申请上海交通大学工程硕士专业学位论文

多级自动布线框架中的时延平衡研究

学　　校： 上海交通大学

院　　系： 微电子学院

班　　级： Z0521091

学　　号： 1052109082

工程硕士生： 李小南

工程领域： 软件工程

导　师： 施国勇（教授）

上海交通大学微电子学院

2007 年 11 月

此研究部分由国家自然科学基金（No. 60572028)和
上海市浦江人才计划（课题编号 07pj14053）资助。

A Dissertation Submitted to Shanghai Jiao Tong University for

Master Degree of Engineering

A SURVEY ON TIMING BALANCE BASED ON MULTILEVEL ROUTING FRAMEWORK

Author：Xiaonan Li

Specialty: Software Engineering

Advisor: Prof. Guoyong Shi

School of Microelectronics

Shanghai Jiao Tong University

Shanghai, P.R.China

November 18, 2007

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，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。

保密□，在__年解密后适用本授权书。

本学位论文属于

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

多级自动布线框架中的时延平衡研究

摘 要

随着科技的进步，芯片制造的技术进入了深亚微米的时代，伴随而产生的问题也越来越多。在一个超大规模集成电路中，晶体管的长度越来越小，电路的工作频率越来越高，以前可以忽略的互连线的延时现在已经不能再忽略了，并且在未来的芯片设计中显得越来越重要，有预料到将来的芯片设计不再是逻辑设计而是互连设计了。所以现在作芯片的自动布线时，所要考虑的问题不是只有可布通率，还需要考虑时序收敛等问题。

本论文在一个著名的多级自动布线框架中的基础上，对提高布线的时延性能做了一些改正的研究，提出了一种新的方法－－即在布线的过程中采用一种平衡树的思想，使得关键线网的上的各个漏点（Sink）的时延趋于平衡，这种平衡的过程将极大地改善最坏路径上的时延，且使得总的布线长度没有增加或者只有微量的增加。

在实验的结果中，我们可以看到在采用改进的时延驱动算法要比一般的时延驱动算法布线结果要好。

关键词：自动布线、最小生成树、多级布线、时延

A SURVEY ON TIMING MODIFICATION BASED ON MULTILEVEL ROUTING FRAMEWORK

ABSTRACT

As the dramatically development of modern industry, the technology of IC manufactory has been upgraded in to the era of sub-micron. With the development, many problems have been brought up. In a VLSI, length of the transistor is becoming smaller and smaller, the working frequency of circuit is higher and higher. The delay of interconnect, as it can be ignored before, should be paid a lot of attention now, and is becoming more and more important in the future IC design. As far as we can see in the future, it is not logic design, but interconnects design. As a result, when doing routing in IC， not only routability, but a lot other questions, such as timing closure, should be considered.

This thesis is based on a novel multilevel routing framework. It has modified enhancing timing performance and brought up a new method, as we call balance tree. The changes made target node delay on critical net reach the balance. This process of reaching balance could greatly improve the timing performance of the worst path, and the total wire length would have no incensement or just a litter incensement. In the result, we could see that the improved timing driven algorithm is better than those before.

Keywords: routing、MST、multilevel routing、timing

目 录

图片目录

第一章 前言

1.1 布线研究前景

集成电路一直在迅速发展，制造工艺在不断进步，现在已经进入了深亚微米时代。电路设计规模也越来越大，由超大规模（VLSI）、甚大规模（ULSI）向极大规模集成（GSI）规模发展。越来越多的功能，甚至是整个系统都被集成到单个芯片之中，出现了系统级芯片（SOC）的设计概念。于是，作为物理设计[8] [9]（physical design，layout）中的重要阶段的布线（routing），其算法研究与工具设计面临巨大挑战。其中一个挑战是：随着集成度更高，芯片上模块和互连线的排列更加紧密，互联线的间距进一步减少；元件数目的增加和线宽的缩少使互连线的相对长度大大增加；电路工作频率更高。这都使得集成电路中互连线的时延和耦合效应明显。因此，在布线阶段能恰当有效地估计线网长度，拥塞和串扰情况，并使布线长尽量短（包括总线长最短和最长路径最短），线网尽量均匀分布，串扰尽可能小，这是目前研究的热点。

在 VLSI 电路设计中，物理设计阶段的布线是一个非常复杂的问题。经过多年的发展，通常布线分为两个步骤：总体布线和详细布线。在总体布线阶段，通常的思想是将整个芯片划分为 N×N 的块(tiles)，并将多端点线网分解成为二端点线网，然后找出所有 2 端点线网的块到块的路径。一些传统的算法如[4]，他们都无法处理很大的问题，特别是现在工艺发展到 90nm 以下，集成电路的规模越来越大。因此，2～3 级的层级布线应运而生，超大的布线问题然而还是无法很好地解决，所以便有了多级布线框架的产生[1，2]。多级布线算法不同于以往的考虑时延或者拥挤的布线算法[1] [2] [3] [7]，它们将布线区域一级级地划分，并将总体布线，详细布线，延时和拥挤估计集成在每一级中，这样每一级布线，时延或拥挤估计的精度都非常高，当布线传递到下一级时，高精度的上一级布线结果和资源估计对下一级布线将有着良好的指导作用，并且多级的方法能处理很大的布线问题。

1.2 布线算法思想介绍

当 VLSI 电路的芯片布局结束后，电路模块的绝对位置和模块引脚的位置已经确定，此后就要实现模块间的连接。通常有两种策略实现布局后的布线，即直接的区域布线和分两步实现的总体布线和详细布线。布线问题从不同的角度来划分则有不同的布线方式：就布线的对象来分，它可分为面向线网的布线和面向区域的布线，面向线网的布线主要以线网作为考虑对象，如总体布线和区域布线都属于面向线网的布线；而面向区域的布线主要以布线区域作为考虑对象，如两边通道和开关盒都属于面向区域的布线。

区域布线是直接在一个区域内（可多层）进行布线。区域可大至整个芯片，也可以是一个子区域。其特点是线网的引线端可分布在区域边界，也可在区域内。总体布线是在布局后产生的总体布线图上完成线网的分配，但不确定具体的走线位置。详细布线则根据布线对象不同，可以是通道布线，也可以是区域布线。通道布线是在总体布线的基础上在一个矩形区域内确定线网的走线位置。从数学上讲，区域布线和总体布线是一类问题，而且许多算法是相通的。当总体布线图上所有边的容量均不超过 1 时，它就是一个基于网格的区域布线问题。下面讲的线网布线是总体布线和区域布线的基础。

1.2.1 总体布线介绍

两端点线网的总体布线[14…16][20…22]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 $\forall N_i \in N, 1\le i\le n$，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 $U\ (e_i) \le (e_j)$， $\forall e_j \in E$ (2.1)

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：

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

如果 e_j 在 T_i 中则 $x_{ij} = 1$ 否则 $x_{ij} = 0$ 。

总体布线[7,8,9,13,14,15]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 routing method) 、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。

针对串行算法的顺序依赖性问题，Vanneli 设计出一种纯并行算法－－线性规划法。这种算法找到各个线网所有可能的连接树及其代价（布线长度或拥挤度的代价），再根据总体布线单元边与单元容量限制列出线性方程组，最终找到最优解。这种方法的最大难处在于其极高的布线复杂度。

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。

基于货物流（MCNF）理论的算法就是先把总体布线问题经过转化定义成一个网络流的问题，然后对该问题实施启发式算法的搜索。总体问题可以简要描述为：对于图 $G=(V, E, C)$，V 代表顶点，E 代表边集，C 代表边权，$C: E \to R^+$ 即 $C(e_i)$ 为边 e_i 的权重（例如反映实际问题中的的路径长度，容量，密度，拥挤度等）。令 D 代表一组需要连通的目标点集，$D \subset V$。则 MCNF 问题即为求 G 的子树 T，使得 T 连通所有 D 并最小化目标函数 $\min \sum (C(e_i))$。这样问题实际上是一组带约束的 Steiner 森林[10,11,12]问题，Steiner 树问题就已被证明为 NP 困难的了，因此采用各种启发式算法是目前唯一可行的方法。

1.2.2 详细布线介绍

（1） 迷宫算法

Lee[3]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[4]，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(maze routing 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 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 于 1997 年提出，称为 Hadlock 最小迂回算法。

（2）线探索法

迷宫算法的最大的缺点是它要搜索较大的布图空间，所以要花费较大的时间和存储空间。为了避免大量的网格空间的搜索人们想到了简单图形搜索方法。Hightower[5]在 1969 年，Mikami 和 Tabuchi[6]在 1986 年分别提出了线探索(line-probe algorithm)的思想和算法。这种算法的基本思想就是用线探索的方法减少存储空间和计算时间，因为探索用到的空间要比探索用的网格所存储的空间少得多。

（3）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。

1.3 论文完成的工作和内容安排

在论文中，主要研究了物理设计中的布线问题以及如何提高布线的时延性能，全文组织如下：第二章介绍了布线问题的描述方法，概要说明了研究目的和论文中用的改正算法；第三章对多级自动布线框架做了比较详细的介绍，介绍了基本的算法和数据结构，Elmore 时延以及程序结构；第四章详细介绍了最短距离生成树和平衡树的时延修正，其中详细介绍了最小生成树算法的基本原理，最短距离生成树以及平衡树的时延修正；第五章给出了实验的数据和相关的实验结果，对改正的性能做了比较。最后在第六章，对作者所做的工作进行了总结和展望。

第二章 问题描述方法

2.1 研究目的

由于传统的布线算法已经渐渐无法处理目前的超大规模的集成电路布线设计，并且在布线过程中的时序和布通率的收敛无法快速或者无法达到预期的目标。如果在布线过程中，我们仅仅只考虑布通率，而不考虑线网的时延信息，则布线的结果仍无法达到预期的布线目的，所以在布线的过程中，我们需要认真分析各个线网的时延，特别是关键线网的时延，这样我们就能有效控制整个电路设计的时延性能。

对于一些非关键路径的线网，我们可以放宽约束，因为即使导线时延变大一些，但是仍然不足以影响电路的时序。

另一个重要的问题就是电路设计中布线的拥挤问题。如果在布线过程中无法布通所有的导线，那么这个设计是无法继续到下一步的，是不可能流片成功的。所以我们现在布线的主要研究方向就是在把握好线网的时延前提下，还必须做拥挤度分析，以提高布线的可布通性。在这方面我们可以使用目前流行的布线框架－－多级自动布线框架。这种方法首先处理局部小区域的线网，再一次一级一级地处理大区域地线网，并且没一级我们都可以做详细地拥挤度和时延估计，这使得布通率和时序性能得到很好地保证。

2.2 问题描述

本文的主要目的是在已有的多级布线方法的框架基础上[1]，对时间延时做出更加精确的分析，采用平衡树的思想，使得关键线网的上的各个漏点(Sink)的时延趋于平衡，这种平衡的过程将极大地改善最坏路径上的时延，且使得总的布线长度没有增加或者只有微量的增加。并且在时延仍然不满足时延约束时，我们采用最小增量的改进方法，使得时延得到满足，且总线长增加达到最小。

实现是基于部分开放的源码，它们是台湾大学 CAD 实验室提供的基于多级布线框架的部分源程序[1],UCLA CAD 实验室提供的 Layout Database library 和测试例子，

以及 LEDA 算法软件包[6]。

输入：

UCLA CAD 实验室提供的测试例子，这些例子包含有标准单元信息，电路布局后的网表信息，这主要包括各个标准单元的端点位置信息和各个端点间的互连信息。本文采用了 UCLA CAD 实验室的提供的数据管理包来管理设计的输入。

输出：

输出布线后的电路信息，并报告所有线网的时延及其运行时间情况。

目的：

考虑关键路径的布线，利用改正的时延驱动算法使得关键路径的延时符合设计需求，并利用拥挤度估计来提高布线的成功率。

2.3 时延平衡算法和最小距离生成树介绍

在集成电路设计中，能不能有效处理时延是芯片设计中的关键部分。在布线中，虽然所有的标准单元已经在布局阶段布置妥当，但是电路完整的设计还是在成功布线之后，使得在布局中预估的布线延时不一定就能保证布线后的设计仍能满足时延约束。因为在布线过程中，由于线网展开，关键路径中的漏点到源点的距离不一定是最小的 Manhattan 距离，这是由于 Steiner 树的展开并不是按照最小路径生成树展开的，这样漏点到源点的负载也有很大变化，且布线过程中可能由于绕线而使得布线路径会更加长，这些都可能导致附加的时延。我们在布线的过程中需要分析时延且尽可能少地增加线长以使得附加时延在预期的要求之内。详细内容将在下一章介绍。

另一方面，台湾大学的多级自动布线框架上在总体布线时采用了最小生成树算法，最小生成树无法处理源点和漏点之间的关系，即无法保证所有的漏点到源点之间的距离是相对最短的。所以我们采用一种到源点距离最短的最小距离生成树算法来降低时序的要求。

本论文提出了一种新的方法－－即在布线的过程中采用一种平衡树的思想，使得关键线网的上的各个漏点的时延趋于平衡，这种平衡的过程将极大地改善最坏路径上的时延，且使得总的布线长度没有增加或者只有微量的增加。

这两个改正的算法的优点是使得时延估计和处理更加合理，且增量的修正方法更适合满足对布线时延的要求，且能照顾到总共的布线长度，相应地降低了绕线的面积。

2.4 本章小结

本章介绍了改论文的研究目的，即如何有效控制整个电路设计的时延性能和总的布线长度，这将直接影响布线的成功率。并且引入了自动布线框架，简要说明了程序的输入输出，最后介绍了通过时延平衡算法和最小距离生成树来达到时延驱动布线的目的。

第三章 Multilevel 布线算法介绍

本章介绍该多级布线框架中的一些主要算法以及处理流程[1…3] [26…30]。

3.1 算法介绍

算法总共包括 3 个部分，（一）读入数据并初始化数据结构，（二）粗化布线，（三）细化布线。在粗化布线和细化布线中均包含了总体布线和详细布线，下面将一一做详细介绍。

第一步，读入数据并初始化数据结构

采用 UCLA CAD 实验室提供的 Layoutdb.h 和 Layoutdb.o 读取电路布局后的网表文件（.gdf 格式），并一一初始化所有的 Cell Instance, Net, Cell Instance Pin, Edge 等数据结构，并引用 LEDA 包中提供的图结构，将每组线网中的边和顶点情况用图表示。

第二步，粗化布线

将布线区域划分成一个 N*N（N 为 2 的 n 次方）的小区域，为每一个仅在这些单一的小区域的局部线网分别做总体布线和详细布线，对一些无法布通的线网进行标记，当布完后分析每一个区域内的拥挤情况，并分析线网的时延情况。当第一级做完后，布线进入下一级，即将 2*2 的小块合成一个大块，整个布线区域划分成了 N/2*N/2 个小区域，在新的小块中重复前面的步骤。一级一级布线后，知道所有的小块合并到不能再合并，即已经是整个布线区域了。则布线进行到细化布线阶段。

第二步，细化布线

由于粗化布线阶段布线方式仅限于一些简单的布线方法，例如水平线、垂直线、L 型布线、Z 型布线等。这些简单布线方法可能使得一些线网无法布通。则需要进入细化布线阶段，主要采用迷宫布线算法来处理，如果迷宫布线算法[10] [11] [12] [25]还不能布通，则需要考虑拆线重布算法。细化布线对 Tile 层次的处理顺序与粗化布线时相反，当所有层级都细化处理完毕，则布线流程基本结束。

3.2 基本数据结构

3.2.1 Cellinstance

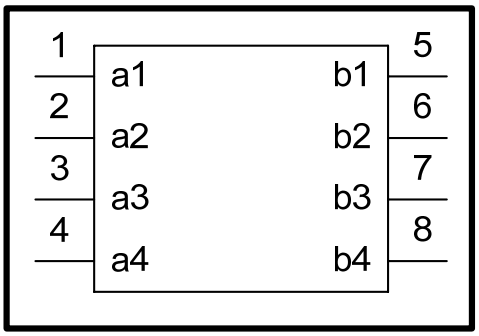

图 1 Cellinstance 实例

int uid:	每一个 Cellinstance 都唯一对应一个 ID；
char* name:	该 Cellinstance 的名字；
double max_delay:	该 Cellinstance 的最大时延；
vector <int> pins_uid:	该 Cellinstance 所有引脚对应的 ID，存储在向量表中

3.2.2 CellinstancePin

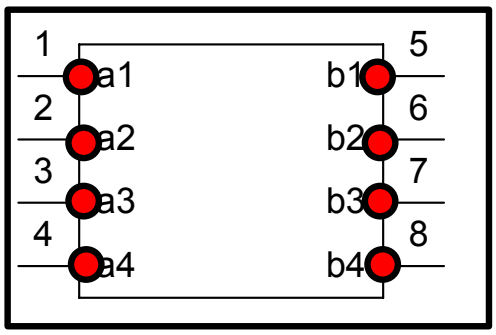

图 2 CellinstancePin 实例

int uid:	每一个 CellinstancePin 都唯一对应一个 ID；
int ci_uid:	该 CellinstancePin 所在的 Cellinstance 对应的 ID；
int net_uid:	该 CellinstancePin 所在的 Net 对应的 ID；
char* name:	该 CellinstancePin 的名字；
lead_node n:	该 CellinstancePin 在图的数据结构中所对应的顶点；
double d_c:	该 CellinstancePin 的 downstream 电容;
double delay:	该 CellinstancePin 的到源点的时延;
int ax,ay;	该 CellinstancePin 的 x,y 坐标;
PinType type:	该 CellinstancePin 的类型，为 IN,OUT 或 UNKNOWN。

3.2.3 Net

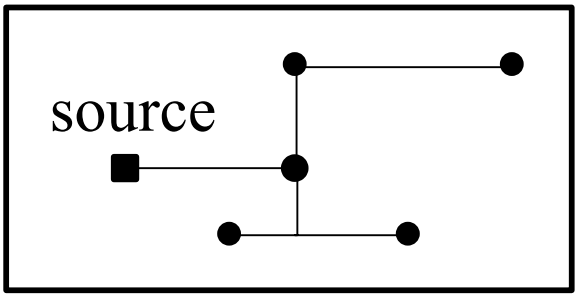

图 3 Net 实例

int uid:	每一个 Net 都唯一对应一个 ID；
char* name:	该 Net 的名字；
int source:	每个 Net 仅有一个源点，source 即为源点的 ID
vector <int> pins_uid:	该 Net 所有引脚对应的 ID，存储在向量表中;
int edge_routed:	该 Net 对应的布线方法
double max_delay:	该 Net 所允许的最大时延，即时延约束；
double delay_lower_bound:	最短路径布线时漏点到源点的最大时延；
double delay_upper_bound:	最小生成树布线时漏点到源点的最大时延。

3.2.4 Edge

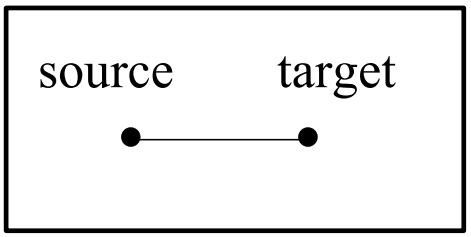

图 4 Edge 实例

int uid:	每一个 Edge 都唯一对应一个 ID；
CellinstancePin *source:	该 Edge 的源点；
CellinstancePin *target:	该 Edge 的漏点；
char* name:	该 Edge 的名字；
lead_node e:	该 CellinstancePin 在图的数据结构中所对应的边；
vector <int> tile_routing:	该 Edge 在总体布线时所经历的 tile 序号，存储在向量表中；
int vias:	该 Edge 在详细布线时所需要的通孔数目；
int dt_grids:	该 Edge 在详细布线时所经历的 grid 的数目；
vector <int*> dt_route_grids:	该 Edge 在详细布线时所经历的 grid 序号指针，存储在向量表中，这样可以方便改变布线；
int distance:	该 Edge 源漏点间的 Manhattan 距离；
bool routed:	该 Edge 是否已经布线成功；
int level:	该 Edge 在多级布线框架中的第几级。

3.2.5 Level

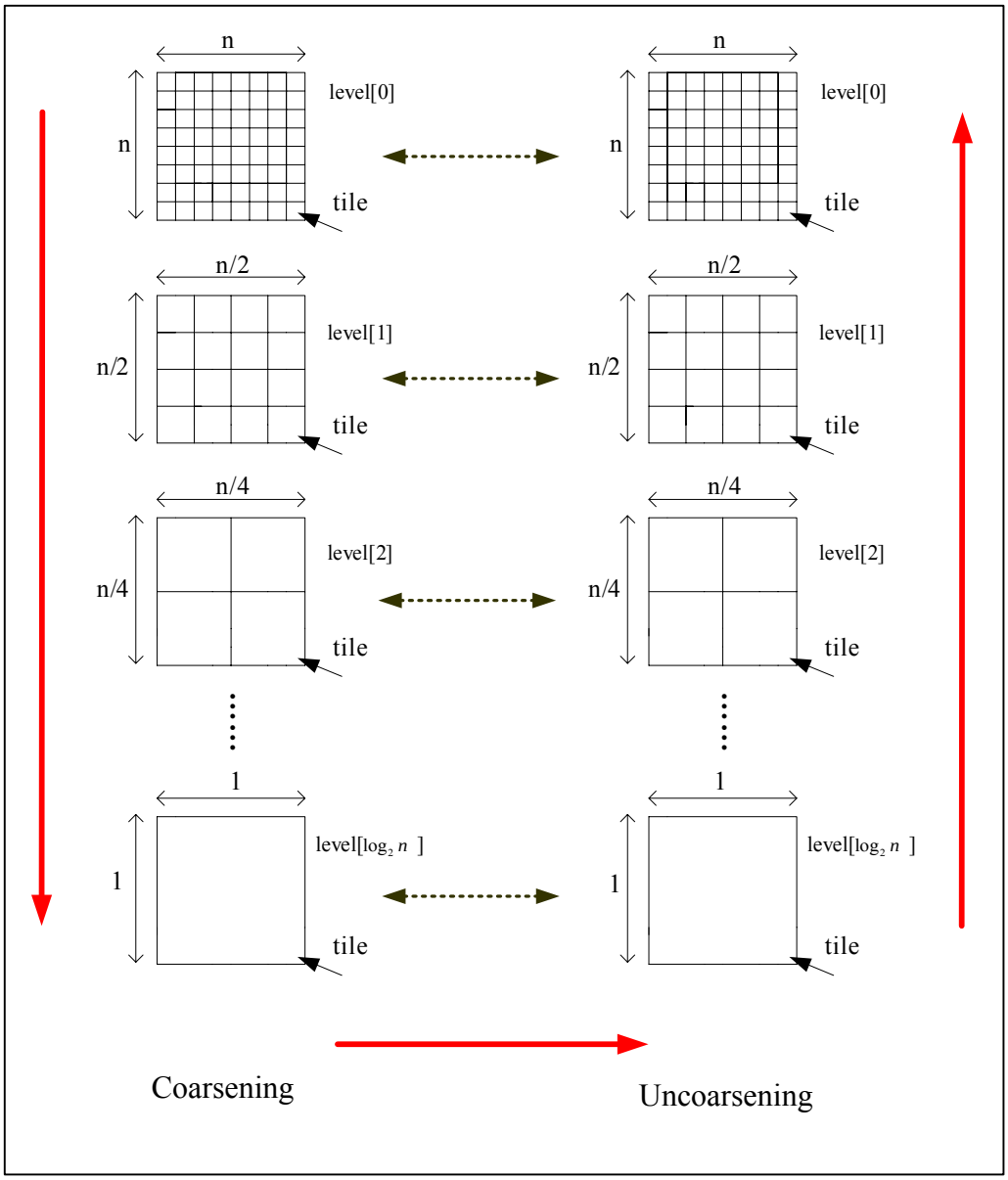

图 5 多级布线框架结构

每一个 Level 至少包含以下定义：

vetors <Tile> tiles:　　该 Level 中所有 Tile，存储在是一个向量中；

int n:　　该 Level 水平方向或者垂直方向上 tile 的数目；

int hcapacity,vcapacity:　　该 Level 中的每个 Tile 的水平布线容量和垂直布

线容量；

int tile_width,tile_height:　　该 Level 中的每个 Tile 的宽度和高度。

3.2.6 Tile

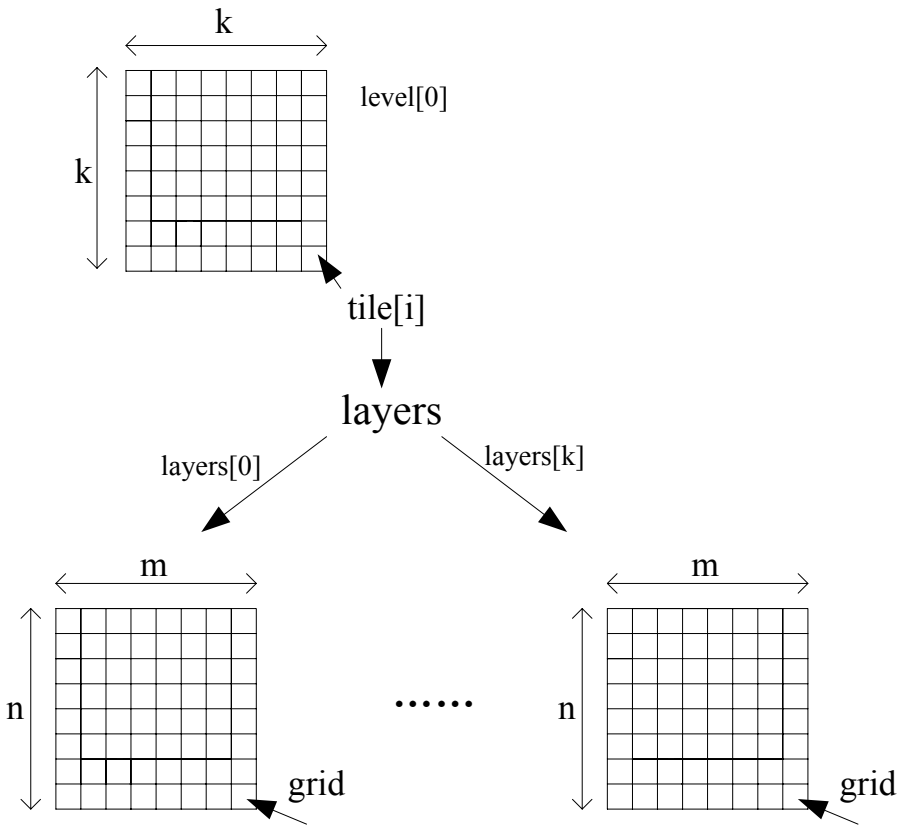

图 6 多级布线中 Tile 的结构

Tile 至少包含以下定义：

int lx,rx,by,ty:　　该 Tile 的左下角和右上角坐标；

int hdemand,vdemand:　　该 Tile 水平或垂直方向上已使用的布线通道数；

DetailTile *dt:　　该 Tile 所指向的详细的 grid 结构；

DetailTile 的定义为：

int grids_used;	该 DetailTile 包含的 grid 数目,m*n 个；
//int overflow;	该 DetailTile 中 grid 溢出的数目；
vector <Layer> layers;	该 DetailTile 中包含的 Layer 向量列表。

Layer 的定义为：

HV_Type type:	该 Layer 的走线方式，HV_H 或 HV_V，即该层为水平或垂直布线方式；
Vector <int> grids:	该 Layer 中包含的 grids 向量列表，每个 grid 对应一个整数编号。

TilePath：存储总体布线路径的结构

Vector <Tile> tiles:	该 TilePath 所经历过的 Tile，存储在向量中；
double cost:	该 TilePath 的布线代价。

3.3 Elmore 时延

3.3.1 Downstream 电容的计算

定义单位长度的电容为 c0，此单位长度即为 grid 的间距

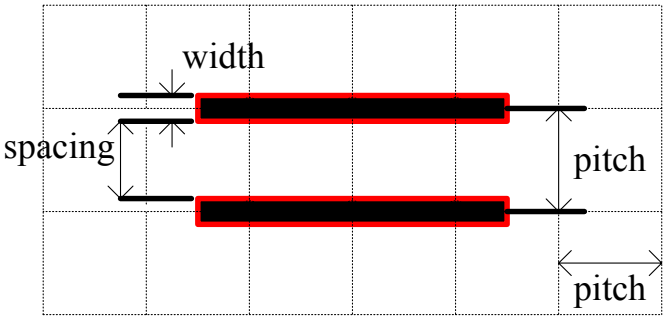

图 7 导线最小间隔及其线宽

设该导线经过了 n 个 grid，且每单位长度的电容为 WireCapacitance，则该导线的电容近似为 C＝WireCapacitance×n×pitch；

同理，设每单位长度的电阻为 WireReistance，则该导线的电阻近似为 R＝WireReistance×n×pitch；

另外对设每一个 pin 脚的电容和电阻分别为 MinGateCapacitance 和 MinGateReistance。

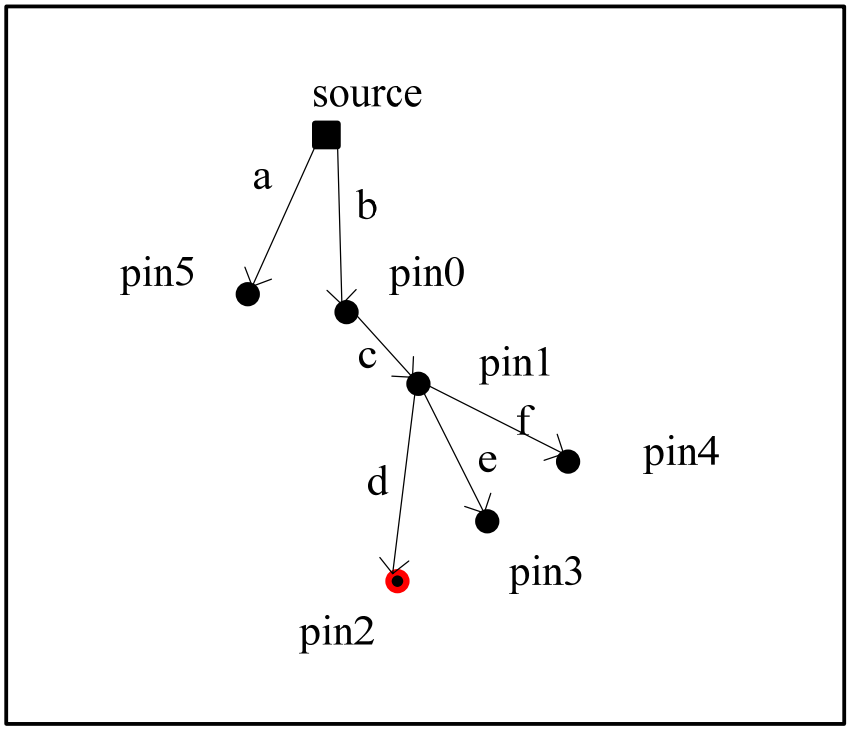

图 8 计算某个 pin 脚电 Downstream 电容实例

上图中 Cpi_id0 引脚的 Downstream 电容为：

down_Cpi_id0＝c 导线电容＋MinGateCapacitance＋down_ Cpi_id1；

上图中 Cpi_id1 引脚的 Downstream 电容为：

down_ Cpi_id1＝d 导线电容＋e 导线电容＋f 导线电容＋3×MinGateCapacitance；

显然求某一 Pin 脚的 Dowmstream 电容的关键是遍历该 Pin 对应节点下面的所有边。

3.3.2 Elmore 时延的计算

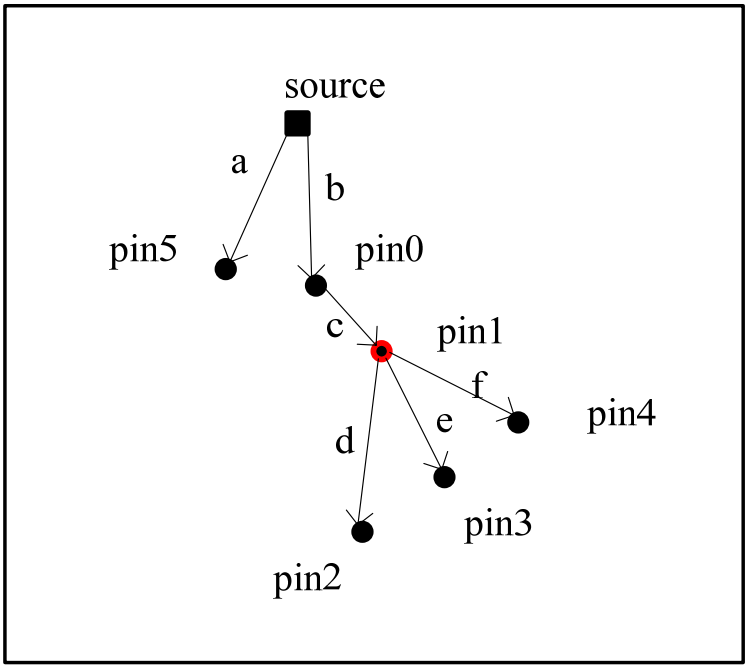

图 9 计算某个 pin 脚 Elmore 时延实例

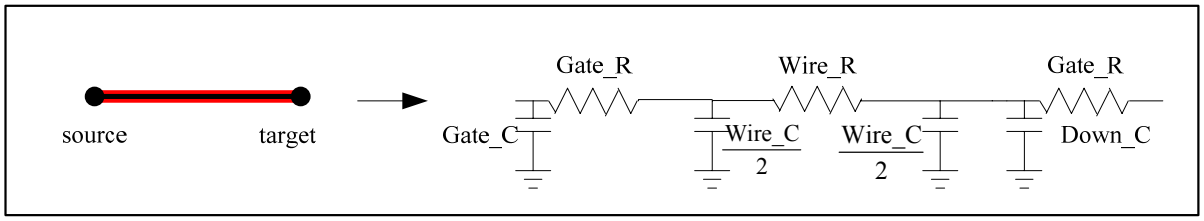

图 10 导线的π形结构

由上图所示，一条边的 source 点到 target 点的 Elmore 时延为：

$$Gate\_R\times(Wire\_C+Down\_C)+Wire\_R\times(\frac{Wire\_C}{2}+Down\_C)$$

为遍历 Cpi_id2 到 source 点的 Elmore 时延，依次获得 Cpi_id2 到 source 点的边 d , c, b, 分别求出这些边的源漏点之间的时延，则总时延即为这些时延之和：

$$Delay\_pin2 = Gate\_R \times (d\_C + Down\_pin2\_C) + Wire\_d\_R \times (\frac{d\_C}{2} + Down\_pin2\_C)$$
$$+ Gate\_R \times (c\_C + Down\_pin1\_C) + Wire\_d\_R \times (\frac{c\_C}{2} + Down\_pin1\_C)$$
$$+ Gate\_R \times (b\_C + Down\_pin0\_C) + Wire\_d\_R \times (\frac{b\_C}{2} + Down\_pin0\_C)$$

3.4 程序结构

3.4.1 多级布线框架介绍

(1) 输入
(2) 建立基本数据结构
(3) 分解所有的线网成为二端点的连线，MDST 算法
(4) 线网分析
(4.1) 拥挤度分析和时延分析
(5) 粗化布线
(5.1) 局部线网搜索
(5.2) 拥挤度驱动的总体布线
(5.1) 时延驱动的详细布线，最小增量的时延修正
(6) 细化布线
(6.1) 对未布通的线网进行绕线布线
(6.1) 对绕线布线尚未布通的线网进行拆线重布

详细的程序结构描述如下：

(1) 输入：

首先，为了研究的改正性能的对比，我们读入的测试例子都是由 UCLA CAD 实验室提供的标准单元的电路实例，这些实例都是已经完成了布局后，即所以的标准单元的摆放坐标位置，方向都已经确定。读入例子的接口程序也是由 UCLA CAD 实验室提供的 Layoutdb.h 和 Layoutdb.o 库，在读入电路信息时，会建立好电路的基本信

息库，这个库的内容包括标准单元，线网，标准单元的引脚的坐标信息和互连信息，以及电路的布线层数，导线的间隔、线宽等信息。根据库中的信息，可以初始化算法的基本的数据结构。

(2) 数据结构的初始化：

根据电路基本信息库中的内容，我们可以逐一初始化数据结构。并根据电路的信息，将设计划分为 K 级，并由此决定了每一级中块（Tile）的数量为 n×n。再设水平方向和垂直方向的网格（Grid）数目分别为 GridX 和 GridY。假设多级布为第 K 级时，则第 K 级的块（Tile）数目为 $2^k \times 2^k$，所以可以计算出在第 K 级中平均每个块（Tile）中包含网格（Grid）数量为 $\frac{GridX}{2^k} \times \frac{GridX}{2^k}$，则 K 从 0 开始递增，直到满足下列表达式便停止递增，此时的 K 值即为总共的布线级数。

$2^{k+1} > \frac{GridX}{2^{k+1}}$ 或者 $2^{k+1} > \frac{GridY}{2^{k+1}}$

也即是说在第 k+1 级时，块（Tile）内的 x 或 y 轴上的网格（Grid）数会小于此级中的 x 或 y 的块（Tile）数，这样每一个块（Tile）中不足一个网格（Grid）了，继续划分下去是没有意义的。

下面即是计算层级数的伪码：

```
Calcu_levels( )
/* Input: 电路信息 */
/* Output: 划分级数 */
/* Init: k=0 */
{
      n=1;
      while (1) {
      xgrids = (int)ceil(width / n); /* xgrids为第k级的一个Tile中x轴上的grid数 */
      ygrids = (int)ceil(height / n); /* ygrids为第k级的一个Tile中y轴上的grid数 */
      if ( xgrids < n || ygrids < n ) { /* 当x或y的grid数小于n时结束 */
            break;
      }
      else {
            n*=2; /* 将分解系数n乘22 */
            k++;
                  }
            }
      k--;
      return k; /*返回K值 */
}
```

图 11 计算层级的伪码

计算 k 值后，第一步会建立一个 $2^k \times 2^k$ 的块（Tile），这些块（Tile）即将是我们做总体布线时的布线图。

(3) 分解所有的线网成为二端点的连线，在作时序驱动的多级布线时，我们要将线网划分为二端点的连线互连，将多端点线网划分为多个二端点线网的过程需要使用 MST（Minimal Spanning Tree）算法。再使用 MDST 算法，或者最小增量修正算法，或者新的平衡树时延修正算法来提高时延性能，在此基础上可以对该树进行总体布线和详细布线，总体布线时需要求出其 Steiner 树[17] [18] [19]。

(4) 参考图 5 所示的流程，对局部的 Tile 内的线网分别做总体布线和详细布线，此时的布线方式仅限于一些简单的布线方法，例如水平线、垂直线、L 型布线、Z 型布线等。当这一级的 Tile 全部处理完，即跳入下一级的布线，这样重复下去，直到 Tile 不能再合并。然后再做上述过程的逆过程，逆过程中主要处理时序无法满足、或

者简单布线方法无法布通的线网。需要采用迷宫布线算法，如果迷宫布线算法还不能布通，则需要考虑拆线重布算法，这是一个非常复杂的部分，将不在本文的讨论之中。当逆过程的细化布线完成只好，布线流程基本完整。

3.4.2 输入文件格式介绍

输入文件为.gdf 格式，详细可以参阅[32]。

Layout Database User Manual (Revision： 1.12) UCLA Computer Science Dept., Los Angeles, CA 90095

```
Cell name
   Port name type x y layer
      CellInstance name cellRef x y orient
      Path name
         PathObj type=Segment x y x1 y1 width layer
         PathObj type=Via x y width layer
         PathObj type=String string
      Net name group
         NetRef type=PortRef namexyportRef
         NetRef type=PathRef namexypathRef
         NetRef type=InstPortRef namexyportRef instRef
      Text name string
Tech name
      TechObj name type width
LayoutRules name
      LayoutRule type obj obj1 minValue maxValue typValue
```

图 12 Cellinstance 实例[32]

gdf 文件中 Cell 的描述结构

其他一些公共参数：

number_of_layers: 布线的层数;

wire_spacings: 导线间的最小距离，每层可以不同，所以它的值是一个列表，每

层对应一个值；

via_spacings: 通孔间的最小距离，每层可以不同，所以它的值是一个列表，每层对应一个值；

wire_widths: 导线的宽度，每层可以不同，所以它的值是一个列表，每层对应一个值；

via_widths: 通孔的宽度，每层可以不同，所以它的值是一个列表，每层对应一个值；

vertical_wire_costs: 垂直方向布线的代价，每层可以不同，所以它的值是一个列每层对应一个值；

horizontal_wire_costs: 水平方向布线的代价，每层可以不同，所以它的值是一个列每层对应一个值；

via_costs: 通孔的代价，每层可以不同，所以它的值是一个列每层对应一个值。

一个线网的例子如下：

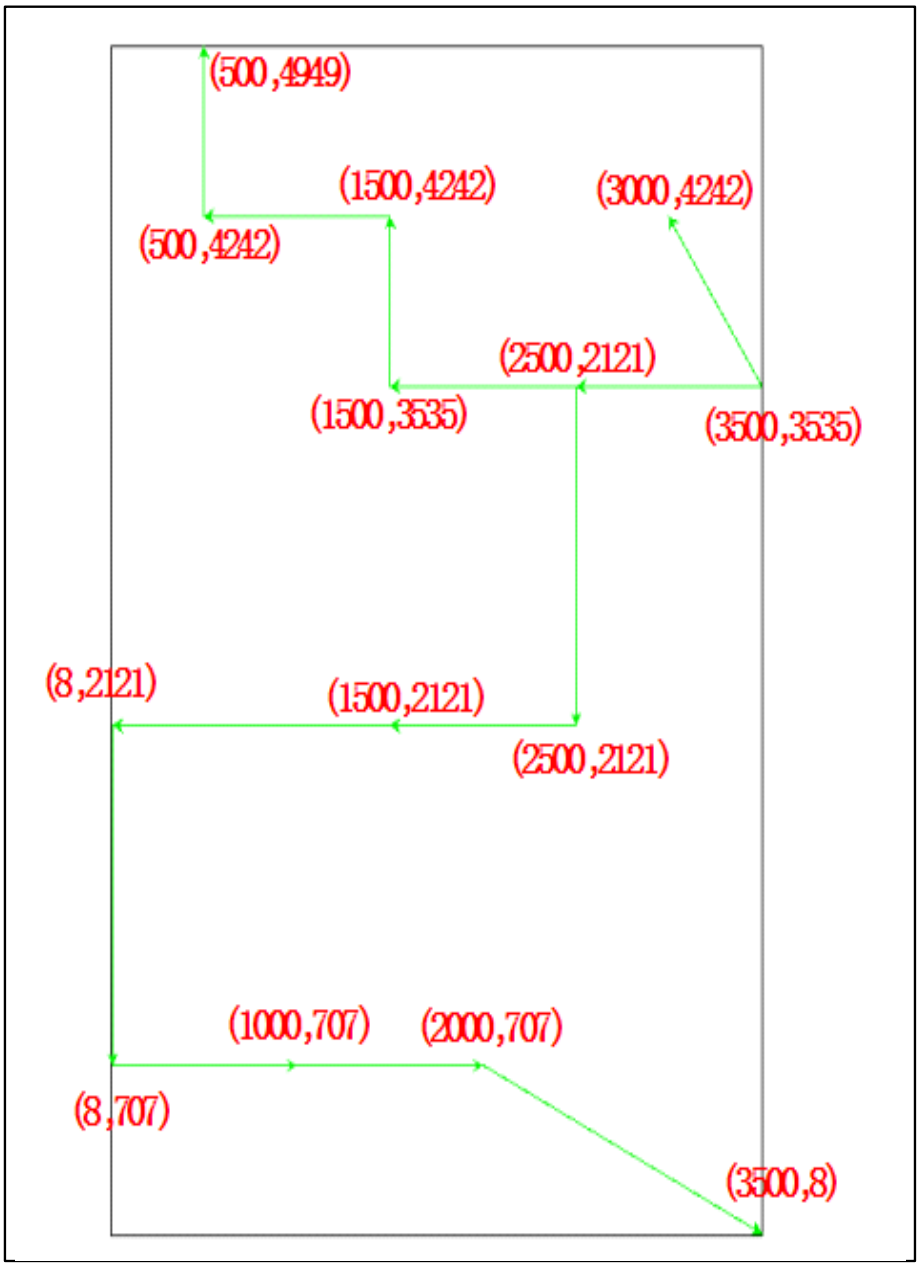

图 13 一个线网的实例

该线网所对应的 gdf 文件如下所示：

```
(gdif
 (gdifVersion 1 0 1)
 (comment Generated by tw2gdif)
 (cell:top
                (text:number_of_layers "3")
                (text:wire_widths   "0.6 0.6 0.6")
                (text:via_widths    "0.6 0.6 0.6")
                (text:wire_spacings "1.2 1.2 1.2")
                (text:via_spacings  "1.2 1.2 1.2")
                (text:vertical_wire_costs    "2 1 2 1 2 1")
                (text:horizontal_wire_costs  "1 2 1 2 1 2")
                (text:via_costs "10 10 10 10 10 10")

                (path:BBOX
                        (new)(layer LEV)(width 0)(pt -5 52)(pt -5 4956)
                        (new)(layer LEV)(width 0)(pt -5 4956)(pt 4898 4956)
                        (new)(layer LEV)(width 0)(pt 4898 4956)(pt 4898 52)
                        (new)(layer LEV)(width 0)(pt 4898 52)(pt -5 52)
                )

  (port:pad_26_A_0_A_0 (pt 8 707))
  (port:INS241_a (pt 8 2121))
  (port:INS225_a (pt 500 4242))
  (port:INS209_a (pt 500 4949))
  (port:INS193_a (pt 1000 707))
  (port:INS177_a (pt 1500 2121))
  (port:INS161_a (pt 1500 3535))
  (port:INS145_a (pt 1500 4242))
  (port:INS129_a (pt 2000 707))
  (port:INS113_a (pt 2500 2121))
  (port:INS97_a (pt 2500 3535))
  (port:INS81_a (pt 3000 4242))
  (port:INS65_a (pt 3500 8))
  (port:INS49_a (pt 3500 3535))
  (net:A_0
   (portRef pad_26_A_0_A_0)
   (portRef INS241_a)
   (portRef INS225_a)
   (portRef INS209_a)
   (portRef INS193_a)
   (portRef INS177_a)
   (portRef INS161_a)
   (portRef INS145_a)
   (portRef INS129_a)
   (portRef INS113_a)
   (portRef INS97_a)
   (portRef INS81_a)
   (portRef INS65_a)
   (portRef INS49_a)
  )
 )
)
```

图 14 一个线网的实例对应的 gdf 文件

3.5 本章小结

本章详细介绍了多级自动布线框架，主要包括如何读入数据并初始化数据结构，如何进行粗化布线和细化布线；另外对多级自动布线框架中引用的基本数据结构做了一一介绍。最后还分析了线网Elmore时延的详细模型。

第四章 最短距离生成树和平衡树时延修正方法

由于 IC 设计中互连延时变得越来越严重，决定了 IC 的性能，所以为了是设计中的时延符合设计的时延要求越来越重要，基于时延驱动的布线研究也越来越受到重视。我们都知道最短路径布线 SPT（Shortest Path Tree）可以获得最佳的布线时延，即所以漏点到源点的距离都是最短的，但是这使得总线长增长的很大，布线所需的芯片面积大幅增加，这将直接导致布线的可布通性或者导致芯片成本提高了。

另一方面，最小生成树 MST（Minimal Spanning Tree）可以使的每组待布线的线网总长度达到最短，但是由于 MST 算法没有考虑源点和漏点之间的关系，这将使得一些关键路径上的负载很大，从而导致时延不能符合设计要求。

显然，既能使得总线长最短，又能获得最好时延性能的线网拓扑结构是非常难以求解的，或者可能是不存在的。

台湾大学的多级布线框架和时延修正算法在某些成都上可以修正一些关键路径上时延问题，在研究他们的算法后，我们找到了一种更有效的方法来处理关键路径上的时延问题。下面将详细介绍最最短距离生成树和平衡树的时延修正。

4.1 最小生成树算法的基本原理

因为具有 n 个顶点的无向网络 G 的每个生成树刚好具有 n-1 条边，所以问题是用某种方法选择 n-1 条边使它们形成 G 的最小生成树。至少可以采用三种不同的贪婪策略来选择这 n-1 条边。这三种求解最小生成树的贪婪算法策略是：Kruskal 算法，Prim 算法和 Sollin 算法，下面分别一一介绍。

1. Kruskal 算法

Kruskal 算法每次选择 n-1 条边，所使用的贪婪准则是：从剩下的边中选择一条不会产生环路的具有最小代价的边加入已选择的边的集合中。注意到所选取的边若产生环路则不可能形成一棵生成树。Kruskal 算法分 e 步，其中 e 是网络中边的数目。按代价递增的顺序来考虑这 e 条边，每次考虑一条边。当考虑某条边时，若将其加入到已选边的集合中会出现环路，则将其抛弃，否则，将它选入。

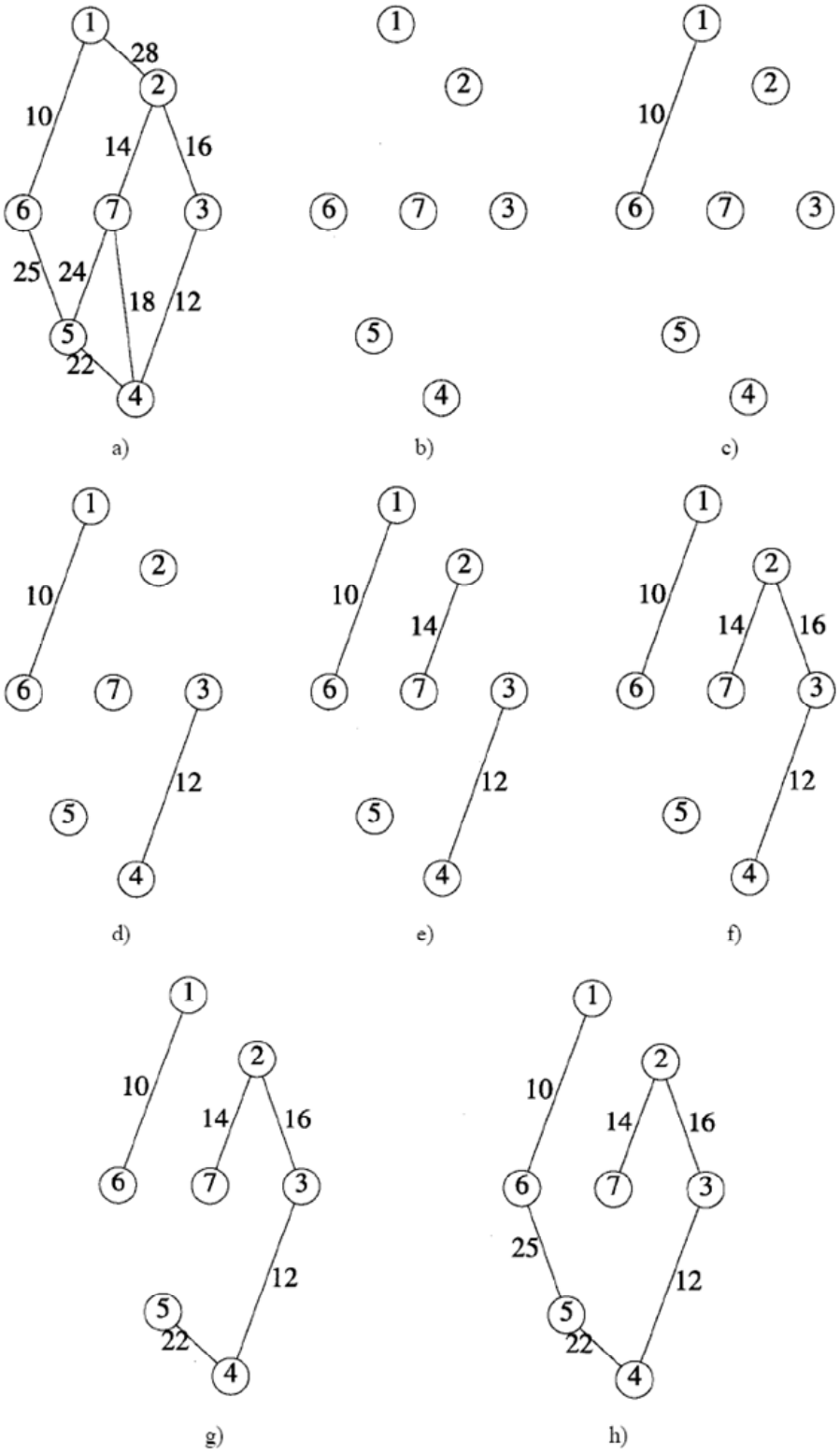

图 15 构造最生成树[33]

考察上图 15a 中的网络。初始时没有任何边被选择。图 15b 显示了各节点的当前状态。边（1, 6）是最先选入的边，它被加入到欲构建的生成树中，得到图 15c。下一步选择边（3，4）并将其加入树中（如图 15d 所示）。然后考虑边(2，7)，将它加入树中并不会产生环路，于是便得到图 15e。下一步考虑边（2，3）并将其加入树中（如图 15 f 所示）。在其余还未考虑的边中，（7，4）具有最小代价，因此先考虑它，将它加入正在创建的树中会产生环路，所以将其丢弃。此后将边（5，4）加入树中，得到的树如图 15g 所示。下一步考虑边（7，5），由于会产生环路，将其丢弃。最后考虑边（6，5）并将其加入树中，产生了一棵生成树，其代价为 99。图 16 给出了 Kruskal 算法的伪代码。

```
//在一个具有n 个顶点的网络中找到一棵最小生成树
令T为所选边的集合，初始化T=Φ
令E 为网络中边的集合
while (E≠Φ)&&(|T|≠n-1) {
    令(u,v)为E中代价最小的边
    E=E-{(u,v)} //从E中删除边
    if( (u,v)加入T中不会产生环路）将（u,v）加入T
}
if(|T|==n-1) T是最小生成树
else 网络不是互连的，不能找到生成树
```

图 16 Kruskal 算法伪码[33]

2. Prim 算法

与 Kruskal 算法类似，Prim 算法通过每次选择多条边来创建最小生成树。选择下一条边的贪婪准则是：从剩余的边中，选择一条代价最小的边，并且它的加入应使所有入选的边仍是一棵树。最终，在所有步骤中选择的边形成一棵树。

相反，在 Kruskal 算法中所有入选的边集合最终形成一个森林。

Prim 算法从具有一个单一顶点的树 T 开始，这个顶点可以是原图中任意一个顶点。然后往 T 中加入一条代价最小的边 (u,v) 使 $T\cup\{(u,v)\}$ 仍是一棵树，这种加边的步骤反复循环直到 T 中包含 n-1 条边。注意对于边 (u,v)，u、v 中正好有一个顶点位于 T 中。Prim 算法的伪代码如图 17 所示。在伪代码中也包含了所输入的图不是连通图的可能，在这种情况下没有生成树。图 18 显示了对图使用 Prim 算法的过程。

```
/ /假设网络中至少具有一个顶点
设T为所选择的边的集合，初始化T=Φ
设TV为已在树中的顶点的集合，置TV= {1}
令E为网络中边的集合
while (E< >Φ) & & (|T| < > n-1) {
   令（u , v）为最小代价边，其中u  TV, v  TV
   if（没有这种边） break
   E=E-{(u,v)} / /从E中删除此边
   在T中加入边（ u , v）
}
if (| T | = =n- 1 ) T是一棵最小生成树
else 网络是不连通的，没有最小生成树
```

图 17 Prim 最小生成树算法伪码[33]

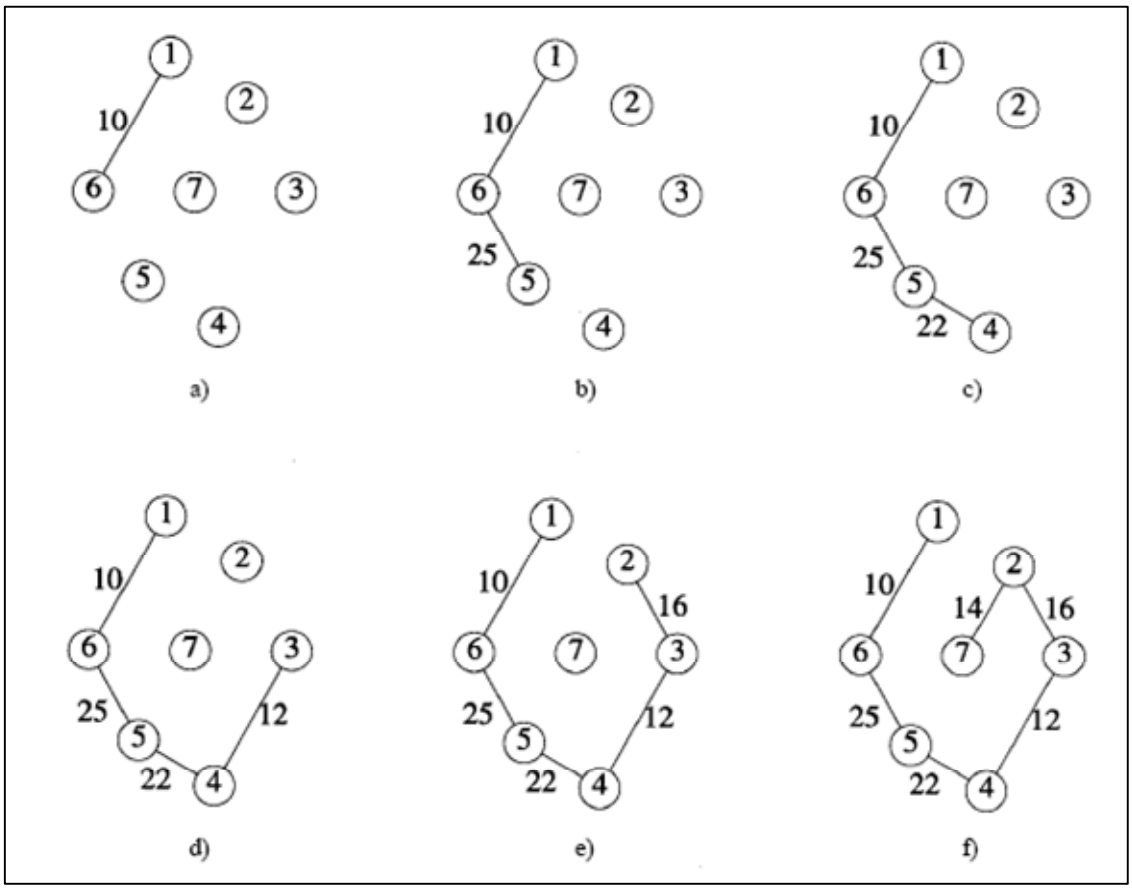

图 18 Prim 算法步骤[33]

如果根据每个不在 TV 中的顶点 v 选择一个顶点 $near(v)$，使得 $near(v) \in TV$ 且 $\cos t\ (v,\ near\ (v))$ 的值是所有这样的 $near\ (v)$ 节点中最小的，则实现 Prim 算法的时间复杂性为 $O(n^2)$。下一条添加到 T 中的边是这样的边：其 $\cos t\ (v,\ near\ (v))$ 最小，且 $v \notin TV$。

3. Sollin 算法

Sollin 算法每步选择若干条边。在每步开始时，所选择的边及图中的 n 个顶点形成一个生成树的森林。在每一步中为森林中的每棵树选择一条边，这条边刚好有一个顶点在树中且边的代价最小。将所选择的边加入要创建的生成树中。注意一个森林中的两棵树可选择同一条边，因此必须多次复制同一条边。当有多条边具有相同的代价时，两棵树可选择与它们相连的不同的边，在这种情况下，必须丢弃其中的一条边。开始时，所选择的边的集合为空。若某一步结束时仅剩下一棵树或没有剩余的边可供选择时算法终止。

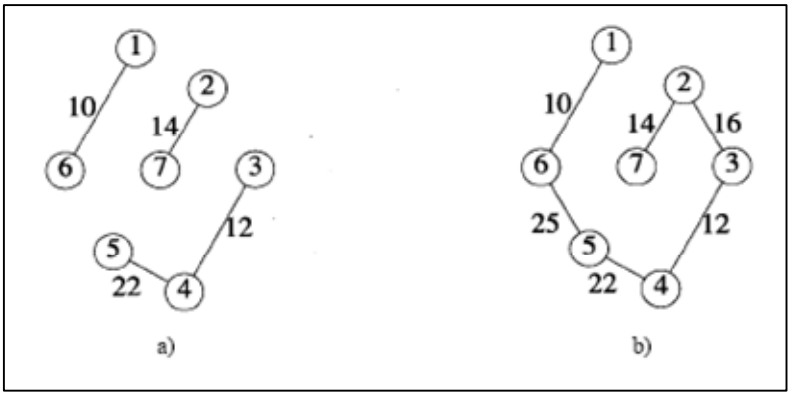

图 19 Solin 的算法步骤[33]

图 19 给出了初始状态为图 15a 时，使用 Sollin 算法的步骤。初始入选边数为 0 时的情形如图 15a 时，森林中的每棵树均是单个顶点。顶点 1,2,…,7 所选择的边分别是(1,6)，(2,7)，(3,4)，(4,3)，(5,4)，(6,1)，(7,2)，其中不同的边是(1,6)，(2,7)，(3,4)和(5,4)，将这些边加入入选边的集合后所得到的结果如图 19a 所示。下一步具有顶点集{1,6}的树选择边(6,5)，剩下的两棵树选择边(2,3)，加入这两条边后已形成一棵生成树，构建好的生成树见图 19b。

4.2 最短距离生成树

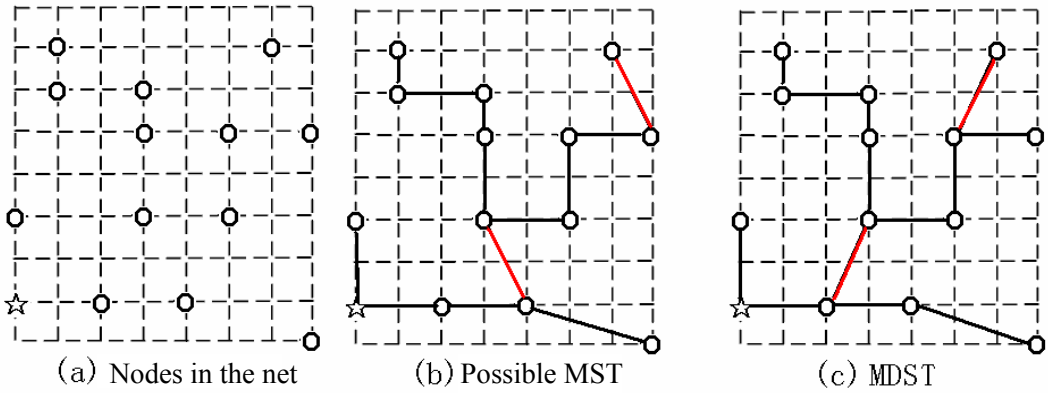

图 20 最短距离生成树的实例

由上图所示，当我门对一组线网作 MST 展开时，可能存在多个解，并且这些解的布线代价是一致的，图 20-b 和图 20-c 均为图 20-a 的 MST，且二者总的布线长度是相等的，但是图 20-c 更加符合我们的设计需求，这是因为图 20-c 中的红色边的连接使得漏点到源点的距离会更短一些，这样显然图 20-c 比图 20-b 中的关键路径上的时延性能要好。算法描述如下：

算法：最小距离生成树算法
输入：图G＝（V,E）以及源点S
输出：最小距离生成树－MDST
Begin
1. 将所有长度相同的边分为一类，并且按照长度从小到大为类排序 $(e_1 \cdots e_N)$;
2. 对每一类中的所有边
   Dis＝该边的中点到源点S的距离；
   将该类中的边按Dis从小到大排序；
3. while（存在超过2颗不同的最小生成树）do
   for $(e_1 \cdots e_N)$
      遍历所有ei中的边，该边的顶点记为m，n
      $f(T_m != T_n)$
         $T_m = T_m \cup T_n$
End

图 21 最小距离生成树算法[3]

4.3 回溯修正算法和最小增量修正算法

为了处理深亚微米的 IC 设计，互连延时已经越来越重要，通常最小生成树 MST（Minimal Spanning Tree）算法虽然能使得总共的布线长度最短，且此时拥挤控制是最方便的，因为布线空间相对很大；但是 MST 算法得到的拓扑结构可能导致很长的关键路径（Critical Path），这将容易破坏时延约束而影响电路的性能。另一种方法是使用最短路径树 SPT（Shortest Path Tree）算法，这种算法能使得时延性能到最佳状况，时延一般都能满足，但是总线长将显著增加，从而使得布线拥挤度显著增加，并将直接影响到可布通性。

台湾大学的学者们提出了一种回溯修正算法(Recalling Modification)[1]，以此来修正 MST 树中的时延不满足约束的情况，这相当于折中了拥挤和时延性能。基于多级布线框架中采用 Recalling Modification 算法的伪码描述如下图所示：

```
算法: 性能驱动的多级自动布线算法(G, N, C)
输入: G - 布局后的网表;
      N - 多端点线网集合;
      C - 时序约束.
输出：符合时序约束的N和G
begin
1   使用MST算法将多端点线网划分为二端点线网;
2   //粗化布线
3   对所以的粗化布线层级
4     选取局部线网 n;
5     if n 不满足时序约束,
        采用recalling modification算法修正;
6       if n属于改布线层级
7         Global_Pattern_Routing( );
8         Detailed Routing( );
9    //细化布线
10  对所以的细化布线层级
11    Timing_Analysis_on_All_Nets( );
12    选取不满足时序约束的局部线网或者粗化布线阶段没有成功
        布通的线网
13    if n 不满足时序约束,
        采用recalling modification算法修正;
14    Global_Maze_Routing( );
15    Detailed_Routing( );
16  Output_Result();
end
```

图 22 使用 Recalling Modification 算法的时延驱动的多级布线流程[1]

一般地，我们处理多端点线网时，首先使用 MST 算法将所有的线网构建成 MST 的拓扑结构，然而我们考虑到时延的 MST 应该是一颗有向树，而一般的 MST 算法生成的都是无向的边，即没有考虑源点(Source)与漏点(Target)的关系，则所得的拓扑结构是一些无向边集合组成了一颗无向树。在作分级布线阶段，每一个块(Tile)内进行局部布线之前都需要对线网进行一次时延分析，时延分析均是基于 Elmore 延时模型。如果线网中某一节点不满足时延约束，则调用 Recalling Modification 算法来修正时延约束。

Recalling Modification 算法思想是：如果线网中的某一目标节点不满足时延约束，我们删除这一局部连接，即一条边，并从该边的漏节点(Target node)向源节点(Source node)回溯，即找到该点父节点的父节点作为该目标节点的父节点。计算新的连接是否符合时延需求，如果不符合，再重复下去直到找到一个合适的节点作为该目标节点的父节点，这样一定存在一个新的连接可以满足时延约束的。虽然这种方法可能导致总的布线长度有所增加，且分支导线的电容也会有所增加，但是该节点到源点的时延将会显著降低。一个详细的实例如下图所示。

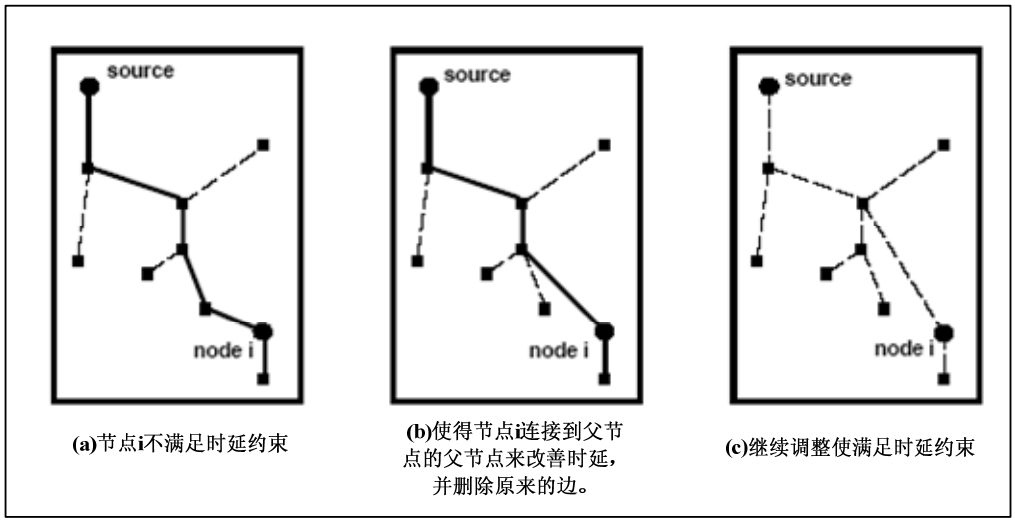

图 23 Recalling Modification 实例[1]

然而上面的算法存在一个不容忽视的问题，如果上图 23-c 中的时延如果仍然不能满足需求，则需要将节点 i 和 source 节点直接相连了，如下图 24-a 所示，看起来似乎可行，时延性能可以满足，但是这样布线将使得总线长增长非常显著，如果我们采用下图 24-b 的拓扑结构，则时延约束也可能满足，且导致的总线长增加的也不是很显著。下面所述算法的妙处在于采用了最小增量的时延修正算法。

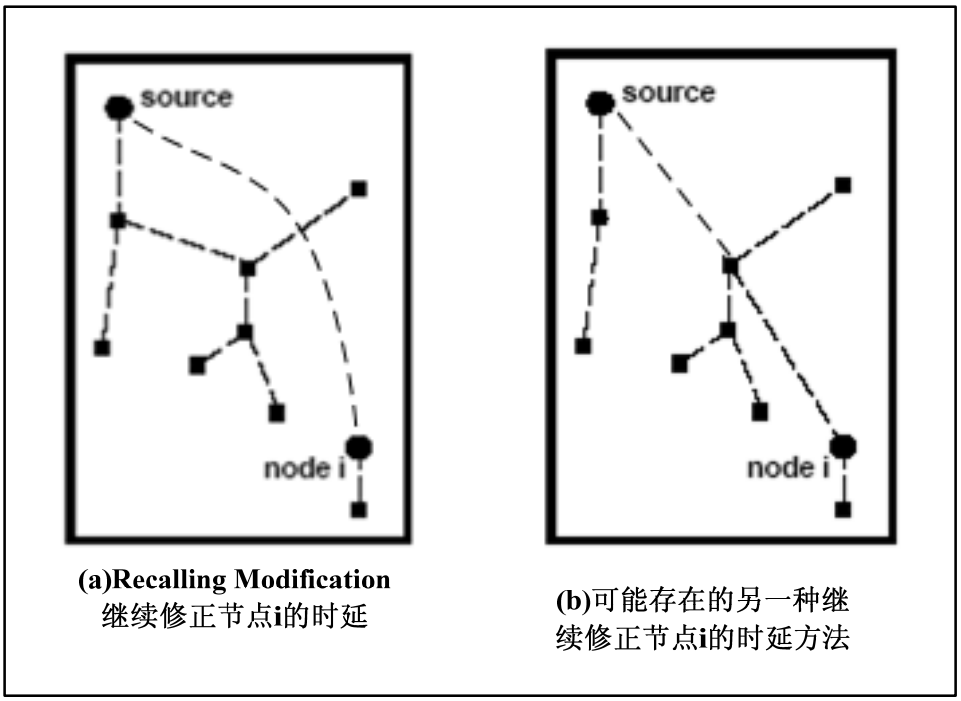

图 24 Recalling Modification 与最小增量的时延修正算法[1]

最小增量的时延修正算法是：如果线网中的某一目标节点不满足时延约束，我们将不直接删除这连接，将从该目标节点向源节点回溯，找到父节点的父节点作为该目标节点的父节点，计算新的连接对总的布线长度的增量。同理再分析父节点的情况，计算改变父节点的父节点作为其父节点时对布线长度的增量。如此重复下去，一直到根节点，找出该条路径上所有可能的修正方法，对改正后能符合时序要求的改正方法按照其对总的布线长度的增量的大小排序，取增量最小的，这样就可以实现总长度最小的增量时延修正结果，且性能将会明显优于简单的回溯修正算法。

4.4 平衡树时延修正算法

比较回溯法和最小增量修正算法，我们发现，他们修正时仅仅考虑了修改该节点到源点的路径的互连关系，而没有考虑到与其他分支上的重新组合效果。基于这一原因，我们提出了平衡树的修正算法思想是，即修正时不局限于改点到源点路径上的修

正，还要考虑其他分支上的互连关系，并且我们将各个节点的负载情况（导致时延的关键因素）和到源点的距离结合起来考虑，这将使得的调整后的结构显示各个分支上的负载情况趋向于一种平衡，我们将这种算法称为平衡树的时延修正算法。

当我们通过 MST 算法分析一个线网时，结果是获得了一组边的列表，这些边将组成一颗 MST 树，根据 MST 的边以及这个线网的源点，我们可以索引出一颗以源点为根节点的树。但是由于 MST 算法中的边均为无向边，所以漏点到源点的路径不是最优的路径。我们从根结点所在的边开始分析，设置所在的

遍历这颗新生成的 MST 树，分别记录每一个 Pin 脚的 ID、将所有的边存在一个边的向量中，记录每条边的源漏 Pin 的 ID，该边的源节点离改线网源节点的深度 Deep，该边的 Manhattan 距离，以及该边漏点处的负载大小 Load，Load 的值由该节点的后继节点数目和后继的各边 Manhattan 总长度组成，根据不同的工艺情况，节点的负载和边的负载对总的负载权重是不一样的，需要适当地调整，在我们的程序中，我们设置一个节点的权重为 2000，边的权重为 1，且边长度单位采用的是微米。最后将所有的边按照 Deep 的大小从小到大排序，各条边上记录的详细信息如下图 25 中的示例所示。

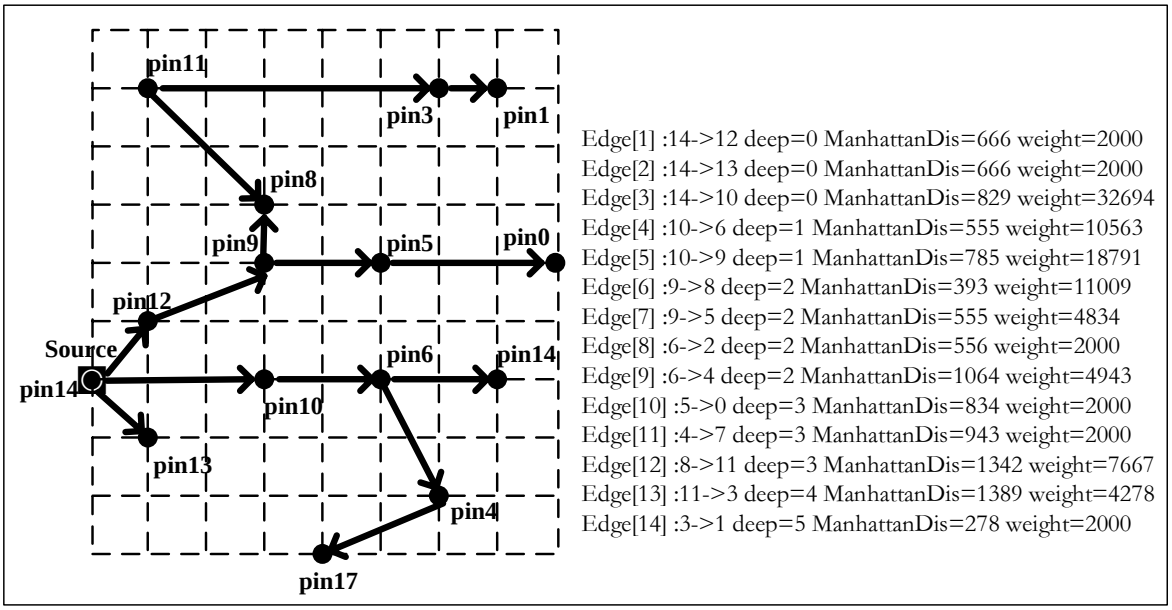

图 25 平衡树时延修正实例

.详细的平衡树时延修正流程如下图 26 所示。当遍历的两条边在同一路径上时我们采用类似于 Recall Modification 和最小增量的修正算法；当不在同一条路径上时，两条边的负载之比对大于改正时对总布线长度增加的一个代价时，可以进行修正，即将下游的边的源点修改为上游的那条的漏点。K1 和 K2 的值可以估算出来，也可以通过测试不同的实例性能的改正情况来得到最优的 K1 和 K2 值，我们在第五章的实

验中的 K1＝3，K2＝0.02。

```
算法：平衡树修正算法
输入：最小距离生成树(MST)边集
输出：平衡修正的最小距离生成树－BMDST
Begin
1. 遍历所有边Edge[1…N]
   记录所有边的信息，包括源点漏点，离源点深度，边长，以及
     该边漏点的负载情况
   按照离源点的深度从小到大排序
2. For Edge[i]=Edge[1… N-1]
          PinA=Edge[i]的漏点；
          For Edge[j]=Edge[i+1…N]
             PinB=Edge[j]的漏点；
             Length1 = PinA到源点的路径上Manhattan距离之和；
             Length2 = PinB到源点的路径上Manhattan距离之和；
             Distance＝PinA 到PinB的Manhattan距离；
             if PinA和PinB不在同一分支的路径上
               if Edges[i].Load/Edges[j].Load > K1*((Length1 + Distance)/Length2)
                  改变Edge[j]的源点为PinA；
             else //PinA和PinB在同一分支的路径上
                Length=PinA到PinB路径上的Manhattan距离只和
                If (Distance < Length) && (新线网的线长增量 <K2*线网的总Manhattan距离)
                    改变Edge[j]的源点为PinA；
3. End
```

图 26 平衡树时延修正算法流程

平衡树的修正算法思想同最小增量的时延修正算法最大的不同点在于修正的不仅仅是该条路径上的节点连接关系，还考虑了所有的父节点的兄弟节点，这种更广泛的平衡时延方法使得时延性能有了更大改善空间。参考 3.3.2 节中对 Elmore 时延模型的讨论，这种改善的原理非常简明，参照图 27（a）所示，A 部分的电容对 B 点的时延起了很大的作用，如果我们将 A 部分拉到源点的另外一个分支上，这将大大减少 A 部分电容的 B 处时延的影响，从而减少源点到 B 的时延，这样便达到了修正时延的目的。

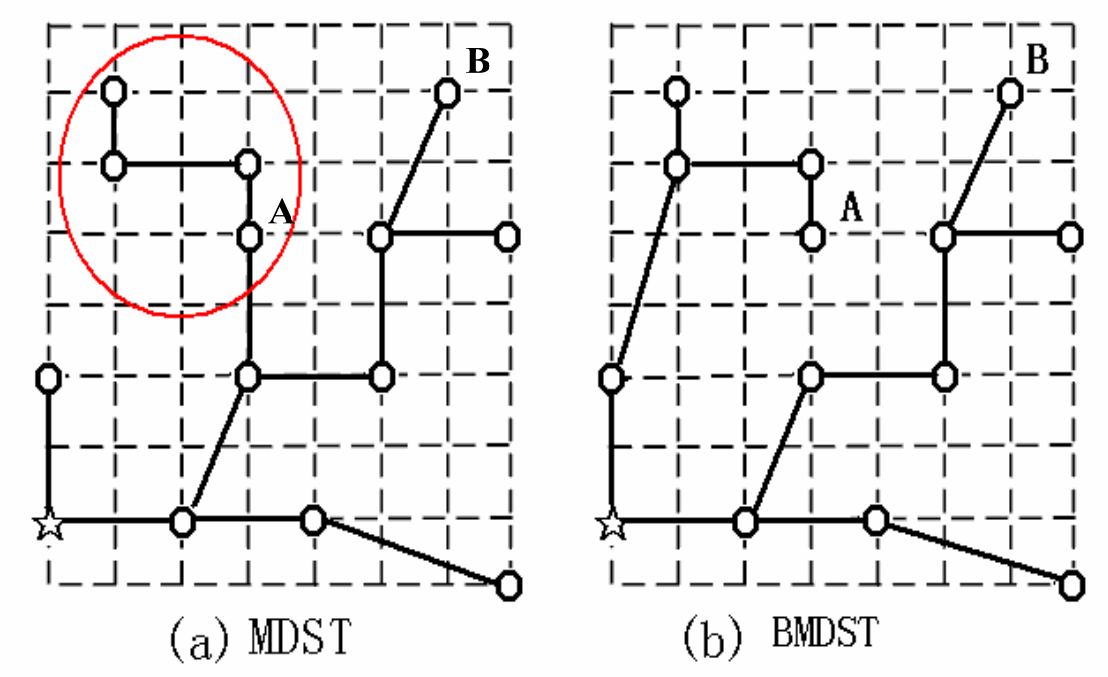

图 27 MDST 算法与 BMST 算法

4.5 本章小结

本章首先详细介绍了最小生成树的三种经典算法，Kruskal 算法，Prim 算法和 Sollin 算法。他们的算法复杂度和时间复杂度有各自的优缺点，在我们的程序中可以任意选取一种。然后针对最小生成树是基于无向图的，我们必须对生成的最小生成树做修正，因此产生了最短距离生成树算法，回溯修正算法以及最小增量的修正算法，在分析这些算法优缺点的基础上，我们找到了一种新的算法，即平衡树的时延修正算法，该算法不仅吸取了前面 3 中算法中的优点，而且还考虑了节点和父亲节点的兄弟节点之间的联系，这样使得布线树的时延性能得到进一步优化。

第五章 实验数据及实验结果

5.1 实验数据

测试实例来源于 UCLA CAD 实验室提供测试例子，针对单个线网的时延性能改正采用了另外 2 个测试的实例，其 gdf 文件参见附录 2。

测试平台和程序语言：

本论文的程序是采用 C++语言实现的，并大量使用 STL，既方便了程序设计，也提高了程序效率[21-23]，运行的操作系统是 Ubuntu 6.10，计算机的 CPU 是 AMD 3000＋，1.81GHz，64 位，内存为 1GB。

5.2 实验结果

程序实现是基于部分开放的源码，它们是台湾大学 CAD 实验室提供的基于多级布线框架的部分源程序[1],UCLA CAD 实验室提供的 Layout Database library ("layoutdb.h" and "layoutdb.a")和测试例子：Struct[2]，以及 LEDA 算法软件包[6]。。

下图 27 即为测试例子 Struct 的布线结果，但是我们实验的主要目的是比较对线网的时延性能的改正情况，所以为了方便比较改正的算法性能和前面提到的其他方法的性能，我们主要考察了该改正算法对单一的多端点线网时延性能提高的实例，在这儿我们将列出 2 个具有代表性的实例，分别参考图 20－33，和图 34－37。

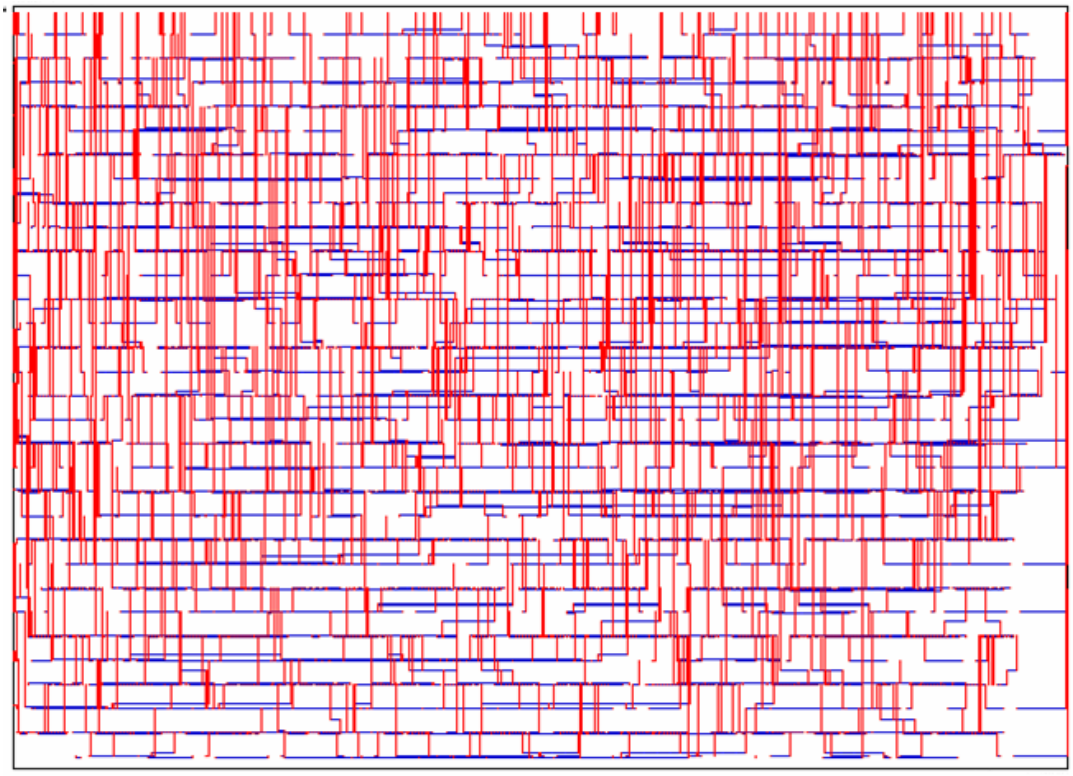

图 28 改正前的布线结果, 为 2 层布线

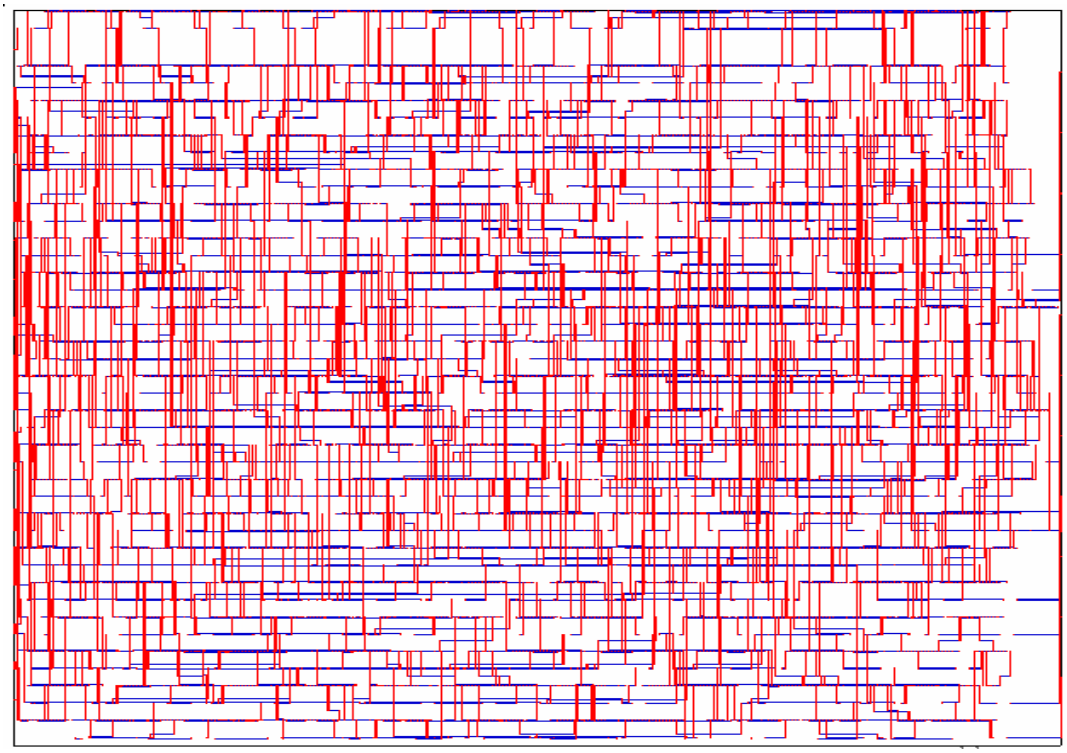

图 29 采用 BMST 算法的布线结果, 为 2 层布线

测试性能对比：

参数	未修正	最小增量修正算法	BMST 算法
网表中最大多端点线网的端点数	17 个	17 个	17 个
线网总数	1920 个	1920 个	1920 个
MDST 分解后的二端点线网数	3551 个	3551 个	3551 个
实际成功布线的线网数	3551 个	3551 个	3551 个
布线成功率	100%	100%	100%
所有线网中最大的时延（ps）	1.6347e+06	1.57071e+06	1.51973e+06
平均的最大的时延（ps）	16239.9	15767.9	14982.8
平均时延（ps）	71255.8	69600.8	64705.8
所有线网的 Manhattan 距离	477800	509963	494232
运行时间	23.31 s	23.67 s	24.29 s

由上面的数据可见，BMST算法相比于最小增量修正算法可以更加有效地改善布线的时延性能。

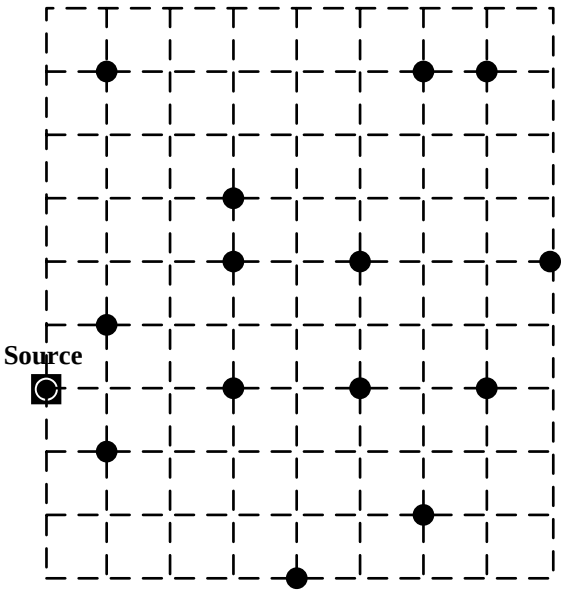

图 30 测试线网实例 1

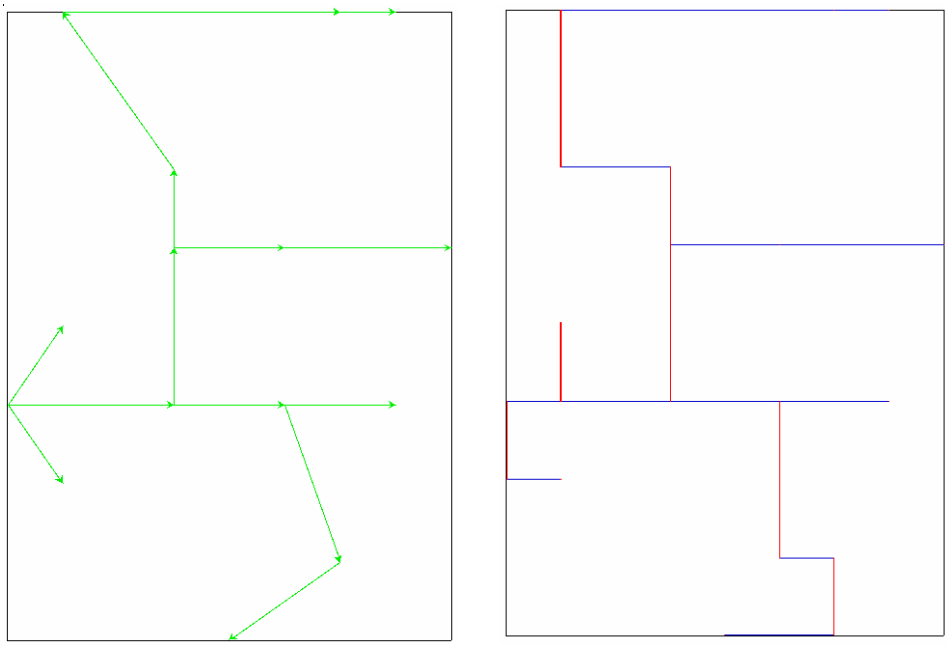

图 31 没有做时延驱动的布线结果

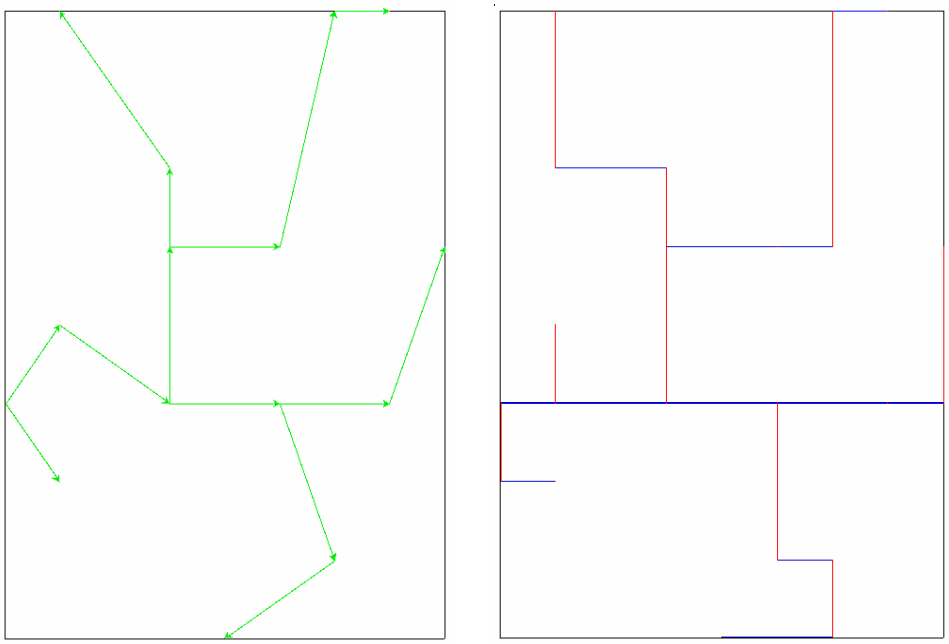

图 32 最短距离生成树的总体布线结果

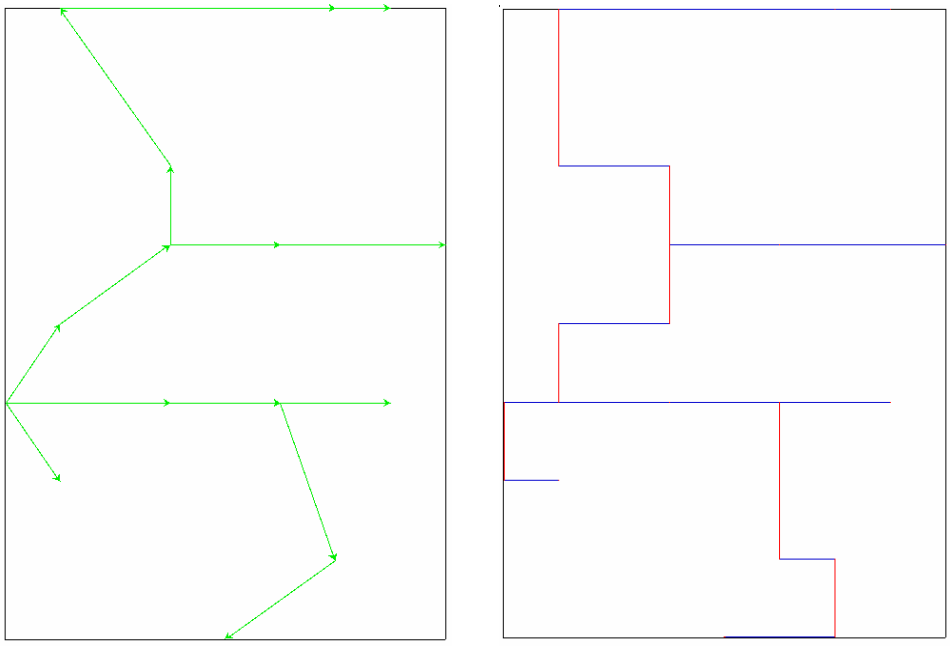

图 33 平衡树时延驱动的总体布线结果

测试线网 1 的性能对比(线网的端点数:15)

参数	未修正	最小增量修正算法	BMST 算法
曼哈顿距离总长	10855	11594 (1.068)	11018(1.015)
线网中最大的时延(ps)	636288	588967（0.93）	553547(0.87)
平均时延	391566	382876（0.98）	305959(0.78)
运行时间	0.91 s	0.82 s	0.75 s

由上面的数据可见，BMST 算法相比于最小增量修正算法可以更加有效地改善布线的时延性能，且对总的布线长度增加更小。

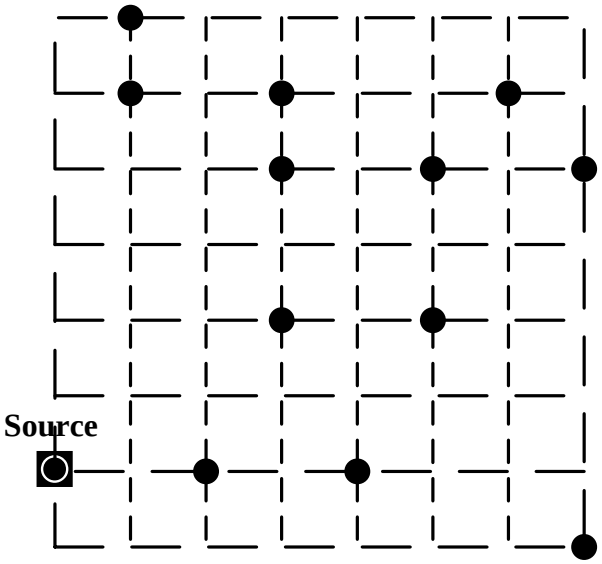

图 34 测试线网实例二

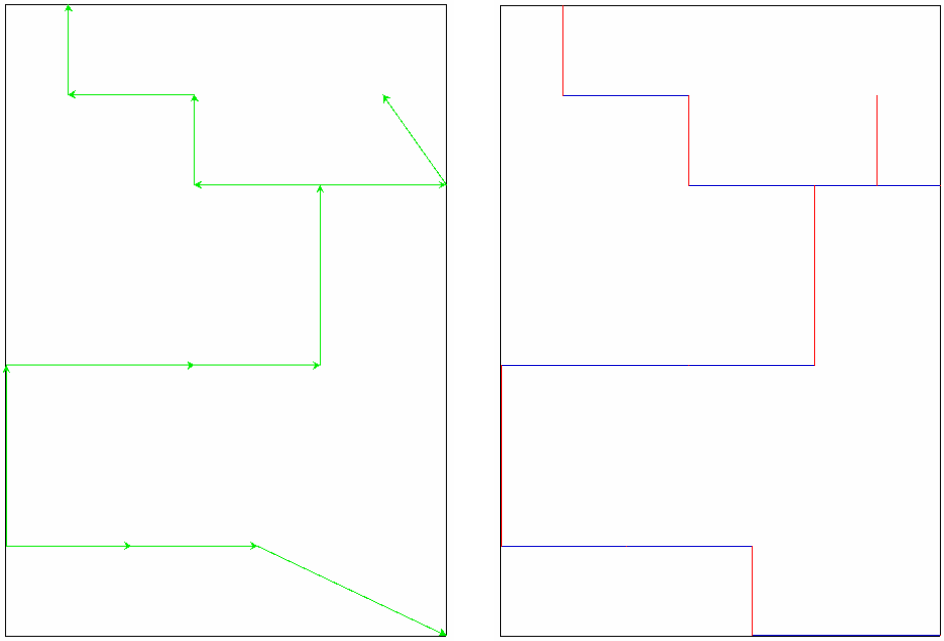

图 35 没有做时延驱动的布线结果

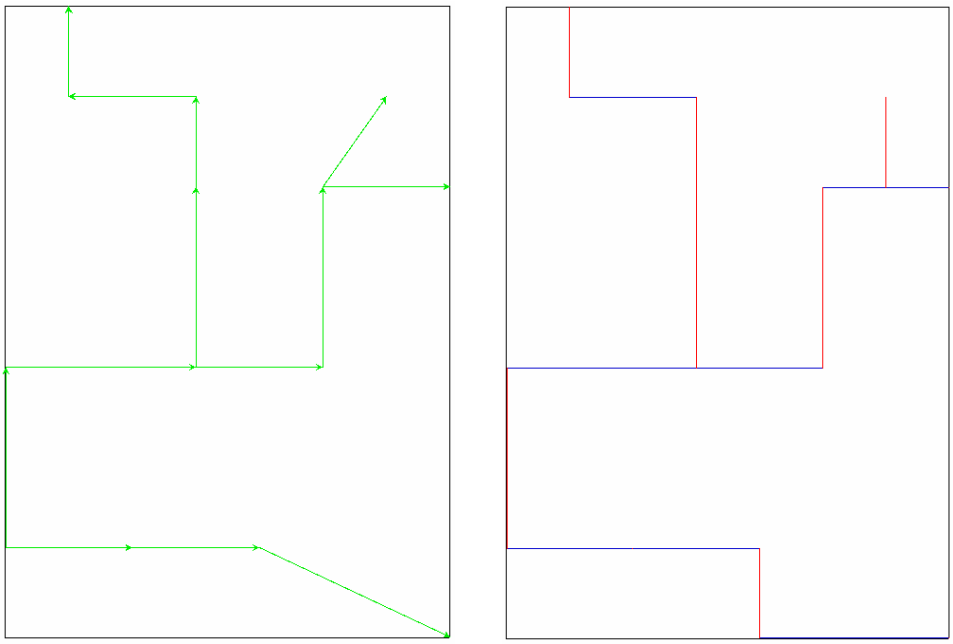

图 36 采用最小增量修正算法的布线结果

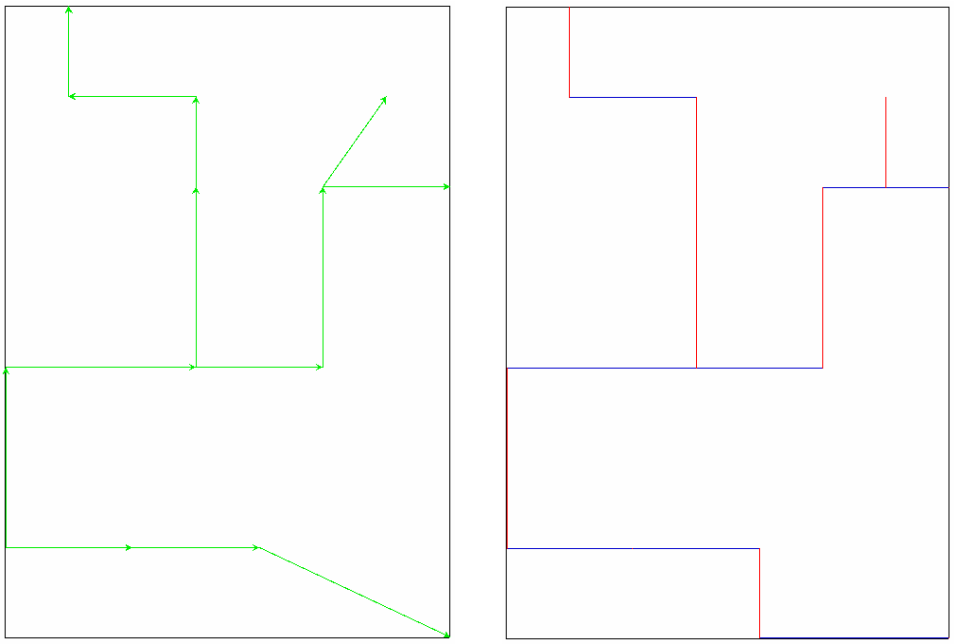

图 37 平衡树时延驱动的布线结果

测试线网 2 的性能对比(线网的端点数:13)

参数	未修正	最小增量修正算法	BMST 算法
曼哈顿距离总长	8407	8637 (1.027)	8637 (1.027)
线网中最大的时延(ps)	646169	458268 (0.709)	458268 (0.709)
平均时延(ps)	411485	324018 (0.787)	324018 (0.787)
运行时间	0.64 s	0.92 s	0.78 s

由上面的数据可见，在某些特殊情况下，BMST 算法相相当于最小增量修正算法。这是因为 BMST 算法包含了最小增量的修正算法的所有优点。则 BMST 算法的性能只会优于，而不会劣于最小增量的修正算法，

5.3 本章小结

本章介绍了实验的操作系统和计算机 CPU，内存等，以及多级自动布线算法对 Struct 例子的布线结果；为了方便比较改正的算法性能和前面提到的其他方法的性能，我们主要考察了该改正算法对单一的多端点线网时延性能提高的实例，我们通过两个多端点的单一线网的实例比较了三种情况下的时延性能，这三种情况分别为没有对最小生成树做时延修正、使用最小增量的时延修正算法，以及使用平衡树的时延修正算法进行时延修正，由实验结果可以看出平衡树的时延修正算法对时延的修正效果比较明显，在最坏的情况下也能达到最小增量的时延修正算法；而一般地，BMST 算法相比于最小增量修正算法可以更加有效地改善布线的时延性能，且对总的布线长度增加更小。

第六章 全文总结

6.1 主要结论

[1][2]中提出的多级布线框架的每一级中引入总体布线，详细布线和拥挤概率估计，且粗化布线和细化布线的划分使得整个布线高效，灵活；[3]通过将 MST 算法修正为 MDST 算法，很好地折中处理了线网的总长最小和最长的线网最短的问题；最小增量的修正方法使得时延能得到很好的满足。

仍然值得研究的是，一般关键路径都是线网长度较长，时序不容易满足，我们可以改变一下布线顺序，

考虑信号完整性的布线的趋势，DRC 和 DFM 等：

6.2 研究展望

随着工艺制造的进步，现在已经进入了深亚微米时代，电路连线中所产生的问题如布线拥挤度，布线时延，以及线网串扰等问题对芯片性能的影响越来越大，这给传统的电路自动布线系统带来了许多新的挑战。在新的工艺下，芯片上能摆放的晶体管数目越来越多，信号的连线数目也大幅增加，所以如何在有限的资源内妥善安排好布线的位置，并且能同时对布线过程中所遇到的布线拥挤，布线时延，以及线网串扰等问题进行最优化是一门很重要的课题。

布线拥挤可能造成一些布线路径必须迂回布线，这将恶化设计性能，甚至导致一些线路无法布通。信号延迟的问题将影响到电路设计的性能，也将对布线结果产生影响。减少串扰的关键在于减少或消除连线之间的耦合电容和耦合电感。在这里主要考虑耦合电容，我们可以通过加大连线的间距，改变连线的形状进而减少并行走线长度的办法。

参考文献

[1] S. P. Lin and Y. W. Chang, “A novel framework for multilevel routing considering routability and performance”, in Proc. of Int. Conf. Computer-Aided Design, pp.44-50, November 2002.

[2] J. Cong, J. Fang, and Y. Zhang, “Multilevel approach to full-chip gridless routing”, in Proc. of Int. Conf. Computer-Aided Design, pp. 396-403, Nov. 2001.

[3] Yih-Lang Lin, Pei-Yu Huang, Chih-Hong Hwang, and Yu-Min Lee, "Performance-and congestion-driven multilevel router", The 13th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI) 2006, 2006-04

[4] J. Heisterman and T. Lengauer, “The efficient solution of integer programs for hierarchical global routing”, IEEE Trans. on CAD, vol. 10, no. 6, pp. 748-753, June 1991.

[5] K. mehlhorn, s. Naher, LEDA A Platform for Combinatorial and Geometric Computing, Cambridge University Press, 1999.

[6] C.-W. Sham, E. F. Y. Young, “Congestion Prediction in Early Stages”, in Proc. System Level Interconnect Prediction, pp. 91-98, April 2005.

[7] T. Chan, J. Cong, T. Kong, and J. Shinnerl, “Multilevel optimization for large-scale circuit placement,” Proc. ICCAD, pp. 171-176, Nov. 2000.

[8] 洪先龙,严晓浪,乔长阁.超大规模集成电路布图理论与算法科学出版社,1998

[9] 庄文君,李玉兴.集成电路布图设计自动化,上海交通大学出版社,1986

[10] C. Y. Lee, an Algorithm for Connections and Its Applications, IRE Trans. on Electronic Computers, 1961, 346-365

[11] S. B. Akers, A Modification of Lee’s Path Connection Algorithm, IEEE Trans. on Electronic Computers, 1967, 16(4), 97-98

[12] D. W. Hightower, A Solution to the Line Routing Problem on the Continuous Plane, Pro. Of the 6th Design Automation Workshop, 1969, 1-24

[13] J. Cong and P. H. Madden, “Performance driven global routing for standard cell design” Proc. ISPD, pp. 73-80, April 1997.

[14] Chiang and M. Sarrafzadeh, Global Routing Based on Steiner Min-Max Tress, IEEE Trans. on CAD, 1990, 9(12), 1318-1325

[15] A. Vanneli, An Adaptation of the Interior Point Method for Solving the Global Routing Problem, IEEE Trans. on CAD, 1991,10(2),193-203

[16] R.C Carden IV and C.K Cheng, A Global Router Using an Efficient Approximate Multicommondity Multiterminal Flow Algorithm, Proc. of IEEE/ACM Design Automation Conference, 1991, 316-321

[17] F. K. Hwang, An O(nlogn) Algorithm for Rectilinear Steiner Trees, Journal of the Association for Computing Machinery, 1976, 26(1), 177-182

[18] F. K. Hwang, An O(nlogn) Algorithm for Suboptimal Rectilinear Steiner Trees, IEEE Trans. on CAS, 1979, 26(1), 75-77

[19] J.M. Ho, G. Vijayan and C. K. Wong, A New Approach to the Rectilinear Steiner Tree Problem, IEEE Trans. on CAD, 1985, 9(2), 185-193

[20] J. Heisterman and T. Lengauer, The Efficient Solution of Integer Programs for hierarchical Global Routing, IEEE Trans. on CAD, 1991, 10(6),748-753

[21] J. Huang, X. L. Hong, C. K. Cheng and E. S. Kuh, An Timing-Driven Global Routing Algorithm, Proc. of 30th IEEE/ACM Design Automation Conference, 1993, 596-599

[22] E. Shragowith and S. Keel, “A global router based on a multicommodity flow model” Integr. VLSI J., vol. 5, no. 2, pp. 3–16, 1987.

[21] Algorithms in C++ (Third Edition), Part 5: Graph Algorithms ISBN 7-302-07251-5

[22] C++高级编程/索尔特（Nicholas A. S.），凯乐普 (Scott J. K.)著；刘鑫等译. －北京：机械工业出版社，2006.1，ISBN 7-111-17778-9

[23] C++ STL 程序员开发指南/彭木根，王淑凌编著.北京：中国铁道出版社，2003.3 ISBN 7-113-05164-2

[24] S.-C. Lee, J.-M. Hsu, and Y.-W. Chang, “Multilevel large-scale module placement/floorplanning using B*-trees,” Proc. The 12th VLSI Design/CAD Symposium, Hsinchu, Taiwan, Aug. 2001.

[25] J. Soukup, “Fast maze router,” Proc. DAC, pp. 100-102, June 1978.

[26] D. Wang and E. Kuh, “A new timing-driven multilayer MCM/IC routing algorithm,” Proc. Multi-chip Module Conf., pp. 89–94, Feb. 1997.

[27] C. J. Alpert, J.-H. Huang, and A. B. Kahng, “Multilevel circuit partitioning,” IEEE Trans. on Computer-Aided Design, vol. 17, no. 8, pp. 655–667, August 1998.

[28] T. Chan, J. Cong, T. Kong, and J. Shinnerl, “Multilevel optimization for large-scale circuit placement,” Proc. ICCAD, pp. 171–176, Nov. 2000.

[29] J. Cong, J. Fang and Y. Zhang, “Multilevel approach to full-chip gridless routing,” Proc. ICCAD, pp. 396-403, Nov. 2001.

[30] J. Cong, J. Fang and K. Khoo, “DUNE: A multi-layer gridless routing system with wire planning,” Proc. ISPD, pp. 12-18, April 2000.

[31] J. Cong, A. Kahng, and K. Leung, “Efficient algorithms for the Minimum Shortest Path Steiner Arborescence Problem with Applications to VLSI Physical Design,” Trans. on Computer-Aided DEsign, vol 17, pp. 24-39, 1998.

[32] Layout Database User Manual (Revision：1.12), Kei-Yong Khoo. UCLA Computer Science Dept., Los Angeles, CA 90095. September 21, 2000

[33] 数据结构算法与应用-C++语言描述/Sartej Sahni, 汪诗林等译.- 一北京：机械工业出版社，2005.10，ISBN 7111076451

符号与标记（附录 1）

VLSI	Very Large Scale Integrated Circuit 超大规模集成电路
Routing	布线
MST	Minimal Spanning Tree 最小生成树
MDST	Minimal Distance Spanning Tree 最小距离生成树
BMST	Balanced Minimal Spanning Tree 平衡的最小距离生成树
Source	源点
Sink/Target	漏点，目标点
Source	源点
Source	源点
Coarsening Routing	粗化布线
Uncoarsening Routing	细化布线
Recalling Modification	回溯修正方法
Shortest Modification	最小增量的修正方法

测试实例（附录 2）

测试实例一：

```
(gdif
  (gdifVersion 1 0 1)
  (comment Generated by tw2gdif)
  (cell:top
      (text:number_of_layers "3")
      (text:wire_widths      "0.6 0.6 0.6")
      (text:via_widths       "0.6 0.6 0.6")
      (text:wire_spacings "1.2 1.2 1.2")
      (text:via_spacings    "1.2 1.2 1.2")
      (text:vertical_wire_costs       "2 1 2 1 2 1")
      (text:horizontal_wire_costs    "1 2 1 2 1 2")
      (text:via_costs "10 10 10 10 10 10")
      (path:BBOX
           (new)(layer LEV)(width 0)(pt -5 52)(pt -5 4956)
           (new)(layer LEV)(width 0)(pt -5 4956)(pt 4898 4956)
           (new)(layer LEV)(width 0)(pt 4898 4956)(pt 4898 52)
           (new)(layer LEV)(width 0)(pt 4898 52)(pt -5 52)
      )
    (port:INS241_a (pt 8 2121))
    (port:INS225_a (pt 500 4242))
    (port:INS209_a (pt 500 4949))
    (port:INS193_a (pt 1000 707))
    (port:INS177_a (pt 1500 2121))
    (port:INS161_a (pt 1500 3535))
    (port:INS145_a (pt 1500 4242))
    (port:INS129_a (pt 2000 707))
    (port:INS113_a (pt 2500 2121))
```

```
    (port:INS97_a (pt 2500 3535))
    (port:INS81_a (pt 3000 4242))
    (port:INS65_a (pt 3500 8))
    (port:INS49_a (pt 3500 3535))
    (port:pad_26_A_0_A_0 (pt 8 707))
    (net:A_0
      (portRef INS241_a)
      (portRef INS225_a)
      (portRef INS209_a)
      (portRef INS193_a)
      (portRef INS177_a)
      (portRef INS161_a)
      (portRef INS145_a)
      (portRef INS129_a)
      (portRef INS113_a)
      (portRef INS97_a)
      (portRef INS81_a)
      (portRef INS65_a)
      (portRef INS49_a)
      (portRef pad_26_A_0_A_0)
    )
  )
)
```

测试实例二：

```
(gdif
  (gdifVersion 1 0 1)
  (comment Generated by tw2gdif)
  (cell:top
      (text:number_of_layers "3")
      (text:wire_widths      "0.6 0.6 0.6")
      (text:via_widths       "0.6 0.6 0.6")
      (text:wire_spacings "1.2 1.2 1.2")
      (text:via_spacings    "1.2 1.2 1.2")
      (text:vertical_wire_costs       "2 1 2 1 2 1")
      (text:horizontal_wire_costs    "1 2 1 2 1 2")
      (text:via_costs "10 10 10 10 10 10")
      (path:BBOX
           (new)(layer LEV)(width 0)(pt -5 52)(pt -5 4956)
           (new)(layer LEV)(width 0)(pt -5 4956)(pt 4898 4956)
           (new)(layer LEV)(width 0)(pt 4898 4956)(pt 4898 52)
           (new)(layer LEV)(width 0)(pt 4898 52)(pt -5 52)
      )
    (port:INS241_a (pt 4000 3535))
    (port:INS225_a (pt 3500 5656))
    (port:INS243_a (pt 3500 2121))
    (port:INS244_a (pt 3000 5656))
    (port:INS193_a (pt 3000 707))
    (port:INS177_a (pt 2500 3535))
    (port:INS161_a (pt 2500 2121))
    (port:INS145_a (pt 2000 8))
    (port:INS129_a (pt 1500 4242))
    (port:INS113_a (pt 1500 3535))
    (port:INS97_a (pt 1500 2121))
    (port:INS81_a (pt 500 5656))
```

```
    (port:INS65_a (pt 500 2828))
    (port:INS49_a (pt 500 1414))
    (port:pad_26_A_0_A_0 (pt 8 2121))
    (net:A_0
      (portRef INS241_a)
      (portRef INS225_a)
      (portRef INS243_a)
      (portRef INS244_a)
      (portRef INS193_a)
      (portRef INS177_a)
      (portRef INS161_a)
      (portRef INS145_a)
      (portRef INS129_a)
      (portRef INS113_a)
      (portRef INS97_a)
      (portRef INS81_a)
      (portRef INS65_a)
      (portRef INS49_a)
      (portRef pad_26_A_0_A_0)
    )
  )
)
```

致　谢

首先，感谢我的导师施国勇教授，是施教授的谆谆导引，才让我有机会进入的EDA 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、信号完整性、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。

在这两年的学习研究生活中，也要感谢所有陪伴我一同经历、成长的同学，他们是王骝、修宇、志刚和历国。也要感谢世杰等学第学妹们在这些日子的陪伴和论文撰写中的协助。

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] 李小南. 多级自动布线框架的应用介绍. 信息技术 （已录用）