Раздел II. Автоматизация проектирования

УДК 658.512.2.011.5

Б.К. Лебедев, О.В. Рябов

ПОСТРОЕНИЯ ГРАФА ОГРАНИЧЕНИЯ МЕТОДОМ СПИСКА ГЛУБИН ДЛЯ ЗАДАЧИ СЖАТИЯ*

Как правило, после применения алгоритмов размещения и трассировки к схеме она потенциально может быть уплотненна. Задача такого уплотнения называется задачей сжатия. Существует несколько алгоритмов реализующих сжатие, в частности можно упомянуть сжатия на основе виртуальной сетки и один из наиболее эффективных алгоритмов сжатия – сжатия при помощи графа ограничений. Для алгоритма сжатия на основе виртуальной сетки [1] требуется предварительно наложить на схему виртуальную сетку. После этого производятся деформации этой сетки таким образом, чтобы сократить занимаемой схемой пространство, с учетом технологических ограничений.

Сжатие на основе виртуальной сетки обычно дают худшие по качеству результаты, чем сжатие на основе графа ограничений. Для сжатия на основе графа ограничений требуется построить граф ограничений. Граф ограничений это ориентированный граф. В настоящие время наиболее распространенными являются алгоритмы построения графа ограничения методом отброса тени [2] и методом сканирующей строки [3]. После того, как был построен граф, требуется найти в нем критический путь. Критическим путем называют путь, имеющий максимальную длину. Элементы, входящие в критический путь – это элементы, которые задают размерность платы и должны быть размешены максимально плотно. Остальные элементы, не входящие в критический путь, можно, без отрицательных последствие для размеров, передвигать в некотором диапазоне и при их размещении можно учитывать дополнительные условия. Для повышения эффективности сжатия с помощью процедур реорганизации осуществляется модификация топологии. Эти процедуры заключаются в перемещение элементов по осям, что позволяет получить более плотную упаковку. Процедуры модификации осуществляются с помощью алгоритмов имеющих итерационной характер. В настоящие время наиболее эффективны: алгоритмы генетического поиска, метод отжига, метод адаптации. В частности, можно отметить, что предлагаемый алгоритм был разработан авторами в процессе исследования роли генетических алгоритмов в задачи сжатия, когда авторы были неудовлетворенны скоростью существующих алгоритмов. На каждом этапе итерационного метода требуется строить новый граф ограничений и находить критический путь в нем.

Аналогично и для генетических алгоритмов – для каждого индивидуума в каждой генерации каждой популяции требуется произвести оценку пригодности решения, которая сводится к построению графа ограничения и нахождению в нем критического пути. Из вышесказанного видно, что в современных алгоритмах сжатия граф ограничений строится многократно на протяжении работы алгоритма и его много-

* Работа выполнена при поддержке: РФФИ (гранты № 07-01-00174, № 08-01-00473), РНП 2.1.2.3193, РНП 2.1.2.2238.

кратное нахождение составляет существенное время работы алгоритмов сжатия. В связи с вышесказанным, разработка новых более быстродействующих алгоритмов построения графа ограничений представляется актуальной задачей. При этом важно, чтобы алгоритм построения графа ограничений не создавал избыточных ребер, в противном случае, это приведет к дополнительным затратам времени на их перебор в процессе нахождения критического пути. С целью решения этой задачи авторами был разработан быстродействующий алгоритм построения графа ограничений, не создающий избыточных ребер, который основан на построении списка глубин.

Рассмотрим особенности основных алгоритмов построения графа ограничений.

Алгоритм построения графа ограничения методом сканирующий строки заключатся в том, что через поле содержащие элементы, проводится воображаемая линия перпендикулярно той плоскости, для которой ищется граф ограничений. Линия передвигается в одном из направлений перпендикулярных плоскости, для которой необходимо построить граф ограничений, последовательно проходя через все элементы. Каждый раз, когда линия проходит через начало элемента, если движение происходит в прямом направление (если в обратном – то через конец), элемент добавляется в список. При прохождении через конец элемента (если движение происходит в обратном направлении то через начало) элемент удаляется из списка.

Таким образом, на каждом шаге алгоритма мы будем иметь списком элементов перекрывающих друг друга в выбранном нами направлении. Пройдя, таким образом, всю схему и проводя на каждом шаге ребра ко всем элементам, находящимся в текущий момент в списке, мы получим граф ограничений. Недостатком этого метода является построение графа ограничений содержащего много избыточных ребер, что увеличивает продолжительность вычислений необходимых для нахождения критического пути. Предлагаемый авторами алгоритм не создает избыточных ребер в графе ограничений.

Рис. 1. Построение графа ограничения методом сканирующий строки

Алгоритм метода отбрасывания тени заключается в том, что для определения перекрывающегося элемента строятся с некотором шагом лучи в плоскости перпендикулярной той которой требуется построить граф ограничений, на протяжение всего размера элемента в плоскости параллельной плоскости искомого графа ограничений с учетом конструкторских ограничений. Каждый такой луч распространяется до тех пор, пока не упрется в другой элемент или в стенку. Получив список элементов, в которые уперлись лучи данного элемента, можем провести ребра к вершине графа обозначающий данный элемент. Проведя такую операцию со всеми элементами, мы получим граф ограничений, не содержащий избыточных ребер. Недостатком данного алгоритма является большой объем вычислений, требуемых для его выполнения. Это связано с тем, что требуется перебирать точки на некотором прямоугольном пространстве с целью найти элементы в нем находящиеся или обнаружить их отсутствие. Предлагаемый авторами алгоритм требует значительно меньших вычислений, поскольку не анализирует пространство позади элемента, а рассматривает только критические точки.

Алгоритм построения графа ограничений с помощью списка глубин. Проекции координаты начала или конца элемента на ось, по которой будет строится граф, будем называть точкой. Точка содержит номер элемента и информацию о том – является она конечной или начальной, и значение координаты, которую будем называть высотой точки.

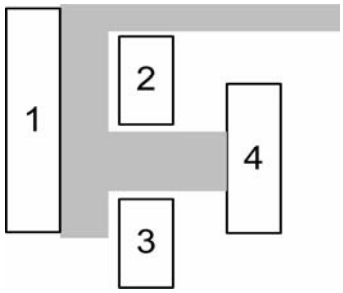

Рис. 2. Построение графа ограничения методом отбрасывания тени

Для работы алгоритма требуется предварительно подготовить входные данные.

На первом этапе подготовки входных данных составляется список точек в проекции по выбранной координате построения графа ограничений. Для получения списка точек из элементов мы просматриваем список элементов и для каждого из них добавляем две точки, первая из которых соответствует началу элемента с учетом конструкторских ограничений, а вторая концу элемента с учетом конструкторских ограничений.

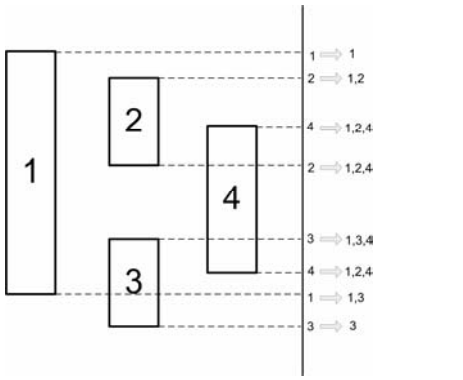

Рис. 3. Построение графа ограничения методом списка глубин

На втором этапе к каждой точке прикрепляется список глубин. Список глубин строится путем определения всех элементов располагающихся в этой точке и занесением их номеров и глубин в упорядоченный по убыванию глубины список. Для этого просматриваем список элементов, выбирая те из них, для которых выполняется условие:

$$\text{НачалоЭлемента} \le \text{ВысотаТочки} \le \text{КонецЭлемента},$$

с учетом проектных ограничений. Если элемент попадает в данный диапазон, то заносим его номер и значение по другой координате, которое будем называть глубиной.

Получив входные данные можно приступить к выполнению алгоритма. Для каждого элемента выполняется перебор всех точек на диапазоне от начальной точки рассматриваемого элемента до конечной включая их. Выделятся переменная, которая хранит номер текущего элемента, первоначально инициализируемая выбранным элементом. Текущим элементом называем последний элемент, добавленный в граф. На всех точках диапазона происходит рассмотрения списка глубин и сравнения их с глубиной рассматриваемого элемента, если присутствуют элементы с большей глубиной чем имеет элемент, то для той из них, которая имеет наименьшую глубину, проводится ребро в графе ограничений от рассматриваемого элемента. Для начальных точек рассматриваются все значения глубин, а для конечных – все, кроме значения того элемента, которому принадлежит конечная точка при условии, что он является текущим. Весом ребра будет размер элемента с учетом проектных ограничений по координате, принятой за глубину соответствующей точки. Рассмотрев таким образом каждый элемент, мы получим граф ограничений с весами на ребрах.

```
Формальная запись алгоритма:
In: ListOfDots, ListOfElements
Out: Graph[,]
For I = 0 to ListOfDots.Count do
 Begin
  CurNum = I;
   For J= 0 to ListOfDots.ListOfDeep.Count do
    Begin
      IF  (ListOfDots.ListOfDeep[J].Deep > ListOfDots.ListOfDeep[J].Deep) AND
       (NOT((ListOfDots[i].ListOfDeep[J].Number = CurNum) AND
       (ListOfDots[i].Number = CurNum) ) OR
       (ListOfDots[i].ListOfDeep[J]. Start = true)))  THEN
         Begin
           CurNum = ListOfDots[i].ListOfDeep[J].Number
           Graph[I,j] = ListOfElements[j].Size
           Break
         End
    End
 End
```

Экспериментальные исследования. Были проведены экспериментальные исследования, заключающиеся в построения графика зависимости продолжительности вычислений от количества элементов схемы. В процессе исследований для топологий с разным количество элементов находились граф ограничений и критический путь в нем. Результаты представлены на рис. 4.

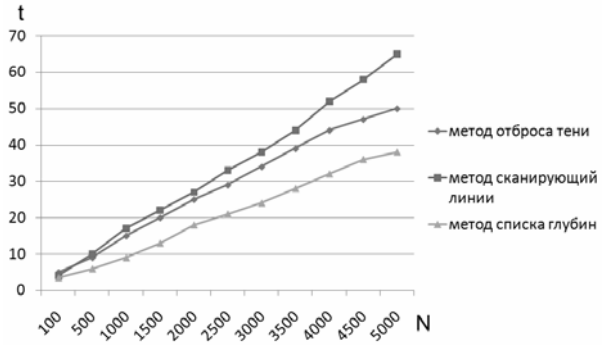

Рис. 4. Экспериментальные данные

Меньшее значение на графике соответствует меньшим временным затратам. Как видно из рисунка (см. рис. 4), лучшую производительность показал алгоритм сжатия, использующий метод списка глубин для построения графа ограничения. Наиболее медленным оказался алгоритм, использующий метод сканирующий линии. Это объясняется тем, что он создает граф ограничений, содержавший большое количество избыточных ребер, причем, оно нелинейно возрастает при увеличении числа элементов. Избыточные ребра осложняют поиск критического пути, в связи с чем, происходит увеличение общего времени работы. Алгоритм строящий граф ограничения методом отброса тени занимает промежуточное место, что объясняется тем, что хотя он и медленней чем алгоритм списка глубин, тем не менее, в отличие от алгоритма сканирующий линии, он не создает избыточных ребер в графе, требующих дополнительных затрат на обработку.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *S.B. Akers, J.M. Geyer and D.L. Roberts IC* Mask Layout with a Single Conductor Layer // Proceedings of 7th Design Automation Workshop, pages 7-16,1970.
2. *M.Y. Hsueh and D.O. Pederson* Computer-Aided Layout of LSI Circuit Building-Block // Ph.D. thesis, University of California at Berkeley, December, 1979.
3. *C.W. Carpenter and M. Horowitz,* Generating Incremental VLSI Compaction Spacing Constraints, Proceedings of the 24th ACM/IEEE Design Automation Conference, pp. 291-297, IEEE Computer Society Press, June 1987.

УДК 621.03

Е.В. Нужнов, А.А. Полупанов

ОСОБЕННОСТИ И ВОЗМОЖНОСТИ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ ПЛИС РАЗЛИЧНОЙ АРХИТЕКТУРЫ*

Введение. Среди программируемых логических интегральных схем (ПЛИС) наибольший спрос сегодня имеют кристаллы Field Programmable Gate Array (**FPGA**) и комплексные программируемые логические устройства (Complex Programmable Logic Device, **CPLD**). ПЛИС состоит из матрицы блоков программируемой логики, между строками и столбцами которой имеются программируемые соединения. Пользователь ПЛИС может перепрограммировать блоки и связи даже в режимах эксплуатации своего устройства без ограничения числа перепрограммирований. Современные высокоинтегрированные кристаллы ПЛИС содержат, кроме упомянутых матриц, встроенную мощную память, трансиверы и даже микропроцессоры, которые пользователь может подключать для решения своих задач с помощью программируемых соединений внутри кристалла.

В настоящее время, ведущим мировым производителем ПЛИС является фирма Xilinx, которая предлагает пользователям множество семейств своих устройств [1, 2]. Рассмотрим особенности некоторых семейств ПЛИС, предлагаемых фирмой Xilinx.

Особенности ПЛИС CPLD семейства XC9500. Семейство XC9500 имеет структуру CPLD, которая напоминает структуру **EPLD** – (Erasable Programmable Logic Device) – основу их составляют макроячейки PAL-типа (рис. 1, где PLA – Programmable Logic Array; PAL Programmable Array Logic), позволяющие получать логические функции многих переменных с ограниченным числом термов.

Микросхемы этого типа могут быть использованы для создания нестандартных АЛУ, дешифраторов, мультиплексоров и т.д., где требуется логические функции многих переменных и небольшое число триггеров.

Микросхемы семейства XC9500 могут использоваться в крупносерийной аппаратуре, а также в системах, где требуется перепрограммирование «на ходу». Для программирования микросхем семейства XC9500 не требуется программатор. Перепрограммирование осуществляется сигналами от элементов с напряжением питания 5 В через специальные выводы микросхемы (JTAG-порт) в той же системе,

* Работа выполнена при поддержке: РФФИ (гранты № 05-08-18115, № 08-01-00473), РНП 2.1.2.3193, РНП 2.1.2.2238.