第 22 卷第 1 期　　半 导 体 学 报　　Vol. 22, No. 1
2001 年 1 月　　CHINESE JOURNAL OF SEMICONDUCTORS　　Jan., 2001

ECOP: 一种基于单元行划分的标准单元模式增量布局算法*

于 泓 姚 波 洪先龙 蔡懿慈

(清华大学计算机科学与技术系, 北京 100084)

摘要: 针对标准单元模式超大规模集成电路增量式布局问题, 提出了一个全新的增量布局算法 ECOP. 该算法一改以往布局算法中以单元为中心的做法, 变为以单元行为中心, 围绕单元行来进行单元的插入, 移动以及各种约束条件的处理. 在划分单元行时, 始终保持单元行的内部连通性, 并对单元移动路径进行搜索与优化. 对一组来自美国工业界的设计实例进行了测试. 实验结果表明, ECOP 算法是非常实用而高效的.

关键词: 增量式布局; 单元行划分; 内部连通性; 优化

EEACC: 2570　　**CCACC**: 7410D

中图分类号: TN47　　**文献标识码**: A　　**文章编号**: 0253-4177(2001)01-0096-06

1 引言

在现代集成电路工业的实际应用中, 当详细布局完成后, 甚至在布线工作也已完成的情况下, 由于各种原因, 需要对部分或全部完成的电路设计加以修改. 按照以往的方法, 修改电路设计的办法有两种: 从头再布局, 或者手工加以改动. 重新布局的方法代价太大, 即使是局部的小改动, 也要将所有单元重新布局, 使设计周期大大延长. 而交互式的手工修改版图的方法, 对于小规模电路还能处理, 但对于超大规模电路却无能为力. 这样, 就迫切需要寻找一种能自动地以最小代价对原电路设计进行局部调整的布局方法, 这就是所谓的增量式布局, 也叫做 ECO 布局(Engineering Change Order).

增量式布局, 是随着集成电路设计规模的急剧扩展而出现的一种新型的自动布局问题, 这在电路规模很小时是没有的(因为手工修改可以胜任). 这是近些年才出现的新问题, 目前国内外这方面的文献资料并不多见. 1996 年, Choy[1] 提出了一种基于门阵列模式电路的增量布局算法, 这是针对增量布局问题的较早的有益探索. 然而, 对于目前在工业界非常流行的标准单元模式的增量布局方法, 迄今却未见有任何文献发表. 本文提出一种新的基于单元行划分的标准单元模式增量布局方法 ECOP, 并已经成功应用到各种规模电路的布局问题中. 实验结果和在工业应用中的实测数据表明, 该方法对于解决标准单元模式增量布局问题是卓有成效的.

2 问题的形成

增量式布局是为了满足超大规模集成电路设计的需要而提出来的. 在布局(甚至布线)完成以后, 由于各种原因, 需要移动一些单元的位置, 或插入和删除一些单元. 比如, 为使线网能布通, 在某些位置需要插入走线道(Feed-Through)单元; 为改善时钟布线的时延特性而插入缓冲器(Buffer)单元; 为完成电源线/地线网络拓扑方案而不得不移动一些单元的位置等等. 这些单元插入, 删除或移动的数量和位置往往要知道详细的布图结果以后才能确定, 所以

* 国家基础研究(G1998030413)和国家自然科学基金(6977602)资助项目.

于 泓 博士生, 从事电子设计自动化(EDA)系统中布图软件的研究与开发工作.

洪先龙 教授, 1964 年进入清华大学工作.

1999-11-29 收到, 2000-03-30 定稿

布局(甚至布线)完成后的局部电路修改,及布局调整问题,很多时候是无法避免的. 一般,这样的布局调整分两个步骤进行: 首先,由电路设计者根据已有设计方案,提出对原始电路的修改意见. 即由设计者确定在什么地方移动,插入和删除哪些单元,并且说明在调整时所必须满足的约束条件. 比如: 哪些单元不能被移动,哪些区域不能被占用等等. 然后进入第二个阶段,按照前面的修改意见及提出的约束条件,对原设计方案进行布局调整. 增量式布局的任务就是完成这第二阶段的工作,按照电路设计者的修改意见并在满足设计者提出的约束条件的要求下,以最小的代价,获得一个"合理的"新设计方案.

2.1　目标函数的选取

增量式布局算法本身,究其数学本质,是一个优化问题. 对于布局问题而言,优化的目标函数可以有多种选择: 最小线长、最小时延、最小割线和最小拥挤度等. 具体到增量式布局,则有其问题本身的特殊性. 我们之所以不去重新布局,而是在原有设计方案的基础上对原有方案进行布局调整,设计周期太长和设计代价太大固然是一个很重要的考虑因素,而更为重要的一点则是: 我们已经默认原有布局方案是一个"好"方案,否则,如果是对一个很差的布局方案进行局部调整,则我们的工作将变得毫无意义.

既然原方案是一个"好"方案,所以对原方案的修改当然是"越少越好". 于是,一个很自然的优化目标函数就是: 单元的总位移量最小. 在优化单元总位移这一目标函数下,我们可以期望在新的布局方案中,能够尽量保持原电路设计方案的"原貌",以期所获得的新布局方案也将是一个"好"方案.

2.2　增量式布局的约束条件

在原有布局设计方案的基础上进行修改和调整,就需要满足原设计方案的一些特点和约定. 设计者在提出修改要求时,也会提出一些诸如某些区域不能占用,某些单元不能移动等约束条件. 我们将这些要求和约束条件统称为约束条件. 不同的电路设计问题,提出的约束条件会有所不同. 对于标准单元模式电路设计方案,比较典型的约束条件有以下几种.

2.2.1　多种高度单元行与单元行重叠

在一个芯片中,往往会存在多种 SITE,每种 SITE 对应不同高度的单元,相应的也就存在多种高度的单元行,比如: 单高、双高和三高单元行等. 相同高度的单元行之间,一般不会存在相互重叠. 然而,在不同高度的单元行之间,则往往会存在着重叠,如图 1 所示,Row1 为双高单元行,Row2 为单高单元行,在 Row1 和 Row2 之间就存在着重叠. 在这种情况下,当双高单元 A 被放置在 Row1 上时,单元 A 所占据的 Row2 中的位置就不允许放置任何单高单元了. 同样,当单高单元 B 放置在 Row2 中,在放置双高单元时,亦不允许与单元 B 相重叠.

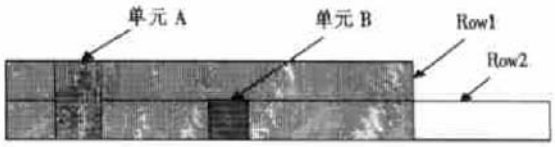

图 1　多种高度的单元行和单元行重叠

FIG. 1　Cell Row in Varied Height and Its Overlapping

2.2.2　KEEPOUT 约束和固定单元约束

KEEPOUT 约束,即指一定的区域(一般为矩形),在进行增量布局时,区域内不得放置任何单元. 固定单元约束就是在进行增量布局前,固定某些单元的位置,在进行增量布局的过程中,位置不得改变.

2.2.3　位置栅格约束

由于后续布线等的需要, 每个单元行上设有等间距的一系列栅格. 标准单元在被放置在单元行上时, 要求单元的左端线与单元行上的某一个栅格点相对齐, 而不是在单元行里随意摆放, 如图 2 所示.

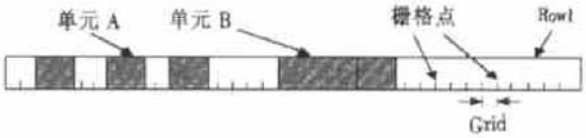

图 2　位置栅格约束

FIG. 2　Constraint for Seat Grid

2.2.4　初始位置冲突

对原方案提出修改意见后, 在增量布局软件的初始输入数据中, 常会出现单元之间相互重叠在一起等"发生冲突"的情形, 这就要求增量布局软件能自动进行检测和修正. 还可能出现诸如单元行不连续, 单元初始位置不合法, 单元行纵向排列等特殊情况, 这都要求增量布局软件能识别并进行相应处理.

3 ECOP 算法描述

3.1 栅格宽度与数据格式转换

在标准单元电路中,高度不同的单元,分别被定义成不同种类的 SITE. 对于每一种 SITE,都定义了一系列的单元行,并对应一组单元,标准单元只能放置到同种 SITE 的单元行中. 一般来说,单元的宽高、位置和单元行的长宽、位置等信息都是浮点数,为了满足标准单元的位置必须位于所在单元行的"栅格"点上的要求,我们采取如下策略:把单元宽度和相应单元行的长度都转换成栅格间距 grid 的整数倍,分别叫做 gridWidth 和 gridLength,这样当我们按照"整数宽度"来安置单元时,单元就会"自动"位于单元行的"栅格"点上. 如图 2 所示,假设单元 A 的宽度为 26μm,单元 B 的宽度为 50μm,单元行 Row1 的行长为 50μm,栅格 grid 为 12.5μm. 则按照我们的转换方法,单元 A 的 gridLength 为 3,单元 B 的 gridLength 为 4,单元行 Row1 的 gridLength 为 40. 在图 2 中,单元 A 位于"栅格点"6 处,单元 B 位于"栅格点"17 处,它们的左端线都自动对齐在单元行的栅格点上.

3.2 处理 KEEPOUT 约束

如图 3 所示,当单元行 Row1 和 Row2 与 KEEPOUT 相重叠时,我们采取将一个单元行"切断"的方法,将 Row1 和 Row2 分别一分为二,成为四个新的单元行. 这样做的好处是:可以保证每一个单独的单元行内部都将是连通的. 用同样策略,也可以很容易地处理"不连续的单元行"约束. 如果把固定单元看作是一种特殊的 KEEPOUT,则固定单元约束问题也可得到解决.

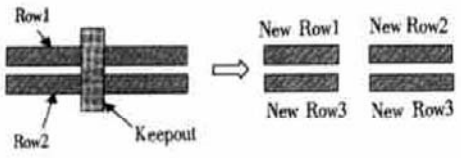

图 3 处理 KEEPOUT 约束

FIG. 3 Processing KEEPOUT Constraint

3.3 位置检测

在初始输入电路中,单元之间可能存在相互重叠,或虽然某单元不与其它单元相重叠,但它的位置却不是位于单元行上,是"不合法"的. ECOP 算法在布局调整之前,首先把位置"合理"的单元放置在单元行上(应该是零位移),然后把位置"不合理"的单元放入 TroubleCells 集合中,等待后续处理.

3.4 布局调整

随后对电路进行布局调整. 首先,按照 SITE 的种类,把单元和单元行分别归类. 每一种 SITE 对应与其同种的单元和单元行. 对于某一种 SITE,相应的单元行就相当于"空房间",而单元就相当于"房客",ECOP 的任务就是把单元放置到与其对应的同种单元行上,并使得总位移量最小. 因为不同种 SITE 的单元行之间通常会有重叠,为了避免它们在放置时发生重叠,我们规定 SITE 的高度越大,与其对应的单元优先级越高. 我们把 SITE 按照高度从大到小的顺序,逐个 SITE 进行处理,把属于当前种 SITE 的单元,安置到与该种 SITE 相对应的单元行上去. 排位靠后的单元在安置时,如果遇到某位置已被排位靠前的单元所占据,则排位靠后的单元必须"迁就"排位靠前的单元.

3.5 安置 Trouble 单元

Trouble 单元是指由前面检测出来,位置不合理并被放入 TroubleCells 集合中的那些单元. 安置 Trouble 单元就是分别以最小代价为 TroubleCells 集合中的每一个单元找到一个合适的位置,并把此单元插入这个位置中. 具体处理时,首先对 TroubleCells 集合中的单元进行排序,属于不同种 SITE 的单元之间,SITE 高度大的单元排在前面;属于同种 SITE 的单元之间,宽度大的排在前面. 这样做的道理很简单:因为在刚开始时,单元行的"空块"资源比较多,为尺寸较大的单元寻找位置较容易.

3.5.1 代价函数

前面已经叙述,ECOP 的优化目标函数为单元的总位移量最小. 单元的总位移量 M_{total} 包括水平位移 M_x 和垂直位移 M_y 两部分:$M_{total} = M_x + M_y$. 根据标准单元电路的特点,无论对于边界框模型,还是星形模型,等量的水平位移和垂直位移对于线网的 Bounding-Box 的改变是不一样的,因此对原电路性能改变的贡献量也是不一样的. 通常单元的垂直位移对原电路性能的改变比等量水平位移对原电路性能的改变要大得多. 于是代价函数取为如下形式:

$$\mathrm{Price}_{\mathrm{tottal}} = \alpha M_x + (1-\alpha) M_y \qquad (1)$$

式中　$\mathrm{Price}_{\mathrm{total}}$为总位移; M_x 和 M_y 分别为水平和垂直方向的位移; α 是取值范围为[0, 1]的加权系数. 对于单元行横向排列的标准单元模式电路, 单元水平方向移动对于电路性能(包括线长、时延、可布性、功耗均匀等)的改变, 远没有垂直位移的影响显著, 因此 α 取值应小些, 比如当取 α= 0.2 时, 垂直位移的权重(1- 0.2= 0.8)将为水平位移权重的 4 倍. 值得指出的是: 当需要调整的单元属于某种"阵列"形式的电路(比如寄存器组)时, 为保持时延的对称性, 需要"成组移动", 方法类似, 只是操作复杂得多.

3.5.2　寻找目的单元行

放置一个 Trouble 单元, 需要给它找到合适的位置. 采取按行寻找的办法: 对于每一个单元行, 先检查此单元行是否有足够的空块资源. 不是要求单元行内有一个现成的足够放置此 Trouble 单元的空块, 而是只要该单元行内的空块之和大于等于此单元的宽度就行. 如不够则继续检测下一个单元行; 如果当前单元行的空块资源足够大, 则按照 (1) 式来计算移动当前 Trouble 单元的代价. 随后再搜索下一个单元行, 直到所有单元行. 最后取代价最小的单元行作为安置此 Trouble 单元的目的单元行.

3.5.3　计算移动代价

在按照(1)式计算代价时, 首先应确定此 Trouble 单元在当前单元行中的"假想插入位置". 在这里, 我们采取"最小移动"原则: 即把 Trouble 单元的"假想插入位置"定位在水平方向移动最小的位置. 如图 4 所示, 目前的 Trouble 单元为 TroubleCell, 当计算移动到单元行 Row 1 的代价时, 它的左端线应与图中 A 所指处对齐, 而右端线则应与单元行 Row 1 的右边界对齐(即处于 Row 1 的最右端); 当计算移动到单元行 Row 2 的代价时, 它在 Row 2 中的"假想插入位置"的 X 坐标应维持不变, 即仅仅垂直平移到 B 处.

3.5.4　行内单元位置调整

当 TroubleCell 被放置到目的单元行中的"假想插入位置"时, 很可能与行内的原有单元发生位置重叠. 在图 4 中, 当计算移动到 Row 1 的代价时, TroubleCell 被放置到 Row 1 的 A 处时将与原来行内单元 Cell3 发生位置重叠; 当计算移动到 Row 2 的代价时, TroubleCell 被放置到 Row 2 的 B 处时将与 Cell1 和 Cell2 发生重叠. 因此, 还需要对单元行内的单元进行移动调整. 这里我们采取"挤"的办法,

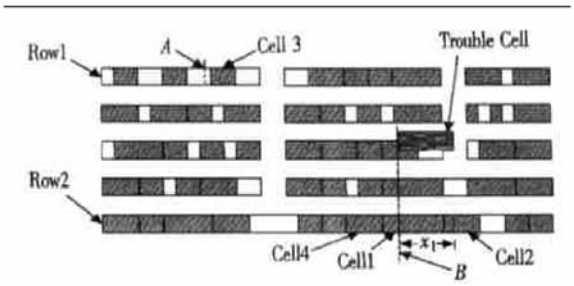

图 4　计算移动代价

FIG. 4　Calculating Shifting Cost

这又分成三种方式: 向左挤, 向右挤, 两边挤. 下面假设图 4 中 TrouleCell 被移动到 Row 2 中 B 的位置, 这时, TroubleCell 与 Cell1 和 Cell2 发生重叠, 首先查看 TroubleCell 的插入点 B 的左面的空块资源是否大于等于重叠宽度 x_1, 只有当左边的空块资源大于等于 x_1 时, 才尝试向左挤. 在向左挤时, 可能会引起一连串单元移动位置. 如图 4 中, 在 TroubleCell 向左挤时, 推动 Cell1 向左移动, Cell1 进而又推动 Cell4 向左移动, 这种移动将持续到左边所有单元均无重叠时才停止. 由于事先已经确认插入位置左边有足够的空块资源, 因此, 只要决定做"向左挤"这个动作, 就一定可以成功. 与此类似, 可以尝试做"向右挤"的动作. "两边挤"则是同时尝试向左和向右推动其他单元, 以消除单元位置重叠. 每一次做单元行内位置调整时, 向左挤, 向右挤和两边挤三个动作都要进行尝试, 分别计算三种情况下的 M_x:

$$M_x = \left| \mathrm{new}X_{\mathrm{trouble}} - \mathrm{old}X_{\mathrm{trouble}} \right| + \sum_{C_i \in \mathrm{Row}} \left| \mathrm{new}X_i - \mathrm{old}X_i \right| \qquad (2)$$

式中　$\mathrm{new}X_{\mathrm{trouble}}$和 $\mathrm{old}X_{\mathrm{trouble}}$分别为 TroubleCell 的新旧 X 方向坐标; $\mathrm{new}X_i$ 和 $\mathrm{old}X_i$ 则是目标单元行 Row 中其他原有单元在进行行内位置调整前后的新旧水平坐标. (1)式中的 M_y 则由下式来确定:

$$M_y = \left| \mathrm{new}Y - \mathrm{old}Y \right| \qquad (3)$$

$\mathrm{new}Y$ 和 $\mathrm{old}Y$ 分别为 TroubleCell 移动前后位置的垂直坐标. 原来处于目标单元行中的单元进行行内调整后, 垂直坐标不会变化, 因此并没有在(3)式中出现.

一般来讲, 在同一个目标单元行上, 向左挤, 向右挤和两边挤三种方式 M_x 是不同的, 我们要取其中最好的(M_x 取值最小)按照(1)式计算 TroubleCell 移动到当前目标行的代价 Price. 应该说明的一点是: 向左挤和向右挤这两种移动方式并不是总可

以进行的,它们只有在插入点左或右边的空块资源充足的情况下,才可以进行;但两边挤这种方式却总是可以进行的,因为事先已经知道,此目标单元行拥有足够的空块资源以摆放当前的TroubleCell.

3.5.5 寻找交换路径

对于当前的TroubleCell,在找到将要放置它的目标单元行以后,就要确定一条合适的交换路径,以便使得在总位移量最小的情况下,单个单元的最大位移量最小,即:

$$\mathrm{Min}(\mathrm{Max}(\mathrm{Price}_i))\ \ \mathrm{st}: \mathrm{Min}(\mathrm{Price}_{\mathrm{Total}}),\ \mathrm{cell}_i \in \mathrm{Cell}_{\mathrm{move}} \tag{4}$$

式中 $\mathrm{Cell}_{\mathrm{move}}$为插入TroubleCell时所移动单元的集合.如图5所示,现假定已经找出当前TroubleCell应放置的目标单元行为Row1,TroubleCell应放置的位置为Row1中的A处.注意到当TroubleCell按照图5右侧路径被直接由初始位置放置到A处时的代价price1与按照图5中左侧路径,先将TroubleCell放置到Cell1的位置,再将Cell1放置到Cell2处,最后将Cell2放置到A处的代价price2是相等的(因为两种情况下M_x和M_y是相等的).可是,后一种情形下,单个单元的最大位移量却要小得多,因此,后一种移动方案较优.为了减小算法的时间复杂度,我们规定搜寻工作只在TroubleCell的初始位置和目标插入位置之间的矩形区域进行,只搜索处在此区域中的单元,且每步查看一个单元,最多搜索五步.为了避免不必要复杂情形的引入,规定在搜寻交换路径时必须保证不产生新的单元重叠.显然,搜寻工作不可能失败,因为最坏情况无非就是象图5中右侧路径所示,把TroubleCell一步就放到目标位置A处.

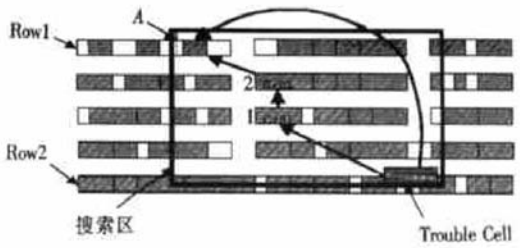

图5 寻找交换路径

FIG.5 Finding Exchanging Route

3.6 处理当前SITE单元与后续SITE单元行之间的重叠

为避免放置属于不同SITE的单元时发生重叠,我们按照不同SITE的高度按由大到小的顺序依次处理.设由属于当前SITE的所有单元组成的集合为CurrentCells,当CurrentCells中单元已全部安置好后,需要将这些单元加以处理,以便在处理属于后面的高度较小的SITE的单元时,不会与已布好的高度较大的CurentCells中的单元相重叠.这里我们把与CurrentCells中单元相重叠的高度较小的SITE所对应的单元行,按照类似前面处理KEEPOUT约束的策略,在中间打断,把每个重叠单元行一分为二,产生两个新的较小的单元行.注意:用这种方式可以保证所产生的与特征尺寸较小的SITE相对应的单元行,在单元行的内部依然是连通的.保持单元行的"内部连通性"很重要,因为只有这样,才能保证在接下来处理下一个SITE时,进行目标单元行寻找,及行内单元位置调整等工作时不会出现意外的麻烦.

表1 测试电路主要特征参数

Table 1 Main Characteristic Parameters

Circuits	Cells*	Chip_X** /μm	Chip_Y*** /μm
Adptc	46710	1471.2	1225.0
Cnt100	646	394.5	393.9
Cnt200	1309	556.5	555.1
Cnt500	3636	955.5	955.5
Cnt1000	7146	1567.5	1569.1
M16_4	2802	874.5	874.9
Mult_32	7018	1351.5	1350.7
Po300	2005	739.5	739.7
Sny	24584	3469.5	3425.5
Tsb	16056	1487.2	1487.2

* Cells: 单元数目; ** Chip_X: 芯片宽度尺寸;
*** Chip_Y: 芯片高度尺寸.

表2 ECOP算法测试结果

Table 2 Measuring Results for ECOP Algorithm

Circuits	X_{Total}/μm	X_{Avg}/μm	Y_{Total}/μm	Y_{Avg}/μm	$\mathrm{Cells}_{\mathrm{Moved}}$	CPU Time/s
Adptc	648878.28	201.64	39779.38	287.63	3219	756.91
Cnt100	147.66	2.38	126.02	3.94	62	0.18
Cnt200	216.60	1.65	180.38	2.78	131	0.52
Cnt500	1523.49	4.01	864.64	4.78	380	3.42
Cnt1000	2302.75	4.03	2800.67	7.85	572	14.48
M16_4	247.98	1.48	612.36	4.37	168	1.94
Mult_32	1419.14	2.74	2363.55	6.75	517	12.90
Po300	334.18	2.03	3.69	3.69	165	1.03
Sny	71319.51	16.38	1318.72	1.07	4.35	147.31
Tsb	20671.43	6.28	1112.14	1.39	3291	77.43

X_{Total}: 水平方向总位移; Y_{Total}: 垂直方向总位移; X_{Avg}: 水平方向平均移动一次的位移量; Y_{Avg}: 垂直方向平均移动一次的位移量; $\mathrm{Cells}_{\mathrm{Moved}}$: 总共移动的单元数; CPU Time: 运行时间.

4 实验结果

我们在 Sun UltraSparc-1 工作站上用 C 语言实现了 ECOP 算法, 并对 ECOP 进行了实验测试. 本文测试电路取自美国某公司的一组版图设计实例, 其主要特征参数指标如表 1 所示, 实验测试结果列在表 2 中. 从小到 600 多个单元的电路, 到大到 46000 单元的电路, 我们都给出了测试数据. 从实验结果可以看出, ECOP 算法对于求解标准单元模式电路增量布局问题是卓有成效的, 无论对于算法的稳定性、实用性, 还是高效性, 测试结果都是令人满意的.

5 结论

本文首先介绍了超大规模集成电路增量布局问题, 随后提出了一种全新的标准单元模式增量布局算法 ECOP. 在该算法中, 我们一改以往布局算法中以单元为中心的通常做法, 变为以单元行为中心, 围绕单元行来进行单元的插入, 移动以及各种约束条件的处理. 在划分单元行时, 始终保持单元行的内部连通性, 并对单元移动路径进行搜索与优化. 实验结果表明, ECOP 算法具有稳定性好、实用性强、求解质量高和运算速度快等特点. ECOP 算法不仅实验结果令人满意, 而且目前已被工业界制造商成功应用于各种实际电路的设计中.

参考文献

[1] G. De Micheli, A. Sangiovanni-Vincentelli and P. Antognetti, "Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation", in Proc. NATO Adv. Study Instit., 1987, 113—195.

[2] B. Preas and M. Lorenzetti, Physical Design Automation of VLSI Systems, Menlo Park, CA: Benjamin/Cummings, 1988, 461—497.

[3] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, San Francisco, CA: Freeman, 1979.

[4] Chiu-sing Choy, Tsz-Shing Cheung and Kam-Keung Wong, IEEE Trans. CAD, 1996, **15**(4): 437—445.

[5] KONG Tianming, HONG Xianlong and QIAO Changge, Chinese Journal of Semiconductors, 1997, **18**(9): 692—700(in Chinese).

ECOP: A Row-Partition Based Incremental Placement Algorithm for Standard Cell Layout Design*

YU Hong, YAO Bo, HONG Xian-long and CAI Yi-ci

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

Abstract: Incremental placement is a new type of automatic placement which appears with the rapid expanding scale of VLSI circuits. A novel incremental placement algorithm ECOP is presented for the standard cell layout design. This approach deals with the cell inserting, cell moving and constraints processing based on rows instead of on cells as most placement algorithms usually do. In the course of row partition, ECOP manages to maintain the inner- integrality of each row, and tries optimize the shifting paths of cells. Algorithm on a set of sample circuits from American industry has been tested and the results have shown that our approach is very efficient and effective.

Key words: incremental placement; row partition; inner-integrality; optimization

EEACC: 2570　　**CCACC**: 7410D

Article ID: 0253-4177(2001)01-0096-06

* Project Supported by National Foundation Plan(973) of China Under Grant No. G1998030413 and National Natural Science Foundation of China Under Grant No. 69776027.

Received 29 November 1999, revised manuscript received 30 March 2000