УДК 004.051 DOI: 10.17587/prin.13.259-271

Д. С. Романова[1, 2], аспирант, ассистент, daryaooo@mail.ru,
О. В. Непомнящий[1], канд. техн. наук, проф., зав. кафедрой, 2955005@gmail.com,
А. И. Легалов[3], д-р техн. наук, проф., legalov@mail.ru,
И. Н. Рыженко[1], ассистент, rodgi.krs@gmail.com,
Н. Ю. Сиротинина[1], канд. техн. наук, доц., nsirotinina@sfu-kras.ru
[1] Сибирский федеральный университет, Красноярск
[2] Красноярский государственный аграрный университет
[3] Национальный исследовательский университет "Высшая школа экономики", Москва

Методы редукции параллелизма в процессе высокоуровнего синтеза цифровых интегральных схем[1]

Рассмотрены проблемы и решения в области обеспечения архитектурной независимости и организации процесса сквозного проектирования цифровых интегральных схем. Представлены метод и язык параллельного программирования для функционально потокового синтеза проектных решений. При реализации метода функционально-потокового синтеза выделены задачи редукции параллелизма и оценки занимаемых ресурсов. Предложен способ свертки (сокращения) параллелизма, базирующийся на введении дополнительного слоя в процесс синтеза. Разработаны принцип и алгоритмы редукции параллелизма. Представлены результаты разработки программного инструментария поддержки проектирования и реализованные на практике проекты СБИС.

Ключевые слова: *интегральная схема, модель параллельных вычислений, алгоритм, программа, высокоуровневый синтез, функционально-потоковый язык*

Введение

Современный уровень развития цифровых интегральных схем (ЦИС) характеризуется постоянно возрастающими требованиями к системной организации всего процесса проектирования. В связи с этим одной из основных задач проектирования является обеспечение архитектурной независимости (первая задача), т. е. переносимости проектных решений между целевыми платформами ЦИС. Не менее важным является сокращение сроков получения конечного результата (вторая задача), что обеспечивается максимально возможным исключением из процесса проектирования итерационных операций возврата к предыдущим этапам, т. е. обеспечение процесса "сквозного проектирования" в процессе всех стадий маршрута проектирования ЦИС (структурного, поведенческого и физического уровней). На рис. 1 приведен весь маршрут проектирования ЦИС, где выделены первые пять этапов, иллюстрирующие место VLSI-дизайна (*Very Large Scale Integration*). В случае обеспечении архитектурной независимости про-

Рис. 1. Маршрут проектирования ЦИС:
ФПП язык — функционально-потоковый параллельный язык; HDL — *Hardware Description Language* (язык описания аппаратуры); RTL — *Register Transfer Level* (уровень регистровых передач)

[1] Статья подготовлена по материалам доклада на Седьмой Международной конференции "Актуальные проблемы системной и программной инженерии" АПСПИ 2021.

ектных решений переносимость позволит отказаться от привязки к конечной форме реализации на верхних уровнях абстракции проекта, когда происходит отработка алгоритмов функционирования. Это обеспечит наиболее эффективный выбор целевой платформы для конечной реализации при переходе на нижние уровни проектирования. Тем самым обеспечиваются наиболее оптимальные технические характеристики, такие как быстродействие, занимаемая площадь кристалла, энергопотребление и т. п. В случае сокращения сроков получения итогового результата сквозное проектирование ведет к уменьшению финансовых затрат и обеспечению конкурентоспособной продукции.

Интегральная схема, по существу, является системой параллельной обработки информационных потоков, а алгоритмы ее функционирования и архитектура на заключительных этапах синтеза представляются на языках описания аппаратуры. Поэтому эффективные решения по обеспечению архитектурной независимости могут быть найдены в области переносимых параллельных программ. Это позволит не только обеспечить требуемые технические характеристики СБИС, но и автоматизировать некоторые стадии работы над проектом. В свою очередь сквозное проектирование можно обеспечить за счет использования модели функционально-потоковых параллельных вычислений и описания исходных алгоритмов в виде ациклических конструкций [1—4]. Описание исходных алгоритмов на ФПП языке происходит на первоначальном этапе в маршруте проектирования СБИС.

Ключевой особенностью известных исследований в данном направлении является использование соответствующей модели вычислений и языка для описания исходных алгоритмов функционирования ЦИС.

1. Известные решения

В настоящее время наблюдается устойчивая тенденция повышения уровня абстракции исходных алгоритмов от конечной реализации проекта. При этом развитие способов описания архитектуры ЦИС идет несколькими путями. Наиболее распространенным является введение в существующие языки описания аппаратуры (HDL) конструкций для высокоуровневого описания [5]. Такие решения привели к появлению на базе классического Verilog языка SystemVerilog [5]. Но, как и VHDL, SystemVerilog позволяет описывать системы на алгоритмическом уровне и применяется только на низких уровнях абстракции, где имеется привязка программы к определенной архитектуре, что обуславливает наличие семантического разрыва между представлениями схем на высоких уровнях и воплощении в кристалл.

Существует ряд решений, поддерживающих использование адаптированных, императивных языков программирования высокого уровня (в первую очередь C и C++) в качестве языков описания аппаратуры, например, SystemC [6], Handel-C [7] и Impulse-C [8]. Но стоит отметить, что эти языки создавались для решения узких задач, например, для реализации потоковых приложений или для поддержки альтернативных моделей программирования, и при моделировании и отладке системы происходит жесткая привязка к конкретному архитектурному решению.

Особый интерес вызывает язык программирования COLAMO [1], который является языком высокого уровня с неявным описанием параллелизма, а распараллеливание достигается с помощью объявления типов доступа к переменным и индексации элементов массивов, что характерно для языков потока данных. В настоящее время этот язык применяется для программирования реконфигурируемых вычислительных систем и позволяет создавать параллельные прикладные программы с высокой удельной производительностью [1]. Однако этот язык ориентирован на решение прикладных задач в области высокопроизводительных вычислений для многокристальных систем.

Для описания ЦИС есть возможность использования и функциональных языков, которые имеют ряд преимуществ по сравнению с императивными языками. Во-первых, более мощный механизм абстракции позволяет таким языкам справляться с решением сложных задач. Во-вторых, система типов современных функциональных языков обеспечивает повышенную надежность вычислений. Исходные алгоритмы ЦИС, описанные на подобных языках, более лаконичны, проще поддаются преобразованиям и верификации [9]. Например, в Hydra [10] используются потоки для описания сигналов и рекурсивных выражений для обеспечения преобразований в схеме. Для описания схем на логическом уровне удобно использовать язык Wired [11]. Последняя разработка программистов для этого языка — библиотека Haskell — за счет использования отношений вместо функций обеспечивает более простое создание логических типов данных, которым могут быть поставлены в соответствие некоторые паттерны из библиотеки. Получающиеся описания схем сильно параметризованы, что поднимает вопрос о том, как предоставить общие методы проверки [11]. Язык

Chisel [12], являясь встроенным в язык Scala, позволяет применить всю мощь объектно-ориентированного и функционального программирования к разработке логических схем. Программы, описанные на Chisel, могут быть преобразованы на Verilog. Однако вследствие нечитаемости сгенерированного на Verilog кода зачастую затруднена его отладка.

Также как и Hydra, язык Lava [13] является встроенным подмножеством языка Haskell и имеет мощные средства для проектирования схем, полученные от своего предшественника. При этом благодаря расширенной системе типов для описания аппаратуры Lava является более простым и удобным в использовании. Однако свойственный Haskell механизм "ленивых" вычислений и последовательная структура списков накладывают ограничения на преобразования параллелизма и не позволяют эффективно осуществить автоматическое распараллеливание программ, т. е. добиться полной "свободы" при преобразовании параллелизма с учетом требуемых ограничений.

Таким образом, задача обеспечения эффективной переносимости исходного описания ЦИС может быть решена при использовании функциональных языков и описания алгоритмов в виде графа потока данных, что показано в работах Доннагары [2, 14]. На примере параллельного программного обеспечения (PaRSEC) для высокопроизводительных вычислений на практике показана высокая эффективность данного подхода [2].

В этом случае на этапах высокоуровневого синтеза решение лежит в применении определенной парадигмы программирования, которая должна соответствовать следующим условиям [2]:

1) отсутствие явного управления вычислениями;

2) поддержка модели потока данных;

3) параллелизм на уровне операций.

Такая модель с параллелизмом на уровне операций для обработки потока данных, без использования явного управления вычислениями лежит в основе ФПП языков [3]. Это дает основание считать методы ФПП программирования и соответствующую модель вычислений наиболее соответствующей решаемым задачам высокоуровневого, архитектурно-независимого синтеза СБИС.

Среди функциональных языков, поддерживающих ФПП программирование, выделен язык ПИФАГОР [3], который обеспечивает исходное описание с бесконечными ресурсами, поддерживает модель потока данных и параллелизм на уровне операций, что является наиболее значимым для организации процесса архитектурно-независимого, сквозного проектирования ЦИС. В этом языке архитектурная независимость достигается за счет описания в программе только информационных связей. В отличие от базовых возможностей Haskell, ПИФАГОР поддерживает только явное описание задержанных вычислений. Это позволяет не допускать выполнения альтернативных фрагментов программы до того момента, пока они не потребуются. Кроме того, в ПИФАГОРе отсутствуют операторы цикла, что позволяет избежать конфликтов при использовании различных данных одних и тех же фрагментов параллельной программы. Теоретически это позволяет начинать выполнение любых функций сразу же по готовности их данных. Практически же на максимальный параллелизм, задаваемый языком, накладываются только ресурсные ограничения. Примеры кода, демонстрирующие реализацию мультиплексора 2 к 1 на языках ПИФАГОР и Verilog, представлены в листингах 1 и 2.

```
// Функция, реализующая логический элемент "Не"
Not << funcdef Param {
  return << Param: -; }
// Функция, реализующая логический элемент "2-И"
And2 << funcdef Param {
   return << Param: *; }
// Функция, реализующая логический элемент "ИЛИ "
Or2 << funcdef Param {
  return << Param: +; }
MUX2 _ 1 << funcdef Param { // Основная функция,
реализующая мультиплексор 2 к 1
X0 << Param: 1; // Первый информационный вход
X1 << Param: 2; // Второй информационный вход
A << Param: 3; // Адресный вход
Y << ((X0, A:Not):And2, (X1,A):And2):Or2; // подсчет
выходного сигнала
return << Y; } // вывод результата работы программы
```

Листинг 1. Программа "Мультиплексор 2 к 1" на языке ПИФАГОР

```
module m21(D0, D1, S, Y);
output Y;
input X0, X1, A;
assign Y = (A)?X1:X0; // подсчет выходного сигнала
endmodule
```

Листинг 2. Программа "Мультиплексор 2 к 1" на языке Verilog

На основании ФПП модели вычислений и поддерживающего ее языка ПИФАГОР авторами предложен метод архитектурно-независимого высокоуровневого синтеза ЦИС [3]. При реализации метода динамическая система типов была заменена на статическую, которая поддерживается в языках описания аппаратуры, исключены

задержанные вычисления и предложен метод их преобразования при переходе на целевую платформу ЦИС. Также в модифицированный ПИФАГОР введен механизм преобразования рекурсии в итеративную схему с последующим переходом к конвейерной схеме [4].

Для обеспечения архитектурной независимости при разработке метода высокоуровнего синтеза используется промежуточное представление программы на языке ПИФАГОР в виде информационного графа (ИГ), задающего информационные связи, и управляющего графа (УГ), использование которого в явном виде позволяет более детально описывать процесс управления вычислениями. Информационный граф формируется в ходе трансляции программы, а УГ может порождаться как динамически, как и статически. Последний способ используется для перехода от программы на ПИФАГОРе к программе описания ЦИС на HDL-языке.

При трансляции построение промежуточного представления ФПП программы осуществляется в два этапа. На первом этапе выполняется трансляция исходного текста в ИГ. Затем по полученному ИГ формируется УГ. Управляющий граф может формироваться по ИГ различными способами, например, в соответствии с принципами управления потоками данных модели вычислений, или сводиться к последовательному обходу ИГ, либо обеспечивать другую стратегию управления вычислениями.

Для перехода на платформу ЦИС выполняется преобразование ИГ к конвейерной схеме. Конвейерная схема вычислений представляет собой ярусно-параллельную форму (ЯПФ) информационного графа, в которой операторы, исполняющиеся независимо друг от друга (фактически параллельно), располагаются на одном уровне (ярусе). В данном случае длина конвейера показывает степень параллелизма решаемой задачи.

Для решения задачи преобразования параллелизма под конкретные ресурсные ограничения выполняются следующие шаги: определение границ изменения параллелизма, реализация алгоритма изменения параллелизма и оценка результата по ресурсным ограничениям.

При переходе от неограниченного параллелизма ФПП модели на целевую платформу решается основная задача — редукция (свертка) параллелизма. Под сверткой параллелизма следует понимать процесс сжатия или сокращения параллелизма. Это является ключевым моментом всех проводимых преобразований в архитектурно-независимом методе синтеза ЦИС. Для реализации механизма редукции был введен новый метаслой, называемый "HDL-графом". Он представляет собой дополнительный слой для внесения изменений в ФПП модель. По мнению авторов, термин "HDL-граф" наиболее полно соответствует методу синтеза ЦИС. HDL-граф позволяет указывать связи между элементами списков связанных вершин, что в свою очередь позволяет выполнять оптимизационные преобразования с помощью вычисления операции выбора данных из списка. С помощью введенного дополнительного слоя при обработке типов в модели со статической типизацией вводится ограничение на динамическое изменение размерности списков во время вычислений.

Рекурсивные вычисления зачастую становятся препятствием при переносе таких программ на некоторые реальные вычислительные платформы, так как длительные повторяющиеся вычисления порождают переполнение памяти. А это в первую очередь критично для СБИС с ограничениями по объему встроенного ОЗУ, а также для СБИС, где не используется блочная память или применяется память, реализуемая на триггерах. Введение HDL-графа позволило решить эту проблему путем преобразования таких вычислений в итеративные с использованием хвостовой рекурсии и заданием глубины рекурсии на этапе трансляции.

2. Преобразование параллелизма

2.1. Алгоритм редукции параллелизма

Основной особенностью разрабатываемого метода синтеза является переход от распараллеливания задачи под конкретную архитектуру к методу свертки параллелизма под конкретные ресурсные ограничения платформы ЦИС. Такой переход выполняется из исходного максимально-параллельного описания алгоритма. Как показано в работах [2, 14], это обеспечивает переносимость параллельных алгоритмов на различные платформы. Основной особенностью методов редукции параллелизма по сравнению с методами распараллеливания является существенное сокращение числа шагов для получения конечного результата. При редукции (сжатии) максимально-параллельного ИГ число максимально допустимых вариантов преобразований задается на этапе синтеза. В процессе синтеза решаются следующие задачи:

- оценка ресурсов G_k получаемого архитектурного решения;
- оценка производительности получаемого решения;

- расчет коэффициента редукции по каждому классу ресурсов;
- редукция параллелизма схемы для достижения требуемых коэффициентов.

Для оценки ресурсов используется промежуточное представление программы — HDL-граф, в котором уже заданы архитектурно-зависимые данные. HDL-граф представляет собой ациклический граф в ЯПФ, в каждом узле которого заданы типы и разрядности данных [4]. В зависимости от целевой платформы определяются классы ресурсов N_k, по которым будет оцениваться схема.

Например, для платформы ПЛИС к основным классам ресурсов можно отнести:
- число регистров N_r;
- число логических ячеек N_{lc};
- число блоков памяти N_m;
- число арифметических и иных специализированных вычислительных блоков N_{dsp}.

В данном случае для свертки параллелизма ключевыми моментами являются оценка ресурсов памяти и оценка вычислительных ресурсов.

Ресурс памяти представляет собой множество $N_{mem} = \{N_r, N_m\}$, а вычислительный ресурс N_{comp} является множеством $\{N_{lc}, N_{dsp}\}$.

Внутри подмножества существуют ограничения для взаимозаменяемых ресурсов. Для ресурсов памяти любое хранение данных может быть реализовано на блочной памяти, а на триггерах оно реализуется с ограничениями по объему. В случае вычислительных ресурсов для операции любого типа можно реализовать схему на логических ячейках, при этом тип и набор операций для специализированных блоков ограничен.

Результаты оценки ресурсов используются для дальнейшей редукции параллелизма.

2.2. Оценка ресурсов памяти

Для оценки требуемой памяти суммарный объем ресурсов может быть приведен к общему объему, измеряемому в бит/Кбит.

В качестве примера для оценки ресурса схемы рассмотрим ее HDL-граф. Каждый k-й слой ЯПФ состоит из набора информационных входов B_k и набора операций O_k. Каждый информационный вход вершин HDL-графа после типизации имеет разрядность W_k. Исходя из числа входов и разрядности каждого входа для конкретного слоя графа, можно определить количество ресурсов памяти, требуемых для хранения результата в соответствующем состоянии графа:

$$NR_k = \sum (W_k * B_k).$$

Для расчета ресурса выполняется обход графа и суммирование разрядностей входов и выходов всех вершин.

После первоначальной оценки требуемого ресурса памяти для исходной максимально-параллельной реализации схемы возможны следующие ситуации:

— требуемый ресурс меньше доступного $NR < N_{m/lc}$ ($N_{m/lc}$ — доступный ресурс);

— требуемый ресурс больше доступного $NR \geqslant N_{m/lc}$.

Первый вариант не требует расчета коэффициента редукции по ресурсам памяти G_m, но при этом при изменении схемы в процессе ее редукции по другим ресурсам оценку и проверку ресурса памяти необходимо проводить вновь.

Во втором случае коэффициент редукции NM рассчитывается согласно формуле

$$G_m = \frac{NR}{N_{m/lc}}.$$

Этот коэффициент используется в алгоритме редукции параллелизма схемы.

Помимо ограничения на объем памяти, необходимо учитывать также и ограничение на производительность памяти (ширина интерфейса данных). Если память построена на регистрах/триггерах, то данные отсутствуют, так как ширина шины данных равна объему данных. Для блочной памяти объем считываемых за такт данных меньше, чем объем хранимых данных. В случае, если ресурс памяти NR, требуемый для хранения результата решения конкретной проблемы, превышает доступный объем регистров N_r, необходимо рассчитывать коэффициент редукции по интерфейсу памяти G_{md}, иными словами, сократить параллелизм на G_{md}.

2.3. Алгоритм определения коэффициента редукции

Для определения минимально-необходимого коэффициента редукции по интерфейсу памяти из всех стадий конвейера выбирается набор стадий с максимальным суммарным реализуемым на регистрах интерфейсом. Для этого из множества стадий конвейера выбирается подмножество стадий, такое что:

$$\max \sum NR \| \sum NR_k < \left(N_{m/lc} - NR_r\right),$$

где $\|$ — логическое сложение.

Исходя из изложенного выше, коэффициент G_{md} определяется как отношение суммарного интерфейса памяти оставшихся стадий к суммарному интерфейсу блочной памяти IM.

Для определения коэффициента редукции по интерфейсу памяти используется следующий алгоритм.

Шаг 1. Выбрать подмножество стадий конвейера, такое что сумма ресурсов NR_k выбранных стадий будет меньше NR_r и сумма выбранных значений NR_k будет максимальна.

Шаг 2. Рассчитать ресурс памяти, реализуемый на блочной памяти/памяти с последовательным доступом согласно формуле

$$NR_m = NR - \sum NR_k,$$

где k принадлежит подмножеству стадий конвейера, выбранных на шаге 1.

Шаг 3. Рассчитать коэффициент NR_r согласно формуле

$$NR_r = 1 + Int\left(NR_m / IM\right),$$

где Int — округление до целого значения.

Рассмотрим в качестве примера вычисление 4-точечного БПФ (быстрого преобразования Фурье), где дискретное преобразование Фурье вычисляется как [15]

$$a_n = \frac{1}{n}\sum_{j}^{n-1} S_j * W_n^{-kj} + \frac{1}{n}\sum_{j}^{n-1} S_j * W_n^{kj},$$

где a_n — коэффициент ряда Фурье; n — число элементов ряда; S_j — измеренные значения сигнала; $W_n^{\pm kj} = \exp\left(-j\frac{2\pi}{n}nk\right)$ — фазовые множители (действительные (Re) и мнимые (Im) соответственно).

При этом тип входных данных — 16-битный знаковый.

Информационный граф после преобразования в HDL-граф и приведения в ЯПФ показан на рис. 2. На нем показано, что вычисление БПФ проводится в 4 стадии (Stages 1, 2, 3, 4). Все вычисления в рамках одной стадии выполняются параллельно. Количество ресурсов, необходимых для выполнения конкретных вычислений, приведено на каждой стадии для решения определенной подзадачи и обозначено как int16, int33 и т. п.

Значение ресурсов NR_k для каждой стадии конвейера рассчитывается следующим образом:

$$NR_1 = 10 * 2 * 16 = 320;$$

$$NR_2 = 16 * 4 + 8 * 32 = 320;$$

$$NR_3 = 16 * 4 + 4 * 32 = 196;$$

$$NR_4 = 33 * 4 + 4 * 34 = 268.$$

Общее значение ресурса $NR = 1104$ бита.

Рассмотрим две архитектуры — A_1 и A_2. Значение $N_{m/lc}$ для обеих архитектур равно 1536 бит. В архитектуре A_1 весь ресурс памяти находится в регистрах. Для такой архитектуры схема БПФ 4 в максимально-параллельной форме реализуется без изменений, как показано на рис. 2.

В архитектуре A_2 ресурс регистров составляет 512 бит, а 1024 бит представлены в виде блочной памяти с интерфейсом данных 36 бит. Значение суммарного интерфейса блочной памяти IR составит 36 бит.

В соответствии с описанным выше алгоритмом выберем подмножество стадий, реализуемых на регистрах и имеющих максимальный интерфейс. В данном примере это может быть либо стадия 1 (Stage 1), либо стадия 2 (Stage 2).

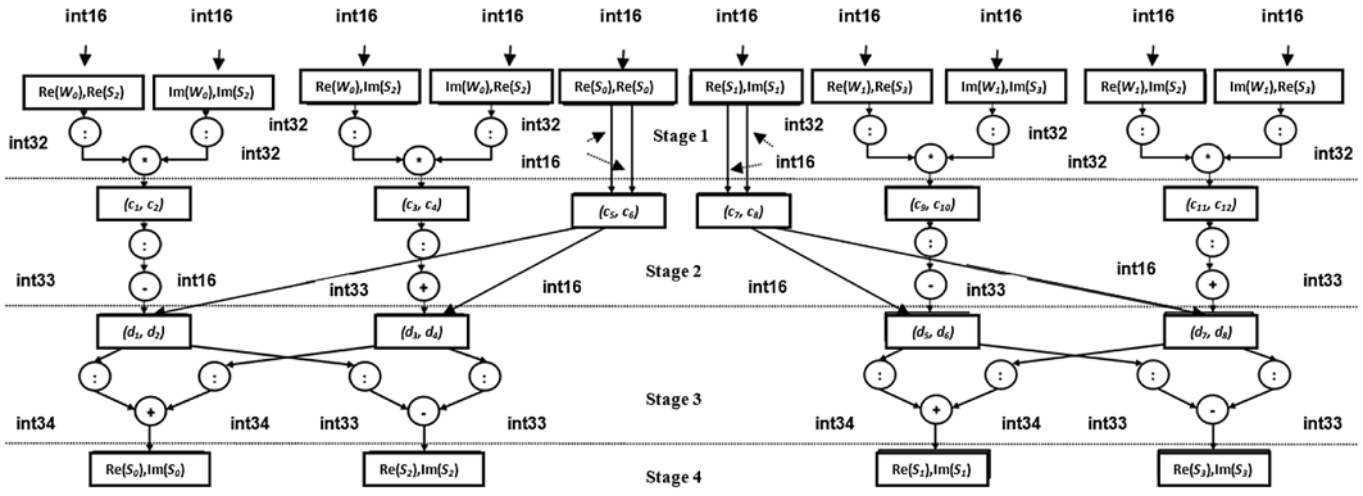

Рис. 2. HDL-граф максимально-параллельной формы для БПФ

Значение NR_m в таком случае будет рассчитываться следующим образом:

$$NR_m = NT = NR_1 1104 - 320 = 784 \text{ бита}.$$

Значение коэффициента редукции по интерфейсу памяти составит

$$G_{md} = \frac{784}{36} = 22.$$

В этом случае период подачи данных становится равным G_{md} и стадии конвейера 2, 3, 4 или 1, 3, 4 реализуются последовательно, так как результат записывается в один блок памяти. HDL-граф схемы после редукции на G_{md} приведен на рис. 3.

Запись значений $c_1 - c_{12}$, вычисление значений $d_1 - d_8$, $S_0 - S_3$ проводится в данной схеме последовательно. Так как стадии 2, 3 исходной схемы после редукции требуют 22 такта на выполнение, стадия 1 также может быть увеличена до 22 тактов (clock cycles) без влияния на общую производительность системы, что приведет к пропорциональному уменьшению вычислительного ресурса стадии 1.

2.4. Оценка вычислительных ресурсов

Вычислительный ресурс обозначим как N_{comp} — множество $\{N_{lc}, N_{dsp}\}$. В данном случае для операции любого типа можно реализовать схему на логических ячейках, при этом тип и набор операций для специализированных блоков ограничен.

Общее число слоев (стадий конвейера) равно M. На каждом j-м слое графа реализуется определенное множество операций O^j, где $j = 1, ..., k$.

Для всего HDL-графа число каждой операции

$$Fk = \sum_{j=0}^{M} O_k^j.$$

Обозначим общее число различных типов операций через L, $i = 0, ..., L$. Под типом операции понимается тип арифметической/логической и т. д. операции вместе с указанием разрядностей данных, например, сложение 16-битных данных, сравнение 20-битных данных и т. д.

После расчета общего числа операций каждого типа, исходя из доступного ресурса конкретной архитектуры, необходимо провести оценку степени параллелизма, с которой возможно реализовать схему.

Пусть количество ресурса для каждого конкретного типа операции известно. Обозначим Y — тип ресурса (логические ячейки, специализированные арифметические блоки DSP и т. д.), $V(Y)_k$ — количество ресурса типа Y, необходимое для реализации операции типа k. Тогда суммарный ресурс типа T для всех операций в HDL графе:

$$V(Y) = \sum_{k=0}^{L} V(Y)_k * F_k.$$

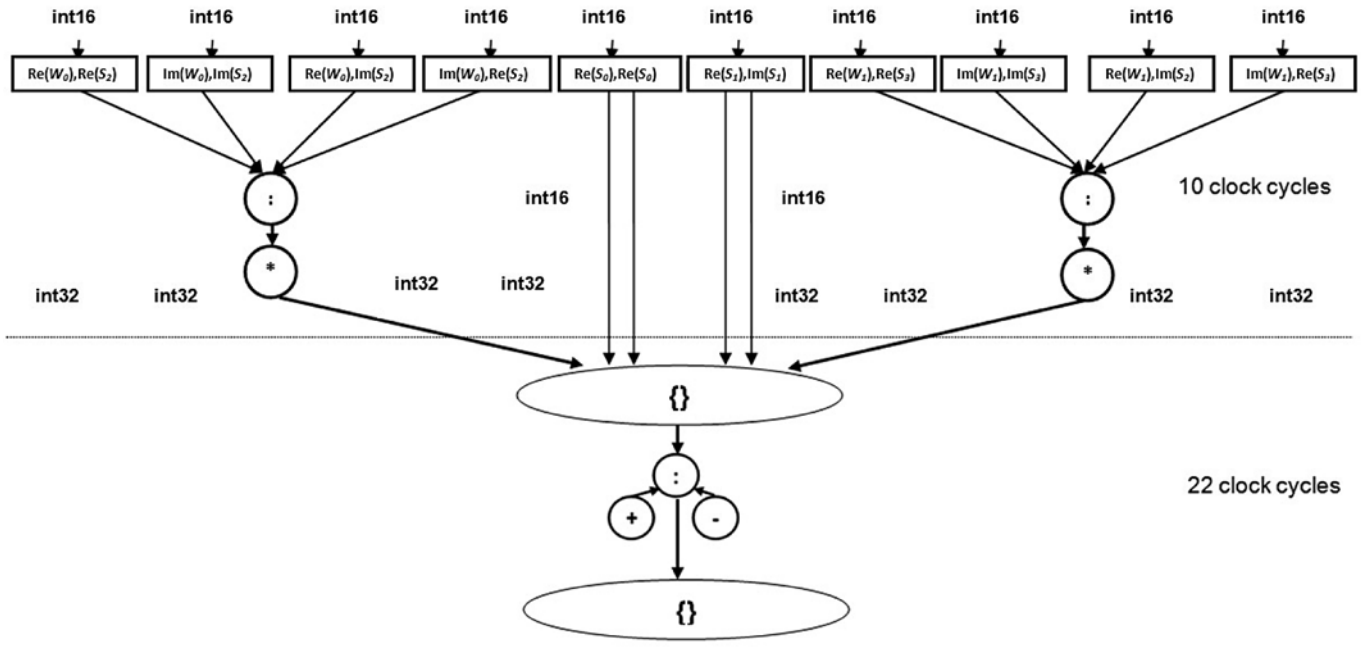

Рис. 3. HDL-граф после редукции

По каждому классу вычислительных ресурсов можно рассчитать коэффициент редукции G_y как отношение суммарного требуемого ресурса к ресурсу целевой архитектуры, округленное до ближайшего большего целого:

$$G_y = Int\left(V(Y)/N_y\right).$$

Итоговый коэффициент редукции по вычислительным ресурсам определяется как максимальный среди всех G_y:

$$G_{calc} = \max\{G_y\},$$

Рассмотрим данный алгоритм на примере графа, представленном на рис. 2. Общее число стадий конвейера данного графа 4. Число типов операций L составит 5: 16-битное умножение ($i = 0$), сложение и вычитание 33 и 16 бит ($i = 1, 2$), сложение 34-битных данных и вычитание 33-битных данных ($i = 3, 4$). Число операций по слоям графа составит

$$O_1 = \{8,\ 0,\ 0,\ 0,\ 0\};$$
$$O_2 = \{0,\ 2,\ 2,\ 0,\ 0\};$$
$$O_3 = \{0,\ 0,\ 0,\ 4,\ 4\};$$
$$O_4 = \{0,\ 0,\ 0,\ 0,\ 0\}.$$

Количество каждой операции k-го типа:

$$F_0 = 8,\ F_1 = 2,\ F_2 = 2,\ F_3 = 4,\ F_4 = 4.$$

Пусть имеется архитектура с двумя типами ресурсов для реализации вычислений: логические ячейки ($Y = 0$) и секции DSP ($Y = 1$). Значения ресурса для каждого типа операции возьмем следующие:

для логических ячеек получаем:

$$V(0)_0 = 0,\ V(0)_1 = 5,$$
$$V(0)_2 = 5,\ V(0)_3 = 10,\ V(0)_4 = 10;$$

для секций DSP получаем:

$$V(1)_0 = 1,\ V(1)_1 = 0,$$
$$V(1)_2 = 0,\ V(1)_3 = 0,\ V(1)_4 = 0.$$

Суммарный ресурс по каждому типу по всем операциям в графе составит

$$V(0) = \sum_{k=0}^{L} V(Y)_k * F_k = 0*8 + 5*2 + 5*2 + 4*10 + 4*10 = 100,$$

$$V(1) = \sum_{k=0}^{L} V(Y)_k * F_k = 1*8 + 0*2 + 0*2 + 0*10 + 0*10 = 8.$$

Рассмотрим архитектуру, где число DSP $N_1 = 2$, число логических ячеек $N_0 = 400$. Тогда коэффициент редукции по каждому типу ресурсов составит

$$G_0 = Int\left(\frac{V(0)}{G_0}\right) = \frac{100}{400} = 0,25,$$

$$G_1 = Int\left(\frac{V(1)}{G_1}\right) = \frac{8}{2} = 4.$$

Значение коэффициента редукции для вычислительных ресурсов G_{calc} равно максимальному из всех G_y, следовательно коэффициент G_{calc} равен

$$G_{calc} = \max\{0,25,\ 4\} = 4.$$

После редукции каждой стадии с коэффициентом 4 задержка каждой стадии вырастет до 4 тактов, ресурс при этом снизится также в 4 раза. Результирующий граф показан на рис. 4.

Отметим, что при изменении вычислительного ресурса может измениться ресурс памяти.

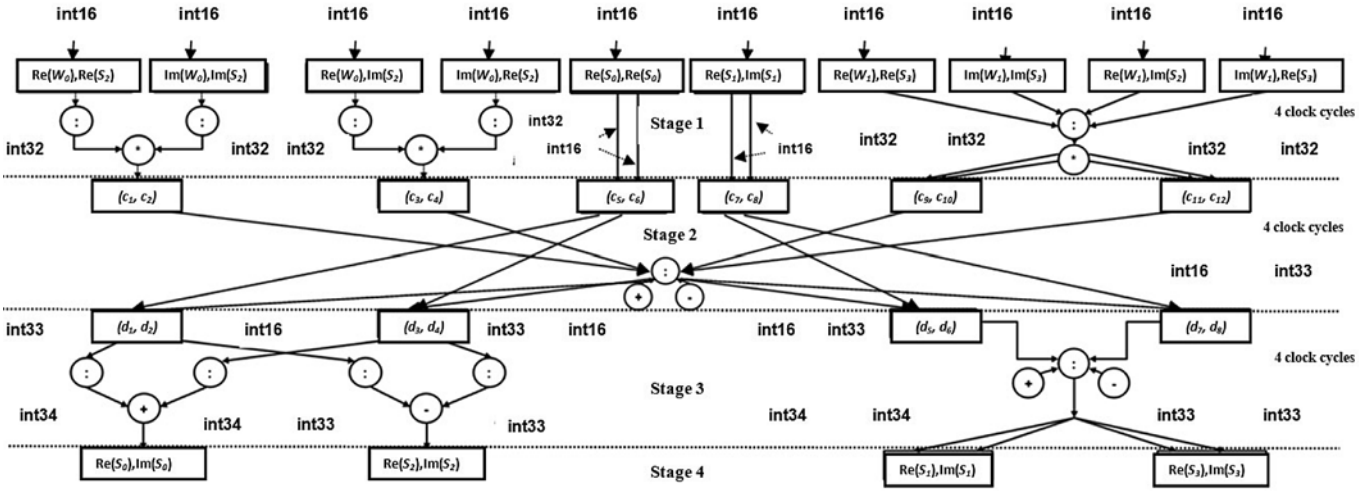

Рис. 4. HDL-граф после редукции на G_{calc}

2.5. Обобщенный алгоритм преобразования параллелизма

Пусть отношение ресурса, требуемого для реализации схемы в исходном максимально-параллельном виде к доступному ресурсу, обозначается как R. Предполагается, что доступный ресурс — это наименьший из ресурсов. При решении задачи трансформации параллелизма в общем виде в зависимости от доступного ресурса целевой платформы можно выделить три варианта преобразования исходной максимально параллельной схемы.

Первый вариант: если $R > 1$, то необходима редукция параллелизма.

Здесь возможно увеличение производительности путем размещения нескольких схем параллельно:

$$S = Int(1/R).$$

Рассмотрим алгоритм редукции с использованием коэффициентов редукции, описанных в подразд. 2.2/2.4.

Алгоритм состоит из следующих шагов.

Шаг 1. Рассчитываем коэффициенты редукции N_{calc} и G_m.

Шаг 2. Выбираем максимальный коэффициент $N_{\max} = \max(G_m, N_{calc})$.

Шаг 3. Редуцируем параллелизм схемы на $N_{\max}$.

Шаг 4. Для измененной схемы пересчитываем коэффициенты G_m и G_{calc}, если они меньше 1 — завершение алгоритма.

Шаг 5. Если какие-либо операции возможно реализовать с использованием другого типа ресурсов — изменяем данные операции на другой тип ресурсов без изменения коэффициента R и пересчитываем коэффициенты G_m и G_{calc}.

Шаг 6. Если какой-либо из коэффициентов больше 1: $G_{\max} = 1 + G_{\max}$ и возвращаемся на шаг 3.

Последовательное увеличение коэффициента редукции позволяет подобрать минимально возможный коэффициент для удовлетворения требований по ресурсам и достижения при этом максимальной производительности (минимально возможной редукции по производительности относительно исходного максимально параллельного варианта).

Второй вариант преобразования исходной максимально параллельной схемы:

при $R < 1/2$ возможно увеличение числа схем.

Третий вариант преобразования исходной максимально параллельной схемы:

$1/2 < R < 1$, ресурса достаточно для размещения одного максимально параллельного варианта схемы.

Во втором и третьем случаях никаких преобразований максимально параллельной схемы не требуется.

3. Практические результаты

В рамках проводимых исследований авторами реализован набор программных инструментальных средств, позволяющих:

— выполнять преобразование исходного кода на ФПП языке в промежуточное представление в виде информационного и управляющего графов;

— осуществлять оптимизационные преобразования, что позволяет повысить эффективность ФПП программ;

— проводить отладку и анализ ФПП кода во время выполнения, обеспечивая поиск ошибок и трассировку;

— осуществлять трансляцию промежуточного представления ФПП программ в описание СБИС на HDL-языках [15].

На рис. 5 представлена архитектура разработанных инструментальных средств поддержки

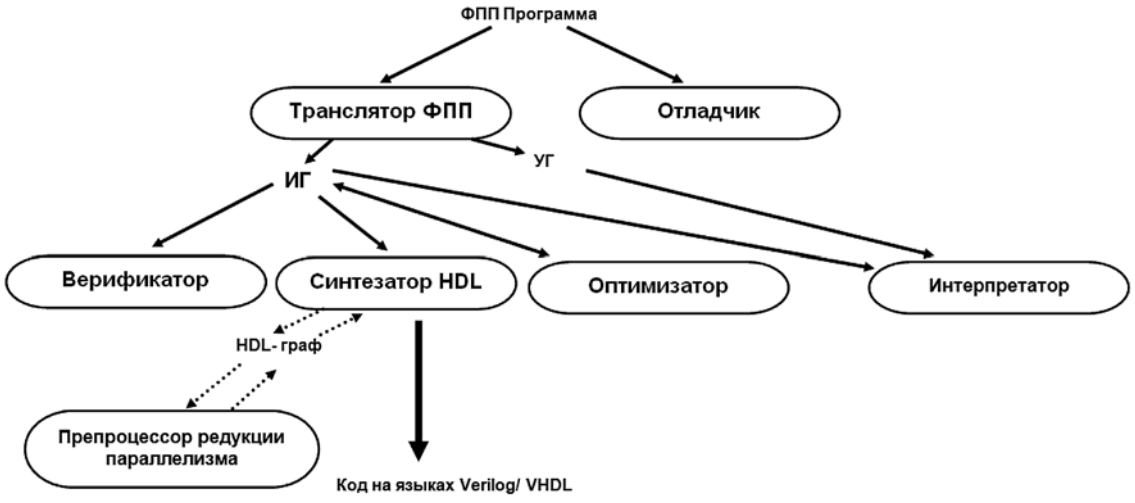

Рис. 5. Архитектура инструментальных средств поддержки проектирования

проектирования на основе предложенного метода высокоуровневого синтеза.

Интерпретатор позволяет исполнить текст программы на ФПП языке. Входными данными для интерпретатора являются информационный и управляющий графы, а также аргумент функции верхнего уровня. Аргумент представляется в формате описания ИГ, для этого он обрабатывается транслятором.

Оптимизатор также использует входное промежуточное представление, осуществляет оптимизирующие преобразования над ИГ, результат сохраняется в виде промежуточного представления ИГ.

К оптимизирующим преобразованиям, осуществляемым на данном этапе, относятся:

- удаление неиспользуемого кода;
- оптимизация повторяющихся вычислений;
- непосредственная подстановка функций;
- удаление повторяющегося кода;
- оптимизация на основе эквивалентных преобразований алгебры ФПП модели.

Транслятор позволяет проводить проверку синтаксиса программ, разработанных на языке ФПП программирования ПИФАГОР, и преобразование программы в ее промежуточное представление в виде информационного и управляющего графов [16]. Он реализует механизм трансформации с функционально-потокового описания в описания на уровне комбинационных логических и автоматных схем. В состав транслятора входит отладчик, генератор ИГ и генератор УГ. Результатом функционирования транслятора является набор отлаженных функций, реализованных на языках Verilog/VHDL.

Пакет разработанных программных средств также включает препроцессор редукции параллелизма и препроцессор оценки ресурсов. Препроцессор редукции параллелизма позволяет проводить автоматические преобразования параллелизма программ, предназначенных для трансляции в HDL-язык с учетом ресурсных ограничений. Препроцессор использует в качестве входных данных промежуточное представление алгоритма СБИС на ФПП языке в виде типизированного ИГ (HDL-графа) и ресурсные ограничения целевой платформы, полученные при помощи препроцессора ресурсных ограничений. Результатом функционирования препроцессора редукции параллелизма является ИГ алгоритма, преобразованный с учетом ресурсных ограничений. Полученное представление используется синтезатором схем для получения описания схемы СБИС на HDL-языках.

Разработаны трансляторы для языков Verilog и VHDL [17]. Программа реализует проверку исходного описания на пригодность к синтезу, сборку исходного описания из множества функций, назначение типов данных в исходном описании и синтез выходного описания схемы на языках Verilog/VHDL.

Набор разработанных программных инструментов функционирует в составе интегрированной среды разработки и позволяет сформировать набор отлаженных функций для их реализации в виде СБИС. Оболочка предоставляет пользователю информационные ресурсы для организации всего процесса высокоуровневого синтеза СБИС на основе ФПП подхода.

Посредством разработанных инструментальных средств на практике получен ряд научно-технических решений, перечисленных далее.

- Комплект сложно-функциональных блоков однокристального устройства управления бортовой сети космического аппарата [18]. Программа, входящая в состав блоков, реализует бортовой коммуникационный модуль стандарта Space Wire, который представляет собой законченный приемопередающий узел бортовой сети космического аппарата и выполняет функции приемника, передатчика и контроллера канала между другими узлами сети.
- Сложно-функциональные блоки из состава СБИС спутникового модема ЯР-1040 [19, 20].
- СБИС блока ЦОС на базе БМК К5540ТН014А из состава навигационного прибора [21]. Программа, входящая в состав, реализуется на ПЛИС Xilinx Spartan-4 и предназначена для использования в составе приемника навигационных сигналов ГЛОНАСС/GPS и осуществляет первичную обработку сигналов на ПЛИС либо заказной микросхеме.

Заключение

Проведенный обзор современных языков и способов проектирования логических схем позволил обосновать выбор модели функционально-потоковых параллельных вычислений и языка ФПП программирования ПИФАГОР для разработки архитектурно-независимого метода синтеза интегральных схем.

В процессе разработки архитектурно-независимого высокоуровневого метода синтеза ЦИС, основанного на модифицированной модели ФПП, был предложен метод преобразования параллелизма, заключающийся в сокращении максимального параллелизма решаемой задачи при переходе к конкретной целевой архитектуре. Этот подход обеспечивает переносимость параллельных архитектур на различные платформы.

Алгоритм преобразования параллелизма включает оценку ресурсов, вычисление коэффициента редукции параллелизма по каждому классу ресурсов и редукцию параллелизма схемы для достижения требуемых характеристик конкретной целевой платформы.

Представленные методы редукции позволяют реализовать изменение параллелизма исходного описания алгоритма и обеспечить реализацию механизма переноса на различные архитектуры с различными ресурсными ограничениями.

В отличие от методов распараллеливания, предлагаемый в работе метод снижает сложность процесса переноса за счет уменьшения перебора количества вариантов реализации, получаемых в процессе синтеза под новые ресурсные ограничения. Вместе с тем методы оценки ресурса на высокоуровневом этапе требуют учета накладных расходов при изменении параллелизма к более последовательным схемам. Для данного случая необходимо увеличение точности оценки. Для этого могут использоваться нейронные сети и методы машинного обучения, которые на основе параметров оценки схемы на высокоуровневом этапе могут с необходимой точностью предсказать значения параметров схемы после реализации на низком уровне.

Методы прямого подсчета ресурса результата реализации схемы осложнены по причине множества преобразований, которые происходят при реализации схемы на этапах синтеза и имплементации на низкоуровневом этапе предлагаемого метода. Реализация точных методов оценки ресурса позволит дополнительно уменьшить количество вариантов схемы, рассматриваемых в процессе синтеза под ресурсные ограничения.

На основе предложенных методов редукции параллелизма и оценки ресурсов реализован ряд инструментальных средств, таких как транслятор архитектурно независимого описания автоматных и комбинационных схем, препроцессоры оценки ресурсов и редукции параллелизма. Эти препроцессоры позволяют рассчитывать и количественно оценивать требуемый ресурс схемы по выбранным типам элементарных блоков (ячеек, памяти, триггеров, LUT-таблиц и т. д.) на поддерживаемых целевых платформах.

Список литературы

1. **Левин И. И., Гудков В. А.** Расширение языка высокого уровня COLAMO для программирования реконфигурируемых вычислительных систем на уровне логических ячеек ПЛИС // Вестник компьютерных и информационных технологий. 2010. № 12. С. 10—17.

2. **Dongarra J., Bosilca G., Bouteiller A.** et al. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability// IEEE Computing in Science and Engineering. 2013. Vol. 15, No. 6. P. 36—45.

3. **Легалов А. И.** Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. Том 10, № 1. С. 71—89.

4. **Nepomnyashchy O. V., Legalov A. I., Tyapkin V.** et al. Methods and algorithms for a high-level synthesis of the very-large-scale integration// WSEAS Transactions on Computers. 2016. No. 15. P. 239—247.

5. **IEEE** Std 1800-2012: IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language, 2013.

6. **Алехин В. А.** SystemC. Моделирование электронных систем. М.: Горячая линия — Телеком, 2018. 320 с.

7. **Vivado** Design Suite User Guide. High-Level Synthesis. UG902—Xilinx—2015. URL: https://www.yumpu.com/en/document/view/7034022/xilinx-vivado-design-suite-user-guide-high-level-synthesis-ug902

8. **Handel**-C Language Reference Manual. Celoxica Limited, 2005. 350 p.

9. **Sérot J., Michaelson G.** Compiling Hume down to gates// Draft Proceedings of 11th International Symposium on Trends in Functional Programming. Madrid, 2011. P. 191—226.

10. **O'Donnell J., Rünger G.** Derivation of a logarithmic time carry lookahead addition circuit// Journal of Functional Programming. 2004. Vol. 14, No. 6. P. 697—713.

11. **Axelsson E., Ciaessen K., Sheeran M.** Wired: wire-aware circuit design// Proceedings of the Conference on Correct Hardware Design and Verification Methods (CHARME '05). 2005. Vol. 3725. P. 5—19.

12. **Bachrach J., Vo H., Richards B.** et al. Chisel: Constructing hardware in a Scala embedded language// DAC Design Automation Conference 2012. 2012. P. 1212—1221.

13. **Gill A., Bull T., Kimmell G.** et al. Introducing Kansas Lava// Implementation and Application of Functional Languages, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Sep. 2009. P. 18—35.

14. **Dongarra J., Danalis A., Bosilca G.** et al. PTG: An Abstraction for Unhindered Parallelism// Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing, 2014. P. 21—30.

15. **Нуссбаумер Г.** Быстрое преобразование Фурье и алгоритмы вычисления сверток. М.: Радио и связь, 1985. 248 с.

16. **Nepomnyashchiy O. V., Ryzhenko I. N., Shaydurov V. V.** et al. The VLSI High-Level Synthesis for Building Onboard Spacecraft Control Systems// Proceedings of the Scientific-Practical Conference "Research and Development — 2016". Springer, Cham. 2017. P. 229—238.

17. **Непомнящий О. В., Романова Д. С., Рыженко И. Н., Легалов А. И.** Транслятор архитектурно-независимого описания автоматных и комбинационных схем. Свидетельство о государственной регистрации ПО для ЭВМ № 2021610682, 01.02.2021.

18. **Комаров А. А., Рыженко И. Н., Непомнящий О. В.** Программа синтеза описания схем на языках описания аппаратуры HDL с языка функционально-параллельного программирования "ПИФАГОР". Свидетельство о государственной регистрации ПО для ЭВМ № 2015619175, 26.08.2015.

19. **Непомнящий О. В., Комаров А. А., Рыженко И. Н.** и др. Программа драйвера бортовой сети космического аппарата. Свидетельство о государственной регистрации ПО для ЭВМ № 2015616896, 25.06.2015.

20. **Рыженко И. Н., Комаров А. А., Андреев А. С.** Программное обеспечение спутникового модема "ЯР-1040". Свидетельство о государственной регистрации ПО для ЭВМ № 2015614726, 27.04.2015.

21. **Непомнящий О. В., Комаров А. А., Рыженко И. Н.** Сложно-функциональный блок понижающего сумматора-ограничителя. Свидетельство о государственной регистрации ПО для ЭВМ № 2016619714, 26.08.2016.

Parallelism Reduction Methods in the High-Level VLSI Synthesis Implementation

D. S. Romanova, daryaooo@mail.ru, Siberian Federal University, Krasnoyarsk, 660041, Russian Federation, Krasnoyarsk State Agrarian University, Krasnoyarsk, 660049, Russian Federation,
O. V. Nepomnyashchiy, Siberian Federal University, Krasnoyarsk, 660041, Russian Federation,
A. I. Legalov, legalov@mail.ru, National Research University — Higher School of Economics, Moscow, 101000, Russian Federation,
I. N. Ryzhenko, rodgi.krs@gmail.com, **N. Y. Sirotinina,** nsirotinina@sfu-kras.ru, Siberian Federal University, Krasnoyarsk, 660041, Russian Federation

Corresponding author:

Romanova Darya S., Postgraduate Student, Krasnoyarsk State Agrarian University, Krasnoyarsk, 660049, Russian Federation
E-mail: daryaooo@mail.ru

Received on July 14, 2021
Accepted on April 15, 2022

The problems and solutions in the field of ensuring architectural independence and implementation of digital integrated circuits end-to-end design processes are considered. The paper focuses on the need to find a solution to the problem of program portability during the development of integrated circuits. A review of the main software and tools used to design digital circuits (Verilog, System-C, Handel-C, Lava, Hydra, Wired, COLAMO, Chisel and etc.) is presented. The method and language of parallel programming for functional flow synthesis of design solutions PIFAGOR is presented. The example of the source and generated code in the PIFAGOR and Verilog languages is given. During the method implementation, the tasks of reducing parallelism and estimating the occupied resources were highlighted. The main feature of the developed method is the introduction of the additional layer (HDL graph) into the synthesis process. Algorithms for the parallelism reduction have been developed. This method is demonstrated on the example of parallelism reduction while going to the FPGA platform solving the problem of calculating a 4-point FFT (Fast Fourier Transform). As part of the solution of this task, an assessment of memory resources and an assessment of computing resources were carried out. The results of software tools development for design support including the parallelism reduction preprocessor and resource estimation preprocessor and practical VLSI projects are presented.

Keywords: *integrated circuit, parallel computing model, algorithm, program, high-level synthesis, functional-stream language*

For citation:

Romanova D. S., Nepomnyashchiy O. V., Legalov A. I., Ryzhenko I. N., Sirotinina N. Y. Parallelism Reduction Methods in the High-Level VLSI Synthesis Implementation, *Programmnaya Ingeneria,* 2022, vol. 13, no. 6, pp. 259—271.

DOI: 10.17587/prin.259-271

References

1. **Levin I. I., Gudkov V. A.** Extension of the high-level language COLAMO for programming reconfigurable computing systems at the level of logical FPGA cells., *Vestnik komp'juternyh i informacionnyh tehnologij,* 2010, no. 12, pp. 10—17 (in Russian).

2. **Dongarra J., Bosilca G., Bouteiller A., Danalis A., Faverge M., Herault T.** PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability. *IEEE Computing in Science and Engineering*, 2013, vol. 15, no. 6, pp. 36—45.

3. **Legalov A. I.** A functional language for creating architecture-independent parallel programs, *Vychislitel'nye tehnologii*, 2005, vol. 10, no. 1, pp. 71-89 (in Russian).

4. **Nepomnyashchy O. V., Legalov A. I., Tyapkin V., Ryzhenko I. N., Shaydurov V. V.** Methods and algorithms for a high-level synthesis of the very-large-scale integration, *WSEAS Transactions on Computers,* 2016, no. 15, pp. 239—247.

5. **IEEE** Std 1800-2012: IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language, 2013.

6. **Alekhin V. A.** *SystemC. Simulation of electronic systems.* Moscow, Goryachaya liniya — Telecom, 2018, 320 p. (in Russian).

7. **Vivado** Design Suite User Guide. High-Level Synthesis. UG902—Xilinx—2015, available at: https://www.yumpu.com/en/document/view/7034022/xilinx-vivado-design-suite-user-guide-high-level-synthesis-ug902

8. **Handel-C** Language Reference Manual / Celoxica Limited, 2005.

9. **Sérot J., Michaelson G.** Compiling Hume down to gates, *Draft Proceedings of 11th International Symposium on Trends in Functional Programming,* Madrid, 2011. pp. 191—226.

10. **O'Donnell J. , Rünger G.** Derivation of a logarithmic time carry lookahead addition circuit. *Journal of Functional Programming*, 2004, vol. 14, no. 6, pp. 697—713.

11. **Axelsson E., Ciaessen K., Sheeran M.** Wired: wire-aware circuit design, *Proceedings of the Conference on Correct Hardware Design and Verification Methods (CHARME '05),* 2005, vol. 3725, pp. 5—19.

12. **Bachrach J., Vo H., Richards B.** et al. Chisel: Constructing hardware in a Scala embedded language, *DAC Design Automation Conference 2012*, 2012, pp. 1212—1221.

13. **Gill A., Bull T., Kimmell G.** et al. Introducing Kansas Lava, *Implementation and Application of Functional Languages, ser. Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, Sep. 2009, pp. 18—35.

14. **Dongarra J., Danalis A., Bosilca G.** et al. PTG: An Abstraction for Unhindered Parallelism, *Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, 2014, pp. 21—30.

15. **Nussbaumer G.** Fast Fourier Transform and Convolution Computing Algorithms. Moscow, Radio i svyaz', 1985, 248 p. (in Russian).

16. **Nepomnyashchiy O. V., Ryzhenko I. N., Shaydurov V. V., Sirotinina N. Y., Postnikov A. I.** The VLSI High-Level Synthesis for Building Onboard Spacecraft Control Systems, *Proceedings of the Scientific-Practical Conference "Research and Development — 2016". Springer*, Cham, 2017, pp. 229—238.

17. **Nepomnyashchy O. V., Ryzhenko I. N., Romanova D. S., Legalov A. I.** Translator of architecture-independent description of automata and combinational circuits, Certificate of state registration of software for computers No. 2021610682, 02/01/2021 (in Russian).

18. **Komarov A. A., Ryzhenko I. N., Nepomnyashchy O. V.** Program for synthesizing circuit descriptions in HDL hardware description languages from the Pifagor functional-parallel programming language, Certificate of state registration of software for computers No. 2015619175, 08/26/2015 (in Russian).

19. **Nepomnyashchy O. V., Komarov A. A., Ryzhenko I. N.** Program for the driver of the onboard network of the spacecraft, Certificate of state registration of software for computers No. 2015616896, 06/26/2015 (in Russian).

20. **Ryzhenko I. N., Komarov A. A., Andreev A. S.** Software of the satellite modem "YAR-1040", Certificate of state registration of software for computers No. 2015614726, 04/27/2015 (in Russian).

21. **Nepomnyashchy O. V., Komarov A. A., Ryzhenko I. N.** Complex-functional block of the lowering adder-limiter, Certificate of state registration of software for computers No. 2016619714, 08/26/2016 (in Russian).

ИНФОРМАЦИЯ