УДК 621.3

Д.И. Лазоренко

Институт проблем моделирования в энергетике им. Г.Е. Пухова, Киев

АЛГОРИТМ ОБЪЕДИНЕНИЯ ОДНОМЕРНЫХ ЦИКЛОВ ИСХОДНОГО ТЕКСТА ОПИСАНИЯ ЦИФРОВЫХ СИСТЕМ С ЦЕЛЬЮ СНИЖЕНИЯ ИХ ЭНЕРГОПОТРЕБЛЕНИЯ

В данной статье предлагается алгоритм анализа циклов на возможность их объединения, что позволяет уменьшить количество обращений к памяти и её объём.

одномерные циклы, энергопотребление, цифровая система

Введение

Развитие полупроводниковых технологий привело к возникновению концепции Системы-на-Кристалле. Сложность современных приложений и использование субмикронных технологий обуславливают необходимость снижения энергопотребления таких систем путём применения оптимальных решений в процессе проектирования.

Современные цифровые системы, например, мультимедийные приложения, переносные телефоны, карманные персональные компьютеры, обрабатывают большие массивы данных по сложным алгоритмам. Развитие технологий переносных источников питания не успевает за увеличением энергопотребления новых приложений. Кроме того, пониженное энергопотребление позволяет упростить разводку шин питания на кристалле, приводит к уменьшению шумов на шинах питания, проявления эффекта электромиграции и электромагнитного излучения. Далее рассмотрим основные источники энергопотребления в микросхемах и более детально остановимся на методах преобразования кода с целью уменьшения обращений к памяти. После этого предлагается графический метод объединения циклов, позволяющий преобразовывать описание устройства на языке высокого уровня так, чтобы минимизировать операции с памятью (чтение/запись). Показана работоспособность метода, когда другие подходы не работают.

Источники энергопотребления в КМОП схемах. Подавляющее число современных микросхем производится с помощью КМОП (Комплиментарный Металл-Оксид-Диэлектрик) технологии, поэтому источники энергопотребления будут рассматриваться применительно к этой технологии. Существует четыре составляющие энергопотребления КМОП схем – токи короткого замыкания, статические токи, паразитные токи утечки, и динамическое рассеяние энергии.

$$P_{total} = P_{short} + P_{stat} + P_{leak} + P_{dyn}.$$

Токи короткого замыкания. Токи короткого замыкания существуют в процессе нормального функционирования логических схем. Под ними по-

нимаются токи, протекающие через транзисторы из шины питания непосредственно в шину земли.

Статические токи. Под статическими понимаются временные или постоянные токи.

Паразитные токи утечки. Под паразитными токами утечки понимаются подпороговые токи транзисторов и токи в подложке.

Динамическое рассеяние энергии. Динамическое рассеяние энергии происходит из-за зарядки/разрядки узлов схемы, и может быть представлено как

$$P_{dyn} = C \cdot V_{dd}^2 \cdot \alpha \cdot f ,$$

где C – суммарная ёмкость в узлах схемы; V_{dd} – величина напряжения питания; f – частота переключений; α – коэффициент переключательной активности (среднее число логических вентилей, переключающихся в течение одного цикла сигнала синхронизации) [1]. Динамическое рассеяние энергии может составлять до 80% от полных потерь энергии. Переключательная активность в значительной мере определяется программным обеспечением цифровой системы [2, 3].

Потенциальный выигрыш в энергопотреблении. В работе [4] обоснован вывод, что потенциальный выигрыш в энергопотреблении тем выше, чем выше уровень абстракции процесса проектирования, на котором принимается решение. Для системного уровня выигрыш может быть от 50 до 90%, на поведенческом уровне – от 40 до 70%, на RTL уровне – от 30 до 50%, на уровне вентилей – от 20 до 30%, на уровне транзисторов – от 10 до 20%, на уровне топологии – от 5 до 10%.

Энергопотребление схем памяти. Современные цифровые системы используют большие объёмы памяти. Схемы памяти могут занимать от 50 до 80% площади полупроводникового кристалла. Известно, что схемам памяти присущи большие паразитные токи утечки. Кроме того, на обращение к памяти тратится много энергии, например, на операцию чтения из внешней памяти расходуется в 33 раза более энергии, чем на операцию 16-битного сложения. Согласно прогнозу международной организации International Technology Roadmap for Semiconductors схемы памяти будут занимать всё больше площади на полупроводниковых кристаллах: в 2008 году – 83%, в 2011 – 90%, в 2014 – 94%. Также, величина динамического энергопотребления схем памяти в ближайшем будущем будет только увеличиваться. Например, по прогнозу на 2006 г. динамическое рассеяние энергии на схемах памяти будет на 25% больше, чем на логических схемах, к 2010 г. эта величина увеличиться до 2 раз, к 2015. – до 2,5 раз, к 2020. – до 3 раз [5].

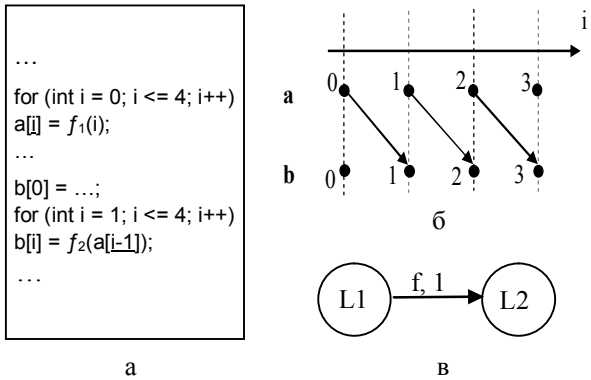
Очевидно, в процессе проектирования нужно добиваться уменьшения объёма требуемой приложению памяти и количества обращений к ней. Для этого необходимо оптимизировать систему на поведенческом уровне. Важно, чтобы такая оптимизация проводилась перед разделением системы на программную и аппаратную части, поскольку это позволяет применить однообразный подход к обработке всей памяти. Циклы «for» представляют собой именно ту часть исходного кода приложения, которая ответственна за использование массивов в приложениях, обрабатывающих большие объёмы данных по сложным алгоритмам [6 – 8].

Для снижения объёма требуемой памяти необходимо уменьшить размер и количество временных массивов, создаваемых в процессе обработки данных. Уменьшения объёма памяти можно также добиться путём повторного использования одних и тех же адресов памяти разными массивами.

Алгоритм объединения одномерных циклов

Известно, что основным источником энергопотребления схем памяти являются операции чтения и записи. В данной статье предлагается алгоритм анализа циклов на возможность их объединения, что позволяет уменьшить количество обращений к памяти и её объём. Данный алгоритм использует представление исходного текста программы в виде графов и основывается на предложенном в [9] графическом методе объединения одномерных циклов исходного текста описания цифровых систем. Рассмотрим далее формирование и преобразование упомянутых графов.

Пусть есть некоторый исходный текст программы, содержащий одномерные циклы (рис. 1, а). На рис. 1, б приводится графическое представление связей между элементами массивов в процессе вычислений. Так, стрелки соединяют элементы одного массива, значения которых используются при вычислении элементов второго. Над точками соответствующим элементам массивов **a** и **b** горизонтально проведена ось итераций. Как видно из рис. 1, б, проекции стрелок на эту ось положительны и равны 1. На рис. 1, в изображён граф исходного текста программы. Его вершины L1 и L2 соответствуют массивам **a** и **b**. Между циклами существует зависимость по данным, эта зависимость представлена в графе дугой направленной от вершины L1 к вершине L2. Дуге присваивается вес 1, равный величине проекций стрелок на ось итераций на рис. 1, б. Ярлык «f» нужен для обозначения типа зависимости по данным, поскольку ниже будет рассматриваться также тип связи по выходу.



а б в

Рис. 1. Одномерный цикл: а – исходный текст; б – графическое представление зависимостей; в – граф вычислений

Пусть элементы массива вычисляются внутри цикла, используя значения других элементов этого же массива (рис. 2, а). Графическое представление зависимостей показано на рис. 2, б. В графе данная ситуация отображается наличием направленных дуг, которые начинаются и заканчиваются на одной и той же соответствующей определённому массиву вершине графа (рис. 2, в).

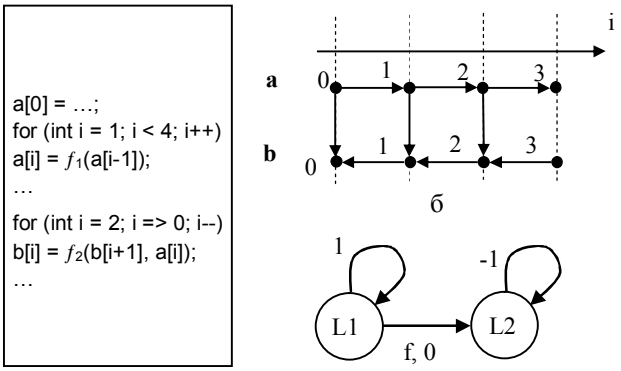


Рис. 2. Вычисление элементов массива внутри цикла с использованием других элементов массива: а – исходный текст программы; б – графическое представление зависимостей; в – граф вычислений

Вес замкнутой на одну вершину дуги определяется таким же образом, как это было показано на примере рис. 1. Проекции стрелок, связывающих элементы массива **a** между собой, на ось итераций равны 1. Проекции же стрелок, связывающих элементы массива **b** между собой, на ось итераций равны -1. Циклы, приведенные на рис. 2, а невозможно объединить. Этому на графе соответствует наличие у соединённых вершин L1 и L2 замкнутых направленных дуг с весами противоположных знаков.

На рис. 3, а представлен некоторый исходный текст. Циклы в исходном виде невозможно объединить. Данной ситуации соответствует отрицательное значение проекций стрелок, соединяющих элементы массивов **a** и **b** на рис. 3, б, и отрицательное значение веса дуги, соединяющей вершины L1 и L2, на рис. 3, в.

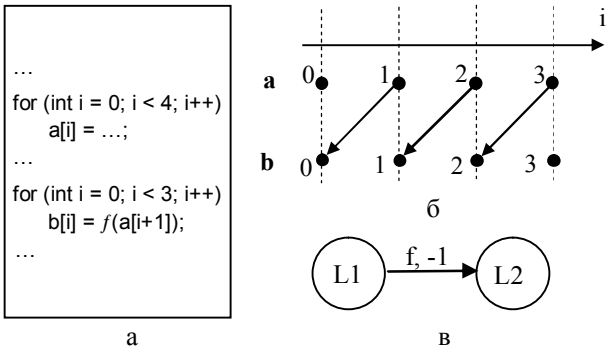


Рис. 3. Необъединенные циклы: а – исходный текст; б – графическое представление зависимостей; в – граф вычислений

В работе [9] описан способ трансформации графического представления зависимостей при вычислении циклов, благодаря которому становится возможным их объединение. Ниже предлагается основанный на упомянутом способе алгоритм преобразования графов.

Перед началом преобразования графа присвоим каждой его вершине нулевой вес. Данный вес будет изменять по правилам выведенным ниже.

Рассмотрим преобразования графа на примере некоторого исходного текста на рис. 4, а. На рис. 4, б показано соответствующее данному тексту исходное графическое представление зависимостей, а на рис. 4, в – исходный граф.

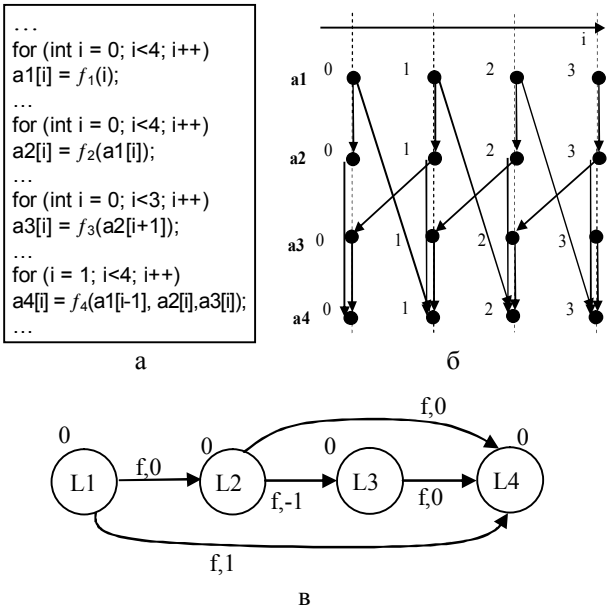
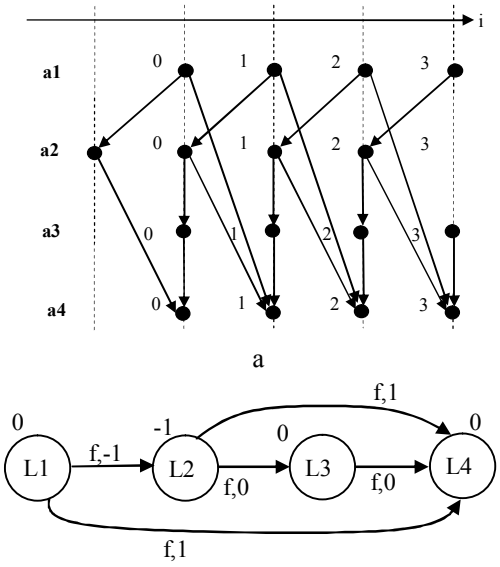


Рис. 4. Анализируемый фрагмент:: а – исходный текст программы; б – графическое представление зависимостей; в – граф вычислений

Как было предложено в работе [9], трансформируем графическое представление зависимостей так, чтобы в нём не осталось стрелок с отрицательными проекциями на ось итераций. Сначала сдвигаем точки, соответствующие элементам массива **a2** на расстояние одной итерации влево. На рис. 5, а показаны результаты этого действия, и на рис. 5, б – соответствующий новому представлению зависимостей граф.

Как видно из рис. 5, б вес дуги L2→L3 стал равен 0. Вес дуги L2→L3 также увеличился на 1 и стал равным 1. Вес же дуги L1→L2 уменьшился на 1 и стал равен -1. Вес вершины L2 уменьшился на 1 (что соответствует сдвигу точек, представляющих элементы массива, влево на расстояние одной итерации) и стал равен -1. Таким образом, если вес какой-либо дуги увеличивается на определённую величину, то вес вершины, из которой выходит данная дуга, уменьшается на данную величину. Вес дуг, входящих в эту вершину, также уменьшается на ту же величину, а вес дуг, выходящих из этой вершины, увеличивается на упомянутую величину. Двигаясь дальше к вершине L1 (началу графа), проводим аналогичные преобразования. Вес дуги L1→L2 увеличивается на 1 и становится равным 0. Вес дуги L1→L4 увеличива-

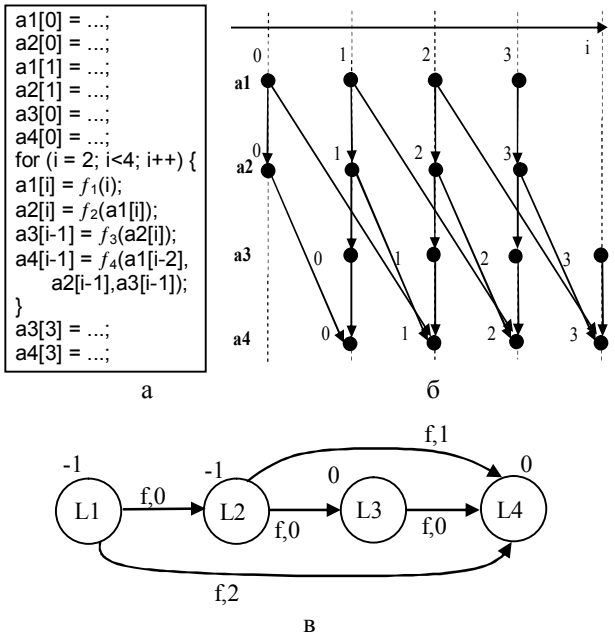
ется на 1 и становится равным 2. Вес вершины L1 уменьшается на 1 и становится равным -1.



а б

Рис. 5. Трансформация графа: а – графическое представление зависимостей; б – граф вычислений

На рис. 6 показан преобразованный граф и текст объединённого цикла.

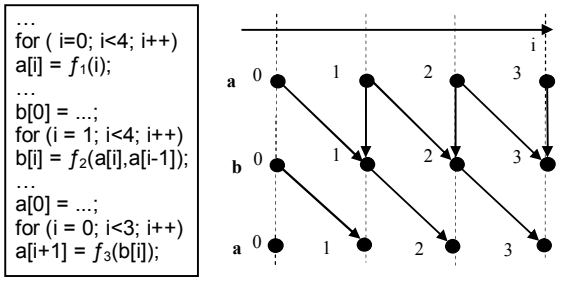


а б в

Рис. 6. Объединение циклов::
а – текст объединённого цикла; б – итоговое графическое представление зависимостей; в – итоговый граф

Вес вершин графа используется при формировании текста объединённого цикла.

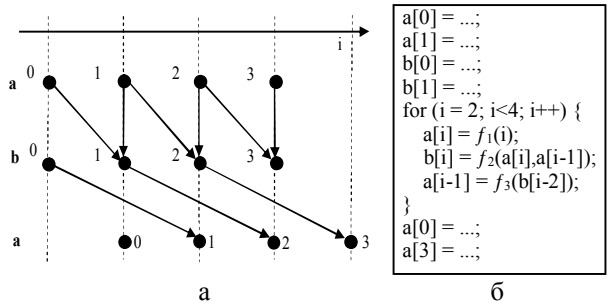
Теперь рассмотрим объединение циклов, если между ними присутствует связь по выходу. На рис. 7, а представлен некоторый исходный текст программы, содержащий три цикла. Между первым и третьим существует связь по выходу. На рис. 7, б изображено исходное графическое представление зависимостей.



а б

Рис. 7. Объединение циклов, имеющих связь по выходу:
а – исходный текст программы;
б – графическое представление зависимостей

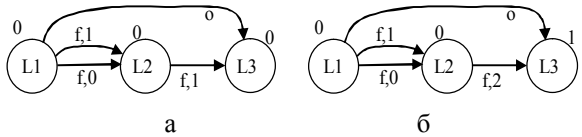
В исходном виде данные три цикла объединить невозможно. Например, значение элемента a[1] необходимо для вычисления b[2], но до этого оно уже перезаписывается во время выполнения третьего цикла. Чтобы сделать объединение циклов возможным, необходимо перезаписывать значение a[i] уже после того, как оно было использовано для вычисления b[i+1]. Преобразованное графическое представление зависимостей для этого случая показано на рис. 8, а, а текст объединённого цикла – на рис. 8, б.



а б

Рис. 8. Результат объединения циклов:
а – преобразованное графическое представление зависимостей; б – текст объединённого цикла

На рис. 9, а представлен граф, соответствующие исходному тексту программы на рис. 7, а. На рис. 9, б – итоговый граф вычислений. Дуга с ярлыком «о» отображает наличие связи по выходу и не имеет веса. Вначале каждой вершине присваивается нулевой вес. Вес дуг вычисляется аналогично тому, как это было сделано выше.



а б

Рис. 9. Графы программ рис. 7,8:
а – исходный граф; б – итоговый граф

Аналогично тому, как это было сделано выше, сдвиг точек, изображающих элементы массива **a** в третьем цикле, вправо на расстояние одной итерации соответствует увеличению веса вершины L3 на 1. Эта величина равна максимальному весу f-дуг, исходящих из вершины L1 связанной с L3 o-дугой. При этом вес всех f-дуг, входящих в L3, должен

увеличиться на ту же величину, на которую увеличился вес вершины L3, вес же всех исходящих из данной вершины дуг должен уменьшиться на упомянутую величину.

Для обобщения, ниже приводятся алгоритмы построения и преобразования графического отображения циклов.

Алгоритм построения графа исходного текста программы:

1. Каждому вычисляемому в циклах массиву ставится в соответствие вершина графа с начальным нулевым весом.

2. Вершинами соединяются направленными дугами. Дуги исходят из вершин, соответствующих тем массивам, которые вычисляются первыми в исходном тексте программы.

2.1. Дуги, соответствующие связи по выходу, не имеют веса и обозначаются ярлыком «o».

2.2. Дуги, соответствующие связи по данным, обозначаются ярлыком «f». Начальный вес дуги определяется как разница между индексными выражениями для массива, который вычисляется в левой части выражения, соответствующего вершине, из которой выходит дуга, и правой части выражения, использующего этот массив и соответствующего вершине, на которой дуга заканчивается.

Проиллюстрируем пункт 2.2 на примере рис. 1. Между первым и вторым циклами существует связь по данным (массив **a**), поэтому из вершины L1, соответствующей вычислению массива **a**, выходит дуга в вершину L2, соответствующую вычислению массива **b**. Дуга отмечается ярлыком «f». Индексное выражение для массива **a** в левой части выражения, соответствующего вершине L1, есть i. В правой части выражения, соответствующего вершине L2 – i-1. Разница этих индексных выражений есть i–(i-1)=1, начальный вес дуги будет равен 1.

Алгоритм преобразования графа выглядит следующим образом:

1. Если существуют «f»-дуги с отрицательными значениями веса, тогда, двигаясь от такой дуги к началу графа, сделать все значения веса «f»-дуг неотрицательными следующим образом: увеличить вес дуги до нуля на необходимую величину, затем уменьшить вес вершины, из которой вышла данная дуга, на упомянутую величину; все другие «f»-дуги, исходящие из данной вершины, увеличивают свой вес на указанную величину, а «f»-дуги входящие в эту вершину, наоборот, на данную величину свой вес уменьшают.

2. Если никакими преобразованиями нельзя сделать значения веса «f»-дуг неотрицательными, то все исследуемые циклы объединить невозможно.

3. В случае, если все циклы не могут быть объединены, разбить циклы на группы, которые расположены выше и ниже вершин, соединённых «f»-дугами с отрицательными значениями веса.

3.1. Если внутри какой-либо из групп есть циклы со связями по выходу, то вес вершины, в которую входит «o»-дуга, должен быть увеличен на величину, равную весу «f»-дуги, которая обладает максимальным весом из всех выходящих из вершины исходной для данной «o»-дуги.

Выводы

Предлагаемый алгоритм представляет собой простой и наглядный способ анализа циклов с точки зрения возможности их объединения. Как показано в работе [9], данный способ позволяет добиться лучших результатов в объединении циклов, чем, например, предлагаемый в [7]. Также, данный способ позволяет анализировать влияние связей внутри одного цикла на возможность объединения его с другим, что отсутствует в методе, предлагаемом в [7]. В результате объединения циклов, выполненного согласно предлагаемому методу, можно существенно сократить количество операций обращения к медленной памяти, а также уменьшить её размер.

Автор планирует обобщить предлагаемый подход на случай многомерных циклов, а также рассмотреть вопросы автоматизации данного метода для ввода его в процесс оптимизации кода компилятором с языка SystemC.

Список литературы

1. Veendrick H.J.M. Deep-Submicron CMOS ICs. From Basics to ASICs. – Kluwer academic publishers, 2000. – 539 p.

2. Poppen F. Low Power Design Guide. – OFFIS Research Institute [Электрон. ресурс]. – Режим доступа: http://www.offis.de.

3. Kim H.S., Irwin M.J., Vijaykrishnan N., Kandemir M. Effect of compiler optimizations on memory energy // IEEE Workshop on Signal Proc. Systems. SiPS 2000. – 2000. – P. 663-672.

4. Sproch J. High Level Power Analysis and Optimization. Tutorial // International Symposium on Low Power Electronics and Design. – 1997. – 340 p.

5. Int. Technology Roadmap for Semiconductors [Электрон. ресурс]. – Режим доступа: http://public.itrs.net.

6. Fraboulet A., Huard G., Mignotte A. Loop Alignment for Memory Accesses Optimization // Twelfth International Symposium on System Synthesis. Proc. (ISSS'99). IEEE Computer Society Press. – 1999. – P. 71-77.

7. Fraboulet A., Kodary K., Mignotte A. Loop fusion for memory space optimization // The 14th Int. Symposium on System Synthesis. Proc. 2001. – 2001. – P. 95-100.

8. Catthoor F., Franssen F., Wuytack S., Nachtergaele L., De Man H. Global communication and memory optimizing transformations for low power signal proc. systems // Workshop on VLSI Signal Processing, VII. – 1994. – P. 178-187.

9. Лазоренко Д.И. Метод высокоуровневых трансформаций исходного кода описания цифровых систем с целью снижения их энергопотребления // Збірник наук. праць ІПМЕ ім. Г.Є. Пухова. – К.: ІПМЕ, 2007. – Вип. 38. – С. 41-53.

Поступила в редколлегию 11.07.2007

Рецензент: д-р техн. наук, проф. Ю.М. Коростель, Институт проблем моделирования в энергетике им. Г.Е. Пухова, Киев.