АКАДЕМИЯ НАУК СССР
СИБИРСКОЕ ОТДЕЛЕНИЕ
А В Т О М Е Т Р И Я
№ 3 1991

АВТОМАТИЗИРОВАННОЕ ПРОЕКТИРОВАНИЕ

УДК 621.38 : 519.87

З. А. ЛИВШИЦ, Д. Г. ТИТОВ
(Новосибирск)

АЛГОРИТМЫ РАБОТЫ С ТАЙЛОВЫМИ ПРЕДСТАВЛЕНИЯМИ ТОПОЛОГИИ СБИС

В процессе автоматизированного проектирования интегральных схем топологическая* информация (т. е. данные, описывающие геометрию масок) обрабатывается большим количеством разнообразных программ, таких, например, как редактор топологии, средства для проверки конструктивно-технологических и электрических ограничений, экстракции из топологии принципиальной электрической схемы, трассировки межсоединений, компактизации топологии и т. п. Естественным требованием к системе проектирования является наличие единой базы топологических данных, т. е. унификация представления информации, с которой оперируют перечисленные выше программы. При выборе такого представления необходимо иметь в виду то обстоятельство, что объемы данных, обрабатываемых топологическими алгоритмами, в практических задачах весьма велики: топология схемы, содержащей $\sim 10^k$ транзисторов, состоит из $\sim 10^{k+1}$ полигонов.

Наиболее распространенными в настоящее время структурами данных, применяемыми для описания топологии БИС, являются различного рода линейные списки образующих топологию геометрических объектов. Подобные структуры весьма удобны для реализации операций включения в топологию или уничтожения некоторого объекта, широко используемых в графических редакторах. В то же время органический недостаток таких структур заключается в том, что расположение объекта в списке никак не связано с его «географическим» размещением на «плоскости кристалла». Это крайне существенно, так как преобладающими в программах обработки топологической информации являются локальные операции, базирующиеся на анализе взаимодействия между «географически близкими» элементами топологии. Поэтому алгоритмы, работающие с линейными списками, вынуждены включать сортировку объектов и их временная сложность не может иметь порядок, меньший, чем $O(N \log N)$, где N — число объектов, что неприемлемо для программных средств «массового применения».

Отмеченные принципы стимулировали исследование альтернативных подходов к представлению двумерных геометрий. Наиболее известные варианты решения этой проблемы — многомерные бинарные деревья (k-d-деревья) [1], деревья квадрантов [2] и структуры тайлового типа [3]. Общим для всех этих подходов является «размен» памяти на скорость, т. е. применение при кодировании топологической информации некоторой избыточности, благодаря чему повышается быстродействие ее обработки.

* В данной статье мы употребляем не вполне удачный, но укоренившийся в отечественной литературе термин «топология» в качестве эквивалента английского слова layout.

В [4—6] проведено подробное изучение структур данных на основе различных модификаций k-d-деревьев и деревьев квадрантов применительно к задачам, возникающим при автоматизированном проектировании СБИС. Выяснилось, в частности, что подобные структуры хорошо приспособлены для перечисления всех объектов, находящихся в заданной прямоугольной области — одной из наиболее распространенных процедур обработки топологии. К сожалению, однако, на этих структурах недостаточно эффективно реализуется другая (не менее важная) процедура — поиска ближайшего соседа объекта: ее сложность логарифмически зависит от общего числа объектов.

Кардинальное решение «проблемы соседства» предложено Дж. Остуру. Так, для избежания трудностей работы с данными со сшиванием метрическими объектами данные определенным образом разносятся по нескольким плоскостям и канонизируются. Для того чтобы соседи существовали у каждого из элементов структуры, данные «симметризуются», т. е. равноправными объектами в такой структуре являются как прямоугольники, представляющие физические или логические слои СБИС, так и «пустые» прямоугольники.

В [3] показано, что сложность основных процедур, оперирующих над тайловыми структурами, не превосходит линейной, а в [7] описана система «Magic»— мощный интегрированный комплекс программных инструментов топологического проектирования, работающих над единой «тайловой» базой данных. «Magic» по своим характеристикам на порядок превосходит коммерческие пакеты аналогичного назначения. В более поздней разработке [8] сняты ограничения, связанные с использованием в [3, 7] лишь топологии манхэттенского типа (т. е. образованной прямыми, параллельными осям координат): система «Tailor», основанная на трапезоидных структурах со сшиванием углов, обеспечивает работу с топологиями, допускающими все углы, кратные 45°.

Концепция тайловой организации данных применяется и в проводимой в Институте автоматики и электрометрии СО АН СССР разработке системы топологического проектирования, содержащей встроенные средства оптимизации и верификации топологии. В данной статье представлена часть этой работы, связанная с реализацией основных алгоритмов над структурами со сшиванием углов. Тщательности отработки этих алгоритмов придавалось большое значение, так как они используются, по существу, всеми программами системы; поэтому от их качества в значительной мере зависят параметры системы в целом.

Следует отметить, что ряд приведенных ниже алгоритмов существенно отличается от предложенных Остерхаутом в [3]. Дело в том, что объем памяти является критичной характеристикой при работе с тайловыми структурами; поэтому всюду, где это возможно, мы избегали рекурсивных конструкций, которые активно применялись в [3].

1. Тайловые структуры: основные понятия. В этом разделе приведем формальные определения, необходимые для дальнейшего, и краткие комментарии.

Топология (кристалла, ячейки) представляется совокупностью *тайловых плоскостей*. Каждая тайловая плоскость образована непересекающимися прямоугольниками — *тайлами различных типов*. Каждая точка тайловой плоскости принадлежит одному и только одному тайлу: для определенности принимается, что тайл содержит свою левую и нижнюю границы.

(Содержательно тип тайла соответствует принадлежности топологического объекта некоторому физическому (например, поликремний, ме-

талл, диффузия) или символическому (например, транзистор, образованный пересечением поликремния и диффузии) слою; как уже отмечалось, в качестве самостоятельного типа используется тип «space», присваиваемый «пустым» тайлам, не относящимся ни к физическим, ни к символическим слоям. Разбиение тайловой плоскости на непересекающиеся прямоугольники возможно за счет того, что «невзаимодействующие типы», т. е. такие, что точка с координатами x, y может одновременно принадлежать им обоим (например, металл и диффузия), разносятся по различным плоскостям.)

Структура тайловой плоскости удовлетворяет двум условиям канонизации: тайлы одного типа не должны иметь общих вертикальных границ, т. е. каждый тайл должен вытягиваться в ширину, насколько это возможно; тайлы одного типа, имеющие совпадающую горизонтальную границу, должны быть слиты в один.

(Канонизация тайловой плоскости, во-первых, гарантирует единственность представления любой геометрии независимо от порядка, в котором она создавалась, и, во-вторых, позволяет уменьшить фрагментацию структуры, т. е. снизить избыточные затраты памяти. Эксперименты с «практическими» топологиями показывают, что для них отношение числа тайлов типа «space» к общему числу тайлов $\sim 1/2$.)

В тайловой плоскости тайлы организованы в четырехсвязный список. С каждым тайлом ассоциируется следующая информация:

— тип тайла;

— точка привязки (x, y) — координаты левого нижнего угла прямоугольника;

— четыре указателя (rt, tr, bl, lb) — соответственно на тайл, являющийся самым правым среди верхних соседей, самым верхним среди правых, самым нижним среди левых, самым левым среди нижних.

(Рис. 1 наглядно демонстрирует происхождение названия «сшивание углов».)

Для обеспечения корректности описания необходимы дополнительные соглашения об устройстве тайловой плоскости «на бесконечности» (по этому поводу см. п. 2.1).

Приведем простой пример, иллюстрирующий операции над тайловой структурой данных: вычислим размеры sx и sy некоторого тайла t. При этом будем (как и всюду в этой статье) использовать синтаксис языка C:

$$(t \text{ — указатель на тайл})$$
$$sx = (t \to tr \to x) - (t \to x)$$
$$sy = (t \to rt \to y) - (t \to y)$$

2. Базовые алгоритмы на тайловых структурах. 2.1. *Процедура MakePlane — инициализация тайловой плоскости.* Все описанные ниже алгоритмы работают на инициализированных тайловых плоскостях. Это обеспечивает определенность в любой момент времени всех ссылок на соседей и избавляет от необходимости включать в каждый алгоритм анализ граничных условий.

Процедура MakePlane использует понятие максимальных рабочих координат — Max X и Max Y; она создает тайл типа «space» с координатами левого нижнего и правого верхнего углов — $(-\text{Max } X, -\text{Max } Y)$ и $(\text{Max } X, \text{Max } Y)$ соответственно. Этот тайл окружается тайлами специального типа «border», у которых не определяются ссылки «наружу» полученной таким образом тайловой структуры (рис. 2).

После проведения описанной инициализации алгоритмы, оперирующие в рабочем пространстве, гарантированы от осложнений, связанных с граничными условиями.

2.2. *Процедура InWhich — поиск тайла, содержащего заданную точку.* Это самый простой из базовых алгоритмов над тайловыми структурами, и мы опишем его явно.

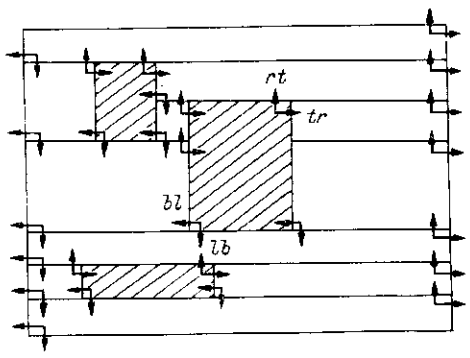

Рис. 1. Пример тайловой структуры

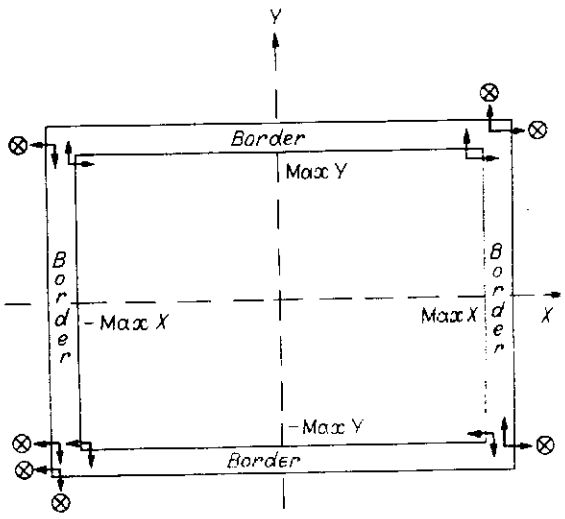

Рис. 2. Инициализированная тайловая плоскость:
⊗ — указатель «наружу»

Проверка принадлежности точки (X, Y) тайлу t осуществляется вычислением выражения

(точка (X, Y) принадлежит тайлу t) $\equiv (t \rightarrow x <= X)$ AND $(t \rightarrow y <= Y)$ AND $(t \rightarrow tr \rightarrow x > X)$ AND $(t \rightarrow rt \rightarrow y > Y)$.

Алгоритм поиска нужного тайла может быть тогда сформулирован следующим образом.

Пусть t — указатель на некоторый тайл:

loop
  while $(t \rightarrow x > X)\, t = t \rightarrow bl$
  while $(t \rightarrow y > Y)\, t = t \rightarrow lb$
  if (точка (X, Y) принадлежит тайлу t) break
  while $(t \rightarrow tr \rightarrow x <= X)\, t = t \rightarrow tr$
  while $(t \rightarrow rt \rightarrow y <= Y)\, t = t \rightarrow rt$
end_loop

Работа алгоритма иллюстрируется рис. 3.

Нетрудно показать (см., например, [8]), что приведенный алгоритм всегда сходится к требуемому решению (т. е. не приводит к циклам). При обычных в таких задачах предположениях о равномерности двумерного распределения тайлов по рабочему пространству, случайном выборе точки (X, Y) и начального тайла вычислительная сложность алгоритма, пропорциональная среднему числу перебираемых в процессе поиска тайлов, составляет $O(\sqrt{N})$, где N — общее число тайлов в тайловой плоскости. Однако практически в силу локальности большинства операций над топологиями поиск может быть значительно ускорен за счет использования в качестве начального приближения тайла, найденного в результате предыдущего применения процедуры InWhich.

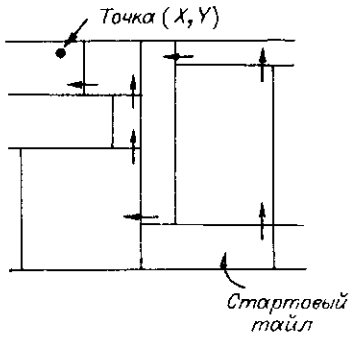

Рис. 3. Алгоритм поиска тайла ($\rightarrow$ — путь обхода тайлов при поиске тайла, содержащего точку (X, Y))

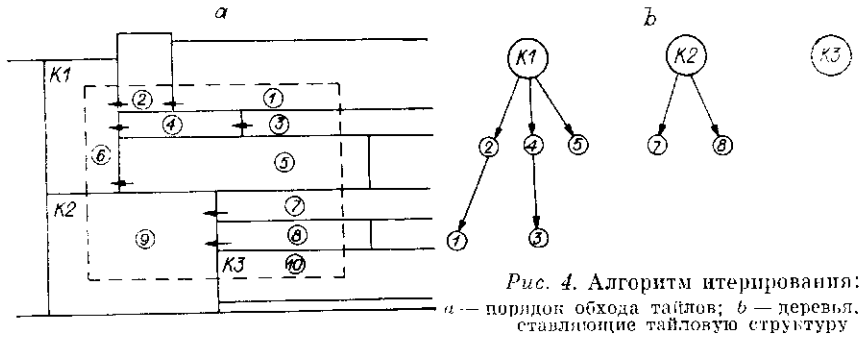

Рис. 4. Алгоритм итерирования:
a — порядок обхода тайлов; *b* — деревья, представляющие тайловую структуру

2.3. *Процедура Iter — итерирование тайлов в прямоугольной области.* Задача, решаемая процедурой Iter, состоит в следующем. Пусть на некоторой тайловой плоскости задана прямоугольная область с координатами диагональных вершин (*X*1, *Y*1) и (*X*2, *Y*2). Задана также некоторая процедура (экзекутор), которая имеет входным параметром тайл и может производить с ним некие действия (например, отображать его на экране). Требуется обойти все тайлы, пересечение которых с заданной областью непусто, и передать все найденные тайлы экзекутору (причем каждый тайл должен быть передан ровно один раз). При этом необходимо обеспечить такой порядок обхода, чтобы алгоритм не «ломался», даже если действия экзекутора связаны с перестройкой тайловой структуры (например, если он делит полученный тайл пополам). Другими словами, итератор, передав тайл экзекутору, уже не должен пользоваться никакой информацией об этом тайле (в том числе значениями его ссылок *tl*, *tr*, *bl*, *lb*). Алгоритмы такого назначения, о которых упоминается в литературе ([3], [8]), существенно рекурсивны, что не всегда приемлемо с точки зрения экономии памяти.

Ниже описывается нерекурсивный алгоритм, идея которого состоит в отображении множества тайлов, пересекающихся с заданной прямоугольной областью, на множество двусвязных деревьев.

Мы будем использовать в этом разделе следующую терминологию. Если T — тайл, определяемый указателем *bl* тайла t, то T называется отцом t, а t — сыном T. Таким образом, каждый тайл может иметь несколько (в том числе ни одного) сыновей и единственного отца. Потомками тайла являются все тайлы, получаемые в результате последовательного определения его сыновей, сыновей сыновей и т. д. Будем называть далее законным сыном тайла t такого сына t, точка привязки которого находится внутри заданной прямоугольной области; аналогично приведенному выше определяется понятие законного потомка.

Таким образом, с каждым тайлом может быть связано дерево его законных потомков, причем если ни один из тайлов t_1 и t_2 не является потомком другого, то соответствующие деревья не пересекаются.

Нетрудно убедиться, что множество тайлов, имеющих непустое пересечение с областью итерирования, однозначно отображается на множество вершин деревьев законных потомков, корнями которых являются тайлы, пересекаемые левой или нижней границей области (рис. 4). Действительно, все корневые тайлы не являются законными потомками друг друга, поэтому множества их законных потомков не пересекаются; для каждого внутреннего тайла (т. е. тайла с точкой привязки внутри области итерирования) может быть указан единственный путь (по указателям *bl*), ведущий к границе области и, следовательно, к корневому тайлу.

В терминах деревьев правило обхода, гарантирующее, что каждый тайл области будет передан экзекутору ровно один раз, формулируется просто: любой тайл должен посещаться после того, как посещены все его потомки. Пример итерирования области на основе этого правила приведен на рис. 4, где порядок обхода обозначен цифрами в кружках.

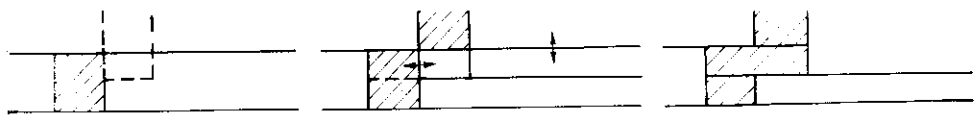

Рис. 5. Применение процедуры ModRect при создании нового тайла. Изображение тайловой структуры:

a — до создания очередного тайла (— — — создаваемый тайл); *b* — после разрезания и смены типов до канонизации (↔ — тайлы должны быть слиты, — — — — тайл должен быть разрезан); *c* — после канонизации

Ясно, что процедура Iter оперирует с каждым тайлом дважды: при определении его потомков и при «обходе дерева». Таким образом, сложность этой процедуры (без учета предварительного шага — нахождения первого корневого тайла) пропорциональна числу n тайлов, имеющих непустое пересечение с областью итерирования.

2.4. *Процедура ModRect — модификация тайловой структуры.* Эта процедура обеспечивает перестройку тайловой структуры, вызванную наложением на существующую геометрию прямоугольника заданного типа. При таком наложении точки, попавшие внутрь прямоугольника, меняют тип в соответствии с заданной функцией перекрытия (которая вычисляет результирующий тип в зависимости от существовавшего типа и типа прямоугольника). В результате некоторые тайлы старой структуры изменили тип, некоторые оказались разрезанными границами прямоугольника; все это, а также применение правил канонизации к получившейся геометрии требует преобразования списка тайлов.

Предлагаемый для этих целей алгоритм использует два прохода итератора по области наложенного прямоугольника — основной и вспомогательный.

Во время основного прохода каждый тайл, имеющий непустое пересечение с прямоугольником, разрезается (если необходимо) вдоль границы прямоугольника, в соответствии с функцией перекрытия меняется тип пересечения тайла с прямоугольником, а затем осуществляется канонизация модифицированной геометрии. С канонизацией связана существенная тонкость: если тайл будет слит с соседом, который еще не посещался итератором, то это может привести к ошибочным окончательным результатам.

Чтобы избежать такого рода трудностей, используется вспомогательный проход, проводимый перед основным, на котором все тайлы заданной прямоугольной области помечаются как «несливаемые». На основном проходе пометки с уже обойденных тайлов снимаются. Такая конструкция обеспечивает перестройку тайловой структуры в результате канонизации только в тех местах, где итератор уже побывал, что гарантирует ее корректность.

Из приведенного описания следует, что вычислительная сложность алгоритма пропорциональна n — числу тайлов, имеющих с прямоугольником непустое пересечение.

Иллюстрация действия алгоритма — на рис. 5. Процедура ModRect является необходимой, в частности, для реализации распространенных в графических редакторах топологии операторов типа PAINT и ERASE.

3. Реализация пакета. Показатели производительности. Комплекс программ, поддерживающий все основные операции над тайловыми структурами, реализован на языке C. Эти процедуры используются в ряде инструментальных средств для работы с топологией, разработанных и разрабатываемых в ИАиЭ СО АН СССР.

Мы не будем приводить в этой статье результаты измерений производительности каждого из описанных алгоритмов на «модельных» приме-

Схема	Количество транзисторов	Число тайлов в раскрытой топологии	Время создания тайловой структуры, с	Объем памяти, Мбайт
ALU 16	1 260	13 100	12	0,35
FIFO 8K	32 500	555 200	960	14,8
SER-LINK	2 200	37 700	44	0,98
ADRGEN	8 600	227 000	340	6,05

рах с генерацией «случайных» топологий. Вместо этого приведем характеристики затрат времени и памяти в некоторой реальной задаче, при решении которой все описанные выше процедуры применяются многократно.

Речь идет о преобразовании иерархического (нераскрытого) описания топологии СБИС (или фрагмента СБИС) в формате CIF [9] в плоское тайловое представление.

Измерения проводились на схемах, разработанных в нашей лаборатории; результаты сведены в таблицу. Времена приведены к производительности компьютера VAX-11/780.

Заключение. Рассмотренные в этой статье алгоритмы и реализованные на их основе программы, видимо, сравнимы по эффективности с описанными в [3, 7]. О практической же полезности «работы с тайлами» свидетельствует тот факт, что, как показывает опыт, при проверке конструктивно-технологических ограничений, например, использование программы, оперирующей с тайловыми структурами, позволяет в несколько раз уменьшить временные затраты (по сравнению с известными нам коммерческими продуктами), даже если топология разрабатывалась на базе традиционного редактора и требуется ее предварительное преобразование в тайловое.

СПИСОК ЛИТЕРАТУРЫ

1. **Bentley J. L.** Multidimensional binary search trees used for associative searching // Comm. ACM.— 1975.— 18, N 9.— P. 509.
2. **Finkel R. A., Bentley J. L.** Quad-trees: A data structure for retrieval on composite keys // Acta Informatica.— 1974.— 4.— P. 1.
3. **Ousterhout J. K.** Corner-stitching: A data-structuring technique for VLSI layout tools // IEEE Trans. on CAD.— 1984.— **CAD-3**, N 1.— P. 87.
4. **Rosenberg J. B.** Geographical data structures compared: A study of data structures supporting region queries // IEEE Trans. on CAD.— 1985.— **CAD-4**, N 1.— P. 53.
5. **Brown R. L.** Multiple storage quad trees: A simpler faster alternative to bisector list quad trees // IEEE Trans. on CAD.— 1986.— **CAD-5**, N 7.— P. 413.
6. **Pitaksanonkul A., Thanawastien S., Lursinsap C.** Comparisons on quad trees and 4-d trees: New results // IEEE Trans. on CAD.— 1989.— **CAD-8**, N 11.— P. 1157.
7. **Ousterhout J. K. et al.** The Magic VLSI layout system // IEEE. J. Des. and Test of Comp.— 1985.— 2, N 1.— P. 19.
8. **Marple D., Smulder M., Hegen H.** Tailor: A layout system based on trapezoidal corner stitching // IEEE Trans. on CAD.— 1990.— **CAD-9**, N 1.— P. 66.
9. **Mead C., Conway L.** Introduction to VLSI systems.— Addison-Wesley, 1980.

Поступила в редакцию 10 декабря 1990 г.