РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
АВТОМЕТРИЯ
И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКЕ

УДК 621.38 : 519.87

Ю. А. Пичуева
(Новосибирск)

АЛГОРИТМ ВЕТВЛЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ СВЕРТЫВАНИЯ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ МАТРИЦ

Представлен алгоритм свертывания программируемых логических матриц по столбцам и строкам, позволяющий уменьшить временные затраты путем сокращения дерева перебора. Приведены сведения о его программной реализации, предусматривающей возможность генерации топологии с учетом ограничений, возникающих при проектировании БИС.

Введение. *Свертывание программируемых логических матриц (ПЛМ). Основные виды свертывания.* ПЛМ широко используются для реализации комбинационной логики при проектировании БИС [1]. Основным недостатком такой реализации является низкая плотность транзисторов в плоскостях ПЛМ. Эффективность использования площади кристалла, занимаемой программируемой логической матрицей, может быть повышена за счет ее свертывания, при котором два входных (или два выходных) сигнала располагаются в одном столбце матрицы. При этом один из них подводится снизу, другой — сверху, а строки (термы) должны быть переставлены таким образом, чтобы свернутая ПЛМ реализовывала ту же функцию, что и исходная (рис. 1). Такой вид свертывания называется свертыванием по столбцам.

Аналогичным образом можно произвести свертывание по строкам. При этом одна из плоскостей матрицы делится на две части и располагается по обе стороны от другой плоскости. При конфигурации AND—OR—AND входные, а при конфигурации OR—AND—OR выходные сигналы делятся на два непересекающихся множества (рис. 2).

Свертывание по столбцам и по строкам можно совершать одновременно [2] или последовательно (рис. 3) [3].

Существует понятие сложного свертывания [4], при котором разрешается размещать в одном столбце не два, а несколько сигналов. При этом возникают

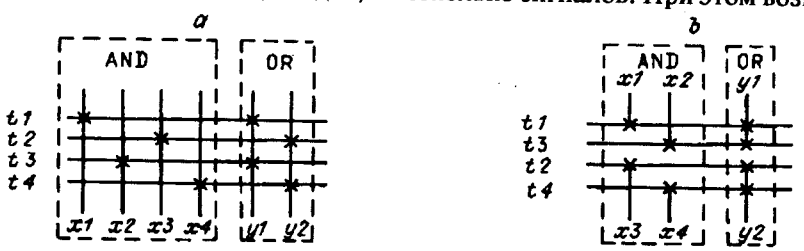

Рис. 1. Свертывание по столбцам:
a — исходная матрица (*x* — входные, *y* — выходные сигналы, *t* — термы), *b* — матрица, свернутая по столбцам

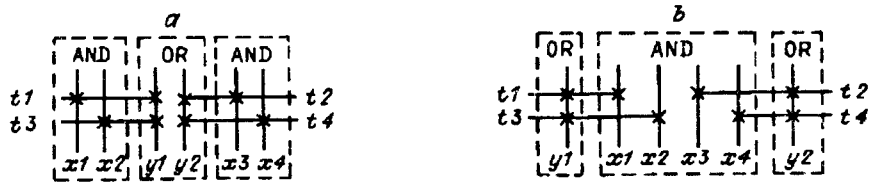

Рис. 2. Свертывание по строкам:
a — конфигурация AND – OR – AND, *b* — конфигурация OR – AND – OR

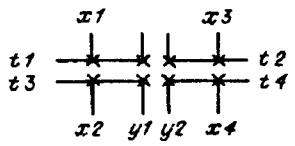

Рис. 3. Исходная матрица, свернутая по строкам, а затем по столбцам

дополнительные проблемы, связанные с разводкой сигналов. В данной работе речь пойдет только о простом свертывании.

Постановка задачи. Сформулируем задачу свертывания ПЛМ более строго. Для простоты ограничимся свертыванием по столбцам.

Пусть задано множество $\{V1, ..., Vn\}$ вертикальных и множество $\{t1, ..., tm\}$ горизонтальных сигнальных линий, которые будем называть *входами* (или переменными) и *термами* соответственно. Каждый из входов может быть подсоединен к некоторым (или ко всем) термам. При этом если вход Vi подсоединен к терму tj, будем говорить, что переменная Vi *входит* в терм tj или терм tj *зависит* от Vi.

Для каждой переменной Vi обозначим $R(Vi)$ — множество всех термов, зависящих от Vi. Заметим, что два входных сигнала Vi и Vj могут разделять один столбец (при некоторой перестановке термов) тогда и только тогда, когда $R(Vi) \cap R(Vj) = 0$. В этом случае будем считать, что переменные Vi и Vj *сворачиваемы*, а упорядоченную пару (Vi, Vj) будем называть *сверткой* (Vi подводится сверху, Vj — снизу). Очевидно, что совершение такой свертки накладывает ограничения на порядок расположения термов: все термы, принадлежащие множеству $R(Vi)$, должны быть расположены выше термов, принадлежащих множеству $R(Vj)$. Запишем это как $R(Vi) > R(Vj)$. Пусть теперь имеем множество сверток $\psi = \{(Vi, Vj)_k\}$, $k = 1, ..., l$. Если существует такой порядок расположения термов, который удовлетворяет всем ограничениям, т. е. $R(Vi) > R(Vj)$ для любой свертки из ψ, то такое множество назовем *совместимым*.

Таким образом, задача свертывания ПЛМ заключается в нахождении множества совместимых сверток с максимальным числом элементов. Она тесно связана с одной из центральных задач теории графов — поиском гамильтонова пути в графе [5, 6]. Учитывая тот факт, что данная задача принадлежит к классу *NP*-полных (или труднорешаемых) [7], особый интерес представляет разработка эффективных алгоритмов нахождения решения, близкого к оптимальному. Одним из методов решения *NP*-полных задач являются алгоритмы с возвратом [5], к которым и принадлежит приводимый в данной работе алгоритм.

Существующие подходы. В настоящее время имеется несколько алгоритмов свертывания. Одни из них используют интерактивный подход, другие полностью автоматизированы. Интерактивные алгоритмы заключаются в том, что пользователь сам выбирает, какие столбцы или строки нужно свернуть, а затем программа совершает свертку этих столбцов или строчек и выдает результат пользователю для следующей итерации [8]. Так как этот метод не требует перебора многих вариантов, то он затрачивает гораздо меньше процессорного времени, чем автоматизированные (около минуты для средних ПЛМ). При этом общее время работы примерно в 10 раз (для больших ПЛМ существенно больше) превышает процессорное за счет времени, затрачиваемого пользователем. Это время, как и результат, сильно зависит от

умения и опыта человека. Полученные результаты находятся в широком диапазоне — от 20 до 55 % экономии площади.

Полностью автоматизированные алгоритмы делятся на эвристические и алгоритмы ветвления [6]. Эвристические алгоритмы последовательно выбирают «лучшую» свертку, совместимую с уже выбранными [3, 9, 10]. Понятие «лучшая» может означать, например, малое количество ограничений, накладываемых этой сверткой на порядок термов, или какой-либо другой критерий. Эти алгоритмы достаточно быстры, однако не могут гарантировать оптимальности варианта. Результаты лежат в пределах 25—40 % площади [9].

Алгоритмы ветвления [2] перебирают возможные варианты свертывания (часто работа этих алгоритмов иллюстрируется деревом, где варианты решения — это ветви). Когда найден один вариант, допускается возврат к предыдущему шагу (алгоритм с возвратом) и ищется решение по другой ветви. Теоретически оптимальное решение может быть найдено перебором всех вариантов, но для этого требуется очень много времени ($O(n!)$). Поэтому реально в этих алгоритмах используются некие ограничительные функции, позволяющие «обрезать», т. е. не рассматривать некоторые ветви. Алгоритмы ветвления характеризуются большими затратами времени и памяти, но зато в большинстве случаев находят оптимальный вариант.

Описание алгоритма. Рассмотрим последовательные этапы работы алгоритма.

Чтение входной матрицы. Матрица вводится в программу в виде таблицы, состоящей из двух частей. Первая часть (плоскость И) состоит из 0, 1 и « – ». 0 означает, что к данному терму подключен инверсный вход данной переменной, 1 — подключен прямой вход, « – » — отсутствует подключение. Вторая часть матрицы (плоскость ИЛИ) состоит из 0 и 1. 1 означает наличие подключения, т. е. присутствие данного терма как слагаемого в данной выходной переменной, 0 — отсутствие подключения.

При чтении этой таблицы программа раскладывает каждый столбец плоскости И на два, соответствующие прямой и инверсной переменной. Каждая прямая переменная получает четный номер, инверсная — следующий за ним нечетный. Результирующая матрица состоит только из 0 и 1.

Построение графа. Информацию о всех возможных свертках удобно представить в виде списка пар *сворачиваемых* переменных. Назовем этот список LIST.

Для определения совместимости сверток построим граф, в котором вершины ассоциируются с входными переменными ПЛМ [11]. Две вершины соединены между собой ненаправленным ребром в том случае, если две соответствующие переменные *невозможно* расположить в одном столбце (рис. 4). Этот граф удобно хранить в памяти в виде массива GRAF [i] размером N (количество переменных во входной матрице), элементами которого являются списки переменных, соединенных с i-й [5, 6]. Каждая вершина графа имеет свой вес (массив $w[i]$), т. е. количество входящих в нее ребер. При свертке переменных i и j будем соединять эти переменные направленным ребром (от переменной, располагаемой сверху, к переменной, располагаемой снизу).

Совместимость сверток. Чередующийся цикл — это циклический путь в графе, ребра которого являются попеременно направленными и ненаправленными. Докажем следующее утверждение:

Если две свертки несовместимы, то соответствующие им направленные ребра образуют в графе чередующийся цикл.

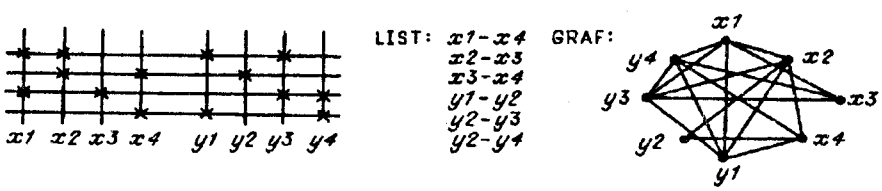

Рис. 4. Матрица и соответствующие ей список пар сворачиваемых переменных и граф

Рис. 5. Чередующийся цикл в графе

Пусть имеем две свертки Vi — Vj и Vk — Vl. Соответствующие ограничения на порядок расположения термов запишутся как

$$R(Vi) > R(Vj)$$
$$R(Vk) > R(Vl)$$

Не существует соответствующего порядка термов в том случае, если эти неравенства противоречат друг другу, т. е. существует некий терм, принадлежащий $R(Vi)$ и $R(Vl)$, и некий терм, принадлежащий $R(Vj)$ и $R(Vk)$. В терминах построенного графа это будет означать, что вершины i и l и вершины j и k соединены ребром. Тогда две совершенные свертки (два соответствующих направленных ребра $i \to j$ и $k \to l$) образуют в графе цикл $i-j-k-l$ (рис. 5). Очевидно, что верно и обратное утверждение. Аналогичные рассуждения можно провести для произвольного количества сверток. На основании доказанного можно сделать следующий вывод:

Для того чтобы n сверток были совместимы, необходимо и достаточно, чтобы n соответствующих направленных ребер не создавали в графе ни одного чередующегося цикла.

На этом утверждении, полное доказательство которого приведено в [11], основан алгоритм свертывания, описанный ниже.

Сортировка списка. Будем последовательно выбирать из списка LIST свертки, совместимые с уже выбранными. Для того чтобы выбрать как можно больше сверток, список нужно предварительно отсортировать. Критерий сортировки списка определяется из следующих соображений. Чтобы возможность появления цикла в графе была минимальной, необходимо, чтобы вершины, являющиеся концами направленных ребер, создавали в графе как можно меньше путей, т. е. имели минимальный вес.

Начала направленных ребер нужно выбирать с максимальным весом, чтобы, во-первых, быстрее свернуть те вершины, которые свернуть трудно, а во-вторых, чтобы не «занимать» вершины с большим весом, которые хорошо использовать в качестве концов.

Таким образом, получаем критерий сортировки списка: для каждой пары переменных Vi и Vj вычисляем разность весов $w[i]-w[j]$ и сортируем список по убыванию этой величины.

Процесс свертывания. Построим теперь алгоритм свертывания. Для этого определим массив FOLD размером $N/2$ (максимальное число сверток). Элементами массива являются списки сверток. Каждый список имеет текущий указатель PTR[k] (рис. 6).

Нулевой элемент массива содержит отсортированный список LIST. Указатель PTR[0] в начальный момент указывает на голову списка. Список FOLD[1] содержит свертки, совместимые с той, на которую в данный момент указывает PTR[0]. Далее элемент массива FOLD[k] вычисляется из предыдущего — FOLD[$k-1$] — путем отбрасывания сверток, которые несовместимы с PTR[0] ... PTR[$k-1$] (т. е. образуют чередующийся цикл в графе). Таким образом, на некоем l-м этапе список обращается в нуль, и мы имеем набор совместимых сверток PTR[0], ..., PTR[l]. Если это решение лучше уже имеющегося, оно запоминается и поиск продолжается: указатель PTR[l] сдвига-

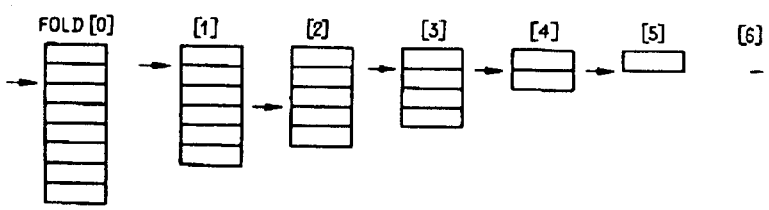

Рис. 6. Процесс свертывания

ется на одну позицию и вычисляется FOLD[l + 1]. Эта процедура повторяется до тех пор, пока PTR[l] не дойдет до конца списка. Затем происходит возврат «назад»: указатель PTR[l − 1] сдвигается вниз по списку FOLD[l − 1] и перевычисляется FOLD[l]. Работа заканчивается, когда указатель PTR[0] дойдет до конца списка.

Очевидно, этот алгоритм находит оптимальное решение (поскольку он перебирает все возможные решения). Время его работы экспоненциально растет с ростом размерности матрицы [5], в то время как эффективным может считаться лишь алгоритм с полиномиальной зависимостью [7]. Поэтому предлагается ввести два параметра, определяющие «глубину» поиска, которые можно менять в зависимости от размерности матрицы. Во-первых, пусть указатели PTR передвигаются не до конца списка, а просматривают первые p ребер. Во-вторых, пусть возврат «назад» совершается не к предыдущему списку, а к списку с номером q.

Учитывая,что списки отсортированы, можно ожидать,что близкое к оптимальному решение будет найдено в начале поиска. Практически установлено,что для средних ПЛМ оптимальными значениями параметров являются $q = 2, p = 3$.

Для упрощения программы и ускорения ее работы создана специальная версия для $q = 2$. Имеются только три списка: head, neck и body. Для фиксированных значений head и neck составляется список body совместимых с ними сверток, и, двигаясь по нему до конца, выбираются свертки, совместимые с уже выбранными. Затем меняем neck и повторяем процесс. Это повторяется p раз, после чего меняется head. Параметр p задается во входном файле. Выбирается лучший из перебранных вариант. Таким образом, существенный эффект сокращения дерева перебора достигается отсеканием ветвей, которые увеличивают вероятность появления цикла в графе и, следовательно, не приводят к оптимальному результату.

Программная реализация алгоритма. *Последовательность свертывания. Особенности второго этапа свертывания.* Поскольку свертывание по столбцам оказывает влияние на свертывание по строкам, и наоборот, последовательность их осуществления имеет большое значение. В программе предусмотрена возможность совершить свертывание в два этапа: по строкам, а затем по столбцам, и наоборот, после чего подсчитать сэкономленную площадь и выбрать лучший вариант. Кроме того, при свертывании по строкам выбирается лучший вариант между конфигурациями OR—AND—OR и AND—OR—AND.

На втором этапе свертывания необходимо учитывать ограничения на порядок расположения строк (или столбцов), внесенные на первом этапе. В работе [3] предлагается это делать следующим образом. Пусть для определения на первом этапе проводилось свертывание по столбцам. Построим направленный граф CONS, в котором вершины ассоциируются с термами, а ребро $ti \to tj$ означает, что терм ti должен располагаться выше терма tj. CONS содержит ограничения, внесенные свертыванием столбцов (рис. 7).

Очевидно, что этот граф ацикличный. Очевидно также,что если существует путь в нем от вершины ti к вершине tj, то термы, соответствующие этим вершинам, нельзя расположить в одной строке. Это означает, что при свертывании строк, кроме проверки графа GRAF на наличие циклов, необходимо осуществлять проверку графа CONS на наличие пути. Это, в свою очередь, означает, что необходимо внести добавление в критерий сортировки списка. Для этого введем два о п р е д е л е н и я:

В направленном графе вершина i называется *предком* вершины j, а j — *потомком* вершины i, если существует путь из i в j.

Посчитаем количество потомков и предков для каждой

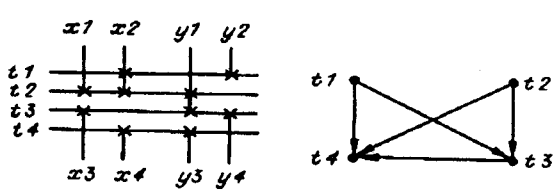

Рис. 7. Свернутая по столбцам матрица и соответствующий граф ограничений

вершины, заведем соответственно массивы anc и dec. Для того чтобы сделать максимальное количество сверток на втором этапе, количество пар термов, находящихся друг с другом в «родственных отношениях», должно быть минимальным.

Осуществление свертки термов *ti* и *tj* означает слияние вершин *ti* и *tj* в графе CONS. Все потомки *ti* становятся потомками *tj*, и наоборот. Все предки *ti* становятся предками *tj*, и наоборот. Следовательно, необходимо минимизировать величину

$$\mathrm{anc}[i]\mathrm{dec}[j] + \mathrm{anc}[j]\mathrm{dec}[i].$$

На втором этапе свертывания эту величину нужно добавить к ранее найденному критерию сортировки.

Реализация. Программа предназначена для проектирования ПЛМ, являющейся частью интегральной схемы, поэтому в ней предусмотрены некоторые средства для удобства проектировщика:

1. В результирующей матрице прямая переменная всегда находится рядом с инверсной. Для этого при свертывании двух входов проверяется, можно ли свернуть соответствующие им инверсные входы. Если нельзя, то они исключаются из дальнейшего рассмотрения. Кроме того, при свертывании по строкам (AND—OR—AND) прямой и инверсный входы всегда расположены по одну сторону от плоскости OR.

2. Применение свернутой ПЛМ в качестве функционального блока реальной топологии может накладывать ряд ограничений на расположение входов и выходов результирующей матрицы. Пользователь может задать во входном

ПЛМ	*I/O*	*T*	*TR*, %	*RC/CR*	STRUC	FOLD	*S*	CPU, с
1	30/20	50	16,8	*RC*	*A*	3/10	75	45,9
2	20/8	15	24,3	*RC*	*O*	3/10	51	11,9
3	32/8	16	18,1	*RC*	*A*	1/15	58	12,5
4	24/6	18	18,0	*CR*	*A*	11/1	59	10,0
5	18/11	20	18,6	*RC*	*A*	2/10	58	8,0
6	22/20	17	20,2	*RC*	*A*	1/13	64	2,1
7	18/13	16	17,7	*RC*	*A*	4/9	53	1,4
8	22/14	23	14,9	*RC*	*O*	5/11	54	23,5
9	26/6	19	17,9	*RC*	*A*	2/12	55	10,5
10	52/44	36	11,7	*CR*	*A*	30/2	64	64,6
11	56/39	179	19,4	*C*	—	22	76	24,4
12	12/64	64	7,9	*CR*	*O*	38/5	46	74,3
13	16/8	61	26,6	*CR*	*A*	5/5	72	30,5
14	16/16	10	18,7	*C*	—	0/16	50	5,6
15	32/4	16	29,2	*C*	—	0/16	55	1,5
16	20/4	46	27,4	*CR*	*A*	7/1	69	7,2

Примечание. I/O — количество входных/выходных переменных матриц, *T* — количество термов, *TR* — плотность транзисторов, *RC/CR* — последовательность свертывания, давшая лучший результат, STRUC — расположение плоскостей: *O* (OR—AND—OR) или *A* (AND—OR—AND), FOLD — число полученных пар строк/столбцов или столбцов/строк, *S* — новая площадь матрицы в процентах к старой, CPU — процессорное время (VAX серии 8000).

файле списки переменных и термов, которые должны находиться сверху или снизу (слева или справа для термов). Эти ограничения обычно не оказывают влияния на эффективность результата. Более сложные ограничения описаны в [4].

3. Можно ограничить работу программы только свертыванием столбцов (или строк) или только свертыванием входов (или выходов).

4. Имеется подпрограмма генерации топологии, позволяющая получить реальную топологию ПЛМ (КМОП или *n*МОП), удовлетворяющую правилам проектирования.

Основные результаты. Описанная программа реализована на языке Си, работает в операционных средах VAX/VMS и UNIX. Основные результаты ее работы для нескольких ПЛМ приведены в таблице.

Средний процент экономии площади матрицы 40. Поскольку топология свернутой ПЛМ имеет ряд особенностей (разрывы сигнальных линий, расположение усилителей входных сигналов сверху и снизу матрицы и т. п.), эта цифра не отражает реальной экономии площади кристалла, однако дает представление об эффективности работы алгоритма.

Программа активно используется при проектировании БИС, наряду с другими средствами оптимизации ПЛМ.

СПИСОК ЛИТЕРАТУРЫ

1. Ульман Дж. Д. Вычислительные аспекты СБИС.—М.: Радио и связь, 1990.
2. Grass W. A depth-first branch and bound algorithm for optimal PLA folding // Proc. 19-th ACM/IEEE Design Automatic Conference.—Las Vegas, Nevada, 1982.—P. 133.
3. Hachtel G. D., Newton A. R., Sangiovanni-Vincentelli A. L. Techniques for PLA folding // Ibid.—P. 147.
4. De Micheli G., Sangiovanni-Vincentelli A. L. PLEASURE: A computer program for simple/multiple constrained/unconstrained folding of PLA // Proc. 20-th ACM/IEEE Design Automation Conference.—Miami Beach, 1983.—P. 530.
5. Липский В. Комбинаторика для программистов.—М.: Мир, 1988.
6. Maciej M. Syslo Discrete Optimization Algorithm with Pascal Program.—Englewood Cliffs, New Jersey: Prentice-Hall, 1983.
7. Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов.—М.: Наука, 1990.
8. Xuequn Xiang, Saburo Muroga. Interactive reduction of folded PLAs // IEEE.—1986.—P. 592.
9. Sun Young Hwang, Dutton R. W., Blank T. A best-first search algorithm for optimal PLA folding // IEEE Trans. on CAD.—1986.—CAD-5, N 3.
10. Kuo Y. S., Chen C., Hu T. C. A heuristic algorithm for PLA block folding // Proc. 22nd Design Automation Conference.—Las Vegas, Nevada, 1985.—P. 744.
11. Hachtel G. D., Newton A. R., Sangiovanni-Vincentelli A. L. An algorithm for optimal PLA folding // IEEE Trans. Computer-Aided Design.—1982.—CAD-1.—P. 63.

Поступила в редакцию 28 марта 1992 г.

УДК 621.138 : 519.87

Е. Г. Антонянц

(Новосибирск)

АЛГОРИТМ «ПЛУГИРОВАНИЯ» ДЛЯ КОМПАКТИЗАЦИИ ТОПОЛОГИИ СБИС

Представлен алгоритм для модификации топологии СБИС, который можно использовать, с одной стороны, как одномерный компактизатор, с другой — как средство для переорганизации топологии СБИС без изменения электрической схемы, сохраняя при этом корректность по отношению к конструкторско-технологическим ограничениям. Приведены сведения о программной реализации алгоритма «плугирования» и других алгоритмов, существенных для эффективной работы этой операции.