n)f_2(2n)\right] &= (1-p_c)p_b p_a + (1-p_c)p_b^2(1-p_a) + \\ &\quad (1-p_c)p_a^2(1-p_b) + (1-p_c)(p_b-p_b^2)(p_a-p_a^2) + \\ &\quad p_c(1-p_a)^2(1-p_b)^2 \\ &= \frac{5}{16} \end{aligned} \tag{17-25}$$

用同样的方法，概率值可以计算为:

$$\begin{aligned} p_{f_0} = p_{f_2} &= (1-p_c)p_b + (1-p_c)p_a(1-p_b) + p_c(1-p_a)(1-p_b) \\ &= \frac{1}{2} \\ p_{f_1} &= (1-p_c)p_a + (1-p_c)p_b(1-p_a) + p_c(1-p_a)(1-p_b) \\ &= \frac{1}{2} \end{aligned} \tag{17-26}$$

利用公式(17-14)，相关系数可以计算为

$$\rho_{f_{0,1}} = 0,\ \rho_{f_{1,2}} = \frac{1}{4} \tag{17-27}$$

同时开关活动率因子可以计算为:

$$\alpha_f = \frac{1}{4} + \frac{3}{16} = \frac{7}{16} \tag{17-28}$$

如果假设门电路用零延迟模型，那么，开关活动率因子就是 1/4。因此，有毛刺时的开关活动率将是没有毛刺时开关活动率的两倍。

B) 时间相关的输入: 这里假设输入a是时间相关的。也就是说，对于输入a，每出现一个0，接着出现的一定是1。用相似的方法，假设输入b每出现一个1，接着出现的一定是0。这就暗示了 $t_a = E[a(Sn)\ a\ (Sn-S)] = 1/3$ 和 $t_b = E[b(Sn)\ \mathrm{b}\ (Sn-S)]=0$。而且，$p_a = 2/3$，$p_b = 1/3$。利用前面的事实，可以获得:

657

$$\begin{aligned} E[f_0(2n)f_1(2n)] &= \frac{1}{4} \\ E[f_1(2n)f_2(2n)] &= (1-p_c)p_bp_a + (1-p_c)t_b(1-p_a) + \\ &\quad (1-p_c)t_a(1-p_b) + (1-p_c)(p_a-t_a)(p_b-t_b) + \\ &\quad p_c(1-2p_a+t_a)(1-2p_b+t_b) \\ &= 5/18 \\ p_{f_{0,1}} = p_{f_{1,2}} &= \frac{1}{2} \end{aligned} \tag{17-29}$$

那么，相关系数可以计算为:

$$\rho_{f_{0,1}} = 0,\ \rho_{f_{1,2}} = \frac{1}{9} \tag{17-30}$$

最后，开关活动率可以计算为:

$$\alpha_f = \frac{1}{4} + \frac{2}{9} = \frac{17}{36} \tag{17-31}$$

如果假设门电路使用零延迟模型，那么，开关的活动率就是 1/4。这个例子又一次证明了在输入之间出现时间相关的情况下，考虑出现毛刺的开关活动率将大约是没有考虑出现毛刺的开关活动率的两倍。

C) 空间相关的输入: 假设输入信号a、b具有空间相关性。特别地，假设ab仅仅可能的取值为00或11。这暗示 $a(Sn) = b(Sn)$，还有 $t_{ab} = E[a(Sn)\ b\ (Sn)] = 1/2$。

像前面一样的方式进行处理，可以计算得:

$$
\begin{aligned}
E[f_0(2n)f_1(2n)] &= (1-p_c)^2 t_{ab} + (1-p_c)^2(p_a - t_{ab}) + \\
&\quad (1-p_c)^2(p_b - t_{ab}) + p_c^2(1-p_a) \\
&= \frac{1}{4} \\
E[f_1(2n)f_2(2n)] &= (1-p_c)p_a^2 + p_c(1-p_a)^2 \\
&= \frac{1}{4}
\end{aligned} \tag{17-32}
$$

那么，相关系数可以计算为:

$$\rho_{f_{0,1}} = 0,\ \rho_{f_{1,2}} = 0 \tag{17-33}$$

658

因此，开关活动率因子可以计算为:

$$\alpha_f = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \tag{17-34}$$

值得注意的是，如果使用零延迟模型，那么开关活动率因子就是 1/4。这个例子显示，甚至在输入之间空间相关的情况下，有毛刺的开关活动率都要比没有毛刺时的开关活动率大。

D) 时间和空间都相关的输入：假设输入a、b具有时间和空间的相关性。这种相关性的一个例子就是仅仅当b发生变化时，a才跟着发生变化。结果输入 $a(n)$ 可以表示为:

$$
\begin{aligned}
a(Sn) &= a(Sn-S)b(Sn)b(Sn-S) + \\
&\quad a(Sn-S)\{1-b(Sn)\}\{1-b(Sn-S)\} + \\
&\quad \{1-a(Sn-S)\}b(Sn)\{1-b(Sn-S)\} + \\
&\quad \{1-a(Sn-S)\}\{1-b(Sn)\}b(Sn-S)
\end{aligned} \tag{17-35}
$$

期望值的计算需要计算下面的量:

$$
\begin{aligned}
t_1 &= E[a(Sn)b(Sn)] = E[a(Sn-S)b(Sn)b(Sn-S)] + \\
&\quad E[1-a(Sn-S)1-b(Sn)b(Sn-S)]
\end{aligned} \tag{17-36}
$$

$$= \frac{p_b(1-p_b-p_a)}{1-2p_b} \tag{17-37}$$

对公式（17-35）的两边同时取期望值，可以获得:

$$
\begin{aligned}
p_a &= p_b t_1 + (1-p_b)(p_a - t_1) + \\
&\quad p_b(1-p_b-p_a+t_1) + (1-p_b)(p_b - t_1) \\
&= \frac{(1-2p_b)(p_b-2t_1)}{2p_b} + \frac{1}{2}
\end{aligned} \tag{17-38}
$$

如果 $p_b = 1/2$，显然，$p_a = p_b = 1/2$。因此，对于该电路，当 $p_b = 1/2$ 时，t_1 将简化为等于 $p_b/2 = 1/4$。

$$
\begin{aligned}
t_2 &= E[a(Sn)b(Sn)b(Sn-S)] = p_b t_1 = \frac{1}{8} \\
t_3 &= E[a(Sn)b(Sn-1)] = p_b t_1 + (1-p_b)(p_b - t_1) = \frac{1}{4} \\
t_4 &= E[a(Sn)b(Sn-S)a(Sn-S)] = p_b t_1 = \frac{1}{8} \\
t_5 &= E[a(Sn)b(Sn)a(Sn-S)b(Sn-S)] = p_b t_1 = \frac{1}{8}
\end{aligned} \tag{17-39}
$$

659

$$
\begin{aligned}
t_6 &= E[a(Sn)a(Sn-S)] = t_1(2p_b-1)+p_a(1-p_b) = \frac{1}{4} \\
t_7 &= E[a(Sn)b(Sn)a(Sn-S)] = p_b t_1 = \frac{1}{8}
\end{aligned}
$$

利用 公式(17-21)、公式(17-23)和公式（17-24），可以得到

$$
\begin{aligned}
E[f_0(2n)f_1(2n)] &= (1-p_c)^2 t_1 + (1-p_c)^2(p_a-t_1) + \\
&\quad (1-p_c)^2(p_b-t_1) + p_c^2(1-p_a-p_b+t_1) \\
&= \frac{1}{4} \\
E[f_1(2n)f_2(2n)] &= (1-p_c)p_b p_a + (1-p_c)(p_b^2-p_b t_1) + \\
&\quad (1-p_c)(t_6-t_7) + (1-p_c)(t_3-t_4-t_2+t_5) + \\
&\quad p_c(1-2p_a-2p_b+2t_1+t_6+t_3-t_4+p_a p_b+p_b^2- \\
&\quad p_b t_1 - t_7 - t_2 + t_5) \\
&= \frac{3}{8}
\end{aligned}
\tag{17-40}
$$

相应的概率可以很容易地计算为:

$$
\begin{aligned}
E[f_0(2n)] &= E[f_1(2n)] \\
&= p_b(1-p_c) + (1-p_c)(p_a-t_1) + p_c(1-p_a-p_b+t_1) \\
&= \frac{1}{2}
\end{aligned}
\tag{17-41}
$$

最后，开关活动率可以计算为:

$$
\alpha_f = \frac{1}{4} + \frac{1}{8} = \frac{3}{8} \tag{17-42}
$$

做一个相似的分析可以知道，如果假设门电路使用零延迟模型，那么，开关活动率因子就是 1/4。因此，我们可以看出，即使在输入信号时间和空间都相关的情况下，也可以用这里推荐的方法计算在毛刺出现情况下的开关活动率。

电路风格也是一个影响功耗的主要因素。上面的分析都是基于用静态CMOS电路风格实现所有的逻辑门。如果是用动态的CMOS电路风格实现所有的逻辑门，那么开关活动率的结论也就不一样了。作为一个例子，假设图17-5中的OR门用动态CMOS电路实现，如图17-7。如果输入是不相关的，那么这个OR门的开关活动率就仅仅是输出为1的概率。这是因为在动态电路中，逻辑门在求值之前都是先进行预充电的。于是，对于两输入的OR门，α_f 等于3/4。读者可能还记得用静态逻辑实现OR门的开关活动率是3/16。因此可以看出，用动态逻辑设计的电路开关活动率将比用静态逻辑设计的电路开关活动率高很多。尽管动态逻辑电路有更大的开关活动率，但是它们只有很少的毛刺。

660

17.4.2 物理电容

一个数字电路的功耗直接正比于从其输出端看进去的物理电容。但是，在抽象的行为级或逻辑级对电容进行估算是很不精确的，因为它需要估算负载电容。*然而，在一个单元库里*，负载电容并没有映射到逻辑门。因此，解析的模型被开发以用于估计物理电容。该解析模型

是电路输入、电路扇出、电路复杂度和工艺信息的函数。

当器件尺寸缩小时，互连线的电容将显得很重要。同样，由于缺乏详细的布局布线信息，甚至在完成工艺映射以后，估计互连线的电容仍然是一个很困难的问题。

从直觉上来看，通过用小的面积、更短的连线和更小的器件就可以减小物理电容。例如，通过利用资源共享，门的尺寸优化和逻辑最小化技术就可使得面积降低。通过相同子功能的提取，寄存器共享等技术可以减小互连线的电容。尽管更小的器件有助于物理电容的减小，但是对速度却有负面影响。因此，需要一个合理折中。

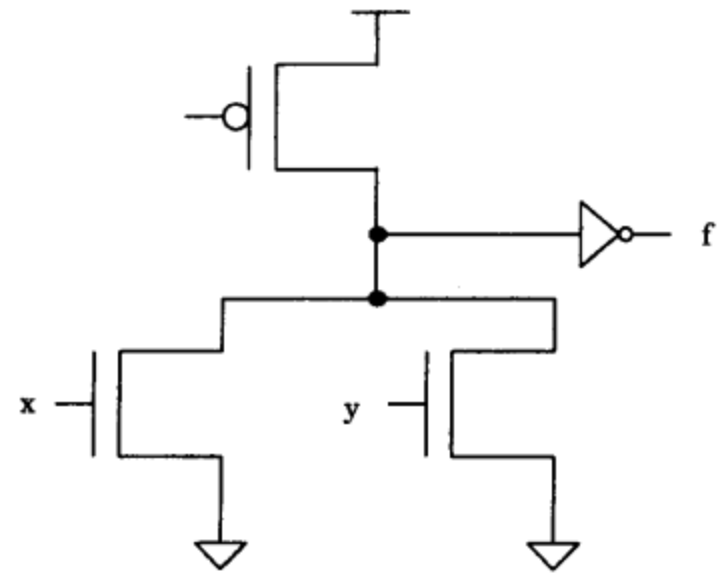


图17-7 用动态逻辑实现的一个OR门

661

17.5 功耗降低技术

功耗优化是指在数字电路设计中不同的抽象级别——从软件和算法级别到版图级别——减小功耗的问题。传统的算法转移如流水线和并行处理可通过用更低的电源电压来降低功耗（参见第3章）。在算法级（参见第9章）或数值级（参见第13章），通过强度减小变换减小电容也可以降低功耗。本节中讨论一些更新的降低功耗技术。

17.5.1 路径平衡

为了减小电路中毛刺的活动率，汇聚到每一个门的所有真正路径延迟一定要粗略地平衡，因为路径平衡可以在门的不同输入端产生接近同时开关，因此在门输出端消除可能的竞争，如图17-8所示。进而该方法可以减小电路的平均功耗。路径平衡可以在工艺映射之前或者映射之后实现。在工艺之前映射，可以通过分解和选择性的压平来实现。在工艺之后映射，可以通过插入延迟和引脚重新排序来实现。

图17-8 图解路径平衡效应

选择性压平的观点是通过压平一个节点的扇入数，导致那个节点输出的到达时间发生变化。逻辑分解可以最小化那些驱动高电容节点的输入节点之间的级别差。延迟插入过程是试图平衡电路中所有路径的延迟。在延迟插入中的一个关键问题是用最小数量的延迟元件来获得寄生开关活动率的最大降低。最后，改变引脚安排平衡路径延迟。这是可能的，CMOS门

662

电路的延迟特性的变化是引起输出发生变化的输入引脚的函数。

17.5.2 晶体管和逻辑门尺寸优化

大家都知道，功耗直接正比于晶体管的尺寸，因此，晶体管的尺寸越大，功耗也就越大。因此，为了获得最小的功耗，在整个设计中一定要用最小的晶体管。但是，这将导致电路速度极慢。在给定的延迟约束下，晶体管尺寸的优化问题简化为寻找功耗最小的解。

[9]中给出了基于线性可编程的方法，用于基于产生器的门尺寸优化。在[10]中，该方法被扩展用于处理建立和保持时间约束。该方法忽略了短路电流，在分析中没有考虑输入上升时间效应。在[11]中，基于线性可编程的方法被扩展应用于处理短路电流。

在[12]中给出了在静态CMOS版图中进行晶体管尺寸优化的方法，用于最小化短路和短路容性功耗。在驱动给定负载的门电路中，功率优化的晶体管尺寸可能比最小尺寸晶体管要大。Borach 等也推导出功耗-延迟优化的晶体管尺寸，并提出一个贪婪算法用于计算功率优化电路。提出该算法的这些作者对每一个门电路计算出初始功耗优化的晶体管尺寸。如果功耗最小的版图满足延迟约束，该过程就结束；否则，用功耗-延迟方法优化晶体管尺寸，直到延迟约束被满足。利用该方法，作者报道说整体功耗可以降低15%~20%。

[13]中提出一个基于数据库的门电路尺寸优化的启发式算法，用于在给定延迟约束的情况下最小化功耗。这里，以最小尺寸门的版本作为初始值。然后，尺寸优化沿着有负时间空闲的路径调整，以便最小量地提高电路的开关电容值时，满足延迟约束。或者，可以从最快的可能设计开始，然后尺寸优化沿着有正时间空闲的方向降低。通常，一个动态可编程的方法可以用于建立功耗-延迟折中曲线，然后执行门尺寸的选择，使得延迟约束被满足，而且开关电容最小化。在[14]中，一个满足时序约束的电路作为初始值。然后，门（不必在关键路径上）的尺寸朝着功耗降低的方向调整。

663

17.5.3 晶体管重排序

大家都知道，库里的门电路有功能上等价的引脚，例如输入引脚之间可以互换而不改变门的输出函数。但是，这些等价的引脚之间可能有不同的引脚负载和与引脚相关的延迟。因此，在CMOS逻辑门中，信号到引脚的布置将使得通过门电路的传输延迟是可变化的。在一个CMOS逻辑门中，如果忽略由于对源/漏到体扩散区的电容充电和放电产生的寄生功耗，那么高开关活动率的输入引脚应该与那些有低输入电容的引脚相匹配。该方法对半定制的库不是很有效，因为逻辑上等价的引脚之间所对应的引脚电容值差别很小。反之，作为输入信号的引脚布置和开关活动率的函数，寄生功耗是变化的[15]。这里，对于一个考虑内部功耗的门电路而言，寻找最小功耗-引脚布置的优化问题得到了解决。只要功能上等价的引脚数不超过6，该方法就是很有效的。

另一个方法是把可能的控制值中最大概率的输入信号输入到靠近门输出端的晶体管[11]。该方法的理念是这个晶体管将经常被关断，因此阻止内部节点进行不必要的充电和放电。在[16]中推荐的另一个方法是把具有最高开关活动率的输入端布置在最靠近输出端的位置。该方法的理念是，如果布置那样的一个信号接近电源端将导致更大的功耗。在[17]中指出，如果这两个规则之间发生冲突，那么具有最高比值（假设控制值的概率与发生变化的概率之间的比值）的输入端应该安排在最靠近输出端的位置。

17.5.4 用于低功耗的重定时

重定时（参见第4章）是最早在[18]中提出的，在保持输入-输出功能不变的情况下，该技术通过在电路内移动寄存器，可以改进吞吐量。基本目的就是重新分配电路中的寄存器用于最大化吞吐量。

[19]中提出用重定时来最小化开关活动率，这是基于对寄存器的观察。寄存器的输出发生变化的几率要比寄存器输入发生变化的几率小很多。考虑图17-9中的例子。在图17-9a中，电路在触发器之前没有毛刺，而在图17-9b中，电路在触发器之前有毛刺。结果，在触发器的输入端后一种情况的开关活动率更高。在第一种情况中，有效的开关电容值是 $\alpha_A C_f + \alpha_f C_B + \alpha_B C_C$，然而后一种情况的有效开关电容值是 $\alpha_A C_B + \alpha'_B C_f + \alpha'_f C_C$。根据模块A、B和C的实现功能，有一种实现的开关活动率更低。这个方法可以推广应用到大的电路，启发可以用于对电路进行重新定时，进而获得最小的开关活动率。 664

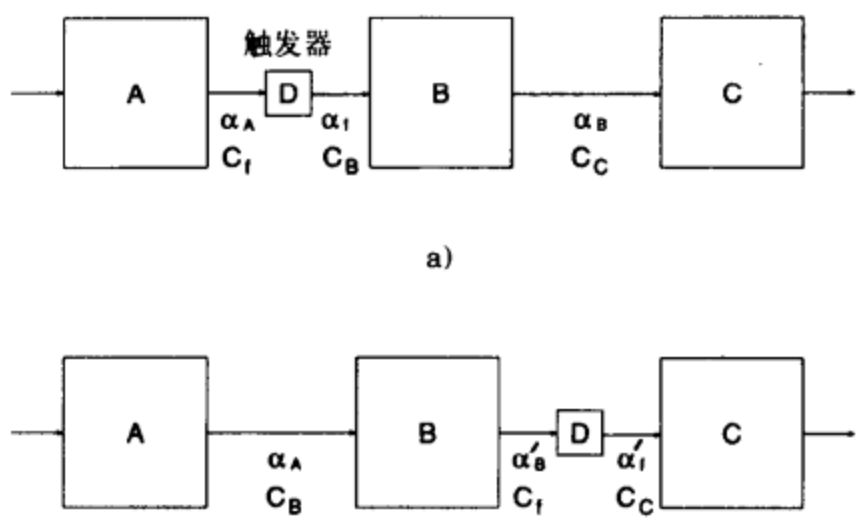


图17-9 图解重定时效应

17.5.5 电压按比例缩小和多电源电压

电压水平的按比例缩小在未来的数字系统设计中将成为一个重要问题。主要的驱动力是希望在一个芯片上生产复杂的、高性能的系统和便携式与无线系统市场的指数式增长。不同的ASIC和存储器将计划降低电源电压，用于满足很紧的功耗约束。该方法的一个主要问题是在降低电源电压的情况下，组成系统的所有芯片组的可利用性。通过在芯片上混合和匹配不同的电源电压，这个问题可以在一定程度上获得解决。

在[20]中，讨论了两个CMOS器件和电压按比例缩小的可能性，其中被优化的一个用于获得最高的速度，另一个是折中高性能而获得极低的功耗。作者指出，低功耗方案类似于电场的按比例缩小理论。通过按比例缩小CMOS器件到亚微米区域，与5V高性能的0.6微米器件相比，可以在速度和功耗-延迟积方面获得改进。使用多电源电压的理念可以利用下面的例子来说明。 665

例17.5.1 考虑图17-10中的系统，假设所有模块是相同的。假设系统运行在3V的电源电压时，系统中的每个模块有一个时间单位的关键路径。工艺的阈值电压 V_{th} 是0.5V。假设最多可以用3个电源电压。同时假设可以利用的最小电源电压是1.33V。用多电源电压来获得最小的功耗，同时不改变系统6个时间单位的延迟。忽略负载效应。

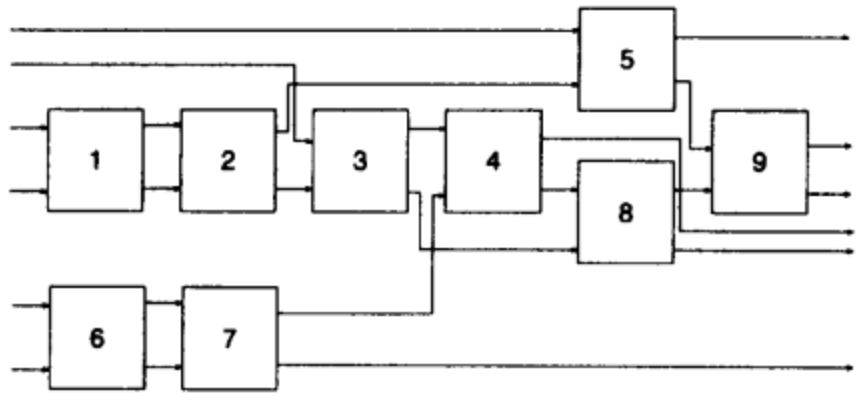


图17-10 用例子说明多电源电压的使用

解：大家知道，CMOS电路的延迟正比于如下表达式：

$$t_d \propto \frac{V_{dd}}{(V_{dd}-V_{th})^2} \tag{17-43}$$

因此，对于3V的电源电压和0.5V的阈值电压，延迟 $t_d = 12/25$。因为延迟是6个时间单位，模块5、6和7可以比1个时间单位延迟更慢的速度运行。

为了保持6个时间单位的延迟，通过观察可以知道，模块6和7的运行延迟可以为 $1.5t_d$，模块5的运行延迟可以为 $3t_d$。

对于 $1.5t_d$ 的延迟，电源电压可以减小，新的电源电压可以通过求解下面的表达式来获得：

$$\frac{V_{dd}}{(V_{dd}-0.5)^2} = \frac{18}{25} \tag{17-44}$$

因此，对于 $1.5t_d$ 的延迟，新的电源电压 $V_{dd} = 2.28\text{V}$。同样，对于 $3t_d$ 的延迟，新的电源电压可以通过求解下面的表达式来获得：

$$\frac{V_{dd}}{(V_{dd}-0.5)^2} = \frac{36}{25} \tag{17-45}$$

结果为 $V_{dd} = 1.53\text{V}$。

666 现在，系统运行将用三个电源电压，模块1、2、3、4和8用3V的电源电压，模块6和7用2.28V的电源电压，模块5用1.53V的电源电压。因为功耗正比于电源电压的平方，初始的功耗可以计算为：

$$\text{初始功耗} \propto 9\times(3)^2 = 81 \tag{17-46}$$

最后的功耗可以计算为：

$$\text{最后功耗} \propto 6\times(3)^2 + 2\times(2.28)^2 + (1.53)^2 = 66.74 \tag{17-47}$$

因此，可以看到，在没有牺牲性能的前提下，通过使用多电源电压可以降低功耗大约18%。

■

17.5.6 双/多-V_{th}

众所周知，CMOS电路的传输延迟可以近似表示为:

$$T_d = \frac{C_l V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \tag{17-48}$$

其中 α 代表速度饱和效应，对亚微米MOSFET来说其典型值为1.3。

以上表达式说明功耗和延迟之间存在一个折中，但是如果仔细选择 V_{dd} 和 V_{th}，就可以获得更好的折中，而且要尽量减小 V_{dd} 和 V_{th} 的波动。尽管在传统的CMOS设计把 V_{dd} 和 V_{th} 当作常数，而在低功耗高性能设计中就必须对它们进行优化。这里引出两个重要的考虑：（1）低 V_{dd} 时，由于 V_{th} 波动引起的最坏情况速度降低[7],[21]（2）低 V_{th} 引起闲置功耗的增加[22]~[24]。

为了解决这些问题，提出了几个方案，包括多阈值电压CMOS（MTCMOS）方案、可变阈值电压CMOS（VTCMOS）方案和弹性开启电压 CMOS（EVTCMOS）方案。

通常，在PC中当芯片没有被激活时，功耗管理控制器会切断它的电源。MTCMOS方案采用两个阈值电压：逻辑电路高速工作时采用低阈值电压；闲置模式采用高阈值电压以关断亚阈值漏电流。由于片上寄生电容比片外寄生电容小得多，所以片上电源控制执行速度更快以获得高效的功耗管理。该方法的缺点是需要大量的晶体管来进行电源控制，尤其在电源电压低于0.9V时由于高 V_{th} 的晶体管在激活模式下工作得太慢而使得面积牺牲极其严重。图17-11是一个工作于MTCMOS方案的反相器。

667

在VTCMOS中，可以通过调节衬底偏置 V_{BB} 来动态改变 V_{th}。通常在工作模式下，V_{BB} 被用来补偿阈值电压的波动，而在闲置模式用更大的 V_{BB} 来增大阈值电压从而关断亚阈值泄漏电流。图17-12是一个工作于VTCMOS方案的反相器。

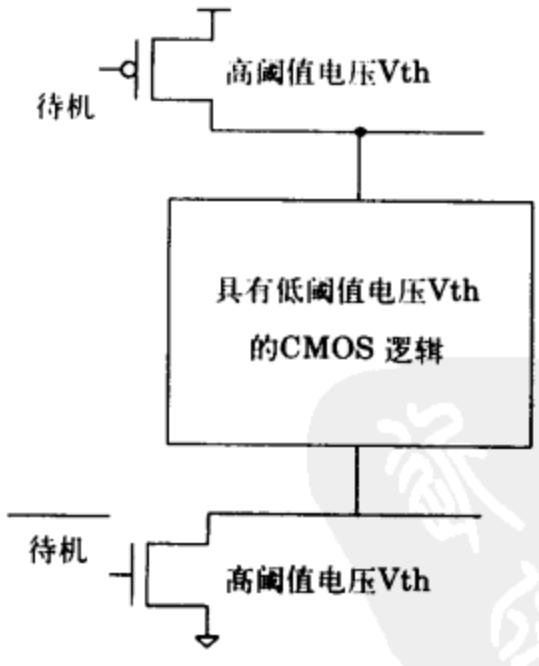


图17-11 多阈值CMOS反相器

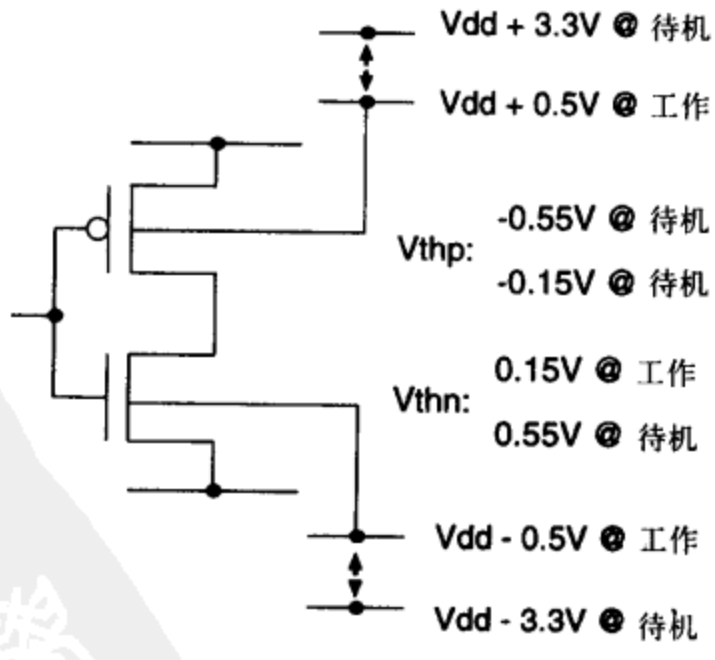


图17-12 可变阈值CMOS反相器

EVTCMOS按照这样的方式同时控制电源电压和 V_{BB}，即当 V_{dd} 降低时，V_{BB} 变得更大，从而提高了 V_{th}，降低了闲置功耗。该方法中，内部电源通过栅极受控的源随器（source-follower）nMOS和pMOS晶体管来提供。

这三种方案的本质区别在于VTCMOS控制衬底电压，而另外两种方案控制电源线。由于

流过衬底的电流比流过电源线的电流小得多，所以用来控制衬底偏置的电路规模也小得多，相应的面积和速度损失可以忽略。用在VTCMOS阵列中的锁相环和SRAM表明，衬底噪声影响可以忽略。用VTCMOS方法设计的离散余弦变换宏电路的功耗比传统CMOS设计低一个数量级。

17.5.7 时钟

在很多同步应用中，时钟消耗了大量的功率。一般这是惟一始终开关的信号并且经常要驱动很大的时钟树。而且，很多时候时钟的开关导致了很多不必要的开关活动率。因此，近来很多电路的设计采用可控时钟，也就是说在某些情况下，相对于主时钟而言，源自主时钟的其他时钟可以慢下来或完全停下。因此电路可以分割成有各自时钟的小模块。使用这种方法节省功耗与具体应用有关，但是效果很显著。

在[27]中，通过停止向待机模块输送时钟来节省功耗。把时钟树中的每个节点与二进制的0或1串联系起来，0或1串表示在该时间段时节点的工作和待机状态，通过选通或关断时钟树的枝节来控制钟控元件的工作和待机。时钟树的叶是接收器，它们的工作可以从系统的高层描述中找到。时钟树内部节点的工作方式是通过对其子树工作方式进行逐位或运算得到的。有报道说使用这种方式可以显著地节省功耗。图17-13是一个真单相时钟锁存器（tspc)[28]。这里，通过晶体管G，在时钟为低电平时输出被限制在0。如果没有这个器件，选通时钟就不可能在两个连续的系统时钟上升沿中产生上升沿。

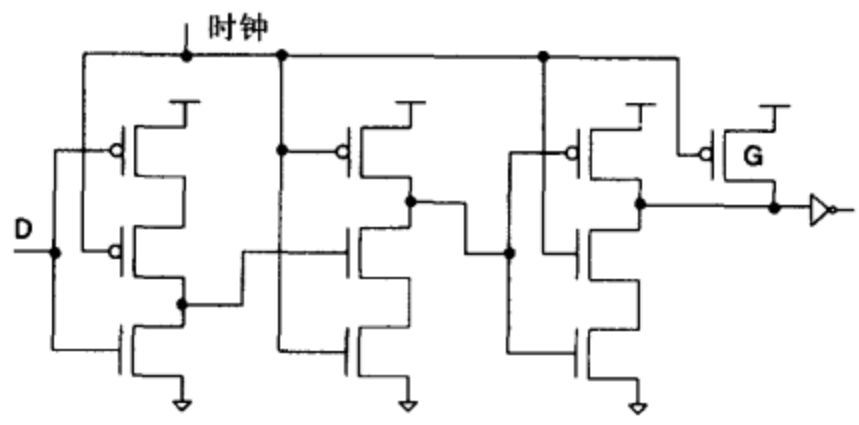


图17-13 说明采用时钟选通的电路

另一个常用的方法是使用双边沿触发器，如图17-14所示。这里，输出在两个时钟沿都发生改变，因此有效数据流速率是系统时钟的两倍。这个额外速率可用于换取电源电压的降低，从而降低功耗。

非同步结构用握手信号进行系统间的通信。因此，它们只在需要时进行工作。尽管这种方法被认为是优化的选通时钟，但是通常过头的握手控制限制了它的性能。而在一些应用中，如压缩数字磁带纠错芯片，就很容易达到性能要求，并且非同步方法产生的消耗只有相应的同步方法的1/5 [29]。

17.5.8 电路风格

一般来说，一个给定的逻辑函数可以用多种不同的逻辑风格实现，例如静态CMOS、动态CMOS、传输晶体管、差分级联电压开关逻辑，等等。在这些逻辑风格中，传输晶体管逻辑风

格最适合低功耗设计，因为相对于传统的静态CMOS电路，传输晶体管逻辑需要更少的晶体管。例如，静态CMOS全加器需要40个晶体管，而传输晶体管逻辑只需28个晶体管。因此研究人员研究出了各种各样的传输晶体管逻辑风格，如互补传输晶体管逻辑（CPL）[30]、摆动恢复传输晶体管逻辑（SRPL）[31]，以及带传输晶体管门的差分级连电压开关（DCVSPG）。

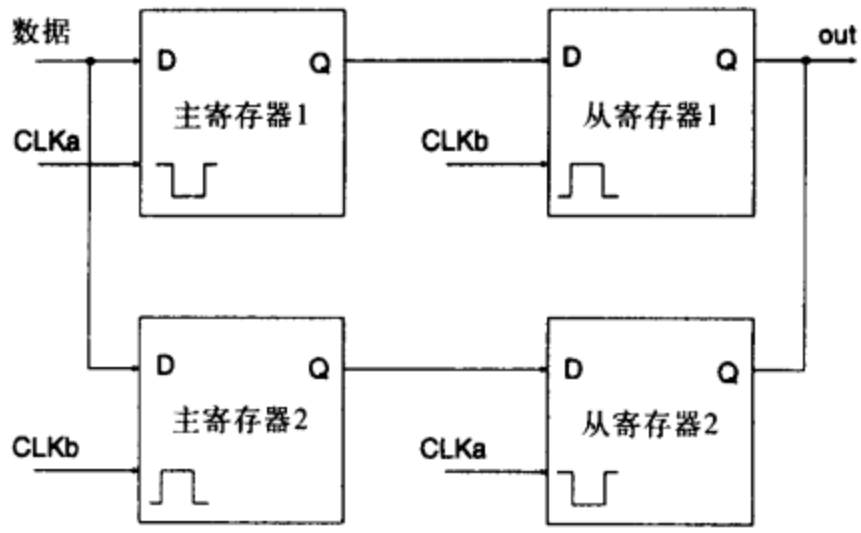


图17-14 双边沿触发器

CPL设计只使用nMOS传输管，因此高电平降低了一个 V_{th}。在输出级用CMOS反相器来补偿信号电平的降低并增强输出驱动能力。尽管输出电平被恢复了，降低的高电平却增加了CMOS反相器的泄漏电流。为了避免这种现象，可以增加交叉耦合pMOS负载来恢复高电位并扩大工作范围。一般用小尺寸的pMOS管，以防止开关速度的降低。图17-15是一个用CPL风格设计的电路。DCVS风格与CPL风格稍有不同，它使用带有交叉耦合pMOS负载的nMOS传输晶体管逻辑，如图17-16所示。

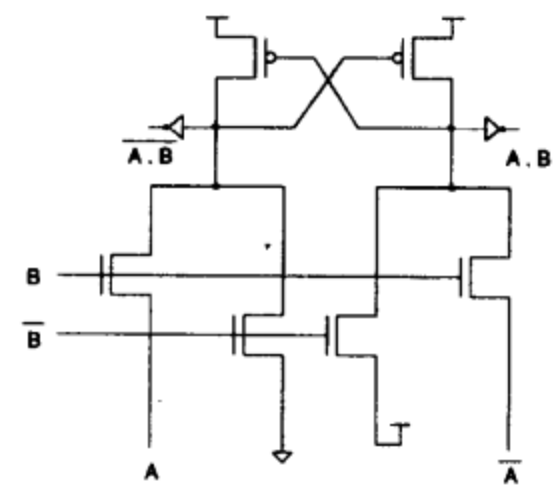


图17-15 采用CPL设计结构实现的电路

SRPL设计采用与CMOS锁存器相耦合的nMOS传输晶体管逻辑。因为CMOS锁存器以推-拉形式翻转，所以与交叉耦合CMOS负载结构相比具有更大的工作范围、更小的静态电流和更快的速度。这种方法对小负载电容的电路尤为理想。而且这些电路具有更好的抑制工艺变化的能力。图17-17是一个应用SRPL结构设计的电路。

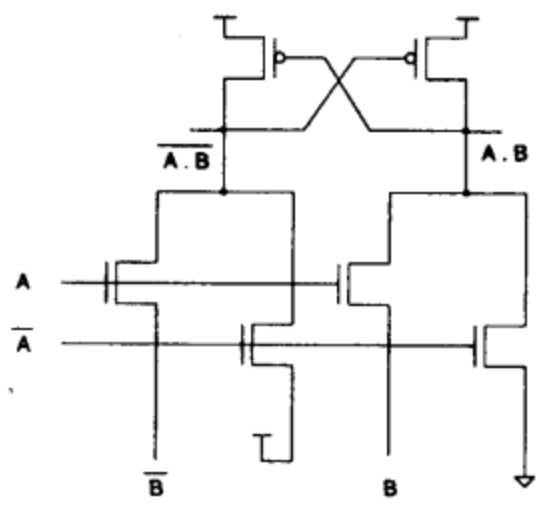


图17-16 采用DCVS设计结构实现的电路

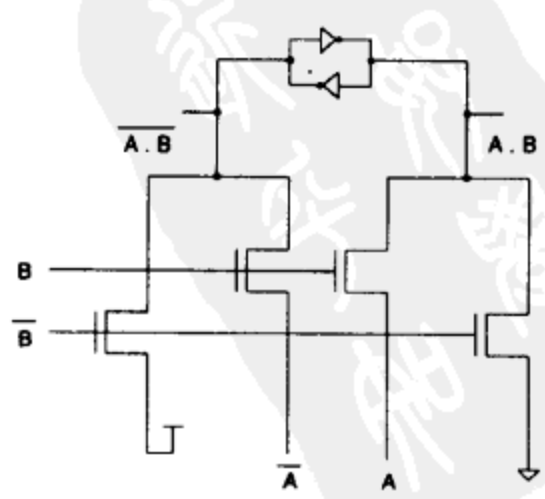


图17-17 采用SRPL设计结构实现的电路

17.6 功耗估计方法

功耗估计（power estimation）是指估计数字电路的平均功耗(average power dissipation)。理想情况下平均功耗应该包括静态功耗和动态功耗，然而在精心设计的CMOS电路中，容性功耗占主导地位，因此平均功耗一般指的是容性功耗。这与模拟为电压降低问题的瞬时功耗或最坏情况功耗的估计有很大不同。

如果没有精确的功耗预测和优化工具，就不能获得低功耗数字CMOS电路的设计。因此在设计过程中必须用CAD工具来估计功耗以满足对功率的约束，避免花费很大代价进行重新设计。功耗估计技术可以大致分为两种：基于仿真的和基于非仿真的。

17.6.1 基于仿真的方法

668 ~ 672

这些技术的主要优点在于可以自动考虑诸如竞争产生、空间/时间相关性等问题。这类方法又可分为直接仿真和统计仿真两种。

17.6.1.1 直接仿真

这类方法基本上都是采用类似SPICE的电路仿真器仿真大量的任意向量来测量平均功耗。这些方法可以处理不同的器件模型、不同的电路设计风格、三态驱动器、单/多相位时钟技术，等等。这些方法的主要缺点是需要大量的存储器和很长的运行时间。因此不适合大规模、基于单元的设计，而且很难产生一个紧缩的矢量集来计算不同节点的活动因子。

直接仿真也可以采用基于事件驱动的时序仿真算法，用晶体管级的功率仿真器进行。它采用简化的表驱动器件模型，这样对大多数电路来说，当保持误差在10%以内时，对电路进行划分就可以获得比SPICE快2~3个数量级的速度。它还能给出详细的信息，例如瞬时功耗、平均电流功耗、短路功耗、容性功耗等。

其他技术例如基于verilog的门级仿真程序可以用来计算用户定义输入序列情况下的数字电路功耗。这项技术密切依赖于ASIC库中门级宏模型和详细的门级时序分析工具的精确性。运行时间比SPICE短3~4个数量级。开关级仿真器例如IRSIM[32]在电路仿真过程中稍加改动就可以报告开关电容的情况（由此可以得到动态功耗）。它比电路级仿真技术速度快，但其通用性和准确性稍差。

17.6.1.2 统计仿真

这类技术都是建立在蒙特卡罗仿真（MCS）方法的基础上。该方法通过合理的选择输入向量减轻了模式相关问题，在电路的输入加上随机产生的输入模式并用仿真器监视T个时钟周期内的功耗。每个测量结果就是一个功耗样本，被看作是随机变量。应用中心极限理论可知，当T接近无穷大时，样本密度趋于正态曲线。一般对大部分组合电路来说，30~50个样本就能确保正态样本密度。在功耗估计中，对于预期的百分比误差 ε，给定的置信度 ϑ，样本均值μ和样本标准方差 σ，可得需要的样本数N:

$$N > \left(\frac{t_{\vartheta/2}\sigma}{\epsilon\mu}\right)^2 \tag{17-49}$$

其中 $t_{\vartheta/2}$ 定义为确保其右边的标准正态分布曲线之下的面积等于 $\vartheta/2$。在对数字电路的平均功耗进行估计时，如果误差约束和置信度要求较低，则MCS方法所需的收敛时间很短。而如

果样本密度不符合正态分布，则该方法有可能过早地收敛于错误的功耗估计值，并且该方法不能处理电路输入的空间相关性。 673

17.6.2 非仿真性方法

非仿真性模型是建立在库模型、随机模型和信息理论模型的基础上，这些方法可大致分为应用于行为级的方法和应用于逻辑级的方法两类。

17.6.2.1 行为级方法

这里，类似加法器、乘法器、寄存器和存储器的功能单元的功耗估计是直接从设计库得到的，其中每个功能单元已经用白噪声数据进行仿真，而每个时钟周期的平均开关电容也已经被计算并且存储在库里。功能单元的功耗模型可以按照其输入字长参数化。例如，加法器（或乘法器）的功耗与其输入字长是线性关系（或平方关系），尽管这种方法是不精确的，但是在比较不同的加法器和乘法器结构的开关活动率时是有效的。因此库里包含有每个模块的接口描述、参数描述、面积、延迟和内部功耗（假设输入为白噪声数据）。后者是从实际模块版图中提取电路或逻辑级模型并用一长串随机产生的输入模式进行仿真得到的。这些特性以方程或表的形式存储起来。为了提供对输入活动率敏感的功耗估计，需要产生并存储在库里的每个模块的功耗模型，必须用实际输入开关活动率进行调整。

数据输入的字级行为可以用其概率密度函数（pdf）得到。同样，数据输入之间的空间相关性可以由它们的联合概率密度（joint-pdf）得到。这个想法可以发展成概率技术用于行为级功耗估计。该方法可以总结为四步：（1）基于给定的输入向量，建立数据流图（DFG）的输入变量的联合概率密度，（2）对于DFG的某些内部弧线的组合，计算其联合概率密度，（3）用输入的联合概率密度计算DFG中每个寄存器输入的开关活动率，（4）用第3步得到的输入统计来估计每个功能模块的功耗。该方法是鲁棒的，但是联合概率密度计算的最坏情况复杂性较高，而且相关的库特征数据不够精确。

[33]~[35]描述了信息理论方法，它用类似熵的活动率测量来获得在算法和结构行为级进行快速和准确的功耗估计。熵表示了一个输入向量序列的不确定性，因此这种测量与开关活动率有关。[34]中指出在时序独立的假设下，一比特的平均开关活动率的上限是其熵的一半。 674
对于给定了输入流统计和电路结构与功能信息的控制电路和随机逻辑来说，输出熵比特是输入熵比特以及独立于结构和功能信息的缩放因子的函数。DFG的输出熵可以用与其电路规模具有线性复杂度的合成技术来计算。这种技术的主要优点是非仿真性的，因此可以提供快速而精确的功耗估计。

以上大部分技术也都适用于数据路径。[36]、[37]中给出了控制器电路的行为级功耗估计方法。基于目标实现风格的知识，例如动态、预充电伪nMOS等，该技术提供了控制器电路的快速功耗估计方法。

17.6.2.2 逻辑级方法

从17.6.2.1节的讨论可以清楚地看到数字电路的大部分功率都是在对负载电容的充放电过程中消耗的。因此，为了进行功耗估计必须确定数字电路不同节点的开关活动率 α。如果输入信号之间是时间上相互独立的，则在概率为 p_n 的条件下某个节点的开关活动率为：

$$\alpha = p_n(1-p_n) \tag{17-50}$$

如果某个节点的两个连续值是时间相关的，则开关活动率为:

$$\alpha = p_n(1-p_n)(1-\rho_n) \tag{17-51}$$

以前曾有很多研究者花费了巨大的精力计算信号的概率。[38]给出了最早的关于组合网络信号概率的计算。其中每个电路输入都有一个变量表示输入信号的概率，然后计算每个内部电路线路包含这些变量的代数表达式，这些表达式代表这些线路的信号概率。尽管这种算法简单而具有普遍性，但其最坏情况复杂度是指数性的。基于有序二进制决策图（OBDD）的精确程序也可以用来计算信号概率[39]。该程序与相应的函数图的规模呈线性关系，而与电路的输入数目呈指数关系。这里，输出节点的信号概率可以通过以下步骤计算：首先建立对应节点全局函数（例如，基于电路输入的节点函数）的OBDD，然后执行OBDD的后序遍历，产生一个对信号概率估计非常有效的计算程序。例如，假设 x_1、x_2、x_3 和 x_4 是四输入异或门的输入，则用OBDD计算的输出概率如图17-18所示，并可表示成

675

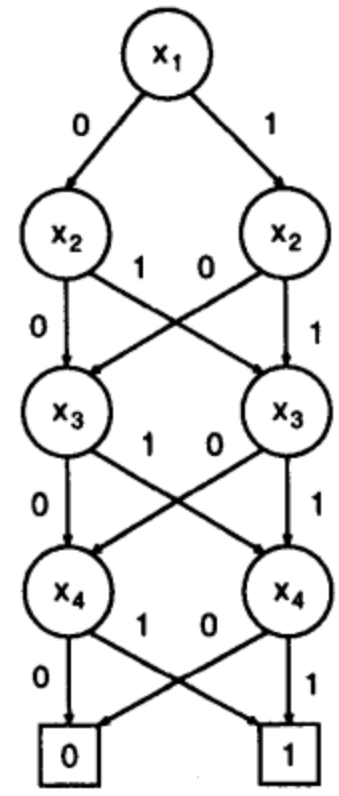


图17-18 用OBDD计算信号概率

$$\begin{aligned} p_z = {} & \overline{p_{x_1}}\,\overline{p_{x_2}}\,\overline{p_{x_3}}\,p_{x_4} + p_{x_1}p_{x_2}\,\overline{p_{x_3}}\,p_{x_4} + \overline{p_{x_1}}\,p_{x_2}p_{x_3}p_{x_4} + \\ & p_{x_1}\,\overline{p_{x_2}}\,p_{x_3}p_{x_4} + \overline{p_{x_1}}\,\overline{p_{x_2}}\,p_{x_3}\,\overline{p_{x_4}} + p_{x_1}p_{x_2}p_{x_3}\,\overline{p_{x_4}} + \\ & \overline{p_{x_1}}\,p_{x_2}\,\overline{p_{x_3}}\,\overline{p_{x_4}} + p_{x_1}\,\overline{p_{x_2}}\,\overline{p_{x_3}}\,\overline{p_{x_4}} \end{aligned} \tag{17-52}$$

其中 p_{x_i} 代表输入信号的概率。

以上所有技术都针对电路的稳态行为，忽略了竞争和毛刺，因此前面的技术定义为基于零延迟模型的技术。前面也有一些工作是针对实际延迟模型进行估计的。在[40]中，给定组合逻辑的精确功耗估计是通过产生一系列符号函数来表示布尔条件而实现的，布尔条件就是电路节点在一对输入向量下，假定在不同时间点产生的所有值。[41]引入了概率波形的概念，该波形由事件列表组成，事件列表仅仅是从最初的稳态到最终的稳态时间内的一系列转移跳变沿，其中每个事件被指定了一个概率值。节点的概率波形是该节点所有可能逻辑波形集合的紧凑表示。[2]中的基于布尔差分等式的有效算法可以将转移密度从电路的输入传输到输出。电路中每个节点的转移密度 $D(y)$ 可以表示为:

676

$$D(y) = \sum_{i=1}^{n} P\left(\frac{\partial y}{\partial x_i}\right) D(x_i) \tag{17-53}$$

其中 y 是节点的输出，x_i 是节点输入，y 与 x_i 的布尔差分函数给出了 y 与 x_i 相关的所有组合。尽管该方法很有效，但它的前提是假设 x_i 之间是独立的。该假设是不正确的，由于电路的再收敛扇出结构使得 x_i 之间是相关的。这个问题可以通过仍然假设电路输入之间独立并且用这些独立的输入描述 y 来解决。尽管这样提高了精确性，但计算布尔差分项的代价很大。在精确和效率之间折中的办法是将 y 用一系列电路的中间变量来表示。

17.6.2.3 层次化方法

在[42]的基础上，本章介绍一个数字电路功耗估计的随机方法和基于本方法的叫做层次化能量分析的工具（HEAT工具）。本方法的突出特点是可以在短时间内对包括乘法器、除法器等在内的大型数字电路进行功耗估计。用随机方法对大型数字电路进行功耗估计的典型办法是将它们用状态转移图（STD）表示。但这将是个艰巨的工作，因为随着节点数的增加状态数会指数增长。因此数字电路被分解成子电路，子电路再用STD表示。这样将会大大减少STD中的状态数，因此计算时间按数量级减小。例如一个典型的博茨（Booth）乘法器（用全加器、编码器和多路选择器组成）分成三个子集合，第一个子集合包含全加器，第二个子集合包含编码器，第三个包含多路选择器。然后属于每个子集合的电路用STD来表示，这有利于解析表达式的产生，用于对STD中每个节点的状态更新。然后用HSPICE计算STD中与每个跳变沿相关的能量，通过将STD中所有相关沿的能量求和来计算属于给定子集合电路的总功耗。对所有子集合重复该过程，将所有子集合的能量相加就可以计算出数字电路的总能量。通过求取该电路的总能量与总时间的比值可以得到平均功率的估计。

考虑一个由规则单元阵列组成的典型数字电路，如图17-19所示。该阵列可以看作是按行列排列的子电路之间的互联，则整个电路的能量可以通过对单个子电路的能量求和得到。该方法的步骤总结为如下算法。 677

算法17.6.1输入：电路中的行数、列数、子电路类型、参数等，亦即信号和所有输入信号的条件概率。
输出：估计的平均功耗

```
est_power () {
总能量 = 0;
for r = 1 to 行数
   for c = 1 to 列数
      用一个STD来模拟子电路 (r, c);
      通过把STD看作一个不可约马尔可夫链，利用MATLAB，从子电路 (r, c) 的输入信号参数中计
      算出稳态概率;
      用稳态概率和MATLAB计算STD中边沿的沿活动率;
      用SPICE计算与STD的边沿相关的能量 (r, c);
      /* 该步骤仅执行一次 */
      总能量 = 总能量 + 能量 (r, c);
      计算子电路 (r, c) 的输出信号参数;
   end;
end;
```

$$\text{平均功耗} = \frac{\text{总能量}}{\text{该能量耗费所用的时间}}\}$$

■

状态转移图建模

这里介绍一个用STD表示数字电路的系统化方法。建模过程是通过推导出相应的数字电路中所有节点状态更新的解析表达式而实现的。

(1) 静态CMOS NOR门　考虑如图17-20所示的典型静态CMOS NOR门，其中 x_1 和 x_2 分别代表两个输入信号，x_3 代表输出信号。从图17-20中可以清楚地看出节点2和节点3的值会在0和1之间变化，在这些节点出现的对电容的充放电使得我们推导出这些节点的状态更新

算术方程:

$$node_2(n+1) = (1 - x_1(n)) + x_1(n) * x_2(n) * node_2(n) \tag{17-54}$$

$$node_3(n+1) = (1 - x_1(n)) * (1 - x_2(n)) \tag{17-55}$$

用前面的等式可以获得NOR门的STD，如图17-21所示，例如，这里 S_1 代表$node_2=node_3=0$的状态，边沿 e_1 代表从状态 S_1 到 S_3 的转移（开关活动率）。

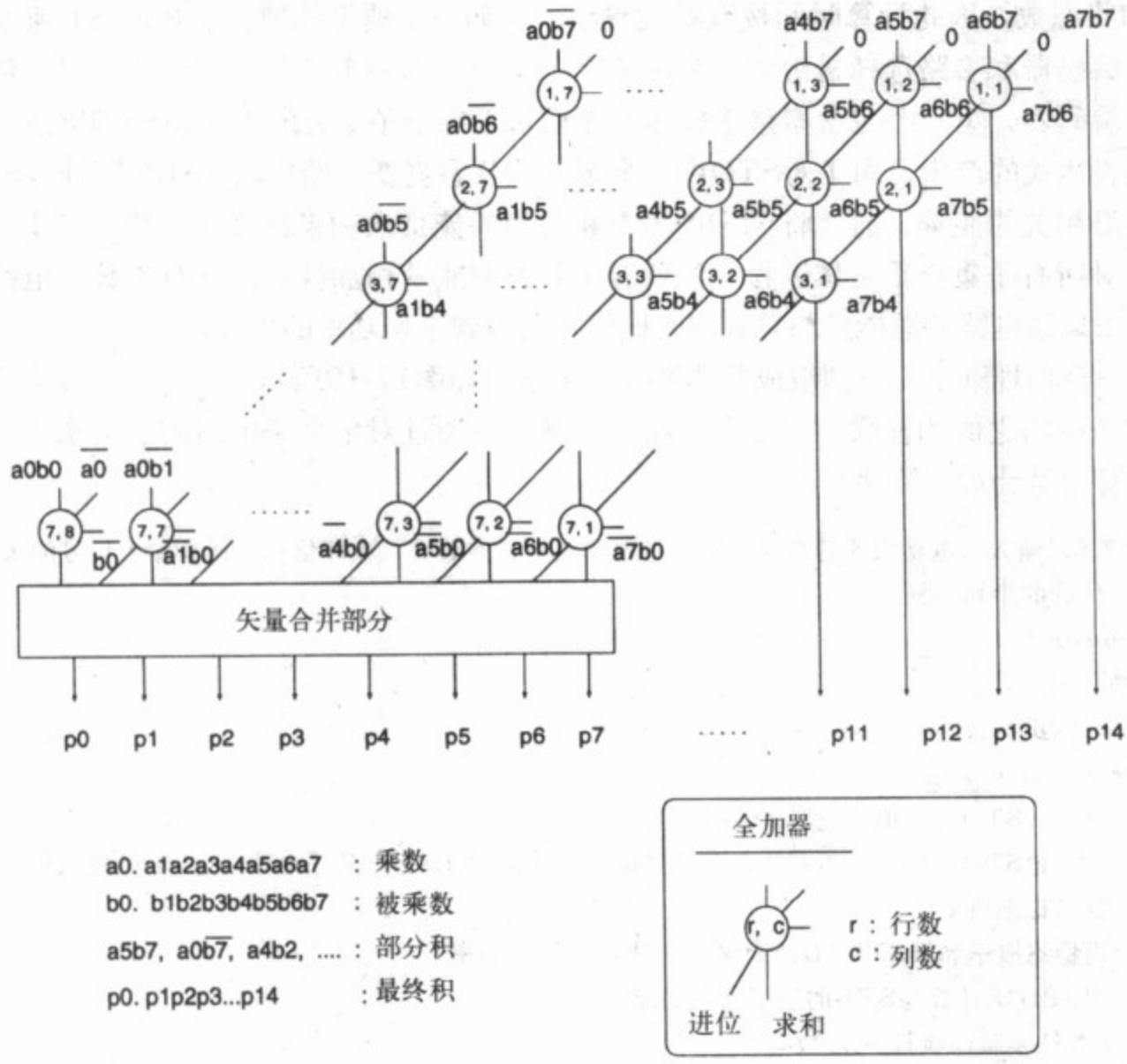


图17-19 8×8 Baugh-Wooley乘法器

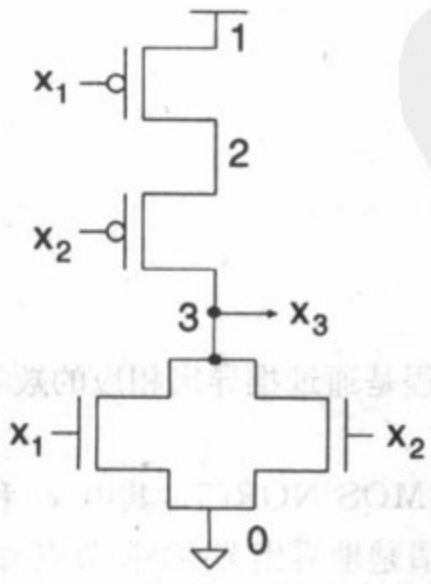


图17-20 静态CMOS NOR门

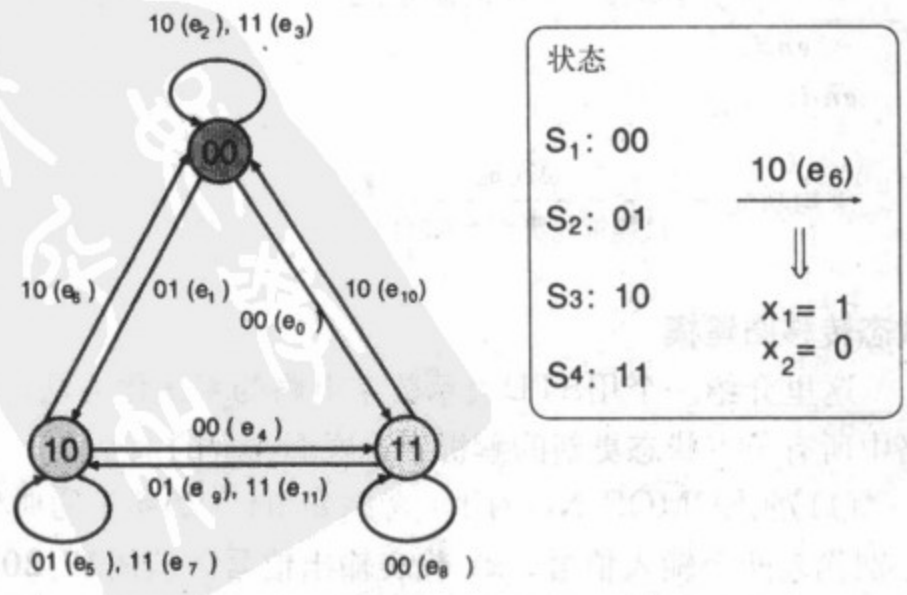


图17-21 静态CMOS NOR门的状态转移图

(2) 静态CMOS NAND门　静态CMOS NAND门如图17-22所示，x_1 和 x_2 分别代表两个输入信号，x_3 代表输出信号。静态CMOS NAND门的状态更新方程可表示为:

$$node_2(n+1) = (1 - x_2(n)) * (x_1(n) + (1 - x_1(n)) * node_2(n)) \tag{17-56}$$

$$node_3(n+1) = 1 - x_1(n) * x_2(n) \tag{17-57}$$

等式(17-56)和(17-57)可以用来获得NAND门的STD，如图17-23所示。

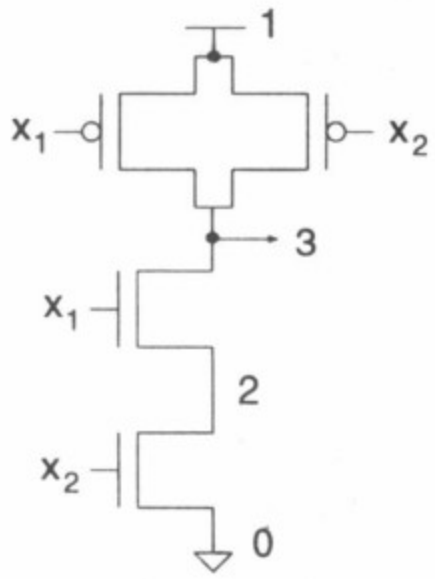


图17-22　静态CMOS NAND门

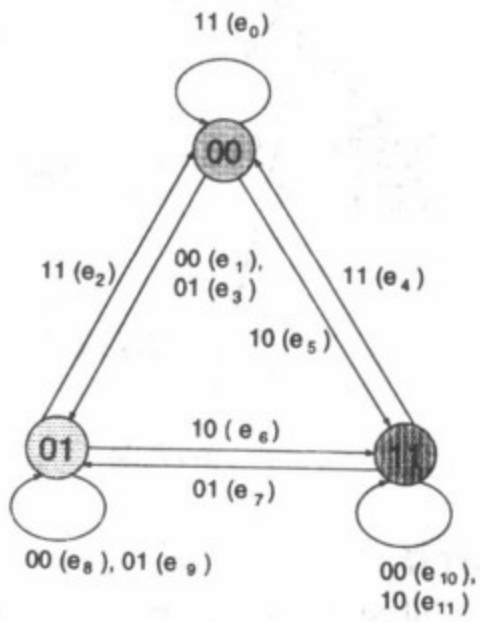


图17-23　静态CMOS NAND门的状态转移图

(3) 静态CMOS 全加器　考虑如图17-24所示的典型静态CMOS全加器。从图中可以看出该结构由进位产生部分和求和产生部分组成。

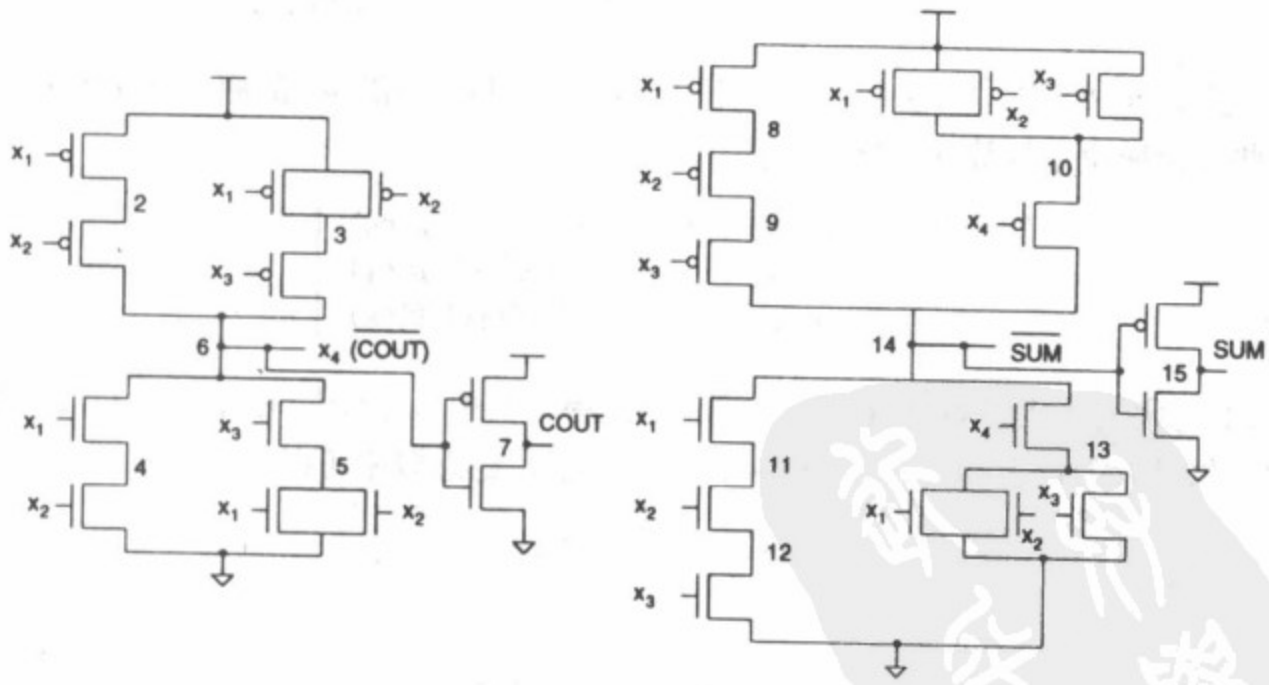


图17-24　静态CMOS全加器

可以用类似的方法产生全加器进位和求和部分的状态更新方程，然后可以得到两部分独立的状态转移图。例如，全加器进位部分的状态转移图如图17-25所示。这里，为了方便只给出了很少的几个跳变沿。由图17-25可知,该STD包含8个状态，每个状态与全加器进位部分的6个节点相关，每个边沿与全加器的3个输入相关。

678 ~ 681

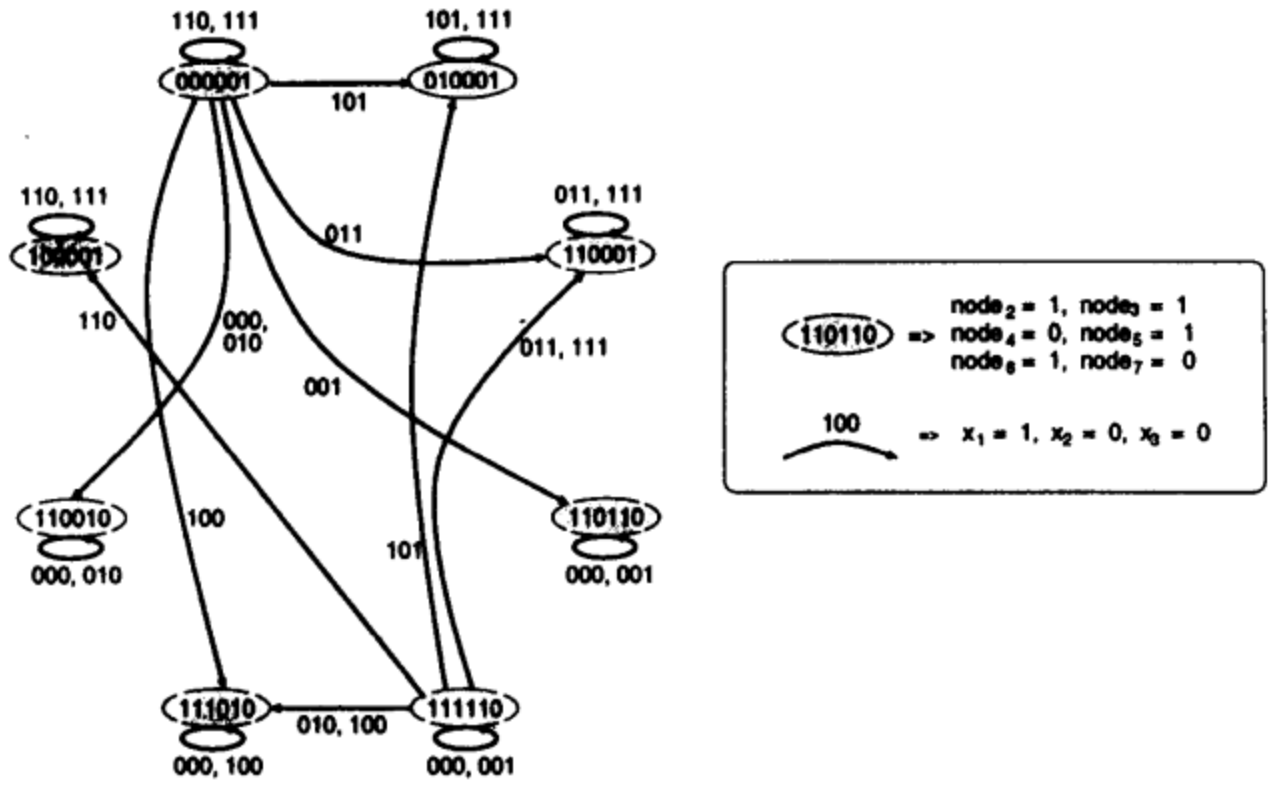


图17-25 静态CMOS全加器进位部分的状态转移图

稳态概率计算

基于不可约马尔可夫(irreducible Markov)链的方法被用来计算STD中不同状态的稳态概率。考虑图17-21中CMOS NOR门的STD，这里，假设输入信号 x_1 和 x_2 是独立的，与不同边沿相关的概率 $p(e_j)$ 计算如下:

$$p(e_j|e_j : x_m = q, x_n = r) = (q * p^1_{x_m} + (1-q) * p^0_{x_m}) \times (r * p^1_{x_n} + (1-r) * p^0_{x_n}) \tag{17-58}$$

其中 $j \in \{0, 11\}$, $m,n \in \{1, 2\}$ 及 $q, r \in \{0, 1\}$。例如，STD中边沿 e_1 的概率是 $p(e_1) = p^0_{x_1} * p^1_{x_2}$。然后用这些边沿概率来计算状态转移矩阵 $\prod_{nor}$:

$$\Pi_{nor} = \begin{bmatrix} p(e_2)+p(e_3) & p(e_1) & p(e_0) \\ p(e_6) & p(e_5)+p(e_7) & p(e_4) \\ p(e_{10}) & p(e_9)+p(e_{11}) & p(e_8) \end{bmatrix} \tag{17-59}$$

682 这里元素 $\prod_{nor_{ij}}$ 代表从状态 S_i 到状态 S_j 的转移概率，其中 $i, j \in \{1, 4\}$ 并且 $i, j \neq 2$。把转移图用不可约马尔可夫(irreducible Markov)链表示之后，稳态概率可以由下式求得:

$$P_S = P_S * \Pi_{nor} \tag{17-60}$$

其中

$$P_S = [\ P_{S_1} \quad P_{S_3} \quad P_{S_4}\ ] \tag{17-61}$$

代表不同状态的稳态概率。求解公式（17-60）的一个简单方法是首先计算与 $\prod^T_{nor}$ 相关的特征值。这样，对应于特征值1的归一化特征向量应该是稳态概率向量 P_S，需要说明的是 P_S 是用MATLAB计算得来的。

用上述马尔可夫（Markov）模型计算的稳态概率可以用来计算边沿活动率 EA_j（其中 $j \in \{0, \#ofedges-1\}$）。例如，对于NOR门而言，边沿活动率数值可以用MATLAB计算如下:

$$EA_0 = P_{S_1} * N * S * P(00/10) + EA_3 * (P(00/11) - P(00/10)) \quad (17\text{-}62)$$

$$EA_1 = P_{S_1} * N * S * P(01/10) + EA_3 * (P(01/11) - P(00/10)) \quad (17\text{-}63)$$

$$\vdots$$

$$EA_7 = P_{S_3} * N * S * P(11/01) + (EA_{11} + EA_7) * (P(11/11) - P(11/01)) \quad (17\text{-}64)$$

$$\vdots$$

$$EA_{11} = P_{S_4} * N * S * P(11/00) \quad (17\text{-}65)$$

其中，例如假设 $x_1(n) = x_2(n) = 0$，$P(11/00)$ 表示 $x_1(n+1) = x_2(n+1) = 1$ 的概率。用这里介绍的方法得到的边沿活动率数值的误差小于1.5%。

STD中每个边沿的能量计算

本节给出用SPICE计算STD中与每个边沿相关的能量计算的算法。该算法的第一步是识别初始状态和导致产生该状态的输入序列。两个标志向量，一个表示状态，另一个表示STD中定义的每个边沿。一方面，只要状态是第一次出现，就让状态标志向量置位，另一方面，只要相应的边沿发生变化就让边沿标志向量置位。变量 i 储存状态数，变量 k 储存输入序列数。例如在图17-21中 i 可以从1变化到4（对应于状态S_1到S_4），而k可以从1变化到4（对应于输入序列00，01，10，11）。一个称为edge_mat的边沿矩阵中储存输入序列的行导致STD中边沿的变化。本算法中的步骤总结如下：

683

算法17.6.2输入：子电路的STD；初始状态（init_state），输入到子电路的输入端数（num_inputs），初始化的边沿矩阵（edge_mat）。

输出：STD中每个边沿的能量。

```
energy_edge() {
状态标志和边沿标志复位到0;
i = init_state; k = 1;
while（仍有边沿标志未置位时）
  m = 新状态;
  if（相应于输入k的边沿标志矢量没有置位）
    将标志矢量置位;
    更新 edge_mat;
    if（相应于状态m的标志没有置位）
      将相应于状态m的标志置位;
      更新 edge_mat;
      prev_state (i) = i;
      i = m; k = 0;
    else
      更新edge_mat;
    end;
  end;
  k = k + 1;
  if (k > 2^num_inputs)
    k = 1;
    i = prev_state (i);
  end
```

```
end; }
/* 矩阵edge_mat已经形成，它的行是在STD中导致边沿往返移动的输入信号序列 */
rows = 矩阵edge_mat中的行数;
cols = 矩阵edge_mat中的列数;
for j = 1 to rows;
   对于输入序列形成的矩阵edge_mat (j, cols-1)，运行SPICE;
   energy1 = 求解的能量;
   对于输入序列形成的矩阵edge_mat (j, cols)，运行SPICE;
   energy2 = 求解的能量
   W_j = energy2 - energy1;
end
```

■

684 对于图17-21中的NOR门，用这个算法求得的初始状态为11，边沿矩阵edge_mat为

$$edge_mat = \begin{bmatrix} 00 & 01 & 10 & 00 \\ 00 & 01 & 10 & 01 \\ 00 & 01 & 10 & 10 \\ 00 & 01 & 10 & 11 \\ 00 & 01 & 00 & \\ 00 & 01 & 01 & \\ 00 & 01 & 10 & \\ 00 & 01 & 11 & \\ 00 & 00 & & \\ 00 & 01 & & \\ 00 & 10 & & \\ 00 & 11 & & \end{bmatrix} \begin{matrix} \rightarrow e_0 \\ \rightarrow e_1 \\ \rightarrow e_2 \\ \rightarrow e_3 \\ \rightarrow e_4 \\ \rightarrow e_5 \\ \rightarrow e_6 \\ \rightarrow e_7 \\ \rightarrow e_8 \\ \rightarrow e_9 \\ \rightarrow e_{10} \\ \rightarrow e_{11} \end{matrix} \tag{17-66}$$

那么，与子电路相关的能量可以通过加权求和STD中代表子电路的不同边沿的相关能量而得到，表示如下：

$$energy = \sum_{j=0}^{\#\ of\ edges\ -\ 1} W_j * EA_j \tag{17-67}$$

输出信号参数的计算

用层次化方法进行功耗估计的最后一步是计算子电路输出的信号参数。这可以用一个简单例子来进行很好的说明。考虑两个级联的NOR门，如图17-26所示。令 $x_1(n)$ 和 $x_2(n)$ 分别代表输入信号 x_1 和 x_2 在时间点 n 的二进制值，然后就可以计算信号 x_3 的值，它将是第二个NOR门在时间点（n+1）的输入。因此如果 $x_3(n)$ 和 $x_4(n)$ 在所有时间点都被定义，则第二个NOR门的信号特征就可以计算出来。为了用公式（17-67）来计算第二个NOR门的能量，我们使用了先前计算获得的第一个NOR门的 W_j 值和第二个NOR门的新获得的 EA_j 值。这使得能量值计算可以在很短的时间内完成。

该方法可以很容易地应用于0型和1型加法器以特定形式级联设计的乘法器（除法器）。

负载和布线考虑

上述方法的一个缺点是它没有直接考虑负载和布线电容效应。本节我们给出一个考虑了这些效应的方法。

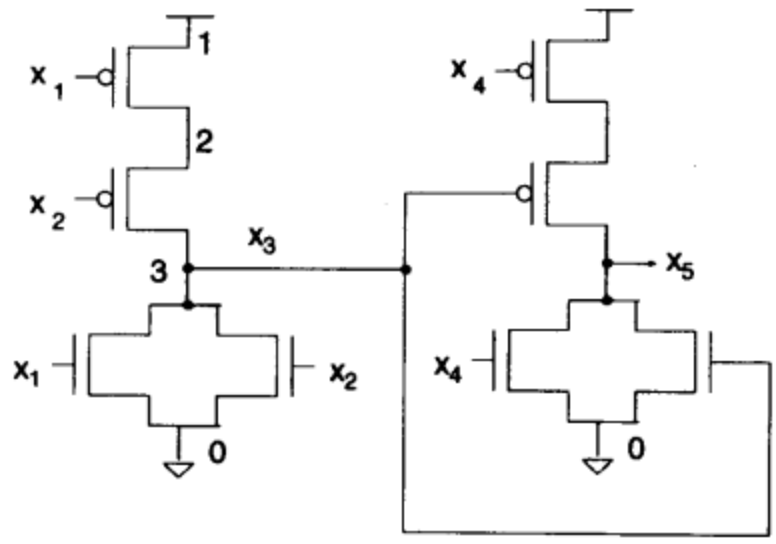


图17-26 级连的两个NOR门

考虑如图17-27所示的CMOS数字电路，这里，加入了等效负载/布线电容。

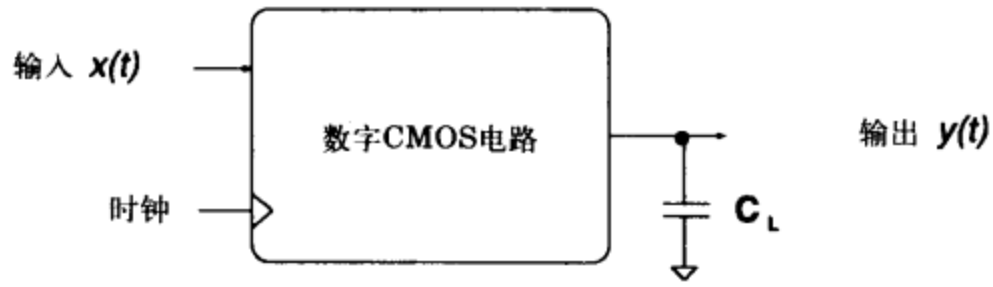


图17-27 有负载效应的电路

该方法需要重新计算适当位置带有负载电容的CMOS电路状态转移图中的边沿能量。这个思想是在计算边沿能量时，仿真负载效应。因此，我们可以看出，在边沿能量计算中稍加改动，就可以将负载效应考虑进去。

该方法的主要特点是采用SPICE来仿真负载效应，因此，可以用精确的器件模型来考虑负载效应。该方法是在用SPICE估计边沿能量之前，加入负载/布线电容。这样就考虑了负载效应。

时序电路的功耗估计

在本节中，用于组合电路的算法也被扩展应用到处理时序电路，一个时序电路包括一个组合模块和一些也能用STD表示的存储单元，例如触发器。

考虑如图17-28所示的边沿触发D触发器。这里D代表输入信号，Q代表输出信号，$\phi_{1,2}$代表两相不重叠时钟信号。从图17-28中可以清楚地看到，D触发器可以看作是由不同时钟控制的两个相同锁存器的级联。因此，只要用STD表示一个锁存器就足以进行功耗估计。第一个锁存器状态更新的算术表达式如下: 685~686

$$node_2(n+1) = D(n) * \phi_1(n) + (1 - \phi_1(n)) * node_4(n) \tag{17-68}$$

$$node_3(n+1) = 1 - node_2(n+1) \tag{17-69}$$

$$node_4(n+1) = node_2(n+1) \tag{17-70}$$

采用上述公式可得到锁存器的STD，如图17-29所示。

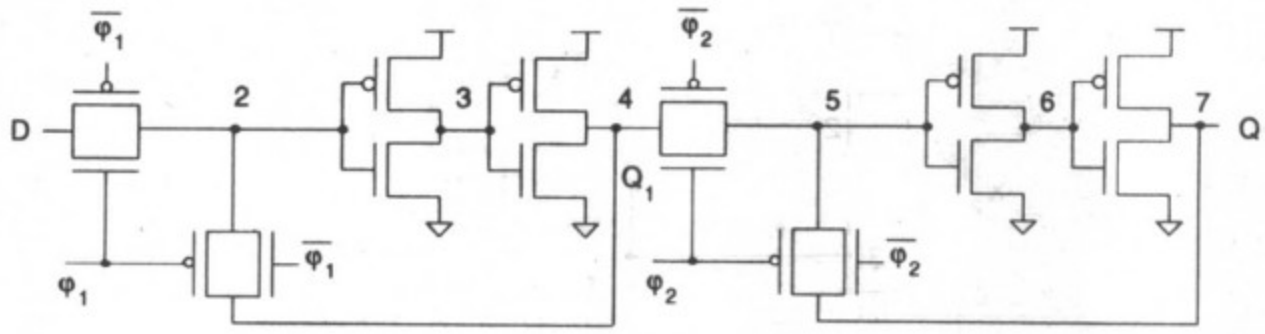


图17-28 边沿触发D触发器

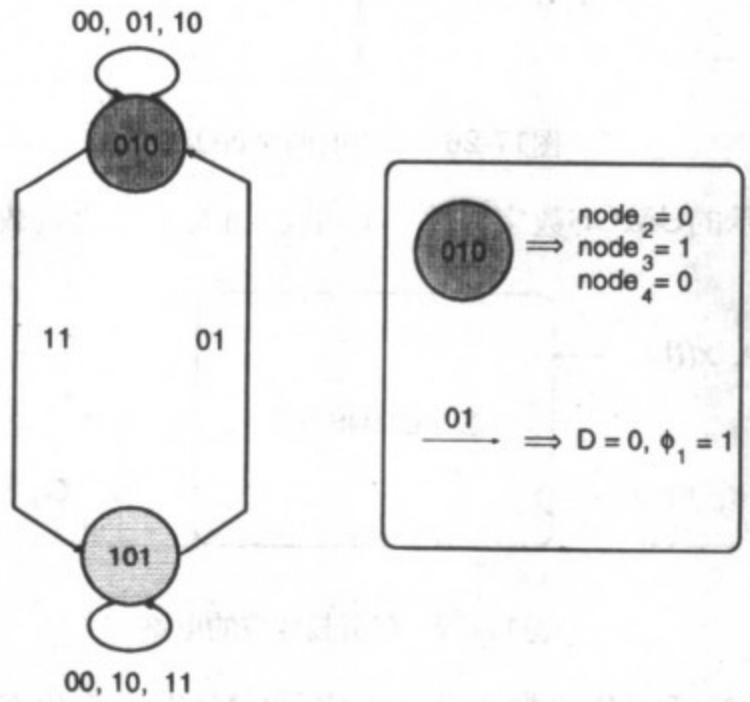


图17-29 锁存器的状态转移图

这里，在某个时间点，状态代表节点 $node_2$、$node_3$ 和 $node_4$ 的值。例如，S_1 代表节点值 $node_2 = 0$、$node_3 = 1$ 和 $node_4 = 0$ 的状态。与边沿相关的值代表序列D、phi_1。有趣的是尽管锁存器中有三个节点，但是，只有两个状态。直觉上，这意味着锁存器的出现减少了毛刺的活动率。那么，通过将 phi_1 用 phi_2 代替，D 用 Q_1 代替就可以很容易地得到第二个锁存器的STD。

687

一旦触发器被定义好了，下一步就是仅仅仿真整个时序电路而无需计算任何能量值。这使得在所有可能的时间段，可以计算直接和反馈输入信号的值，然后通过考虑这些值就可以决定转移概率。结果，输入信号值之间的时序相关性就被考虑了进去。那么，用于组合电路功耗估计的方法也就可以用于时序电路的功耗估计。

17.7 结论

本章对关于低功耗DSP系统的设计进行了全面的讨论。提出了不同的功耗优化技术，例如路径平衡、晶体管尺寸优化、晶体管重排序、电路风格等。这些技术可以显著地降低数字系统的功耗。最后提出了一种用于数字系统功耗估计的工具。该工具由于具有层次化特点，所以快速而准确。对于少量的系统性能退化，在算法级也可以用近似处理降低功耗，其滤波器长度或DFT的长度可以动态减小[43]。而且，通过动态算法选择，像字长这样的参数也可以减小，这里，系统SNR与低功耗进行了折中[44]。

17.8 习题

1. 给定一个一阶Markov过程，证明对于信号 x_i，概率 $P^0_{x_i}$ 和 $P^1_{x_i}$ 可以用条件概率表示为:

$$p^0_{x_i} = \frac{p^{0/1}_{x_i}}{p^{0/1}_{x_i} + p^{1/0}_{x_i}}, \quad p^1_{x_i} = \frac{p^{1/0}_{x_i}}{p^{0/1}_{x_i} + p^{1/0}_{x_i}} \tag{17-71}$$

2. 假设一个一阶Markov过程，证明对于信号 x_i，转移概率可以用条件概率表示为: 688

$$\begin{aligned} p^{0\to0}_{x_i} &= \frac{p^{0/1}_{x_i} p^{0/0}_{x_i}}{p^{0/1}_{x_i} + p^{1/0}_{x_i}}, \quad p^{0\to1}_{x_i} = \frac{p^{0/1}_{x_i} p^{1/0}_{x_i}}{p^{0/1}_{x_i} + p^{1/0}_{x_i}} \\ p^{1\to0}_{x_i} &= \frac{p^{0/1}_{x_i} p^{1/0}_{x_i}}{p^{0/1}_{x_i} + p^{1/0}_{x_i}}, \quad p^{1\to1}_{x_i} = \frac{p^{1/0}_{x_i} p^{1/1}_{x_i}}{p^{0/1}_{x_i} + p^{1/0}_{x_i}} \end{aligned} \tag{17-72}$$

3. 假设一个一阶Markov过程，证明对于信号 x_i，条件概率可以用转移概率表示为:

$$\begin{aligned} p^{0/0}_{x_i} &= \frac{p^{0\to0}_{x_i}}{p^{0\to0}_{x_i} + p^{0\to1}_{x_i}}, \quad p^{1/0}_{x_i} = \frac{p^{0\to1}_{x_i}}{p^{0\to0}_{x_i} + p^{0\to1}_{x_i}} \\ p^{0/1}_{x_i} &= \frac{p^{1\to0}_{x_i}}{p^{1\to0}_{x_i} + p^{1\to1}_{x_i}}, \quad p^{1/1}_{x_i} = \frac{p^{1\to1}_{x_i}}{p^{1\to0}_{x_i} + p^{1\to1}_{x_i}} \end{aligned} \tag{17-73}$$

4. 用前面三个习题得出的1比特信号 x_i 的转移和条件概率表达式以及其如下的相关系数

$$\rho_{x_i} = \frac{E[x_i(n)x_i(n-1)] - p^2_{x_i}}{p_{x_i} - p^2_{x_i}} \tag{17-74}$$

来证明那个信号的开关活动率如下表达式所示。

$$\alpha_{x_i} = p_{x_i}(1 - p_{x_i})(1 - \rho_{x_i}) \tag{17-75}$$

5. 考虑具有两个输入 x 和 y 的简单AND门，假设电路由静态CMOS风格实现。在下面假设条件下，计算该门输出的开关活动率，(a) 具有均匀概率分布的非相关输入。(b) 输入之间时间相关：假设 x 当前状态为0则下一个状态必为1，并且假设 y 当前状态为1则下一个状态必为0。(c) 输入之间空间相关：假设只允许的输入组合为01或10。
6. 假设电路由 n 型动态逻辑实现，重复前面习题。
7. 考虑如图17-30所示电路，并假设使用零延迟模型。这里，a、b 和 c 是独立的，并且只在每个时钟周期开始时改变，将 t_x 定义为 $E[x(n)\,x(n-1)]$。
   (a) 令 a、b 和 c 的连续值不相关，令 $p(a)$、$p(b)$ 和 $p(c)$ 都等于0.5。将 $p(f)$ 和 t_f 用 $p(a)$、$p(b)$ 和 $p(c)$ 的形式表示，然后计算 ρ_f 和 α_f。
   (b) 令 b 和 c 不相关，同时令 $p(b)$ 和 $p(c)$ 都等于0.5，令 a 是相关的，即当 a 的前一状态为0则下一个状态必为1，$p(a)=2/3$。将 $p(f)$ 用 $p(a)$、$p(b)$、$p(c)$ 和 t_a 来表示，计算 ρ_f 和 α_f。

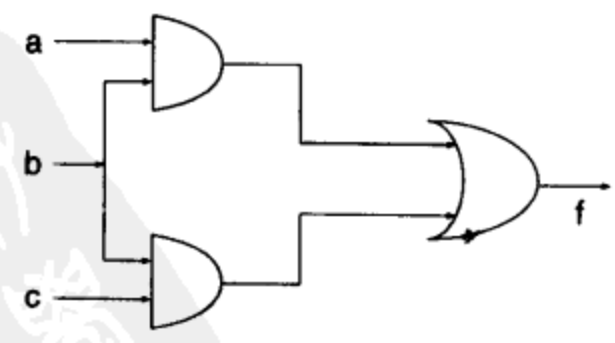
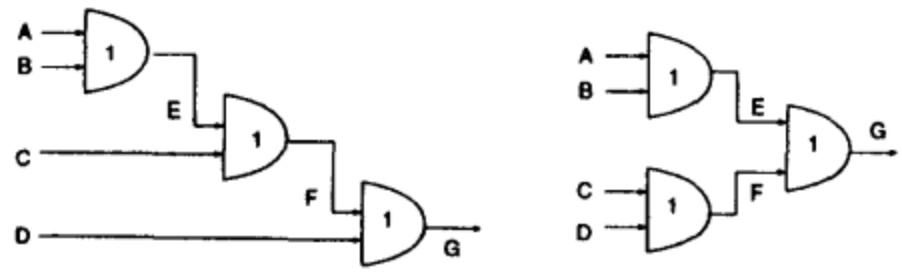


图17-30 习题7的电路

689

(c) 令 c 不相关，$p(c) = 0.5$。令a相关，即当a的前一状态为1则下一个状态必为0，$p(a)=1/3$。令b相关，即当b的前一状态为0，则下一个状态必为1，$p(b)=2/3$。将 $p(f)$ 和 t_f 用 $p(a)$、$p(b)$、$p(c)$、t_a 和 t_b 来表示，然后计算 ρ_f 和 α_f。

8. 应用路径平衡使得图17-10中电路所有路径的延迟相等。
9. 考虑如图17-31所示电路。计算有毛刺和无毛刺时节点E、F和G的开关活动率。所有门具有1个时间段的延迟。设总的开关活动率是节点E、F和G的开关活动率之和。哪个电路具有更小的开关活动率？并加以解释。假设所有输入之间不相关。



a)链结构　　b)树结构

690

图17-31 说明电路结构对开关活动率的影响

10. 重复例17.5.1，假设有10个时间单位的延迟。假设最多3个电源电压，其中1个电源电压为3V，同时，假设最小电源电压为1.33V。求最小功耗解。
11. 画出静态CMOS XOR门的状态转移图。它有多少个状态和边沿？有多少状态不存在？
12. 画出tspc n-latch的状态转移图。与tspc p-latch有何不同？
13. 推导出静态CMOS NOR门边沿活动率数值的方程式。
14. 考虑如图17-32所示的电路。

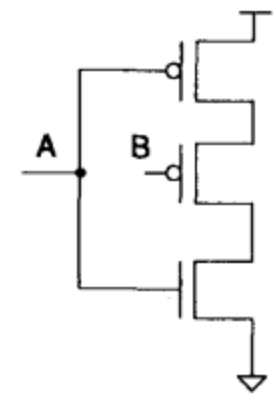


图17-32 习题14的电路

(a) 画出该电路的STD。

(b) 令

$$X = \begin{bmatrix} 4/8 & 1/8 & 1/8 & 2/8 \\ 1/8 & 4/8 & 1/8 & 2/8 \\ 4/8 & 1/8 & 2/8 & 1/8 \\ 1/8 & 2/8 & 4/8 & 1/8 \end{bmatrix} \tag{17-76}$$

代表输入条件概率矩阵。例如，第3行第4列的输入代表概率 $P(AB=11/AB=10)$。用该矩阵计算输入转移概率矩阵，其中每个输入代表 $p(i \to j)$。

(c) 计算状态的稳态概率。用稳态概率和输入条件概率推导出边沿活动率的表达式。假设时钟周期用N表示，每个时钟周期内的时间段数用S表示。

691

(d) 假设每个边沿的能量都相同，为1nJ，$N=1000$，$S=66$，$T_{clk}=20$nsec，计算电路的平均功耗。

参考文献

1. M. Pedram, "Power minimization in IC design: principles and applications," *ACM Trans. on Design Automation of Electronic Systems*, vol. 1, no. 1, pp. 3–56,

Jan. 1996.

2. F. N. Najm, "Transition density: a new measure of activity in digital circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 2, pp. 310–323, Feb. 1993.

3. N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*. MA: Addison-Wesley, 1994.

4. S. Turgis, N. Azemard, and D. Auvergne, "Explicit evaluation of short circuit power dissipation for CMOS logic structures," in *Proc. IEEE International Symposium on Low Power Design*, pp. 129–134, April 1995.

5. A. Papoulis, *Probability, random variables, and stochastic processes*. McGraw-Hill, 2nd ed., 1984.

6. A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid State Circuits*, vol. 27, no. 4, pp. 473–483, April 1992.

7. T. Kobayashi and T. Sakurai, "Self-adjusting threshold-voltage scheme (SATS) for low-voltage high-speed operation," in *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, pp. 271–274, May 1994.

8. J. H. Satyanarayana, *Design of Low-Power DSP Systems*, Ph.D. Thesis, University of Minnesota, 1998.

9. M. Berkelaar and J. Jess, "Gate sizing in MOS digital circuits with linear programming," in *Proc. ACM European Design Automation Conference (EDAC)*, pp. 217–221, 1990.

10. Y. Tamiya, Y. Matsunaga, and M. Fujita, "LP based cell selection with constraints of timing, area, and power consumption," in *Proc. IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 4378–4381, 1994.

11. M. Pedram, "Power estimation and optimization at the logic level," *International Journal of High Speed Electronic Systems*, vol. 5, no. 2, pp. 179–202, June 1994.

12. M. Borah, R. M. Owens, and M. J. Irwin, "Transistor sizing for minimizing power consumption of CMOS circuits under delay constraint," in *Proc. of IEEE International Symposium on Low Power Design*, pp. 167–172, 1995.

13. C.-H. Tan and J. Allen, "Minimization of power in VLSI circuits using transistor sizing, input ordering and statistical power estimation," in *Proc. of IEEE International Workshop on Low-Power Design*, pp. 75–80, 1994.

14. H.-R. Lin and T.-T. Hwang, "Power reduction by gate sizing with path-oriented slack calculation," in *Proc. of 1st Asia-Pacific Design Automation Conference*, pp. 7–12, 1995.

15. C.-Y. Tsui, M. Pedram, and A. M. Despain, "Power efficient technology decomposition and mapping under an extended power consumption model," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 13, no. 9, Sept. 1994.

16. S. C. Prasad and K. Roy, "Circuit optimization for minimization of power consumption under delay constraint," in *Proc. of IEEE International Workshop on Low-Power Design*, pp. 15–20, 1994.

17. W.-Z. Shen, J.-Y. Lin, and F.-W. Wang, "Transistor reordering rules for power reduction in CMOS gates," in *Proc. of 1st Asia-Pacific Design Automation Conference*, pp. 1–5, 1995.

18. C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. 3rd Caltech Conference on VLSI*, pp. 23–36, March 1983.

19. J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 398–402, Nov. 1993.

20. B. Davari, R. H. dennard, and G. G. Shahidi, "CMOS scaling for high performance and low power," *Proc. IEEE*, vol. 83, no. 4, pp. 408–425, April 1995.

21. S.-W. Sun and P. Y. Tsui, "Limitation of CMOS supply-voltage scaling by MOSFET threshold-voltage variation," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 8, pp. 947–949, Aug. 1995.

22. T. Kuroda and T. Sakurai, "Threshold-voltage control schemes through substrate-bias for low-power high-speed CMOS LSI design," *Jour. of VLSI Signal Processing Systems*, vol. 13, no. 2/3, pp. 191–201, Aug./Sep. 1996.

23. S. Mutoh et al., "1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 8, pp. 847–854, Aug. 1995.

24. K. Seta *et al.*, "50% active-power saving without speed degradation using standby power reduction (SPR) circuit," in *ISSCC Dig. Tech. Papers*, pp. 318–319, Feb. 1995.

25. T. Kuroda *et al.*, "A 0.9V 150MHz 10mW $4mm^2$ 2-D discrete cosine transform core processor with variable-threshold voltage scheme," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1770–1779, Nov. 1996.

26. M. Mizuno *et al.*, "Elasti-Vt CMOS circuits for multiple on-chip power control," in *ISSCC Dig. Tech. Papers*, pp. 300–301, Feb. 1996.

27. G. E. Tellez, A. Farrahi, and M. Sarrafzadeh, "Activity-driven clock design for low power circuits," in *Proc. of the IEEE International Conference on Computer Aided Design (ICCAD)*, pp. 62–65, 1995.

28. A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, April 1995.

29. C. H. Van Berkel et al., "A fully-asynchronous low-power error corrector for the digital compact cassette player," in *ISSCC Dig. Tech. Papers*, Feb. 1994.

30. K. Yano et al., "A 3.8-ns CMOS 16×16-b multiplier using complimentary pass transistor logic," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 388–395, April 1990.

31. A. Parameswar, H. Hara, and T. Sakurai, "A high speed, low power, swing restored pass-transistor logic based multiply and accumulate circuit for multimedia applications," in *Proc. of Custom Integrated Circuits Conference (CICC)*, pp. 278–281, May 1994.

32. A. Salz and M. A. Horowitz, "IRSIM: an incremental MOS switch-level simulator," in *Proc. 26th IEEE/ACM Design Automation Conference (DAC)*, pp. 173–178, June 1989.

33. F. N. Najm, "Towards a high-level power estimation capability," in *Proc. IEEE International Symposium on Low Power Design*, pp. 87–92, April 1995.

34. D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures for energy consumption at register transfer level," in *Proc. IEEE International Symposium on Low Power Design*, pp. 81–86, April 1995.

35. N. R. Shanbhag, "A mathematical basis for power-reduction in digital VLSI systems", *IEEE Trans. on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 44, no. 11, pp. 935–951, Nov 1997.

36. P. E. Landman and J. Rabaey, "Activity-sensitive architectural power analysis for control path," in *Proc. IEEE International Symposium on Low Power Design*, pp. 93–98, April 1995.

37. S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "Analytical estimation of signal transition activity from word-level statistics", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 718–733, July 1997.

38. K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinatorial networks," *IEEE Trans. on Computers*, vol. C-24, pp. 668–670, June 1975.

39. R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, Aug. 1986.

40. A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in *Proc. 29th IEEE/ACM Design Automation Conference (DAC)*, pp. 253–259, June 1992.

41. R. Burch, F. Najm, P. Yang, and D. Hocevar, "Pattern independent current estimation for reliability analysis of CMOS circuits," in *Proc. 25th IEEE/ACM Design Automation Conference (DAC)*, pp. 294–299, June 1988.

42. J. H. Satyanarayana and K. K. Parhi, "HEAT: Hierarchical Energy Analysis Tool," in *Proc. IEEE/ACM Design Automation Conference (DAC)*, (Las Vegas, NV), pp. 9–14, June 1996.

43. S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate signal processing," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, vol. 15, no. 1/2, Jan. 1997.

44. M. Goel, and N. R. Shanbhag, "Low-power reconfigurable signal processing via dynamic algorithm transformations (DAT)", in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Seattle, WA), May 1998.

北京大学图书馆藏

第18章 可编程数字信号处理器

18.1 引言

DSP有着广泛应用，包括数据/传真调制解调器、数字语音广播、数字照片、数字视频、多媒体计算机、数字蜂窝电话、个人通信系统等。对于高性能的DSP应用，通常采用特殊的（专用的）结构。这个趋势一直持续到现在，因为更多复杂的数字信号处理和图像/视频处理的算法都采用单芯片实现。而且，与此同时，随着按比例缩小VLSI工艺的改进，已经使得可编程的微处理器有了好几个数量级的改进。

数字化表示的语音、音频和视频信号（多媒体）可以用类似于文本数据处理的方式来处理。但是，多媒体信号的信息速率要比文本信号数据的速率高几个数量级。结果，传统的微处理器不能实时处理这些多媒体数据。这直接导致了可编程数字信号处理器（通常写为PDSP或DSP处理器）的进化，它是一个特殊的微处理器，它通过可下载或常驻软件实现高强度算术计算和DSP功能。PDSP在1980年首次开发成功并被用于通信应用中，诸如语音频段数据调制解调器和低位速率语音编解码的实时语音信号处理。该PDSP已进化了几代。它们区别于通用目的微处理器的地方在于：低功耗、低成本和高的信号处理能力；它们优于专用结构DSP的地方是可以提供高度的灵活性、低的设计成本，缩短的产品上市时间。当前的DSP处理器特别适合于无线应用和多媒体处理。

本章阐述了PDSP结构的基本特点和它们的进化，用于适应日益增长的无线通信和多媒体处理的应用需要。18.2节描述了PDSP在过去20年中的进化过程。18.3节描述了DSP处理器结构的两个重要特点，包括快速乘法累加单元的数据路径和多访问存储器架构。18.4节和18.5节分别讨论了PDSP在无线通信领域（特定域DSP）和多媒体处理领域（媒体处理器）的设计趋势。在18.5节也探讨了通用目的微处理器中的多媒体辅助部件。

18.2 可编程数字信号处理器的进化

通用目的DSP早在1980年就开发成功，用于迭代和计算强度很大的数字信号处理。这些应用的运行中大都受到严格的实时信号处理约束，涉及到大量存储单元的访问和重复执行同样的算法。早期的DSP处理器起源于引入用于快速乘法累加运算的硬件乘法器和多访问存储器的通用目的微处理器，以满足性能需求和能够实现可编程性。PDSP不能提供所有通用目的微处理器可以提供的友好编程器，因此，这些DSP处理器可以获得很高的性能，同时成本很低，功耗也很小。第一代DSP的开发是为了高吞吐量（百万次运算每秒）的音频信号处理。它们的特性与今天的M68000微处理器之间的比较列入表18-1[2]。从表中可以看出，在大致同样成本和功耗的前提下，DSP-1与M68000之间的性能比为50：1[2]。

表18-1 第一代DSP处理器和Motorola 68000微处理器之间性能特性的比较

	DSP-1 (1980)	M68000(1980)
工艺	5μ NMOS	3μ NMOS
时钟速率	5MHz	8MHz
总线宽度	20位数据,16位指令	16位数据,16位指令
峰值性能	每秒5兆次乘法累加	每秒0.1兆次乘法累加
功耗	1.25瓦	1.5瓦

在过去的20年中，DSP处理器的性能已经获得很大的提高，价格和功耗也有很大降低。通常，DSP处理器主要采用并行处理技术，一条指令包涵几个并行的操作。结果，DSP处理器的百万次运行每秒（MOPS）指数比微处理器要高很多。但是，这个结论不再正确，因为最新一代的单指令多数据（SIMD)增强的微处理器（例如Intel的MMX）采用子字并行处理，使得微处理器的性能比传统微处理器的性能高很多。但是，由于DSP独特的低功耗和低成本特色，使得DSP在目前的便携式和无线市场中仍然占主导地位。

近来，随着有效的音频和视频压缩技术以及其标准的出现，产生了一个新型的设备：媒体处理器。这些处理器是一系列特殊DSP处理器的集合，包括了音频、视频和图像加速器，适合于多媒体信号处理的应用。这些媒体处理器利用指令级别（例如，用超长指令字VLIW）和数据级别(例如，SIMD)的并行处理技术在多媒体信号处理算法中的潜力，使得这些媒体处理器在时钟信号为100MHz的情况下，可以获得数十亿次运算每秒的峰值性能。这些微处理器即可以单独用作一个CPU（中央处理单元），例如，可用在可视电话中，也可以用在个人计算机（PC）中的多媒体加速器上。

18.3 DSP处理器的重要特性

设计DSP处理器的目的是用于支持重复的、数值强度很大的任务[3]。到目前为止，大多数的DSP处理器不仅具有很强的数据路径，而且有能力在存储器中快速地进行大量的数据传输。进一步，DSP处理器提供特殊的指令集合，用于充分利用硬件的效率。

在本节中将阐述DSP处理器结构的两个最重要特性，就是包涵快速乘法累加单元的数据路径和多访问存储架构[4]。一个N抽头的有限冲击响应（FIR）滤波器可以表示为：

$$y[n] = c_0x[n] + c_1x[n-1] + \cdots + c_{N-1}x[n-N+1] \tag{18-1}$$

以此作为在本节中一个典型的例子。

696 ~ 697

流水线（参看第3章）技术常常可以用来提高处理器的性能。今天，几乎市场上所有的处理器都在某种程度上使用了流水线技术。但是，在性能改进的过程中，流水线使得实现可编程有一定的难度。在本节中，探讨了把两个普通的可编程技术用于可编程的流水线PDSP中，这两个技术是：时间固定和数据固定编码技术[5]。

18.3.1 数据路径

在本节中，仅仅考虑定点的DSP处理器。它们通常包括一个乘法器、累加器、算术逻辑单元(ALU)、一个或多个移位器、操作数寄存器和一些其他的特殊单元，例如饱和算术单元。

18.3.1.1 乘法器和累加器

事实上，在所有的DSP算法中，乘法是一个必需的运算。在许多应用中，处理器所执行的一半和更多的指令涉及到乘法运算。一个硬件乘法器的存在对定义一个DSP处理器是很关键的。

在许多DSP算法中，乘法做完以后紧接着就是加法。于是，在许多DSP处理器中，乘法器和加法器集成在一起，使得乘法累加（MAC）运算可以在一个指令周期中完成。

18.3.1.2 算术逻辑单元（ALU）

DSP处理器的算术逻辑单元可以在一个时钟周期内实现基本的算术和逻辑运算。通用的运算包括加/减、增1、求反、逻辑与、或、非以及异或。在有些处理器中，ALU被用于执行乘法累加（MAC）中加法的运算，还有的处理器提供单独的加法器。

18.3.1.3 移位器

乘法和加法运算将导致结果的位宽增加，因此，在许多定点DSP应用中，缩放成为一个重要的运算。在数据路径中的移位器可以对输入到移位器中的数据进行幂2的缩放运算。移位器通常紧跟在乘法器和ALU的后面。有些处理器在乘法器和ALU之间加入了移位器，用于有效地对乘积进行缩放。移位能力受限的移位器（每次只能移动一位）和桶型移位器（在一个指令周期中可以移动任意位）都被用于处理器中。

18.3.1.4 溢出和饱和

大多数的DSP算法涉及到对一个级数进行累加。在这种情况下，和值就会增大。最终，和值可能超出累加器中寄存器可表示的最大或最小值范围，这种情况被称为溢出。当溢出发生时，一个不正确的值被保存，而且该值可能与实际的正确值相差很大。溢出可以用缩放结果值的运算或饱和算术运算来处理。缩放是有效的，但是，它不能保持足够的信号保真度。在DSP处理器中，溢出通常是用饱和算术运算来处理的。在饱和算术运算中，一个特殊电路检测溢出的发生，当发生正方向或反方向溢出时，相应地选择最大可能的数值或最小可能的数值作为输出。定点DSP处理器通常提供特殊的硬件实现饱和算术运算。 698

需要指出的是，通常不用DSP处理器中的数据路径进行存储地址的寻址计算，而是一个分离的地址产生单元。

18.3.2 存储器架构

对于一个高性能的DSP处理器而言，存储器架构，也即存储器的组织结构和存储器与处理器的数据路径之间的互联，与数据路径是同等重要的。

考虑公式（18-1）的N抽头FIR滤波器，可知，一个滤波运算涉及到N个乘法累加运算。用处理器数据路径中的乘法累加单元，可以在每N个指令周期之后产生一个新的输出采样。但是，要获得这样的性能，处理器必须能够在一个指令周期之内多次访问存储器。

通常，一个处理器包含一个单一的存储器单元库，通过一个惟一的地址和数据总线访问，如图18-1所示。这通常被称为是冯·诺伊曼（Von Neumann）结构。程序的指令和数据都存储在单一的存储器中，处理器在一个指令周期中只能访问一次存储器。这种类型的存储器结构不适合DSP处理器的应用。考虑到FIR滤波器的例子，尽管处理器的数据路径完全可以在一个指令周期中完成一个MAC运算，但是对于一个采用时序存储器访问和一个指令周期只

能访问一次存储器的处理器而言，完成一个MAC计算需要的时钟周期多于3个。

DSP处理器利用许多种变化的冯·诺伊曼结构来改进存储器访问的带宽，这些处理器就是所谓的哈佛(Harvard)结构。在这种结构中，处理器通过两个独立的数据总线连接到两个独立的存储单元库中，如图18-2a所示。在这种结构中，一个存储器保存程序指令，一个存储器保存数据。它也可以扩展为用一个存储器既可以保存程序指令又可以保存数据，而使另一个存储器仅仅保存数据。不过，在一个指令周期中，至少可以访问存储单元2次。对于DSP处理器而言，用3个存储单元库也是常见的，其中一个存储单元库用于保存程序指令，另外两个用于保存数据。在这种情况下，处理器在一个指令周期中完成1抽头的FIR滤波运算就没有问题。除了使用多个独立的存储单元库以外，一定数量的DSP处理器使用快速的存储单元，使得在一个指令周期内，这些快速的存储单元可以经由总线时序地对存储器访问多次；还有的DSP处理器使用多端口的存储器，这种结构允许处理器在一个指令周期中，经过2个或更多个独立的总线集合多次并发地访问存储器库。最常用的多端口存储器是双端口的，它提供两个并发访问。如果按照芯片的面积来计算，使用多端口存储器的缺点是实现的成本高。有些DSP处理使用的存储器结构是多端口存储器结构和改进的哈佛存储器结构的组合，如图18-2b所示。该存储器结构包括了一个单端口的程序指令存储器和一个2端口的数据存储器；在一个指令周期中，可以实现1次程序指令访问和2次数据访问。

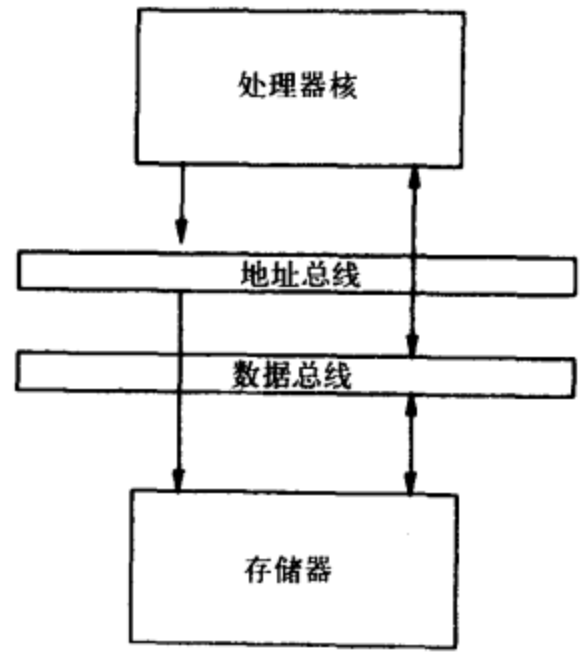
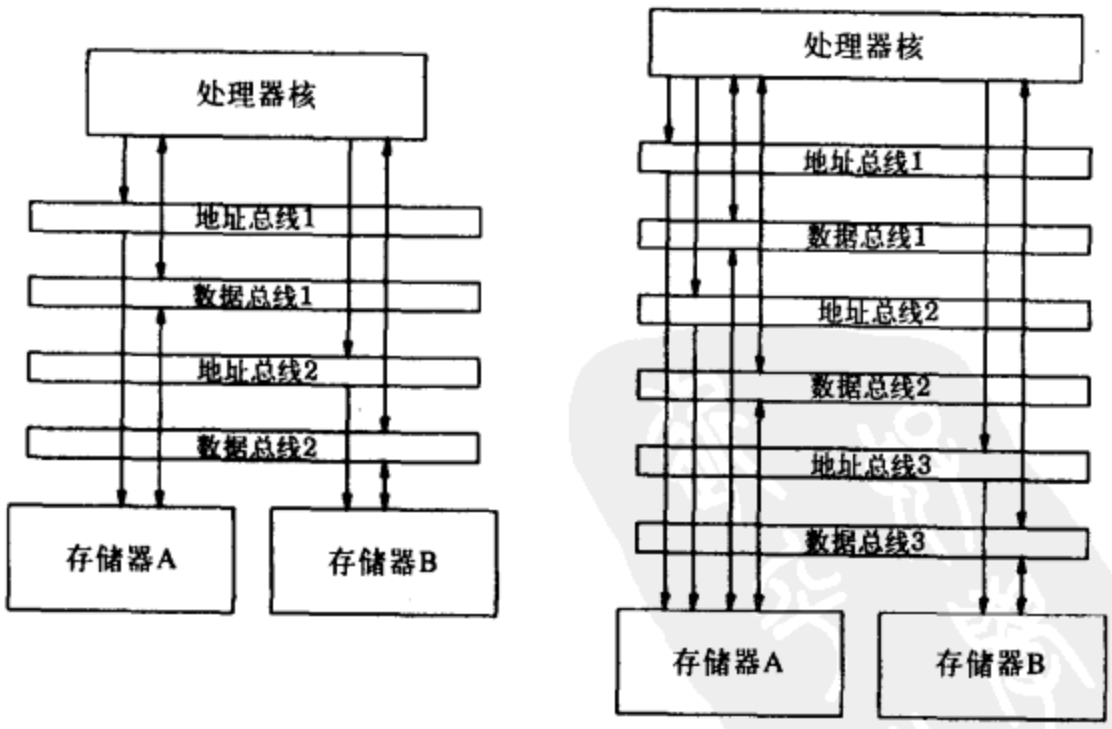


图18-1 简单的Von Neumann存储器结构，用于许多非DSP的处理器中



a) 基本哈佛结构，该处理器可用两组独立的总线同时访问两组存储器

b) 有一个双端口数据存储单元(A)和一个单端口数据存储器(B)的哈佛结构，该处理器可以通过三组总线同时执行两次存储器A的访问和一次存储器B的访问

图 18-2

699 ~ 700

并行存储器库可以提高存储器带宽，但是却产生了一个基本问题，因为指令一定要指定几个存储器的访问。对于一个合理的地址空间，指定每个地址所需要的位数大了很多，以至于指令字的宽度也变大了。因此，需要一个更大的空间存储程序，或者说访问一条指令需要更多的存取周期来实现（在一个多字指令的情况中）。到目前为止，使用间接寄存器寻址模式(向前或向后进行递增或递减)。为了实现这个目的，通常需要一系列的地址寄存器。上一小节已经提到，大多数DSP处理器包括一个或多个特殊的地址产生单元（AGU），专门用于地址计算。不使用处理器的数据路径，一个AGU在一个指令周期中可以执行一个或多个复杂的地址计算。这使得地址计算和数据的算术运算可以并行处理。

有些DSP处理器也提供特殊设计用于减小执行某种运算时对存储器的访问量。这些技术包括使用存储程序的高速缓冲存储器、以地址模式求模等。存储器的访问量也可以通过用算法（程序）来减小，也就是尽可能地利用数据的局域性。

用多功能单元和多访问存储器库，DSP处理器可以在一个指令周期内发出特殊的指令，指定不同的功能单元执行几个并行的操作，例如并行操作数的获取和执行操作。构建这些特殊的指令集可以尽可能地利用硬件资源。而且，把几个操作包装在一个特殊的指令中，也可以减小程序的大小。

18.3.3 流水线作业DSP处理器中的时间固定编码和数据固定编码

流水线作业是一种实现技术，它通过把一个时序的操作划分为更小的操作，同时使得这些小的操作并行执行。在处理器中，流水线作业把指令的执行划分为几个流水作业的进程，允许多条指令在执行中相互重叠。流水线作业就是尽可能地并行执行一个时序指令流中的指令，同时减小每条指令的平均执行时间。

流水作业加速了计算，但使得编程更加复杂[6]。通常，两个集合编码格式——时间固定编码和数据固定编码——可用于流水线作业的DSP处理器的编程。

18.3.3.1 时间固定编码

在时间固定编码中，程序员可以更加清晰地控制流水线的进程。一条指令清晰地指定几个操作，例如同时取操作数和执行操作，这些操作可以用不同的功能单元并行地执行。例如，下面的指令指定2个存储器取数操作和1个乘法累加（MAC）操作同时进行。

701

```
MAC X0, Y0, A   X:(R0)+, X0  Y:(R4)-, Y0
```

在这个情况下，乘法器和加法器被集成在一起，以至于MAC运算可以在一个周期中算完。MAC运算的操作数是X0和Y0寄存器中的内容，这两个寄存器中的内容是在前一个指令周期中从存储器中读出来的；该MAC运算计算X0和Y0的乘积，然后和A寄存器中的数相加，把和值存储在A寄存器中。同时，另外两个部分是给下一条指令取操作数。

另外一个时间固定编码的乘法累加指令的好例子如下：

```
a0=a0+p   p=x*y   x=*r0++   y=*r1++
```

来自寄存器存储的前一指令运算获得的乘积 p 和 $a0$ 相加，然后存放在 $a0$ 中；同时，x 和 y 寄存器中的值进行相乘形成了一个新的乘积；与此同时，x 和 y 寄存器又用地址寄存器r0和r1所指定的地址从存储器中加载了新的值；r0和r1寄存器中的值递增，指向它们相应的下一个存储器的位置。值得注意的是，在这个例子中，处理器中的乘法器和加法器是相互独立的，一个MAC运算需要两个指令周期。

18.3.3.2 数据固定编码

在数据固定编码中，一条指令指定了对来自存储器的一系列操作数的所有操作，但是没有确切地指定这些操作在什么时间进行。考虑下面的例子:

```
a1=a1+ (*r5++ = *r4++) * *r3++
```

上述表达式指定了如下操作：从存储器中读出以地址寄存器r3和r4中存储的数值为地址的数据，并且进行相乘；以地址寄存器r4中数值为地址，相应的存储器中的数据被写入到以地址寄存器r5中数值为地址的存储器中；乘积的结果在寄存器a1中累加。值得指出的是，数据固定方法中所用的操作数是直接指向存储器的。相乘指令进行操作的数直接是两个存储器中的数值，而不是两个寄存器中的数值进行相乘。存储器中的数值被取出，然后送到乘法器中，但是，与时间固定方法不同的是该方法中程序员不必直接指定这些。

时间固定和数据固定方法各有自己的优点和缺点。在时间固定编码中，程序员要执行部分调度任务，然而在数据固定编码方法中，程序员指定需要进行的操作，把调度的任务留给了编译器。通常，数据固定模型更容易读懂，但不像时间固定方法那样灵活。

702

18.4 移动通信与无线通信用的DSP处理器

在适当的时钟频率下，利用特殊的数据路径进行并行处理，DSP处理器可以获得很高的吞吐量。如表18-2总结的那样，用变化的基本哈佛结构，DSP已经历了许多代[7]。不同代之间的差别主要表现在总线结构和存储器带宽上，以至于乘法累加运算可以在一个周期或更短的时间内完成。

尽管VLIW和SIMD方法（详细内容参见下节阐述）可以用于进一步改进处理器的性能，但是从面积和功耗的角度考虑，这两种方法的代价是昂贵的，使得这两种方法不适用于移动和无线通信，因为在移动和无线通信领域，小的面积和低功耗是关键的指标。于是，特定域的概念引入到处理器的特定目标应用中。表18-2中列出的Lode DSP[7]是DSP的第5代，是一个适用于无线通信和语音信号处理的特定域处理器。它的数据路径包括2个MAC单元和1个算术操作单元（AMU），而且是可以并行操作的。通常，这样的方法可以使得存储器的带宽提高3倍。在这个特定域的DSP处理器中，连接到这3个功能单元的输入信号的组合受到限制。输入信号的组合和总线结构已经被选定，用以最优化地支持典型的DSP运算，特别是无线通信和语音信号处理用到的操作。指令集也反映出对一个特定域DSP的选择。例如，像用于矢量量化的平方距离和累加指令，用于Viterbi算法中的加-比较-选择指令，用于前向纠错码的Galois域运算指令，都在指令集中给出。

表18-2 DSP处理器的进化

年 份	发展阶段	特 色	例 子
1982	第一代	基本哈佛结构，一个数据总线，一个程序总线，一个乘法累加单元	TMS32010, NEC7720
1986	第二代	修改的哈佛结构，一个数据/程序总线，一个数据总线	TMS320C25, AT&T DSP16A
1990	第三代	附加的地址模式，附加功能	TMS320C5x, AT&T DSP161x
1994	第四代	两个数据总线，一个数据总线，分离的MAC,ALU	TMS320C540
1995	第五代	两个数据总线，一个数据总线，两个MAC,一个ALU	Lode

703

18.5 多媒体信号处理用的处理器

多媒体信号处理涉及到对不同表现形式的数字信息进行处理。它覆盖一个很宽范围的应用，包括音频和语音处理、图像和视频处理、二维和三维图像等。多媒体信号处理的应用中通常有很高的数据速率，同时需要实时处理能力。从算法角度来看，多媒体信号处理的重要特点可以归纳如下[8]:

- 经常使用小的整数操作数
- 高度规则化的、高强度的计算操作
- 大量的输入/输出或存储器访问、数据再利用和数据的局域性
- 在较小计算强度任务中，采用复杂的控制运算

传统的微处理器和DSP处理器不支持多媒体信号处理中的这些特点。因此，采用特殊结构来获得所需要的高信号处理能力，并且有效地利用硬件资源。到目前为止，已推出一种特殊种类的媒体处理器，用于高质量地实时处理多媒体应用。它们可以单独地作为一个CPU或作为PC中的多媒体加速器。同时，通用目的微处理器通过使用多媒体辅助部件提供的新指令和资源，改进了其多媒体处理程序的性能。

在本节中，引介了几种通用的用于加速多媒体信号处理的方法。然后，讨论了用于媒体处理器和通用目的微处理器中多媒体辅助部件的特殊策略。

18.5.1 增强对多媒体信号处理的能力

两个通用的增强技术都适用于多媒体应用的可编程DSP。一种方法是利用指令或数据级的并行处理，以便在计算能力上获得巨大的提高；另外一种方法是引入特殊的指令和集成专有的硬件。在这个很宽的框架里，本节中对许多种可能的方法进行了概述。

18.5.1.1 多重处理

多重处理通常采用的形式是SIMD或多指令多数据 (MIMD)[6]。在一个SIMD机器里，同样的指令被多个处理器在不同的数据流中执行。每个处理器有自己的数据存储器，但是，仅仅有一个指令存储器和控制处理器用于取指令和分配指令。一个简化的SIMD结构如图18-3a所示。在一个MIMD机器中，每一个处理器取自己的指令，对自己的数据进行操作。这些处理器要么利用集中的共享存储器结构，要么每个处理器有自己的存储器，同时使它们之间通过交叉的网络进行通信。一个简化的MIMD结构如图18-3b所示。SIMD处理器可以使数据并行处理，但是没有MIMD处理器灵活。SIMD适合于那些数据处理并行性高而控制流与数据相关性小的算法。MIMD处理器更灵活；它们要么作为一个单用户的机器，在某一特殊应用中可以集中获得高性能，要么作为多线程机器同时执行许多任务。但是，由于控制硬件重复、需求高的指令存储器带宽和数据路径的同步等原因，MIMD更加复杂和昂贵。除了单纯的SIMD和MIMD方法外，SIMD和MIMD方法的组合也是可能的，这些混合结构的目标是尽可能地利用SIMD和MIMD两种结构的优点。

704

18.5.1.2 子字并行处理：数据级并行处理

在图像处理、视频处理和图形着色计算中的数据通常用小于16比特。例如，对于普通图像处理中的像素而言，8比特的精度是足够的，12比特的数据可以用于医学影像。但是，在一

个通用的处理器中，处理器单元的长度，也即字长，通常是32比特或64比特。在多媒体程序中的这些低精度数据无法尽可能地利用处理器的计算能力。到此为止，子字并行处理被引入。

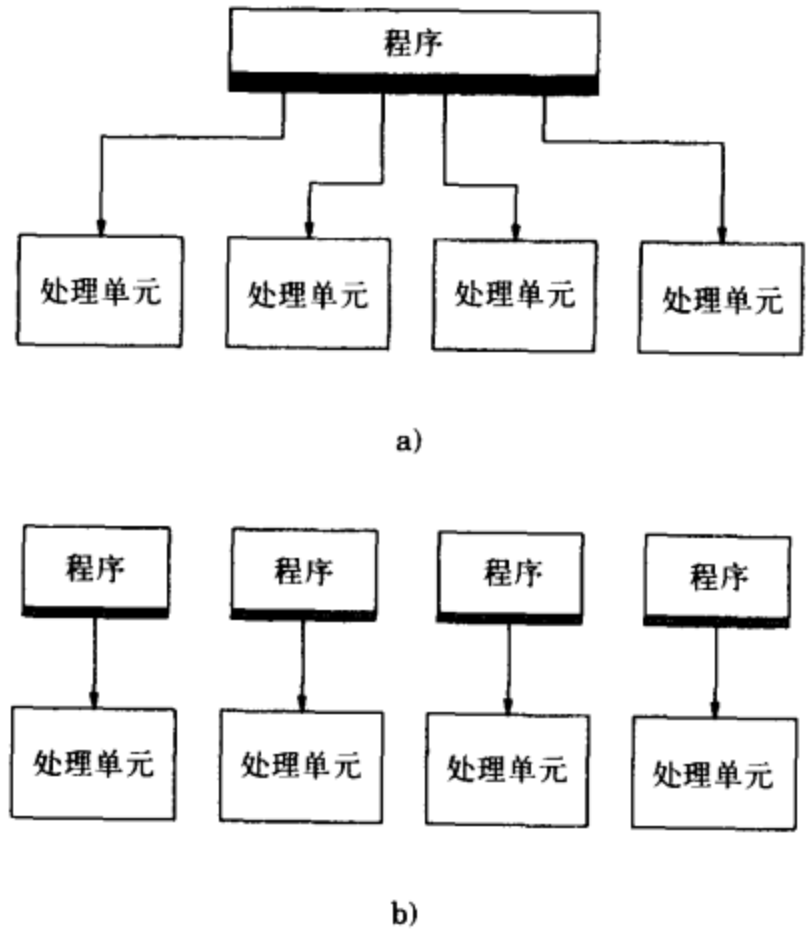


图18-3 a) 简化的SIMD结构 b) MIMD结构

在子字并行处理中，一个标准计算或存储单元，也即一个字，被划分为更小的单元，称为子字。例如，一个64比特的字被划分为4个16比特的子字。可以对这些子字并行地同时执行同样的操作，这种格式适合SIMD的并行处理。子字可以有不同的尺寸，8比特，16比特或32比特，这取决于目标应用的范围。实际上，子字之间是相互不重叠的，是完全填满一个字的。

不同于通常的SIMD方法使用多个处理器，在子字并行处理中，一个字长的功能单元被划分为更小子字长的单元。例如，一个64比特的整数二进制补码加法器可以通过在3个16比特子字相加的边界处有条件地阻塞进位，最终划分为4个16比特的子字整数加法器。这样分裂的加法器可以在一个指令周期中完成4个16比特的加法或1个64比特的加法。通常使用可以分割的ALU。图18-4阐明了在一个有2个可分割ALU的处理器中子字并行处理的概念。注意，包装的字也提高了有效存储器的带宽，因为4个子字可以作为一个字在存储器中来回访问。

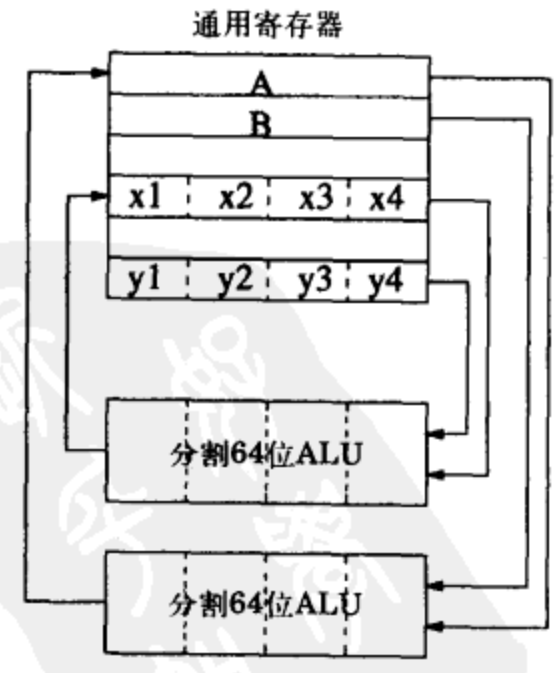


图18-4 子字并行处理的例子，这里同一个ALU可以执行4个16比特子字的并行运算，或者执行1个64比特字的运算

用子字并行处理和它们相应的特殊指令集，许多计算强度很大的任务（环路）可以有效地被编码和执行。例如，考虑下面的矢量矩阵乘法，所有的矩阵元素都是16比特的字长。

705 ~ 706

$$\begin{bmatrix} A0 \\ A1 \end{bmatrix} = \begin{bmatrix} M00 & M01 & M02 & M03 \\ M10 & M11 & M12 & M13 \end{bmatrix} \times \begin{bmatrix} V0 \\ V1 \\ V2 \\ V3 \end{bmatrix}$$

在一个具有64比特字长和具有可分割功能单元的处理器中，这个矢量矩阵的乘法可以如图18-5所示那样进行计算。

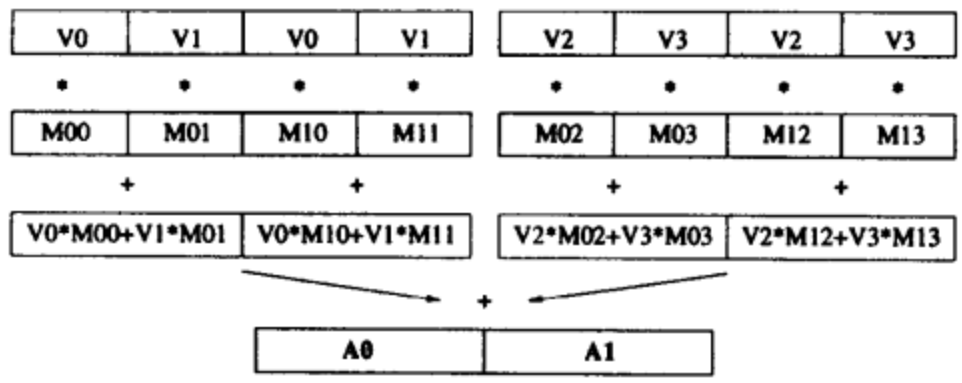


图18-5 作为一个子字并行处理的例子的矢量矩阵乘法的流图

在一个面向字的低成本处理器中，子字并行处理提供了一个小规模的SIMD并行处理[9]。可分割的功能单元有很小的硬件过剩，包装的数据类型减小了存储器访问的需求。这个方法不仅对特殊目的的多媒体处理是适用的，而且对于那些低数据精度和高数据并行的应用而言也是适用的。

18.5.1.3 LIW(长指令字)/VLIW（超长指令字）：指令级的并行处理

（V）LIW结构属于多发射处理器的家族[6]。多发射处理器的目的是允许多条指令在一个时钟周期中被发射，以至于每条指令的周期数（CPI）可以减小到小于1。（V）LIW结构是通过在一个指令字中指定多个LIW或VLIW，以获得指令级的并行处理。这需要多个功能单元，以便可以同时执行一个指令字中包含的所有操作。（V）LIW处理器可以看作是具有集中的存储器结构和多个功能单元的小规模MIMD处理器。VLIW结构用一个划分成固定长度域的更宽指令，这些固定长度域中指定的全部是简单的、单一的和完全独立的操作。图18-6给出了一个典型的VLIW结构。

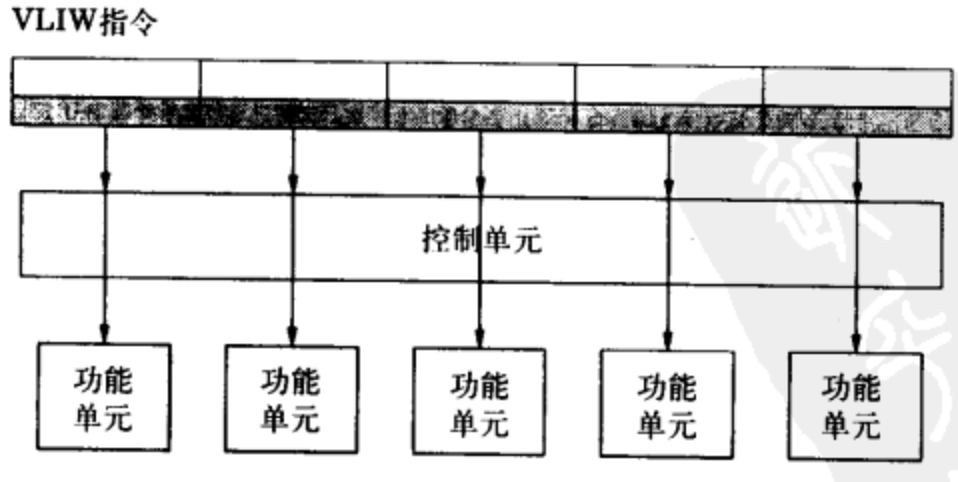


图18-6 一个典型的VLIW结构

VLIW操作依靠编译器执行静态调度，与超标量(superscalar)架构（另一类型的多发射处理器）不同之处在于超标量架构是动态地决定指令的并行执行。这导致了VLIW处理器有简单的结构，因为调度任务是在软件级进行的。另一方面，需要一个很成熟的编译器，以确

保可以有效地利用硬件资源。因此，VLIW结构把复杂度从硬件移到了软件。VLIW结构的有效性与指令级并行利用的程度密切相关，而指令级的并行程度本质上取决于一个特殊的算法。典型的视频压缩算法具有短的内循环，循环展开可用于提高编译器执行静态调度的效率。

除了上述的并行处理技术以外，特殊指令集和协处理器也常常用于多媒体信号处理中。

18.5.1.4 特殊指令集

对于经常重复出现的、有相对较高复杂性的操作，例如具有饱和处理能力的乘法累加运算，传统上和广泛使用的方法是引入特殊的硬件和指令集。特定指令可以显著地减小指令数，以使程序更快地执行。这种在过去20年中被DSP处理器所采用的方法最近用于某些通用目的微处理器中，加速多媒体信号处理程序。

18.5.1.5 协处理器

协处理器结构合并了并行能力和适应能力。一个典型的协处理器结构包含一个处理器核（通常是精简指令集计算机（RSIC）类型）和几个适用于特殊任务的模块。一种可能是把一个标准RISC核和一些特殊模块合并，标准RISC核用于执行不规则的、与数据相关的算法，而特殊模块用于规则的、计算强度很大的程序部分。另外一种方法是给一个另外的计算能力很强的通用目的处理器核中加入高度特定化的模块，用于执行一个特殊的子任务。协处理器对于特殊的算法可以获得更高的效率，因为特定功能的优化是可能的，控制电路也可能是很简单的。

707 ~ 708

18.5.2 媒体处理器

当前的媒体处理器不仅支持类SIMD的子字并行处理，以利用数据的并行；而且也采用VLIW结构用于指令的并行。在大约100MHz时钟的情况下，这些媒体处理器可以获得超过每秒10亿次（GOP/sec）的运算能力。几种类型的媒体处理器的性能特点被列在表18-3中。

表18-3 发表的媒体DSP处理器

处理器	TI C8x	Chromatic Mpact	Philips Tri-Media	TI C6x
结构	4×64位DSP 加31位RISC 加交叉连接	VLIW/SIMD 4个ALU 加多媒体辅助部件引擎 加792位数据总线	VLIW 加25个执行单元 加视频/音频 加VLD单元	VLIW 8条指令/时钟周期 2次MAC计算/时钟周期
工艺	–	0.5μ	–	0.25μ
晶体管数	4M	1.5M	–	–
芯片尺寸(mm²)	340	100	–	196
时钟(MHz)	40	62	100	200
峰值速率	1.2GOPs/sec	2GOPs/sec	4GOPs/sec	1.6GOPs/sec
功耗(瓦)	8	–	–	–
存储器	DRAM 400MB/s	RAMBUS 500MB/s	SDRAM 400MB/s	SDRAM 400MB/s
编程	编译器加汇编程序	内部	VLIW 编译器	VLIW编译器

18.5.2.1 德州仪器C8x

TI TMS320C8x[10]是一个完全可编程的MIMD处理器，把RISC浮点、多个高性能的定点DSP、在片存储器控制器、静态RAM技术集成于一个单芯片上。图18-7给出了TI TMS320C8x的处理器结构。32比特的RISC主处理器用于执行高级语言的处理；它控制在片的

并行处理，支持和别的设备之间的通信（例如一个PC主机，如果C8x用于一个基于PC的环境中），执行大多数高级处理，例如控制协议和浮点计算强度很大的算法。四个先进的DSP处理器包含有效的乘法累加处理，用可分割的ALU与位流操作结合起来增强对图像信号的处理能力。它们用于计算强度很大的整数处理，也可以有效地支持很广范围内的DSP算法，包括频域变换（例如离散余弦变换）、相关、滤波和位流操作。这些处理器执行它们自己的指令流，并且通过共享的存储器结构进行互相通信。利用MIMD结构、类SIMD的子字指令和40MHz的时钟频率，TI C8x处理器可以获得的峰值性能为1.3GOP/sec。TI TMS320C8x数字信号处理器支持桌面视频会议、视频电话、3D虚拟现实、实时MPEG-1解压缩、高质量语音处理和文本图像处理。 709

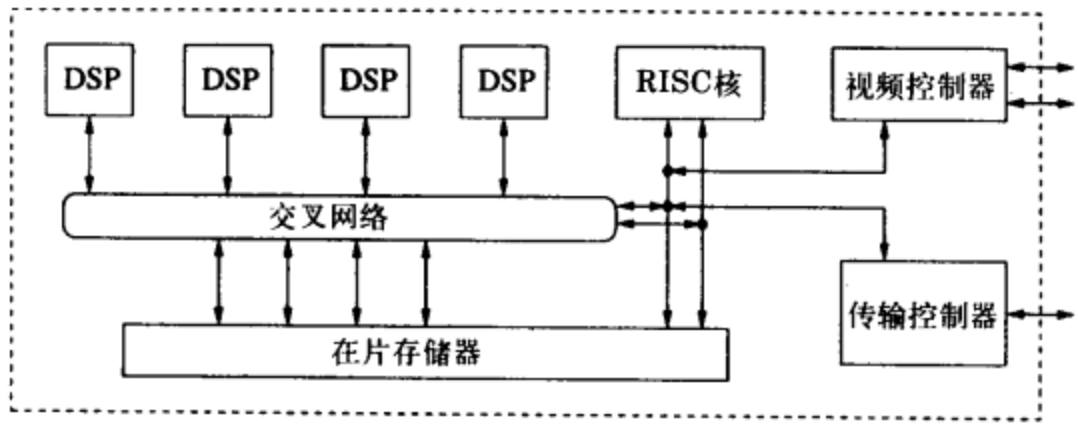


图18-7 TI TMS320C8x 媒体处理器结构

18.5.2.2 Chromatic Mpact

Mpact媒体处理器[11]，[12]被设计用于加速PC的多媒体功能，它的运行需要一个主处理器。它包涵处理机控制器、数据路径、静态RAM（SRAM），如图18-8所示。图18-9给出了处理器的数据路径，包含五个可以并发运行的ALU单元。Mpact指令集用VLIW风格控制数据路径，同时也支持包装数据类型（子字指令）。为了支持VLIW指令集结构（ISA）对高数据带宽的需求，SRAM有4个读端口、4个写端口。在62MHz时钟下，MPact处理器可获得的峰值性能接近2GOPs/sec。Mpact支持实时的MPEG-1的编码、MPEG-1与MPEG-2的解码、二维和三维图形、高质量语音处理、传真/调制解调器、视频会议。

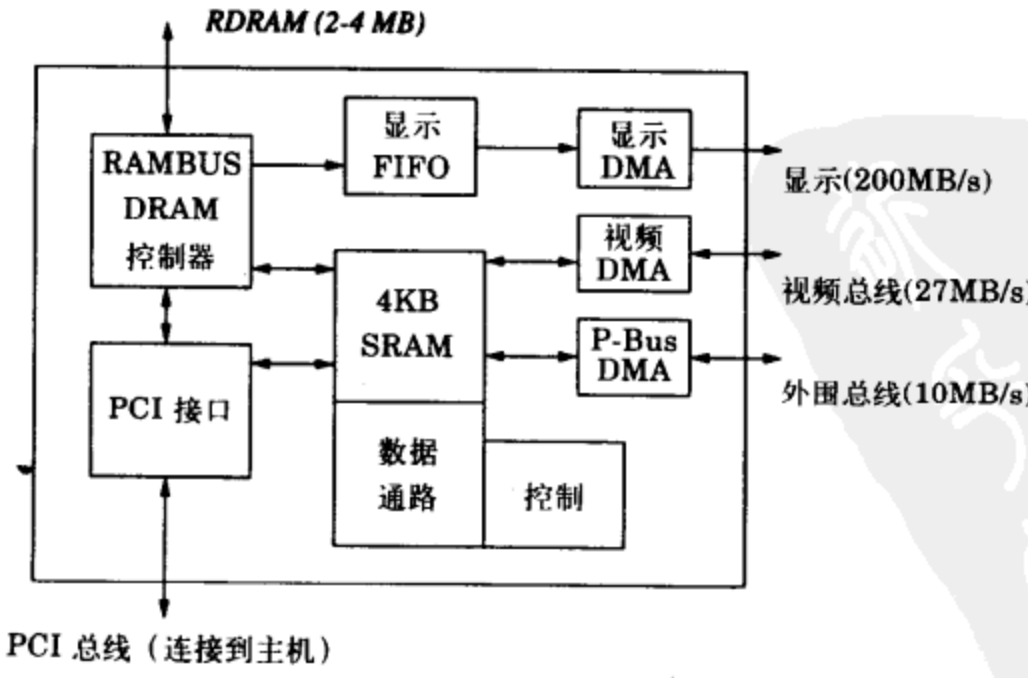


图18-8 Chromatic Mpact 媒体处理器结构

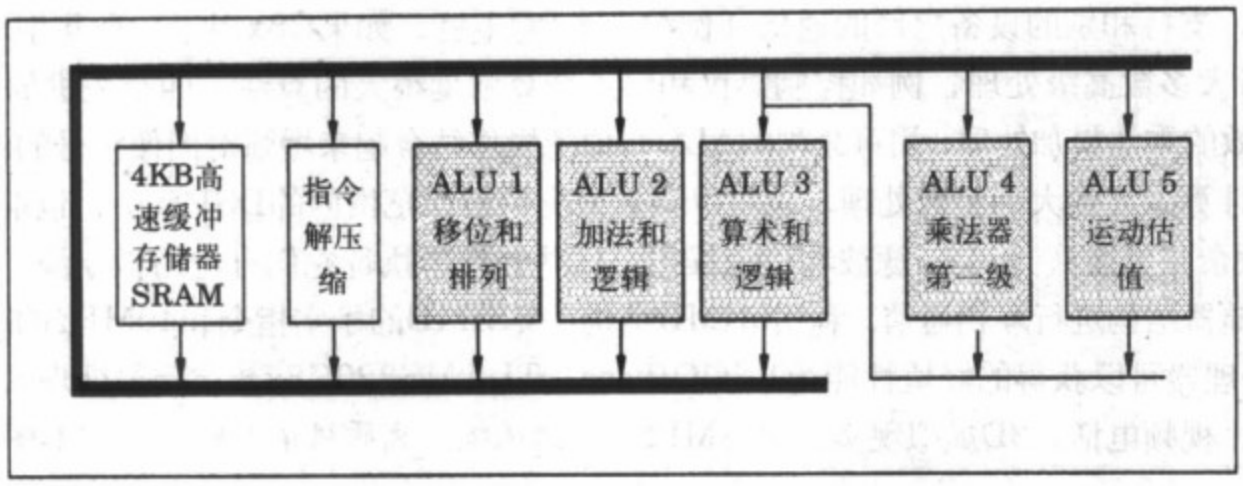


图18-9 Chromatic Mpact 媒体处理器数据路径结构

18.5.2.3 Philips Trimedia

Philips Trimedia处理器[13]有一个多媒体指令系统，类似于Chromatics's Mpact。它可以作为独立的CPU或者作为PC的加速器。Trimedia处理器包含一个具有很强处理能力的VLIW CPU、4个自治的直接存储器访问单元（DMA），还有一个变长译码（VLD）协处理器和一个图像协处理器(ICP)，如图18-10所示。VLIW DSP核有27个执行单元致力于匹配实际多媒体编码的计算需求（参看图18-11）。它的指令集也支持子字并行处理。两个协处理器用于分担处理器核的一部分简单但需要高带宽的任务。在100MHz时钟情况下，Trimedia处理器的峰值处理能力可以达到3.8GOPs/sec。

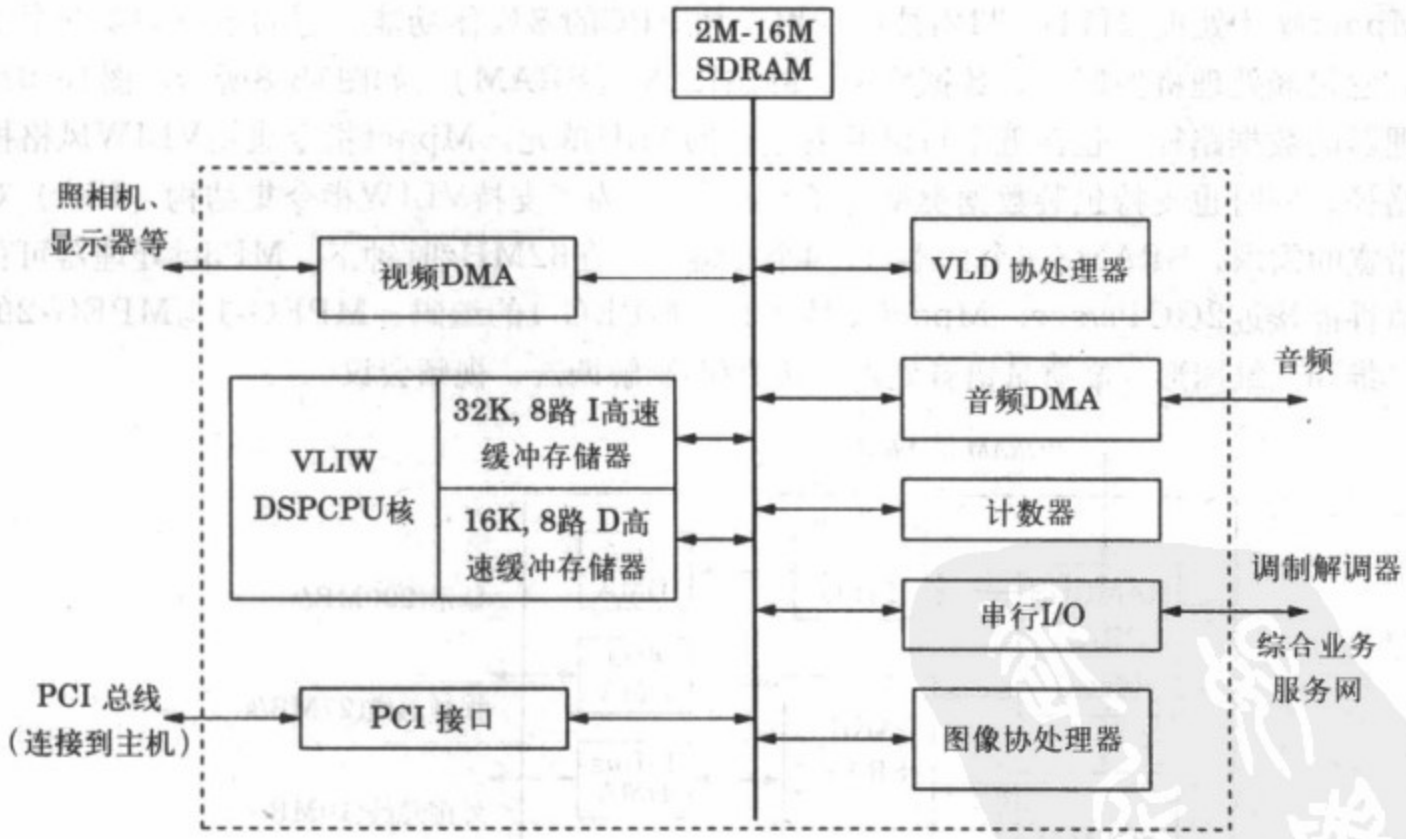


图18-10 Philips Trimedia(TM)处理器结构，TM包括功能很强的VLIW CPU和4个自治DMA单元

18.5.2.4 TI C6x

TI C6x处理器[14]用一个新的VeliciTI结构，这是一个先进的VLIW结构，提供简单、有成本效益的在片并行处理能力。图18-12给出了新颖的VeliciTI VLIW结构。TI C6x的关键特点是：它们是基于处理器-编译器协同设计技术而开发出来的。通常采用高级语言编程、编

译器编译的效率比直接用汇编语言编程低很多。达到84%算是不错了。值得注意的是，对于普通的DSP处理器，编译器仅仅可以获得汇编语言程序员20%~50%的效率[15]。TI 的第C6x代产品适于有线调制解调器、中央办公数据交换、数字图像、先进的多功能无线个人数字助手（PDA）、数字用户环（xDSL）等系统的实时应用中。有关新颖的VeliciTI结构的详细解释，读者可以参看文献[15]和[14]。

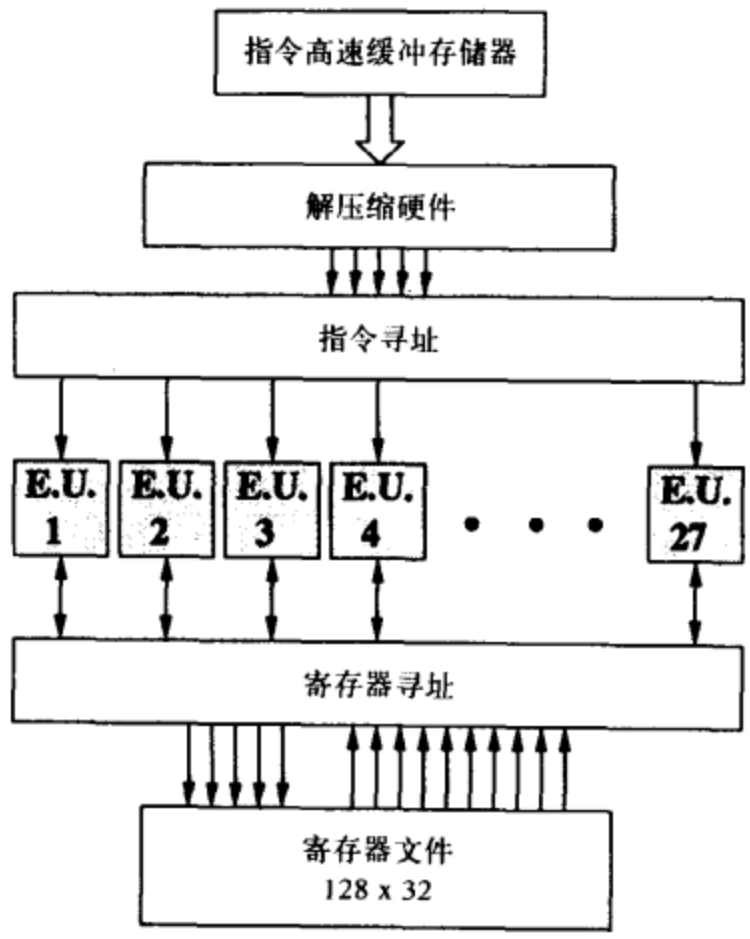


图18-11 Philips Trimedia数据路径结构

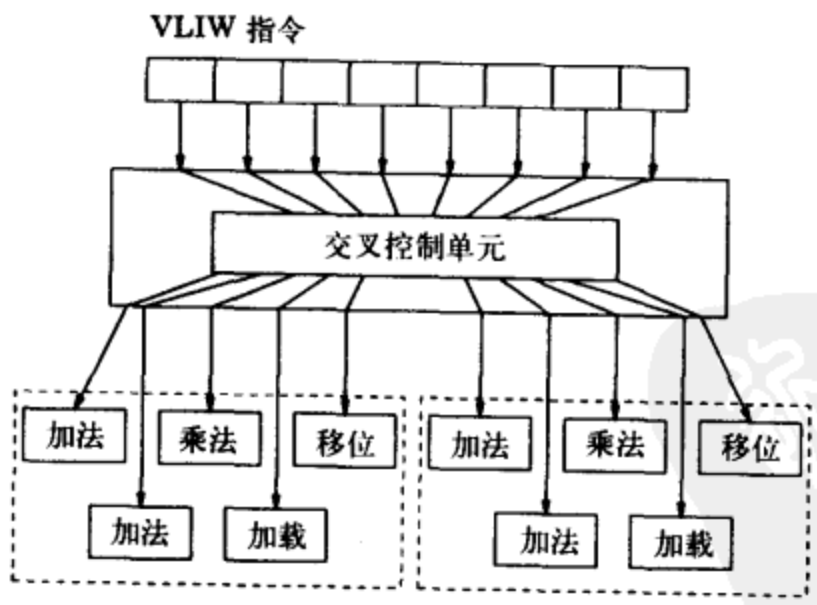


图18-12 TI TMS320C6x *VelociTI* 先进的VLIW结构

18.5.3 具有多媒体辅助部件的通用目的微处理器

在通用目的微处理器设计中，正显现出引入类似DSP的功能以更好地处理多媒体信号任务的趋势。通常，在通用目的处理器中的多媒体辅助部件，例如，在HP PA-RISC工作站中

的多媒体加速辅助部件（MAX）[16]和在Intel ix86处理器中的多媒体辅助部件（MMX）[17]，实现了子字并行处理的概念，加速了所有低精度数据的数据并行处理，包括媒体处理。通过修改处理器结构和增加特殊指令，使得这些处理器结构不仅加速了多媒体程序，而且与以前的指令集结构完全兼容。

因为把ALU运算分裂为子字级的运算是很容易的，所以MAX和MMX都支持并行的子字ALU运算，例如具有饱和算术运算能力的并行加和并行减。但是，在不同的多媒体指令集扩展中，乘法存在很大的差异。乘法的问题是：一个整数乘法器需要占用3到4个加法器所需要的面积，它需要3倍的延迟时间，它产生比每个操作数都长的结果。为了减小面积，当前的MMX仅仅实现一个16比特的整数乘法器，它可以在并行子字乘法指令中重复使用来执行子字乘法。MAX没有任何16比特的乘法器硬件。它用2分叉的策略来支持乘法[9]。那些需要比16比特精度更高的、高强度的、完整功能的乘法器来完成的多媒体计算用已有的PA-RISC浮点功能单元来实现。例如，在这种情况下的乘法用已有的浮点乘法累加单元来实现。那些经常与常数相乘、精度又小于16比特的运算，在这里用已有的带有预移位功能的整数加法器来实现，这个操作要用到并行左/右移和加法指令。

710 ~ 713

18.6 结论

本章介绍了普通DSP处理器的特点，集中讨论了数据路径和存储器架构。探讨了在移动和无线通信中低功耗DSP处理器的特殊需求。阐述了用于多媒体处理的通用的并行和适应技术。讨论了媒体处理器和通用目的微处理器的多媒体辅助部件。

移动和无线通信、多媒体处理将仍然是促进DSP处理器和微处理器发展的动力。DSP处理器的典型的需求包括[2]：

- 灵活性——与所对应的系统相比，它们的应用和它们的标准变化更快。
- 高吞吐量——在几百到几千MOPS的数量级上。
- 低功耗和低成本——用硅片完成桌面和便携式解决方案的需求，成本要小于40美元。

对于移动和无线通信应用而言，灵活性和低功耗、低成本是极端重要的。在该领域，一个有前途的应用就是软件无线电。当从基带调制和信道处理到语音编解码的所有功能都可以用可编程硬件支持的软件实现时，我们就可以带着我们自己的无线终端周游世界了。在不同的国家，为了适应本地的标准和指标，可以通过替换软件来实现。对于多媒体处理应用，灵活性和高吞吐量是至关重要的。在该领域，一个未来的应用实例是带有友好的可编程媒体处理器的数字电视，它支持的标准范围很广，并能传送与电影具有同等质量的图片和声音。

参考文献

714

1. T. Nishitani, "Trend and perspective on domain specific programmable chips," in *Proc. of IEEE Workshop on Signal Processing Systems (SiPS-97)*, pp. 1–8, 1997.

2. B. Ackland, "Programmable multimedia signal processors," in *Circuits and Systems in the Information Age* (Y.-F. Huang and C.-H. Wei, eds.), pp. 23–30, IEEE Press, 1997.

3. P. Lapsley, J. Pier, A. Shoham, and E. Lee, *DSP Processor Fundamentals: Architectures and Features*. Elsvier, 1993.

4. E. A. Lee, "Programmable DSP architectures: part I," *IEEE ASSP Magazine*, pp. 4–19, Oct. 1988.

5. E. A. Lee, "Programmable DSP architectures: part II," *IEEE ASSP Magazine*, pp. 4–14, Jan. 1989.

6. J. Hennessy and D. Patterson, *Computer Architecture A Quantitative Approach*, 2nd ed., Morgan Kaufmann Publishers, 1996.

7. I. Verbauwhede and M. Touriguian, "A low power DSP engine for wireless communications," *Journal of VLSI Signal Processing Systems*, vol. 18, no. 2, Feb. 1998.

8. P. Pirsh, H. Stolberg, Y. Chen, and S. Kung, "Implementation of media processors," in *IEEE Signal Processing Magazine*, July 1997.

9. R. Lee, "Multimedia extensions for general-purpose processors," in *Proc. of IEEE Workshop on Signal Processing Systems (SiPS-97)*, pp. 9–23, 1997.

10. *Digit Signal Processing Selection Guide.* Texas Instruments, 1996.

11. D. Epstein, "Chromatic raises the multimedia bar," *Microprocessor Report*, pp. 23–27, Oct. 1995.

12. P. Kalapathy, "Hardware-software interactions on Mpact," *IEEE Micro*, vol. 17, pp. 20–26, 1997.

13. B. Case, "First trimedia chip boards PCI bus," *Microprocessor Report*, pp. 22–25, Nov. 1995.

14. *TMS320C6x– The Future of DSP Solutions Just Arrived.* Texas Instruments, 1997.

15. R. Simar Jr., "Dsp architectures, algorithms, and code-generation: fission or fusion?," in *Proc. of IEEE Workshop on Signal Processing Systems (SiPS-97)*, pp. 50–59, 1997.

16. R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro*, vol. 16, no. 4, pp. 51–59, Aug. 1996.

17. A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, vol. 16, no. 4, pp. 42–50, Aug. 1996.

附录A 最短路径算法

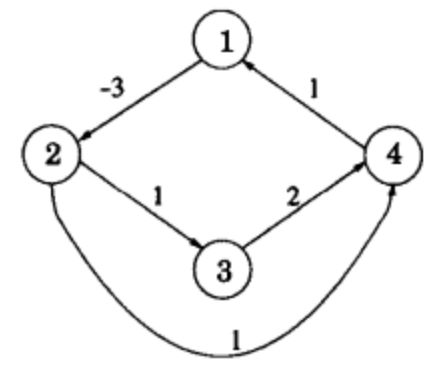
A.1 引言

本附录将介绍最短路径（shortest path）算法中的Bellman-Ford算法和Floyd-Warshall算法。这些算法可以用来解决VLSI信号处理中的几个问题，如递归DFG中的迭代边界计算（见第2章），重定时（见第4章）和展开重定时（见第6章）等。

这里所包含内容的详细程度是按读者利用这些算法能够足以解决上面提到问题的需要设定的。这些算法的相关理论从略；有兴趣的读者可参看专门讨论这些算法相关理论的文献（如[1]，[2]）。

图A-1中的DFG说明了最短路径和负回路(negative cycle)的概念。一条路径的长度是该路径上边的权重之和。路径 $2\to3\to4$ 的长度是$1+2=3$。节点2到节点4有两条路径，即长度为1的路径 $2\to4$ 和长度为3的路径 $2\to3\to4$。从节点2到节点4的最短路径长度是 $\min\{1,3\}=1$。一个有向图中的负回路是指该回路上所有边的长度之和为负数的回路。图A-1中包含两个回路：$2\to4\to1\to2$和$2\to3\to4\to1\to2$。回路$2\to4\to1\to2$是一个负回路，这是因为它所包含边的权重之和是$1+1+(-3)=-1$。另一个回路$2\to3\to4\to1\to2$不是负回路，因为它所包含边的权重之和是$1+2+1+(-3)=1$。基于这一情况，两个最短路径算法的基本思想如下。如果图中没有负回路，则算法返回TRUE，并给出两个节点之间最段路径长度的一些信息（下面将看到，这里所提供的信息量对两个算法是不同的）。如果存在至少一条负回路，则算法返回FALSE。总之，两个算法中的每一个都在搜索负回路，并且当没有负回路存在时提供最短路径的信息。

717



图A-1 包含一负回路的图

我们使用下面的假设和记号。假定一个图中没有并行边存在。也就是说，对节点U和V，从U到V最多有一条边 $(U\xrightarrow{e}V)$存在，从V到U也最多有一条边$(V\xrightarrow{e}U)$存在。记

$$w(U\xrightarrow{e}V)=\begin{cases}\text{边}(U\xrightarrow{e}V)\text{的长度，如果边}(U\xrightarrow{e}V)\text{存在}\\ \infty \qquad\qquad\qquad\qquad \text{否则}\end{cases}$$

这里边的长度是一个与该边相关的数。记 n 为图中节点数，并假设 n 个节点分别是1，2，…，n等。从节点U到节点V之间的最短路径记为 S_{UV}。

A.2 Bellman-Ford算法

Bellman-Ford算法是一个单点最短路径算法。如果没有负回路存在，它将给出从任意选定节点U（称为原点）到图中其他每一个节点之间的最短路径。对于一个有 n 个节点的图，该算法将构造 $n-1$ 个矢量 $\mathbf{r}^{(k)}$, $k=1,2,\cdots,n-1$，其中每一个大小是 $n\times1$，如图A-2所示。注意 $\mathbf{r}^{(k)}(U)$ 表示矢量中的第U个分量。

718

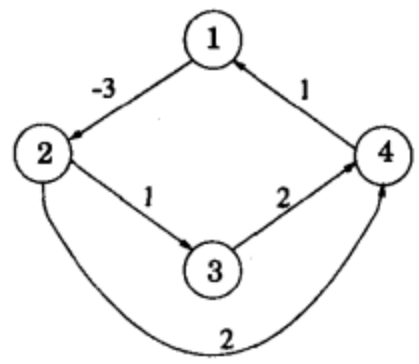
```
r^(1)(U) = 0
For k = 1 to n
    If k ≠ U
        r^(1)(k) = w(U -e-> k)
For k = 1 to n − 2
    For V = 1 to n
        r^(k+1)(V) = r^(k)(V)
        For W = 1 to n
            If r^(k+1)(V) > r^(k)(W) + w(W -e-> V)
                r^(k+1)(V) = r^(k)(W) + w(W -e-> V)
For V = 1 to n
    For W = 1 to n
        If r^(n−1)(V) > r^(n−1)(W) + w(W -e-> V)
            return FALSE and exit
return TRUE and exit
```

图A-2 Bellman-Ford算法

如果返回TRUE，则 $r^{(n-1)}(V)$ 就是 S_{UV}，即从节点U到节点V的最短路径。如果返回FALSE，则该图至少包含一个负回路。

例A.2.1 本例将利用图A-2给出的Bellman-Ford算法求由图A-3给出的图中由节点U=2到其他各个节点的最短路径。k=1, 2, 3, V=1, 2, 3, 4时 $r^{(k)}(V)$ 的各值如表A-1。本例中Bellman-Ford算法返回TRUE，所以 $r^{(3)}(V)$ 表示了由节点2到节点V的最短距离，这里 V=1, 2, 3, 4。 ■

719



图A-3 不包含负回路的一个图

例A.2.2 这一例子将利用Bellman-Ford算法求图A-1所给出图中由原点U=2到其他各点的最短路径。k=1, 2, 3, V=1, 2, 3, 4时，$r^{(k)}(V)$ 的各值如表A-2所示。本例中Bellman-Ford算法返回了FALSE，因为 $r^{(3)}(3) > r^{(3)}(2) + w(2 \xrightarrow{e} 3)$，找到了一个负回路。该负回路能够从图A-1中看出：回路 2→4→1→2 的长度为1+1−3=−1。 ■

表A-1 例A.2.1中k=1, 2, 3, V=1, 2, 3, 4时 $r^{(k)}(V)$ 的各个值

$r^{(k)}(V)$	$k=1$	$k=2$	$k=3$
$V=1$	∞	3	3
$V=2$	0	0	0
$V=3$	1	1	1
$V=4$	2	2	2

表A-2 例A.2.2中 k=1, 2, 3, V=1, 2, 3, 4时 $r^{(k)}(V)$ 的各个值

$r^{(k)}(V)$	$k=1$	$k=2$	$k=3$
$V=1$	∞	2	2
$V=2$	0	0	−1
$V=3$	1	1	1
$V=4$	1	1	1

A.3 Floyd-Warshall 算法

Floyd-Warshall 算法是一个全点（all-points）最短路径算法。如果图中没有负回路，则算法能搜索出图中所有可能点对之间的最短路径。算法将构造出$n+1$个矩阵 $\mathbf{R}^{(k)}$, k=1, 2, …, n+1，其中的每一个矩阵都是 $n \times n$ 阶的。Floyd-Warshall 算法的描述见图A-4。

如果算法返回TRUE，则 $r^{(n+1)}(U, V)$ 就是 S_{UV}，即从节点U到节点V之间的最短路径。如果算法返回FALSE，则该图中包含负回路。

```
For V = 1 to n
    For U = 1 to n
        r^(1)(U,V) = w(U -e-> V)
For k = 1 to n
    For V = 1 to n
        For U = 1 to n
            r^(k+1)(U,V) = r^(k)(U,V)
            If r^(k+1)(U,V) > r^(k)(U,k) + r^(k)(k,V)
              r^(k+1)(U,V) = r^(k)(U,k) + r^(k)(k,V)
For k = 1 to n
    For U = 1 to n
        If r^(k)(U,U) < 0
          return FALSE and exit
return TRUE and exit
```

图A-4 Floyd-Warshall算法

例A.3.1 这一例子中，Floyd-Warshall算法被用来解决如图A-4中所给出图的全点对（all-pairs）最短路径问题。对 U, $V \in \{1,2,3,4\}$ 和 k=1,2,3,4,5，$r^{(k)}(U, V)$ 的值由表A-3给出，这里 $r^{(k)}(U, V)$ 表示矩阵 $\mathbf{R}^{(k)}$ 的第U行第V列元素。因为 k=1,2,3,4 时所有矩阵 $\mathbf{R}^{(k)}$ 的对角线上的元素都非负，所以Floyd-Warshall 算法的返回值是TRUE，从而 $r^{(5)}(U, V)$ 是节点U到节点V之间的最短路径。 ■

表A-3 例A.3.1中 $r^{(k)}(U, V)$ 的各个值

$$\mathbf{R}^{(1)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & \infty & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(2)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(3)} = \begin{bmatrix} \infty & -3 & -2 & -1 \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & 0 \end{bmatrix}$$

$$\mathbf{R}^{(4)} = \begin{bmatrix} \infty & -3 & -2 & -1 \\ \infty & \infty & 1 & 2 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & 0 \end{bmatrix} \quad \mathbf{R}^{(5)} = \begin{bmatrix} 0 & -3 & -2 & -1 \\ 3 & 0 & 1 & 2 \\ 3 & 0 & 1 & 2 \\ 1 & -2 & -1 & 0 \end{bmatrix}$$

A.4 计算复杂度

Bellman-Ford算法能检测到负回路的存在，在没有负回路存在时可在时间 $O(n^3)$ 内解决单原点（single-source）最短路径问题；Floyd-Warshall算法也能检测到负回路的存在，并

且在没有负回路存在时可在时间 $O(n^3)$ 内解决全点对（all-pairs）最短路径问题。Bellman-Ford算法需要 $O(n^4)$ 时间才能解决全点对最短路径问题，这是因为它需要对图中所有点遍历一遍。对于需要给出全点对解的应用场合，Floyd-Warshall算法是首选算法，而对于要给出单原点解的应用场合，Bellman-Ford算法和Floyd-Warshall算法可以随便选用。

Bellman-Ford算法的各种改进和两种算法的提前结束条件可用来提高算法的计算效率。有兴趣的读者可以在[1]中找到这些细节。

表A-4 例A.3.2中 $r^{(k)}(U, V)$ 的各个值

$$\mathbf{R}^{(1)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & \infty & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(2)} = \begin{bmatrix} \infty & -3 & \infty & \infty \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & \infty & \infty \end{bmatrix} \quad \mathbf{R}^{(3)} = \begin{bmatrix} \infty & -3 & -2 & -2 \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & -1 \end{bmatrix}$$

$$\mathbf{R}^{(4)} = \begin{bmatrix} \infty & -3 & -2 & -2 \\ \infty & \infty & 1 & 1 \\ \infty & \infty & \infty & 2 \\ 1 & -2 & -1 & -1 \end{bmatrix} \quad \mathbf{R}^{(5)} = \begin{bmatrix} -1 & -4 & -3 & -3 \\ 2 & -1 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & -3 & -2 & -2 \end{bmatrix}$$

参考文献

1. E. L. Lawler, *Combinatorial Optimization: Networks and Matriods.* Rinehart and Winston, 1976.

720 ~ 722

2. T. H. Cormen, C. E. Leiserson and R. C. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

附录B 调度技术和分配技术

B.1 引言

调度和分配是DSP系统软硬件综合中两类重要的工作。它们既相互关联又相互依存，属于高层次综合(high-level synthesis)中最困难的问题[1]，[2]。*调度*（Scheduling）需要将DFG中的每一个节点安排到*时间控制步*(control time step)。时间控制步（或简称时间步）是同步系统中的基本序列单元且与时钟周期相对应。*资源分配*(Resource allocation)是指在实现预定行为时以硬件需求最小化为目标而安排操作的过程。硬件资源本质上由功能单元、存储单元、乘法器和数据通道组成。

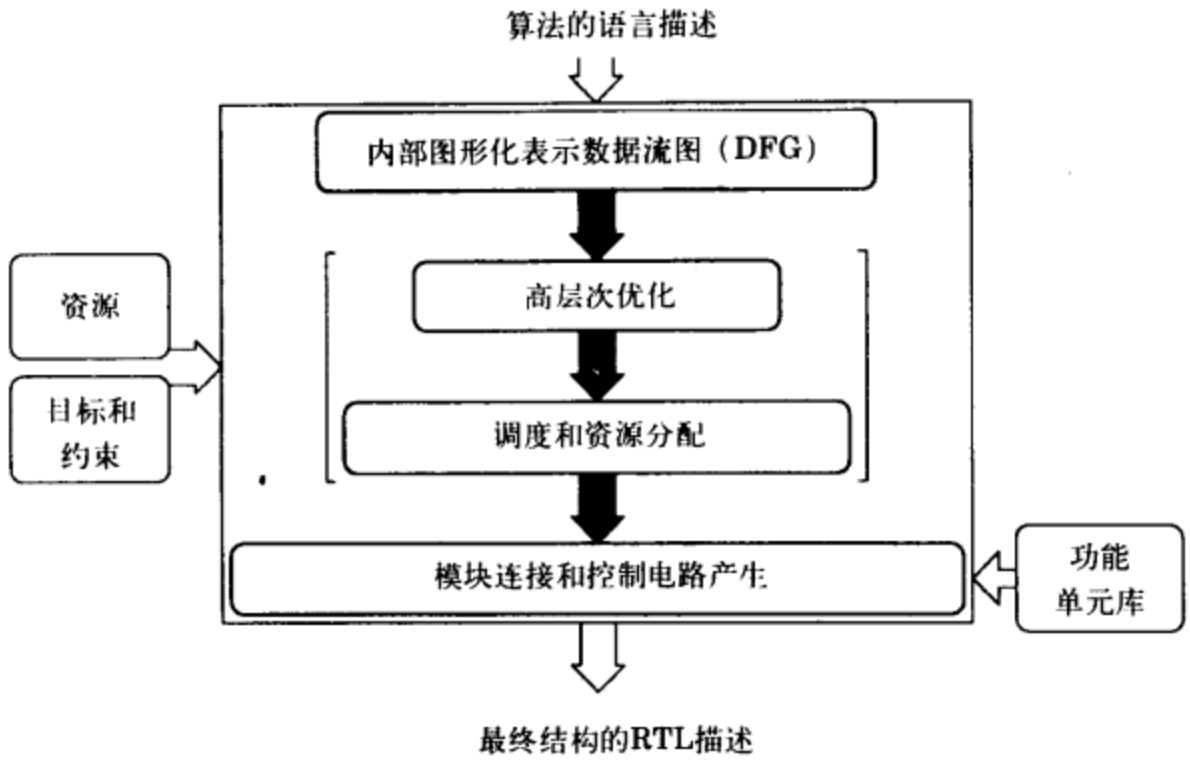
DSP综合系统以一组输入，包括算法的行为描述，一组资源和一组约束条件及目标为依据，产生一个寄存器传输级（RTL）结构。这些约束条件及目标将定义最终硬件结构中要求的性能与特性。面积约束问题提供给设计者的是一组资源（或更具体地是一组功能单元），其目标是应用这些资源以最高的性能实现需要它们完成的应用。这被称作*约束资源综合*(resource-constrained synthesis)。性能约束问题被称为*约束时间综合*(time-constrained synthesis)，其中对设计者给定了一个采样速率或迭代周期，目标是要对最终结构的总面积进行最小化。综合过程中的其他目标包括最小化存储单元的数量、降低功耗、最小化总线数量和增加可靠性设计与可测试性设计等。

图B-1说明了一个高层次DSP综合系统的设计流程。高层次综合的第一步是将行为描述转换为*控制与数据流图*(control and data-flow graph,CDFG)或者DFG形式下基于图的内部表示。高层次综合中的中间任务包括高层次优化、调度、资源分配、模块连接和控制电路产生等。由高层次综合产生的最终结构属于典型的RTL，它能够以由功能单元、寄存器、乘法器和连线的网络组成的网表形式表示。低层次优化工具可以用来将所产生的RTL描述映射成一种物理设计。目前已经设计出了很多高层次综合系统，并且在寻找更好技术以便优化设计和探索设计折中方面有了长足进步。此外，对设计过程中更高层次上更进一步的自动化仍是继续发展的一种趋势。

本附录中，我们认为CDFG和DFG是相同的。这一点对于像不包含有条件操作和嵌套循环的控制优先（control-dominated）DSP系统是可行的。

高层次综合被认为是NP困难的。这是因为综合中的很多子任务，例如调度和分配，已被证明是NP完全的，即找到一个最优解所需要的计算步骤，在问题的大小以线性速度增长时，它将以指数速度增长。因为很多子任务是相互关联的，不能被单独分离出来，所以整个综合任务也就是NP困难的。

本附录讨论各种约束时间调度技术和分配技术，包括迭代型/构造性调度（B.2节）、变换型调度（B.3节）和整数线性规划(ILP)（B.4节）等。



图B-1 高层次DSP综合系统

B.2 迭代型/构造性调度算法

一个迭代型/构造性调度算法通过将DFG的节点按每次增加一个，直到所有点被调度完毕的方法来产生一个调度表。选择将要被调度的节点和决定这些节点被调度到什么地方的方法有很多，两种较简单的方法是“尽可能地早”和“尽可能地晚”的调度算法。对这两个调度算法，下面用一个2阶微分方程的迭代公式作为例子说明。

考虑以下2阶微分方程及其相应的C程序公式:

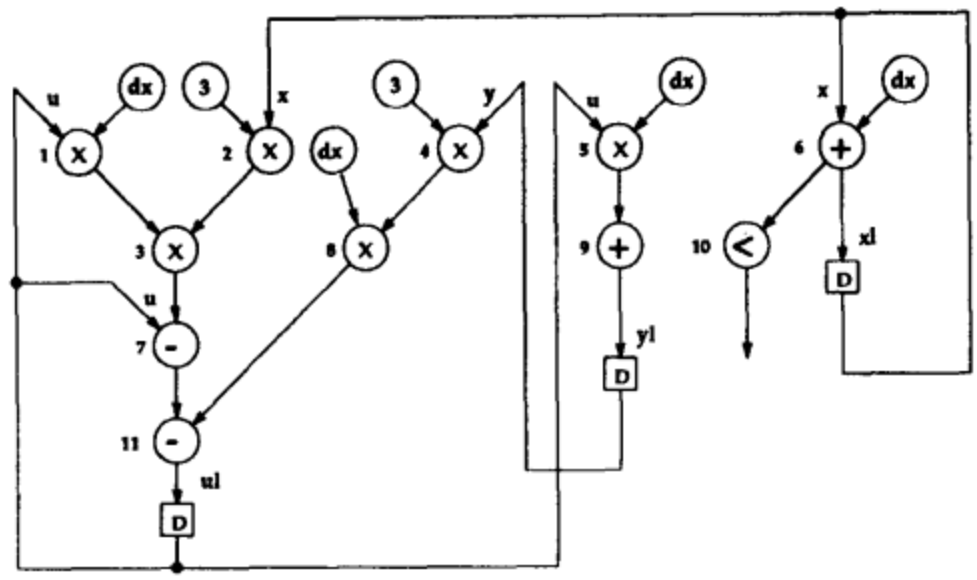
$$y'' + 3xy' + 3y = 0 \qquad u = y' = \frac{dy}{dx}$$

$$\frac{du}{dx} = y'' = \frac{d^2y}{dx^2} = -3xy' - 3y = -3xu - 3y$$

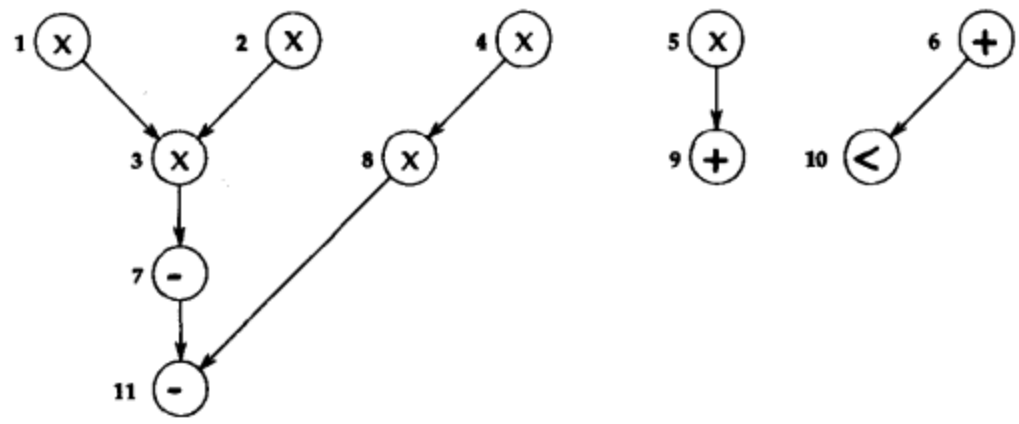
```
当 (x < a) {
    xl = x + dx;
    ul = u - (3 * x * u * dx) - (3 * y * dx);
    yl = y + u * dx;
    x = xl; y = yl; u = ul;
}
```

它的DFG如图B-2所示。这个DFG中，假定乘法和加法操作在一个时间步内执行。很多综合系统通过去掉该DFG图中所有权重大于零的边来构造一个比原图更简单的图并加以使用。这样能够得到一个只包含一次迭代的优先约束或只包含迭代内优先约束（intra-iteration precedence constraint）的图，称为优先图(precedence graph)。图B-2b给出了图B-2a中所示DFG的优先图。注意优先图总能保证无回路。

724 ~ 725



a) 2阶微分方程求解公式的DFG图



b) 从DFG得到的优先图

图 B-2

B.2.1 尽可能早(ASAP)调度算法

"尽可能地早"(As soon as possible，ASAP)调度算法是最早也是最简单的一种调度算法[3]。按照这种方法，在节点选择技术和将节点置于调度表的方法中使用了局部准则。ASAP调度假定存在有无限资源。首先将节点做一个拓扑分类。也就是说，假如节点 n_j 被一个优先约束限制为紧跟节点 n_i，则 n_j 在拓扑上紧跟 n_i。由这一分类表，每次能够处理一个节点，并按照它的优先约束可以将其置于最早可能的时间步。ASAP算法如图B-3所示。图B-4给出了对图B-2中DFG的ASAP调度，其中假定资源无限。 726

输入：DFG $G=(N, E)$

输出：ASAP调度表

1. TS_0=1; /*置初始时间步*/

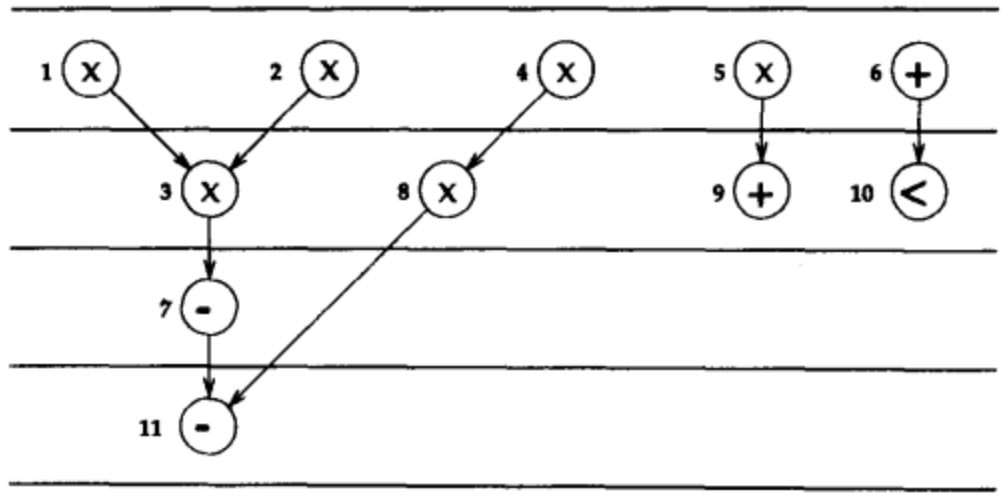
2. 当（存在未调度点）{

  2.1 选一个其前面的节点已经被调度的点 n_j;

  2.2 调度节点 n_j 到时间步 $TS_j=\max\{TS_i+(C_i)\} \quad \forall n_i \rightarrow n_j$

}

图B-3 ASAP调度算法



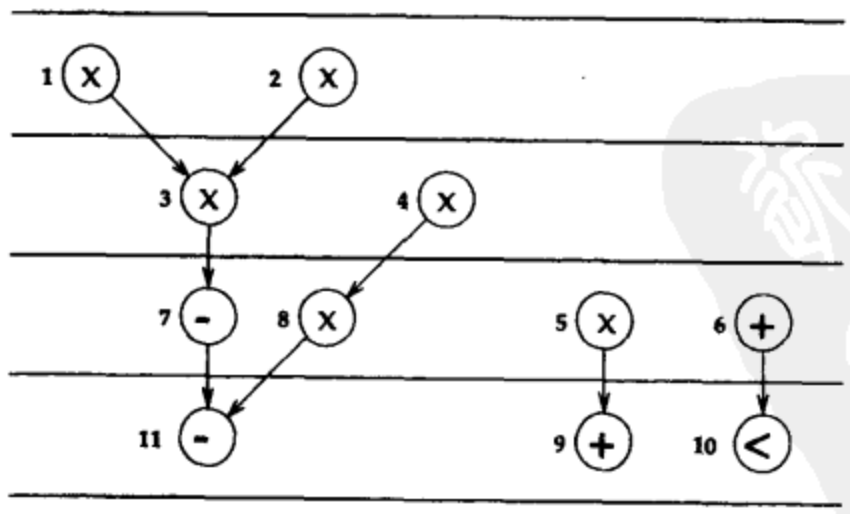
图B-4 2阶微分方程的ASAP调度表

B.2.2 尽可能晚（ALAP）调度算法

“尽可能地晚”（As late as possible）调度类似于ASAP调度算法。ASAP和ALAP调度不同的地方在于将节点放置到调度表的方式不同。如其名称所指出的，ALAP调度算法将从底端由下而上地建立调度表，对节点的分类次序正好相反。因此，这一算法要求必须掌握迭代阶段内的情况以便自下而上地建立调度表。必须有足够长的预期迭代周期以使算法能够对DFG中的所有节点进行调度，否则调度将失败。ALAP调度示于图B-5。图B-6给出了图B-2中DFG的ALAP调度表。

输入：DFG $G=(N, E)$，迭代周期$=T$。
输出：ALAP调度表。
1. $TS_0 = T$; /*置初始时间步骤*/
2. 当（存在未调度节点）{
   2.1 选一个其后面的节点已经被调度的节点 n_i;
   2.2 调度节点 n_i 到时间步骤 $TS_i = \max\{TS_j - (C_i)\} \quad \forall n_i \rightarrow n_j$
}

图B-5 ALAP调度算法



图B-6 2阶微分方程的ASAP调度表

虽然ASAP算法和ALAP算法都不能产生很好的结果，但是正如在本附录后面证实的，作

为一种久经考验的技术，它们仍然广泛地被用来确定更复杂技术的调度域(scheduling range)。一般来说，ASAP调度定义节点开始时间的最小值，而ALAP调度提供最大值。两个不同的值提供了DFG中每一节点的调度域。ASAP调度还用来产生初始调度，这种初始调度又能通过其他更有效的调度技术进行改进。

B.2.3 表单调度算法

ASAP和ALAP调度算法的主要问题是没有给关键路径上的节点赋予优先权。从而，不太关键的节点很可能在真正的关键节点之前被调度。当有无限多功能单元时，这种情况一般说来不是问题；但是，当资源受限时（也就是说，在一个资源受限问题中），不太关键的节点可能阻碍关键节点而产生出不好的调度表。为克服这一问题，提出了*表单调度技术*(list scheduling technique)，它使用了更好的全局性节点选择准则[6]。 727~728

输入：DFG $G=(N, E)$，$R=(FU)$。
输出：最终调度表。
1. $TS_0 = 1$; /*置初始时间步*/
2. 当（存在未调度节点）{
  2.1 查找其前向节点已经被调度的所有节点并将它们置于表单L;
  2.2 按关键性减低对L中节点进行分类;
  2.3 当(L非空){
    2.3.1 从L中选择第一个节点 n_j;
    2.3.2 确定时间步 $TS_j = \max\{TS_i + (C_i)\} \quad \forall n_i \rightarrow n_j$;
    2.3.3 如果(TS_j 处的 FU_S 未满)
      调度节点 n_j 到时间步 TS_j
    否则
      从L中重新取节点。
  }
}

图B-7 一个简单的通过减低关键性方法优选节点的表单算法

在表单调度技术中，首先把所有即将要被调度的节点通过某种优选函数进行分类，使它们列成一个表单，并对表单上的节点进行调度，直到功能单元被使用完毕为止。然后对所有剩余的节点和新的将要被调度的节点再进行分类以得到新的表单。继续这种迭代过程，直到所有节点都被调度。表单调度的缺点是这些算法需要有关资源数量方面的某种知识，因而只能被应用于约束资源问题。图B-7提供了一个简单的表单调度算法，它是通过降低关键性来对将要被调度的节点进行优选的。这里的关键性是由调度域决定的。域越小，节点的关键性越大。图B-7中，R表示资源集合，被限制的资源是功能单元(FU)。

B.3 变换调度算法

一个变换型调度算法开始于一个初始调度，这个初始调度要么是串行最大化的，要么是并行最大化的。将变换应用于这个初始调度和中间调度以得到其他调度，直到产生一个最终 729

解为止。基本的变换是将那些串行（或并行）的节点移动到调度表，使它们在那里更显得像并行（或串行）排列。根据它们所使用变换类型的不同，这里的处理方法也彼此不同。这些调度算法一般说来比迭代型/构造性算法能提供更好的结果。

B.3.1 力度导向调度

变换调度中一个好的例子是HAL系统中发展起来的*力度导向调度*(force-directed scheduling, FDS)。力度导向调度中,节点选择原则和节点调度技术使用了更加全局性的信息[5]。HAL计算一种能够尽量在一个操作和一个特别的时间步之间施加的力（force）。这个力与一个特别的时间步中可以被调度到同一功能单元上的操作数量成正比。这时，调度算法将这种能够平衡DFG图中节点同时出现的总力最小化。图B-8给出了基本的力度导向调度算法。

```
输入: DFG G = (N, E), 迭代周期=T。
输出: 最终的FDS调度表。
1. 当（存在未调度节点）{
    1.1 对每一节点计算其时帧;
    1.2 建立分布图;
    1.3 计算自力;
    1.4 计算前力和后力;
    1.5 调度该节点到总力最小的时间步;
  }
```

图B-8 力度导向调度算法

从图B-8可以看出，基本FDS算法的第一步涉及到对ASAP调度表或ALAP调度表中每一节点时间帧（time frame）的确定。这些时间帧代表的是节点在每一时间步被指派的概率。因而对每一个节点，其时间帧的面积等于1。如果一个节点能在时间步区间 $[S_i, L_i]$ 内被调度，则在此时间步区间内该点被各时间步调度的概率是 $\frac{1}{L_i - S_i + 1}$。图B-9a给出了图B-2中DFG的时间帧图和相应的概率。第二步是对每一时间步中各节点同类型操作的概率进行求和，为每一个功能单元类型创建一个*分布图*（distribution graph）（见图B-9）。这些图是DFG中所出现同类操作的一个图形表示。

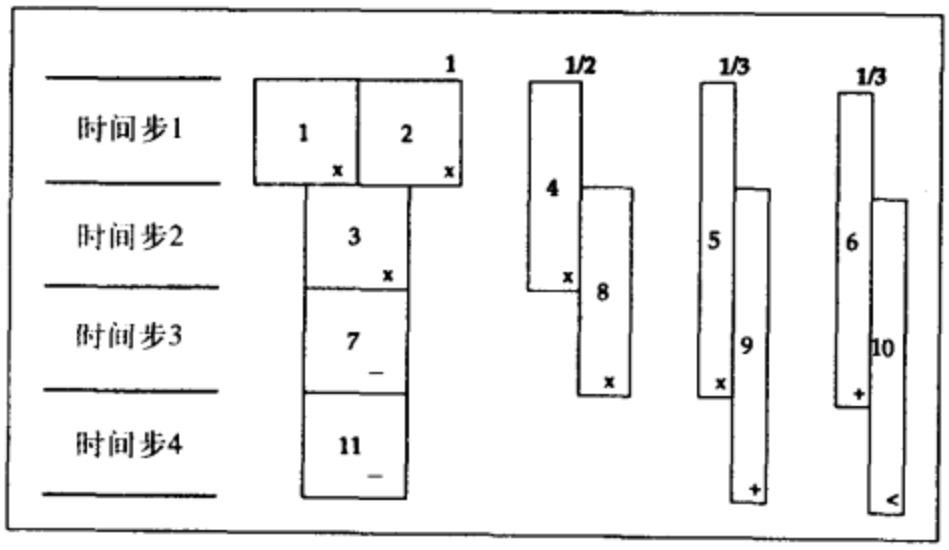
730

从分布图中我们能够计算出每一个节点被调度到一定时间步时所产生的力。对每一节点，从DFG中可以确定两种类型的力: *自力*(self force)和*前/后力*(predecessor/successor forces)。自力反映的是尝试调度一个点到确定时间步时对最终实现这种调度所产生的效应。如果这种指派能引起最终被调度机会的增加，它将取正值，如果能引起被调度机会的减少，它将取负值。对节点 n_j，在每一时间步 TS_j 内（在该节点时间帧内（$S_j < TS_j < L_j$））的各种指派，自力的计算如下:

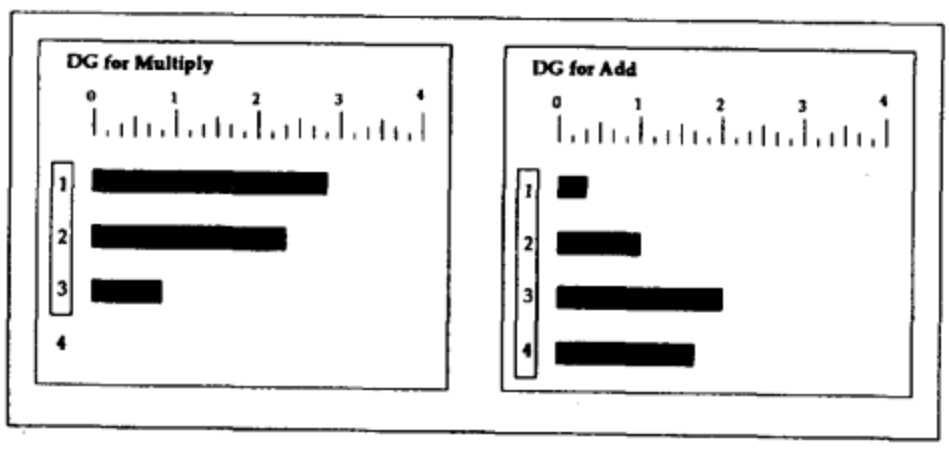
$$Self_Force(j) = \sum_{i=S_j}^{L_j} [DG(i) * x(i)]$$

其中 $DG(i)$ 是时间步 i（在分布图中）的分布值，$x(i)$ 是与时间步 i 相联系的概率变化值。先

/后力是将一个节点指派到一个时间步后所形成的前后节点处的自力。作用在一个点上的力就是自力、先力和后力的总和。于是，具有最小总力的点将被指派到该时间步。当一个节点被指派到一个时间步后，对DFG中的其他剩余节点再重新重复上面的过程，直到所有节点都被指派到一定的时间步。



a) 2阶微分方程的初始时间帧图，它显示出了每一个节点被调度到一个特别时间步的概率(注意各节点对应的面积相同)



b) 2阶微分方程的分布图

图 B-9

作为一个例子，现在应用力度导向调度考虑图B-9中对节点4的调度。可以看出，节点4能够被调度到时间步1或时间步2。因此我们必须计算两种情况下的自力。

$$
\begin{aligned}
Self_Force_4(1) &= Force_4(1) + Force_4(2) \\
&= (DG_M(1) * x_4(1)) + (DG_M(2) * x_4(2)) \\
&= (2.833 * (1 - 0.5)) + (2.333 * (0 - 0.5)) \\
&= (2.833 * (+0.5)) + (2.333 * (-0.5)) \\
&= +0.25
\end{aligned}
$$

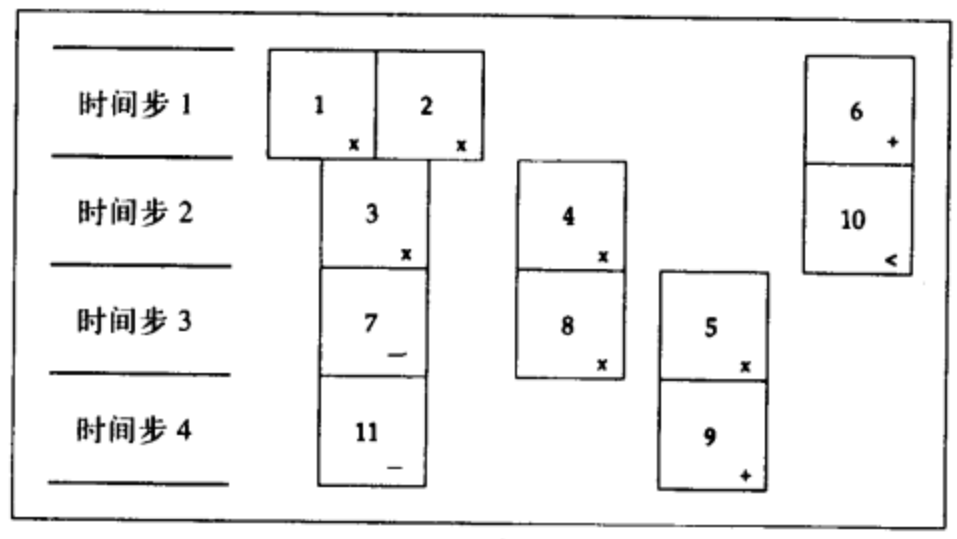
$$
\begin{aligned}
Self_Force_4(2) &= Force_4(1) + Force_4(2) \\
&= (DG_M(1) * x_4(1)) + (DG_M(2) * x_4(2)) \\
&= (2.833 * (-0.5)) + (2.333 * (+0.5)) \\
&= -0.25
\end{aligned}
$$

如果节点4被调度到时间步1，它将不会引起其后续点，即节点8的时间帧变化。因此，将节点4调度到时间步1时的总力是 $Force_4(1)=+0.25$。然而，如果将节点4调度到时间步2，它将对其后续点调度到时间步3造成影响。这时需要计算后力并且需要将该后力包括到总力中。

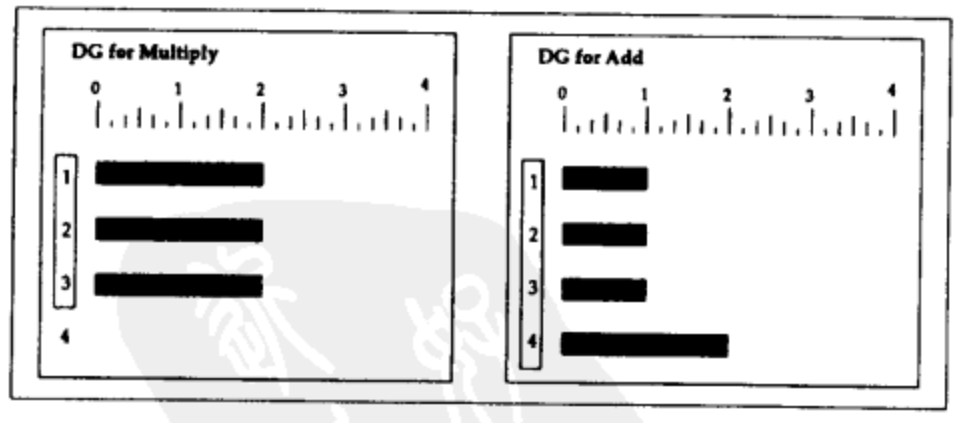
$$
\begin{aligned}
Succ_Force_4(2) &= Self_Force_8(2) + Self_Force_8(3) \\
&= (DG_M(2) * x_8(2)) + (DG_M(3) * x_8(3)) \\
&= (2.333 * (0 - 0.5)) + (0.833 * (1 - 0.5)) \\
&= (2.333 * (-0.5)) + (0.833 * (+0.5)) \\
&= -0.75
\end{aligned}
$$

$$
\begin{aligned}
Force_4(2) &= Self_Force_4(2) + Succ_Force_4(2) \\
&= -0.25 - 0.75 = -1.00
\end{aligned}
$$

当计算完节点4在其时间帧内所包含各时间步的总力后，就可以选择具有最小力的指派，即节点4调度到时间步2。通过确定新的时间帧图和建立新的分布图和进一步调度另外一个点来重复上述过程。图B-10给出了最后的时间帧图和表示了最终调度的分布图。



a) 2阶微分方程的最后时间帧图和调度表



b) 2阶微分方程最后的分布图

图 B-10

每次调度一个剩余的点到一个时间步时都需要计算它的总力。这一过程需要很长时间。为减少调度任务的执行时间，可以使用能够快速产生类似结果的启发式变换。这里需要注意，基于这种目的所讨论的调度算法是一种关键路径（critical path）调度算法，也就是说，它只能产生迭代周期等于或大于关键路径的调度。它们只揭示了单个迭代内的并发性，只利用

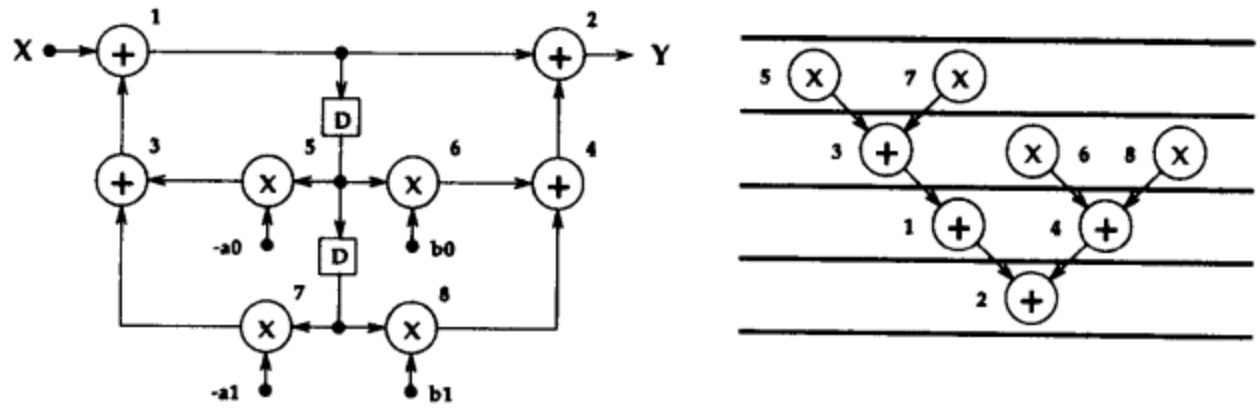
了迭代内优先约束。

作为另一个例子，图B-11a给出了一个简单的2阶IIR滤波器，或实现下列函数的双二次滤波器(biquad filter):

731 ~ 733

$$y(n) = x(n) + b_0x(n-1) + b_1x(n-2) - a_0y(n-1) - a_1y(n-2)$$

如果我们假定加法和乘法操作所需要的计算时间是1u.t.，则该滤波器有一条关键路径CP = 4 u.t.。图B-11b给出了一个关键路径调度器可能会产生的一种调度。注意迭代周期是4 u.t.，这需要2个乘法器和2个加法器。为产生更好的调度，可以使用高层次变换来优化DFG（例如流水线和重定时）以缩短关键路径，使其变成一个预处理步骤。如果图B-11a中的DFG被流水或被重定时，关键路径能够减少到3 u.t.，功能分配单元数也能减少。一些最新的技术通过使用调度过程中的这些高层次变换消除了这个预处理步骤。这里有两个例子中包括了基于调度技术的迭代性环路（[9]~[12]）和使用了整数线性规划模型([13]~[15])。



a) 一个简单双二次滤波器的DFG

b) 一个由关键路径调度器产生的最小迭代周期为4u.t.的调度表

图 B-11

B.3.2 基于调度算法的迭代环路

基于迭代环路综合系统的一个例子是明尼苏达结构综合系统，或叫MARS系统[11]，[12]。MARS使用了一种基于环路的启发式调度技术，它在建立调度表的同时还能够对DFG进行某种流水和重定时。环路调度技术是由反馈或是由递归环路为一个DFG中限制性最强的部分这一事实而发展起来的。环路，这种在DSP应用中比较普遍的现象是由反馈边产生的。因此，在当前的迭代完成之前不能对下一轮迭代进行初始化。这些反馈边必须包括一个或更多的延时以保证DFG是有效的。这些边表示迭代间优先约束(inter-iteration precedence constraint)。MARS调度过程既利用了迭代间优先约束，也利用了非迭代间优先约束，来对DFG进行间接流水和重定时，以便得到更好的结果。基本的基于调度算法的迭代性环路如图B-12所示。

对一个环路调度，第一步是查找所有的环路并通过递减环路边界对它们进行分类。环路边界（loop bound）是执行一次环路迭代所需要时间的下限，对每一个环路其值可计算如下: 734

$$T_{lb_l} = \frac{t_l}{w_l}$$

其中 t_l 是总的计算时间，w_l 是环路 l 上的延迟数。在全部由环路构成的集合中，可以区分出

一个子集合，该子集合中的环路全部由递归（或环路）节点组成。MARS可以利用这一子集合调度递归节点（recursive node）。在考虑递归节点时，对非递归节点不再考虑，因为这些点属于前向路径，必要时它们能很容易地被流水，而递归节点属于环路，它们不易被流水。

```
输入: DFG G=(N, E), 迭代周期=T。
输出: 最终调度表。
1. 查找DFG中的所有环路;
2. 通过递减关键性对环进行分类;
3. 查找一个能够覆盖所有环路上节点的环路子集合;
4. 查找功能单元的下限;
5. 建立初始调度表;
6. 查找所有资源冲突;
7. 当（调度表中存在资源冲突）{
    7.1 对（TS=1; TS<T; TS++; ）{
      7.1.1 如果(在TS处存在资源冲突)
            通过重新调度解决冲突或增加所分配的功能单元;
      7.1.2 更新资源冲突;
      }
    }
8. 查找所有前向路径;
9. 通过递减关键性对路径进行分类;
10. 查找能够覆盖所有路径上节点的一个路径子集合;
11. 分配剩余的功能单元;
12. 建立初始调度表;
13. 查找所有资源冲突;
14. 当（调度表中存在资源冲突）{
    14.1对（TS=1; TS<T; TS++; ）{
      14.1.1如果(在TS处存在资源冲突)
            通过重新调度解决冲突或增加所分配的功能单元;
      14.1.2更新资源冲突;
      }
  }
```

图B-12 基于MARS迭代环路的调度算法

下一步是创建一个列表示环路、行表示时间步的调度矩阵。从环路边界最大的那个环路开始，通过假定具有无限的资源数可以利用，让每一环路上的未调度节点进入该调度矩阵使得迭代内优先约束得到满足。据此可以建立初始调度表并可以反复利用变换产生最终的调度表。在开始任何变换之前，MARS通过下式先计算DFG所需功能单元数的一个下限:

$$M_{t_k} = \left\lceil \frac{N_{t_k} * T_{t_k}}{P_{t_k} * T} \right\rceil$$

其中 M_{t_k} 表示类型为 t_k 的功能单元数的下限，N_{t_k} 表示由 t_k 类功能单元计算的节点数，T_{i_k} 是计算时间，P_{t_k} 是流水级数，T 是可能的迭代周期。这一下限假定DFG中不存在迭代内优先约束。MARS对调度过程中用到的这些功能单元进行分配。这时，MARS开始对资源冲突

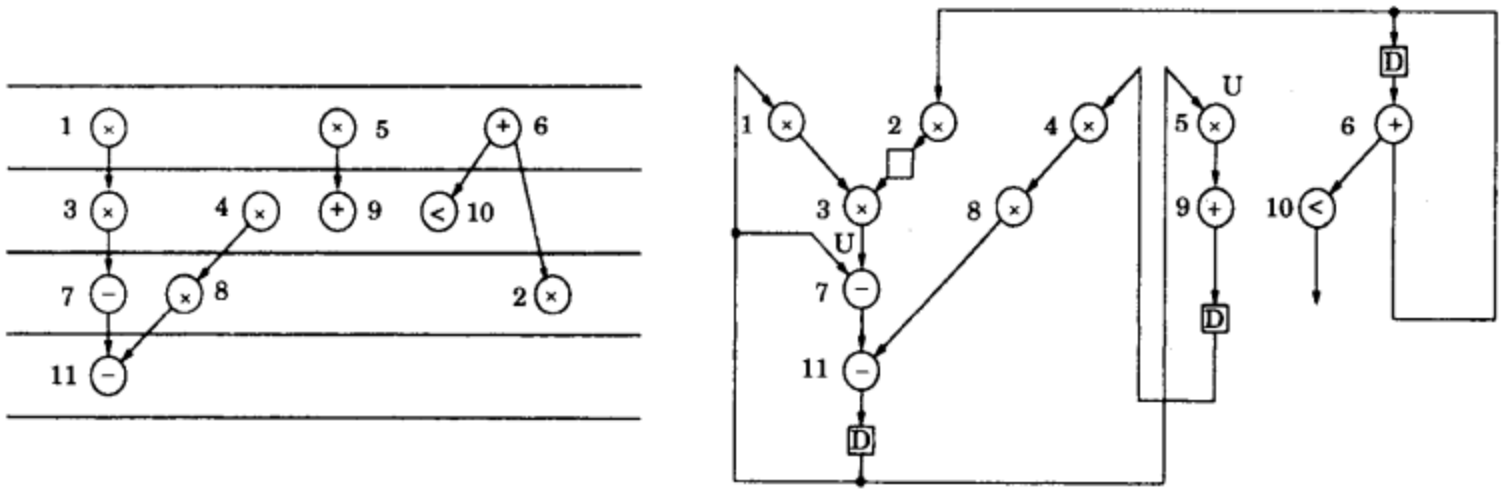
（即调度的节点数比所分配的功能单元数多的那些时间步）迭代性地搜寻初始调度表和尝试通过重新调度节点到不同的时间步来解决资源冲突。在选择和重新调度节点时，MARS利用了迭代间优先约束以获得更有效的调度表。通过首先重新解决面积占用很大的处理器冲突，MARS在更加耗费资源的功能单元中使用了迭代间优先约束以便对任何额外的资源分配需求进行最小化。如果资源冲突不能通过重新调度冲突节点解决，MARS就分配另一个功能单元。

继续2阶微分方程的例子。MARS查找到三个环路，而这些环路对调度递归节点是需要的。接下来MARS确定乘法器数和ALU数的下限如下：

$$\text{乘法器:}\quad M_{mult} = \left\lceil \frac{6\times1}{1\times4} \right\rceil = 2$$
$$ALU:\quad M_{alu} = \left\lceil \frac{5\times1}{1\times4} \right\rceil = 2$$

735
~
736

通过对该功能单元集的分配，MARS产生的最终调度表如图B-13a所示。图B-13b给出了从最终调度表中导出的经过重定时和流水线处理的DFG图。



a) 由MARS产生的2阶微分方程的最终调度表　　b) 由最终调度表导出的重定时和流水线DFG

图 B-13

为说明隐含变换（implicit transformations）的作用，再次考虑图B-11给出的双二次滤波器。首先MARS查找DFG中的两个环路，而这两个环路对调度递归节点都是必需的。接下来，MARS确定乘法器数和加法器数的下限如下：

$$\text{乘法器:}\quad M_{mult} = \left\lceil \frac{4\times1}{1\times4} \right\rceil = 1$$
$$\text{加法器:}\quad M_{add} = \left\lceil \frac{4\times1}{1\times4} \right\rceil = 1$$

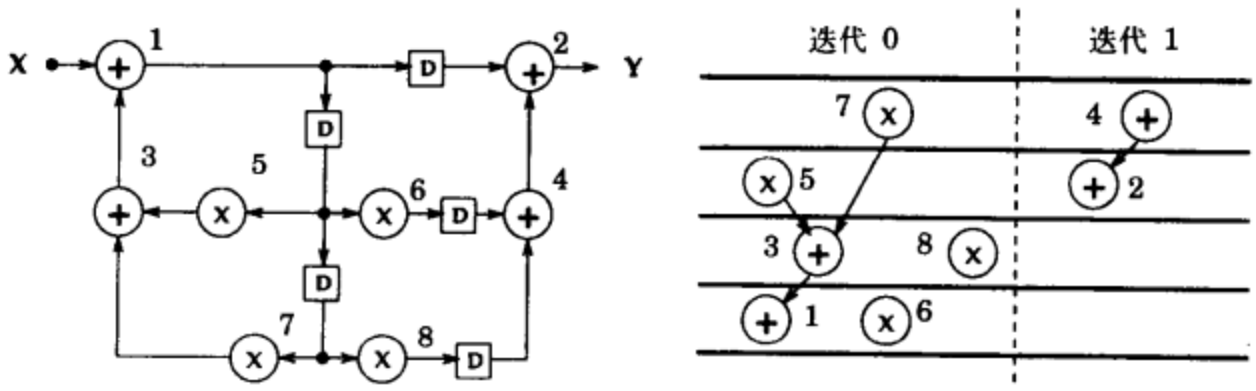
分配一个乘法器一个加法器之后，MARS能够对该双二次滤波器产生最优调度表，如图B-14a所示。图B-14b给出了从最后的调度表中推导出的经过重定时和流水线处理之后的DFG图。注意，在迭代周期4u.t.之下，这一调度是最优的，所需处理器数也得到了降低。与关键路径调度中需要2个乘法器和2个加法器相比，这里只需要1个乘法器和1个加法器。关键路径调度法还需要做一次流水处理才能达到与此相同的结果。

B.3.3 其他启发式调度算法

目前也已提出了其他一些启发式调度算法（heuristic scheduling algorithm），一些新的改进也正在研究之中。其中有一些是从软件设计技术中得来的[7]，另一些则基于一般的迭代方法，例如模拟退火（simulated annealing）。还有一种基于变换的技术是在已有的调度

表中反复应用重定时变换以获得更加有效的调度表[17]。遗传算法也已经被应用到调度问题。对更优的结果，启发式调度技术必须使用DFG中节点的完整调度域，如MARS系统和[18]中那样，以使能够对迭代间优先约束和迭代内优先约束都能得到利用。由此得到的主要结论是，因为高层次综合是一个很困难的问题，有很多启发式方法可以用来在较少时间内得到好的结果。从运行时间上说，大部分方法能产生类似结果。然而，启发式技术却不能保证解是最优的。

737



a) 从最终调度表中导出的简单双二次滤波器的流水线DFG

b) 由MARS产生的使用隐性流水的最优调度表

图 B-14

B.4 整数线性规划模型

整数线性规划（ILP）模型在描述和解决调度问题时能提供一种最佳的形式化方法。这类模型在描述方式和它们的目标函数之间是根本不同的。虽然ILP模型在考虑调度问题时提供这种形式化方法，但是直到最近它们才得到了应用，因为它们需要较长的执行时间，且当约束条件松弛时，执行时间以指数方式增长。目前能够减少执行时间的新模型已经被提出[13]~[15]。ILP模型吸引人的原因是它们能产生最优解并且容易修改以满足新增加的约束条件。

高层次综合中ILP模型的一个一般性描述是，它由一个需要最小化（或最大化）的目标函数和一组需要满足的约束条件组成。很多方法也使用了调度域以减小设计空间和执行时间。在构造ILP模型时，整数变量用来描述目标函数和约束条件。初始变量都是二脚标变量 x_{ij}，它表示了节点 i 是否被调度到时间步 j。其他重要变量是 M_{t_k}，它表示被分配的 t_k 型功能单元的数量。t_k 型功能单元的成本通过常数 c_{t_k} 表示。应用这些变量和常数，调度问题可以设定为对所有功能单元的总成本进行最小化:

738

$$\sum_{k=1}^{m}(c_{t_k} * M_{t_k})$$

其中 m 表示能够用来构造最终结构的功能单元的不同类型数。这一目标函数还要满足设计约束。第一项约束是，没有调度表具有这样一个时间步，即它所包含的操作数比功能单元数 M_{t_k} 还多（即没有资源冲突）:

$$\sum_{o_i \in FU_{t_k}} x_{i,j} - M_{t_k} \leqslant 0, \qquad 当\ 1 \leqslant j \leqslant T, \quad 1 \leqslant k \leqslant m$$

这里T表示最终结构中期望的迭代周期，o_k 是DFG中能够被一个 t_k 型功能单元在一次可能的操作中被处理的节点。FU_{t_k} 表示能够计算操作 o_i 的不同的功能单元类型，$x_{i,j}$ 表示与 o_i 相联

系的二脚标变量（如果 o_i 被调度到时间步 j，则 $x_{i,j}=1$，否则 $x_{i,j}=0$）。

下一个约束条件是 o_i 只能在由 S_i 和 L_i 限定的调度域内被调度（$S_i \leqslant j \leqslant L_i$ 由ASAP调度表和ALAP调度表确定）且只能在调度表中出现一次。

$$\sum_{j=S_i}^{L_i} x_{i,j} = 1, \quad 当 \quad 1 \leqslant i \leqslant n$$

最后一种约束要确保迭代内优先约束被保持。

$$\sum_{j=S_i}^{L_i} (j * x_{i,j}) - \sum_{j=S_i}^{L_i} (j * x_{k,j}) \leqslant -1, \quad \forall \ o_i \rightarrow o_k$$

如果把这种简单的模型应用到如图B-2所示2阶微分方程的例子，则可得到如下的能够通过手工求解（对小的问题），或者能通过一个求解ILP的程序求解（如GAMS[19]）的一组不等式:

$$Assume \quad c_{mult} = 5\ u.t. \quad 和 \quad c_{alu} = 1\ u.t.$$

$$Minimize\ (5 * M_{mult} + 1 * M_{alu})$$

739

无资源冲突约束条件:

$$\begin{aligned} Multiplier: \quad & x_{1,1} + x_{2,1} + x_{4,1} + x_{5,1} - M_{mult} \leqslant 0 \\ & x_{3,2} + x_{4,2} + x_{8,2} + x_{5,2} - M_{mult} \leqslant 0 \\ & x_{8,3} + x_{5,3} - M_{mult} \leqslant 0 \end{aligned}$$

$$\begin{aligned} ALU: \quad & x_{6,1} - M_{alu} \leqslant 0 \\ & x_{9,2} + x_{6,2} + x_{10,2} - M_{alu} \leqslant 0 \\ & x_{7,3} + x_{9,3} + x_{6,3} + x_{10,3} - M_{alu} \leqslant 0 \\ & x_{11,4} + x_{9,4} + x_{10,4} - M_{alu} \leqslant 0 \end{aligned}$$

每个DFG中的节点只能在调度表中出现一次。这一情况可描述成如下的约束条件:

$$\begin{aligned} x_{1,1} = 1,\ x_{2,1} = 1,\ x_{3,2} = 1,\ x_{7,3} = 1,\ x_{11,4} = 1 \\ x_{4,1} + x_{4,2} = 1,\ x_{8,2} + x_{8,3} = 1 \\ x_{5,1} + x_{5,2} + x_{5,3} = 1,\ x_{9,2} + x_{9,3} + x_{9,4} = 1 \\ x_{6,1} + x_{6,2} + x_{6,3} = 1,\ x_{10,2} + x_{10,3} + x_{10,4} = 1 \end{aligned}$$

DFG的迭代内优先约束条件是:

$$\begin{aligned} x_{4,1} + 2x_{4,2} - 2x_{8,2} - 3x_{8,3} \leqslant -1 \\ x_{5,1} + 2x_{5,2} + 3x_{5,3} - 2x_{9,2} - 3x_{9,3} - 4x_{9,4} \leqslant -1 \\ x_{6,1} + 2x_{6,2} + 3x_{6,3} - 2x_{10,2} - 3x_{10,3} - 4x_{10,4} \leqslant -1 \end{aligned}$$

这些不等式组能够产生一个类似于图B-10所示的调度表，具体如下:

$$\begin{aligned} x_{1,1} = 1,\ x_{2,1} = 1,\ x_{3,2} = 1,\ x_{4,2} = 1 \\ x_{5,3} = 1,\ x_{6,1} = 1,\ x_{7,3} = 1,\ x_{8,3} = 1 \\ x_{9,4} = 1,\ x_{10,2} = 1,\ x_{11,4} = 1 \end{aligned}$$

ILP模型潜在的优越性在于它容易重新定义问题和添加新的约束条件。其他模型也具有足够的健壮性使其对一致性环境和不一致性环境都能产生调度表。这里一个不一致性环境被定义为这样一个系统，其中的每一个处理器的数据格式可能会不同。例如，有些处理器是位并行的，而另一些可能是位串行的，还有其他一些可能是字串行的且字的大小还可能有变化。这些模型的目标性函数是要最小化总面积，其中包括数据格式变换器和功能单元的自动分配以及寄存器的最小化等。

740

参考文献

1. M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, vol. 78, no. 2, pp. 301–318, Feb. 1990.
2. R. Camposano and W. Wolf, eds., *High Level VLSI Synthesis*. Kluwer, 1991.
3. T. C. Hu, "Parallel sequencing and assembly line problems," *Operations Research*, no. 9, pp. 841–848, 1961.
4. N. Park and A. C. Parker, "Sehwa: a software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. on Computer-Aided Design*, vol. 7, no. 3, pp. 356–370, March 1988.
5. P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of asic's," *IEEE Trans. on Computer-Aided Design*, vol. 8, no. 6, pp. 661–679, June 1989.
6. H. De Man, J. Rabaey, P. Six, and L. J. Claesen, "Cathedral-II: A silicon compiler for digital signal processing," *IEEE Design and Test*, vol. 3, no. 6, pp. 13–25, Dec. 1986.
7. G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man, "An efficient microcode compiler for application specific DSP processors," *IEEE Trans. on Computer-Aided Design*, vol. 9, no. 9, pp. 925–937, Sept. 1990.
8. J. Rabaey et al., "Fast prototyping of datapath-intensive architectures," *IEEE Design and Test*, vol. 8, no. 3, pp. 40–51, June 1991.
9. P. R. Gelabert and T. P. Barnwell III, "Optimal automatic periodic multiprocessor scheduler for fully specified flow graphs," *IEEE Trans. on Computer-Aided Design*, pp. 858–888, Feb. 1993.
10. T. F. Lee et al., "A transformation-based method for loop folding," *IEEE Trans. on Computer-Aided Design*, no. 4, pp. 439–450, April 1994.
11. C.-Y. Wang and K. K. Parhi, "Resource-constrained loop list scheduler for DSP algorithms," *Journal of VLSI Signal Processing*, vol. 11, Oct. 1995.
12. C.-Y. Wang and K. K. Parhi, "High-level synthesis using concurrent transformations, scheduling, and allocation," *IEEE Trans. on Computer-Aided Design*, vol. 14, no. 3, March 1995.
13. C. H. Gebotys and M. Elmasry, "Global optimization approach for architecture synthesis," *IEEE Trans. on Computer-Aided Design*, vol. 12, pp. 1266–1278, Sept. 1993.
14. C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high-level synthesis," *IEEE Trans. on Computer-Aided Design*, vol. 10, no. 4, pp. 464–475, April 1991.

741

15. K. Ito, L. E. Lucke, and K. K. Parhi, "ILP based cost-optimal DSP synthesis with module selection and data format conversion," *IEEE Trans. on VLSI Systems*, vol. 6, no. 4, Dec. 1998.

16. M. R. Garey and D. S. Johnson, *Computers and Intractability*. Freeman and Co., 1979.

17. L.-F. Chao, A. LaPaugh, and E. Sha, "Rotation scheduling: a loop pipelining algorithm," *IEEE Trans. on Computer-Aided Design*, no. 3, pp. 229–239, March 1997.

18. S. M. Heemstra de Groot, S. Gerez, and O. Herrmann, "Range-chart-guided iterative data-flow-graph scheduling," *IEEE Trans. on Circuits and Systems-I: Fundamental Theory and Application*, vol. 39, no. 5, pp. 351–364, May 1992.

19. A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A User's Guide, Release 2.25*. The Scientific Press, 1992.

742

附录C 欧几里德最大公因子算法

C.1 引言

欧几里德最大公因子（greatest common divisor，GCD）算法有两种形式。其中，一种是关于整数的，另一种是关于多项式的。本附录对两种形式的算法都进行介绍。

C.2 整数欧几里德GCD算法

定理C.2.1（除法算法） 对任何一对整数 c 和 d（其中 d 非零），存在惟一一对整数Q（商）和 r（余数）使得 $c = dQ + r$，其中 $0 \leq |r| < |d|$。

利用除法算法，可以求出两个整数的GCD。

例C.2.1 对$GCD(993,186)$，可计算如下：

$$\begin{aligned} 993 &= 5 \times 186 + 63 \\ 186 &= 2 \times 63 + 60 \\ 63 &= 1 \times 60 + 3 \\ 60 &= 20 \times 3 + 0 \end{aligned} \tag{C-1}$$

743

因为 $GCD(993, 186)$ 整除993和186，它一定整除第一个商63。因为它整除186和63，它也一定整除第二个余数60。因为它整除63和60，故它整除3。另一方面，3整除60，从而整除63，也整除186和993。因此，$GCD(993, 186)=3$。 ■

下面是整数欧几里德GCD算法。

定理C.2.2（欧几里德GCD算法） 给定两个正整数 s 和 r。它们的公因子能够利用除法算法，通过辗转方法计算出。假定 $r < s$，则算法的辗转迭代过程如下：

$$\begin{aligned} s &= Q_1 r + r_1 \\ r &= Q_2 r_1 + r_2 \\ r_1 &= Q_3 r_2 + r_3 \\ &\cdots \\ r_{n-2} &= Q_n r_{n-1} + r_n \\ r_{n-1} &= Q_{n+1} r_n \end{aligned} \tag{C-2}$$

并且这一过程终止于余数为零的时候，最后的非零数 r_n 就是它们的最大公因子。

推论C.2.1 对任何正整数 s 和 r，存在整数 a 和 b 使得

$$GCD(s,r) = as + br \tag{C-3}$$

整数 a 和 b 能够反推辗转公式(C-3)得到，即重写 $r_i = r_{i-2} - Q_i\, r_{i-1}$ 并按如下过程做反推代换：

$$\begin{aligned} GCD(s,r) &= r_n \\ &= r_{n-2} - Q_n r_{n-1} \\ &= r_{n-2} - Q_n(r_{n-3} - Q_{n-1}r_{n-2}) \\ &= (Q_nQ_{n-1}+1)r_{n-2} + (-Q_n)r_{n-3} \\ &= \cdots \end{aligned} \tag{C-4}$$

直到得到如公式（C-3）那样的 s 和 r 的表达式为止。

例C.2.2 通过933和186表示 $GCD(993, 186)$。

744

$$\begin{aligned} GCD(993,186) &= 3 \\ &= 63 - 1 \times 60 \\ &= 63 - 1 \times (186 - 2 \times 63) \\ &= 3 \times 63 - 1 \times 186 \\ &= 3 \times (993 - 5 \times 186) - 1 \times 186 \\ &= 3 \times 993 - 16 \times 186 \end{aligned} \tag{C-5}$$

因此得出对 $a=3$，$b=-16$，$GCD(993,186)$ 能够用公式（C-3）表示为：

$$GCD(993,186) = 3 = a \times 993 + b \times 186 \tag{C-6}$$ ■

这就完成了两个整数GCD的计算。对两个多项式GCD的计算，叙述如下。

C.3 多项式欧几里德GCD算法

定理C.3.1（多项式除法） 对两个多项式 $c(x)$ 和 $d(x)$，其中 $d(x)$ 不等于零，存在惟一一对多项式 $Q(x)$（商多项式）和 $r(x)$（余多项式），使得

$$c(x) = Q(x)d(x) + r(x) \tag{C-7}$$

和 $\deg r(x) < \deg d(x)$。

定理C.3.2（多项式欧几里德GCD算法） 给定一确定域上的两个多项式 $s(x)$ 和 $r(x)$，则应用辗转迭代法可以计算出它们的最大公因式。假如 $deg\ s(x) \geqslant deg\ r(x) \geqslant 0$，则辗转计算过程如下：

$$\begin{aligned} s(x) &= Q_1(x)r(x) + r_1(x) \\ r(x) &= Q_2(x)r_1(x) + r_2(x) \\ r_1(x) &= Q_3(x)r_2(x) + r_3(x), \\ &\cdots \\ r_{n-2}(x) &= Q_n(x)r_{n-1}(x) + r_n(x) \\ r_{n-1}(x) &= Q_{n+1}(x)r_n(x) \end{aligned} \tag{C-8}$$

并且当得到一个余多项式是零的式子时，这一过程停止。这时

$$GCD(s(x), r(x)) = \alpha r_n(x) \tag{C-9}$$

745 其中 α 是域中一元素。

推论C.3.1 对任何多项式 $s(x)$ 和 $r(x)$，存在多项式 $a(x)$ 和 $b(x)$ 使得

$$GCD(s(x), r(x)) = a(x)s(x) + b(x)r(x) \tag{C-10}$$

计算多项式 $a(x)$ 和 $b(x)$ 的过程和整数情况时的过程完全相同。

例C.3.1 给定 $s(x) = x^3 - x^2 + x-1$ 和 $r(x) = x^2 - 2x+1$，计算 $GCD(s(x), r(x))$。将 $GCD(s(x), r(x))$ 用 $s(x)$ 和 $r(x)$ 按公式（C-10）表示出。

从欧几里德算法，可得

$$\begin{aligned} s(x) &= (x+1)r(x) + (2x-2) \\ r(x) &= (\frac{1}{2}x - \frac{1}{2})(2x-2) \end{aligned} \tag{C-11}$$

因此，$GCD(s(x), r(x)) = 2(x-1)$。进行反推代换直到公式（C-12），可得

$$GCD(s(x), r(x)) = s(x) - (x+1)r(x) \tag{C-12}$$

对应的 $a(x) = 1$ 和 $b(x) = -(x+1)$。 ■ 746

附录D Schur多项式的正规直交性

D.1 Schur多项式的直交性

根据公式（12-21），

$$\langle \Phi_N(z), \Phi_i(z) \rangle = 0, \quad 当\ 0 \leqslant i \leqslant N-1 \tag{D-1}$$

这是因为 $0 \leqslant i \leqslant N-1$ 时，每一个 $\Phi_i(z)$ 的 z^N 项的系数为零。

接下来，应用公式（12-4）、(12-26)、(12-27)、(12-21)、(12-22）和（12-37），对内积 $\langle \Phi_{N-1}(z), \Phi_{N-2}(z) \rangle$，可计算如下：

$$\begin{aligned}
&\langle \Phi_{N-1}(z), \Phi_{N-2}(z) \rangle \\
&= \langle \frac{z^{-1}\{\Phi_N(z) - k_N \Phi_N^*(z)\}}{\sqrt{1-k_N^2}}, \frac{z^{-1}\{\Phi_{N-1}(z) - k_{N-1}\Phi_{N-1}^*(z)\}}{\sqrt{1-k_{N-1}^2}} \rangle \\
&= \langle \frac{\Phi_N(z) - k_N \Phi_N^*(z)}{\sqrt{1-k_N^2}}, \frac{\Phi_{N-1}(z) - k_{N-1}\Phi_{N-1}^*(z)}{\sqrt{1-k_{N-1}^2}} \rangle \\
&= \frac{-k_N}{\sqrt{(1-k_N^2)(1-k_{N-1}^2)}} \langle \Phi_N^*(z), \Phi_{N-1}(z) - k_{N-1}\Phi_{N-1}^*(z) \rangle \\
&= \frac{-k_N}{\sqrt{(1-k_N^2)(1-k_{N-1}^2)}} \frac{1}{\phi_N} \{\Phi_{N-1}(0) - k_{N-1}\Phi_{N-1}^*(0)\} \\
&= 0
\end{aligned}$$

这一推导过程可以推广到内积

$$\langle \Phi_i(z), \Phi_j(z) \rangle, \quad 当\ 0 \leqslant j \leqslant i < N \tag{D-2}$$

注意，任何 $\Phi_i(z)$ 都可以如例D.1.1中所示的那样，能表示成 $\Phi_N(z)$ 和 $\Phi^*_N(z)$ 的一个函数，这里符号 $f(a, b, c, d, \cdots)$ 只表示该函数与参数 a、b、c、d 等有关而不代表任何特别的函数。

例D.1.1

$$\begin{aligned}
\Phi_{N-1}(z) &= z^{-1} f(\Phi_N(z), \Phi_N^*(z)) \\
\Phi_{N-1}^*(z) &= z^{N-1}\Phi_{N-1}(z^{-1}) \\
&= f(\Phi_N(z), \Phi_N^*(z)) \\
\Phi_{N-2}(z) &= z^{-1} f(\Phi_{N-1}(z), \Phi_{N-1}^*(z)) \\
&= z^{-1} f(\Phi_N(z), \Phi_N^*(z), z^{-1}\Phi_N(z), z^{-1}\Phi_N^*(z)) \\
\Phi_{N-2}^*(z) &= z^{N-2}\Phi_{N-2}(z^{-1}) \\
&= f(\Phi_N(z), \Phi_N^*(z), z^{-1}\Phi_N(z), z^{-1}\Phi_N^*(z))
\end{aligned}$$

■

由上面的例子，显然对 $0 \leqslant i \leqslant N$，$\Phi_i(z)$ 可表示为

$$\Phi_i(z) = z^{-1} f(\Phi_N(z), \Phi_N^*(z), \cdots, z^{-(N-i-1)}\Phi_N(z), z^{-(N-i-1)}\Phi_N^*(z)) \tag{D-3}$$

而且，$\Phi_j(z)$ 可表示为

$$\Phi_j(z) = \frac{z^{-1}\{\Phi_{j+1}(z) - k_{j+1}\Phi_{j+1}^*(z)\}}{\sqrt{1-k_{j+1}^2}} \tag{D-4}$$

现在，应用公式（D-3）和（D-4），公式（D-2）中的内积变为

$$\langle \Phi_i(z), \Phi_j(z)\rangle = \langle P(z), \frac{\Phi_{j+1}(z) - k_{j+1}\Phi_{j+1}^*(z)}{\sqrt{1-k_{j+1}^2}}\rangle \tag{D-5}$$

这里

748

$$P(z) = f(\Phi_N(z), \Phi_N^*(z), \cdots, z^{-(N-i-1)}\Phi_N(z), z^{-(N-i-1)}\Phi_N^*(z)) \tag{D-6}$$

现在我们首先考虑 $P(z)$ 中 $z^{-m}\Phi_N(z)$ 的如下内积

$$\begin{aligned} &\langle z^{-m}\Phi_N(z), \frac{\Phi_{j+1}(z) - k_{j+1}\Phi_{j+1}^*(z)}{\sqrt{1-k_{j+1}^2}}\rangle \\ = \ &\langle \Phi_N(z), \frac{z^m\Phi_{j+1}(z) - k_{j+1}z^m\Phi_{j+1}^*(z)}{\sqrt{1-k_{j+1}^2}}\rangle \end{aligned} \tag{D-7}$$

由公式（12-21），当 $z^m\Phi_{j+1}(z)$ 的阶，亦即 $m+j+1$ 小于 N 时，上述内积变为零。因此，$m+j+1$ 的最大值是 $N-(i-j)$。从公式（D-2）中关于 i 和 j 的条件可见，这一最大值小于N。所以，当 $m=0$ 到 $N-i-1$ 时，可以将 $P(z)$ 中的项 $z^{-m}\Phi_N(z)$ 去掉而不会影响到内积。

接下来考虑 $P(z)$ 中 $z^{-m}\Phi^*_N(z)$ 的内积如下

$$\begin{aligned} &\langle z^{-m}\Phi_N^*(z), \frac{\Phi_{j+1}(z) - k_{j+1}\Phi_{j+1}^*(z)}{\sqrt{1-k_{j+1}^2}}\rangle \\ = \ &\langle \Phi_N^*(z), \frac{z^m\Phi_{j+1}(z) - k_{j+1}z^m\Phi_{j+1}^*(z)}{\sqrt{1-k_{j+1}^2}}\rangle \end{aligned} \tag{D-8}$$

当 $m=0$ 时上述内积为零，因为

$$\frac{\Phi_{j+1}(0) - k_{j+1}\Phi_{j+1}^*(0)}{\sqrt{1-k_{j+1}^2}} = 0 \tag{D-9}$$

当 $m>0$ 时，公式（D-8）中的内积显然为零，这是因为 $z^m\Phi_{j+1}(z)$ 和 $z^m\Phi^*_{j+1}(z)$ 的常数项等于零。

从以上讨论得出

$$\langle \Phi_i(z), \Phi_j(z)\rangle = 0, \quad 当 \quad 0 \leqslant j,\ i \leqslant N \tag{D-10}$$

这表明Schur多项式 $\{\Phi_N(z), \Phi_{N-1}(z), \cdots, \Phi_0(z)\}$相互直交。

D.2 Schur多项式的正规直交性

前一节应用内积公式证明了Schur多项式的直交性。因此，为证明Schur多项式的正规直交性，只需再证明 749

$$\langle \Phi_i(z), \Phi_i(z) \rangle = 1, \quad 当\ 0 \leqslant i \leqslant N \tag{D-11}$$

不妨设一个 i 阶Schur多项式表示如下:

$$\Phi_i(z) = \sum_{m=0}^{i} \phi_m^{(i)} z^m \tag{D-12}$$

应用这一表示，公式（12-37）可以重新写为:

$$k_i = \frac{\phi_0^{(i)}}{\phi_i^{(i)}} \tag{D-13}$$

这样，由公式（12-4）和公式（D-13），$\Phi_{N-1}(z)$ 中 z^{N-1} 的系数为

$$\begin{aligned} \phi_{N-1}^{(N-1)} &= \frac{\phi_N^{(N)} - k_N \phi_0^{(N)}}{\sqrt{1-k_N^2}} \\ &= \frac{\phi_N^{(N)}(1-k_N^2)}{\sqrt{1-k_N^2}} \\ &= \phi_N^{(N)} \sqrt{1-k_N^2} \end{aligned} \tag{D-14}$$

同理，$\Phi_i(z)$ 中 z^i 项的系数可表示为

$$\phi_i^{(i)} = \phi_{i+1}^{(i+1)} \sqrt{1-k_{i+1}^2} \tag{D-15}$$

反复应用公式（D-15），

$$\phi_i^{(i)} = \phi_N^{(N)} \sqrt{1-k_N^2} \sqrt{1-k_{N-1}^2} \cdots \sqrt{1-k_{i+1}^2} \tag{D-16}$$

为估计公式（D-11）中的内积，利用公式（D-3）和公式（D-6），$\Phi_i(z)$ 可表示为

$$\langle \Phi_i(z), \Phi_i(z) \rangle = \langle P(z), z\Phi_i(z) \rangle \tag{D-17}$$

因为 $z\Phi_i(z)$ 的常数项为零，所以 $P(z)$ 中Schur多项式的反多项式对上述内积没有任何影响。$P(z)$ 中惟一能够产生非零结果的项是 $z^{-(N-i-1)}\Phi_N(z)$，这是因为

$$\langle z^{-(N-i-1)}\Phi_N(z), z\Phi_i(z) \rangle = \langle \Phi_N(z), z^{N-i}\Phi_i(z) \rangle \tag{D-18}$$

这里 $z^{N-i}\Phi_i(z)$ 的次数是N。

反复应用公式（12-4），可以证明 $P(z)$ 中 $z^{-(N-i-1)}\Phi_N(z)$ 的系数是 $1\Big/\sqrt{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}$。 750
因此，

$$\begin{aligned} \langle \Phi_i(z), \Phi_i(z) \rangle &= \frac{1}{\sqrt{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}} \langle \Phi_N(z), z^{N-i}\Phi_i(z) \rangle \\ &= \frac{1}{\sqrt{(1-k_N^2)(1-k_{N-1}^2)\cdots(1-k_{i+1}^2)}} \frac{\phi_i^{(i)}}{\phi_N^{(N)}} \end{aligned} \tag{D-19}$$

根据公式（D-16），显然上述内积为1，这表明Schur多项式是正规直交的。

应用Schur多项式的正规直交性可以证明:

$$
\begin{aligned}
\langle \Phi_i^*(z), \Phi_i^*(z) \rangle &= \frac{1}{2\pi j} \oint_C \frac{\Phi_i^*(z)\Phi_i^*(z^{-1})}{\Phi_N(z)\Phi_N(z^{-1})z} dz \\
&= \frac{1}{2\pi j} \oint_C \frac{\{z^i \Phi_i(z^{-1})\}\{z^{-i}\Phi_i(z)\}}{\Phi_N(z)\Phi_N(z^{-1})z} dz \\
&= \frac{1}{2\pi j} \oint_C \frac{\Phi_i(z)\Phi_i(z^{-1})}{\Phi_N(z)\Phi_N(z^{-1})z} dz \\
&= \langle \Phi_i(z), \Phi_i(z) \rangle \\
&= 1
\end{aligned}
\tag{D-20}
$$

进一步，

$$
\begin{aligned}
\langle \Phi_{i-1}(z), \Phi_{i-1}(z) \rangle &= \langle \frac{z^{-1}\{\Phi_i(z) - k_i \Phi_i^*(z)\}}{\sqrt{1-k_i^2}}, \frac{z^{-1}\{\Phi_i(z) - k_i \Phi_i^*(z)\}}{\sqrt{1-k_i^2}} \rangle \\
&= \frac{1}{1-k_i^2}\{1 - 2k_i \langle \Phi_i(z), \Phi_i^*(z) \rangle + k_i^2\}
\end{aligned}
\tag{D-21}
$$

从Schur多项式的正规直交性可知，公式（D-21）中的值应为1。因此，可以得出结论

751

$$
\langle \Phi_i(z), \Phi_i^*(z) \rangle = k_i \tag{D-22}
$$

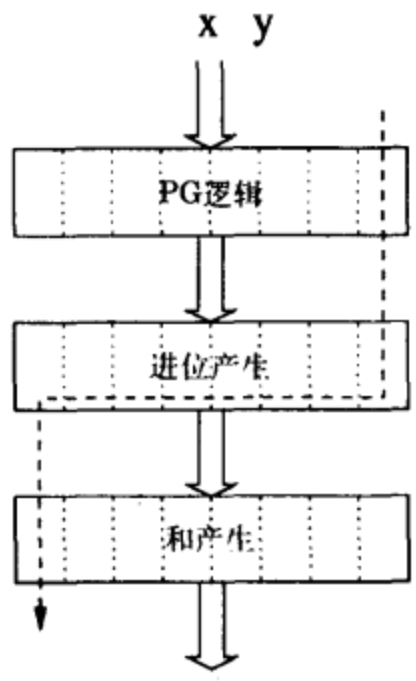
附录E 快速二进制加法器和乘法器

E.1 引言

本附录阐述了快速二进制加法器和乘法器的实现。E.2节给出了基于Manchester、进位选择和超前进位技术的快速加法器的设计。这些加法器与其他书中介绍的加法器的不同之处在于这里仅仅采用多路器来实现。E.3节提出了基于Wallace树和Dadda乘法器的快速乘法器设计。

E.2 基于多路器的快速二进制加法器

本节探讨基于Manchester、进位选择和超前进位的快速二进制加法器，这些加法器仅仅采用多路器来实现。



图E-1 Manchester加法器的方框图

E.2.1 基于多路器的Manchester加法器

在二进制加法中，进位 c_i 既可以在第 i 级产生也可以由前一级传递得到。如果 x_i 和 y_i 都为1，那么在第 i 级产生进位；如果两个操作数 (x_i, y_i) 仅有一个为1，则从第 $i-1$ 级向第 $i+1$ 级传递进位[1]。用 g_i 和 p_i 来表示第 i 级进位的产生和传递，那么可计算为：

$$g_i = x_i \cdot y_i \tag{E-1}$$

$$p_i = x_i \oplus y_i \tag{E-2}$$

这里，⊕代表异或操作。每个阶段的进位 c_{i+1} 及和 s_i 可计算如下：

$$c_{i+1} = g_i + p_i \cdot c_i \tag{E-3}$$

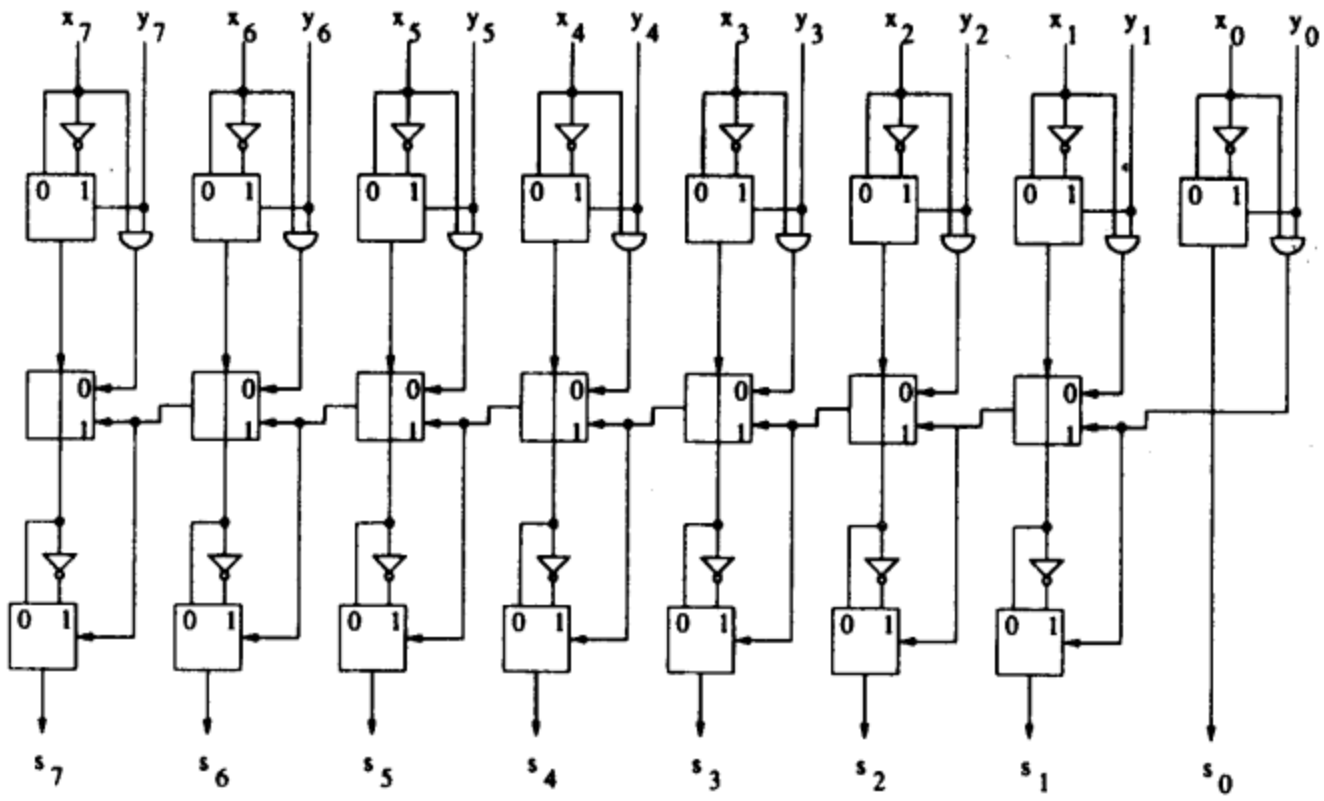
$$s_i = p_i \oplus c_i \tag{E-4}$$

因此基于此原理的加法器包括三个有序阶段:

- PG逻辑: 用公式（E-1）和（E-2）来并行计算所有 (g_i, p_i) 对;
- 进位产生（Carry-generator）: 用公式（E-3）来计算所有的 c_{i+1};
- 和产生（Sum-generator）: 用公式（E-4）来计算所有的 s_i;

此过程如图E-1所示。该加法器关键路径的计算时间（图E-1中的虚线所示）为 $t_{pg} + (W-1)t_{carry} + t_{sum}$，其中 t_{pg}、t_{carry} 和 t_{sum} 分别表示进行一位运算时PG逻辑、进位产生以及和产生所需的计算延迟。进位链的计算延迟 $(W-1)t_{carry}$ 限制了加法器的速度。到此，为了加速进位产生，设计了Manchester加法器。本小节提出了基于多路器的Manchester加法器，这种加法器的进位产生器仅用多路器来实现。

754



图E-2 8位基于多路器的Manchester加法器

从公式(E-1)和公式(E-2)可以看出 g_i 和 p_i 永远不会同时为1。在这种约束条件下，公式(E-3)可写为:

$$c_{i+1} = \overline{p_i} \cdot g_i + p_i \cdot c_i \tag{E-5}$$

此运算只用多路器就可以实现[2]。图E-2为一个8位基于多路器的Manchester加法器。这个加法器的关键路径等于 $9t_{mux}$，这里 t_{mux} 表示一个多路器的计算延迟。

通过采用基于多路器的Manchester加法器可以将二进制加法的计算延迟从 Wt_{FA}(t_{FA}表示一个全加法器延迟)降到了 $(W+1)t_{mux}$，但它仍然与字长W成比例。

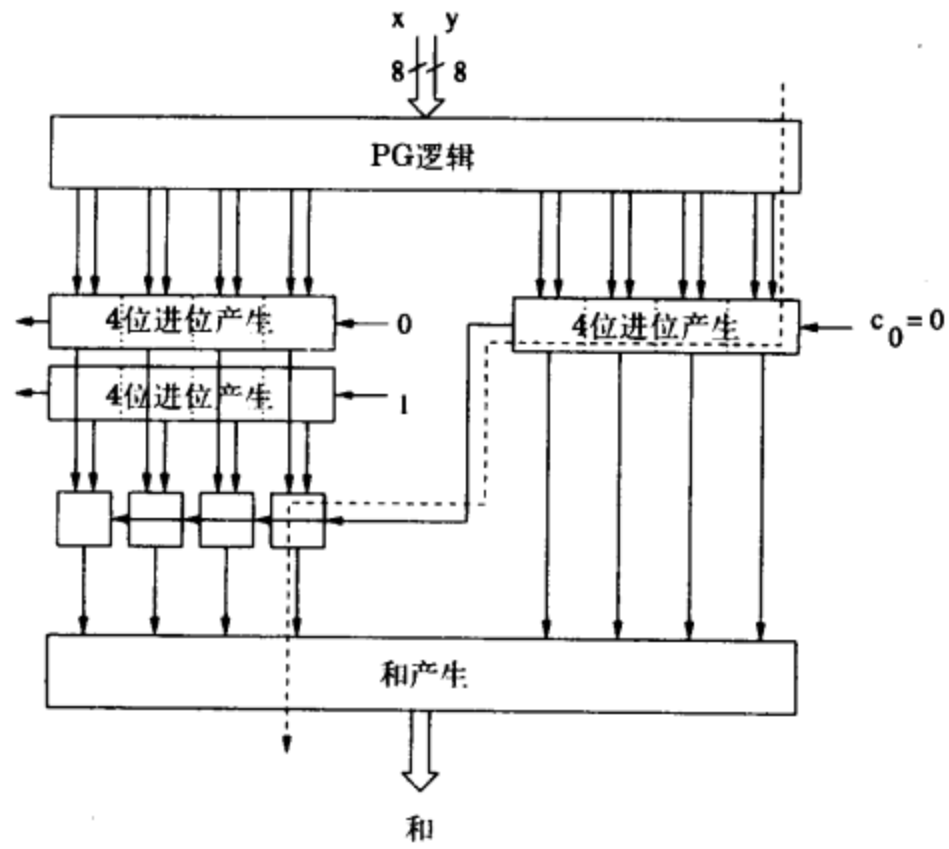
E.2.2 快速二进制加法

快速二进制加法器可基于进位选择（carry-select）或超前进位（也称为二进制超前或基于树的）加法器来设计。

E.2.2.1 进位选择方法

在进位选择加法器中，将字分成多块（每个块的长度不必相同）。对于每一个块，在假定

进位输入为1和0的情况下，产生两组和及进位位。一旦知道来自低位块产生的进位，就可以选择合适的和位[3]。图E-3给出了一个8位基于多路器的进位选择加法器，该加法器的进位产生器（carry-generator）包含了两个块，每个块的长度为L=4。其关键路径或者说等待时间（如图E-3中虚线所示）等于$6t_{mux}$。读者可以验证一下该设计总共需要29个多路器和8个与门。 755



图E-3 8位进位选择加法器

E.2.2.2 基于树的（Tree-Based）方法

基于多路器的快速进位产生器也可以通过采用基于树的二进制超前或者超前进位的概念来设计[4]。

在连续的 (p, g) 对之间定义一个逻辑操作符 o 如下:

$$(g,p)\ o\ (g',p') = (g + p\cdot g',\ p\cdot p') \tag{E-6}$$

注意，这里的 g 和 p 不能同时为1，g' 和 p' 也不能同时为1。因此，操作符 o 可重写如下:

$$(g,p)\ o\ (g',p') = (\overline{p}\cdot g + p\cdot g',\ p\cdot p') \tag{E-7}$$

它可以用一个多路器和一个与门来实现。定义 Γ_i 和 Π_i为:

$$(\Gamma_i,\Pi_i) = \begin{cases} (g_0,p_0) & \text{如果 } i=0, \\ (g_i,p_i)o(\Gamma_{i-1},\Pi_{i-1}) & \text{如果 } 0<i\leqslant W-1 \end{cases} \tag{E-8}$$ 756

这样，我们就得到如下的表达式:

$$(\Gamma_i,\Pi_i) = (g_i,p_i)\,o\,(g_{i-1},p_{i-1})\,o\cdots o\,(g_0,p_0)\,. \tag{E-9}$$

可以证明 $\Gamma_i = c_{i+1}$ 和 $\Pi_i = p_i\ p_{i-1}\cdots p_0$。并且我们可以从中看出操作符$o$是结合的(associative)。因此，对于8位进位产生，公式(E-9)的计算可以按照二叉树的方式计算如下:

(1) 在第一步中，下面的计算是并行进行的:

- $(\underline{\Gamma_0, \Pi_0}) = (g_0, p_0)$ 和 $c_1 = \Gamma_0$

- $(g_1, p_1)\ o\ (\Gamma_0, \Pi_0) = (\underline{\Gamma_1, \Pi_1})$ 和 $c_2 = \Gamma_1$
- $(g_3, p_3)\ o\ (g_2, p_2)$
- $(g_5, p_5)\ o\ (g_4, p_4)$
- $(g_7, p_7)\ o\ (g_6, p_6)$

(2) 第二步计算:

- $[(g_3, p_3)\ o\ (g_2, p_2)]\ o\ (\Gamma_1, \Pi_1) = (\underline{\Gamma_3, \Pi_3})$ 和 $c_4 = \Gamma_3$
- $[(g_7, p_7)\ o\ (g_6, p_6)]\ o\ [(g_5, p_5)\ o\ (g_4, p_4)]$
- $(g_2, p_2)\ o\ (\Gamma_1, \Pi_1) = (\underline{\Gamma_2, \Pi_2})$ 和 $c_3 = \Gamma_2$

(3) 第三步计算:

- $([(g_7, p_7)\ o\ (g_6, p_6)]\ o\ [(g_5, p_5)\ o\ (g_4, p_4)])\ o\ (\Gamma_3, \Pi_3) = (\underline{\Gamma_7, \Pi_7})$ 和 $c_8 = \Gamma_7$
- $[(g_5, p_5)\ o\ (g_4, p_4)]\ o\ (\Gamma_3, \Pi_3) = (\Gamma_5, \Pi_5)$ 和 $c_6 = \Gamma_5$
- $(g_4, p_4)\ o\ (\Gamma_3, \Pi_3) = (\underline{\Gamma_4, \Pi_4})$ 和 $c_5 = \Gamma_4$

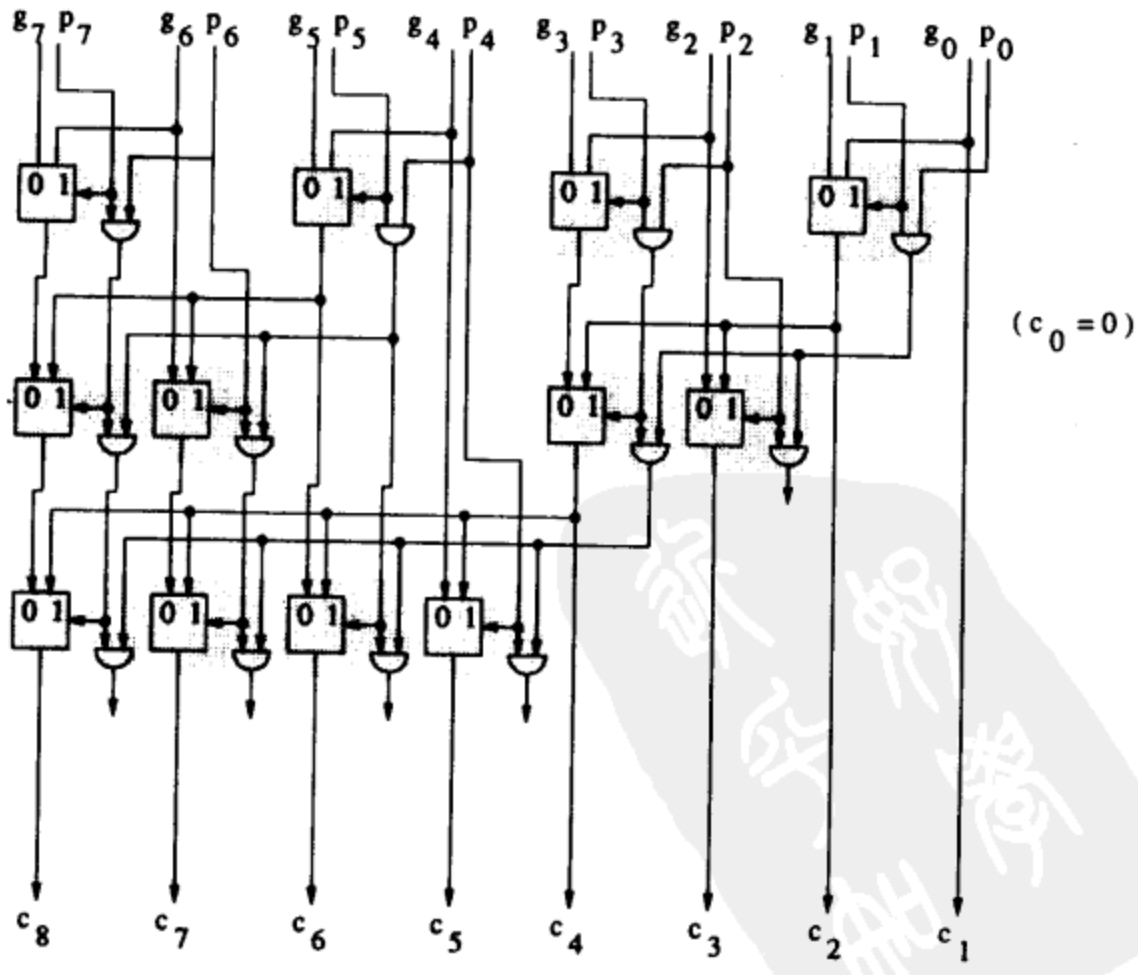
(4) 最后，Γ_6 和 Π_6 可以计算为:

- $(g_6, p_6)\ o\ (\Gamma_5, \Pi_5) = (\underline{\Gamma_6, \Pi_6})$ 和 $c_7 = \Gamma_6$

8位基于树的进位产生器如图E-4所示，o操作符表示为图中高亮的阴影区。该加法器总共需要27个多路器和19个与门，其等待时间为$6t_{mux}$。这样，在二进制超前或基于树的加法器中，进位产生器的等待时间从正比于 Wt_{mux} 降到了正比于$(\log_2)Wt_{mux}$。

可以将进位选择和二进制超前的概念结合起来设计快速、小面积和低功耗的二进制加法器。读者可以参看[2]进一步探讨这一主题。

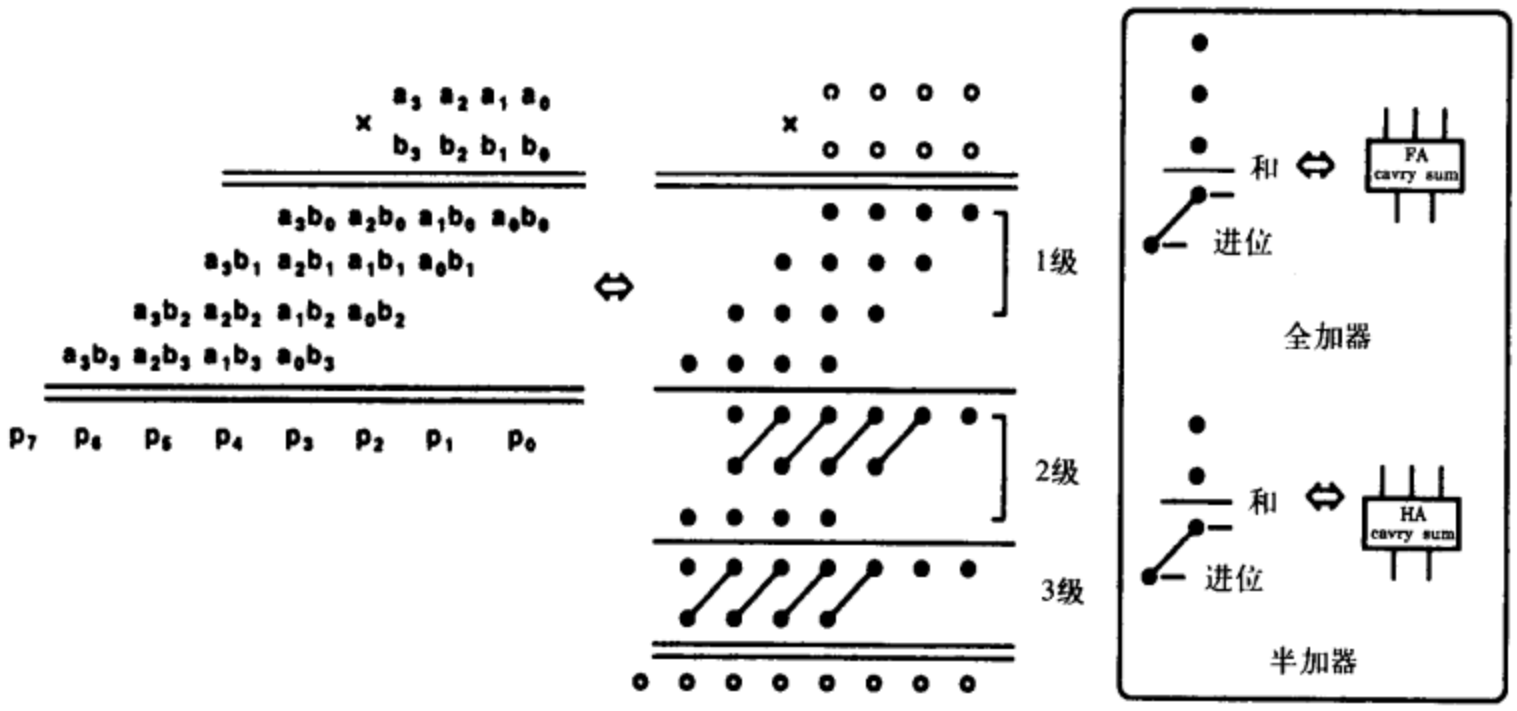
757



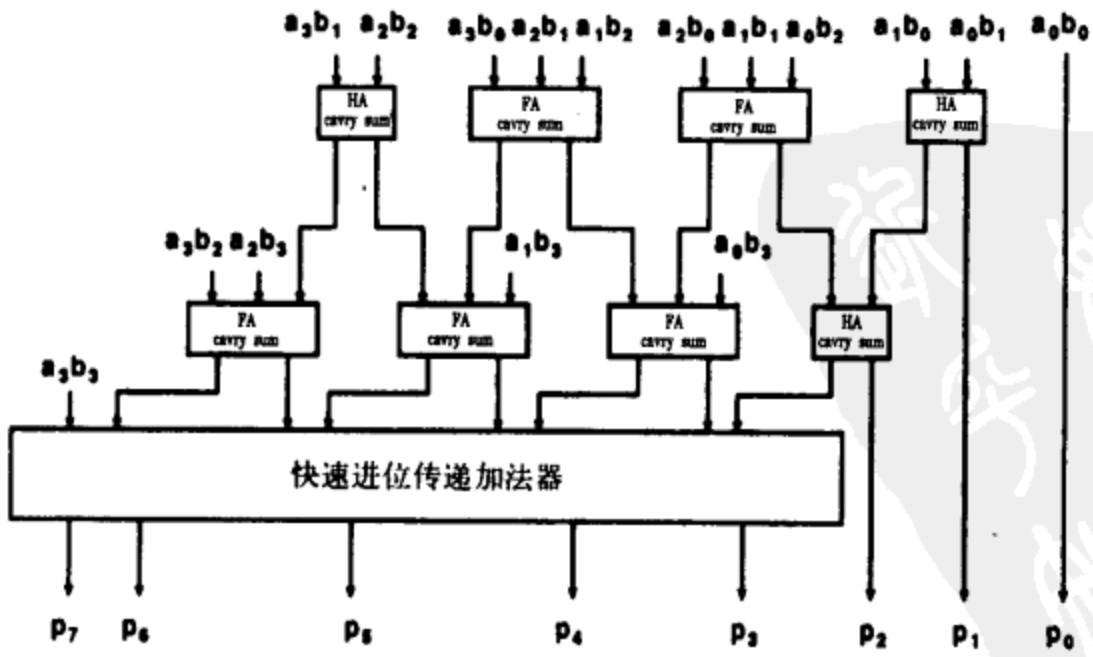
图E-4 8位基于多路器树的二进制超前进位产生

E.3 WALLACE 树乘法器和DADDA乘法器

Wallace树乘法器[5]原理如图E-5a所示。从图中可以清楚地看出总共有16个部分积（在图中为黑点）必须相加。相加过程的第一步是将这些部分积每3个分为一组。例如，如果有 p 行部分积，$3*(\lfloor p/3 \rfloor)$ 行被分组，剩下的p mod 3行传递到下一级。因此，在图E-5的第一级中，3行部分积被分为一组。如果一列中有3个点则采用全加法器对这三行进行求和，如果一列中有2个点则采用半加器对这三行求和。来自全加器与半加器的求和结果和进位信号传递到下一级。结果，这一级恰好有三行部分积进行求和。求和的结果和进位信号被传递到第三级。既然第三级只有两行部分积需要求和，那么在该级就采用一个快速进位传递加法器来实现。4 × 4位的Wallace树乘法器的实现如图E-5b所示。该结构中，仅仅需要5个全加法器和3个半加法器来计算16个部分积的和。



a)Wallace树乘法器的操作



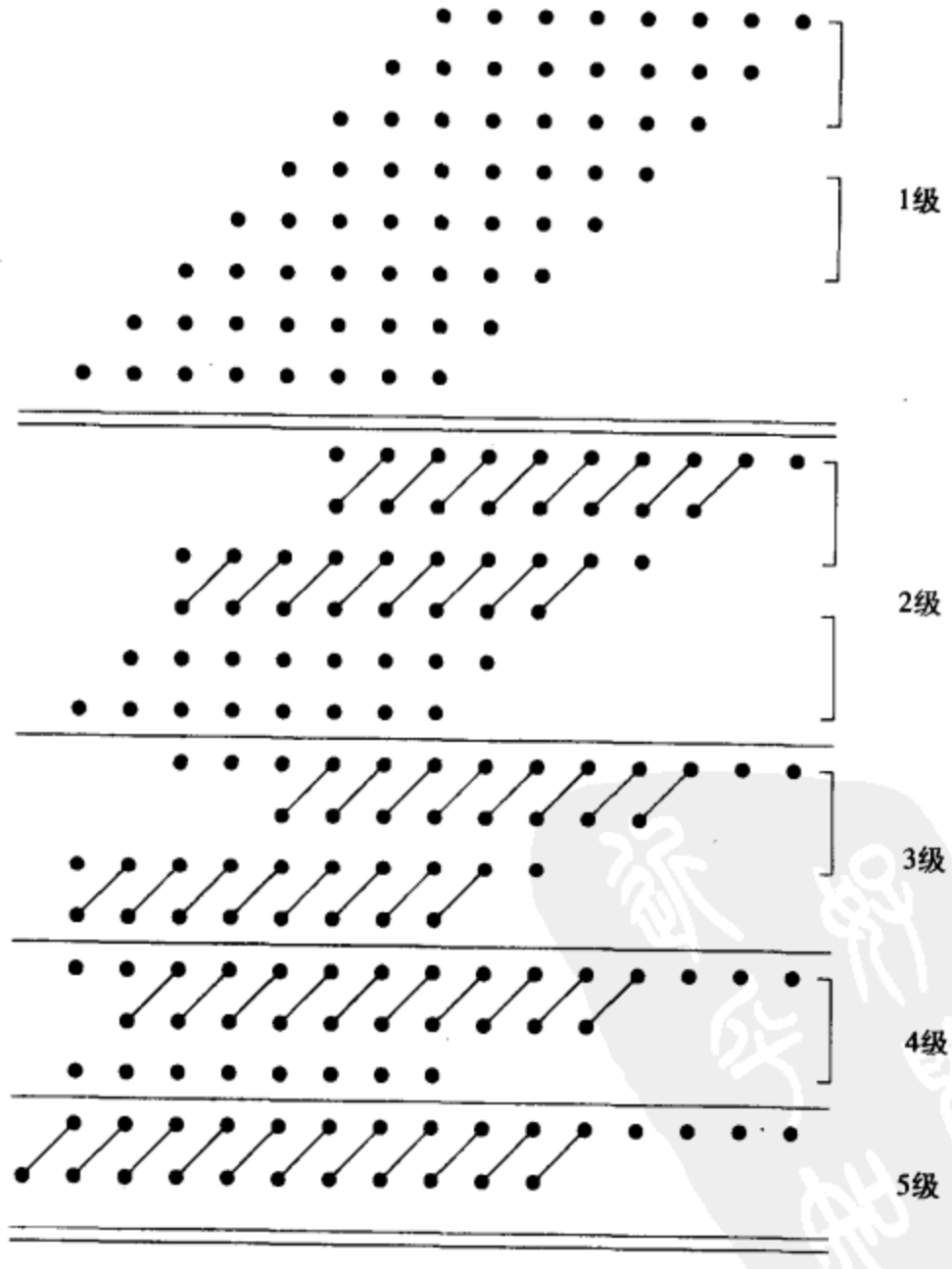
b) 4 × 4Wallace树乘法器的结构. 假设操作数为无符号数

图 E-5

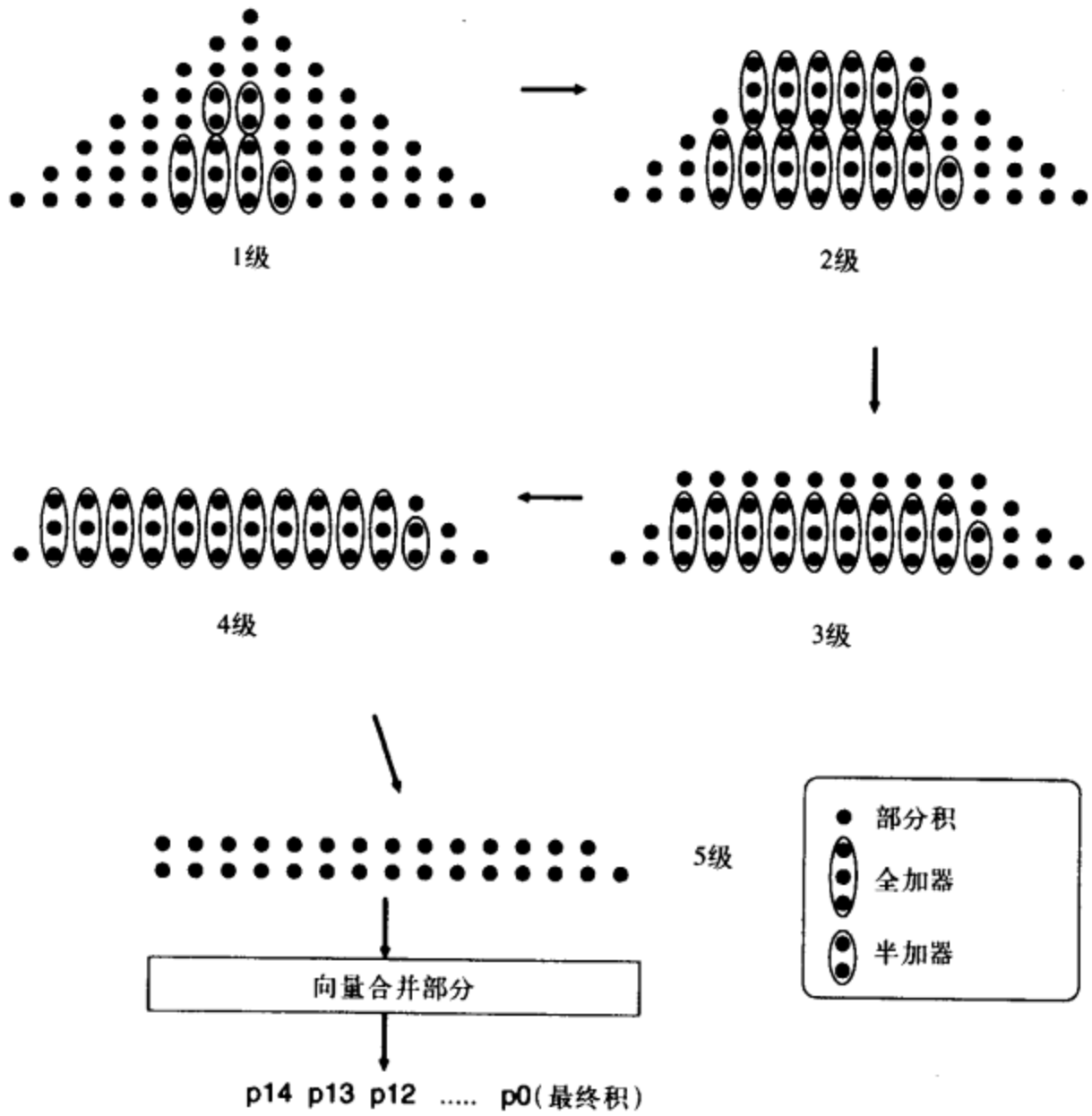
Wallace树乘法器的原理可以很容易地扩展到字长度更长的情况。图E-6示给出了一个8×8位乘法器的操作，这里，部分积在5级中相加。如前所述，第5级使用了快速进位传递加法器。

Wallace树操作的推广见[6]，该文献说明了每级中矩阵高度能够缩小至多1.5倍。推广的乘法器称为Dadda乘法器。在Dadda乘法器中，为了将当前级部分积矩阵的高度减小到下一级的高度，每一级只完成最小数目的部分积行。因此，如果第n级的部分积矩阵的高度为p，那么第$n+1$级的部分积矩阵的最大高度为$\left\lceil\frac{2p}{3}\right\rceil$。例如，如果最初部分积矩阵的高度为8，那么在以后各级中的部分积高度分别为6、4、3、2。图E-7给出了一个8×8位Dadda乘法器。该结构包括5级和一个最终向量合并部分。比较8×8位Dadda乘法器和Wallace树乘法器，可以看出，实现同样的功能，Wallace树乘法器使用38个全加器和15个半加器，而Dadda乘法器仅仅需要35个全加器和7个半加器。但Wallace树乘法器仅需要一个10位的进位传递加法器，而Dadda乘法器需要一个14位的进位传递加法器。Dadda乘法器的另一个缺点是它比Wallace树乘法器更加不规则，使得它在VLSI设计中更加难于布局。

758
~
759



图E-6 8×8位Wallace树乘法器的操作



图E-7 8×8位Dadda乘法器的操作

参考文献

1. A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture, and Implementations.* Prentice Hall International, 1994.
2. K. K. Parhi, "Fast low-power VLSI binary addition," in *Proc. of 1997 IEEE International Conference on Computer Design (ICCD)*, (Austin, Texas), pp. 676–684, Oct. 1997.
3. S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers.* CBS College Publishing, 1982.
4. R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. on Computers*, vol. C-31, no. 3, pp. 260–264, March 1982.
5. C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. on Computers*, vol. EC-13, pp. 14–17, Feb. 1964.
6. L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, 1965.

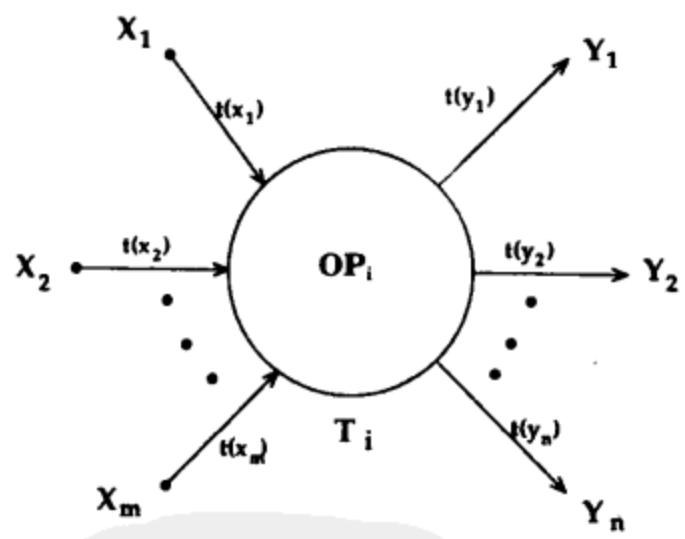
760 ~ 762

附录F 位串行系统中的调度

F.1 引言

位串行架构的综合是高层次综合的一个特例。在位串行系统里，没有执行分配，即每一个算法操作都被映射到一个惟一的位串行功能单元。在位串行架构里，一个完整的字并不是在一个时钟周期里进行计算的，而是在一系列时钟周期里进行计算。因此，位串行架构的调度任务对于保证输入和输出正确同步是非常重要的。本附录提供了一个算法，用于位串行架构的优化资源约束的调度。该算法假设运算到硬件功能单元的映射存在一一对应关系，因此，没有必要进行功能单元的分配。位串行调度算法的目标是在所需同步寄存器数目最小的情况下，将运算分配到不同的时间阶段。

位串行架构经常用图F-1时序模型来表示，用于位串行调度算法。该模型表示了位串行运算器输入和输出的必要相对时序(relative timing)。在图F-1中，功能单元（或运算器）i 位在时间步 T_i 处开始运算，输入 x_1 的第一位会在时间步 $T_i+t(x_1)$出现，输入x_2 在时间步 $T_i+t(x_2)$，…，输入 x_m 在时间步 $T_i+t(x_m)$。运算器 i 在时间步 $T_i+t(y_1)$产生输出 y_1 的第一位，输出 y_2 在时间步 $T_i+t(y_2)$，…，输出 y_m 在时间步 $T_i+t(y_m)$。作为该模型的一个例子，考察图F-2中的位串行结构，从图中可以看出，功能单元M在时间步0被调度，A在时间步4被调度。输入 $y(n-1)$ 和 $x(n)$ 在相对时间0同时到达。M可以立即处理输入 $y(n-1)$ 的第一位，并在4个时间步后产生输出的第一位。就如图中所示的调度时间那样，因为A必须等待M的输出才能开始运算，输入 $x(n)$ 必须延迟4个时间步以保持与输入数据同步。因此，如图所示，在这个例子中位串行调度算法必须在 $x(n)$ 输入插入4个延迟，以保持输入数据的合理同步。



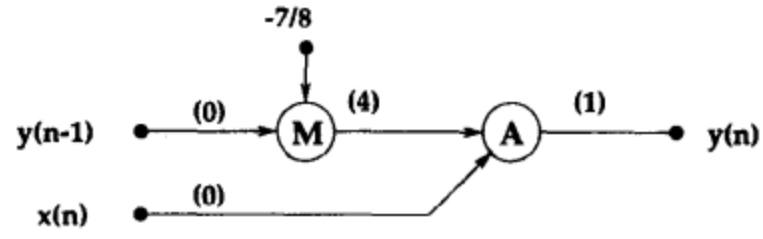
图F-1 通用位串行运算器的时序模型

F.2 调度算法概述

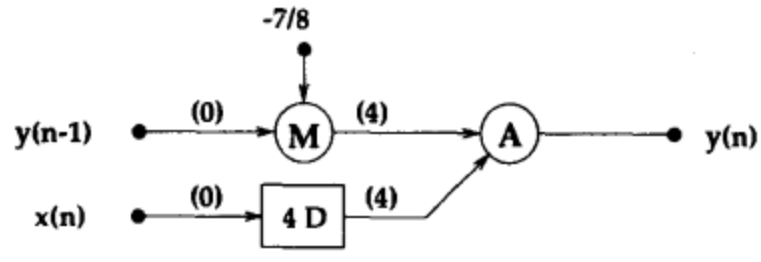
在位串行架构中，位串行调度算法就是为每个运算器分配“调度时间”。该时间代表运算名义上的开始时间。非正式地讲，每个运算器必须等到它的所有输入都到来后才能被调度(即它们已经由之前的运算器产生)。调度问题可以表示为一个线性规划问题，因为所有运算器的调度时间必须是整数，调度算法必须找到该问题的最优整数解。

考虑图F-3中一对用边沿连接的运算器。操作数 $OPND_x$ 是由运算器 OP_i 产生的一个输出，该操作数也是运算器 OP_j 的输入（也可能是其他运算器）。运算器 OP_i 和 OP_j 的调度时

间分别用 T_i 和 T_j 表示。OP_i 和 OP_j 相应的输入输出端口的时序指标分别用 t_i 和 t_j 来表示。



a) 调度之前的DFG



b) 分配了同步寄存器的调度之后的DGF

图F-2 位串行调度问题示例

因为 OP_i 在时间步 T_i 被调度，如图所示，输出 $OPND_x$ 将在时间步 $T_i + t_i$ 得到。进一步，在时间步 $T_j + t_j$ 运算器 OP_j 需要同样的操作数作为输入。因为该操作数在由运算器 OP_i 产生之前不能被运算器 OP_j 所用，就可以得到下面的不等式:

$$T_j + t_j \geqslant T_i + t_i$$

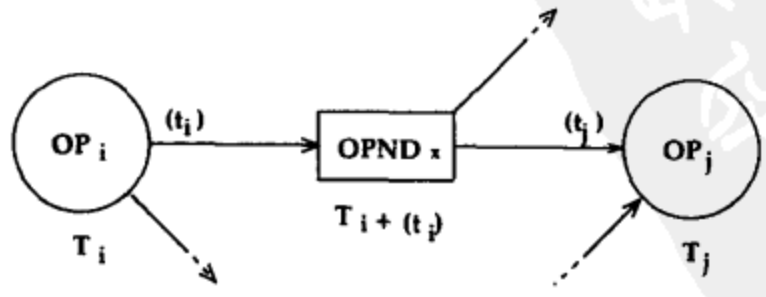
或者表示成:

$$T_j - T_i \geqslant t_i - t_j \tag{F-1}$$

在DFG中，这种不等式对任何一对用边沿连接的操作数都成立。在这些不等式中，T_i 和 T_j 是未知的，而 t_i-t_j 是已知的常数。

为了正确地处理电路的原始或系统输入输出操作数，调度算法采用特殊的输入输出伪运算器。一个只有输出没有输入的输入运算器被用作该电路的一个输入操作数产生源，相应地，一个只有输入的输出运算器被当作该电路的一个输出。

这组不等式的解可以用解线性规划问题的通用技术来求解，例如A.2节和A.3节中的最短路径问题，也可以使用求解整数线性规划问题的软件包例如GAMS[1]来求解。一旦得到这组不等式的解，就可以在运算器 OP_i 和 OP_j 之间插入等于 T_j-T_i-(t_i-t_j) 的延迟时钟周期，来使得电路正确地同步[2]。

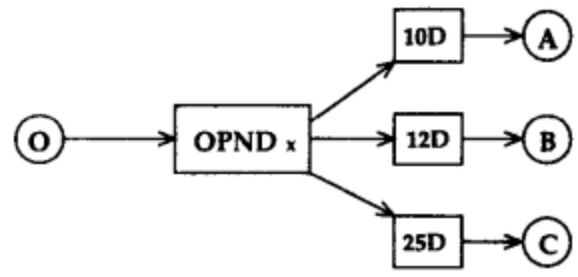


图F-3 一对显示时序信息的运算器

F.3 最小成本解

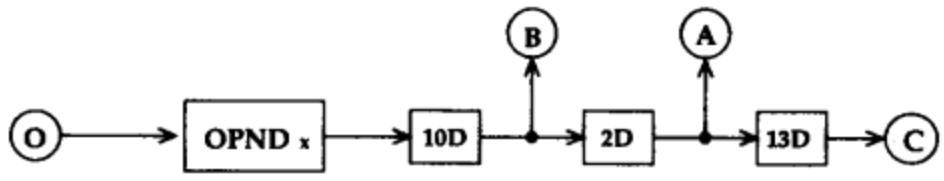
一般来说，对于一个给定的架构，位串行调度问题会有很多的解满足这些不等式，因此，有很多方法使电路同步。位串行调度优化的目标是求得最小成本的解，这里成本定义为使得电路同步所需移位寄存器延迟的总和。如果可以定义一个线性成本函数，那么最小成本问题就可以简单地看作是线性规划问题。

将电路中功能单元之间每个边沿同步所需延迟的总和降到最低是不够的。注意，任何一个功能单元都有可能有多个扇出，因此，一个多扇出输出的延迟应该是串行分配而不是并行分配。举一个简单的例子，某个运算器 OP_o 的一个输出被用作几个其他运算器的一个输入，如图F-4所示:



图F-4 有三个扇出且没有延迟共享的运算器O

OP_o 的输出作为其他三个运算器 OP_A、OP_B 和 OP_C 的输入。本例中，操作数在输入到三个运算器之前分别需要10、12和25个时钟延迟。总的延迟数10+12+25=47。图F-5是另外一种延迟分配方案，这里的延迟都是串行安排的。这种方法总的延迟数等于25，也就是最长延迟的长度。



图F-5 有三个输出和串行延迟的运算器O

由此得到一个结论:

结论F.3.1 电路任意节点所分配的总的延迟数等于所分配的最大延迟。

令 D_x 代表给宽度为 w_x 的操作数 $OPND_x$ 分配的最大延迟，最小化的总成本函数可以表示为:

$$Cost = \sum_x D_x w_x \tag{F-2}$$

该求和表达式包括了电路中每个操作数节点。

对于图F-3中的每个节点，存在一个如公式(F-1)所示的限制条件，这里重复如下:

$$T_j - T_i \geqslant t_i - t_j \tag{F-3}$$

从 OP_i 到 OP_j 的操作数 $OPND_x$ 的最大延迟小于最大延迟 D_x，用下面的约束表示:

$$T_j - T_i - (t_i - t_j) \leqslant D_x \tag{F-4}$$

这两个限制条件公式(F-1)、公式(F-4)和将要最小化的成本函数公式(F-2)描述了一个能为位串行结构提供最小成本调度的线性规划问题。

765 ~ 767

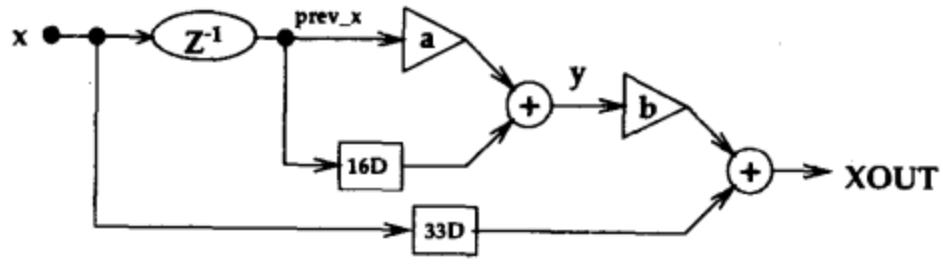
F.4 带延迟的边沿调度

至此为止，我们所描述的位串行调度算法能对于由无延迟边沿组成的DFG产生最优解。许多DSP算法本身就是递归的，并且包含延迟元件。纯粹的前馈电路也不能保证只包含无延

迟边沿。因此位串行调度算法必须能够处理包含延迟的边沿。

考察用下列方程式描述的简单电路:

$$\begin{aligned} prev_x &:= x[-1]; \\ y &:= prev_x * a + prev_x; \\ xout &:= b * y + x; \end{aligned}$$



图F-6 具有 z^{-1} 运算器电路的调度

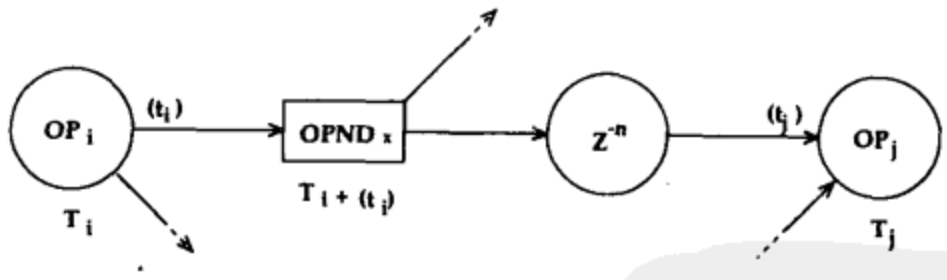
其中a和b是常数，假设W=16，一次常数乘法的等待时间为16个单位时间，一次加法的等待时间为1个单位时间。该电路的具有调度延迟的数据流图如图F-6所示。三角形符号代表常数乘法，如果去掉z^{-1}运算器，16和33延迟周期就可以被合并，但是在DFG中存在z^{-1}运算器，合并是不可能的，因为这两条支路对应两个明显不同的信号 x 和 $x[-1]$。可以发现很多更复杂的例子，其中在调度过程中，由于z^{-1}运算器的存在，它将以非优化的方式影响硬件运算器的调度。

一个比较好的方法是直接将字延迟包含进线性规划方程，这可以通过对公式(F-2)~(F-4)稍微进行修改而将z^{-1}运算器考虑进来。特别地，图F-7给出了一种一般情况，其中某个运算器 OP_i 的输出在用作运算器 OP_j 的输入之前先经过一个z^{-n}的变换。对应于公式(F-1)和公式(F-4)，描述这种调度约束的方程如下:

768

$$T_j - T_i \geqslant t_i - t_j - nW \tag{F-5}$$

$$T_j - T_i - (t_i - t_j) + nW \leqslant D_x \tag{F-6}$$



图F-7 包括 z^{-1} 运算器的一般节点调度

其中W是一个字或字长度的时钟周期数，像前面一样，这里，T_j-T_i-(t_i-t_j)+nW 是图中连接的延迟，D_x 是该变量的最大延迟。约束公式(F-5)和公式(F-6)可以写成一种形式，以便可以作为一个线性规划问题而采用整数调度规划技术求解。

参考文献

1. A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A User's Guide, Release 2.25.* The Scientific Press, 1992.

2. R. I. Hartley and J. R. Jasica, "Behavioral to structural translation in a bit-serial silicon compiler," *IEEE Trans. on Computer-Aided Design*, vol. 7, no. 8, pp. 877–886, Aug. 1988.

769

附录G FIR滤波器的系数量化

G.1 引言

为了用硬件实现数字滤波器，理想滤波器的系数必须首先用具有给定字长K（不包括十进制最左边的符号位）的2的幂表示进行量化。一般量化过程就是结合截断和舍入的方法对系数进行直接二进制转换。这种量化方法是直接而节省时间的。但是量化的滤波器系数包含大量的非零位。在本附录中给出了用于FIR数字滤波器的无乘法实现的新加坡国立大学（NUS）的量化算法，该算法采用有符号的2的幂项[1]。所有的滤波器系数都假设在[-1,1]区间内。

G.2 NUS量化算法

在文献[1]和[2]里说明了如果在滤波器系数进行量化前先进行缩放，所得到的滤波器的频率-空间特性会好很多。通过采用适当的缩放因子，可以使滤波器的所有系数都处在最优的2的幂的量化空间里。NUS算法采用了一种可缩放的量化过程，所得结果的量化滤波器的频率空间特性非常出色。 771

在量化之前，首先对理想滤波器进行归一化，使得滤波器系数的绝对值最大为1，同时用零对量化的滤波器进行初始化。然后，用一个可变的缩放因子（VSF）乘以归一化的理想滤波器。VSF以$2^{-(K+1)}$为步长，从0.4375到1.13进行变化。然后，比较缩放后的理想滤波器和量化滤波器系数之间的差的绝对值，把有符号2的幂(SPT)项分配给其中最大的绝对值所对应的量化滤波器系数。NUS算法重复地分配SPT项，直到所需数目的SPT项均被分配，或者缩放后的理想系数和量化系数之间的所有差的绝对值均小于2^{-K}。一旦停止项的分配，就可以计算归一化峰值纹波(NPR)。然后用一个新的缩放因子重复以上过程，从中选择使得NPR最小的量化滤波器。NUS算法可以用以下伪代码来描述。

NUS量化算法

```
对滤波器的所有系数进行归一化，使得最大系数等于1;
对于VSF =缩放因子的下限: 步长: 缩放因子的上限,
{
    用VSF对归一化的滤波器系数进行缩放;
    用NUS算法中的SPT项分配方法对缩放后的系数进行量化;
    计算量化滤波器的NPR;
}
选择使得NPR最小的量化系数;
```

量化滤波器实现后，使用一个后处理缩放因子(PPSF)对获得的数据流的幅度进行重新缩放，PPSF的值由下式给出:

$$\text{PPSF} = \frac{\text{Max[Abs(Ideal Filter Coeffs.)]}}{\text{VSF}} \tag{G-1}$$

实际上，PPSF是在量化过程中引入的归一化和缩放的逆过程。尽管缩放过程改变了滤波器响应的幅度，但是需要注意的是它并没有改变滤波器的功能。

772 INUS算法采用某种搜索技术对项分配过程稍微进行了改动[2]。尽管量化滤波器的NPR有所改善，但这是以牺牲计算时间为成本的。

为了更好地理解系数量化过程中进行缩放的必要性，这里给出了三个量化的例子。在第一和第三个例子中，缩放过程只使用了一个缩放因子。而在第二个例子中没有进行缩放，导致量化滤波器不能很精确的表示理想滤波器。从这些例子中可以清楚地看到缩放过程带来的好处。

例G.2.1 有缩放的量化

- 字长K等于2。
- 理想滤波器 (IF) =[.26 .131]，初始量化滤波器(QF)=[0 0]。
- 归一化理想滤波器，IF=[1.5038]。
- 用可变缩放因子1/2进行缩放，IF=[.5.2519]。
- 开始量化。
- 一次迭代，QF=[.5 0]。
- 二次迭代，QF=[.5 .25]。
- IF和QF之间的差小于1/4，停止SPT分配。
- 实现滤波器为QF=[.5 .25]和PPSF=.26/.5。 ■

例G.2.2 没有缩放的量化

- 字长K等于2。
- 理想滤波器（IF）=[.26 .131]，初始量化滤波器(QF)=[0 0]。
- 开始量化。
- 一次迭代，QF=[.25 0]。
- IF和QF之间的差小于1/4，停止SPT分配。
- 实现滤波器为QF=[.25 0] ■

例G.2.3 有缩放的量化

- 字长K等于7。
- 理想滤波器（IF）=[.26 .131 .087 .011]，初始量化滤波器(QF)=[0 0 0 0]。
- 773 归一化理想滤波器，IF=[1 0.5038 .3346 .0423]。
- 用可变缩放因子1/2进行缩放，IF=[.5 0.2519 .1673 .0212]。
- 一次迭代，QF=[.5 0 0 0]。
- 二次迭代，QF=[.5 .25 0 0]。
- 三次迭代，QF=[.5 .25 .125 0]。
- 四次迭代，QF=[.5 .25 .1563 0]。
- 五次迭代，QF=[.5 .25 .1563 .0156]。
- IF和QF之间的差小于1/64，停止SPT分配。

对于用SPT项进行并行FIR滤波器的有效量化，读者可以参考[3]，[4]。 ■

参考文献

1. D. Li, J. Song and Y. C. Lim, "A polynomial-time algorithm for designing digital filters with power-of-two coefficients," in *Proc. of IEEE International Symposium on Circuits and Systems*, (Chicago, IL), pp. 84–87, May 1993.
2. C. L. Chen, K. Y. Khoo and A. N. Willson, Jr, "An improved polynomial-time algorithm for designing digital filters with power-of-two coefficients," in *Proc. of IEEE International Symposium on Circuits and Systems*, (Seattle, WA), pp. 223–226, May 1995.
3. D. A. Parker and K. K. Parhi, "Low-area/power parallel fir digital filter implementations," *Journal of VLSI Signal Processing*, no. 1/2, pp. 75–92, Sept. 1997.
4. J.-G. Chung, Y.-B. Kim, H.-G. Jeong, K. K. Parhi, and Z. Wang, "Efficient parallel FIR filter implementations using frequency spectrum characteristics," in *IEEE International Symposium on Circuits and Systems*, (Monterey, CA), vol. 5, pp. 354–358, June 1998.

索　引

注：索引页码为英文原书页码，即本书页边所标页码。

A

ACS（Addition，Comparsion，Selection，加法、比较与选择），21、180
acyclic precedence graph（无环路优先图），131
adaptive filter（自适应滤波器），6，301，351
　LMS（Least-Mean-Square，最小均方），7，355
　stochastic-gradient lattice（随机梯度格子），8，357
adaptive lattice filter（自适应格型滤波器）
　stochastic-gradient（随机梯度），8
additive number splitting（加性数字拆分）
　column-based（基于列的），577
　row-based（基于行的），575
AGU（Address Generation Units，地址产生单元），701
ALAP（As late as possible，尽量晚），727
algorithmic strength reduction（算法强度缩减），227、225
　DCT（离散余弦变换），277
　FIR filter（FIR滤波器），302
　motion estimation（运动估值），305
　parallel FIR filters（并行FIR滤波器），256
　parallel rank-order filters（并行秩-阶滤波器），286
　two-dimensional FIR filter（二维FIR滤波器），304
　vector quantizer（矢量量化器），303
all-pass filter（全通滤波器），431、472
all-pole filter（全极点滤波器），431
allocation（分配），723
ALU（Arithmetic Logic Unit，算术逻辑单元），698
annihilation-reordering look-ahead（超前湮没-重排序），367
applicative language（应用性语言），32
approximate processing（近似处理），688
array multiplier（阵列乘法器）
　carry-ripple（串行进位），481
　carry-save（保留进位），482
ASAP（As soon as possible，尽量早），726
ASIC（Application-Specified Integrated Circuit，专用集成电路），31
associativity（结合），201、503
asynchronous 2-input multiplexer（异步二输入多路器），633
asynchronous circuit（异步电路），2
　first-in first-out buffer（先进先出缓冲器），638
asynchronous pipelining（异步流水线），591，619

B

basic lattice filter（基本格型滤波器），430
　pipelining（流水线），460
Baugh-Wooley multiplication（Baugh-Wooley乘法），484
Bellman-Ford algorithm（Bellman-Ford算法），96，104，155，717，718
binary look-ahead adder（二进制超前进位加法器），756
binary search（二进制搜索），105
biquad filter（二-四滤波器），59
bit-parallel processing（位并行处理），134
bit-parallel system（位并行系统），477
　multiplication（乘法），479
bit-plane architecture（位平面架构），489，492
bit-plane digital filter（位平面数字滤波器），489
bit-repeater（位重复器），494
bit-serial multiplier（位串行乘法器），490
bit-serial processing（位串行处理），134
bit-serial scheduling（位串行调度），763
bit-serial system（位串行系统），477
　filter（滤波器），499
　multiplier（乘法器），490
　redundant multiplier（冗余乘法器），542
block delay（块延时），70
block diagram（框图），33
block processing（块处理），70
　block delay（块延时），70
　block size（块尺寸），70
　L-slow（L倍降速），70
block size（块尺寸），70
block-matching algorithm（块匹配算法），12
　full search（全搜索），12
BMA（Block-matching algorithm，块匹配算法），11
Booth recoding（Booth再编码）

low-power（低功耗），522
race-free（无竞争），523
bundled-data protocol（捆绑数据协议），620

C

C-element（C元件），629，633
canonic signed digit（正则符号数），505，566
two common subexpressions（两种通用子表达式），571
carry completion adder（进位完成加法器），636，637
carry-free addition（免进位加法），529
carry-look-ahead adder（超前进位加法器），756
carry-ripple array multiplier（串行进位阵列乘法器），481
carry-save array multiplier（进位存储阵列乘法器），482
vector merging adder（矢量合并加法器），483
carry-select adder（进位选择加法器），755
CCA（carry-completion adder，进位完成加法器），637
CDFG（control and data-flow graph，控制和数据流图），724
Chinese remainder theorem（中国余数定理），237
for integers（用于整数的），238
for polynomials（用于多项式的），238
Chromatic MPact（Chromatic的MPact处理器），710
chrominance（色度），28
CIF（common intermediate format，通用媒介格式），28
circular convolution（圆周卷积），246
circular lifetime chart（圆形寿命图），160
CLB（computational logic block，计算逻辑模块），601
clock distribution（时钟分布），605
clock period（时钟周期），32
clock period minimization（时钟周期最小化），100
clock skew（时钟歪斜），591，601
clock distribution（时钟分布），605
two-phase clocking（两相时钟），604
clocking style（时钟风格），591
true single phase（真单相），591
two-phase（两相），591
clustered look-ahead（聚类超前），325，327
coding in pipelined DSP（流水线数字信号处理器中的编码）
data-stationary（数据固定的），701，702
time-stationary（时间固定的），701
color format conversion（颜色格式转换），584
column-based additive number splitting（基于列的加性数字拆分），577
communication bound（通信边界），71
compare-and-swap（比较和互换），287
completion signal（完成信号），631
complex multiplication（复数乘法），227
constrained filter design（受约束的滤波器设计），325，335
constraint graph（约束图），103
convergence analysis（收敛分析），359
convolution（卷积），3
Cook-Toom algorithm（Cook-Toom算法），228
coprocessor（协处理器），708
CORDIC（Coordinate rotation digital computer，坐标旋转数字计算机），113，183
correlation（相关），4
CPL（complementary pass transistor logic，互补传输晶体管逻辑），592，671
CPU（中央处理单元），697
critical loop（关键环路），327
critical path（关键路径），31，45，64，66，127
critical path constraint（关键路径约束），102
CRT（Chinese Remainder Theorem，中国余数定理），228，237
CSD（正则符号数位），505
cutset（割集），66
feed-forward（前馈），66
cutset retiming（割集重新定时），97，98
cyclic convolution（循环卷积），246

D

D-latch（D锁存器）
dynamic（动态的），597
static（静态的），596
Dadda multiplier（Dadda乘法器），759
data format conversion（数据格式转换），545
lsd-first（最低有效数位优先），548
msd-first（最高有效数位优先），548
nonredundant to redundant（非冗余到冗余），546
redundant to nonredundant（冗余到非冗余），547
data-broadcast（数据广播），34，68，84，144，189
data-driven property（数据驱动特性），2，36
data-flow graph（数据流图），33，36，43
DCT（离散余弦变换），13，275
algorithm-architecture transformation（算法架构转换），277

decimation-in-frequency（频率抽取），281
transposition（转置），284
DCVSL（differential cascode voltage switch logic，差分级联电压开关逻辑），631，632
DCVSPG（differential cascode voltage switch with pass transistor gate，带传输晶体管门的差分级联电压开关），671
decimator（抽取器），22，170，300
degree reduction（深度缩减），429
delay relaxation（延时弛豫），354
delay-insensitive（延时不敏感的），620
dependence graph（依赖图），33，39
regular（规则的），190
descriptive language（叙述语言），32
DFG（数据流图），33，37，43
atomic（微粒子的），37
coarse-grain（粗粒度），38
nonrecursive（非递归的），45
recursive（递归的），45
DFT（discrete Fourier transform，离散傅立叶变换），13
DG（依赖图），33
digit-serial（数位串行），477
digit-serial processing（数位串行处理），134
digit-size（数位大小），477
digital filter（数字滤波器），5
bit-plane（位平面），489
bit-serial（位串行），499
interleaved floor-plan（交错平面布局），489
state variable description（状态变量描述），382
discrete cosine transform（离散余弦变换），13，275
discrete Fourier transform（离散傅立叶变换），13
discrete wavelet transform（离散小波变换），25，186，219，300
analysis filter bank（分析滤波器组），26，28
synthesis filter bank（合成滤波器组），26，28
distributed arithmetic（分布式运算），511
offset-binary coding（偏移二进制编码），513
ROM decomposition（ROM分解），517
division algorithm（除法算法），743
for polynomials（用于多项式），745
domain-specific DSP（特定域DSP），696
DP（dynamic programming，动态规划），20
DSP synthesis（DSP综合），723
dual-rail protocol（双轨协议），620
DWT（discrete wavelet transform，离散小波变换），25
dynamic circuit（动态电路），661
dynamic power dissipation（动态功耗），648
dynamic programming（动态规划），19，20，180，372

E

edge reversal（边反转），201
Euclidean distance（欧几里德距离），18
Euclidean GCD algorithm（欧几里德最大公因子算法），743，744
for polynomials（用于多项式），745
EVTCMOS（elastic-Vth CMOS，弹性开启电压CMOS），667
exhaustive retiming（穷举重新定时），112
expander（扩展器），22，23，170

F

fast binary adders（快速二进制加法器），753
binary-look-ahead（超前二进制），756
carry-look-ahead（超前进位），756
carry-select（进位选择），755
multiplexer-based（基于多路器的），753
tree-based（基于树的），756
fast convolution（快速卷积），227
Cook-Toom algorithm（Cook-Toom算法），228
cyclic convolution（循环卷积），246
iterated convolution（迭代卷积），244
Winograd algorithm（Winograd算法），237
fast FIR algorithm（快速FIR算法），260
FDS（force-directed scheduling，力度导向的调度），730
feed-forward cutset（前馈割集），66
FFA（fast FIR algorithm，快速FIR算法），260
FFT（快速傅立叶变换），xvii
field-programmable gate array（现场可编程门阵列），631
filter bank（滤波器组）
analysis（分析），25
synthesis（综合），25
tree-structured（树结构的），26
fine-grain pipelining（细粒度流水线），69
finite impulse response（有限冲击响应），4
finite wordlength（有限字长），1
FIR filter（FIR滤波器），4，5
difference coefficient（差分系数），302

lattice filter（格型滤波器），437
linear phase（线性相位），6
sum coefficient（系数求和），303
systolic array（脉动阵列），192
two-dimensional（二维），84，304
flow graph reversal（流图反转），35
Floyd-Warshall algorithm（Floyd-Warshall算法），97，104，155，717，720
folding（折叠），149
folding factor（折叠因子），151
folding order（折叠阶），151
multirate system（多速率系统），170
register minimization（寄存器最小化），163
force-directed scheduling（力度导向的调度），730
forward-backward register allocation（前向-后向寄存器分配），160
FPGA（现场可编程门阵列），31，631
full handshake（全握手），630

G

GAMS（一种解整数线性规划问题的程序），739
GCD（最大公约数），238，743
GOPs（gigaoperations，千兆次运算），27

H

H-tree network（H树型网络），605，606
half handshake（不完全握手），630
handshake circuit（握手电路），592，626
hardware utilization efficiency（硬件利用效率），191
Harvard architecture（哈佛结构），699
HDTV（高清晰度电视），28
HEAT tool（hierarchical energy analysis tool，层次化能量分析工具），677
high-definition TV（高清晰度电视），28
high-level synthesis（高层次综合），37
Horner's rule（Horner准则），479，490，508

I

IDWT（inverse DWT，离散小波逆变换），25
IIR filter（无限冲击响应滤波器），4，5
lattice filter（格型滤波器），421
parallel processing（并行处理），339
ILP scheduling（整数线性规划调度），738
incremental block processing（增量性块处理），313
infinite impulse response（无限冲击响应），4
inter-iteration precedence constraint（迭代间优先约束），37，44
interconnect capacitance（互连电容），661
interconnection circuit（互连电路），626
interleaved floor-plan（交错布局规则），489
intra-iteration precedence constraint（迭代内优先约束），37，44，725
iterated convolution（迭代卷积），244
iteration（迭代），31，44
iteration bound（迭代边界），43，45，47，95，126，183
longest path matrix（最长路径矩阵），43，47
maximum cycle mean（最大周期均值），52
minimum cycle mean（最小周期均值），43，47，50
multirate DFG（多速率数据流图），55
negative cycle detection（负周期检测），47
iteration period（迭代周期），31
iteration period bound（迭代周期边界），43
iteration rate（迭代速率），31
iteration vector（迭代矢量），191

K

K-computing algorithm（*K*计算算法），403
Kronecker product（Kronecker积），271

L

L-slow（*L*倍降速），70，99，120
Lagrange interpolation（拉格朗日插值），228
language（语言）
applicative（应用性的），32
descriptive（叙述性的），32
prescriptive（说明性的），32
latency（迟滞），32，509
lattice filter（格型滤波器），421
basic（基本的），430
FIR（有限冲击响应），437
IIR（无限冲击响应），421
normalized（规格化的），60，444
one-multiplier（单乘法器），437
PARCOR（partial correlation coefficient，偏相关系数），9
reflection coefficient（反射系数），9
reversed Schur polynomial（反转Schur多项式），434
roundoff noise（舍入噪声），454
scaled-normalized（按比例缩放-规格化），447，449

lattice FIR filter（格型FIR滤波器），437
lattice IIR filter（格型IIR滤波器），421
low-power（低功耗），469
pipelining（流水线），458
lifetime analysis（寿命分析），157
lifetime chart（寿命图）
circular（循环的），160
linear（线性的），157
linear lifetime chart（线性寿命图），157
linear mapping（线性映射），189
list scheduling（表单调度），728
LIW（long instruction word，长指令字），707
LMS adaptive filter（最小均方自适应滤波器），7
longest path matrix（最长路径矩阵），47
look-ahead（超前），313，316，319
annihilation-reordering（湮没-重排序），367
clustered（聚类的），325
general decomposition（一般的分解），324
power-of-2 decomposition（按2的幂次分解），321平方幂分解
scattered（离散的），325
loop（环路），45
loop bound（环路边界），46
loop delay synchronization（环路延时同步），502
loop scheduling（环路调度），734
loop unrolling（环路打开），119
low-power（低功耗），74
Booth recoding（Booth再编码），522
lattice IIR filter（格型IIR滤波器），469
parallel processing（并行处理），76，348
pipelining（流水线），74，348
pipelining and parallel processing（流水线和并行处理），81
supply voltage reduction（电源电压降低），75，77
LPM（longest path matrix，最长路径矩阵），43
luminance（亮度），28
Lyapunov equation（Lyapunov方程），385
Lyon's multiplier（Lyon乘法器），494

M

MAC（multiply accumulate，乘法累加），698
MAD（mean absolute difference，均方绝对误差），12
Manchester adder（Manchester加法器），754
MARS（一种基于迭代回路的综合系统），734
matrix-matrix multiplication（矩阵乘法），206
MAX（multimedia acceleration extensions，多媒体加速辅助部件），713
maximally redundant（最大冗余），531
maximum cycle mean（最大环均值），52
MCM
minimum cycle mean（最小环均值），43
multiple constant multiplication（多常数乘法），560
ME（motion estimation，运动估值），2
media processor（媒体处理器），696，704，709
Chromatic MPact（Chromatic的MPact处理器），710
Philips Trimedia（Philips的Trimedia处理器），710
Texas Instruments C8x（德州仪器公司的C8x处理器），709
TI C6x（德州仪器公司的C6x处理器），711
MIMD（multiple instruction multiple data，多指令多数据），704
MIMO（multiple-input multiple-output，多输入多输出），35，70
minimally redundant（最小冗余），531
minimum cycle mean（最小周期均值），50
minimum unfolding factor（最小展开因子），129，130
misadjustment error（失调误差），360
MMP（minus-minus-plus adder，减-减-加 加法器），535
MMX（multimedia extensions，多媒体辅助部件），713
modified bit-plane（改进的位平面），490
modified Booth recoding（改进的Booth再编码），487
MOPs（megaoperations，兆次的运算），30
motion estimation（运动估值），11，222，223
algorithmic strength reduction（算法强度缩减），305
block-matching algorithm（块匹配算法），12
MPEG（由运动图像专家组开发的一种视频通信标准），29
MPEG-2（MPEG标准的一种改进版），29
MRDFG（multirate DFG，多速率数据流图），39，172
MSE（mean-square error，均方误差），12
MTCMOS（multi threshold-voltage CMOS，多阈值电压CMOS），667
multimedia extension（多媒体辅助部件），704
multimedia signal processing（多媒体信号处理），704
multiple constant multiplication（多常数乘法），560
multiple-input multiple-output（多输入多输出），70
multiplexer adders（多路器加法器），753
multiplication（乘法）

Dadda（人名），759
Wallace tree（Wallace树），758
multiplicative number splitting（乘性数字拆分），579
multiprocessing（多重处理），704
multirate DFG（多速率数据流图），55
multirate folding（多速率折叠），170，185
multirate system（多速率系统），22
folding（折叠），170
multirate DFG（多速率数据流图），55
noble identity（惰性恒等变换），23，173
retiming（时序变换），173

N

negative cycle（反向圈），717
noble identity（惰性恒等变换），23，173
nonterminating program（无休止程序），31，43
normalized lattice filter（归一化格型滤波器），60，444
pipelining（流水线），463
NPCPL（normal process complementary pass transistor logic，标准工艺的互补传输晶体管逻辑），592，614
NSR（noise to signal ratio，噪信比），398
number splitting（数字拆分），574
additive（加性的），575
multiplicative（乘性的），579
numerical strength reduction（数字强度缩减），559

O

OBC（offset-binary coding，偏移二进制编码），513
OBDD（ordered binary-decision diagram，有序的二进制决策图），675
odd-even merge-sort（奇偶合并分类），286
on-line arithmetic（在线运算），529
one-multiplier lattice filter（单乘法器格型滤波器）
pipelining（流水线），462
orthogonal filter（正交滤波器），113，385
orthogonal transform（正交变换），16，277
over-redundant（超冗余的），531
overflow（溢出），481

P

parallel multiplication（并行乘法），479
sign extension（符号扩展），479
parallel processing（并行处理），63，69，131
bit-level（位级的），134
IIR filter（IIR滤波器），339
low-power（低功耗），76，348
rank-order filter（秩-阶滤波器），285
word-level（字级的），132
parallel-to-serial converter（并-串转换），73
PARCOR（partial correlation coefficient，偏相关系数），9
path metric（路径度量），20
PDA（personal digital assistant，个人数字助手），711
PDSP（programmable digital signal processor，可编程数字信号处理器），31，695
PE（processing element，处理单元），189
performance（性能），1
performance characteristics（性能特征），645
Philips Trimedia（Philips的Trimedia处理器），710
pipeline interleaving（流水线交织），314
pipelining（流水线），63，64，97，99
basic lattice filter（基本格型滤波器），460
clustered look-ahead（聚类超前），327
fine-grain（细粒度），69
FIR filter（FIR滤波器），64
IIR filter（IIR滤波器），320
lattice IIR filter（格型IIR滤波器），458
LMS adaptive filter（最小均方自适应滤波器），355
low-power（低功耗），74，348
normalized lattice filter（规格化格型滤波器），463
one-multiplier lattice filter（单乘法器格型滤波器），462
roundoff noise（舍入噪声），405
scaled-normalized lattice filter（按比例缩放-规格化格型滤波器），464
scattered look-ahead（离散超前），330
stochastic-gradient lattice filter（随机梯度格型滤波器），357
pipelining and parallel processing（流水线和并行处理），81
pole interlace property（极点交错性质），473
pole-zero cancellation（极-零点消除），324，325，335
polynomial degree reduction（多项式阶次降低），429
polynomial division algorithm（多项式除法算法），745
polynomial expansion algorithm（多项式展开算法），427
polynomial GCD algorithm（多项式最大公因式算法），745
polyphase decomposition（多相分解），256
power analysis（功耗分析），652

power consumption（功耗），74，645
effect of scaling（缩放效应），651
leakage（漏电），646
short-circuit（短路），646
standby（闲置），646
power estimation（功耗估计），671
hierarchical approach（层次化方法），677
nonsimulative（非仿真性的），674
simulation-based（基于仿真的），672
power reduction（功耗降低）
approximate processing（近似处理），688
circuit styles（电路风格），670
clocking（时钟），668
dual/multi-V_{th}（双/多V_{th}），667
multiple supply voltages（多电源电压），665
parallel processing（并行处理），76
path balancing（路径平衡），662
pipelining（流水线），74
pipelining and parallel processing（流水线和并行处理），81
retiming（重定时），664
supply voltage reduction（电源电压降低），75，77，82，348
transistor and gate sizing（晶体管和逻辑门尺寸优化），663
transistor reordering（晶体管重排序），664
voltage scaling（电压按比例降低），665
PPM（plus-plus-minus adder，加-加-减 加法器），532
precedence constraint（优先约束），123
inter-iteration（迭代之间），37，44
intra-iteration（迭代内部），37，44，725
precedence graph（优先图），725
prescriptive language（说明性语言），32
processor space vector（处理器空间矢量），191
programmable DSP（可编程DSP），145，299，305，695
memory architecture（存储器架构），699
projection vector（投影矢量），191
propagation delay（传输延时），74，77
pseudocirculant matrix（准循环行列式矩阵），260

R

race-free Booth recoding（无竞争Booth再编码），523
radix-2 addition（基2加法），532
radix-2 subtraction（基2减法），533
radix-4 addition（基4加法），536
RAM（random access memory，随机存取存储器），145
rank-order filter（秩-阶滤波器），285
low-power（低功耗），294
rate of convergence（收敛速率），360
RDG（reduced dependence graph，简化的依赖图），203
real-time processing（实时处理），2
redundancy factor（冗余因子），531
redundant multiplication（冗余乘法），540
redundant number system（冗余数系统），529
data format conversion（数据格式转换），545
nonredundant to redundant（非冗余到冗余），546
redundant to nonredundant（冗余到非冗余），547
maximally redundant（最大冗余），531
hybrid radix-4 addition（混合基4加法），538
minimally redundant（最小冗余），531
hybrid radix-4 addition（混合基4加法），539
multiplication（乘法），540
bit-serial redundant multiplier（位串行冗余乘法器），542
over-redundant（超冗余），531
radix-2 addition（基2加法），532
radix-2 subtraction（基2减法），533
radix-4 addition（基4加法），536
redundancy factor（冗余因子），531
signed-digit representation（符号数表示），529
register allocation（寄存器分配）
forward-backward（前向-后向），160
register minimization（寄存器最少化），106，157，163
lifetime analysis（寿命分析），157
regular dependence graph（规则依赖图），190
regular iterative algorithm（规则迭代算法），203
relaxed look-ahead（弛豫超前），313，352
delay relaxation（延时弛豫），352，354
product relaxation（乘积弛豫），352，353
sum relaxation（和弛豫），352，354
representations（表示）
block diagram（框图），33
data-flow graph（数据流图），36
dependence graph（依赖图），39
signal-flow graph（信号流图），34
residue arithmetic（余数运算），518
resource allocation（资源分配），723

resource-constrained synthesis（约束资源的综合），724
retiming（重定时），91，127，319，616
2D DFG（二维数据流图），112
clock period minimization（时钟周期最小化），100
critical path constraint（关键路径约束），102
cutset retiming（割集时序变换），97，98
exhaustive（穷举），112
fanout constraint（扇出约束），107
feasibility constraint（可行性约束），102
for folding（用于折叠的），153
L-slow（L倍降速），99
low-power（低功耗），664
MRDFG（多速率数据流图），173
multirate DFG（多速率数据流图），173
pipelining（流水线），97，99
properties（性质），94
register minimization（寄存器最少化），106
roundoff noise（舍入噪声），405
slow-down（减速），99
two-dimensional DFG（二维数据流图），112
reverse polynomial（反转多项式），423
RGB（基于3种基本颜色元素表示的彩色数字图像元素表示法，R-红色，G-绿色，B-蓝色），27
RIA（regular iterative algorithm，规则迭代算法），203
standard input（标准输入），203
standard output（标准输出），203
RISC（reduced instruction set computer，精简指令集计算机），708
RLS（recursive least square，递归最小平方），367
roundoff noise（舍入噪声），2，377，380，382，389
computation（计算），388，403
lattice filter（格型滤波器），454
pipelined IIR filter（流水线IIR滤波器），391
roundoff noise（舍入噪声）
computation（计算），403，404
roundoff noise computation using transposition（使用变换的舍入噪声计算），454
row-based additive number splitting（基于行的加性数字拆分），575
RTL（register-transfer level，寄存器传输级），723
running order merge-sorter（运行次序合并-分类器），293

S

sampling rate（采样率），31
SBD（signed-binary-digit，符号二进制数），531
scaled tangent rotation（按比例缩放正切旋转），367
scaled-normalized lattice filter（按比例缩放-规格化格型滤波器），447，449
pipelining（流水线），464
scaling（缩放），377，378，386
scattered look-ahead（离散超前），325，330
schedule（时间表），37
nonoverlapped（非交迭的），131
overlapped（交迭的），131
scheduling（调度），723
as late as possible（尽量晚），727
as soon as possible（尽早），726
bit-serial systems（位串行系统），763
force-directed（力度导向的），730
integer linear programming（整数线性规划），738
iterative/constructive（迭代的/构造的），725
list scheduling（表单调度），728
loop-based（基于环路的），734
resource-constrained（资源约束的），763
systolic array design（脉动阵列设计），201
transformational（转换的），729
scheduling inequalities（调度不等式），202
scheduling vector（调度矢量），191
Schur algorithm（Schur算法），422
power calculation（功耗计算），428
Schur polynomial（Schur多项式），422，430
orthogonality（正交性），749
orthonormality（正交归一性），424，750
reverse（反转），434，750
SDFG（synchronous data-flow graph，同步数据流图），38，39
self-timed system（自定时系统），592，619
serial-to-parallel converter（串并转换器），72
SFG（信号流图），33
shimming delay（补偿延时），503
short-circuit power dissipation（短路功耗），647
shortest path algorithm（最短路径算法），95，717
Bellman-Ford algorithm（Bellman-Ford算法），96
Floyd-Warshall algorithm（Floyd-Warshall算法），97
sign extension（符号扩展），480
signal transition graph（信号转换图），622
consistency（一致性），623
liveness（可激活性），623
persistence（可持续性），623

safety（安全性），623
semimodularity（半模式化），624
signal-flow graph（信号流图），33，34，66
signed-digit representation（符号数位表示），529，530
SIMD（single instruction multiple data，单指令多数据），696，697
skewing delay（歪斜延时），503
slow-down（降速），99，201，405
SNR（signal to noise ratio，信噪比），1，379
space representation（空间表示），190
SR latch（SR锁存器），629
SRDFG（single-rate DFG，单速率DFG），38，172
SRP（slow-down and retiming/pipelining，降速和重定时/流水线），447
SRPL（swing-restored pass transistor logic，摆动恢复传输晶体管逻辑），671
state transition diagram（状态转移图），679
state variable description（状态变量描述），382
static power dissipation（静态功耗），647
static scheduling（静态调度），707
STD（state transition diagram，状态转移图），677
steady-state probability（稳态概率），683
STG（signal transition graph，信号转换图），592
subexpression elimination（子表达式消除），559，570
digital filter（数字滤波器），566
linear transformation（线性变换），562
multiple constant multiplication（多常数乘法），560
polynomial evaluation（多项式估值），565
substructure sharing（子结构共享），559
subword parallelism（子字并行处理），705
sum relaxation（和弛豫），354
superscalar architecture（超标量架构），707
supply voltage reduction（电源电压降低），75
survivor path（幸存路径），20
switching activity（开关活动率），653
glitching（毛刺），653
spatial correlation（空间相关），653
spatiotemporal correlation（时空相关），654
temporal correlation（时间相关），653
synchronizing registers（同步寄存器）
minimization（最小化），763
synchronous data-flow graph（同步数据流图），38
synchronous pipelining（同步流水线），593
synthesis（综合），723
resource-constrained（资源约束的），724
time-constrained（时间约束的），724
systolic array（脉动阵列），189
bit-serial multiplier（位串行乘法器），495
bit-serial redundant multiplier（位串行冗余乘法器），542
discrete wavelet transform（离散小波变换），219
FIR filter（FIR滤波器），192
matrix-matrix multiplication（矩阵乘法），205
motion estimation（运动估值），222，223
vector quantization（矢量量化），215
systolic array design（脉动阵列设计），189
hardware utilization efficiency（硬件利用效率），191
iteration vector（迭代矢量），191
processor space vector（处理器空间矢量），191
projection vector（投影矢量），191
scheduling（调度），201
scheduling vector（调度矢量），191

T

technology（技术），30
scaled CMOS（按比例缩小CMOS），30
tensor product（张量积），271
Texas Instruments C8x（德州仪器公司的C8x数字信号处理器），709
three-dimensional filter（三维滤波器），302
throughput（吞吐量），31
TI C6x（德州仪器公司的C6x数字信号处理器），711
time-constrained synthesis（时间约束的综合），724
Toeplitz matrix（Toeplitz矩阵），216
trace（迹），389
transformation（变换）
algorithmic strength reduction（算法强度缩减），255
associativity（结合性），201，503
clustered look-ahead（聚类超前），314，327
edge reversal（边反向），201
folding（折叠），149
linear mapping（线性映射），189
look-ahead（超前），314，316
numerical strength reduction（数字强度缩减），559
pipelining（流水线），63
retiming（重定时），91
scattered look-ahead（离散超前），314，330
slow-down（降速），201

tree-height reduction（树高度缩减），509
unfolding（展开），119
transition density（转换密度），647
transposition（转置），35，68
DCT（离散余弦变换），284
parallel FIR filter（并行FIR滤波器），264
roundoff noise computation（舍入噪声计算），454
transposition theorem（转置定理），454
tree-based adder（基于树的加法器），756
tree-height reduction（树高度缩减），509
trellis diagram（格状图），20，373
true single phase clocking（真单相时钟），591，598
two-dimensional FIR filter（二维FIR滤波器），84，304
two-phase clocking（两相时钟），591，594

U

unfolding（折叠），119，271
L-slow（L倍降速），120
parallel processing（并行处理），131
properties（性质），124
sample period reduction（抽样周期降低），128
unfolding algorithm（展开算法），121
unfolding factor（展开因子），119
unit-sample response（单位抽样响应），3

V

vector merging adder（矢量合并加法器），483
vector quantization（矢量量化），17，215，303
differential tree-search（差分树搜索），304
full-search（全搜索），18
tree-structured（树结构），19
video compression（视频压缩），29
Viterbi algorithm（Viterbi算法），19
Viterbi decoder（Viterbi译码器），373
VLD（variable-length decoder，变长译码），710
VLIW（very long instruction word，超长指令字），697，707
VLSI（very-large-scale-integrated，大规模集成），1
VMA（vector merging adder，矢量合并加法器），483
VQ（vector quantization，矢量量化），2
VTCMOS（variable threshold-voltage CMOS，可变阈值电压CMOS），667

W

W-computing algorithm（W-计算算法），404
Wallace tree multiplication（Wallace树乘法），758
wave digital filter（波数字滤波器），60，112
wave pipelining（波流水线），591，607
algorithm（算法），615
constraint space diagram（约束空间图），612
degree of（深度），612
wavelet transform（小波变换），24
Winograd convolution（Winograd卷积），237

Y

YUV（基于人体视觉系统的彩色数字图像元素表示法，Y表示亮度，U和V分别表示Y与蓝色及Y与红色之间的色差），27

本书是一部论述VLSI信号处理中性能优化技术的全面的、富于指导性的著作，由作者根据多年研究生课程教学经验编著而成，是这一领域中的一本标准教科书和重要的参考文献。

本书汇集了VLSI的架构理论与算法，描述了硬件实现层中的各种架构，给出了若干种分析、估计与降低功耗的方法，重点讲述设计应用时所需要的定制或半定制VLSI电路。

本书不仅适合作为大学相关专业的教科书，对于专业技术人员，也是一本很好的参考书。

作者简介

Keshab K. Parhi 教授于1982年在印度理工学院取得工学学士学位，1984年在美国宾夕法尼亚大学电机工程系取得硕士学位，1988年在加州大学伯克利分校电机工程与计算机科学系取得博士学位。从1988年起就一直在明尼苏达大学从事教学与研究工作。他现在是电机与计算机工程系的“McKinght荣誉教授”。他的研究领域包括涉及宽带通信系统物理层方面的VLSI架构设计，当前正在进行的是纠错编码器与密码学架构、高速发送/接收器、超宽带系统、量子纠错编码器与量子密码学方面的研究工作。

VLSI Digital Signal Processing Systems: Design and Implementation （美）Keshab K. Parhi

ISBN 7-111-14108-3
9 787111 141082

HZ BOOKS
华章图书

网上购书：www.china-pub.com
北京市西城区百万庄南街1号 100037
读者服务热线：(010)68995259，68995264
读者服务信箱：hzedu@hzbook.com
http://www.hzbook.com

ISBN 7-111-14108-3/TN · 376
定价：65.00 元