Automatic Layout Design Scaling with SKILL Language

BI Zong-jun, LUO Lan, YAN Jun
(The National ASIC Systems Engineering Technology Research Center, Southeast University, Nanjing 210096, China)

Abstract: A method of scaling layout with program to realize technology transplant is presented. The location and shape of object in the layout is determined by the point list, processing the points with the same factor can scale or relocate the object without shape change. Based on the theory, the function library was established and a recurrence algorithm processing the hierarchical layout was shown. The method and examples to correct the DRC errors by program were discussed and a complete design flow was introduced too. Experimental results with the less design time, area reductions of up to 48% and gate delay reduction of 40% demonstrate the effectiveness of the approach.

Key words: SKILL; DFII; DRC; hard cores; scale
EEACC: 2570A

基于 SKILL 语言的按比例自动缩放版图方法

毕宗军, 罗 岚, 杨 军
(东南大学国家专用集成电路系统工程技术研究中心, 南京 210096)

摘 要: 使用程序自动缩放版图设计实现硬核的快速工艺移植。版图中对象位置和形状由点的序列构成, 对各点乘以相同的缩放因子可以在不改变对象形状的前提下对任意形状对象进行缩放或位置搬移。基于此原理采用建立函数库的方法构建程序, 使用递归算法处理层次化的版图并给出编程修改 DRC 错误的实例。在给出 SKILL 程序实现的基础上给出了一个完整的设计流程。实践结果显示设计时间缩短、硬核性能得到提高, 面积缩小 48%, 门延时缩短 40%。

关键词: SKILL; DFII; DRC; 硬核; 按比例缩放
中图分类号: **TN402** 文献标识码: **A** 文章编号: **1005-9490(2006)04-1187-05**

集成电路制造工艺技术遵循着著名的“摩尔定律”和“按比例缩小定理”不断更新发展。新工艺的集成度更高器件特性更好从而使得集成电路规模和性能在不增加成本的前提下成倍提高, 在新的工艺上实现设计复用可以极大的提高设计效率。

目前复用的设计一般以软核、固核或硬核的形式存在, 软核、固核和硬核间的权衡要考虑到可移植性、灵活性以及复用效率和面市时间等因素。硬核是指在性能、功耗和面积上经过优化并映射到特定工艺技术的可复用模块。传统的硬核复用较之于软核和固核具有较高的效率和较短的面市时间, 但是被局限在特定的工艺上, 如何快速高效的实现设计向新工艺移植是硬核复用的关键。本论文提供了一个使用 Cadence 公司 SKILL 语言实现版图按比例自动缩放的方法来实现设计工艺的移植。这种方法直接从设计的版图层次着手参照工艺规则利用一系列的缩放和搬移操作来完成版图工艺之间的转换, 从而增强了硬核的可移植性和可复用性。

本论文中涉及到一个 0.25 μm 工艺的包含多块 SRAM 单元的硬核, 工程目标是在 0.18 μm 的新工艺中复用这个硬核。分析发现, 由于原工艺的限制硬核的面积比较大并且性能较低(相对于新工艺而言), 不改变特征尺寸的直接复用可能会导致其成为整个设计性能上的瓶颈, 所以决定对复用的硬核进行缩放处理使之完全移植到新的工艺上来。完全移植到新工艺需要解决两个问题: 缩放版图的尺寸

收稿日期: 2005-11-16
作者简介: 毕宗军(1976-), 男, 硕士在读, 研究方向为补充芯片设计, zjbi@hotmail.com;

以及满足新工艺设计规则。Cadence 的 ICFB 设计平台提供的 SKILL 环境能够方便的访问和处理版图数据,满足版图工艺移植的软件需求。

1 自动按比例缩放原理

完整的硬核设计由几何版图(layout)以及原理图(schematics)或网表(netlist)构成。版图包含了设计的物理信息,原理图和网表包含设计的逻辑信息。要实现设计的移植就要同时对版图和原理图或网表进行处理。

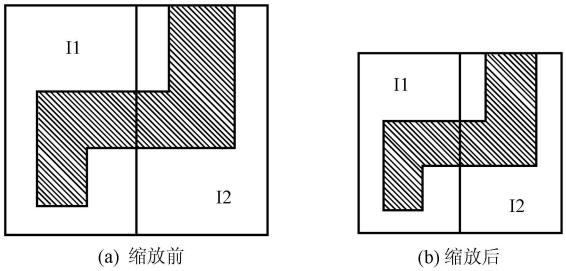
在主要的 EDA 工具中,Cadence 的 ICFB 设计平台使用统一的 DFII(Design Framework II)数据库保存设计的物理和逻辑信息,并且提供了访问数据库的 SKILL 语言接口[1]。DFII 数据库中保存的数据被称作"对象(objects)",按照类型(objType)可以分为多边形(polygon)、矩形(rect)、路径(path)以及实例(inst)等等。每一个对象都具有自己的属性(properties)。其中多边形和矩形包括构成点(points)以及层(layer)的属性等等;路径包括点、线宽(width)以及层的属性等等;实例则有实例名(name)、名称(cellName)以及坐标点(xy)属性等等。

版图中的几何图形都是利用对象来描述的,对对象的点或坐标点乘以相同的缩放因子可以在不改变对象形状的前提下实现对任意形状对象的缩放或位置的搬移。基于这个基本原理,按比例改变多边形和矩形各个点的坐标,路径点的坐标和线宽(width),实例的坐标点就可以实现版图的按比例缩放。式(1)和(2)给出了一个点坐标变换的基本公式,其中 factor 是缩放因子:

$$new\_x = old\_x * factor \quad (1)$$

$$new\_y = old\_y * factor \quad (2)$$

图 1 个给出了一个层次化版图缩放的例子,实例 I1 和 I2 本身以及其在顶层的实例坐标点经过相同缩放因子的处理后,图形的形状和物理连接保持不变。



(a) 缩放前 (b) 缩放后

图 1 层次化版图的缩放

原理图的修改相对比较简单,只要按比例修改原理图中各个器件的宽度和长度即可。

对版图进行缩放要针对不同的设计和目标选择不同的缩放策略。可以对不同层使用不同的缩放因子;可以一步缩放到位或留有余量以换取较小的后期修正代价。版图的缩放涉及到版图中所有对象的所有点,使用软件编程可以快速完成这种重复而繁琐的工作从而实现版图缩放的自动化。

2 SKILL 语言的实现

SKILL 语言[2] 是 Cadence 公司开发的一种语言,提供了一系列访问 DFII 数据库的函数可以方便地对版图数据库进行管理和编辑。

版图数据库中的每个对象由一个标识符(identifier)唯一表述。为了访问版图数据库中的对象首先要获取对象的标识符。在 SKILL 语言中通过版图和指定的区域的交迭操作(Overlap)来获取目标对象标识符。

SKILL 语言提供了多种函数来获取对象的标识符,部分函数列于表 1。第一个函数用于获取定义的区域内所有的图形对象的标识符。第二和第三个函数用于获取定义的区域内特定层次上的所有图形对象的标识符。这两个函数的区别在于是使用真实边界还是轮廓边界决定图形的大小,在图 2 中第二个函数比第三个函数多获取对象 D 而后者的运行速度也较慢。第四个函数用于获取定义的区域内实例的标识符。合理的指定交迭区域或层次属性可以减小数据处理量提高程序的运行效率。

表 1 对象标识符获取函数

1	dbProduceOverlap(cellDb cellDb-> bBox)
2	dbGetOverlaps(cellDb obj-> bBox "M1")
3	dbGetTrueOverlaps(cellDb obj-> bBox "M1")
4	dbProduceOverlapInst(cellDb cellDb-> bBox)

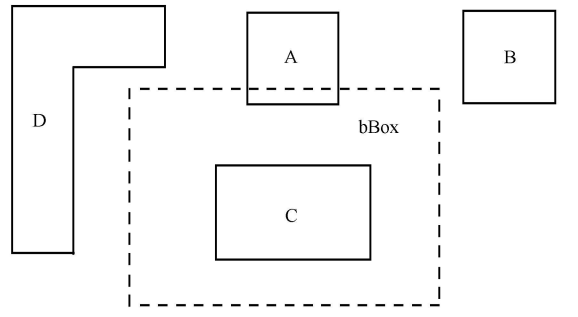


图 2 交迭操作获取对象标识符

序列(LIST)是 SKILL 语言中最常用的数据结构之一。标识符获取函数返回的就是一个序列,例如(db: 249888928 db: 92573044 db: 92572984)。根据对象标识符就可以访问对象的属性。DFII 数据库中不同类型对象使用不同的属性来描述,通过改变相应的属性就可以实现图形的缩放,见表 2。

表 2　缩放图形使用到的属性

对象类型	涉及缩放图形的属性
矩形(rect)	obj->bBox
多边形(polygon)	obj->points
路径(path)	obj->points　obj->width
文本(label)	obj->xy　obj->height

ICFB 环境将一个设计表示成“库—单元—视图”的层次结构, 设计就是调用库中单元的组合。对应于版图层次化组织结构的处理程序构建在一个递归算法基础之上, 程序的流程图见图 3。程序首先需要做参数配置, 包括设计库和顶层单元的名称、缩放因子以及各层的层号定义等。程序从顶层单元开始, 首先处理当前层次的图形(矩形、多边形、路径), 然后获取当前层次中的实例、对需要处理的实例进行同样的图形和实例处理然后再返回到上一级层次继续未完成的处理直至所有的实例处理完毕。

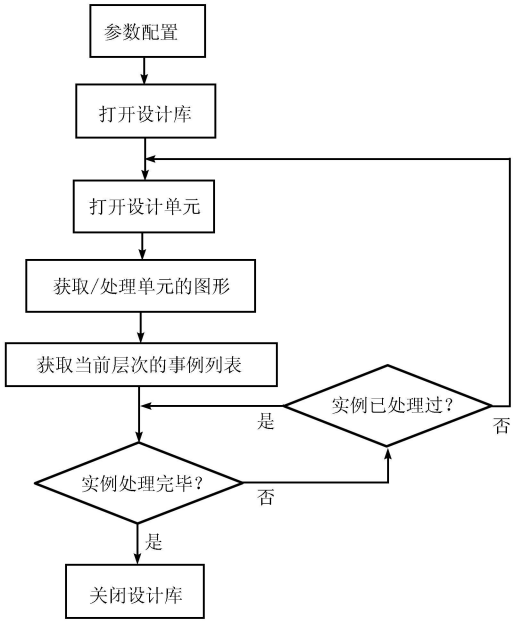


图 3　版图缩放程序流程图

算法 1: 层次化版图缩放算法

```
1  procedure( CELL_RECURE( cellDb hier)
2    prog( (obj child_cellDb instanceList shapeList newx newy
3      xy cell_hit unflatten factor)
4     PRO_FIGURE_CENTER(cellDb factor)
5     instanceList = dbProduceOverlapInst(cellDb cellDb->bBox)
6    foreach( obj instanceList
7    newx = TO_GRID(car(obj->xy)* factor)
8    newy = TO_GRID(car(last(obj->xy))* factor)
9    obj->xy = list(newx newy)
10   cell_hit = exists(xy cell_list (xy= = obj->cellName) )
11   if(cell_hit= = nil
12    then
13     cell_list = cons( obj->cellName cell_list )
14   unflatten = exists(xy unscaleList (xy= = obj->cellName) )
15   if(unflatten= = nil && hier= = 1
16    then
17      child_cellDb= dbOpenCellView(libnameobj->cellName
18                    "layout" "" "a")
19      CELL_RECURE(child_cellDb hier)
20      dbSave(child_cellDb libname obj->cellName "layout")
21      dbClose(child_cellDb)
22     );;endif
23    );;endif
24   );;end foreach
25  );;endprog
26 );;endprocedure
```

算法 1 给出了层次化版图缩放程序的算法, 其中调用到的函数将在后面说明。算法第四行首先调用函数(详见算法 2)处理当前层次中的图形。算法第五行获取当前层次中的实例, 算法第六行到第二十四行分别处理每一个实例。其中, 第七、八、九行实现实例的移位; 第十行用于判断当前实例是否已处理过, 如果没有则读取实例并且调用递归函数处理此实例(第十七到十九行)。当所有层次实例被处理完毕后算法中止。

算法 2: 图形处理算法

```
procedure( SCALE_FIGURE_CENTER( objfactor)
  prog( (xy_sel oldPoints newPoints oldBox newBox xy x))
    if( (obj->objType)= = "polygon")
      then ;;处理多边形图形)
    else if( (obj->objType)= = "rect")
        then ;;处理矩形图形)
      else if( (obj->objType)= = "path")
          ....)
        else)
          printf("Warning: other figures occurs! \n"))
  ....
```

考虑到处理程序的灵活性和通用性, 采用建立函数库的方法来组织 SKILL 程序。一方面可以针对不同的设计调用库函数来编写具体的程序; 另一方面可以不断的扩充函数库以扩展功能。根据函数的通用性我们建立两个函数库: BASIC_LIB 和 MACRO_LIB。表 3 列出了两个函数库中的部分函数。

表 3　部分函数列表

函数库	函数名称
BASIC_LIB	SCALE_FIGURE_CENTER()
	ADJUST_VIA()
	ADJUST_with_MIDLINE()
	MOVE_OBJ()
	MOVE_POINTS()
	TO_GRID()
	EXTEND_PATH()

续表 3

函数库	函数名称
MACRO_LIB	PROCESS_CELL()
	CELL_RECURE()
	PRO_FIGURE_CENTER()
	COVER_VIA()
	FIX_VIA_AREA()

BASIC_LIB 库中包含对版图中对象进行基本处理的函数: 包括对矩形、多边形和路径进行缩放, 对实例和文本(label) 进行移动的函数以及校正通孔大小和格点(grid) 处理的函数等等。

MACRO_LIB 库中包含了较高层次的用于完成特定任务的用户函数: 包括读取设计库函数, 处理层次化单元的递归函数以及修正覆盖通孔金属面积函数等等。这些用户函数一般构建在 BASIC_LIB 库函数基础之上。

3 DRC(Design Rule Check) 错误修正实例

版图缩放程序处理后的版图可能存在很多 DRC 错误需要作进一步的处理。分析可以发现这些错误有很大的重复性, 根据每一类错误特点编写具有针对性程序可以有效的改正绝大多数 DRC 错误。下面给出一些运用 SKILL 程序处理版图 DRC 错误的实例。

图 4 给出了由于尺寸缩小后距离 d 违反金属最小间距 DRC 规则的例子。移动 VIA1 或 VIA2 需要做的修改太大, 一个可行的方法是扩展金属连线使其覆盖住两个孔之间的空隙。首先使用程序找到所有的 VIA1, 再找出和每个 VIA1 交迭的金属连线, 然后判断金属连线的走向(图例中是沿 y 轴方向) , 接着比较 VIA1 和金属连线的沿垂直连线走向(x 轴) 的边界坐标值, 取其中的最大值分别作为金属连线沿垂直连线走向(x 轴) 的坐标值。在图 4 中只要把 VIA1 左边沿的 x 轴坐标赋值给连线的左边沿 x 轴坐标即可。同理, 如果金属连线沿 x 轴方向, 则改变连线的 y 轴坐标。函数 COVER_VIA( ) 用于实现这一功能。

在图 5 的例子中, 多晶栅两端延伸出扩散区的长度 d 太小, 不满足 DRC 规则。多晶栅一般是用路径来画的, 只要使多晶路径向两端延伸就可以修复这个 DRC 错误。其中的关键是分别判断路径两端的走向(上、下、左、右四个方向) 。利用路径点序列中前两个点和倒数后两个点可以分别判断出路径首尾两端的走向, 然后对点序列中第一个和最后一个点的坐标做相应处理。图 5 中多晶路径首端走向为左, 需要减小第一个点的 x 坐标; 尾端走向为上, 需要增加最后一个点的 y 坐标。函数 EXTEND_PATH( ) 用于实现这一功能。

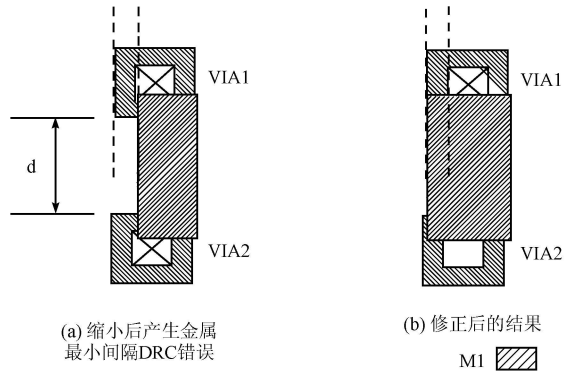


图 4 用 SKILL 程序修正金属最小间隔错误

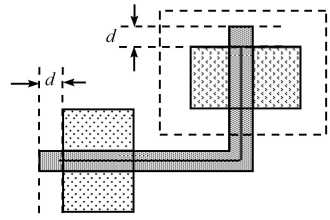


图 5 多晶栅两端延伸出扩散区太小错误

如何处理 SRAM 版图是我们遇到的一个难题。SRAM 结构的特点决定了其版图密度高并且结构紧凑。分析 SRAM 版图发现各行/ 列之间的间隔并不均匀, 同时整个 SRAM 版图四周边缘还有空余可以通过重新放置行列来解决 DRC 错误。首先使用 SKILL 程序计算出各行/ 列之间的距离, 然后根据 DRC 规则算出各行/ 列的修正量、重新放置各行和列使得各行和列之间距离最优化。要注意的是 SRAM 顶层的连线和通孔也要相应的移位和缩放以保证逻辑功能的正确性。这样处理后的版图不但符合 DRC 规则在结构上也更加均匀和紧凑, 同时也没有增加面积。

格点由制造工艺的辨析度决定, 图形中点的坐标都要是格点辨析度的整数倍, 即在格点上。要对缩放后版图中每个点做格点处理使其定位于最近的格点上。如果格点辨析度为 0. 005, 则各个坐标点的千分位要做如图 6 的处理。例如点(2. 513, 2. 709) 格点处理后为(2. 515, 2. 710) 。部分实现程序如下。

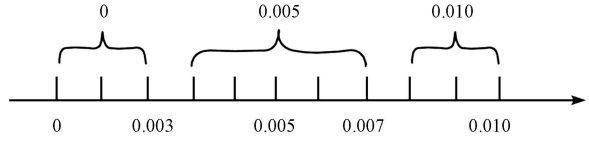


图 6 格点处理区间划分

;; 计算坐标点的千分位

```
    judge = round(point* 100.00) - point* 100.00
;;判断变量 judge 是否在(0,0.3]区间并决定输出值
    if( ((judge< 0.3) && (judge> 0)) ||
      (abs(judge-0.3)/judge <= 1e-6)
     then
      xout = round(point* 100.00)
     );;endif
```

由于 SKILL 语言使用双精度的浮点数,只有都赋值同一常数或者计算产生过程相同的两个浮点数才能确切相等,所以判断两浮点数是否相等时不能用 A == B 而要判断(abs(A-B)/A <= 1e-6)。TO_GRID()函数用于实现这一功能。

4　版图处理流程

要完成原始版图设计到新工艺的移植需要一个完整的处理流程以确保版图 DRC 和 LVS 的正确性。图 7 给出了一个处理流程。可以看到,在全局自动缩放和局部自动修改阶段可以使用 SKILL 程序。在做完全局的缩放处理后针对某一类 DRC 错误用 SKILL 编程进行修改,在达到可以接受的 DRC 错误数量后再进行手动修改。在做 LVS 时需要更新的原理图或者网表文件。有两种处理网表的方法:一种是使用 SKILL 程序按比例缩放原理图中器件的参数然后导出新的网表文件;一种是直接在原来的网表文件中设置比例因子(SCALE),LVS 工具(如 Calibre)会自动的缩放网表中器件的参数。实践中我们使用了后一种方法,快速而简单。

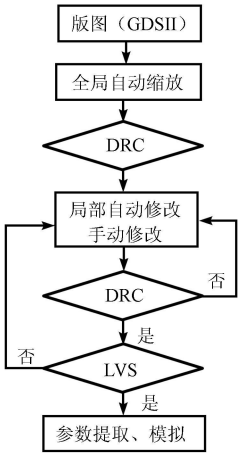


图 7　处理流程

5　结果分析与比较

比较于手工修改或者普通算法,使用递归算法程序自动缩放和修改版图极大的缩短了设计时间,达到了减小硬核面积和提高性能的工程目标。比较缩放前后的硬核版图,电容平均缩小了 33%,标准单元(反向器)门延时减小 40%,硬核面积由原来的 2 400 μm* 2456 μm 缩小到 1 723 μm* 1 787 μm。晶体管级仿真结果表明缩放后硬核功能正确,见表 4、表 5。

表 4　固有电容提取参数

固有电容	尺寸缩小前	尺寸缩小后	缩小百分比
C_{vdd}	0.339968	0.230209	32.29%
C_{gnd}	0.341253	0.228277	33.11%
C_a	0.624916	0.417257	33.23%
C_y	0.311272	0.206919	33.52%

表 5　反向器结构环形震荡电路仿真结果

仿真参数	尺寸缩小前	尺寸缩小后
环形震荡电路工作电压/V	1.8	1.8
环形震荡电路平均电流/μA	370	400
环形震荡电路周期/ns	0.5	0.3
反向器门延时/ns	0.05	0.03

6　总结

本文介绍的方法极大的提高了设计效率,但是现在的方法也存在着不足之处:首先,不能完全的杜绝 DRC 错误。标准的逻辑单元由于简单可以很好的被处理,但是对于复杂的特别是全定制的单元(例如 SRAM)就需要为其编写特别的 SKILL 处理程序同时加上手动的修改;其次,SKILL 程序中用到的多数参数还不能参考 DRC 规则和结果自动修改,这也导致了处理流程中反复的 DRC 迭代。今后的工作就是使程序更加智能化能够参照 DRC 规则文件自动的调整参数来修改 DRC 错误,减少手动修改的工作量。

参考文献:

[1] Cadence, Cadence Design Framework II SKILL Functions Reference.

[2] Cadence, SKILL Language User Guide[J].

[3] Mentor Graphics, SVRF Capacitance and Resistance Statements-Concepts and Usage[S], March 2003.

[4] Mentor Graphics, Calibre Verification User's Manual[S], March 2003.

[5] Rochit Rajsuman, System-on-a-Chip: Design and Test[M],北京航空航天大学出版社.

[6] Jan M. Rabaey, Digital Integrated Circuits-A Design Perspective[M]. 清华大学出版社.

[7] Wayne Wolf. Modern VLSI Design Systems on Silicon[M]. 科学出版社.

[8] 朱正涌. 半导体集成电路[M]. 清华大学出版社.