上海交通大学硕士学位论文

全芯片时钟网络的综合与优化方法

硕士研究生：黄伟坚

学　　号　：1082109015

导　　师：施国勇教授

副 导 师 ：

申请学位：工学硕士

学　　科　：电路与系统

所在单位：微电子学院

答辩日期：2010 年 1 月

授予学位单位：上海交通大学

此研究由国家自然科学基金（编号 60876089）资助

Dissertation Submitted to Shanghai Jiao Tong University
for the Degree of Master

ON GLOBAL CLOCK MESH SYNTHESIS AND OPTIMIZATION

Candidate：	Weijian Huang
Student ID:	1082109015
Supervisor：	Prof. G. Shi
Assistant Supervisor:	Prof.
Academic Degree Applied for：	Master of Engineering
Speciality：	Circuits and System
Affiliation：	School of Microelectronics
Date of Defence：	Jan, 2010
Degree-Conferring-Institution：	Shanghai Jiao Tong University

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文《全芯片时钟网络的综合与优化方法》，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保密□，在___年解密后适用本授权书。

不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

全芯片时钟网络的综合与优化方法

摘 要

随着半导体工艺技术的不断进度，沟道尺寸的不断缩小，65 纳米和 45 纳米已成为主流的工艺技术，并向着 32 和 22 纳米向前发展。然而，由于工艺尺寸缩小造成的不确定性的对芯片时钟的干扰越来越大，时钟设计已成为整个后端设计的重点和难点。传统的时钟树（clock tree）综合已越来越难以满足当今设计对时钟综合提出的功耗和时钟偏差（clock skew）等要求。网格型时钟由于抗干扰能力强，时钟偏差小的特点，越来越受到工业界的关注，网格型时钟分布已经成功应用于大型的芯片设计，如 600-MH Alpha；IBM G5 S/930；Power4；Power PC; SUN Sparc V9。但是由于没有成熟完善的工具支持自动化的网格型时钟综合，限制了网格型时钟的广泛应用。

本课题将实现网格型时钟（clock mesh）的整个综合和优化流程，实现 100%的全自动化网格型时钟设计。本文提出了这个综合和优化流程的框架，主要包括以下六个步骤：网格规划、结构定位、局部布线、缓冲器插入、约束验证和缓冲器优化。本文将对实现上述每个步骤的算法加以研究，使时钟网格在满足时序和功耗约束的两个重要前提下尽可能地优化，同时总结出时钟网格的一般特性。实验结果表明，该自动化流程在学术界标准的测试案例中表现出了较高的质量，综合后的时钟网格在时序和功耗方面满足了严格的约束要求。

关键词（四号黑体）：时钟综合，时钟网格，局部布线，缓冲器插入，约束验证。

ON GLOBAL CLOCK MESH SYNTHESIS AND OPTIMIZATION

ABSTRACT

With the rapid development of semiconductor technology, the characteristic dimension shrinks. 65nm and 45nm technology have become the mainstream, it's believed that 32nm and 22nn technology will be in the near future. But, as side effect of technology advancement, many factors have a great impact on clock quality. Clock design is now the key to the whole backend design. The traditional clock tree structure was no longer able to meet the challenging timing and power constraints. Clock mesh structure, on the other hand, due to its robustness to variances and small clock skew, it has been applied to high-speed chip design successfully, such as 600-MHZ Alpha, IBM G5 S/930, Power4, Power PC, and SUN Sparc V9. However, clock mesh is not applied widely, because of the lack of good EDA tool support.

This dissertation presents a full automation synthesis and optimization framework of clock mesh, which includes six major steps as follows: mesh planning, stem placement, local routing, buffer insertion, verification and buffer relocation optimization. The detail algorithm on each step will be discussed, with the goal to meet the timing and power constraints. Also, we will summarize the general characteristic of clock mesh. The experiment results show that our clock mesh synthesis and optimization framework performances well on standard academic testcases, meeting the strict timing and power constraints.

KEY WORDS: clock synthesis, clock mesh, local routing, buffer insertion, verification.

目 录

图 录

表 录

第一章 背景介绍

自从第一个晶体管问世以后，集成电路就以迅猛的态势在发展。英特尔的创始人 Moore 预言，每个 18-24 个月的时间，芯片单位面积上的器件数量会翻一番，成本降低一半。几十年来，集成电路一直按照摩尔定律在发展，如今芯片晶体管的数量已达到了千万甚至亿的数量级。集成电路已经从小规模集成电路和中规模集成电路（MSI），发展到大规模集成电路（LSI），直到今天的超大规模(VLSI)和特大规模(ULSI)阶段。相应的器件特征也在不断减少，当今主流的半导体工艺技术是 90 和 65 纳米，45 纳米技术也在英特尔的多核处理器中广泛应用，而且向着 32，22 纳米的技术不断发展前进。

1.1 超大规模集成电路的后端设计

超大规模集成电路的设计分为前端设计和后端设计。前端设计主要包括以下几个阶段：总体系统定义和要求，功能模块划分，行为级建模仿真，RTL（寄存器传输级）代码编写和综合，逻辑综合优化。这些步骤不是一次就能完成的，可能会在一个或者多个步骤中反复交替进行直到满足当前阶段的设计要求。后端设计，也叫物理设计，任务是把前端设计输出的电路网表转换为实际的可以由芯片制造商生产的电路。它是超大规模集成电路设计中最费时，最复杂的一步，主要包括一下步骤：模块划分，布图规划，标准单元布局，时钟网络综合，布线，时序和版图验证。

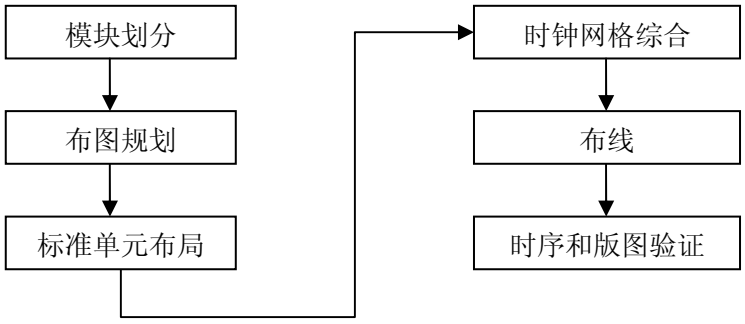


图 1-1 集成电路物理设计流程
Fig. 1-1 VLSI physical design flow

模块划分（partition）：把芯片按其功能划分为相对独立的模块，如数字处理模块，锁相环模块，射频模块，存储模块等，方便后面进行布图规划（floor planning）。

布图规划（floor planning）：把划分好模块根据空间，时序，功耗等约束条件摆放到相应的位置上，同时要为标准单元（standard cell）留出安置空间。如图1-2所示，靠右边的浅灰色背景的都是已放置好的功能模块。

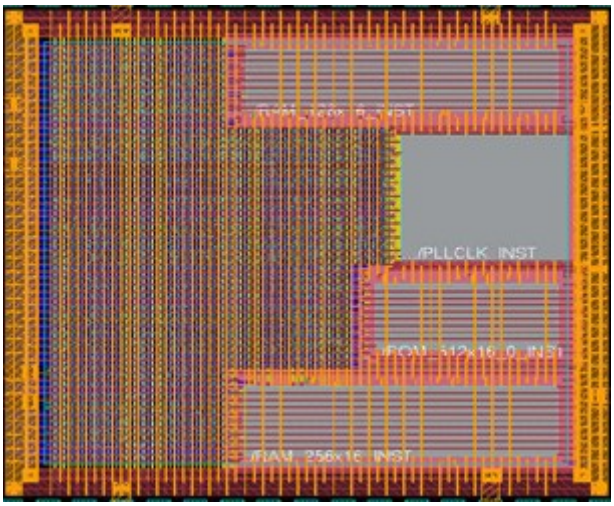


图 1-2 布图规划

Fig. 1-2 An example of floor planning

标准单元布局（placement）：在布图规划后剩下的空间中根据功能，时序，功耗等约束条件摆放标准单元，由于标准单元数量都是百万千万级别，所以这个阶段费时很久。如果最后布局结果达不到约束要求，就要重新做布图规划--标准单元布局这一流程。图1-3显示的标准单元布局的例子,其中浅蓝色边框的小块就是标准单元。

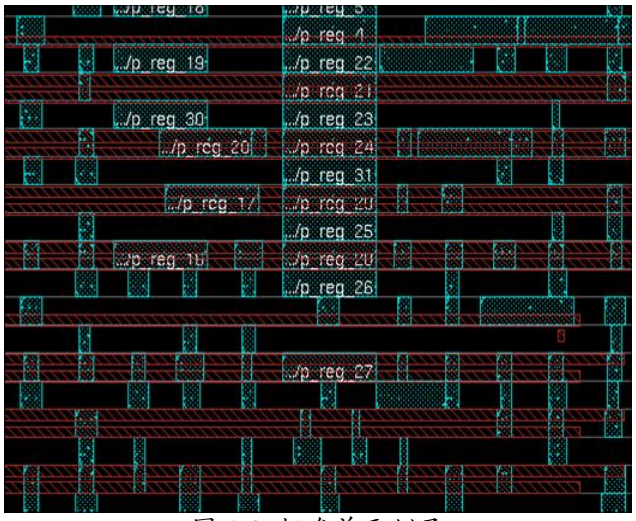


图 1-3 标准单元例子

Fig. 1-3 An example of standard placement

时钟网络综合：在同步设计中，为保证电路正常工作，要求时钟信号到达这些寄存器的时钟管脚的时间（delay）基本一致，为了达到这个要求，必须对时钟

网络进行仔细的设计和优化。时钟网络综合的输入是标准单元的完成布局后的物理位置信息，时钟综合的任务就是在已知的时钟管脚位置信息的情况下，综合出来的时钟网络满足设计约束条件，如时钟偏差，总线长，功耗等约束。时钟网络的综合本质上是布线问题的一种特例，它在一般布线之前，可见其优先级较高。

布线：时钟布线完成后，就要完成标准单元之间，模块之间，标准单元与模块之间的布线。一般而言，布线的目标是使所有布线的连线总长最短。对于当今高性能的芯片设计，需要最小化线网间的串扰、时延(或者通过最小化某些关键线网的连线长度从而最小化它们的时延)等性能，以实现整个芯片的性能优化目标。布线时，根据布线工艺的要求分为单层布线和多层布线，但不同线网不能在同一层相交。分布在不同层上的同一线网通过通孔(via)相连接。总体布线阶段把线网分配给布线区域，规定线网要经过的路线;详细布线阶段确定线网各个线段在布线区域中的具体位置，从而完成线网在布线区域的最后定位。如图 1-4 是布线后的示例[8]。绿线和红线分别表示出于不在同一层的线网。

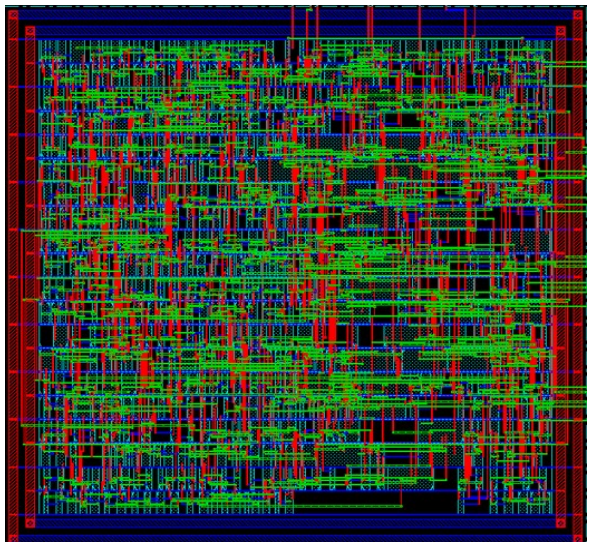


图 1-4 布线示例

Fig. 1-4 An example of routing

时序和版图验证：布线完成之后，因为相邻互连线之间有耦合电容和耦合电感，因此会对原本的电路特性产生较大的影响。时序验证就是要提取电路中的寄生参数，进行静态时许分析（STA）和后仿真，看此时还是否满足时序方面的要求约束。版图验证是指 1.设计规则的验证------看是否满足当前工艺下的设计规则要求；2. 形式验证（formal verification），进行逻辑形式和功能的一致性比较。

1.2 当今物理设计的挑战

集成电路发展到今天，芯片上的晶体管数量上亿计，特征尺寸的不断缩小导致各种复杂的物理效应干扰电路的正常功能。物理设计的任务就是要保证前端设计的电路功能可以被后端的制造厂商正确的制造出来，因此物理设计在整个芯片设计流程中占有举足轻重的地位，是连接前端设计和后端制造的桥梁。当前，物理设计面临的挑战主要来自两方面。

1.2.1 对工具的挑战

面对如此庞大复杂的后端设计任务，没有自动化工具支持那是不可能完成的任务，譬如几百万个线网的布线问题，人工根本无法完成。但是，随着晶体管集成度的不断提高，也给工具提出了越来越多的要求。其一，在保证正确的前提下，要工具提高运行速度，缩短设计周期，准确性和运行时间是一对不可调和的矛盾；其二，对于具体的问题，如何综合考虑各种因素的相互制约，相互平衡，工具需提供多种的可配置性。如功耗和电路速度，两者此消彼长，需根据设计要求进行取舍。

1.2.2 对工程师的挑战

虽然工具给了后端设计工程师巨大的帮助，但工程师的地位是不可取代的。甚至可以这么说，工具的不断进步给工程师提出了更高的要求。后端工程师不仅需要深入了解电路特性，熟悉后端设计的流程；为了能更好地应用工具解决实际问题，工程师有时还有必要了解工具背后的算法，以便出现问题时能判断是电路本身的问题还工具本身的算法对这个问题存在缺陷。熟悉工具背后的算法还有利用工程师把实际问题跟程序抽象描述联系起来，反过来可以促进工程师对工具提出更多的要求。

1.3 时钟综合在物理设计中的重要性

现今大规模集成电路设计中，绝大部分都是同步电路，为保证电路正常工作，要求时钟信号到达这些寄存器的时钟管脚的时间（delay）基本一致，即时钟偏差（clock skew）很小。时钟网络的设计质量关系到整个芯片的工作频率，同时也对整个芯片的功耗[1]，面积等方面都有重大的影响，因此时钟网络的设计是后端物理设计的核心环节之一，具有很高的优先级。由图 1-1 可以看到，时钟网络综合

在一般的信号布线之前，因此时钟布线比一般的信号布线具有更高的优先级。

在大规模的设计中，时钟信号需要驱动上百万的寄存器，连线很长，负载也很大，时钟网络综合通常通过插入大量的缓冲器来减少时钟延迟和平衡时钟偏差，因此必须借助于EDA工具完成时钟网络的综合工作。当今主流的后端工具，如Synopsys 的Astro，IC Compiler，Cadence的Encounter等都对时钟网络的综合有很好的支持。由于每个设计的具体要求的差别，工程师往往需要在工具给出的综合结果上再根据实际需要对时钟网络做局部的调整，如增加或减少缓冲器，移动缓冲器的位置，改变连线宽度等，以满足设计要求。但是，由于工艺，温度和电压的不确定性的影响[2][3][4]，要达到时钟偏差在一定的范围内这个目标变得越来越困难，这给全局时钟网络的设计带来了很大的挑战。

在高性能大规模的芯片设计中，对时钟网络的综合提出了更高的要求，时钟频率要高，连移动设备CPU的频率都到达G-MHZ级别；时钟偏差要小，G-MHZ的设计要求时钟偏差控制在 150ps内；功耗越低越好，时钟网络的总功耗已占到了整个芯片功耗的 40%以上；面积越少越好，时钟网络遍布整个芯片，占用了大量的空间资源；拓扑结构越来越复杂，从H树型，二叉树型到网格型；时钟网络设计已经成为物理设计的一个重点和难点[5][6][7]。

第二章 时钟综合简介

2.1 时钟综合中的基本概念

在现今大规模的集成电路中，同步时序电路设计是主流[9]。在同步时序电路中，各寄存器由全局共同的时钟信号控制，只有时钟信号达到时，寄存器的状态才会改变。在时钟信号没有到达时，输入信号的改变不会引起寄存器状态的改变。理想状态下，时钟信号达到各寄存器的时间是完全一样的，但实际上由于连线长度，负载大小的差异，时钟信号到达各寄存器的时刻并不是完全一样的。这不仅会降低系统的工作频率，甚至会导致逻辑错误。下面介绍时钟网络方面几个重要的概念，以便后阐述需要。

2.1.1 数据通路(data path)

在同步电路中，时钟信号是统一的，即只有一个公共的时钟信号，电路中的寄存器时钟输入端都是连接到这个统一的时钟上的，有它控制。只有当时钟信号的上升沿或下降沿到达时，寄存器的状态才会发生改变。数据通路指的是数据在时钟信号控制下，在寄存器间的传播路径。图 2-1 是数据通路的一个简单示例。

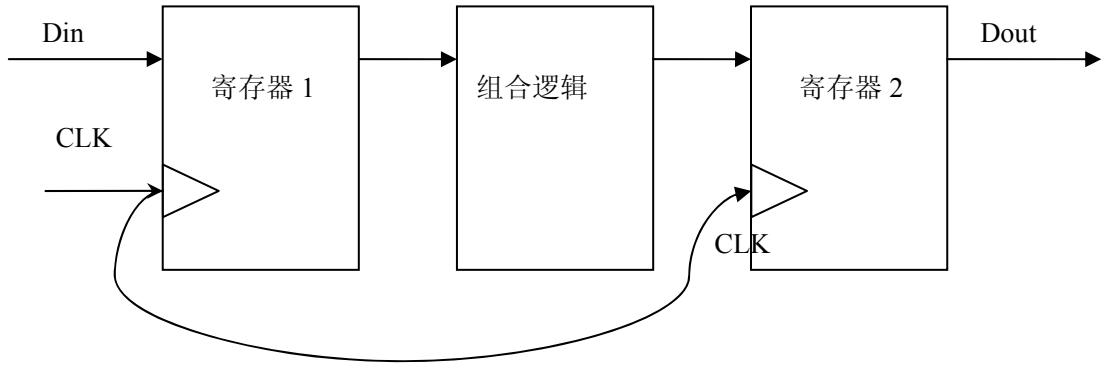


图 2-1 数据通路示意图
Fig. 2-1 An example of data path

当时钟信号到达时，数据信号从寄存器 1 向寄存器 2 传播，中间经过组合逻辑器件。为确保数据能被寄存器 2 正确存贮，数据必须要在下一个时钟到达前到达寄存器 2 的输入端。芯片允许的最大时钟频率f_{clk_max}是由所有相邻两个寄存器间最大的数据传播时间T_{max}决定的。

$$f_{CLK\_MAX} = \frac{1}{T_{\max}} \qquad (2\text{-}1)$$

2.1.2 时钟延迟(clock delay)

即时钟信号从时钟源点到达寄存器的时钟输入端口的时间称为时钟延时[10]。由于时钟源的与寄存器之间存在距离，信号经过互连线会产生延迟，这种延迟是由于互连线上的电阻和电容造成的。时钟延迟的测量方法一般是取在时钟信号在源点和终点 50%幅值的时刻点。如图 2-1 所示。

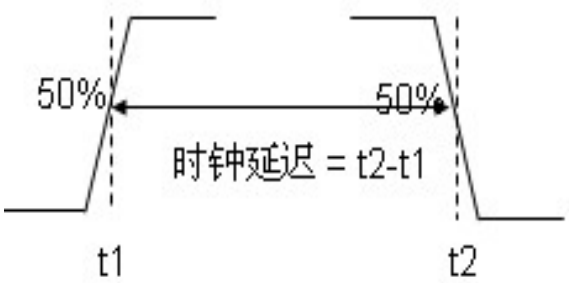


图 2-1 时钟延迟计算方法
Fig. Calculation of clock delay

2.1.3 时钟偏差(clock skew)

在理想状态下，从时钟源到达每个需要同步的寄存器时钟管脚的时钟延迟应该是一样的。但在实际的设计中，从时钟信号源到每个寄存器时钟管脚的时钟延迟(clock delay) 是不一样到的，这种时钟延迟的差别称为时钟偏差[6,7,11]。图 2-2 直观地给出了时钟偏差的定义和计算方法，clock skew= clock delay_1 －clock delay_2。

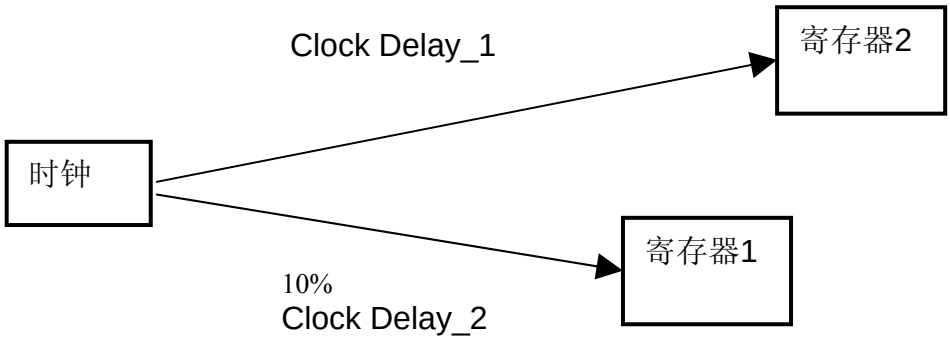


图 2-2 时钟偏差计算方法
Fig. 2-2 Calculation of clock skew

由于多种原因，不同的时钟路径到达不同的寄存器的时钟延迟是不一样，时钟偏差的产生有以下原因[12]：

1) 从时钟源到寄存器的互连线长度的差别，互连线长度的差别导致信号经过连线的延迟不一样。

2) 从时钟源到寄存器经过的缓冲器的数目和大小不一样

3) 由于工艺不确定引起互连线的电阻率，厚度，通孔（via）电阻，介电常数，有源器件中 MOS 管的阈值电压，电子迁移率不一致，导致信号经过时引起不同的延迟。

值得注意的是，虽然多种因素影响着时钟偏差，但在一个设计良好的时钟网络中，时钟网络中的缓冲器是时钟偏差的主要诱因。

在时钟网络综合，时钟延时是评估时钟网络质量的重要指标。在时钟网络综合种，增大时钟延迟可以间接减少时钟偏差，从而更容易满足设计要求。但增大时钟延时意味着更长的传播路径和需要再传播路径上插入更多的缓冲器，而这两个直接导致面积和功耗的增加。所以时钟偏差和时钟延迟也是一对矛盾，需要根据具体的设计进行不同的取舍。如果设计的时钟偏差要求比较宽松，则应尽量减少时钟延时，达到减少面积和功耗的目标。反之，在高速集成电路设计中，时钟频率高，时钟偏差要求严格，则应首先满足时钟偏差的约束要求。

时钟偏差是不可避免的，但使其控制在合理的范围内是后端工程师必须要做的。最有效的减少时钟偏差的办法是通过合理设计时钟网络的拓扑结构，调整时钟网络中缓冲器的位置和大小，调整互连线的宽度等。

2.1.4 时钟抖动(clock jitter)

时钟抖动是指相对于理想的要求到达时间，时钟时而超前时而滞后的偏移。时钟抖动会引起电路时序的不确定性，甚至违反时序要求。因此对时钟抖动有一个准确的估量，那对设计师是很有作用的。时钟抖动的产生原因有很多种，除了锁相环，晶振本身外，信号的传送，晶振的随机机械震动，接收器，连接等，都可能会引入抖动[12]。

2.1.5 过渡时间(transition time)

时钟信号从 10%-90%的上升或下降时间。由于时钟信号有负载，所以其从低电位转到高电位或者从高电位转到低电位是需要时间的。过渡时间也是衡量时钟

网络综合质量的一个标准，如果时钟信号的过渡时间太长，寄存器根本无法再规定的时间内完成状态转换。过渡时间的长短也表明的缓冲器的驱动能力，驱动能力越强，过渡时间越短，反之驱动力越弱，过渡时间越长。图 2-3 直观地给出了时钟偏差的定义和计算方法。

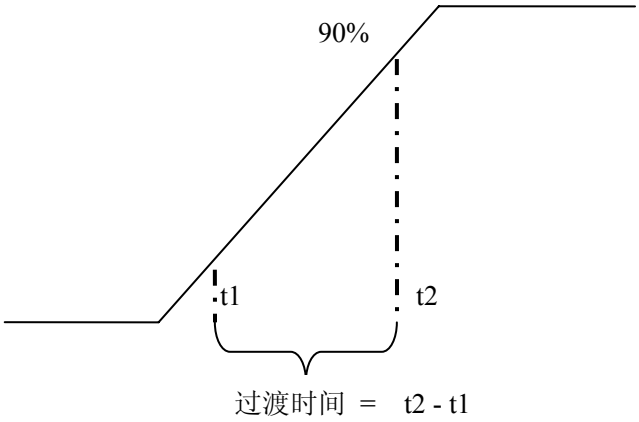


图 2-3 过渡时间的计算方法
Fig. An example of transition time calculation

2.2 时钟网络的分类

现今，针对复杂高速庞大的集成电路设计，要设计一个健壮的时钟网络同步寄存器显得越来越艰难。而且，“同步”仅仅只是其中一个指标，从整个系统设计的角度，时钟网络的设计还要考虑到时钟频率，占用的芯片面积和功耗等，特别是低功耗设计，是当今时钟网络设计的热门话题。各种各样的时钟网络拓扑结构也已被成功地广泛应用,有传统对称的 H 树结构，主流的缓冲器二叉树结构，新兴的网格型结构。下面分别对这种结构进行阐述。

2.2.1 H 树形结构

H 树型时钟分布的目标是时钟信号同时到达寄存器的时钟管脚，即 clock skew 等于零。如图 2-4 所示是典型的 H 树型时钟结构。

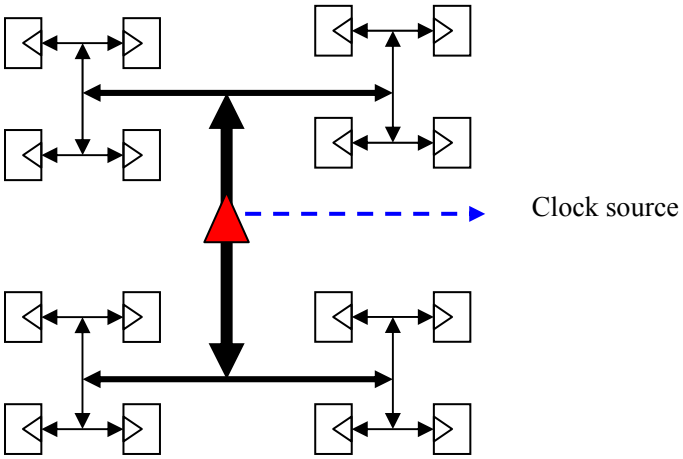


图 2-4 H 型时钟树
Fig. An example of H tree

在H树形结构中，时钟源或者最主要的缓冲器，是放在H型的中心位置，如图2-4 中红色三角形所示，时钟信号首先传输到H型的四个角上，这四个角上的时钟信号作为下一级H树的输入，如此类推，时钟信号可能经过多级的H树才能到达最后一级，最后一级作为局部区域内寄存器的输入缓冲器。由于这种对称的结构，保证了从时钟源到最后一级的所有的寄存的连线长度是相等的。因此在理想情况下，从时钟源到每个寄存器的延迟是相等的，时钟偏差也就等于零[14][15]。考虑到实际情况，H树形结构中的时钟偏差主要是由于工艺不确定性引起的。

从图 2-4 还可以看到H树形结构中的互连线宽度从时钟源到下一级逐步缩少，这样做的目的在于减少高速的时钟信号在转角点的反射效应[16]。

虽然H树形结构优美简洁，但却对时钟布线提出了较高的要求，因为为了减少时钟延迟，水平和垂直的互连线只能处于相邻的布线层，灵活性较低。而且，在实际应用中，寄存器的位置在整个芯片上的分布是没有规律可言的[17]，纯粹使用H树形结构进行时钟网络设计几乎很少见。

2.2.2 二叉树结构

二叉树型结构是目前最流行的时钟分布结构。如图 2-5 所示：

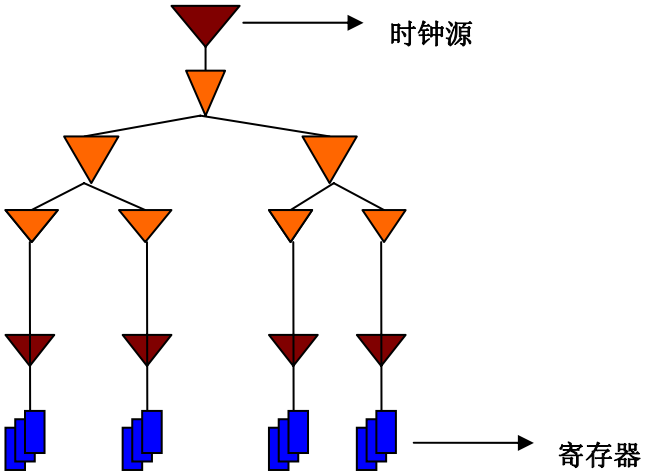


图 2-5 二叉树型时钟树

Fig. An example of buffered clock tree

时钟源作为整棵树的根节点，底下的每一级缓冲器作为时钟树的分支，寄存器是整棵树的叶子节点。设置多级缓冲器的作用主要有两方面：1.时钟信号在互连线上传播时由于互连线上存在电阻电容，会使信号产生衰减，插入缓冲器可以起到恢复信号强度的作用；2. 缓冲器可以起到隔绝前级跟后级的作用，在前一级的时钟信号不会受到缓冲器后面负载的影响[18]。树的级数（即缓冲器的级数）和每一级缓冲器的数量是由寄存器时钟输入端负载之和，互连线的负载和允许的时钟偏差的值决定的[19]。每个缓冲器能驱动多少个下一级的负载则是由该缓冲器的驱动能力及下一级缓冲器的输入负载决定的。

在二叉树型时钟树结构的优点是有利于控制整个芯片的时钟偏差以及减小了时钟源到寄存器的时钟延迟。这是因为互连线上的电容电阻被缓冲器分隔开，缓冲器起到了隔绝的前级和后级的作用。同时缓冲器的插入还有利于恢复时钟信号在互连线上传输过程中引起的衰减，使时钟信号保持较短的上升和下降时间，从而能有效改善过渡时间。但在二叉树型时钟树结构的缺点也是显而易见的，因为晶体管的电导对工艺、电源电压和环境参数等十分敏感，所以实际做出来的时钟网络的效果跟仿真的结果可能会有较大的偏差

2.2.3 网格型结构

网格型时钟（clock mesh）由于其天生的冗余结构，能获得很强的抗干扰能力和很小的时钟偏差，因此被广泛应用于高性能的芯片设计[10-12]。图 2-6 显示的是典型H树+网格型时钟结构。

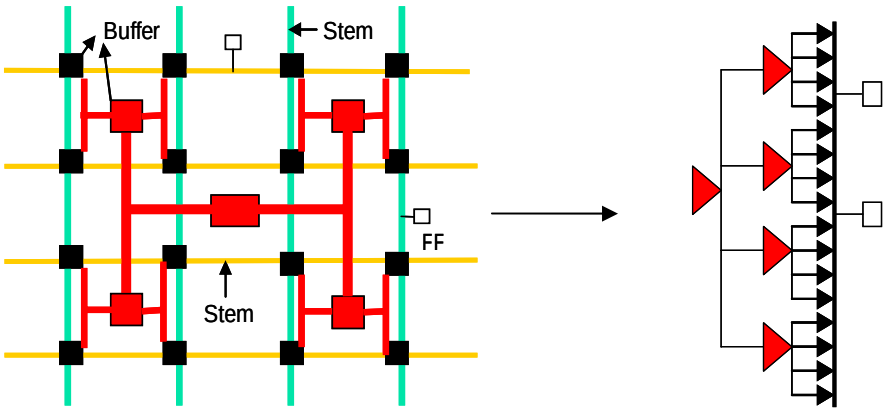


图 2-6 H 树+网格型时钟
Fig. 2-6 H tree and Mesh structure

图 2-6 中红色结构是 H 树，青色和黄色纵横交错的树干（stem）是网格型时钟的金属线，它们在交叉处是通过通孔（via）连通的。黑色的小方框是 H 树最后一级直接驱动时钟网格的缓冲器（buffer）。它们通常处在树干的交叉点上。位于中心位置的红色方块是 H 树的根（时钟源或者是最顶层的驱动缓冲器）。位于其四周的四个红色小方块是第二级时钟驱动缓冲器。白色的小方块是寄存器（FF），它们通过互连线之间连接到树干上（stem）。

图 2-7 是另外一个时钟网格的例子[23]。它是顶层二叉树加底层网格的结构。

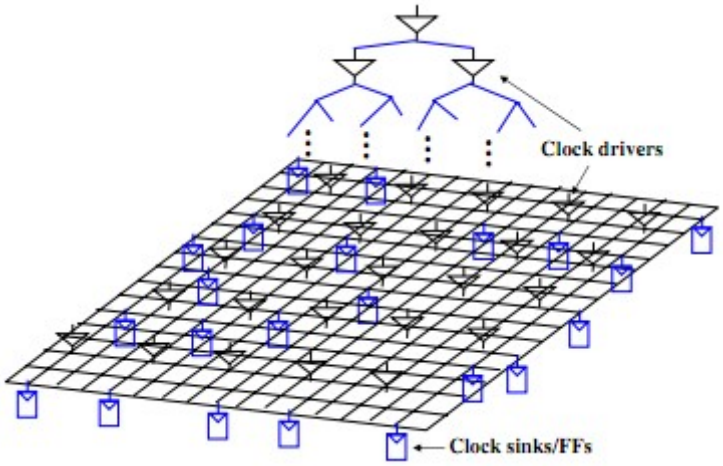


图 2-7 二叉树+网格型时钟
Fig. 2-7 buffered clock tree and Mesh structure

图 2-7 是一个立体的时钟网格分布图。时钟信号首先通过二叉树的多级缓冲器分支到达网格型结构，然后利用网格型结构冗余特性，使通过不同路径到达网格的时钟信号尽量在此“统一”起来，减少时钟偏差。

2.3 低功耗时钟网络设计

低功耗设计已成为贯穿整个集成电路设计流程的重要理念。无论出于哪个设计阶段，都要考虑到功耗这一重要指标，时钟网络设计也不例外。因此这一小节作个简单介绍。

在现今的超大规模集成电路设计中，时钟网络需要驱动上千万的寄存器，即驱动很大的电容负载。时钟信号在每个时钟周期内需要改变两次状态，意味着在一个时钟周期内需要给负载电容充放电各一次。随着对时钟频率的要求越来越高，大大增加了时钟网络消耗的功耗，据统计，时钟网络消耗的功率占到整个芯片消耗功率的 40%[25]以上。

在集成电路中，主要的功耗消耗是在动态功耗上。因此可以通过降低时钟频率的方法来降低功耗，但这个要求跟需要越来越高的时钟频率是相矛盾的。另一个显而易见的方法是减少时钟网络的电容复杂，譬如减少互连线的长度，选择不同的材料改变其介电常数，减少互连线之间的耦合电容等。

目前得到广泛应用并且效果显著的降低时钟网络功耗的方法是采用门控时钟（clock gating），门控时钟是指在原来的电路上增加额外的控制逻辑，使某一部分电路功能模块不需要工作的时候，停止其相应的控制时钟，这部分时钟停止后，就没有原来状态转换时消耗的功耗。图 2-8 是门控时钟的一个简单示意图。

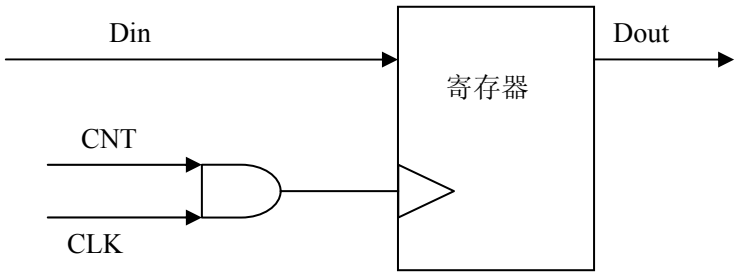


图 2-8 门控时钟示例

Fig. 2-8 An example of clock gating

时钟信号 CLK 不是直接连接到寄存器的时钟输入端，而是连同一个控制信号 CNT 经过一个与门后才作为寄存器的时钟输入，当 CNT 为“1”时，与门的输出为 CLK，寄存器正常工作；当 CNT 为“0”时，与门的输出为“0”，寄存器处于不工作状态。因此当不需要寄存器工作时，把 CNT 设置为“0”，CLK 在寄存器的时钟输入端就不会发生状态转变，从而减少了功耗。

2.4 网格型时钟分布的优缺点

网格型时钟分布相对树形时钟网络来说，优点与缺点并存。因此在设计时要根据实际需要决定用树形结构还是网格型结构作为时钟分布网络。下面是网格型时钟与树形时钟网络的对比。

1．时钟偏差

网格型时钟相对树形结构来说通常会有获得较少的时钟偏差[24,25]。这是由于网格型时钟“冗余”的连线获得的。然而，如果某些叶子节点需要不同的时钟到达时间（如某些宏单元内部有固定的时间延迟），单纯的网格型时钟没法解决这个问题。它的结构决定了它不能直接延迟或提早某些节点的到达时间。

2．抗干扰的能力

网格型时钟相对树形时钟结构来说，其抵抗芯片级不确定性和工艺不确定的能力要强很多。这也是其“冗余”的连线的直接结果。

3．时钟延迟

由于在网格型时钟上有多个缓冲器同时驱动下一级负载，因此可以比时钟树获得更少的时钟延迟。

如果时钟树综合能满足时钟偏差和时钟延迟的性能要求，那么没必要仅仅为了预防工艺的不确定造成的影响而是用网格型时钟。时钟树综合目前自动化程度比网格型时钟高很多，而且功耗低。但是如果时钟树结构不能满足性能要求，则应考虑使用网格型时钟。

4. 功耗

网格型时钟消耗的功耗通常来说要比树形结构的多，因为“冗余”的互联必然造成功耗的增加。但在某些情况下，网格型时钟可能比时钟树消耗更低的功耗。因为绝大部分的功耗都是消耗在最底一级(寄存器时钟输入端)，所以使用局部的分布能减少时钟树跟网格型时钟在功耗上的差别。

5. 布线上的限制

从结构上不能看出，对称性越强，网格型时钟越能获得较小的时钟偏差。但在实际设计中，有些芯片的布局是很不均匀的，再加上芯片上已经存在的电源线和布线障碍区，可能会出现无法实现网格型时钟分布的情况。相反，时钟树却不受到这方面的限制，走线相当灵活。

6. 自动化程度

就当前来说，时钟树综合已经是一个很成熟的领域，自动化程度相当的高，在商业化工具中，只要给出一定的设计约束条件，它就能自动综合出满足要求的时钟网络。但是网格型时钟的自动化程度就相对较低，很多工具还不支持这样的时钟网络设计，即便支持，设计者也要给出具体的网格结构，如水平和垂直的树干数目，位置，缓冲器的数目等，自动化程度相对较低。

7. 门控时钟

对于时钟树来说，引入门控时钟是十分方便的，但对于网格型结构来说，由于其最后一级就是一个通过多个缓冲器驱动的巨大线网，无法引入门控时钟，只能在其上级或者更下一级的树结构中引入门控时钟。

2.5 本章小结

本章首先介绍了跟时钟网络，时钟综合相关的一些基本概念：数据通路，时钟延迟，时钟偏差，时钟抖动，过渡时间。然后对当前流行的时钟网络结构进行了简介，它们是 H 树形结构，二叉树型结构，网格型结构。同时简单介绍了通过门控时钟进行低功耗设计的基本概念。最后，总结了用网格型时钟作为时钟分布的优缺点，分析了其适用场合和应用领域。

第三章 网格型时钟综合流程

3.1 前人的研究成果

网格型时钟由于抗干扰能力强，时钟偏差小的特点，越来越受到工业界的关注，网格型时钟分布已经成功应用于大型的芯片设计，如 600-MH Alpha[20]； IBM G5 S/930[21]； Power4[22] 和Power PC[26]; SUN Sparc V9[27]。 网格型时钟分布也成为EDA领域的一个研究热点，关于网格型时钟的分析，仿真，综合和优化的文章频繁出现在EDA领域具有影响力的会议和期刊。

3.1.1 仿真分析

对整个网格型时钟进行传统的电路仿真获得结果数据时是相当困难的，因为网格型时钟结构相当庞大，电路节点上百万级。传统的电路仿真器，如 Hspice，Hsim 根本无法对如此规模的电路进行仿真。为此，迫切需要新方法对网格型时钟进行仿真分析。

1. 富士康H. Chen 等人首先提出Sliding Wsindow Scheme（SWS）[28]方法，他们考虑到网格型时钟低通滤波器的特性[24]，即时钟信号在RC网络中传播时，其幅值随着其传播距离成指数级衰减，所以需要获取某些寄存器的仿真结果，只需考虑其附近区域的寄存器和时钟信号输入即可，因为距离远的器件对仿真结果影响极小。H. Chen把这个区域定义窗口（window），窗口内的是需要考虑的元素，窗口外的元素则可以忽略不计。如图 3-1 所示是一个网格型时钟里面的窗口示例。

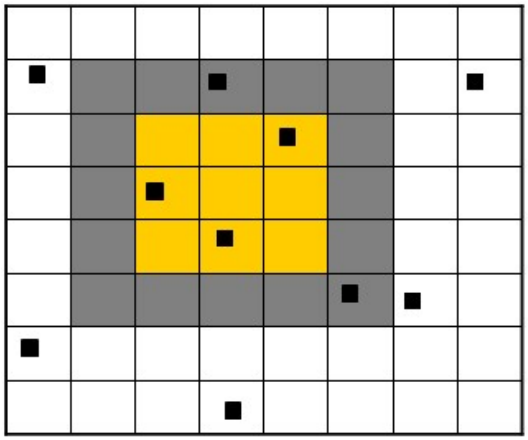


图 3-1 网格中窗口示例
Fig. 3-1 A slide window

图 3-1 中黑色小方框是寄存器，具有背景颜色的是一个窗口，其中黄色背景里面寄存器是要获取仿真结果的寄存器，黄色背景外面一圈灰色作为“缓冲带”，里面的元素也是要考虑的，但不会获取他们的仿真结果。如此，通过移动“窗口”到网格型时钟的不同位置，就能完成就整个网格型时钟的仿真。因此每次仿真只是整个网格型时钟的一小部分，电路节点较少，仿真很快就能完成，而且每个窗口之间是相对独立的，仿真还可以并行的进行。实验结果表明，SWS 方法不仅能对目前商业仿真器无法仿真大型的时钟网格进行仿真，而且精度相当高，跟 Hspice 的结果只有 1%内的差别。

2．P. Li 等人综合利用模型降阶（Model Order Reduction）[29]和电路仿真技巧来处理大规模网格型时钟的仿真问题[30]。首先，他们把网格型时钟的时域仿真结果中时钟偏差和过渡时间等重要因素映射频域的谐波成分，然后通过一般的模型降阶技巧获取降阶后的模型；第二个技巧是端口移动（port sliding），也是利用了网格型时钟“附近影响”原则。在接下来的步骤中，网格型时钟输入驱动点的波形通过FFT转换到频域，传输到寄存器时钟输入端在进行IFFT变换转换为时域，如此完成仿真过程。

3.X. Ye等人在进行网格型时钟电路的瞬态分析时，把电路分为线性部分和非线性部分[18]，如图 3-2 所示。通过合理的构造和划分，可以把线性部分的电路映射成对称正定矩阵，对于正定对称矩阵的求解，通常要比一般的矩阵求解快好几倍[31]。当线性部分完成计算后就可以作为一个黑盒子跟非线性模块一起进行仿真。由于网格型时钟电路中线性部分占了主导地位，因此通过把线性部分跟非线性部分进行分离，加速线性部分的仿真时间，就能提高仿真速度。同时结合矩阵分块和动态变步长的方法，可以对电路进行并行的仿真，是网格型时钟的瞬态分析也成为可能。

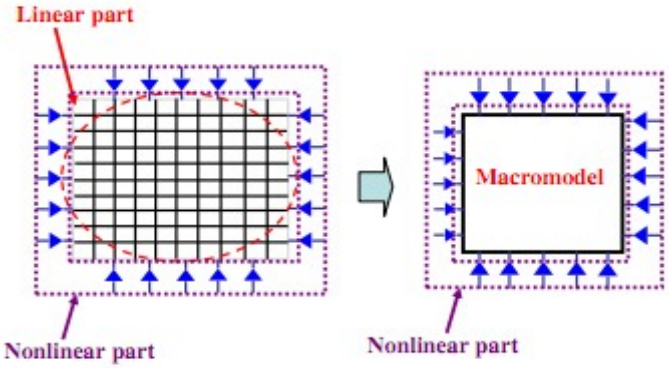


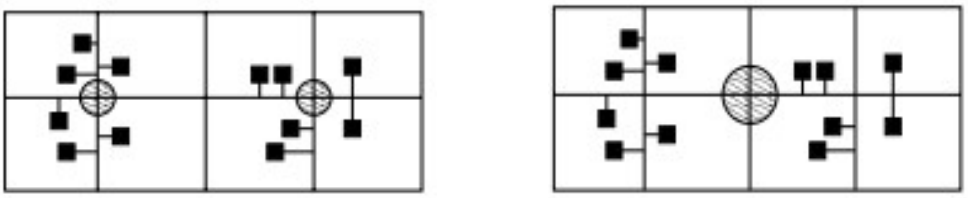
图 3-2 电路划分为线性和非线性部分
Fig. 3-2 Linear part and Nonlinear part partition

3.1.2 综合优化

1.A. Rajaram 等人第一次提出了网格型时钟的整个自动化综合和优化流程[31]，主要包括网格规划和综合，缓冲器插入，网格结构优化。

在网格规划和综合阶段，根据寄存器在芯片上的分布，互连线参数，缓冲器模型对时钟偏差的进行预估。根据作者的分析观察，网格型时钟中的纵横金属连线的数目跟时钟偏差存在一个函数关系，因此给定了时钟偏差的约束，就能确定网格型时钟水平和垂直的金属线数量。

在缓冲器插入阶段，作者改善了[32]中 set-cover 的方法，把网格型时钟低通滤波器的特性和缓冲器驱动能力大小也考虑到代价函数中，从而使缓冲器类型和位置的选择更合理。图 3-3(a) 显示的是用[32]中代价函数综合出来的缓冲器的位置和大小；图 3-3(b) 显示的是用优化的代价函数后综合出来的缓冲器的位置和大小。不难看出，优化后，用一个居中的更大的缓冲器替代原来两个位于左右两个的小缓冲器。



a)优化前的缓冲器大小及位置　　b) 优化后的缓冲器大小及位置

图 3-3 代价函数优化前后对比
Fig. 3-3 Comparison between before and after optimization

在网格结构优化阶段，作者把一些“冗余”的网格边去掉，“冗余”边是指去掉这些边后，对网格的抗干扰能力影响不大的边，去掉“冗余”边能减少缓冲器负载，从而降低网格型时钟的功耗。文中把“边对时钟延迟的敏感性”作为衡量“冗余”的指标，通过延迟对线宽求导获取这一数值，然后按比例去掉冗余边。如图 3-4 所示，虚线部分就是去掉的“冗余”的边。

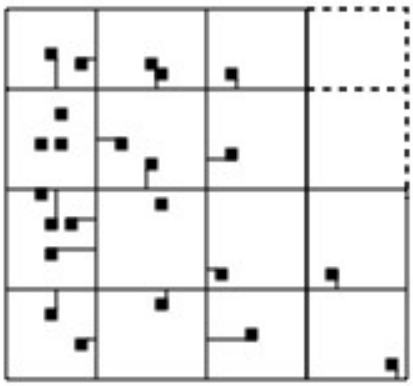


图 3-4 去除“冗余”边
Fig. Mesh optimization

2.G. Venkataraman 首先提出了用set-cover的方法来确定缓冲器位置和大小[32]，文中给每个缓冲器定义了一个“覆盖区域”(Covering Region)，该区域内的总电容（包括寄存器时钟输入端电容和互连线）不能超过该缓冲器能驱动的总电容。

于是，缓冲器插入的问题就转换为如何选择合适的缓冲器，使其能覆盖掉所有的“覆盖区域”，每个寄存器都能被至少一个缓冲器覆盖，同时保证缓冲器的总面积最小。传统的set-cover是NP问题，作者使用贪婪算法（Greedy algorithm）[33]保证在丢失较少精度的情况下大大提高算法运行速度。

文中还把去除网格型时钟中“冗余”边的问题抽象成生存网络（survivable network）[34]的问题。生存网络求解也是一个NP问题，作者也是使用了启发式的近似算法加速。同时读者可以通过设置k和l参数来控制冗余度和功耗。k和l越大，表示去除的“冗余”边数量少，此时网格型时钟抗干扰能力强，但功耗较大；相反，设置较少的k和l，则能去除较多的“冗余”边，降低的网格型时钟的抗干扰能力，却能减少功耗。因此，可以根据实际需要，灵活地设置k和l的值，达到抗干扰能力和功耗的折中。

为了进一步提高仿真速度，文中还对缓冲器的驱动模型进行了简化，在实际的网格型时钟结构中，缓冲器的行为是相当复杂的。如图 3-5 所示，四个缓冲器同时驱动时钟网格，每个缓冲器都受到其它缓冲器的相互影响，寄存器时钟输入端的时钟信号时多个缓冲器综合作用的结果。实际上，这个一个复杂的非线性耦合过程。作者通过 Hspice 仿真，提取相应参数，给出了一个简化的缓冲器模型，大大提高了仿真速度。

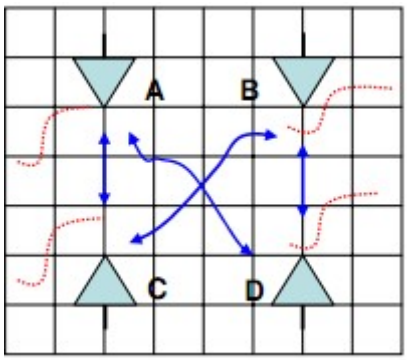


图 3-5 网格型时钟缓冲器相互作用
Fig.3-5 Interaction between buffers

实验结果表明，通过这些组合算法，互连线总长度减少了 29%，功耗降低 28%，但仅仅增加了 4%的时钟延迟的代价。

3.2 网格型时钟自动化综合框架

现今著名的后端设计工具如 Synopsys 的 IC Compiler, Cadence 的 Encounter 都不能实现网格型时钟的完全自动化综合。即只需给出时钟偏差，时钟延迟功耗等约束条件，自动综合处满足设计要求的时钟网络分布。设计者需要根据经验给出网格型时钟的初始构成（给出水平和垂直的金属连线的数量），这对经验不足的设计者造成了很大的不便。但对时钟树而言，自动化综合流程已相当成熟。

自动化网格型时钟的综合和优化已成为时钟网络设计的热点和难点，最新的研究成果也在 3.1 节中做了简单的介绍陈述，但显而易见，它们也存在不足和有待改进的地方：

1． [31]中决定网格型时钟初始的水平和垂直的互连线（树干）的数量的算法是基于对全局时钟偏差的一个估计，　但在网格型时钟中由于其错综复杂的结构，在不做任何仿真的情况下，对时钟偏差的估计肯定是相当粗糙的，计算出来的互连线数量也是不准确的，这必然会对后面的综合步骤造成影响。

2． [31,32] 都只是把水平和垂直互连线（树干）平均地放置在整个芯片上，而没有考虑实际中寄存器的分布。一个好的摆放策略应该是考虑到芯片上寄存器的实际位置，通过互连线的摆放位置能减少时钟偏差。

3. 在布线阶段，[31,32]的布线原则都是简单的“最近原则”，即寄存器时钟输入端连接到距离器最近的时钟网格的互连线上。固然，这种简单的连线规则能保

证连线总长度最短，减少了布线资源；但由于寄存器分布的不均匀性，这个原则会导致某些连线很有很多寄存器连接到其上面去，负载过重；而在寄存器稀疏的地方，网格型时钟的连线上寄存器数量很少，负载小；负载的不均匀很导致较大的时钟偏差。较好的布线策略应该是是网格型时钟每根互连线的负载尽可能地平均。

针对以上缺点和不足，本文在解决上述问题的基础上，提出全新的网格型时钟自动化综合流程，在满足时钟延迟和功耗约束的条件下，使时钟偏差尽可能地小，并给出了关于缓冲器位置优化的方案，这在文献中是从没出现过的。本研究工作的创新点在于：

1. 提出了一种新的决定初始的水平和垂直的互连线（树干）数目的方法，该方法基于缓冲器“平均延时—平均负载”的延迟曲线。

2．水平和垂直的互连线的位置放置考虑了芯片上寄存器的实际分布。

3．在布线阶段，不同于传统的“最近原则”连线规则，开发一个全新的局部布线器，使网格型时钟上每根互连线上的负载尽量平衡。

4．通过后仿真，根据仿真结果对缓冲器进行重定位，即改变缓冲器的位置，减少全局的时钟偏差。

本文提出的网格型时钟自动化综合流程如图 3-6 所示。

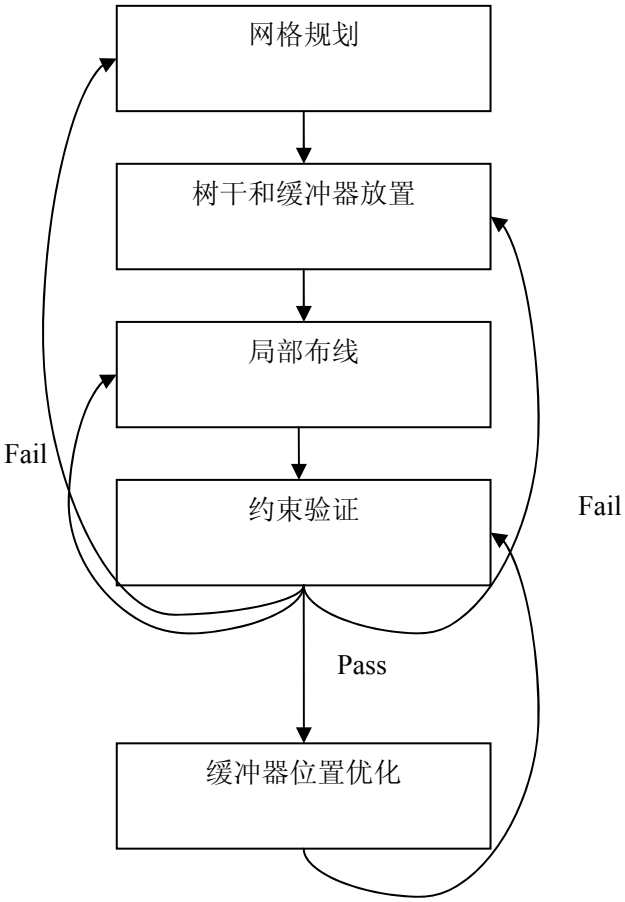


图 3-6 网格型时钟自动化综合流程
Fig. 3-6 The automatic clock mesh synthesis flow

1. 网格规划（mesh planning）这个阶段根据设计约束要求和芯片上可用的空间资源决定整个时钟网格中树干的数量。

2. 树干和缓冲器摆放（stem and buffer placement）根据一定的算法把树干放在合理的位置上，缓冲器的初始位置由[32]中的 set-cover 算法给出。

3. 局部布线（local routing）考虑到树干负载平衡原则，把寄存器时钟输入端连接到树干上。

4. 约束验证（verification） 把此时物理结构的时钟网络转换为电路网表，通过 Hspice 仿真，看是否满足约束要求，如果不满足，则需要回到 1-3 的步骤重做。

5. 缓冲器重定位（buffer relocation）通过启发式算法不断调整缓冲器的位置，

直到时钟偏差在降到较低的值。此时时钟偏差的值是通过 SPICE 仿真获得的。

在下面的小节中，我们将会对上面的每个步骤进行详细的解述。

3.3 概念和标记

为方便后面小节对上述步骤的详细描述，我们首先需要定义一些概念和标记。

Zone(带状区)：由相邻的两根垂直树干或者水平树干包围的区域定义为带状区。如图 3-7 所示，黄色背景的是一个水平带状区，浅蓝色背景的是一个垂直带状区。

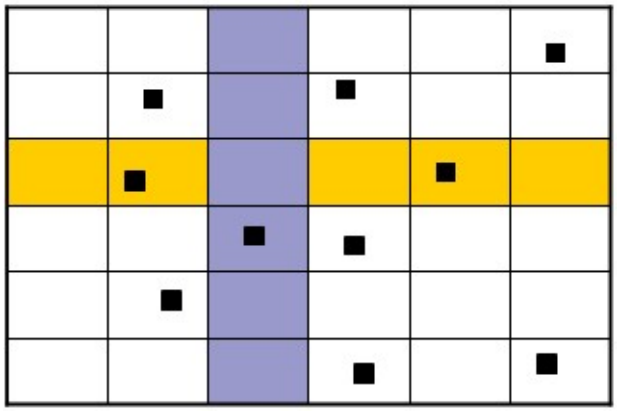


图 3-7 带状区示例
Fig. An example of zone

$H_{zone}(i)$: 第i个水平Zone的高度，即水平相邻的两个树干的距离。

$W_{zone}(j)$: 第j个垂直Zone的宽度，即垂直相邻的两个树干的距离。

L_{est}: L_{est} 是指预估的局部连线的长度。如何估计这个连线的长度，以图 3-8 加以说明。

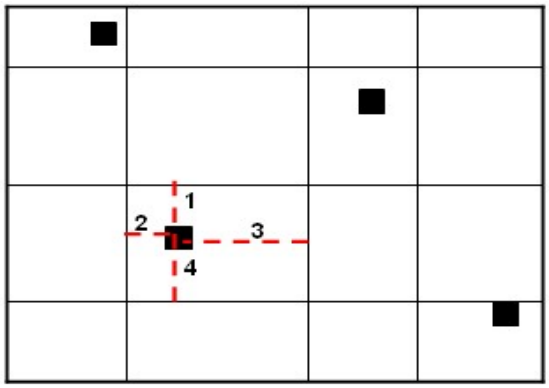


图 3-8 局部布线长度预估
Fig.3-8 Estimation of local wire length.

从图 3-8 可以看到每个寄存器通常都被四根树干包围，也就说在局部布线中，它可能连接到这四根中的任意一根，如图中红色虚线所示是其可能的四根布线。在这里，我们假设是按照“最近原则”连线，所以连线的最大长度不会超过$H_{zone}(i)/2$和$W_{zone}(j)/2$ 中的最大者，最小长度为 0。从平均的角度看，不难得到：

$$L_{est} = \min \{H_{zone}(i), W_{zone}(j), \forall i, j\}/4 \quad (3\text{-}1)$$

C_{total}: 整个网格型时钟的总电容，包括所有树干的总电容，局部连线的总电容和所有寄存器时钟输入端的总电容。即

$$C_{total} = C_{stems} + C_{FF} + C_{interconnects} \quad (3\text{-}2)$$

$Stem_H$: 水平树干的数目。

$Stem_V$: 垂直树干的数目。

N_b：网格型时钟中缓冲器的数量。我们假设缓冲器放在水平和垂直的树干的某些交叉点上。

$$N_b = \rho * Stem_H * Stem_V \quad (3\text{-}3)$$

其中ρ是用户自定义参数，$\rho \le 1$。 当ρ=1时，表示每个树干的交叉点上都有缓冲器。

AveLoad: 每个缓冲器平均负载。

$$AveLoad = C_{total} / N_b \quad (3\text{-}4)$$

Delay(i) 第 i 个寄存器的时钟延迟

Skew：这里的时钟偏差是这个时钟网络的时钟偏差，由如下式子定义：

$$Skew = \max \{abs (delay(i)-delay(j)) \mid \forall \text{ ith \& jth FF } \} \quad (3\text{-}5)$$

3.4 网格规划

3.4.1 寄存器时间特性的刻画

时钟网格的质量跟以下几个时间度量是密切相关的，时钟延迟，过渡时间和时钟偏差。一个标准的自动化时钟网络综合流程应该能让设计者给定这些时间参数的约束。为了满足这些约束，自动化综合器应该按次序决定树干的数量，树干的位置，寄存器的局部布线，缓冲器位置放置。时钟偏差在很大程度上取决于网格型时钟的几何结构，即树干的数量和它们的位置。如果能找到一个好的对时钟

网格时间度量和负载关系的刻画，那将给初始的网格规划很大的帮助。

在数字电路设计中，我们知道，一个门的时间特性可以通过其“延迟----负载”曲线来刻画，就是说，一个门的延迟依赖于其驱动的负载。图 3-9 所示，是一个单独缓冲器的驱动电容的模型。图 3-10 是其“延迟----负载”曲线。

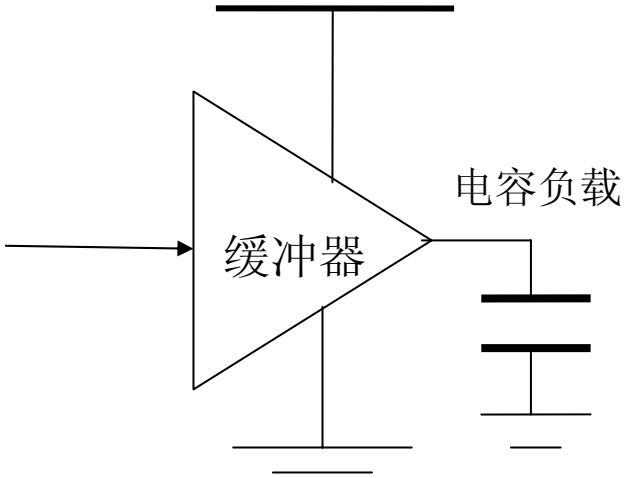


图 3-9 单个缓冲器驱动电容负载模型
Fig. A single buffer driving load model

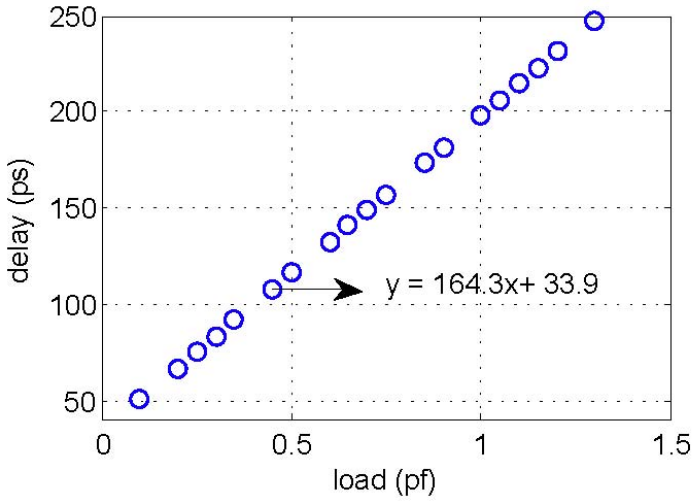


图 3-10 一个缓冲器的“延迟—负载”曲线
Fig. 3-10 delay versus load curve of a single buffer

从图 3-10 可以看到，缓冲器的延迟随着负载的增加而增加，而且在很大的范围内基本成线性关系。图中两者的关系式是通过最小二乘法拟合得到的。不同的缓冲器的“延迟—负载”曲线是不同的，这也是“延迟—负载”曲线能成为刻画缓冲器特性的原因。图 3-11 是另外一个缓冲器的“延迟—负载”曲线。不难看到，图 3-10 中的缓冲器的驱动能力比图 3-11 的强，因为对于相同的负载，图 3-10 中

的延迟远小于图 3-11 中的延迟。

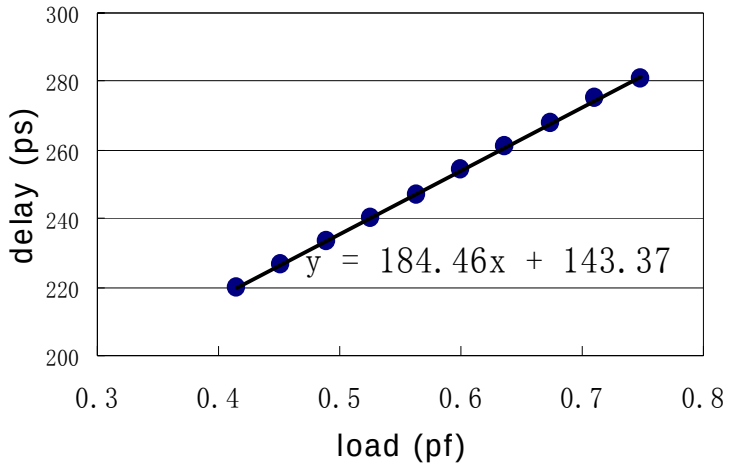


图 3-11 另一个缓冲器的“延迟—负载”曲线
Fig. 3-11 delay versus load curve of another buffer

受到“延迟—负载”能刻画缓冲器特性的启发，我们猜想在网格型时钟中，是否也能用类似的曲线刻画缓冲器在复杂的网格型时钟中的行为特性。通过多次实验，得到了以下一个重要结论：

假设在网格型时钟树干的每个交叉点上都放有缓冲器（如图 3-12,红色三角形为缓冲器），定义平均延迟为所有寄存器时钟输入端的延迟除以寄存器的数目；平均负载为C_{total}除以缓冲器的数目。则网格型时钟中的“平均延迟—平均负载”曲线跟单个缓冲器的“延迟—负载”曲线基本是一样的，只要这两种情况下的缓冲器类型是一样的。

图 3-13 是一个网格型时钟中的“平均延迟—平均负载”曲线，该网格型时钟中的缓冲器跟图 3-10 的是一样的。

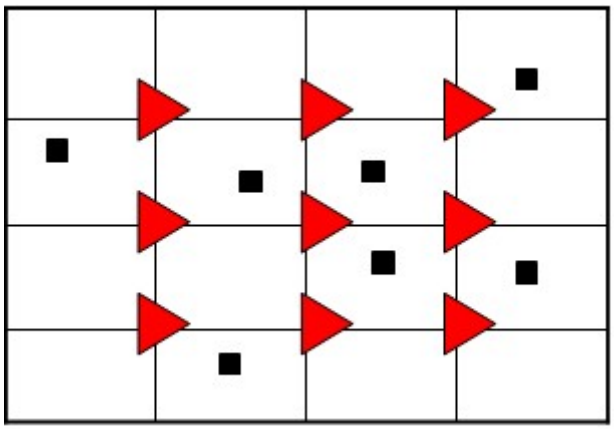


图 3-12 缓冲器位于树干交叉点

Fig.3-12 Buffers placed at stems intersections

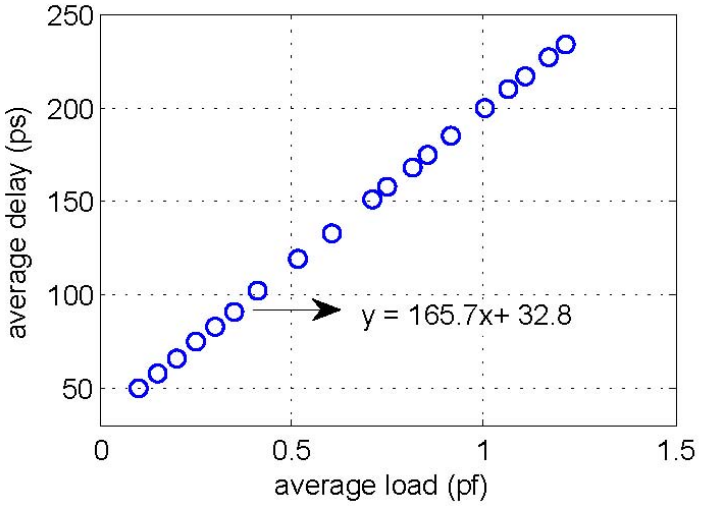


图 3-13 平均延迟—平均负载曲线

Fig.3-13 Average delay versus average load curve

对比图 3-10 和图 3-13 中的曲线，不难发现，两者基本一样。对于图 3-11 中的缓冲器，我们也做同样的实验，把它放到网格型时钟中，观察其“平均延迟—平均负载”曲线，结果如图 3-14 所示。

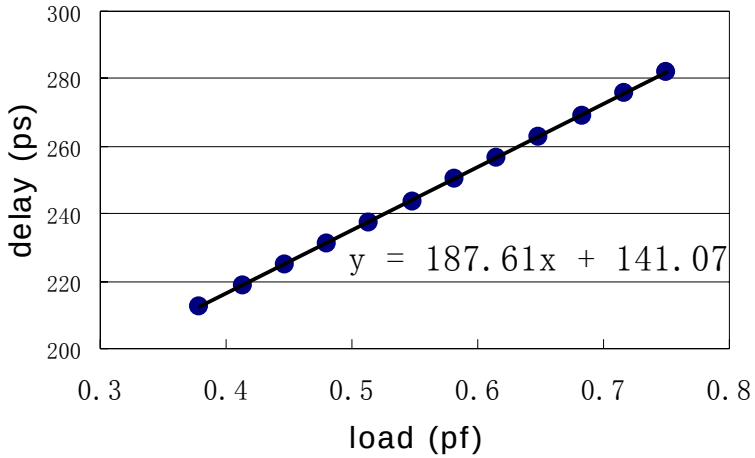


图 3-14 另一个平均延迟—平均负载曲线例子

Fig.3-14 Another example of average delay versus average load curve

同样可以发现，图 3-11 中的曲线跟图 3-14 中的曲线基本一致，再次证明了结论的正确性。

我们还做了另外一个实验，以确保结论的普遍性。对于同一个缓冲器，把它放在不同的网格型时钟结构中，看其“平均延迟—平均负载”曲线是否受到网格

型时钟结构的影响，如果“平均延迟—平均负载”曲线能继续保持一致性，则再能说明该结论不受网格型结构的影响，“平均延迟—平均负载”曲线确实能刻画出缓冲器在网格型时钟的特性。如图 3-15 所示是同一种缓冲器在不同网格型时钟的“平均延迟—平均负载”曲线。

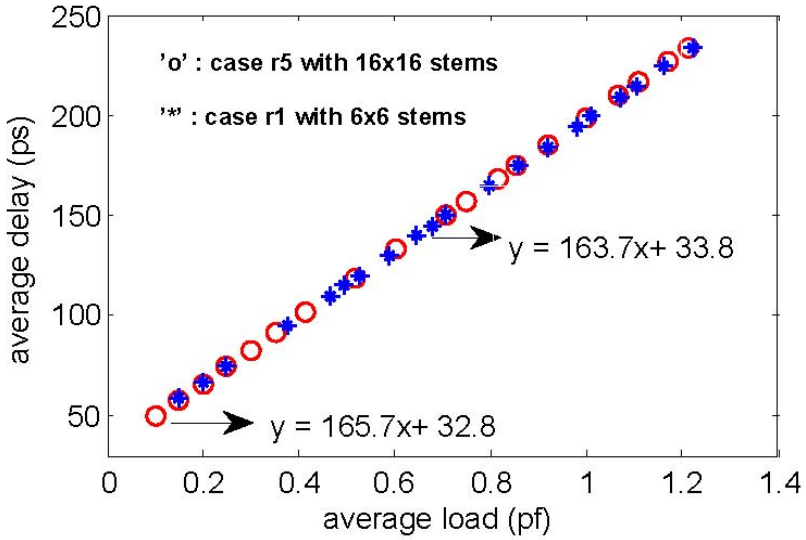


图 3-15 同一缓冲器在不同网格型时钟的“平均延迟—平均负载”曲线

Fig. 3-15 Two delay-aveload relations for two meshes using the same type buffers

红色空心圆圈的曲线是在一个有 3101 个寄存器的芯片上在，分别放置 16 根水平树干和 16 根垂直树干的网格型时钟中缓冲器的“平均延迟—平均负载”曲线；蓝色实心点的曲线是在一个有 267 个寄存器的芯片上在，分别放置 6 根水平树干和 6 根垂直树干的网格型时钟中缓冲器的“平均延迟—平均负载”曲线；不难看出，两根曲线基本重合。这说明了缓冲器的“平均延迟—平均负载”曲线跟网格型时钟的结构是没有关系，该曲线只跟缓冲器的类型有关，再次验证了结论的正确性。

缓冲器不仅存在“平均延迟—平均负载”曲线的不变形，通过实验，我们发现，上述的结论同样适用于“平均过渡时间—平均负载”曲线，只要是同一种寄存器类型，则不管其处于什么结构的网格型时钟中，其曲线都基本是一致的。如图 3-16 和 3-17 所示是同一个寄存器在不同的网格型时钟的“平均过渡时间—平均负载”曲线。曲线只跟寄存器的类型有关。

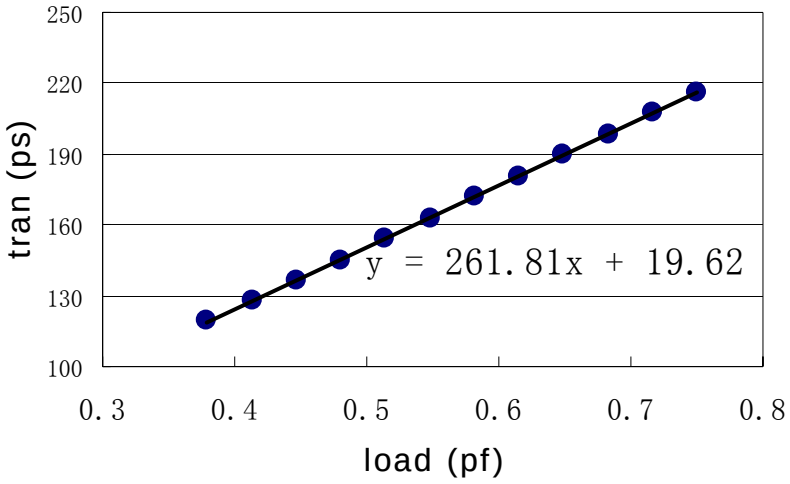


图 3-16“平均过渡时间—平均负载”曲线
Fig. 3-16 average tran. versus average load curve

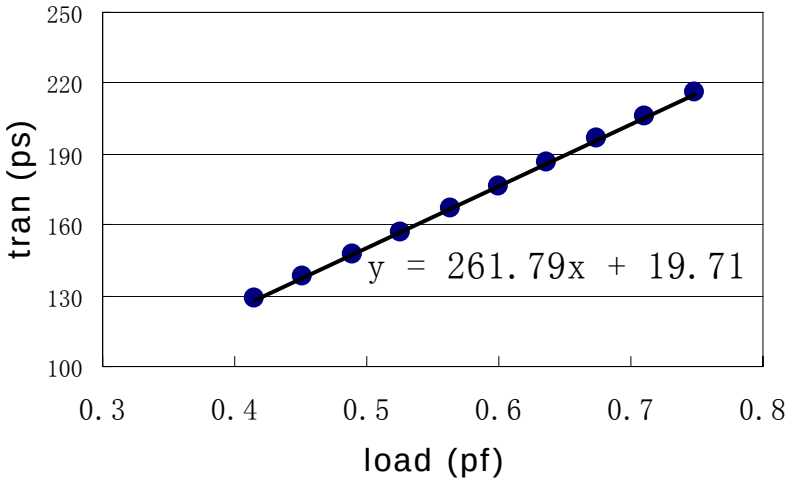


图 3-17 另一条“平均过渡时间—平均负载”曲线
Fig. 3-17 Another average tran. versus average load curve

根据以上的结论，只要知道单个缓冲器的“平均延迟—平均负载”和“平均过渡时间—平均负载”曲线，就能知道获得该缓冲器在网格型时钟中的“平均延迟—平均负载”和“平均过渡时间—平均负载”曲线，因为这两者是基本一样的，而单个缓冲器的“平均延迟—平均负载”和“平均过渡时间—平均负载”曲线很容易获得，在设计的工艺库通过查表就能知道。

3.4.2 网格规划算法

根据上一节的结论，给出如下网格规划的算法。

算法输入：

给定时钟延迟的约束区间*[d1, d2]*, 过渡时间的约束区间*[s1, s2]*，以及要使用的缓冲器的“平均延迟—平均负载”和“平均过渡时间—平均负载”曲线。

算法流程：

设置m的初始值$Stem_H$；n的初始值$Stem_V$。

通过“平均延迟—平均负载”曲线把时钟延迟约束区间转换为平均负载约束区间$A := [c_1', c_2']$ ，如图 3-18 所示。同理，通过“平均过渡时间—平均负载”曲线过渡时间约束转换为平均负载约束$B := [c_1'', c_2'']$。另$I := A \cap B = [c_1, c_2]$.

如果 $I = \phi$, 则无法满足时间约束，不存在解，退出程序。

计算当前的 AveLoad.

1. ***If*** (AveLoad < c1)
   m := m – 1; n := n – 1. 转到步骤 4.
   Else if (AveLoad > c2)
   m := m + 1; n := n + 1. 转到步骤 4.

算法输出：

水平树干的数量$Stem_H$和垂直树干的数量$Stem_V$。

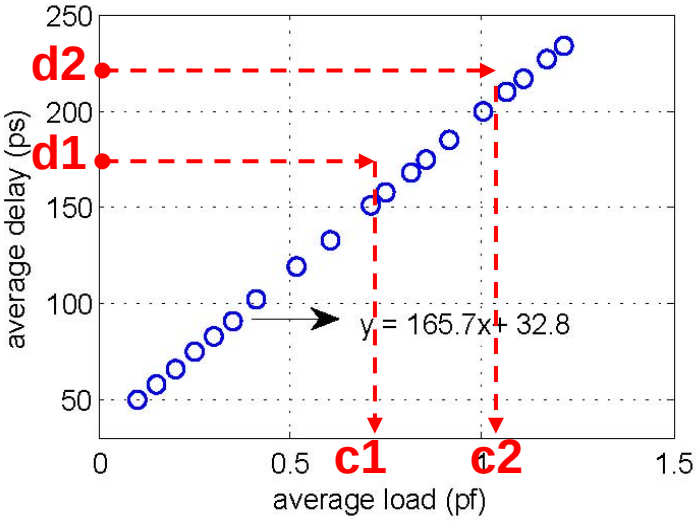


图 3-18 转换示意图
Fig. 3-18 delay load conversion

3.5 树干和缓冲器放置

3.5.1 树干的摆放

在上一节中，网格型时钟中的水平和垂直的树干数量已经决定了，接下来的任务就是把它们摆放在芯片上的位置。

在实际设计中，寄存器在整个芯片上的分布式很不均匀的，如图 3-19 所示，是一个有 267 个寄存器在芯片上分布的例子，椭圆区域内的寄存器分布密集，矩形区域内的寄存器分布稀疏。一个好的树干摆放策略是考虑到寄存器的非均匀分布，这个在[31,32]中没有考虑到了，它们只是把树干平均放置在芯片上。

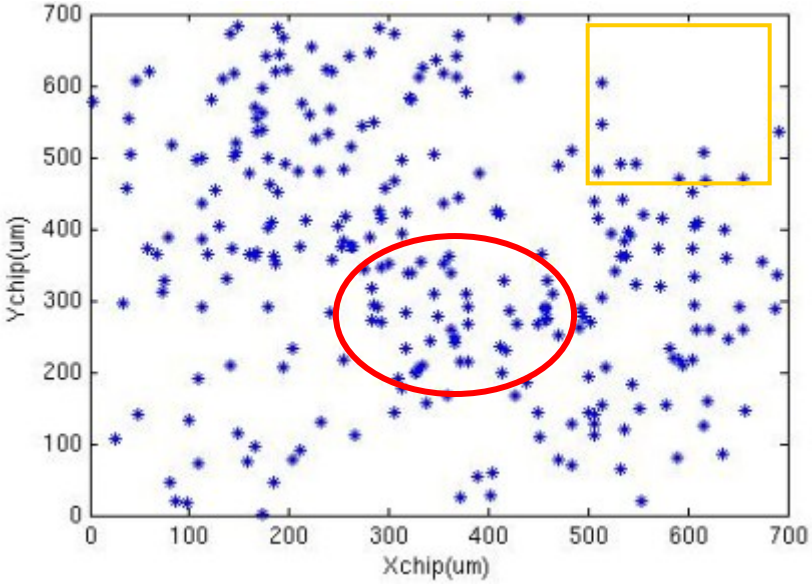


图 3-19 寄存器的非均匀分布

Fig. 3-19 non-uniformal distribution of FFs

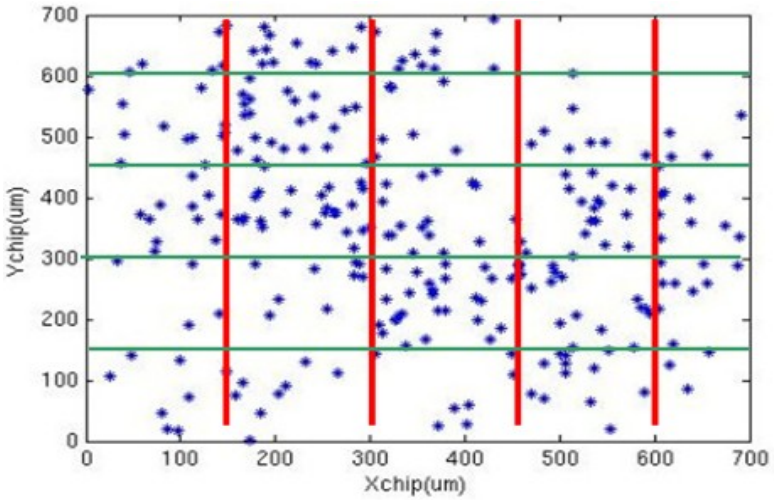


图 3-20 树干均匀分布
Fig. 3-20 uniform distribution of stems

图 3-20 是水平和垂直树干在芯片上均与分布的例子，即相邻水平树干间的距离是一样的，相邻垂直树干间的距离是一样的。其中红色的为水平树干，绿色的为垂直树干。

我们知道，时钟偏差主要是由负载不平衡造成的，因此很自然的想法是在网格型时钟的每个步骤中都考虑负载的平衡性。在前面的章节已经介绍过，每个寄存器基本都被四根树干包围，寄存器会连接到其中的任何一根。树干均匀放置的缺点在于在寄存器密集的区域，该区域附近树干会有很多寄存器连接到其上面，导致负载过重；在寄存器稀疏的区域，该区域附近树干上连接到其上的寄存器数量较少，负载过轻；这种负载不平衡性必然会导致较大的时钟偏差。

为方便后面阐述，先做如下定义。令$\mathbf{C_{zone}(i)}$ 为第i个zone的总电容，包括所有寄存器时钟输入端的总电容和把寄存器连接到树干上的互连线的总电容之和。

本文提出的树干摆放策略是，平衡每个zone的$\mathbf{C_{zone}}$，即通过不断调整树干的位置，直到每个zone内的$\mathbf{C_{zone}}$基本一致。图 3-21 是考虑到zone负载平衡的树干摆放的例子。

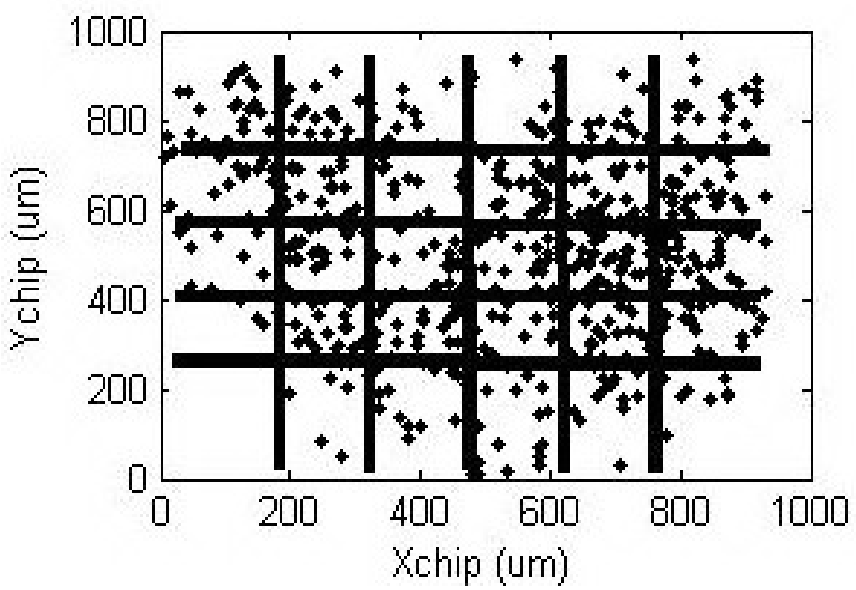


图 3-21 “负载平衡”树干摆放例子
Fig.3-21 ”load aware” stem placement

从图中可以看到，在寄存器密集的地方，相邻树干的距离比较近，这样做可以减少寄存器到树干的连线长度，从而减少电容负载；在寄存器稀疏的地方，相

邻熟饭的距离较远，这样做的目的在于增长寄存器到树干的连线长度，增加电容负载，已达到每个区域内$\mathbf{C_{zone}}$尽量相等的目的。

在实际中，没必要让每个$\mathbf{C_{zone}(i)}$在这个阶段精确相等，因为$\mathbf{C_{zone}(i)}$在这个阶段仅仅是一个估计值，因为没做实际的局部布线前，寄存器到树干的连线长度是估计出来的。定义

$$C_{min}= \min\{C_{zone}(i), \forall i\}, \quad C_{max} = \max\{C_{zone}(i) , \forall i \}.$$

同时人为设置$\alpha = C_{min} / C_{max}$为迭代终止条件，当当前的$C_{min} / C_{max} > \alpha$ 是，程序结束。

算法输入：

寄存器在芯片上的分布，即物理位置。

算法流程：

首先把树干均匀放置作为树干位置的初始值，同时设置每次迭代中树干移动的步长。设置程序结束的阈值α 。

计算当前$\beta = C_{min} / C_{max}$。

If (. . .

增加（或者减少）在寄存器稀疏区域（密集区域）中树干的位置（移动一个步长），转到步骤 2。

Else

退出程序。

算法输出：

每根水平和垂直树干在芯片上的物理位置。

3.5.2 缓冲器放置

在此阶段，我们直接利用[31]的中 set-cover 方法来放置缓冲器，最为初始解，这个阶段缓冲器位置不是其最终的位置，后面的优化步骤还会调整其位置。

Set-cover 问题定义如下：给定一个有限集合 U 及 U 的一个子集集合 S。子集 S 覆盖了有限集合 U。即 U 中的每一个元素至少属于 S 中的一个子集，set-cover 就是要找到 U 中的一个最小的子集 C$\subseteq$F，C 覆盖了 U 中所有元素。下面阐述如何把缓冲器放置问题转换为 set-cover 问题。

给每个缓冲器定义一个“覆盖区域”(Covering Region)，该区域内的总电容（包括寄存器时钟输入端电容和互连线）不能超过该缓冲器能驱动的总电容。令CR_i^j 代

表第 i 个寄存器被第 j 个缓冲器驱动。不难看出，所有寄存器的集合相当于 U，CR_i^j 组成了集合 S。如果 CR_i^j 被选择，就代表第 i 个寄存器被第 j 个缓冲器驱动。于是，缓冲器插入的问题就转换为如何选择合适的缓冲器位置，使其能覆盖掉所有的“覆盖区域”，每个寄存器都能被至少一个缓冲器覆盖，同时保证缓冲器的总面积最小。

传统的set-cover问题是NP问题，通常使用贪婪算法（Greedy algorithm）[33]保证在丢失较少精度的情况下提高算法运行速度。下面是解决set-cover问题的经典贪婪算法。

算法流程：

K=U;

C=ϕ;

while(K!= ϕ){

选择 S 中使|M$\bigcap$ K|最大的子集 M；

K=K-M;

C=C$\bigcup$ {M}; }

return C;

在该算法中，集合 K 用来存放每一阶段中尚未被覆盖的 U 中的元素。集合 C 包含了当前已构造的覆盖。在循环主体中，首先选择 U 中覆盖了尽可能多的未被覆盖的元素子集 M。然后将 K 中被 M 覆盖的元素删去，并将 M 加入 C。算法结束时 C 中包含了覆盖 U 的一个子集。

上述的 set-cover 问题是没有考虑子集和 S 的元素的权重的，可以给每个子集一个权重，使 set-cover 成为一个更具普遍性的问题。在缓冲器放置问题中，就是给每个 CR_i^j 都赋予了不同的权重，即代价函数，但整体的算法框架跟上述的一模一样的。

[32]中的代价函数仅仅考虑了缓冲器的面积和驱动能力，而[31]中的代价函数把网格型时钟低通滤波器的特性（时钟信号强度随着其传播距离成指数级衰减）和缓冲器驱动能力大小同时考虑到代价函数中，从而使缓冲器位置的选择更合理。如图 3-22 所示，是缓冲器位置摆放的一个示例，图中橙色三角形代表驱动缓冲器。黑色的粗线代表水平和垂直的树干，蓝色的实心点代表寄存器。

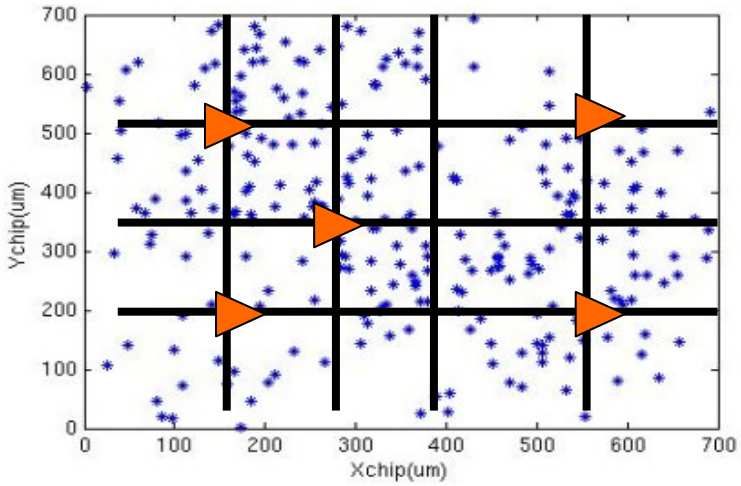


图 3-22 缓冲器摆放示例

Fig. 3-22 An example of buffer placement

3.6 局部时钟布线

当树干的位置和缓冲器的位置确定好后，接下来要做就是把寄存器的时钟输入端连接到树干上，即局部时钟布线阶段。

3.6.1 信号布线跟时钟网络布线的区别

一般的信号布线跟时钟网络布线（把寄存器的时钟输入端连接到相应的时钟信号线上）是有差别的，从图 1-1 可以看到，时钟网络布线在信号布线前，优先级高于信号布线。在时钟网络布线时，芯片上的可用布线资源较多，空间限制较少，这个为了时钟布线在保证布通率到的前提下，尽量减少时钟线之间的耦合窜扰，优化关键路径时间延迟，减少是时钟偏差等。较多的布线资源为达到这些目标提供了基础，是时钟网络布线器在布线时有更多的选择性，因此，总体来说，时钟网络的布线难度相对较低。相反，信号布线在布线时，布线资源已被时钟网络占用了一些，而且由于信号线网的数量远远大于时钟线网，所以信号布线的难度较大。

时钟网络布线通常位于相邻的两个金属层，而且是较高的金属层，譬如第 5，6 金属层，这是为了较少时钟网络负载，减少时钟延迟；而信号布线为了确保布通率，连线可能需要在几个不同的层间穿梭。

信号布线需要考虑的因素比时钟网络布线多得多，如功耗，面积，线长，串扰，可制造性[36-38]等，而且信号线网由于数量巨大，信号布线一直后端设计的难

点之一。为解决这个大型难题，信号布线一般可以范围两个阶段，全局布线和详细布线。

全局布线(global routing):

两端点线网的总体布线主要是在布线图中寻找最短布线路径并使得期望的目标函数最优，同时在线网路径的分配中必须满足布线图中对应的边（通道）容量的限制。而多端点线网的总体布线可定义成一个寻找连接树问题。斯坦纳树是一棵连接特定要求点集合和一些其它成为斯坦纳点的连接树，其中斯坦纳点的数量是任意的。由于它具有比其它方法求得的连接树总长度更小的优点，往往被用来作为在总体布线中构造连接树的方法。于是，总体布线问题实际上就是在一个总体布线图中，在期望目标函数最优化的条件下，针对每条线网寻找一棵斯坦纳树的问题。通常目标函数是所选择的连接树的总长度最小。一般的总体布线问题可定义如下：

给定一个网表 $N=\{N_1,N_2,.....N_n\}$，和一个总体布线图$G=(V,E)$,对于$\forall N_i \in N,1\le i\le n$,找到一棵斯坦纳树$T_i$，使得

最小化$\sum_{i=1}^{n} L(T_i)$，受限于$U(e_i)\le(e_j),\forall e_j \in E$，

其中，$L(T_i)$为T_i的长度；$U(e_j)$是通过通道边e_j的线网数，它由下式决定：

$$U(e_j)=\sum_{i=1}^{n} x_i$$

如果e_j在T_i中，则x_i=1 否则x_i=0；

总体布线问题是集成电路布图设计中的一个重要环节，求其最优解是 NP 问题，近十几年来人们提出了许多行之有效的算法。如串行布线与拆线重布法(sequential routing and rerouting method) 、基于加权的斯坦纳树算法、基于整数规划的方法、线性规划法(linear programming method)、层次布线算法(hierarchical routingmethod) 、基于拥挤度分析的并行层次迭代布线算法，以及基于货物流理论的算法(flow shipping algorithms)等。串行布线与拆线重布法是对版图中的各个线网逐一进行布线，一次布一条线网。算法为每条线网找到当前允许的最短斯坦纳连接树，但要求经过的总体布线单元边界与层面满足容量大于用量的条件。这种方法存在严重的顺序依赖性，先布的线网自由度大，而后布的线网却有可能被前面的布线堵死。因此，它必须有一个拆线重布的处理。

层次布线算法是另外一种总体布线算法。其基本思想是任务分治即把大任务

逐步细分成多个小任务，而后一一解决。这与总体布线和详细布线分开有异曲同工之妙。具体方法是每次将称为“块”的当前布线区分成多个块，对线网做布线块一极的总体布线，并确定穿越块边界的各线网的跨边及其过点，在分别对子块递归地调用层次布线过程，直到块被细分成一个个总体布线单元网格。最后，线网的所有确定边便够成一棵总体布线树。但是它同样有一些不足之处，比如，在做高层块级总布线时，由于对底层布线的局部拥挤状况不了解，会导致一些上层决策不适合实际布线的问题。

详细布线(detail routing):

详细布线阶段确定线网各个线段在布线区域中的具体位置，从而完成线网在布线区域的最后定位。

迷宫算法

Lee[2]于 1961 年提出了一个两端线网的布线算法，称为李氏算法。李氏算法实际上是图论中最短路径算法在布图设计中的一种应用。其算法思想也可描述为对波传播过程的模拟。自李氏算法提出以后，有许多对该算法的改进[3]，包括提高其速度和减小其计算空间。李氏算法及其改进算法形成一组布线算法，统称迷宫算法(mazerouting algorithms)。李氏算法在整个搜索中总是对称的，即向四周同时扩展搜索。1978 年 Soukup[4] 在李氏算法引进一个带有固定意义的非对称搜索的方法。该算法只搜索趋向目标点方向上的点，一直进行到找到目标点为止。该算法的速度是李氏算法的 10 至 50 倍。另外一个加快李氏算法的方法由 Hadlock[5] 于 1977 年提出，称为 Hadlock 最小迂回算法。

计算智能的方法

对于 VLSI 芯片来说，其线网数可能是成千上万的，且对于每个线网，又有几百种甚至更多的布线方案，现在已经证明布线是一个 NP 完全问题。于是很多学者和专家尝试把一些用于解决 NP－完全问题行之有效的计算智能方法用于布线中去。如神经网络技术、遗传算法及蚁群算法。

3.6.2 负载平衡时钟布线

当今的商业后端工具，如 Synopsys 的 IC Compiler 和 Cadence 的 Encounter 在这个阶段的做法是直接调用其一般的信号布线器，而不会特别开发一个新的布线器来完成网格型时钟的局部布线。这些布线器会按照“最近原则”布线，即把寄存器连接到距离其最近的树干上。这样做的目的在于可以是总的连线长度达到最短，占用的空间资源较少，整个网格型时钟的互连线上的负载也达到最少，从

而减少时钟延迟和功耗。

但正如前面章节论述的，在网格型时钟综合的每个阶段，都必须考虑到负载平衡，这样才能最大可能的减少时钟偏差。“最近原则”布线会导致在寄存器密集区域，寄存器全部连接到离它最近的树干上，负载过重；在寄存器的稀疏区域，某些树干上的连接的寄存器数量可能极少，负载很少。这种不平衡性必然会造成较大的时钟偏差。如图 3-23 所示，是由于“最近原则”布线造成树干负载不平衡性的例子。

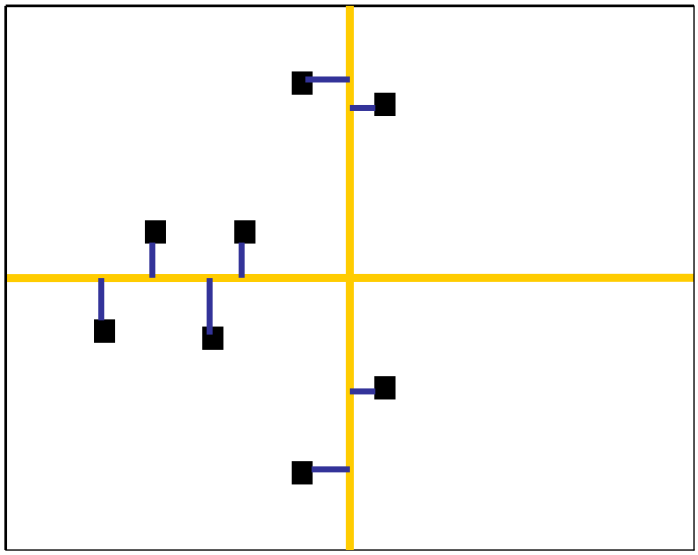


图 3-23“最近原则”布线示例

Fig. 3-23 nearest routing strategy

如图所示，四根树干上连接的寄存器数量最多的一根是 4, 最少的一根 0，其余两根是 2。由“最近原则”布线造成的树干负载不平衡性相当明显。

为此，我们提出了一个“负载平衡”布线器去解决由于“最近原则”布线造成的树干负载不平衡引起的时钟偏差。首先引入如下定义：

$$\textbf{AveStemLoad} := Ctotal / (StemH + StemV) \qquad (3\text{-}6)$$

即每根树干的平均负载。

StemLoad：即每根树干的总负载，包括树干本身的电容，连接到该树干的局部连线的总电容和跟局部连线相连的寄存器时钟输入端的电容之和。

通常来说，一个寄存器被四根树干包围着，它连接到哪一根树干是由其代价函数决定的，譬如“最近原则”的布线代价函数就是寄存器到树干局部连线的距离。在本文中，我们为“负载平衡”布线器设计了如下代价函数，来指导布线器选在合适的树干，把寄存器连接到其上面去。

$$\mathrm{cost}(k,c)=\frac{\eta}{2^{c_0-c}}+\delta\cdot\frac{L_k}{L_{est}} \tag{3-7}$$

式中c_0———等于 AveStemLoad；

c———等于 *StemLoad*，当前树干的电容负载；

L_k———某一树干连接到第k个寄存器的局部连线的长度；

η———用户定义参数，给树干负载的权重；

δ———用户定义参数，给局部连线长度的权重；

参数越大，表明该代价函数越注重 树干负载 平衡性； 参数越大，表明该代价函数更多地考虑布线的长度。如果把 设为，即不考虑树干负载的因素，则该布线器就是 最近原则 布线器。但是，假如我们只是考虑“树干负载”的平衡性而不考虑局部布线的长度，即把 设为，实验表明，不仅会导致总的布线长度的增加，全局时钟偏差反而会比 最近原则 布线的更大。因此， 树干负载平衡 和局部连线长度是两个要相互协调的因素，这样才能达到减少全局时钟偏差的目标。图 是用 负载平衡”布线器对图 3-23 进行重新布线的结果，不难看到，其中一个寄存器改变了其连接到的树干，使得每根树干的负载比“最近原则”布线中的树干负载平衡。

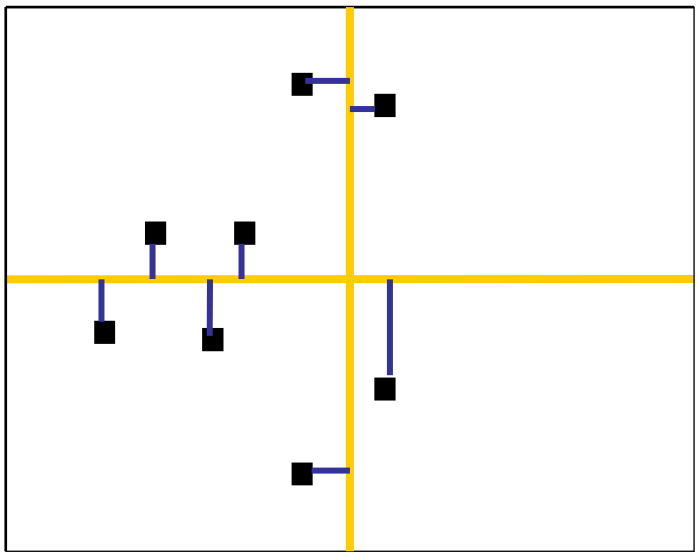


图 3-24 “树干负载”平衡布线器

Fig. 3-24 stem load aware router

3.7 约束验证

时钟布线结束后，网格型时钟的综合流程的主体部分已经完成，接下来的任务就是要进行时序验证，看综合后的网格型时钟是否满足时序约束要求。如果通过仿真发现综合后的网格型时钟无法达到要求，则需要回到上面的步骤重新综合，如图 3-6 所示的综合流程图一样。要进行时序约束验证，首先要把综合后的网格型时钟的物理网表转换为仿真器能识别的电路网表，然后才能用一般的仿真器进行仿真，获取时序信息。

3.7.1 物理版图到电路网表的转化

如何把物理网表转换为电路网表，下面以一个简单示例加以说明。

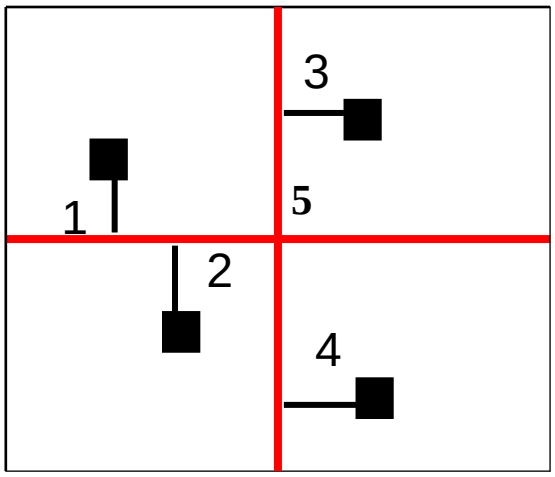
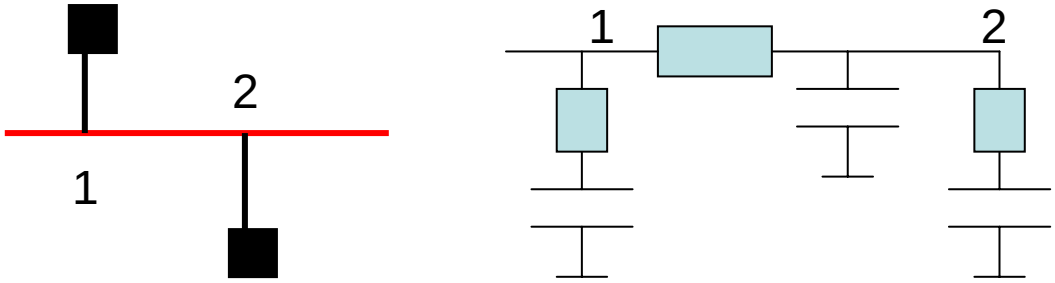


图 3-25 物理网
Fig.3-25 physical net

如图所示，是综合后的时钟网格的物理结构，图中共有 4 个寄存器（四个黑色实心小方框）编号为 1-4，一根水平树干和一根垂直树干。其中 1,2 号寄存器连接到水平树干上，3,4 号寄存器连接到垂直树干上。我们的转换算法以树干为参考，遍历每根树干，看连接到树干上寄存器，每遇到一个寄存器，则在树干上创建一个新的电路节点，新建的电路节点与前一个树干节点构成一段互连线，该互联线可以用一段简单的 RC 电路等效；把寄存器等效为一个电容，电容大小为时钟输入端的电容，那么新建的电路节点和寄存器间也可以用一段简单的 RC 电路等效。图 3-26 显示的是一个简单的物理网表转为电路网表的示意图。



a)物理网表　　b) 对应的电路网表

a) physical net　　b) corresponding circuit net

图 3-26 物理网表转换为电路网表示意

Fig.3-26 physical net to circuit net

前面的转换规则虽然简单，但在实际的程序实现时，需要注意不少细节。譬如水平和垂直的树干的交叉点，这些是公共节点，不管有没有寄存器连接到上面，都要创造这些节点。如图 3-25 中的节点 5，由于有这个特殊节点的存在，节点 3,4 之间不能只用一段 RC 互连线模拟，需要分成两段，节点 3,5 一段；节点 5,4 一段。还要注意的寄存器连接点重合的问题。譬如对于一根水平树干，假如有两个水平坐标一样的寄存器连接到其上面，则会连接到树干上的同一点，这时，不能创建两个新节点，只能是一个，所以在创建一个新节点时要首先判断是否存在上述这种情况。对于垂直的树干也是如此。

为了提高仿真精度，在实际程序中，我们不是把一段互连线模拟成一段简单的 RC 电路，而是采用了更为通用的 π 模型。如图 3-27 所示。

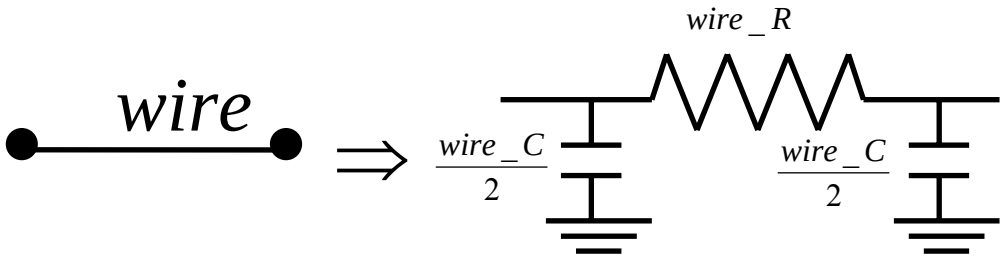


图 3-27 互连线的 pi 模型

Fig.3-27 pi model of an interconnect

这个模型虽然提高了仿真精度，但由于每根互连线都比原来的模型多了一个电路节点，导致电路总的电路节点增加，减慢了仿真速度。

3.7.2 仿真器验证

物理网表经过上述算法的转换成电路网表后，就能用业界常用的电路仿真器进行仿真，获取其时间延迟，时钟偏差等信息了。对于网格中的缓冲器，我们使用一个从 90nm 设计中提取的模型，不使用像[31,32]中的缓冲器简化模型。

在这里，我们用 Hspice 电路仿真器进行仿真，在网表中通过.meas 命令可以直接获取时钟延迟和过渡时间的值，对于时钟偏差，可以通过脚本找到时钟延迟的最大和最小值，两者差的绝对值就是时钟偏差。同时，通过 INTEGM 命令可以计算出整个网格型时钟的功耗。

3.8 本章小结

本章首先介绍了关于网格型时钟最新的研究成果，主要包括仿真分析和综合优化两部分。仿真分析是指通过电路划分，模型降阶等方法提高网格型时钟的仿真速度；综合优化是指自动化网格型时钟综合，同时通过对缓冲器位置大小的选择，去除“冗余”树干的方法减少网格型时钟的功耗是时钟偏差。在前人的基础上，本文提出了一个全新的网格型时钟自动化综合框架，主要包括网格规划，树干和缓冲器放置，局部失踪布线，约束验证几个步骤。文中对上述的每个步骤做了详细的阐述。在网格规划阶段根据设计约束要求和芯片上可用的空间资源决定整个时钟网格中树干的数量。在树干和缓冲器摆放阶段，根据一定的算法把树干放在合理的位置上，缓冲器的初始位置由 set-cover 算法给出。局部时钟布线考虑到树干负载平衡原则，把寄存器时钟输入端连接到树干上。约束验证阶段把此时物理结构的时钟网络转换为电路网表，通过 Hspice 仿真，看是否满足约束要求，如果不满足，则需要回到 1-3 的步骤重做。

第四章 网格型时钟的缓冲器优化

4.1 缓冲器优化

通过 Hspice 仿真，我们可以获得每个寄存器的时钟延迟。图 4-1 是一个测试例子中寄存器的时钟延迟分布。

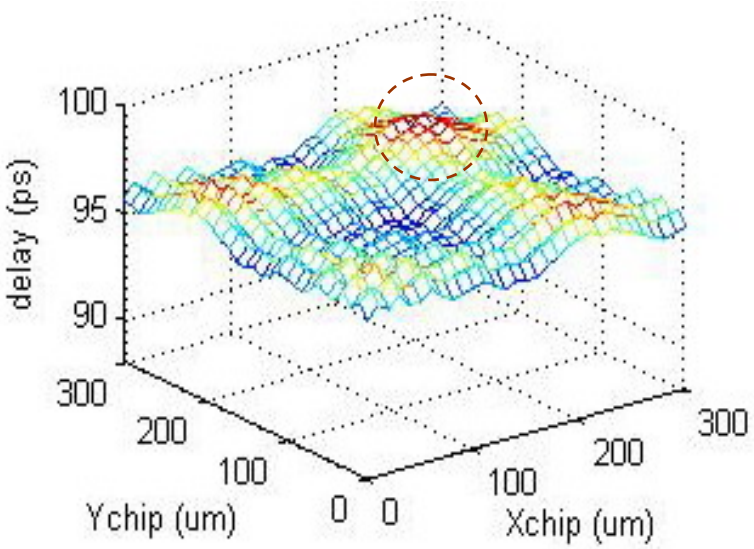


图 4-1 寄存器时钟延迟分布
Fig. 4-1 Delay distribution of FFs

从图中可以看到，网格型时钟中，寄存器的时钟延迟分布是一个连续的图景，红色的区域是寄存器时钟延迟较大的地方，蓝色区域是寄存器时钟延迟较小的地方。红色颜色越深代表延迟越大，蓝色越深代表时钟延迟越小。通过观察这个三维图，我们可以发现一个有趣的现象，就是时钟延迟较大和时钟延迟较小的寄存器有聚集效应，就是说有较大时钟延迟的寄存器在物理位置上是靠近的，同样，有较小时钟延迟的寄存器在物理位置上也是靠近的。这个现象在文献[31]中也阐述过， 文献[24,28]给出了解释，这是由于网格型时钟本质上是一个低通滤波器，信号在网格型时钟传播时，其强度随着传播距离成指数级衰减。全局时钟偏差就是出现在红色区域跟蓝色区域内寄存器时钟延迟的最大差值的绝对值。

图 4-2 是另外一个表明寄存器时钟延迟具有聚集效应的例子，图中黑色的实心“*”代表寄存器 1，水平和垂直的空心粗线代表树干，在树干的交叉点上，实心灰色圆圈代表缓冲器。在图的左下角，用粉色小正方形标记的寄存器是时钟延迟最少的那些寄存器，在图的右上角，用红色三角形标记的寄存器是时钟延迟最大的寄存器。全局时钟偏差就是出现在红色三角形区域跟粉红色正方形区域内寄

存器时钟延迟的最大差值的绝对值。

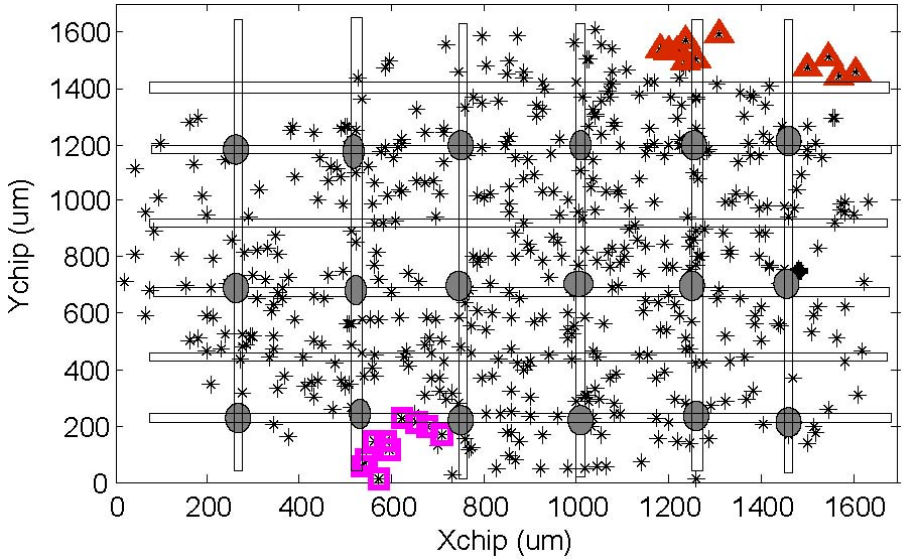


图 4-2 综合后的网格型时钟

Fig.4-2 An example of a synthesized clock mesh

具有较大时钟延迟的寄存器表明其接收到的时钟信号不够强，被缓冲器驱动的能力较弱；具有较小时钟延迟的寄存器表明其接收到的时钟信号过强，被缓冲器驱动的能力很强。根据网格型时钟低通滤波器的性质，我们可以做出以下判断，具有较大时钟延迟的寄存器，其周边的缓冲器驱动负载过重，具有较小时钟延迟对的寄存器，其周边的缓冲器负载过轻。在不改变缓冲器数量的情况下，我们是否可以把具有较少时钟延迟寄存器附近的缓冲器移动到具有较大时钟延迟的寄存器旁边，达到平衡两者的目的。通过这样的移动后，原来具有较小时钟延迟的寄存器旁边的缓冲器的驱动能力减弱了，因为缓冲器数量减少了；原来具有较大时钟延迟的寄存器旁边的缓冲器的驱动能力增强了，因为缓冲器数量增多了。具有较小时钟延迟的寄存器由于驱动能力减弱，其时钟延迟会比原来增大；具有较大时钟延迟的寄存器由于驱动能力增强，其时钟延迟会比原来减少；如此，小者变大，大者变小，两者的差值绝对值必然减少，也就意味着时钟偏差减少。

上述通过缓冲器移动达到减少时钟偏差的做法，充分利用了网格型时钟低通滤波器的特性，该特性使缓冲器移动的影响仅限在局部的范围内，只是影响了缓冲器周边的寄存器时钟延迟。否则，如果缓冲器移动是牵一发而动全军，影响范围很大的话，那该移动算法造成的影响是无法预测，也就没有实用意义，甚至通过移动寄存器后时钟偏差反而变大也是很有可能发生的情况。

图 4-3 是把图 4-2 中的缓冲器移动后的例子，我们把时钟延迟较小的寄存器附

近的两个缓冲器移动到时钟延迟较大的寄存器附近，蓝色圆圈的就是被移动的那两个缓冲器。

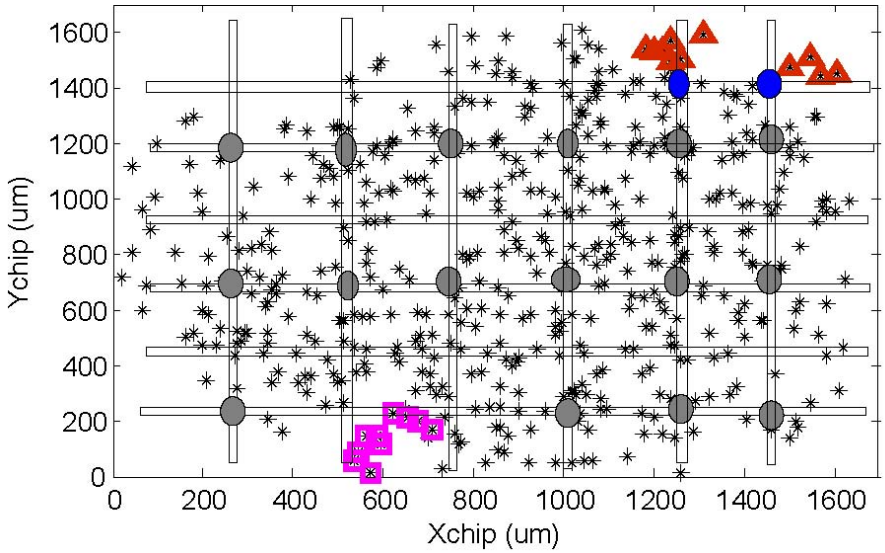


图 4-3 缓冲器移动例子

Fig. 4-3 buffer relocation

在没有移动缓冲前，寄存器的时钟延迟信息是通过 Hspice 仿真得到的，得到缓冲器的时钟延迟后，再通过程序确定最大时钟延迟寄存器附近的缓冲器数量和位置信息和最小时钟延迟附近的缓冲器数量和位置信息，最后确定要移动的缓冲器的数量及位置。缓冲器移动后，同样要通过 Hspice 仿真重新获取每个寄存器的延迟信息，计算全局时钟偏差，看移动缓冲器后是否达到了减少时钟偏差的目的。实验结果表明，对图 4-2 的网格型时钟移动缓冲器后，全局时钟偏差从原来的 50 皮秒降低到 25 皮秒，效果相当明显。

缓冲器重定位可以进行多次，即在已经优化过的网格型时钟中继续进行同样的优化过程，直到如果优化前后时钟偏差变化不大，或者甚至反而变大，则应该停止缓冲器重定位的优化过程。

4.2 本章小结

本章详细阐述了网格型时钟中缓冲器优化的方法----缓冲器重定位。该策略充分利用了网格型时钟低通滤波器的特性----信号在网格型时钟传播时，其强度随着传播距离成指数级衰减，这因为该特性，时钟延迟较大和时钟延迟较小的寄存器有聚集效应，就是说有较大时钟延迟的寄存器在物理位置上是靠近的，同样，有较小时钟延迟的寄存器在物理位置上也是靠近的。所以，具有较少时钟延迟寄存

器附近的缓冲器移动到具有较大时钟延迟的寄存器旁边，就能使原来时钟延迟较大的寄存器时钟延迟变小，原来时钟延迟较小的寄存器时钟延迟变大，从而减少全局时钟偏差。

第五章 实验结果

上面的章节介绍了网格型时钟的整个综合和优化流程，下面通过在学术界标准的电路测试案例中运行本文提出的网格型时钟综合和优化流程，检验该综合和优化流程的效果。

5.1 实验环境及参数设置

上述的网格型时钟的自动化综合和优化流程的整个框架和算法是用 C++实现的，我们把它称为 *Balanced Placement and Routing----平衡布局布线(BPR)*。测试是在一台 CPU 为 2.2G，内存为 2G 的个人电脑上进行的，操作系统是 RedHat Linux。

测试案例是学术界常用的测试电路，其中一部分来源于[44],一部分来源于[31]。但由于半导体技术的飞速发展，某些测试电路的信息已经远远落后于当前的技术，譬如[44]中的电路，芯片尺寸太大及寄存器的时钟输入端电容太大，为了使仿真结果更有参考价值，跟当前的设计相符，我们故意把[44] 中的电路尺寸在横坐标和纵坐标缩小 60 倍，同时把原来寄存器时钟输入端的电容大小换成在一个 90nm 设计中实际的电容大小的值。缓冲器的 SPICE 模型也是从一个真实的 90nm 的设计中提取的，该模型的具体电路网表可以从附录中找到。

上述的自动化综合优化算法中，有不少参数是可以设计者根据设计要求自定义的，如设置缓冲器的数量的参数 ρ ，树干摆放中算法中迭代终止条件参数 α ，局部时钟布线中平衡时钟负载和连线长度的参数η和δ。在测试中，我们做了以下的设置， ρ =0.5； α =0.9；η=1.0；δ=0.01。

输入到缓冲器的时钟信号是一个斜波信号，它的过渡时间是 80 皮秒，时钟频率是1GHz。如下图 5-1 所示：

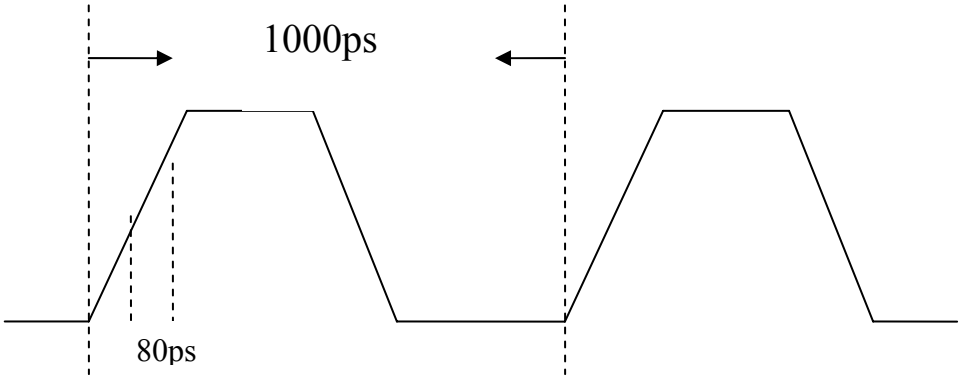


图 5-1 缓冲器的时钟输入信号
Fig.5-1 Input waveform to buffer

在实际电路中，输入到网格型时钟中每个缓冲器的时钟信号的到达时间是不一样的，因为网格型时钟的上一级驱动----时钟树，不可避免的产生时钟偏差，也就导致了时钟信号到达缓冲器时间的不一致性。为了模拟实际电路中的这种情况，我们把到时钟信号到达缓冲器的到达时间模拟成一个随机变量，该随机变量的平均值是 0,最大值是 25ps(皮秒)，最小值是-25p，即到达缓冲器的最大时间差别是 50ps。同时，我们把用于网格规划的时钟延迟和过渡时间约束区间分别设置为[160,220]ps 和[140,200]ps，这些时序约束参数的值是根据[32]中的原则设置的。

为了方便用户使用，使程序的可配置性，灵活性更强，我们把所有的配置参数都设置在一个 MeshSpec.txt 文档中。下面对配置文件中的各项加以阐述。

1. FFPhyFile:xxxxx
   该配置项是指明寄存器物理位置信息(寄存器的横坐标和纵坐标)的输入文件，也就是程序的测试输入。
2. SpiceModel：xxxxx
   xxxxx 文件中包含缓冲器的 SPICE 模型，缓冲器 VDD 的值，以及局部时钟布线中互连线单位长度的电容电阻值，网格型时钟中单位长度树干的电阻和电容值。
3. StemWidth：xxxxx
   Xxxxx 是树干的宽度，默认单位是微米。
4. LocalWidth：xxxxx
   Xxxxx 是局部互连线的宽度，默认单位是微米。

5. AutoPlanning：(true|| false)

如果 AutoPlanning 的值是 true；则表明有程序自动根据约束要求决定水平和垂直树干的数量。如果是 false；则表明手动给出水平和垂直树干的数量。当 AutoPlanning 是 true 时，则要给出（coefdA，coefdB），（coeftA，coeftB）的值，它们分别是图 3-13 中“平均延迟—平均负载”拟合曲线的第一，第二个系数；图 3-16 中“平均过渡时间—平均负载”拟合曲线的第一，第二个系数。这些系数可以通过查找工艺库文件中相应缓冲器的时间特性表获得。同时还要给出 Tran 和 Delay 的约束区间，它们代表过渡时间的约束和时钟延迟的约束。

如果 AutoPlanning 为 false；则要给出 trNum 和 brNum 的值，它们代表水平树干和垂直树干的数量；如果需要，通过设置 location 为 yes，还可以手动给出它们在芯片上具体物理位置，如果不设置，则由程序的算法决定其位置。

6. StemPlace:（uniform || loadWeighted）

该选项确定树干摆放的策略；如果是 uinform,则树干在芯片上是均匀放置的，如果是 loadWeighted 则树干摆放采用 3.5.1 节中的算法进行摆放。

7. BufferLocation：(true || false)

如果值为 true，则要给出缓冲器的位置，该位置是通过矩阵给出的，因为缓冲器只放置在树干的交叉点上，其位置可用矩阵直观表现。如果矩阵中某一元素为 0，则表明相应位置上没有缓冲器，如果为 1，则表明该位置上放有缓冲器。图 5-2 一个网格型时钟的缓冲器摆放实例，其中蓝色实心圆圈的是缓冲器，其对应的矩阵是：

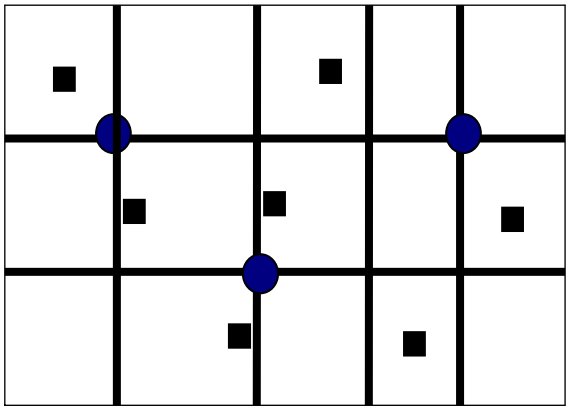


图 5-2 缓冲器摆放实例

Fig. 5-2 An example of buffer placement

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

当 BufferLocation 的值为 false 时，缓冲器采用 3.5.2 中 set-cover 的摆放策略。

8. InputWaveForm

   该选项刻画出斜波输入的信息，要给出其 tran—过渡时间，delay---开始的时刻点，skew----时钟信号到达缓冲器时间的最大差值。

9. MeshBuffer：xxxxx

   Xxxxx 是缓冲器的模型的名字。

10. LocalTree：（true || false）

    是否对寄存器进行cluster[45]，如果为true，则对寄存器进行cluster处理。

5.2 测试结果

我们把上述网格型时钟自动化综合和优化流程应用到表 5-1 中的测试电路中，其中 r1-r5 的电路来源于[44]，s5378-s35932 来源于[31]，如图 5-1 所示，是 r5 测试电路的寄存器分布情况。

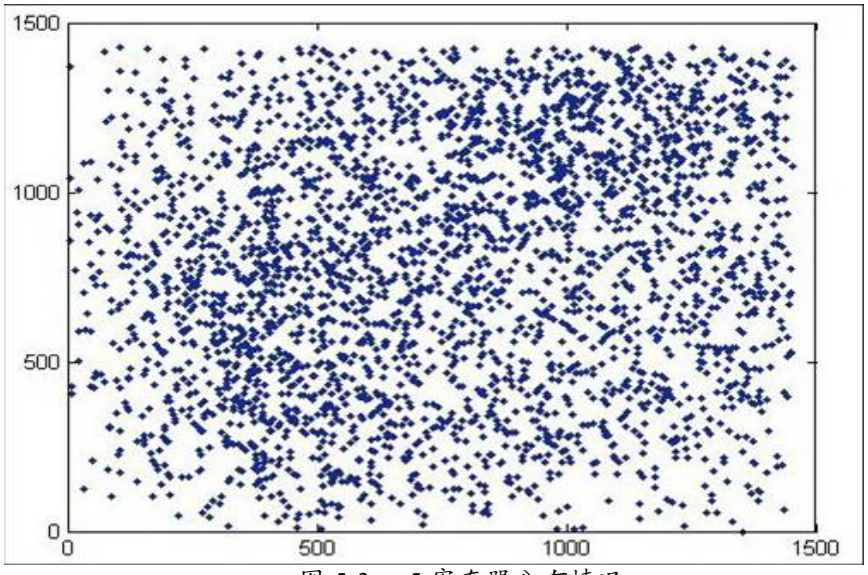


图 5-3　r5 寄存器分布情况

Fig.3-5 FFs distribution of case r5

其中“FFs”列是每个测试电路中寄存器的数量；“Size”列 BPR 综合出来的是水平和垂直树干的数量；“WL”列是局部时钟布线的总长度和树干总长度之和。

表 5-1 实验结果

Cases	FFs	Size	WL(um)	Delay	Tran.	Skew	Pow.(mw)
r1	267	6x6	21099	202.3	175.5	19.9	20.3
r2	598	10x10	45058	195.6	168.3	26.7	39.4
r3	862	10x10	53350	173.5	159.4	28.5	46.2
r4	1903	14x14	102922	191.7	166.3	25.9	93.4
r5	3101	16x16	149283	170.1	158.4	51.4	136.6
s5378	165	5x5	13287	184.6	162.8	20.5	8.7
s13209	500	10x10	42173	173.9	158.2	23.4	25.7
s15850	566	10x10	46214	188.4	164.9	25.1	28.6
s38584	1426	13x13	76539	193.1	169.1	31.2	58.5
s35932	1728	14x14	89835	176.5	160.3	29.3	72.8

“Delay”和“Tran.”列是平均时钟延迟和平均过渡时间，单位是皮秒。“Pow.”列是网格型时钟的总功耗，由于 BPR 运行时间相当的快，在所有的测试案例中都小于 1ms，所以就没有在表格中报告了。同时从表格 5-1 中看到，所有测试例子中的平均延迟时间和平均过渡时间都落在了 5.1 节的约束区间中，这证明了我们在网格规划提出了的结论----“网格型时钟中的‘平均延迟—平均负载’曲线跟单个缓冲器的‘延迟—负载’曲线基本是一样的，只要这两种情况下的缓冲器类型是一样的。”的正确性以及再在此基础上提出的网格规划算法的准确性。

我们把[31,32]中树干在芯片上均匀放置和按“最近原则”进行局部布线的算法称为 *Uniform Placement and Routing----统一布局布线(UPR)*，并对 BPR 与 UPR 进行对比。对比结果在表 5-2 中。

表 5-2 BPR 与 UPR 对比

Cases	WL(um)			Skew		
	UPR	BPR	Inc.%	UPR	BPR	Red.%
r1	21443	21099	-1.6%	22.5	19.9	11.6%
r2	45091	45058	-0.1%	32.8	26.7	18.8%
r3	53086	53350	0.5%	30.6	28.5	6.8%
r4	103238	102922	-0.3%	47.1	25.9	45.0%
r5	150261	149283	-0.7%	57.4	51.4	10.4%
s5378	13351	13287	-0.5%	22.8	20.5	10.1%
s13209	42079	42173	0.2%	27.6	23.4	15.2%
s15850	45805	46214	0.9%	31.2	25.1	19.6%
s38584	77122	76539	-0.8%	41.2	31.2	24.3%
s35932	89835	90110	0.3%	35.6	29.3	17.7%
Avg.			-0.2%			17.9%

表中“Inc.%”为用 BPR 方法比用 UPR 方法，总线长增加的百分比；“Red.%”为用 BPR 方法比用 UPR 方法，时钟偏差的减少百分比。从表中可以看到，我们的网格型时钟综合优化器(BPR)在总线长几乎没有增加的基础上(也意味这功耗没有增加)，全局时钟偏差比 UPR 减少了将近 18%。

我们同时也考察了 BPR 抗干扰能力是否比 UPR 有所增强。如何衡量网格型时钟的抗干扰能力呢？通过故意设置输入波形到达寄存器时间的时钟偏差，就能模拟上一级时钟树产生的时钟偏差。如果输入时钟偏差在变化较大的情况下，输出的时钟偏差都变化不大，则能表明该网格型时钟的抗干扰能力比较强。表 5-3 是在缓冲器输入时钟偏差范围从 50 皮秒变化到 125 皮秒时，由 BPR 和 UPR 综合器综合出来的网格型时钟中寄存器的全局时钟偏差的变化。

表 5-3 输入时钟偏差与输出时钟偏差（1）

Input skew	Output skew					
	Case r3		Case r4		Case r5	
	UPR	**BPR**	UPR	**BPR**	UPR	**BPR**
50	30.6	**28.5**	47.1	**25.9**	57.4	**51.4**
75	33.2	**30.5**	50.3	**28.6**	55.6	**49.1**
100	30.8	**28.9**	43.4	**28.3**	56.4	**51.2**
125	28.0	**27.8**	42.2	**24.7**	49.0	**48.7**
Max variation	5.2	**2.7**	8.1	**3.9**	8.4	**2.7**

表 5-4 输入时钟偏差与输出时钟偏差（2）

Input skew	Output skew					
	s15850		s38584		s35932	
	UPR	**BPR**	UPR	**BPR**	UPR	**BPR**
50	31.2	**25.1**	41.2	**31.2**	35.6	**29.3**
75	30.3	**26.3**	43.7	**32.1**	34.1	**30.3**
100	35.7	**25.4**	38.2	**31.5**	31.8	**32.5**
125	29.5	**27.6**	46.7	**33.3**	43.6	**30.6**
Max variation	6.2	**2.5**	8.5	**2.1**	8.0	**3.2**

Input skew 是缓冲器的输入时钟偏差，Output skew 是寄存器的时钟偏差，即全局时钟偏差。Max variation 是输入时钟偏差从 50 皮秒变化到 125 皮秒时，Output skew 的每一列中时钟偏差的最大差值。从表 5-3 和 5-4 中很容易看到，BRP 的 Max variation 远小于 UPR。这说明通过 BPR 综合出来的网格型时钟的抗干扰能力比 UPR 强。

从第四章缓冲器位置重定位中，我们知道，时钟延迟较大和时钟延迟较小的缓冲器都是有位置上的集聚效应的。图 5-4 是寄存器时钟延迟分布的一个例子，纵坐标是该具有横坐标时钟延迟的寄存器数量占总数的百分比。从图中可以看到寄存器时钟延迟的分布是没有规律的。

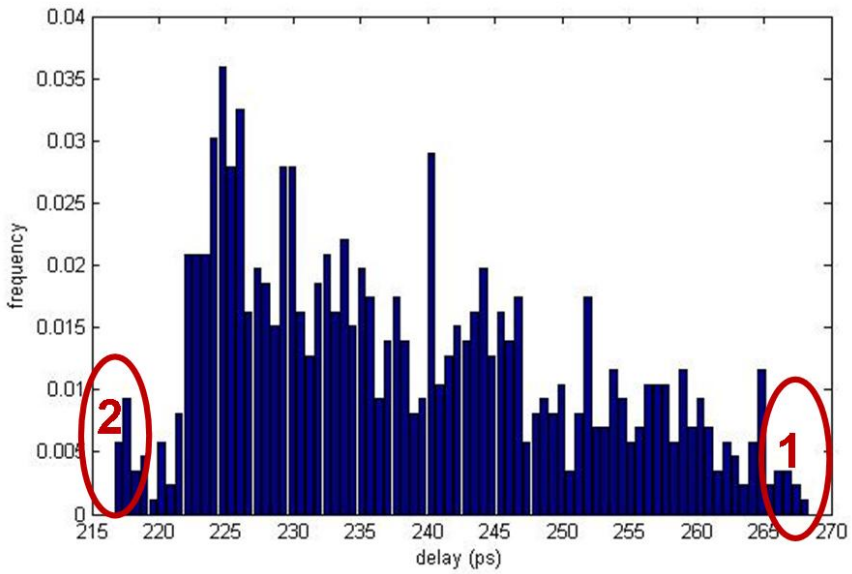


图 5-4 时钟延迟频率分布图(1)
Fig.5-4 Delay distribution(1)

图 5-4 中用红色椭圆框标记的寄存器间的时钟延迟的差值就是全局时钟偏差，

因为它们分别是具有最大时钟延迟和最小时钟延迟的寄存器。图 5-5 是又一个寄存器时钟延迟分布的例子，从图中可以看到，该例子中的时钟偏差月等于 260-180=80 皮秒。该图也再此说明，寄存器时钟延迟的分布是没有规律的。

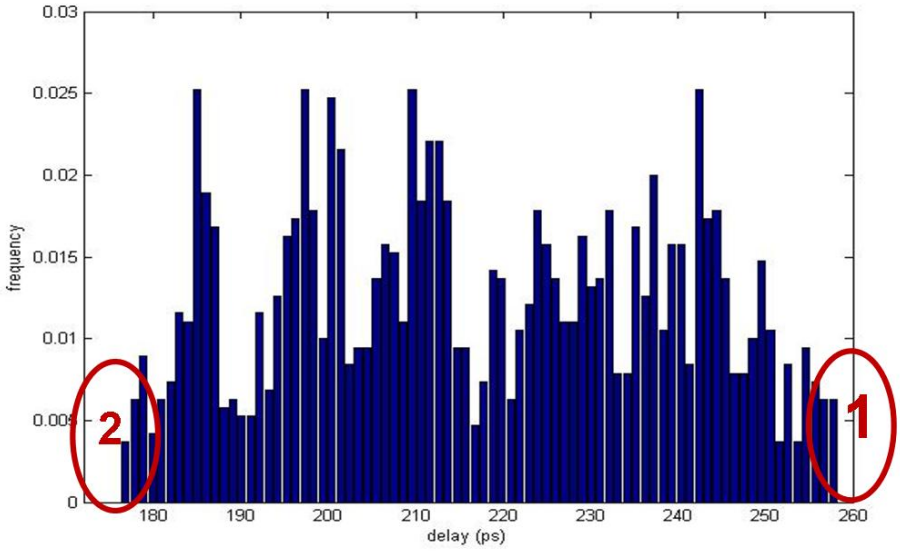


图 5-4 寄存器时钟延迟频率分布图(2)
Fig.5-4 Delay distribution（2）

表 5-5 缓冲器重定位效果

Cases	Initial	Frist	Red.%	Second	Third	CPU(s)
r1	19.9	12.0	39.6%	11.3	11.0	3.7
r2	26.7	15.7	41.0%	10.7	11.6	12.4
r3	28.5	18.6	34.9%	21.8	19.5	13.8
r4	25.9	19.6	24.5%	20.4	22.5	30.2
r5	51.4	21.5	58.2%	17.3	22.2	46.5
s5378	20.5	13.1	36.1%	11.8	12.0	2.1
s13209	23.4	17.2	26.5%	15.2	16.1	9.6
s15850	25.1	15.7	37.5%	13.3	12.8	11.1
s38584	31.2	23.6	24.4%	19.5	21.3	26.2
s35932	29.3	20.1	31.4%	15.6	17.2	28.5
Avg.			35.4%			

为了检验网格型时钟的优化------缓冲器重定位的效果，我们把用 BPR 综合后的网格型时钟进行了 3 次优化工作。优化后的时钟偏差分别在“First”，“Second”和“Third”中，Initial 为优化前的时钟偏差。从表 5-5 看到，只需经过一轮的优化，时钟偏差减少非常明显，达到平均达到 35.4%, 继续第二第三轮的优化效果就

没有第一轮那么明显了。CPU 列列出的是 Hspice 进行一次电路仿真获取时序信息所需要的时间。图 5-5 显示的是通过缓冲器重定位后，测试案例 r3 时钟偏差减少的示意图，蓝色正放形表示的是优化前寄存器时钟延迟的分布，不难看出，优化前，时钟偏差约等于 185-155=30 皮秒；红色圆圈表示的是进行优化后，寄存器时钟延迟分布，不难看出，优化后，时钟偏差约等于 182-162=20 皮秒。缓冲器重定位对于减少时钟偏差效果相当明显。

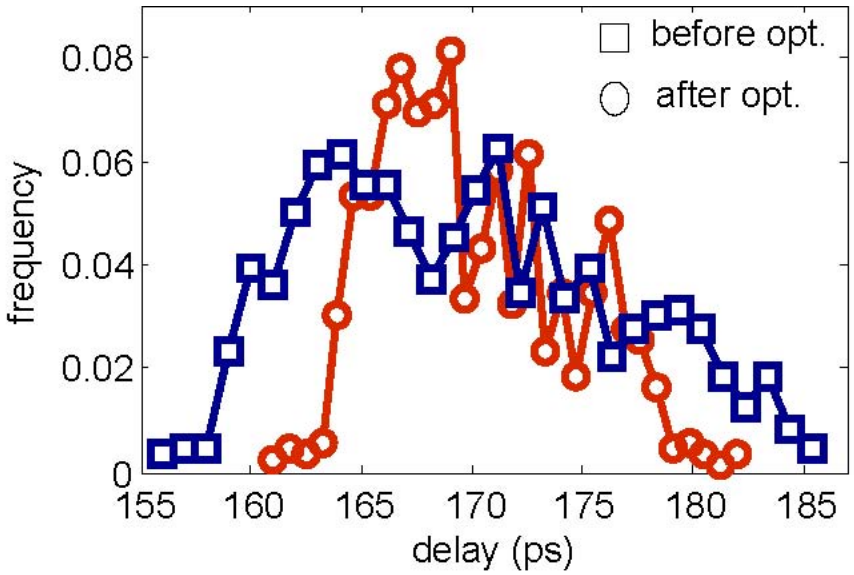


图 5-5 优化前后时寄存器钟延迟分布

Fig.5-5 Delay distributions before and after buffer relocation

5.3 本章小结

本章给出了本文提出的整个网格型时钟自动化综合和优化流程的实验结果，实验结果表明，该通过自动化综合和优化流程出来的网格型时钟，跟学术界已有的设计框架相比，不仅能满足设计约束要求，而且功耗和布线长度不增加的情况下，能获得较少的时钟偏差，而且抗干扰能力较强。通过缓冲器重定位的优化后，还能大大减少时钟网格的时钟偏差。

第六章 总结与展望

6.1 主要工作与创新点

本文提出全新的网格型时钟自动化综合流程，在满足时钟延迟和功耗约束的条件下，使时钟偏差尽可能地小，并给出了关于缓冲器位置优化的方案，这在文献中是从没出现过的。本研究工作的创新点在于：

1. 提出了一种新的决定初始的水平和垂直的互连线（树干）数目的方法，该方法基于缓冲器“平均延时—平均负载”的延迟曲线。

2. 水平和垂直的互连线的位置放置考虑了芯片上寄存器的实际分布。

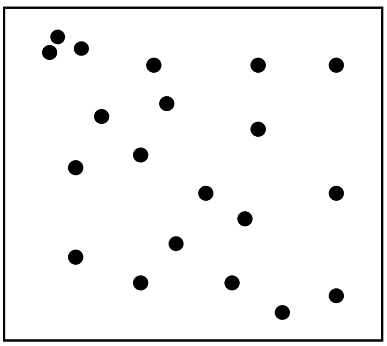
3. 在布线阶段，不同于传统的“最近原则”连线规则，开发一个全新的局部布线器，使网格型时钟上每根互连线上的负载尽量平衡。

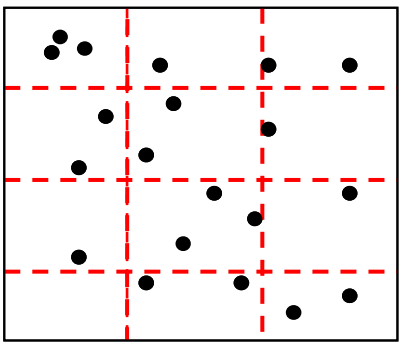
4. 通过后仿真，根据仿真结果对缓冲器进行重定位，即改变缓冲器的位置，减少全局的时钟偏差。

6.2 后续研究工作

为了提高仿真速度和降低功耗，可以采用cluster[45]算法对寄存器进行预处理，即把寄存器先分块，分块后，属于同一个块的寄存器用一棵时钟树（local tree）驱动，对于上一级的时钟网络来说，该分块只有一个local tree的输入电容，这样做不仅能大大简化时钟树的层次化设计，而且对于上一级时钟网络来说，因为它只看到一个输入电容，电路节点大大减少，提高了电路仿真的速度，把寄存器分块的思想符合模块化top-down设计流程。

如何对寄存器进行最合理的分块，这是个 NP 问题。需要考虑多种因素，譬如寄存器间物理位置的远近，分块的大小，局部连线的长度，时钟延迟时钟偏差等。因此需要算法能根据具体的设计要求可进行灵活性的配置。





(a) 寄存器分布　　(b) 寄存器划分
(a) distribution of FFs　　(b) FFs cluster

图 5-1　寄存器分块
Fig.5-1 clustering of FFs

如图 5-1 所示，是一个寄存器分块的例子。该图中采用最简单的位置均匀分块的算法。该算法虽然简单，但存在明显缺陷，有些分块寄存器较多，有些分块寄存器较少，这会在 local tree 这一级产生较大的时钟偏差。

寄存器分块算法一直以来是时钟网络分布的一个难点之一，今后的研究方向是设计出一个合理的寄存器分块算法，整合到现有的网格型时钟的综合优化框架中，使其更完善，更强大。

参 考 文 献

[1] M. Donno, E. Macii and L. Mazzoni, “Power-Aware Clock Tree Planning,” ISPD04, pp.374-384, April 2004.

[2] D. Wann and M. Franklin, “A synchronous and clocked control structures for VLSI based interconnection networks,” IEEE Trans. Computer Aided Design, pp. 284-293, Mar. 1983.

[3] J. M. Rabaey, A. Chandrakasan and B. Nikolie, “Digital Integrated Circuits: A Design Perspective (2nd Edition),” 2004.

[4] Y. Liu, S. R. Nassif, L. T. Pileggi, and A. J. Strojwas, “Impact of interconnect variations on the clock skew of a gigahertz microprocessor,” in Proc. ACM/IEEE Design Automation Conf., pp. 168–171, 2000.

[5] D. A. Joy and M. J. Ciesielski, "Placement for clock period minimization with multiple wave propagation," in Proc. ACM/IEEE 28th Design Automation Conf., pp. 640-643, June 1991.

[6] R. Saleh, S. Z. Hussain, S. Rochel, and D. Overhauser, “Clock skew verification in the presence of IR-drop in the power distribution network,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 19, pp. 635–644, Jun. 2000.

[7] S. Pullela, N. Menezes, and L. T. Pillage, “Reliable non-zero skew clock trees using wire width optimization,” in Proc. ACM/IEEE Design Automation Conf., pp. 165–170, 1993.

[8] J. Grad, “Cadence Interoperability using OpenAccess 2.0,” http://www.chiptalk.org, Oct, 2005.

[9] E. G. Friedman, “Clock Distribution Networks in Synchronous Digital Integrated Circuits,” Proceedings of the IEEE Volume 89, pp. 665-692, May 2001.

[10] 汪珺, 陆生礼. 基于Garfield5设计中时钟树综合技术研究[硕士论文]. 东南大学.2006.

[11] S. Lin and C. K. Wong, “Process-variation-tolerant clock skew minimization,” in Proc. IEEE/ACM Int. Conf. Computer-Aided Design, pp. 284–288, 1994.

[12] 陈彦白，李文宏. Fishbone 和 CTS 时钟树的比较[硕士论文]. 复旦大学.2008.

[13] D. Wann and M. Franklin, “Asynchronous and clocked control structures for VLSI based interconnection networks,” IEEE Trans. Comput., vol. C-32, pp. 284–293, Mar. 1983.

[14] H. B. Bakoglu, “Circuits, Interconnections, and Packaging for VLSI,” MA: Addison Wesley, 1990.

[15] H. B. Bakoglu, J. T. Walker, and J. D. Meindl, “A symmetric clock-distribution tree and optimized high-speed interconnections for re-duced clock skew in ULSI

and WSI circuits," in Proc. IEEE Int.Conf. Computer Design, pp. 118–122, Oct. 1986.

[16] D. C. Keezer and V. K. Jain, "Clock distribution strategies for WSI:A critical survey," in Proc. IEEE Int. Conf. Wafer Scale Integration, pp. 277–283, Jan. 1991.

[17] D. Somasekhar and V. Visvanathan, "A 230-MHz half-bit level pipelined multiplier using true single-phase clocking," IEEE Trans.VLSI Syst., vol. 1, pp. 415–422, Dec. 1993.

[18] E. G. Friedman and S. Powell, "Design and analysis of a hierarchical clock distribution system for synchronous standard cell/macrocell VLSI," IEEE J. Solid-State Circuits, vol. SC-21, pp. 240–246, Apr.1986.

[19] D.Mijuskovic, "Clock distribution in application specific integrated circuits," Microelectron. J., vol. 18, pp. 15–27, July 1987.

[20] D.W. Bailey and B. J. Benscheneider. "Clocking Design and Analysis for a 600-MHz Alpha Microprocessor," In IEEE Journal of Solid-State Circuitsm, Vol 33, pp. 1627-1633, November 1998.

[21] G. Northrop et. al."A 600-MHz G5 S/390 Microprocessor," In ISSCC Tech. Dig., pages 88–89, February 1999.

[22] P. J. Restle et. Al, "The Clock Distribution of the Power4 Microprocessor," In ISSCC Dig. Tech. Papers, pages 144–145, February 2002.

[23] X.Ye et. al, "Accelerating Clock Mesh Simulation Using Matrix-Level Macromodels and Dynamic Time Step Rounding," in 9th ISQE, pp.627-632, 2008.

[24] S. M. Reddy, G. R. Wilkem and R. Murgai, "Analyzing Timing Uncertainty in Mesh-based Clock Architectures," in Proc. of DATE, pp. 1097-1102, 2006.

[25] W. Shen, Y.Cai, X. Hong and J. Hu, "Activity-Aware Registers Placement for Low Power Gated Clock Tree Construction," IEEE Computer Society Annual Symposium on VLSI, pp. 383-388, 2007.

[26] L. Vandenberghe, S. Boyd, and A. E. Gamal. "Optimal Wire and Transistor Sizing for Circuits with Non-tree Topology," In ICCAD, pages 252–259, 1997.

[27] R. Heald et. al, "Implementation of a 3rd-Generation SPARC V9 64b Microprocessor," In ISSCC Dig. Tech. Papers, pages 412–413, February 2000.

[28] H. Chen, C. Yeh, G. Wilke, S. Reddy, H. Nguyen, W. Walker, and R.Murgai, "A SlidingWindow Scheme for Accurate Clock Mesh Analysis," in Proc. of ICCAD, pp. 939-946, 2005.

[29] L. Pillage and R. Rohrer, "Asymptotic waveform evaluation for timing analysis. IEEE Trans. Computer-Aided Design," pp.352-366, April 1990.

[30] P. Li et. al, "Analysis of large clock meshs via harmonic-weighted model order

reduction and port sliding," in ICCAD, pp. 627-631, 2007.

[31] A. Rajaram, D. Z. Pan "MeshWorks: an efficient framework for planning, synthesis and optimization of clock mesh networks," in ASPDC, pp.250-257, 2008.

[32] G. Venkataraman, Z. Feng, J. Hu, P. Li, "Combinatorial Algorithms for Fast Clock Mesh Optimization," in Proc. of ICCAD, pp. 79–84, 2006.

[33] V. V. Vazirani. Approximation Algorithms. Springer 2001.

[34] H. Kerivin and A. R. Mahjoub, "Design of Survivable Networks: A survey," In Networks, pages 1–21, April 2005.

[35] J. Qian, S. Pullela, and L. Pillage, "Modeling the effective capacitance for the RC interconnect of CMOS gates," IEEE Trans. on Computer-Aided Design, pp.1526-1535, 1994.

[36] J. Cong and Y. Zhang, "Thermal-driven multilevel routing for 3-dics," in Proc. IEEE Asia South Pacific Design Automation Conference, pp. 121-126, Jan. 2005.

[37] H. Yao, Y.Cai, X.Hong, "CMP-aware Maze Routing Algorithm for Yield Enhancement," ISVLSI, pp. 239-244, March 2007.

[38] H. Yao, Y.Cai, X.Hong, "Congestion-Driven Multilevel Full-Chip Routing Framework," Tsinghua Science & Technology, Volume 13, Issue 6, Pages 843-849, 2008.

[39] 李小南. 多级自动布线框架中的时延平衡研究 [硕士论文]. 上海交通大学. 2008.

[40] C. Y. Lee, "an Algorithm for Connections and Its Applications," IRE Trans. On Electronic Computers, pp. 346-365, 1961.

[41] S. B. Akers, "A Modification of Lee's Path Connection Algorithm, IEEE Trans. On Electronic Computers," pp. 97-98, 1967.

[42] J. Soukup, "Fast maze router," Proc. DAC, pp. 100-102, June 1978.

[43] Hadlock ,"A Shortest Path Algorithm for Grid Graphs," Networks, pp.323-334, 1977.

[44] J. Cong, A. B. Kahng, C. K. Koh and C. W. A. Tsao, "Bounded-skew clock and steiner routing under Elmore delay," Proc. IEEE Int. Conf. on Computer-Aided Design, pp. 66-71, 1995.

[45] R. S. Shelar, "An Efficient Clustering Algorithm for Low Power Clock Tree Synthesis", International Symposium on Physical Design, pp. 181-188, 2007.

附录 1 缓冲器 SPICE 模型

MeshBuffer:

.SUBCKT TR3BUFZAXT A VDD VSS Y

* devices:

MI118/Mmn0 VSS A NODE91 VSS NM l=0.1u w=0.96u ad=0.1008p as=0.2064p

+pd=1.17u ps=2.35u nrd=0.0989583 nrs=0.0989583 $sa=280n $sb=4.68u

+$model=mn_m

MI117/Mmn0_SMASHED1 NODE91 A VSS VSS NM l=0.1u w=0.96u ad=0.1008p

+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=720n

+$sb=4.24u $model=mn_n

MI117/Mmn0_SMASHED2 VSS A NODE91 VSS NM l=0.1u w=0.96u ad=0.1008p

+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=1.16u

+$sb=3.8u $model=mn_n

MI114/Mmn0 Y NODE91 VSS VSS NM l=0.1u w=0.96u ad=0.1008p as=0.1008p

+pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=1.6u $sb=3.36u

+$model=mn_n

MI111/Mmn0_SMASHED1 VSS NODE91 Y VSS NM l=0.1u w=0.96u ad=0.1008p

+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=2.04u

+$sb=2.92u $model=mn_n

MI111/Mmn0_SMASHED2 Y NODE91 VSS VSS NM l=0.1u w=0.96u ad=0.1008p

+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=2.48u

+$sb=2.48u $model=mn_n

MI111/Mmn0_SMASHED3 VSS NODE91 Y VSS NM l=0.1u w=0.96u ad=0.1008p

+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=2.92u

+$sb=2.04u $model=mn_n

MI111/Mmn0_SMASHED4 Y NODE91 VSS VSS NM l=0.1u w=0.96u

ad=0.1008p
+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=3.36u
+$sb=1.6u $model=mn_n
MI111/Mmn0_SMASHED5 VSS NODE91 Y VSS NM l=0.1u w=0.96u ad=0.1008p
+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=3.8u
+$sb=1.16u $model=mn_n
MI111/Mmn0_SMASHED6 Y NODE91 VSS VSS NM l=0.1u w=0.96u ad=0.1008p
+as=0.1008p pd=1.17u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=4.24u
+$sb=720n $model=mn_n
MI111/Mmn0_SMASHED7 VSS NODE91 Y VSS NM l=0.1u w=0.96u ad=0.2064p
+as=0.1008p pd=2.35u ps=1.17u nrd=0.0989583 nrs=0.0989583 $sa=4.68u
+$sb=280n $model=mn_m
MI80/Mmp0 VDD A NODE91 VDD PM l=0.1u w=1.46u ad=0.1533p as=0.3139p
+pd=1.67u ps=3.35u nrd=0.0650685 nrs=0.0650685 $sa=280n $sb=4.68u
+$model=mp_m
MI86/Mmp0_SMASHED1 NODE91 A VDD VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=720n
+$sb=4.24u $model=mp_n
MI86/Mmp0_SMASHED2 VDD A NODE91 VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=1.16u
+$sb=3.8u $model=mp_n
MI104/Mmp0 Y NODE91 VDD VDD PM l=0.1u w=1.46u ad=0.1533p as=0.1533p
+pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=1.6u $sb=3.36u
+$model=mp_n
MI109/Mmp0_SMASHED1 VDD NODE91 Y VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=2.04u
+$sb=2.92u $model=mp_n
MI109/Mmp0_SMASHED2 Y NODE91 VDD VDD PM l=0.1u w=1.46u

ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=2.48u
+$sb=2.48u $model=mp_n
MI109/Mmp0_SMASHED3 VDD NODE91 Y VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=2.92u
+$sb=2.04u $model=mp_n
MI109/Mmp0_SMASHED4 Y NODE91 VDD VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=3.36u
+$sb=1.6u $model=mp_n
MI109/Mmp0_SMASHED5 VDD NODE91 Y VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=3.8u
+$sb=1.16u $model=mp_n
MI109/Mmp0_SMASHED6 Y NODE91 VDD VDD PM l=0.1u w=1.46u ad=0.1533p
+as=0.1533p pd=1.67u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=4.24u
+$sb=720n $model=mp_n
MI109/Mmp0_SMASHED7 VDD NODE91 Y VDD PM l=0.1u w=1.46u ad=0.3139p
+as=0.1533p pd=3.35u ps=1.67u nrd=0.0650685 nrs=0.0650685 $sa=4.68u
+$sb=280n $model=mp_m
C1 VSS Y 6.582e-16
C2 VDD Y 9.146e-16
C3 VDD VSS 5.753e-17
C4 NODE91 Y 2.261e-15
C5 NODE91 VDD 1.626e-15
C6 NODE91 VSS 6.704e-16
C7 A NODE91 1.132e-15
C8 A VDD 3.987e-16
C9 A VSS 2.414e-16
C10 A Y 1.192e-17
C11 0 VSS 2.55e-16
C12 0 VDD 2.577e-16
C13 0 Y 5.034e-17

```
    C14     0    NODE91     7.806e-16
    C15     0    A    3.547e-16
.ENDS
```

附录 2 程序输入配置文件

```
FFPhyFile: mr1.txt
SpiceModel: Spice.txt

StemWidth: 1.08
LocalWidth: 0.18

AutoPlanning: ture

StemPlace: loadWeighted

BufferLocation: false

InputWaveForm
   Delay: 0
   Skew: 50
   Tran: 80

MeshBuffer: TR3BUFZAXT
LocalTree: false
```

附录 3 网格型时钟的 Spice 网表例子

```
* clock mesh netlist

.inc './l90n.mdl'

.param VDD=1.200000
V0 vdd 0 VDD
.global gnd vdd

***
VIN IN 0 PWL (0ps 0 133ps VDD 500ps VDD 633ps 0 1000ps 0)
VIN0 IN0 0 PWL (41ps 0 174ps VDD 541ps VDD 674ps 0 1000ps 0)
VIN1 IN1 0 PWL (17ps 0 150ps VDD 517ps VDD 650ps 0 1000ps 0)
VIN2 IN2 0 PWL (34ps 0 167ps VDD 534ps VDD 667ps 0 1000ps 0)
VIN3 IN3 0 PWL (0ps 0 133ps VDD 500ps VDD 633ps 0 1000ps 0)
VIN4 IN4 0 PWL (19ps 0 152ps VDD 519ps VDD 652ps 0 1000ps 0)
......省略电路输入部分.....
...............

********
.options list node post

.SUBCKT TR3BUFZAXT A    VDD    VSS    Y
* devices:
.....  缓冲器的 spice 模型.....

.ENDS

R1    1    2    2.373626
C1    1    0    0.010179p
C2    2    0    0.010179p
R2    2    3    10.000000
C3    2    0    0.002482p
C4    3    0    0.005982p
```

```
R3    2    4    0.703297
C5    2    0    0.003016p
C6    4    0    0.003016p
R4    4    5    19.444444
C7    4    0    0.004826p
C8    5    0    0.008327p
R5    4    6    3.516483

....省略网格型时钟网表部分....
.....................

Xdrive0 IN0 vdd gnd 10 TR3BUFZAXT
Xdrive1 IN1 vdd gnd 18 TR3BUFZAXT
Xdrive2 IN2 vdd gnd 30 TR3BUFZAXT

.........例化缓冲器.............
.....................................
.tran 1ps 1000ps


.meas tran Source trig v(IN) val='0.2*VDD' rise=1  targ v(IN) val='0.8*VDD'
rise=1
.meas tran FF0 trig v(141) val='0.2*VDD' rise=1  targ v(141) val='0.8*VDD'
rise=1
.meas tran FFd0 trig v(IN) val='0.5*VDD' rise=1  targ v(141) val='0.5*VDD'
rise=1
.meas tran FF1 trig v(528) val='0.2*VDD' rise=1  targ v(528) val='0.8*VDD'
rise=1
.........  获取每个寄存器的时钟延迟和过渡时间  .................


.meas tran avgvalI AVG I(V0) FROM=0ps TO=1000ps
.meas pow PARAM='avgvalI*VDD'
.........  获取整个网格型时钟的功耗  ........

.end
```

致　谢

首先，感谢我的导师施国勇教授，是施教授的谆谆导引让我走进了 EDA 领域的研究；也是在施教授的悉心指导和支持下，我才能得以完成论文中的工作。施老师带领的研究小组的组会极大的拓宽了我的知识范围，使我对 VLSI 物理设计、BDD、模型降阶等方面的研究前沿都有所了解。在这两年的学习研究中，受恩师的严谨治学态度和广博学识的直接熏陶，使我在人格道德和个人综合能力方面都得到了突破，受益良多。

在此，我也要感谢我的身边的同学，他们给了我很多无私的帮助，也让我从他们身上学到各种各样的闪光点，让我思考甚多。

最后要谢谢我的父母和家人对我的支持与鼓励，让我能更专心于学习和研究。

在此，仅以此篇论文献给所有关心和支持我的人，愿你们能一起分享这份喜悦。

攻读硕士学位期间已发表或录用的论文

[1] W. Huang, G. Shi, “heuristic technique for automatic synthesis of clock mesh,” International Symposium on Integrated Circuit, pp. 405 – 408, 2009.