第18卷第3期 半导体学报 Vol. 18, No. 3
1997年3月 CHINESE JOURNAL OF SEMICONDUCTORS Mar., 1997

# 分级的时延驱动布局算法

孔天明 洪先龙

(清华大学计算机科学与技术系 北京 100084)

**摘要** 本文针对门阵列和标准单元设计系统提出一种分级的时延驱动布局算法. 以前的时延驱动布局算法除了文献[22]以外都不是分级的,因而运算时间很长,而且最长路径上的信号延迟达不到最优;而文献[22]的算法只能处理时序关系是DAG图(有向无环图)的电路,也就是说,电路中不能包含寄存器元件. 本文的算法是适用于一般的电路. 与RITUAL/Tiger系统比较,我们用比较短的运算时间得到了较小的信号延迟.

**EEACC**: 2570; **CCACC**: 7410D

## 1 引言

随着超大规模集成电路工艺的飞速发展,高速、高性能的系统越来越受到人们的重视. 在集成电路工业中存在两种趋势:其一是器件的特征尺寸越来越小,门的开关速度则越来越快,使得互连线延迟的影响越来越明显;另一趋势则是应用对于高性能、高速度的集成电路的需求越来越迫切. 因此,对于电子设计自动化的研究人员来说,性能日益成为研究的重要课题. 在物理布图设计领域,这一目标就是要精确地控制关键路径的信号延迟.

以前的时延驱动布局算法要么是基于线网[1,4,7,9,12,14,19,20,22]的,要么是基于路径[2,5,8,11,14,15,17,18]的. 基于线网的算法一般将关键路径上的时延信息转化成线网的长度上界,然后在布局的过程中努力控制这些线网的长度. 这种做法通常来说过分地限制了布局算法;因为满足这些线网的长度上界只是满足时延约束的充分条件而不是必要条件. 所以,基于线网的布局算法得到的结果布线质量往往很差. 基于路径的算法则正确地描述了时延问题,但以前的算法都存在一个共同的问题:随着电路的规模的增长,关键路径的数目将会急剧增长;因此,以往的算法对于大规模电路的问题其求解时间都会很长. 其中的原因之一是这些算法都不是基于分级设计的;如果采用分级设计的方法,这一问题将会得到缓解(这一点在我们的实验中已经得到证实). 只有一个算法文献[22]是基于分级设计方法的,但是其基本思想只能适用于DAG图(有向的无环图),也就是说,其处理的电路中不能包含寄存器元件,而这一点在实际的电路设计中是不现实的.

---

孔天明 男,1971年生,博士生,本文通讯联系人(Email Address: {kongtm,hong}@tiger.dcs.tsinghua.edu.cn),目前研究领域为ICCAD,侧重于布图设计

洪先龙 男,1940年生,教授,主要研究领域为ICCAD

1996年2月5日收到初稿,1996年5月22日收到修改稿

本文第2节给出算法过程的概述;第3节给出延迟的计算公式.结群算法和布局算法分别在第4和第5节给出.第6节中给出了复杂度估计;最后,我们给出了实验结果.

## 2 算法过程概述

在给定一个待设计的大规模电路时,我们认为最有效的设计方法是"分而治之".因此,我们采用了处理布局问题的一种古老然而有效的方法,将结群技术与好的时延驱动布局算法结合起来.图1给出我们提出的分级布局总的框架.

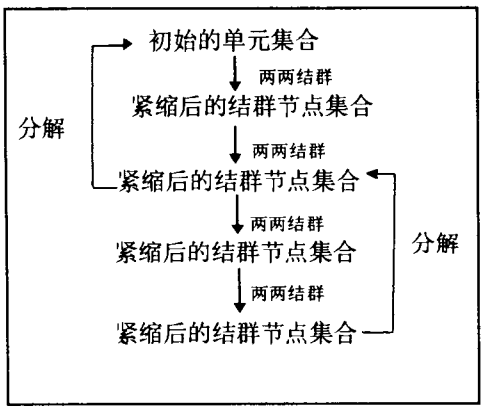


图1 分级布局

我们的结群过程和传统的结群方法有3点很大的不同:1. 一般的结群算法并不局限于两两结群,而我们则强制要求两两结群;这是因为,随着设计的电路规模的增大,布局问题本身越来越复杂,要求的运算时间越来越长,所以作为布局算法的预处理阶段的结群算法耗费过多的时间将是不合时宜的.我们认为,结群算法所用的时间占总的系统时间的比例在5%的限度内是比较合适的;2. 在传统的结群方法中,初始的网表要被压缩成第二、第三级网表,同时,单元之间的连接关系图也被修改以反映这种压缩后的状况.在布局阶段利用这些压缩后的网表对结群节点进行布局.这种方法比较适合于线长目标的优化,但并不适合于时延的优化,尤其是时序关系图是一个一般有向图(含有寄存器元件)的时候.事实上,结群的唯一目的是降低计算复杂度,因此我们在结群过程中并不修改时序关系图,从而保证了正确的时序关系;3. 在传统的方法中,较高一层的布局完成以后,立即进行下一层的布局.事实上,这并不是必须的.我们的系统可以处理下面两层甚至三层的布局.

## 3 互连线的延迟计算

给定线网 $n_i$,其互连线的延迟可以估计为:

$$D_n = R_s \times (C_n + C_n^g) + R_n \times C_n^g \tag{1}$$

式中 $R_s$ 是驱动单元的输出电阻;$C_n^g$ 是驱动单元所见到的总的负载电容;$C_n$、$R_n$ 分别是分布电容和分布电阻.在布局过程中,我们用线网的星形连接模型来估计分布电容和分布电阻:

$$C_n = C_h \times \sum_{j \in n_i} |x_j - x_s| + C_v \times \sum_{j \in n_i} |y_j - y_s| \tag{2}$$

$$R_n = R_h \times \sum_{j \in n_i} |x_j - x_s| + R_v \times \sum_{j \in n_i} |y_j - y_s| \tag{3}$$

式中 $C_h$,$C_v$ 分别是 $x$ 和 $y$ 方向单位长度互连线上的电容;$R_h$,$R_v$ 分别是 $x$ 和 $y$ 方向单位

长度互连线上的电阻；$x_s$，$y_s$ 表示驱动单元的坐标. 将(2)和(3)式替换入(1)式，我们得到：

$$D_n = R_sC_n^g + (R_sC_h + R_hC_n^g) \times \sum_{j\in n_i} |x_j - x_s| + (R_sC_v + R_vC_n^g) \times \sum_{j\in n_i} |y_j - y_s|$$

$$= T_n + S_n^x \times \sum_{j\in n_i} |x_j - x_s| + S_n^y \times \sum_{j\in n_i} |y_j - y_s| \tag{4}$$

式中　$T_n = R_sC_n^g$，$S_n^x = R_sC_h + R_hC_n^g$，$S_n^y = R_sC_v + R_vC_n^g$.

任何一条路径 $p$ 的到达时间(延迟)$a_p$ 是该路径上所有线网延迟 $D_n$ 和门延迟 $G_g$(在布局过程中是一个常数)的总和，即 $a_p = \sum_{\forall n\in p} D_n + \sum_{\forall g\in p} G_g$. 电路时序是正确的当且仅当电路中的任何一条路径 $p$，其到达时间(延迟)小于或等于一个给定的延迟上界 $T$(可根据芯片的时钟频率给定)，即：$a_p = \sum_{\forall n\in p} D_n + \sum_{\forall g\in p} G_g \leqslant T$.

## 4　时延驱动的结群算法

我们采用了一种两两递归的结群算法. 算法的工作过程如下.

首先，我们将时序信息转化成无向图中的边权；其结果是我们得到了一个带权的无向图 $G(V,E)$，其中 $V$ 中的顶点 $\nu$ 对应于一个单元，该单元我们写作 Cell($\nu$). $(\nu_1,\nu_2)\in E$ 当且仅当存在线网 $n$ 连接 Cell($\nu_1$)，Cell($\nu_2$)而且 Cell($\nu_1$)或者 Cell($\nu_2$)是该线网的驱动单元. 因而对于任意线网 $n_i$，假设其由单元 Cell($\nu_1$)驱动，同时连接单元 Cell($\nu_2$)，Cell($\nu_3$)，…，Cell($\nu_{|n|}$)，则将有 $|n|$ 条边$(\nu_1,\nu_i)\in E$，$i=1,2,\cdots,|n|$. 在第 3 节，我们提到，每条线网 $n$ 有两个相关的项 $S_n^x$，$S_n^y$. 现在，我们对所有线网的 $S_n^x$，$S_n^y$ 求和得到 $S^x$，$S^y$. 我们对每条边$(\nu_1,\nu_i)$赋权为 $S_n^x/S^x + S_n^y/S^y$. 图 2 左边是电路的时序关系图，右边是转化后的带权的无向图.

给定带权的互连图 $G(V,E)$以后，结群的任务就是寻找顶点之间的最大带权匹配. 这个问题已经有现成的算法，但复杂度是 $O(n^3)$. 因此，我们给出了一个启发式算法来求解，基本思想是：从一个已有的匹配出发，寻找一个更好的匹配. 算法基于下面的定理(证明略)：

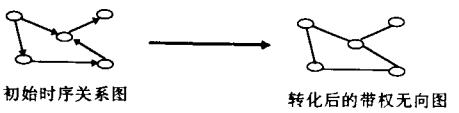


图 2　结群所需的带权无向图的构造

给定初始匹配 IM，存在一个单调的匹配修改序列达到最优匹配 OM.

根据上面的定理，我们只需在图中寻找增益为正的回路. 但是，困难在于：一个增益为正的回路可能其局部某一段的增益为负. 幸运的是，我们有这样的事实：给定一个总增益为正的回路，总存在一点，如果从之出发遍历，可以使得到达任何一点时当前所得的增益为正. 因此，我们的结群算法就是在图中迭代寻找尽可能多的增益为正的回路；每次找到一个回路后，将这些回路中的顶点锁定，防止它们参加下一次的迭代. 上面的事实用来剪除不必要的搜索. 注意到这一算法的复杂度仍然是较高的，因此我们在实现中限定了迭代的次数，防止迭代搜索的时间过长.

## 5　时延驱动的布局算法

在结群以后，在布局的每一阶段，时延驱动的布局问题可以表示为下面的数学规划问题：

Minimize $L = 1/2w^TQw + b^Tw$

Subject to: $a_j \geqslant a_i + D_n$, $i, j \in n$, $i$ 是线网 $n$ 的驱动单元

$a_i = 0$ $i$ 是原始输入或寄存器的输出端

$a_j \leqslant T$ $j$ 是原始输出或寄存器的输入端

式中 $L$ 是总线长的二次估计；$w$ 是表示结群节点的坐标的向量；目标函数中的常向量 $b$ 是由于我们将某些单元如原始输入、原始输出等视为固定的单元.

除了上面的时序约束外，还有一类约束需要考虑：我们必须将所有的单元均匀地分散到芯片上. 为了求解这类约束，我们采用了层次划分的办法. 在划分的第一层，我们在约束方程中增加如下两个约束条件(记当前的结群节点集合为 $CN$)：

$$\sum_{i \in CN} x_i / |CN| = \mathrm{Chip}X/2.0$$

$$\sum_{i \in CN} y_i / |CN| = \mathrm{Chip}Y/2.0$$

也就是说，我们强制要求所有的结群节点的"质心"必须在芯片的中心. 在第一层的处理完毕之后，我们根据结群节点的当前坐标对它们进行排序，将芯片划分成大小相等的出四个区域，同时为每个节点规定对应的区域，然后增加相应的约束方程..... 依次如此处理. 一般地，设某个区域的中心坐标为($mcx_i$, $mcy_i$)，其对应的结群节点集合为 $MC_i$，则对应的约束方程为：

$$\sum_{j \in MC_i} x_j / |MC_i| = mcx_i$$

$$\sum_{j \in MC_i} y_j / |MC_i| = mcy_i$$

因此，该数学规划问题可以用矩阵表示为：

Minimize $L = 1/2w^TQw + b^Tw$

Subject to: $Aw \leqslant c$

我们采用拉格朗日松弛法(Lagrange Relaxation)来求解这个问题. 设约束不等式对应的拉格朗日乘子向量为 $\lambda$. 对于以等式成立或不满足的约束不等式，对应的乘子向量分量 $\lambda_i \geqslant 0$. 根据拉格朗日松弛法，原问题等价于解一系列的无约束的拉格朗日子问题：

$$\begin{aligned} \text{Minimize} \quad L &= 1/2w^TQw + b^Tw + \lambda^T(Aw - c) \\ &= 1/2w^TQw + (b^T + \lambda^TA)w - \lambda^Tc \end{aligned}$$

这个问题有一个非常简单的解析解：

$$w = -Q^{-1}(b + A^T\lambda)$$

由此分析，我们的时延驱动布局算法由以下主要步骤组成：

1. 求解无约束的拉格朗日子问题；
2. 计算最优步长；
3. 更新拉格朗日乘子和延迟表示；
4. 如果达到了相应的精度，则：
   4.1 如果划分的层次已经足够，则返回；
   4.2 否则，进行新的划分，增加约束方程，转 1；
5. 否则，转 1.

### 5.1 求解无约束的拉格朗日子问题

每次迭代过程中，我们需要进行如下的矩阵计算：

$$w^{k+1} = -Q^{-1}(b + A^{k^T}\lambda^k)$$

如果我们不采用分级设计的方法，这个矩阵计算将是极其耗费机时的. 由于我们采用了分级设计的方法，这个矩阵的维数较小. 我们对矩阵 $Q$ 进行 LU 分解以得到其逆矩阵. 而且在以后迭代过程中，求逆运算不用进行了，因而我们仅需计算 $A^{k^T}\lambda^k$.

### 5.2 计算最优步长

当我们得到一个解以后，我们还需要计算最优步长(相当于一维搜索)；这是因为互连延迟函数表达式中存在绝对值项. 数学规划中处理绝对值项的传统方法是引进额外的变量. 这里我们不采用这种方法. 我们直接根据当前的解 $w^{k^*}$ 写出延迟函数表达式. 然后由 5.1 中得到一个解；某些绝对值项的符号可能要变化. 我们如下计算最优步长 $t^{k+1}$，使得最后的目标解 $w^{(k+1)^*}$ 中最多仅有一个绝对值项改变符号；从而得到目标解：

$$w^{(k+1)^*} = (1 - t^{k+1})w^{k^*} + t^{k+1}w^{k+1}$$

直觉地，关键路径越多，需要考虑的绝对值项就越多，从而最优步长就越小，向最优解的收敛速度也就越慢. 在这一点上，分级设计的算法相对优越于非分级的算法，因为在分级设计的算法中，每次迭代中考虑的关键路径数目以及绝对值项数都比较少. 我们在实验中发现，关键路径的数目对于算法的速度影响是至关重要的；分级设计的方法能有效地减少关键路径的数目：在 RITUAL/Tiger 系统对电路 s13207 的实验中，我们发现关键路径的数目急剧地增加，从而显著地降低了收敛速度；而在我们的算法中，关键路径的数目从来没有超过 300.

### 5.3 更新拉格朗日乘子向量

我们采用 SOR 法来更新拉格朗日乘子向量，即：

$$\lambda^{k+1} = \max\{0, \lambda^k + (A^{k+1}w^{(k+1)^*} - c)\}$$

这里的 max 是对每个分量进行的. 对于时序约束而言，增量 $A^{k+1}w^{(k+1)^*} - c$ 就是对应路径的松弛量；而对于节点分布约束，该增量就是节点实际的“质心”位置与要求的“质心”位置的差值.

### 5.4 更新延迟表示

得到一个新的解以后，我们还有两步工作要作，其一是更新矩阵 $A$(因此我们用 $A^k$，$A^{k+1}$来表示其变化)，正如 5.3 中谈到的，延迟函数的表达式需要重写；此外我们还必须更新关键路径集合. 所有非关键的路径都是不需要的，因为它们的拉格朗日乘子为 0. 对于每一条新加入的关键路径，我们假定其初始的拉格朗日乘子为 0. 这里我们再一次看到：关键路径的数目是一个影响收敛速度的重要因素.

## 6 复杂度估计

假定：1)结群的深度为 $h$，最后的结群节点数目为 $k$；2)每次布局过程中，需要进行 $d$ 次划分；3)每次迭代过程中，关键路径的平均数目为 $p$，每条关键路径经过的平均节点数为 $s$. 若记总的单元个数为 $n$，则我们有 $n \approx 2^h \times k$. 假定每个划分中的节点单元个数为 $q$，则 $2^d \times q \approx k$，因此 $d \approx \log_2(k/q)$.

每次布局之前要进行一次矩阵($k$ 阶)分解,其复杂度为 $O(k^3)$. 每次迭代过程中要进行下列工作:1)矩阵相乘运算 $O(k)$;2)关键路径的延迟更新,复杂度为 $O(ps)$;3)关键路径集合的更新,复杂度为 $O(n)$. 因此每次迭代的复杂度为 $O(k+ps+n)$. 从而每次布局的复杂度为 $O(k^3+d(k+ps+n))$.

设总的布局次数为 $f$,则总的复杂度为 $O(fk^3+fd(k+ps+n))$. 考虑最简单的分解方法:做完最高层的布局之后,直接进行叶子层的布局,则我们有 $f=2^h+1\approx 2^h$. 因此,总的复杂度为 $O(n^3/2^{2h}+n(\log_2(n/q)-h)+2^h(\log_2(n/q)-h)(ps+n))$. 在实验中,我们发现,在规模较大的电路中,有两项起到很大的作用:一项是 $O(n^3/2^{2h})$,这是最重要的复杂度贡献者,它随着 $h$ 的增加急剧下降,这正是我们采用结群算法的初衷;另一项是与 $ps$ 即关键路径数目以及长度有关的,从理论上无法找出 $ps$ 与 $h$ 的准确关系,但实际计算中发现 $ps$ 随着 $h$ 的增加而减少,而且对收敛速度有着重要的影响.

## 7 实验结果

我们共测试了四个电路:C2,C5,C7,s13207. 电路的特征数据列于表 1 中. 表 2、3、4、5 中,我们分别比较了我们的算法和 RITUAL/Tiger 系统所得到的初始布局的最长路径延迟、总线长、边界框线长、水平线长、垂直线长以及运行时间(我们的结果用 MIXED 表示)等等指标.

**表 1 电路特征**

| 电路名称 | #原始输入 | #原始输出 | #单元 |
|---|---|---|---|
| C2 | 233 | 140 | 590 |
| C5 | 178 | 123 | 1586 |
| C7 | 207 | 108 | 2150 |
| s13207 | 700 | 790 | 4267 |

**表 2 C2 的布局结果**

| C2 | 最长路径延迟 | 总线长 | 边界框线长 | 水平线长 | 垂直线长 | 运行时间 |
|---|---|---|---|---|---|---|
| RITUAL | 67.13 | 559196.19 | 544853.62 | 321786.34 | 237409.72 | 587.06 |
| MIXED | 42.03 | 573869.00 | 559720.75 | 315502.44 | 258366.47 | 998.81 |
| 提高(%) | 37.4 | −2.6 | −2.7 | 2.0 | −8.8 | −70 |

**表 3 C7 的布局结果**

| C7 | 最长路径延迟 | 总线长 | 边界框线长 | 水平线长 | 垂直线长 | 运行时间 |
|---|---|---|---|---|---|---|
| RITUAL | 78.84 | 1949249.75 | 1936646.00 | 1196407.50 | 752842.00 | 1084.62 |
| MIXED | 65.93 | 2012434.00 | 1969010.62 | 1128578.50 | 883854.94 | 1174.21 |
| 提高(%) | 16.4 | −3.2 | −1.7 | 5.7 | −17.4 | −8.3 |

**表 4 C5 的布局结果**

| C5 | 最长路径延迟 | 总线长 | 边界框线长 | 水平线长 | 垂直线长 | 运行时间 |
|---|---|---|---|---|---|---|
| RITUAL | 65.52 | 940513.31 | 810693.44 | 528304.88 | 412208.19 | 1498.91 |
| MIXED | 41.41 | 1223049.62 | 1127694.25 | 658020.19 | 565030.38 | 823.59 |
| 提高(%) | 36.8 | −30 | −39 | −24.5 | −37 | 45 |

**表 5　s13207 的布局结果**

| s13207 | 最长路径延迟 | 总线长 | 边界框线长 | 水平线长 | 垂直线长 | 运行时间 |
|---|---|---|---|---|---|---|
| RITUAL | 197.86 | 7163237.50 | 6796474.50 | 4006561.25 | 3156650.75 | 6427.78 |
| MIXED | 88.35 | 11277667.00 | 10866050.00 | 6637266.50 | 4640408.00 | 3604.97 |
| 提高(%) | 55 | −57.4 | −14.9 | −65.7 | −47 | 43.9 |

对于电路 C2，最长路径延迟有显著的减少(37.4%)，而线长只有很少的增加(3.0%). 虽然运行时间增加了 70%，但总的运算时间在 15 分钟之内，是可以接受的. 原因在于，较小规模的电路，分级的开销占了主要的运算时间；C7 的比较结果比较类似，延迟的提高达到 16%，线长也只有很少的增加. 图 3 中绘出了两个系统对于电路 C7 的布局结果(图中每个点表示一个基本单元)，由图中可以看出，我们得到的布局中，单元的分布比较均匀，而 RITUAL 的结果中，单元大量集中于芯片的四周，因此不利于后续的消除重叠操作.


图 3　RITUAL/Tiger 和 MIXED 产生的布局结果(电路 C7)

但在电路 C5，s13207 中，最长路径延迟的减少受到几乎是同等的线长增加的惩罚，当然执行时间有了显著的减少. 线长的增加有两个方面的原因：1) 我们的结群算法没有照顾线长目标；2) 在每次迭代过程中，我们在一个较小的集合上优化，因此，节点的分布约束作用就较为严厉.

注意到电路 s13207 中，本算法得到的最长路径延迟只有 RITUAL/Tiger 的一半不到. 我们跟踪了 RITUAL/Tiger 的计算过程，发现其中的一个严重问题：关键路径的数目在不同的迭代之间是指数增长的(最多的时候超过了 2880 条)，这种情况严重影响了其算法的有效性；我们的算法中没有出现这个问题(最大的关键路径数目没有超过 300 条)，主要是因为我们算法是基于分级设计的.

### 7.1　结群对于布局质量的影响

针对电路 C7 和 s13207，我们测试了不同结群次数时布局质量的不同，结果见表 6.

**表 6　结群对于布局质量的影响**

| | 2 次结群 | | 1 次结群 | | 不结群 | |
|---|---|---|---|---|---|---|
| 电路 | 延迟 | 线长 | 延迟 | 线长 | 延迟 | 线长 |
| C7 | 1.00 | 1.00 | 0.92 | 0.93 | 0.90 | 0.98 |
| s13207 | 1.00 | 1.00 | 0.91 | 0.92 | 0.90 | 0.93 |

由此可见，结群使得布局质量有所下降，延迟的最大下降幅度为 10%，线长的最大下降幅度为 7%. 但不结群到 1 次结群之间布局质量只有很小的下降，因此适当的结群方法是有效的.

## 8 结论和进一步工作

我们针对小单元(small-cell based)电路提出了一个分级的时延驱动布局算法.以往的算法除了文献[22]以外都不是分级的,而文献[22]的算法只能处理时序图是DAG图的电路,因而是不能够实用的.我们的算法与RITUAL/Tiger系统比较,在最长路径延迟这一目标上有显著的提高.从实验结果来看,我们可以断言:适当的结群技术是有利于时延目标的.由于目前我们的结群算法相对简单,处理线长目标不是很理想,因此我们将进一步寻求更好的结群算法,在保证较好的延迟目标的前提下,尽可能减少连线总长.

## 参 考 文 献

[1] M. Burstein and M. N. Youseff, "Timing Influenced Layout Design", Proc. 22th DAC, 1985, 124~130.

[2] W. Donath, R. Norman, B. Agrawal *et al.*, "Timing Driven Placement using Complete Path Delays," Proc. DAC, June 1990, 84~89.

[3] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," Operations Research, **17**, 1969, 395~412.

[4] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch *et al.*, "Chip Layout Optimization Using Critical Path Weighting," Proc. 21st DAC, 1984, 133~136.

[5] P. de Forcrand and H. Zimmermann, "Timing-Driven Auto-Placement," Proc. Int. Conf. on Comp. Design, 1987, 518~521.

[6] T. Gao, P. M. Vidya and C. L. Liu, "A New Performance Driven Placement Algorithm," Proc. Int. Conf. on Computer-Aided Design, November 1991, 44~47.

[7] T. Gao, P. M. Vidya and C. L. Liu, "A Performance Driven Macro-Cell Placement Algorithm," Proc. DAC, June 1992, 147~152.

[8] T. Hasegawa, "A New Placement Algorithm Minimizing Path Delays," Proc. Int. Conf. on Computer-Aided Design, 1991, 2052~2055.

[9] P. Hauge, R. Nair and E. Yoffa, "Circuit Placement for Predictable Performance," Proc. Int. Conf. on Computer-Aided Design, 1987, 88~91.

[10] R. B. Hitchcock, G. L. Smith and D. D. Cheng, "Timing analysis of computer hardware", IBM Journal of Research and Development, 1983, **24**(1): 100~105.

[11] M. Jackson and E. Kuh, "Performance-Driven Placement of Cell Based IC's," Proc. DAC, June 1989, 370~375.

[12] M. Jackson, E. Kuh and M. Marek-Sadowska, "Timing Driven Routing for Building Block Layout," Proc. of Int. Symp. Circuits and Systems, 1987, 518~519.

[13] E. Lawler, Combinatorial Optimization: Networks and Matroids, (New York: Holt, Rinehart, and Winston, 1976), 98~106.

[14] M. Marek-Sadowska and S. Lin, "Timing Driven Placement," Proc. Int. Conf. on Computer-Aided Design, 1989, 94~97.

[15] A. Srinivasan, "An Algorithm for Performance-Driven Initial Placement of Small-Cell ICs," Proc. DAC, June 1991, 636~639.

[16] W. J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits", Proc. Int. Conf. on Computer-Aided Design, 1993, 170~177.

[17] S. Sutanthavibul and E. Shragowitz, "Dynamic Prediction of Critical Paths and Nets for Constructive Timing-Driven Placement", Proc. DAC, June 1991, 632~635.

[18] W. Swartz and C. Sechen, "New Algorithm for the Placement and Routing of Macro Cells", Proc. Int. Conf. on Computer-Aided Design, 1990, 336～339.

[19] M. Terai, K. Takahashi and K. Sato, "A New Min-Cut Placement Algorithm for Timing Assurance Layout Design Meeting Net Length Constraint", Proc. DAC, June 1990, 96～102.

[20] R. Tsay and J. Koehl, "An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement", Proc. DAC, June 1991, 620～625.

[21] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network", Management Science, **17**, July 1971, 712～716.

[22] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits", Proc. Design Automation Conference, June 1995, 211～215.

# Hierarchical Timing-Driven Initial Placement for Row-Based ICs

Kong Tianming and Hong Xianlong

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Received 5 February 1996, revised manuscript received 22 May 1996

**Abstract** We present a hierarchical timing driven initial placement algorithm for row-based ICs. Previous approaches to timing driven placement were not hierarchical, thus suffered from long running timing and non-optimal longest path delay; except [22], while which could only handle DAG(directed acyclic graph), i.e., circuits cannot contain memory element(s). Our algorithm is generally applicable to each kind of small-cell circuits. Compared with RITUAL/Tiger system, promising results of longest path delay objective are reported with less execution time.

**EEACC**: 2570; **CCACC**: 7410D