ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

На правах рукописи

Рыженко Игорь Николаевич

Методы, алгоритмы и программные инструменты архитектурно-независимого высокоуровневого синтеза однокристальных цифровых схем

2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
кандидат технических наук, доцент
Непомнящий Олег Владимирович

Красноярск – 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ....4
1 ТЕХНОЛОГИИ СИНТЕЗА СБИС....13
1.1 Методы и способы представления архитектурного описания СБИС....13
1.2 Маршруты проектирования СБИС....15
1.2.1 ТРАДИЦИОННЫЙ МАРШРУТ....15
1.2.2 ESL-ПРОЕКТИРОВАНИЕ....17
1.3 Формы представления архитектуры СБИС и модели вычислений....22
1.4 Языковые средства разработки СБИС....25
1.5 Выводы....31
2 МОДЕЛЬ И ЯЗЫК ДЛЯ МЕТОДА ВЫСОКОУРОВНЕВОГО СИНТЕЗА. .32
2.1 Модель вычислений на основе функционально потокового параллельного программирования....32
2.2.1 СИСТЕМА ТИПОВ В МОДИФИЦИРОВАННОЙ ФПП МОДЕЛИ....33
2.2.1.1 СКАЛЯРНЫЕ ТИПЫ ДАННЫХ....34
2.2.1.2 Векторные типы данных....35
2.2.1.3 Преобразование типов....35
2.2.1.4 Контроль и автоматическое назначение типов....37
2.2.2 Рекурсивные вычисления....38
2.2.3 Обработка списков и задержанных вычислений....41
2.3 Язык ФПП программирования для синтеза СБИС....44
2.3.1 Информационный граф....46
2.3.2 Управляющий граф....47
2.4 Стратегии управления вычислениями и преобразование параллелизма....50
2.4.1 Обобщенный алгоритм преобразования параллелизма....63
2.5 Выводы....64
3 ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ МЕТОДА ВЫСОКОУРОВНЕВОГО СИНТЕЗА....66
3.1 Инструментальная поддержка процесса разработки СБИС....66

3.2 Синтез описания СБИС.....69
3.3 Этапы высокоуровневого синтеза.....70
3.3.1 Разработка алгоритма на языке ФПП программирования.....72
3.3.2 Тестирование программного кода.....72
3.3.3 Синтез информационного и управляющего графов.....73
3.3.4 Типизация данных.....73
3.3.5 Оптимизация кода и промежуточного представления.....73
3.3.6 Синтез HDL-графа.....77
3.3.7 Синтез схемы обработки данных.....79
3.3.8 Синтез описания на HDL языке.....82
3.3.9 Синтез регистрово-вентильного представления СБИС.....85
3.4 Выводы по главе.....85
4 ПРАКТИЧЕСКИЕ РЕЗУЛЬТАТЫ И АНАЛИЗ АРХИТЕКТУРНЫХ РЕШЕНИЙ.....86
4.1 Применение инструментальных средств для разработки СБИС.....86
4.2 Критерии сравнения.....90
4.3 Методы синтеза для сравнения.....93
4.4 Тестовые задачи.....94
4.5 Результаты.....96
4.6 Выводы по главе.....102
ЗАКЛЮЧЕНИЕ.....103
СПИСОК СОКРАЩЕНИЙ.....106
ПРИЛОЖЕНИЕ А ЗАРЕГИСТРИРОВАННЫЕ РЕЗУЛЬТАТЫ ИНТЕЛЛЕКТУАЛЬНОЙ ДЕЯТЕЛЬНОСТИ.....121
ПРИЛОЖЕНИЕ Б АКТЫ ВНЕДРЕНИЯ.....122

Введение

Актуальность работы. Процесс совершенствования технологии производства кристаллов и появление новых технологических норм приводит к активному внедрению передовых достижений микроэлектроники. На рынок выходят технологии изготовления трехмерных кристаллов [11.], активно внедряются однокристальные, многопроцессорные системы, развивается производство сверхбольших интегральных схем (СБИС) с динамически реконфигурируемой архитектурой (ПЛИС) и т.д.

Развитие технологий, в свою очередь, порождает целый ряд проблем проектирования. Основной проблемой является несоответствие количества логических элементов на СБИС и возможностей проектирования и верификации. Количество элементов на СБИС для современных технологий производства превышает возможности проектирования в требуемые сроки.

Кроме того, при разработке СБИС решаются задачи:

– системного, алгоритмического и топологического проектирования;

– оптимизация алгоритмов и топологии при недостатке вычислительных ресурсов и времени разработки;

– оптимизация архитектуры при условии ограниченности времени и наличия экономических ограничений.

Для проектов СБИС важнейшими становятся проблемы тестирования готовых систем. Сегодня в маршруте проектирования верификация алгоритма функционирования достигает 70-90% от всего времени разработки СБИС. При этом остаются и остальные задачи проектирования: обеспечение надежности, функционирования в режиме реального времени, работа в тяжелых условиях эксплуатации и т.д.

На сегодняшний день в достаточной степени разработаны и автоматизированы низкоуровневые этапы создания проекта СБИС. Тем не менее,

большинство задач организации процесса проектирования и эффективного архитектурного проектирования СБИС остаются нерешенными.

В отличие от низкоуровневых этапов, при описании проекта на верхних уровнях иерархии определяется общесистемная организация процесса проектирования. Необходимо развитие маршрутов проектирования, основанных архитектурно-независимом подходе, реализующих комплексный подход на всех стадиях создания проекта.

Исследованиям методов проектирования СБИС с использованием высокоуровневого подхода посвящены работы И.А. Каляева, А.Л. Стемпковского, С.Г. Русакова, А.Н. Терехова, В.В. Топоркова, А.К. Кима, А.Е. Платунова, В.Г. Немудова, А.И. Легалова, из зарубежных специалистов в первую очередь следует отметить работы Д. Доннгары, А. Санджованни-Винсентелли, Е. Ли, А. Феррари, Г. Мартина, Г. Аха, А. Джеррайи и др.

Тем не менее, работы в области высокоуровневых подходов к проектированию СБИС не позволяют утверждать о создании эффективных методологий высокоуровневого проектирования. В большинстве методов проектирования СБИС проектируется под определенную архитектуру на основе стандартных библиотек и IP блоков с дальнейшей реализацией на конкретной архитектуре.

Таким образом, в области проектирования СБИС и перспективных высокоуровневых маршрутов проектирования СБИС существует ряд проблем:

– отсутствуют методы эффективного описания архитектурных решений для параллельной обработки данных, не зависящие от конечной архитектуры реализации;

– отсутствуют инструментальные средства, обеспечивающие эффективную реализацию архитектурно-независимого, высокоуровневого описания на различных целевых платформах;

– за редким исключением языки, применяемые для описания алгоритмов и архитектуры СБИС, либо предназначены для схемотехнического описания, либо ориентированы на традиционное последовательное программирование.

Существующие попытки решения этих проблем направлены в основном на устранение семантического разрыва между высокоуровневым и низкоуровневым описанием СБИС с использованием существующих универсальных языков программирования или разработкой языков от схемотехнических языков до высокоуровневых языков параллельного программирования, таких, как например COLAMO [2]. Такие подходы в основном реализуют решение задачи высокопроизводительных вычислений на реконфигурируемых платформах, таких как ПЛИС. Кроме того, эти инструментальные средства не нашли пока широкого применения и, как правило, не выходят за рамки академических проектов [65, 17, 60, 64].

Следовательно, разработка методов и программных средств, реализующих архитектурно-независимое описание однокристальных вычислительных систем, обладающих естественным параллелизмом, является актуальной задачей.

При этом требуется создание новых подходов к описанию СБИС на архитектурном уровне, позволяющих обеспечить максимальную абстракцию алгоритмов функционирования от архитектуры целевого кристалла и их представление на языках описания аппаратуры для адекватного отображения на нижние слои в иерархии проектирования. Такой подход обеспечит переносимость алгоритмов функционирования СБИС на различные целевые кристаллы.

Целью работы является повышение надежности и скорости разработки цифровых СБИС посредством разработки метода и инструментальных средств для высокоуровневого синтеза цифровых однокристальных систем.

Для достижения поставленной цели в работе решаются следующие **задачи**:

1) исследование моделей, методов, алгоритмов, языков и программных инструментов высокоуровневого проектирования сверхбольших интегральных схем;

2) выбор и адаптация модели вычислений, языка параллельного программирования и разработка метода архитектурно-независимого описания параллельных алгоритмов и программ для цифровой обработки данных на СБИС;

3) разработка алгоритмов и программных инструментов для процесса архитектурно-независимого проектирования и трансляции высокоуровневого представления СБИС в языки описания аппаратуры;

4) разработка с использованием предложенного метода и реализованных программных инструментов проектов цифровых интегральных схем и оценка эффективности предложенных решений на основе результатов практической реализации.

Методы исследований.

Поставленные задачи решены с использованием теории графов, системного анализа, имитационного моделирования, анализа и синтеза цифровых логических схем. При разработке основных положений диссертации применялись методы системного анализа разнородных вычислительных систем, функционально-потокового и объектно-ориентированного проектирования и программирования.

Научная новизна.

1. Впервые разработан метод высокоуровневого синтеза цифровых однокристальных интегральных систем на основе функционально-потоковой параллельного языка, являющийся архитектурно-независимым. В отличии от существующих методов, разработанный метод позволяет реализовать переносимость исходного высокоуровневого описания алгоритма функционирования СБИС на различные целевые платформы за счет свертки параллелизма.

2. Разработана модифицированная модель вычислений ФПП языка для описания функционального состава и алгоритмов управления вычислениями, позволяющая выполнять описания цифровых СБИС.

3. Впервые разработаны алгоритмы преобразования высокоуровневого архитектурно-независимого описания цифровых систем на основе ФПП языка, позволяющие выполнять его автоматическое преобразование в низкоуровневое архитектурно-зависимое представление.

Теоретическая значимость работы. Результаты работы могут быть использованы для повышения эффективности и скорости процесса разработки

цифровых СБИС. Результаты диссертационных исследований включают развитие теории параллельного программирования, создания и сопровождения программных средств поддержки проектирования цифровых интегральных схем и систем.

Практическая значимость работы заключается в повышении эффективности и надежности процессов разработки сверхбольших интегральных схем и компьютерных систем на их основе. Результаты исследования использованы при разработке и изготовлении на действующем производстве СБИС блока цифровой обработки сигналов ГНСС приемника, СБИС блока цифровой обработки сигнала из состава системы спутниковой связи, СБИС бортового комплекса управления для малых космических аппаратов. Результаты практического применения диссертационного исследования подтверждены актами о внедрении.

Получены следующие, основные практические результаты работы:

1) разработано алгоритмическое и программное обеспечение в виде комплекта прикладных программ для поддержки процесса высокоуровневого проектирования цифровых схем на базе функционально-потоковой парадигмы, получены свидетельства о регистрации программ для ЭВМ;

2) разработаны и внедрены на действующее производство комплекты сложно-функциональных блоков для реализации сверхбольших интегральных схем из состава бортовых систем управления космическим аппаратом, получены свидетельства о регистрации программ для ЭВМ;

3) разработаны и внедрены на действующее производство интегральные схемы для организации систем спутниковой связи;

4) разработаны и внедрены в процесс подготовки кадров высшей квалификации ФГАОУ ВО “Сибирский федеральный университет” курсы лекционного материала и курс лабораторных работ на действующем оборудовании, получен акт о внедерении.

Положения, выносимые на защиту.

1. Разработанный метод высокоуровневого синтеза цифровых СБИС на основе функционально-потоковой парадигмы(ФПП) позволяет реализовать архитектурную независимость и переносимость исходного высокоуровневого описания СБИС на различные целевые платформы.

2. Разработанные модели и алгоритмы высокоуровневого синтеза позволяют автоматически выполнять преобразование высокоуровневого представления на функционально-потоковом языке параллельного программирования в низкоуровневое представление на языке описания аппаратуры и осуществлять свертку параллелизма исходного высокоуровневого описания СБИС.

3. Разработанное инструментальное средство синтеза описания СБИС позволяют реализовать автоматическое преобразование высокоуровневого архитектурно-независимого представления цифровых схем в низкоуровневые архитектурно-зависимые представления для разных целевых платформ.

Достоверность изложенных в работе результатов обеспечивается применением методов теории графов и достаточным объемом экспериментальных исследований, сравнением эффективности разработанных программных инструментов с существующими аналогами, практическим внедрением результатов работы.

Использование результатов работы.

Результаты диссертационной работы использовались при выполнении работ:

- АО «Информационные спутниковые системы» им. академика М.Ф. Решетнева» для реализации БА КА Луч-5М, Луч-5ВМ, Экспресс-АМУ3, Экспресс-АМУ7, Экспресс 90, Экспресс 103;

- изготовлении СБИС К5540ТН014А на базе БМК «Алмаз-13» для АО “Радиосвязь”;

- ФЦП по Гос. контрактам №№ 14.578.21.0021, 14.513.11.0117, 14.A18.21.0396, 02.G25.31.0202;

- РФФИ проект № 17-07-00288.

Полученные практические и научные результаты используются в учебном процессе института космических и информационных технологий ФГАОУ ВО «Сибирский федеральный университет».

Использование вышеприведенных результатов диссертационной работы подтверждено соответствующими актами.

Апробация работы. Основные положения и результаты проведенных исследований докладывались и обсуждались в 2010-2018 годах на открытых межкафедральных семинарах ИКИТ СФУ, открытых лекциях и семинарах НУЛ «Микропроцессорные системы СФУ», а также на отраслевых, Всероссийских и Международных конференциях и семинарах:

- международной научно-практической конференции «Интеллект и наука» (Железногорск, 2013 г.);

- международной конференции «Решетневские чтения» (Красноярск, 2015 г.);

- II и III Всероссийских научно-технических конференциях «Системы связи и радионавигации» (Красноярск, 2015, 2016 г.г.);

- международной практической конференции "Research and Development - 2016" (Москва, 2016 г.);

- 5-й Всероссийской научно-технической конференции «Суперкомпьютерные технологии» (Геленджик, 2018 г.);

- Международной IEEE-сибирской конференции по управлению и связи «SIBCON-2019». (Томск, 2019 г.).

Личный вклад. В диссертации представлены результаты работы, в которых автору принадлежит определяющая роль. Постановка задачи и разработка концепции метода высокоуровневого синтеза проводилось с научным руководителем.

Публикации. По результатам диссертационных исследований и их практического применения опубликовано 17 печатных работ, из них 5 в ведущих рецензируемых журналах, включенных в список ВАК РФ, 3 работы в

рецензируемых изданиях, индексируемых в ведущих зарубежных БД WoS/Scopus. Получены регистрационные свидетельства на 8 программ для ЭВМ.

Структура и объем работы. Диссертация состоит из введения, 4 глав, заключения, списка литературы и приложений. Основной объем диссертации составляет 106 страниц, в том числе: 34 рисунка, 7 таблиц, список литературы из 110 наименований и 2 приложения.

В первой главе изложены результаты анализа маршрутов и способов проектирования СБИС. Определены основные тенденции развития при проектировании однокристальных систем. Выделены недостатки и достоинства существующих методов и определены перспективные подходы к проектированию. Рассматриваются известные в данной области научные направления, существующие на современном этапе создания средств вычислительной техники. Определены роль и место языковых средств описания аппаратуры в современной иерархии системного проектирования. Рассмотрены модели вычислений, парадигмы и языки программирования с точки зрения применимости к области разработки СБИС. На основе проведенного анализа определены основные задачи диссертационного исследования. В результате анализа определены требования к разрабатываемому методу высокоуровневого синтеза СБИС.

Во второй главе предложены способы описания алгоритмов функционирования параллельных систем на кристалле при использовании функционально-потоковой модели параллельных вычислений (ФПМПВ). Предложена модифицированная модель вычислений с массовым параллелизмом для реализации топологии СБИС на основе функционально-потокового языка параллельного программирования. Для привязки к предметной области на соответствующих этапах проектирования предложено использовать дополнительный слой - HDL - граф. Изложены принципы их формирования в процессе синтеза архитектурных решений. Представлен разработанный метод высокоуровневого синтеза СБИС на базе функционально-потокового представления исходных алгоритмов. При разработке метода обоснован и

выполнен выбор базового языка функционально-потокового параллельного программирования, введены требуемые для синтеза СБИС расширения, в том числе: типизация данных, статические размеры списков, замена циклов хвостовой рекурсией. Предложен маршрут проектирования СБИС, в том числе приведен общий алгоритм преобразования (сокращения) параллелизма при переходе на целевую платформу.

В третьей главе описаны разработанные инструментальные средства, применяемых для алгоритмического и программного представления, тестирования и трансляции в языки описания аппаратуры. Предложены расширения промежуточного представления в виде элементов, отображаемых на топологию интегральных схем, результаты разработки методов перехода от функционально-потоковой модели к представлению на языках описания аппаратуры. Разработаны методы и способы редукции промежуточных представлений исходных алгоритмов согласно требуемым критериям параллелизма, а также техническим требованиям к целевой платформе реализации.

В четвертой главе рассмотрено использование полученных результатов для разработки и тестирования проектов СБИС, полученных на основе функционально-потокового представления исходных алгоритмов. Проведено сравнение различных способов высокоуровневого синтеза на основе метрик процесса и результата разработки. Выбраны классы и конкретные задачи для сравнения разработанного метода и существующих методов высокоуровневого синтеза. Приведены результаты оценки полученных результатов на выбранных тестовых задачах. Изложены результаты сравнительного анализа и использования полученных результатов.

1 Технологии синтеза СБИС

1.1 Методы и способы представления архитектурного описания СБИС

С момента появления цифровых электронных схем началось развитие средств автоматизированной разработки и проектирования (EDA – Electronic design automation) для интегральных схем (ИС). К настоящему времени средства EDA прошли путь от ручного изображения схемы в интерактивном графическом редакторе до систем интеллектуального принятия решений.

Существующие на текущий момент методы описания архитектуры СБИС можно разделить на следующие группы:

1) описания схемы и соединений (netlist);
2) функциональное описание схемы СБИС на уровне регистровых передач;
3) поведенческое описание.

Первый вариант описания СБИС – netlist [13], в настоящее время является одним из самых распространенных и используется для обмена между различными системами автоматизированного проектирования. Примером стандарта на формат netlist является стандарт представления EDIF (EDIF – Electronic Design Interchange Format) [7]. Формат netlist удобен для электронного представления и обработки, но не используется для восприятия человеком.

Описание схем на уровне регистровых передач (RTL – Register Transfer Level) [11, 14] также используются компоненты и связи между ними, но более высокого уровня абстракции. При RTL-описании компоненты представляются как функции «черного ящика» (blackbox) от входных сигналов, формирующие выходные сигналы результата. Для описания схемы на базе RTL подхода используются языки описания аппаратуры (HDL – Hardware description language), посредством которых архитектура ИС представляется как описание функциональных блоков. В HDL языках используются высокоуровневые конструкций из языков программирования, такие как условные выражения (if, switch), массивы, параметризация, циклы и т.д. Применяемые в настоящее время

HDL-языки, например Verilog и VHDL [12, 5], являются типичными представителями языков функционального описания СБИС. Описание схемы на уровне RTL является первым уровнем абстракции, стоящим над низкоуровневым описанием СБИС, и используемым как конечное при преобразовании с более высоких уровней абстракции.

Поведенческое описание схемы [12] реализует представление структуры схемы, а не ее функции. Данный метод поддерживается языками Verilog и VHDL наряду с функциональным описанием входящих в структуру схем. Поведенческое описание (behavioral) еще более абстрагировано от особенностей реализации конкретной схемы на кристалле и по существу является ее высокоуровневым описанием. Несмотря на высокий уровень абстракции, поведенческое описание не избавляет разработчика от понимания процесса синтеза, знания низкоуровневых представлений функциональных блоков и особенностей реализации схемы на кристалле.

Несмотря на все более высокий уровень абстракции в некоторых методах (RTL, поведенческое) эти методы описания СБИС не избавляют разработчика от необходимости знания низкоуровневого представления. В отличие от методов разработки СБИС, языки программирования достигли на текущий момент такого уровня абстракции, который не требует при разработке знания особенностей архитектур вычислительных систем.

В настоящее время развитие способов описания архитектуры СБИС идет по пути повышения уровня абстракции несколькими основными путями, среди которых следует выделить:

1) расширение существующих HDL – языков конструкциями для более высокоуровневого поведенческого описания [6];

2) создание инструментов синтеза архитектурных решений СБИС посредством высокоуровневых языков, ранее применявшихся для разработки прикладного программного обеспечения [8];

3) создание новых языков высокоуровневого описания аппаратных систем на основе языков программирования [16].

Примером первого варианта развития является введение высокоуровневых конструкций для поведенческого описания СБИС в HDL – языки Verilog/VHDL, и появление в результате «гибридных» языков, например SystemVerilog [6].

Второй подход поддерживается компанией Xilinx и получил название высокоуровневого синтеза (HLS – High Level Syntheses) [15]. Этот способ основан на введении в базовый, высокоуровневый язык программирования ряда дополнений для использования его для описания топологии ИС.

Последний вариант реализован в таких языках, как HandelC [16]. Так же известен ряд проектов по использованию языков функционального программирования для синтеза СБИС (Lava, Hume, Erlang) [17-19].

Все три способа высокоуровневого представления архитектурных решений используются при различных маршрутах проектирования СБИС [9].

1.2 Маршруты проектирования СБИС

Маршрут проектирования (design flow) определяет последовательность этапов разработки СБИС — от технического задания до тестирования готового кристалла СБИС [9].

На сегодня существуют следующие маршруты проектирования [21]:

1. маршрут нисходящего проектирования;
2. маршрут абстракции на системном уровне (ESL) [20];
3. маршрут восходящего проектирования.

Примером маршрута восходящего проектирования является технология восходящего проектирования компании Synopsys (ACS –Automated Chip Synthesis) [10] или RTL Budgeting, которая заключается в предварительном описании временных и других ограничений для каждой составной части проекта.

1.2.1 Традиционный маршрут

Традиционный маршрут проектирования базируется на трехуровневой структуре СБИС (рисунок 1) [21]. Верхний уровень носит название

функционального и описывает функции СБИС на поведенческом уровне. Следующий уровень называется структурным и описывает уровень регистровых передач (RTL). Самый низший уровень носит название топологического уровня описания СБИС.

При более детальной классификация уровней СБИС можно выделить следующие:

– IP-блоки;

– схемотехническое описание;

– логические ячейки;

– уровень регистровых передач (RTL).

Традиционный маршрут проектирования базируется на HDL-языках. Маршрут проектирования является последовательным “сверху-вниз”.

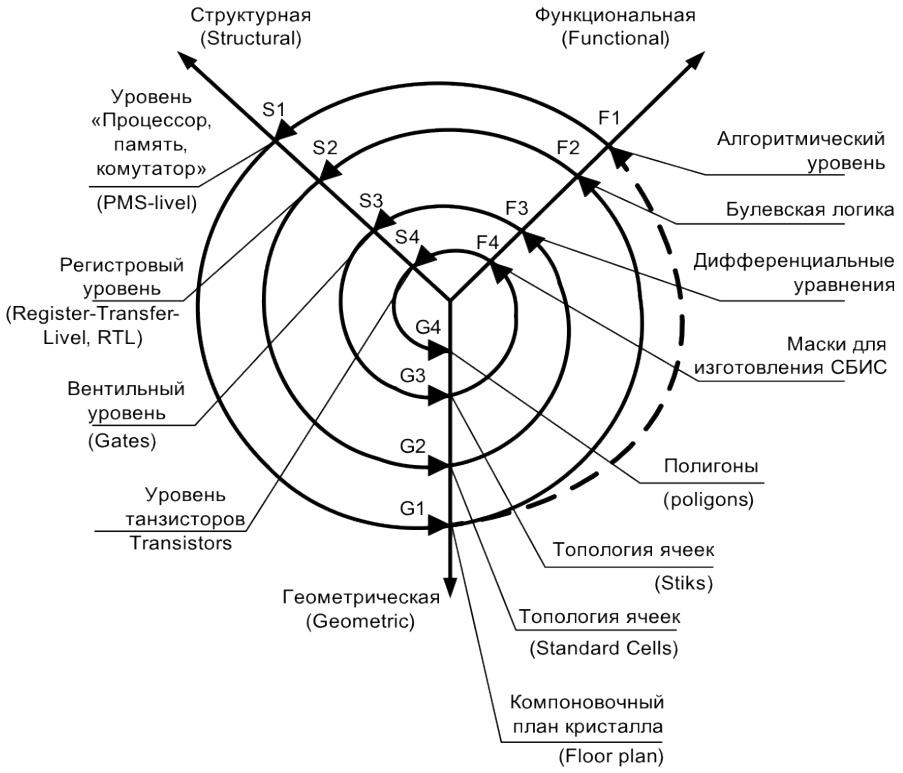


Рисунок 1 – Области и уровни моделей в проектировании СБИС

Первым этапом является разработка технического задания на проектирование системы. На основе ТЗ разрабатывается поведенческая модель на языках программирования (C++/SystemC) либо используя модельный подход. (C+

+). Модельный подход реализуют с использованием Matlab/Simulink, LabView [23]. В любых вариантах модели возможно применение программных моделей готовых IP-блоков.

Следующим этапом описываются интерфейсы СБИС и формируется высокоуровневая структура. Полученная модель проходит тестирование и верификацию. В зависимости от формата описания модели тестирование может осуществляется с помощью Matlab/Simulink/LabView либо средствами отладки языков программирования. Результатом является структурная модель системы.

Недостатками традиционного маршрута проектирования являются:

1) семантический разрыв между высокоуровневым системным представлением проекта и RTL описанием;

2) итерационный процесс разработки с возвратом к предыдущим этапам;

3) проектирование и моделирование программной и аппаратной частей отдельно;

4) отсутствие автоматизации преобразования проекта при переходе между уровнями проектирования;

5) привязка библиотек к целевой архитектуре начиная с функционального уровня;

6) количество рассматриваемых архитектурных вариантов реализации системы ограничено.

Между системным и RTL уровнем описания СБИС существует семантический разрыв. На уровне HDL описания системы существуют параметры, которые не учитываются при высокоуровневом проектировании. Семантический разрыв приводит к проблеме при переходе от алгоритмического описания (системного, высокоуровневого) к описанию на HDL языках.

1.2.2 ESL-проектирование

В основе ESL проектирования лежит моделирование системы в виде набора компонентов, интерфейсов и протоколов обмена информацией между ними.

Проектирование системы осуществляется «сверху вниз» с описанием структуры и протоколов.

В ESL подходе верхний уровень абстракции описывается в виде TLM (Transaction-level modeling) – модели передачи сообщений. В состав модели входит уровень сообщений (Message Level) и уровень обмена сообщениями (Transaction Level) [25, 26]. Спецификация такой системы представлена на рисунке 2.

Рисунок 2 – Общесистемный уровень разработки

Разделение проекта на программную и аппаратную части в традиционном маршруте проектирование происходит в начале процесса проектирования. В ESL

маршруте проектируется функциональный состав системы и протоколы обмена между блоками. Для каждого функционального блока выполняется разработка и тестирование алгоритма функционирования, тестирование взаимодействия блоков. Только после отладки прикладных алгоритмов производится проектирование архитектуры блоков СБИС. Прикладной алгоритмический и архитектурный уровень СБИС разделены.

На следующем этапа разрабатывается архитектура системы. Алгоритмы функционирования блоков модели реализуются на универсальных языках программирования (C). Для программных блоков разрабатывается отладочное ПО, для аппаратных блоков реализуются симуляционные тесты. Программная модель верифицируется с использованием полученного тестового ПО. По завершении данного этапа верифицированную модель разделяют на программную и аппаратную части.

Совместная отладка ПО и аппаратного обеспечения осуществляется на финишной стадии ESL маршрута проектирования. Этот этап выполняется для проверки совместимости ПО и аппаратного обеспечения. Каждый аппаратный блок описывается на языках SystemC/SystemVerilog [8, 6]. Из описания блоков генерируются тесты для аппаратной части системы. Симуляция аппаратных блоков позволяет определить несоответствия параметров блоков техническому заданию на проектирование. По результатам симуляции возможен возврат к предыдущим уровням в маршруте проектирования для устранения выявленных несоответствий.

Разделение на аппаратную и программную части реализуется в нескольких вариантах. Результаты моделирования анализируются и выбирается оптимальный вариант разбиения, соответствующий требованиям. Результатом является высокоуровневая модель системы на алгоритмических языках. Переход на RTL-уровень осуществляется различными способами. Самым распространённым способом является разработка по высокоуровневой модели HDL кода вручную. Вторым вариантом является полуавтоматическое получение HDL описания – в методе HLS из описания на языке C генерируется код на HDL языках.

Процесс перехода с высокоуровневого описания на уровень RTL требует участия разработчика. Процесс трансляции высокоуровневого описания в RTL выполняется полуавтоматически или вручную. Язык SystemC позволяет упростить процесс перехода за счет синтезируемых конструкций. В результате синтезируется микроархитектура СБИС и дальнейшее проектирование происходит по традиционному маршруту разработки СБИС.

Маршрут проектирования ESL позволяет автоматически разделить описание ИС на программные и аппаратные функции. Такой подход позволяет реализовать и протестировать множество вариантов разбиения системы на программную и аппаратную части. Анализ вариантов реализации позволяет оптимизировать систему по параметрам производительности, потребления энергии, размерам и получить более высокие характеристики по сравнения с традиционными маршрутами проектирования. Преимуществом ESL проектирования является возможность проектирования аппаратуры разработчиками, программирующими на универсальных языках программирования.

Отметим, что СБИС представляет собой схему параллельно-конвейерной обработки данных [43]. Эффективность реализации такой схемы зависит от метода представления и способа реализации исходного параллельного алгоритма, а также результата синтеза вентильной структуры СБИС для целевого кристалла с учетом имеющихся ресурсных ограничений.

Несмотря на преимущества ESL подхода перед традиционным маршрутом, у этого метода много недостатков. Основным недостатком является необходимость анализа множества вариантов реализации. Реализация множества вариантов на системном и алгоритмическом уровне достаточна проста, в то время как перебор вариантов на RTL уровне требует полного изменения проекта и повторной верификации. Поддержка параллелизма на высокоуровневом этапе проектирования затруднена из-за применяемых универсальных языков. СБИС является вычислительной системой с массовым параллелизмом и потоковой обработкой данных.

Модель вычислений, применяемая в таком подходе, основана на последовательной модели. Переход к RTL представлению преобразует её к параллельной модели обработки данных. Преобразование может выполняется либо автоматически, либо полуавтоматически. Полуавтоматическое преобразование требует участие разработчика, например указание директив преобразования как в методе HLS [15].

Высокоуровневое представление без поддержки описания параллелизма усложняет процесс перехода к RTL описанию. Автоматическое преобразование высокоуровневого описания порождает проблему автоматического распараллеливания. Эта проблема детально изучена в параллельном программировании, алгоритмы автоматического распараллеливания обладают высокой сложностью и не дают результата с необходимым качеством [28].

Как видно из анализа, существующие методы проектирования требуют изменения принципов проектирования. Требуется абстракция описания алгоритма СБИС от целевой архитектуры и реализация поддержки параллелизма на уровне операций для всех стадий маршрута проектирования. Метод описания алгоритмов на верхнем уровне иерархии должен обеспечивать сохранение параллелизма на всех последующих уровнях абстракции. Это позволит решить проблемы эффективного описания и распараллеливания исходного алгоритма функционирования СБИС, а также осуществить эффективный синтез для реализации на целевой платформе.

Результаты анализа современных методов и маршрутов высокоуровневого синтеза СБИС показывают, что при описании исходного параллельного алгоритма последний либо изначально представляется параллельным, либо его последовательное представление поэтапно заменяется параллельным вручную или в полуавтоматическом режиме.

Аналогично решаются задачи разработки программ для параллельных вычислительных систем и задачи эффективного распараллеливания и переносимости ПО на различные параллельные вычислительные архитектуры.

В данном случае эффективные решения получают при использовании соответствующих парадигм параллельного программирования и специальных языковых и инструментальных средств. Следовательно, решение означенных проблем может быть найдено в области технологий систем параллельного программирования.

Однако, в отличие от разработки параллельных программ, где алгоритм разрабатывается с учетом специфических особенностей и для известной вычислительной архитектуры, при разработке СБИС, архитектура вычислительной системы разрабатывается для реализации исходного алгоритма. Таким образом, требуется анализ языковых средств, применяемых при проектировании СБИС, а также разработка требований к модели и языку параллельного программирования применимого для описания алгоритмов в маршруте проектирования СБИС.

1.3 Формы представления архитектуры СБИС и модели вычислений

Рассмотрим цифровую схему на кристалле. Цифровая СБИС представляет собой структуру из функциональных блоков и связей между ними. Сигналы данных и сигналы управления являются связями между блоками. Каждый блок может быть как элементарным, так и составным. Цифровые схемы являются синхронными схемами обработки потока данных на входе. Такая схема получила название схемы потока данных [31]. Схема потока данных представляет собой граф. Узлы графа – блоки обработки данных, дуги – пути передачи данных и сигналов управления. Данные в блоках обрабатываются независимо и параллельно. Готовность данных на входе блока определяется сигналами управления.

В основе высокоуровневых языков, используемых в современных HLS подходах, лежит последовательная императивная модель. Низкоуровневое описание на языке HDL представляет собой параллельную схему потоковой обработки данных. Переход между высокоуровневым описанием и описанием на HDL языках сопровождается изменением модели вычислений. Отсутствие в

известных методах высокоуровневого синтеза описания параллелизма на верхнем уровне представления приводит к сложности перехода к представлению на RTL уровне. Процесс перехода выполняется разработчиком и требует больших временных затрат.

Описание схемы СБИС при синтезе представляет собой управляющую схему и схему потока данных: управляющего графа и графа данных (Control and Data Flow Graph) [32, 33]. Схема на СБИС представляет собой схему параллельной обработки данных. Количество параллельно выполняемых операций (степень параллелизма) зависит от конкретной задачи, реализации метода её решения и максимальной степени параллелизма, ограниченной ресурсами целевой платформы. При этом количество параллельных операций на разных стадиях решения задачи на СБИС может быть разным, в зависимости от распределения имеющихся аппаратных ресурсов между блоками. Если проводить аналогию с разработкой ПО для параллельных вычислительных систем, то можно заметить существенную разницу: в параллельных вычислительных системах количество выполняемых операций задано архитектурой системы и не может изменяться разработчиком. При этом количество операций, выполняемых в СБИС параллельно, может значительно превосходить количество операций в параллельных вычислительных системах. Отсюда можно сформулировать первое требование к языку разработки для СБИС: поддержка в языке модели параллельных вычислений на уровне операций.

Используя одно и то же описание алгоритма на языках высокого уровня на разных платформах или при разных ограничениях можно получать переносимые на разные архитектуры реализации. Примером реализации переносимости высокоуровневых программ в среде разработки ПО являются такие языки как JAVA, C#.

Переносимость и архитектурная независимость для разработки СБИС означает, прежде всего, возможность из исходного параллельного описания алгоритма получать варианты с различным параллелизмом при переходе на различные целевые архитектуры СБИС. Реализация данного принципа требует

изменения степени параллелизма без изменения исходного алгоритма. Достичь этого можно при использовании в языке принципа массового параллелизма на уровне операций и сокращении его при переходе на целевую архитектуру СБИС. Из вышеизложенного вытекает второе требование: необходимо наличие возможности сжатия степени параллелизма исходного алгоритма при переходе на различные целевые платформы без изменения исходного высокоуровневого описания схемы. Аналогичный подход для решения проблем переносимости параллельного ПО приведен в [45].

Рассмотрим обобщенный способ обработки данных в СБИС. Любой подход, подразумевает схему, которая состоит из набора сигналов данных и набора сигналов управления. Сигналы управления включают в себя сигналы готовности данных. Каждый блок из состава системы может независимо и параллельно с другими обрабатывать данные при условии наличия данных и сигнала их готовности. Если проводить аналогии с языками параллельного программирования, то такая модель является «моделью потоковых вычислений по готовности данных». Такая модель наиболее полно соответствует архитектуре СБИС.

Для разработки схем на СБИС характерно множество ресурсных ограничений [34, 35]. Одним из таких ограничений является объем памяти (регистров/блочной памяти). При разработке ПО данное ограничение не существенно, так как в этом случае используются механизмы динамического выделения памяти, стека и т.д. Использование таких механизмов в СБИС невозможно. Данное ограничение выдвигает требование статического определения необходимых ресурсов памяти на этапе компиляции/синтеза. Из этого же требования вытекает необходимость наличия в языке статической системы типов, поскольку динамическая типизация на платформе СБИС эффективно нереализуема.

Таким образом, для эффективного синтеза СБИС-архитектур требуется:

1) использовать модель массовых параллельных вычислений;
2) модель потока данных;

3) описывать алгоритмы с максимальным параллелизмом при неограниченных ресурсах;

4) обеспечить сокращение/преобразование параллелизма при неизменном исходном описании алгоритма;

5) применять общую модель вычислений на всех уровнях разработки проекта;

6) выполнять распределение аппаратных ресурсов только на этапе трансляции;

7) использовать статическую систему типов.

Рассмотрим существующие классы языков для разработки СБИС и моделей вычислений с целью выбора, соответствующего изложенным требованиям.

1.4 Языковые средства разработки СБИС

На рисунке 3 показано место языковых средств, применяемых на различных этапах создания проекта СБИС [29].

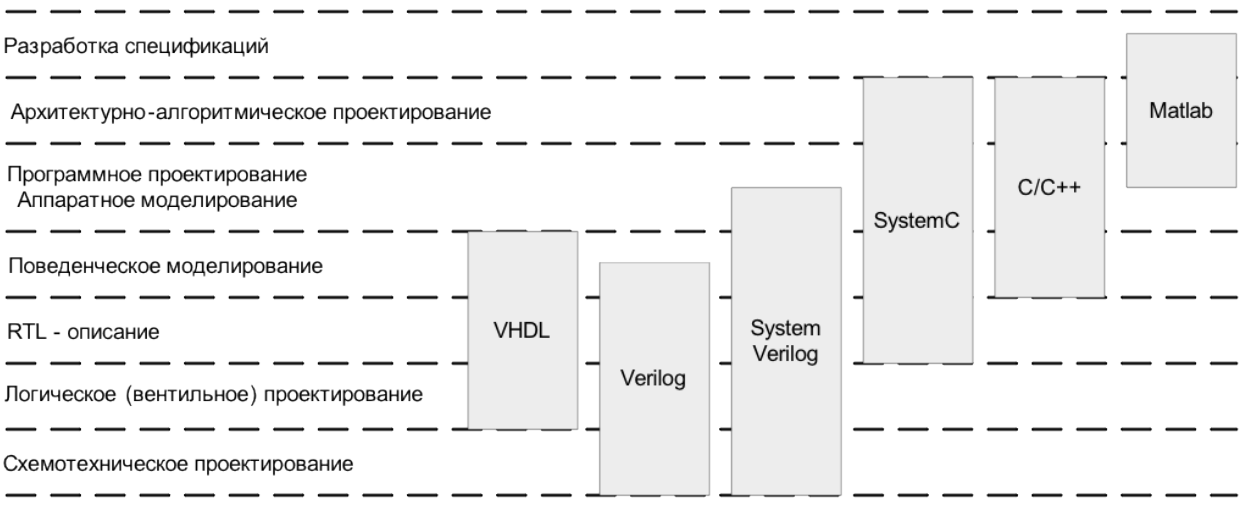


Рисунок 3 – Этапы и языки проектирования

Языки и парадигмы программирования следует рассматривать с точки зрения решения основных проблем разработки СБИС, а именно: описания алгоритма с учетом его параллелизма и эффективного переноса между

различными архитектурами. В классификации языков программирования, используемых для разработки СБИС, выделим следующие группы:

1) универсальные высокоуровневые языки;
2) проблемно-ориентированные языки;
3) функциональные языки.

Применяемые на сегодняшний день универсальные высокоуровневые языки (C/C++) используются для высокоуровневого описания системы. На этом уровне описывается структура, алгоритмы и общая модель системы. Языковые средства описания аппаратуры (HDL-языки) позволяют описывать низкоуровневое представление СБИС и архитектурные особенности –возникает семантический разрыв между высокоуровневым представлением и архитектурной реализацией СБИС. Применение универсальных высокоуровневых языков приводит к неэффективности преобразования между высокоуровневой моделью и описанием СБИС.

Другим используемым вариантом высокоуровневого описания СБИС являются специализированные языки – SystemC [8], HandelC [16]. Эти языки являются специализированными C-подобными языками описания аппаратуры. Целью их разработки было повышение уровня абстракции описания СБИС. При отладке системы на этих языках происходит привязка к конкретной целевой платформе – это снижает уровень их абстракции. Недостатками этих языков являются сложные средства синтеза - для эффективной трансляции в RTL-код требуется высокая квалификация инженера.

Так же к высокоуровневым языкам, используемым для разработки архитектуры ИС, можно отнести языки, используемые для организации вычислений на высокопроизводительных реконфигурируемых вычислительных системах (РВС).

Из отечественных разработок для РВС следует отметить язык COLAMO [2]. Основной областью применения COLAMO является разработка ПО для РВС. В языке используется разбиение на процедуры и вычислительные структуры для

специализированной вычислительной структуры. Описание параллелизма реализуется с помощью объявления типов доступа к данным.

Данные инструментальные средства ориентированы на построение инфраструктуры решения прикладных задач в области высокопроизводительных вычислений, используют многокристальные решения и для однокристальных систем и разработки СБИС являются сложными и избыточными.

Функциональное программирование [39] не задает порядок вычислений и получения результата, а описывает конечный результат вычисления. Описание программы на функциональном языке изначально является параллельным, следовательно, нет необходимости выделять параллельные участки [39].

Отметим, что наиболее полно проблема исходного описания алгоритма с учетом его параллелизма решена для функциональных языков при описании программы в виде графа потока данных. В работах Доннагары [45, 46] показано, что решение проблем переносимости параллельных программ лежит в области изменения парадигмы программирования. Такая парадигма должна соответствовать следующим условиям [45]:

1) параллелизм на уровне операций;
2) модель потока данных;
3) неявное управление вычислениями (управление по готовности данных).

Переносимость параллельных алгоритмов может быть реализована с использованием свертки параллелизма. На примере параллельного ПО (PaRSEC) для высокопроизводительных вычислений на практике показана высокая эффективность данного подхода [45, 47]. Модель параллелизма на уровне операций с моделью потока данных и отсутствием явного управления вычислениями лежит в основе языков функционально-потокового параллельного (ФПП) программирования [41]. Это дает основание считать методы ФПП программирования и соответствующую модель вычислений наиболее подходящей для высокоуровневого синтеза СБИС.

В работах [48, 49] рассматриваются реализации модели графа потока данных для СБИС, в частности для платформы ПЛИС. Теоретические

исследования и практическая реализации в данных работах показали эффективность реализации модели потока данных [48]. Вместе с тем данные реализации не поддерживают переносимость, содержат явное управление вычислениями и ориентированы на узкое применение (цифровая обработка сигналов) [50].

Из функциональных языков наибольший интерес вызывают языки потока данных (функционально-потоковые) [41]. В этих языках, ориентированных на параллельное программирование, применяется модель потока данных. Данное направление в функциональном программировании получило название функционально-потоковой парадигмы параллельного программирования. Основными особенностями языков данного класса являются модель потока данных с параллелизмом на уровне операций и отсутствием явного управления вычислениями. Программа на таком языке представляет собой информационный граф. В информационном графе отсутствуют циклы и такой граф является ориентированным графом. Организации вычислений основана на принципе готовности данных. Вершины графа содержат операторы обработки данных, а дуги – пути передачи информации. В модели вычислений такого языка ресурсы, доступные для организации вычислений, считаются бесконечными. При переходе на конкретную вычислительную платформу происходит сокращение/преобразование параллелизма. Языки данного класса позволяют описывать параллельные программы, ориентированных на обработку потоков данных. Выполнение операторов происходит по готовности данных. Параллелизм в таких программа ограничен только информационными связями и методом решения задачи (массовый параллелизм). Результаты анализа языков данного класса показали соответствие требованиям 1-4,6 (см. стр. 34). При этом п.п. 7 и 8 – статическая система типов и статическое определение ресурсов памяти на этапе трансляции – зависят от конкретной реализации языка, дополнительных конструкций и ограничений, вводимых в язык для поддержки синтеза.

Идея использования функционального подхода для синтеза ИС прослеживается в ряде работ. В частности, использование формального описания аппаратуры в функциональном стиле впервые высказано в 1972 году в работе John Lee [51].

Идея функционального описания отражена в ряде работ по применению функциональных языков для симуляции алгоритмов СБИС. Среди них можно отметить языки uFP, Daisy, νFP/FHDL [53, 54, 55, 56]. Язык uFP [53] является языком без типизации. Его прототип - язык FP был расширен добавлением потоков, что позволило описывать логические цепи в СБИС. Язык νFP/FHDL [55, 56] также был разработан на основе языка FP путем добавления специальных структур, описывающих функцию перехода от входа к выходу элемента и его начальное состояние. Язык Daisy [54], основанный на Scheme, описывал схемы как спецификацию из рекурсивных выражений и ее реализацию в виде системы сигналов. Аналогичный подход использовался в языке Hydra[57]. Общим для этих языков является использование потоков для описания сигналов и рекурсивных выражений для описания преобразований в схеме.

В работах по языкам Lava, HML [17, 58, 59] преследуется цель реализации процесса синтеза схемы СБИС. В качестве промежуточного представления для синтеза в этих языках используются xHDL и Verilog/VHDL. Lava является встроенным в Haskell подмножеством языка для симуляции, синтеза и верификации СБИС. Язык HML [58, 59] является подмножеством языка SML используемым для описания СБИС. HML близок по структуре к языку VHDL, поддерживает структурное и поведенческое описание схемы. В отличие от VHDL в HML имеется ряд отличительных особенностей, присущих функциональным языкам, например поддержка вывода типов, рекурсия и функции высокого порядка. Рекурсии и функции высокого порядка могут быть симулированы, однако их трансляция в VHDL не поддерживается. Еще одним вариантом языка SML для СБИС является язык SAFL/SAFL+ [60, 61]. В этом языке вводится понятие статического выделения памяти на этапе компиляции, а функции могут

быть не рекурсивными или с хвостовой рекурсией. Для представления потоковых вычислений вводится понятие каналов.

Целью разработки языка Hawk [62] являлась моделирование архитектуры микропроцессоров. Язык не используется для синтеза схем СБИС, а применяется только для формальной верификации структуры микропроцессоров. Язык HDCaml [66], реализованный на базе языка Ocaml, описывает схемы с использованием сигналов и цепей. Цепи определяются через рекурсивное добавление новых сигналов и других цепей.

Отдельно стоит выделить функциональные языки Bluespec [63] и FL/reFLect [64, 65], так как они является промышленно используемыми решениями, в отличие от ранее описанных. Bluespec является функциональным языком высокоуровневого синтеза СБИС. Функционал языка Bluespec основывается на Haskell. Язык FL/reFLect [65] разработан компанией Intel. Язык основан на функциональном языке Standard ML. Его основным предназначением является описание и верификация схемы, при этом он не используется для синтеза схем. В работах [67, 68] описано использование данного языка для верификации блока арифметики для работы с числами с плавающей точкой, декодера инструкций микропроцессора и т.д.

К функциональным языкам, поддерживающим модель потока данных и параллелизм на уровне операций, относятся такие языки как Пифагор [41]. Его основными особенностями являются:

1) программа в виде информационного графа с управлением по готовности данных;

2) параллелизм на уровне операций.

Вместе с тем в языке отсутствует ряд необходимых для применения в качестве языка высокоуровневого синтеза СБИС средств, как-то строгая типизация, преобразование рекурсии и т.д.

1.5 Выводы

1. Проведенный анализ языков и высокоуровневых методов разработки СБИС показывает, что существующие подходы имеют недостатки:

a) семантический разрыв между высокоуровневым описанием алгоритма и низкоуровневым представлением СБИС в универсальных языках, используемых для разработки СБИС;

b) отсутствие учета параллелизма на уровне операций, свойственного архитектуре СБИС;

c) архитектурная зависимость в методах на основе специализированных языков разработки.

2. Описание исходного алгоритма функционирования СБИС для высокоуровневого синтеза на основе функционально-потокового подхода позволяет уменьшить семантический разрыв и является перспективным.

3. Для использования ФПП языка в методе высокоуровневого синтеза СБИС необходимо разработать алгоритмы преобразования архитектурно-независимого описания интегральных схем в архитектурно-зависимые представления на языках описания аппаратуры для последующего синтеза.

4. На основе разработанных алгоритмов необходимо разработать инструментальные средства, реализующие метод высокоуровневого синтеза СБИС.

2 Модель и язык для метода высокоуровневого синтеза

2.1 Модель вычислений на основе функционально потокового параллельного программирования

Проведенный анализ позволил остановиться на использовании в качестве основы для метода высокоуровневого синтеза функционально-потокового подхода, обеспечивающего поддержку требований, предъявляемых к высокоуровневому проектированию СБИС. В качестве исходной модели взята модель вычислений ФПП языка [41].

В п. 1.4 приведены требования к модели вычислений для СБИС. Взяв за основу ФПП модель, можно выделить следующие особенности имеющейся модели, которые требуют их адаптации к модели вычислений, присущей СБИС:

- динамическая система типов;
- ограничение типов данных;
- обработка списков;
- задержанные вычисления;
- рекурсия.

Использование динамической типизации для СБИС нецелесообразно поскольку ресурсы памяти на СБИС ограничены, и динамическая типизация не используется в языках описания аппаратуры. Помимо этого, набор типов данных, используемых в СБИС также ограничен.

Использование задержанных вычислений на СБИС не имеет смысла, так как теряется изначальная концепция, заложенная в них. На платформе СБИС любой участок алгоритма, независимо от того, будет он использоваться при вычислениях в зависимости от исходных данных или нет, будет синтезирован. Блоки отложенных вычислений будут синтезированы и не реализуют изначально заложенную в них концепцию экономии ресурса/отсутствия вычислений без необходимости в них. Поэтому при переходе к платформе СБИС отложенные вычисления требуют модификации в исходной модели.

Невозможность реализации рекурсии на СБИС в исходном виде также связана с отсутствием динамического выделения ресурсов и их ограниченностью. Рекурсия также требует преобразований для поддержки на платформе СБИС.

Помимо этого, для реализации архитектурно-независимого описания СБИС необходимо разработать методы изменения параллелизма исходного описания алгоритма при переходе на разные платформы с разными ресурсными ограничениями. Методы изменения параллелизма позволят получать из одного исходного описания алгоритма всевозможные варианты реализации в зависимости от размерности исходных данных и ресурсных ограничений целевой платформы. При этом исходное описание алгоритма остается неизменным и архитектурно-независимым (переносимым).

Далее подробно описаны изменения, которые требуется внести в базовую ФПП модель вычислений и методы (стратегии) изменения параллелизма исходного алгоритма.

2.2.1 Система типов в модифицированной ФПП модели

ФПП модель изначально имеет динамическую систему типов. Однако, для описания алгоритмов, которые могут быть реализованы на СБИС, необходимо статическое определение размера структур данных на этапе трансляции/синтеза. Помимо этого, набор типов данных, реализуемых на СБИС, ограничен. При реализации на платформе СБИС следующие типы данных не используются (не поддерживаются):

- строки и символьные типы данных;
- числа в формате с плавающей запятой.

Следует отметить, что числа в формате с плавающей точкой имеют ограниченное применение на платформе СБИС и в некоторых редких случаях их применение возможно.

Для реализации статической типизации предлагается ввести 4 скалярных типа данных: логический тип, целочисленный знаковый тип, целочисленный тип

без знака и битовое поле. Данные типы чаще всего используются в языках описания аппаратуры [5, 6]. Эти типы могут быть заданы программистом либо определены на этапе синтеза при контроле и автоматическом назначении типов. Для автоматического вычисления и назначения типов необходимо только задание типов входных и выходных аргументов функции верхнего уровня. Автоматизированное вычисление типов гарантирует отсутствие ошибок потери точности. В случае выхода за границы имеющихся аппаратных ресурсов выдается предупреждение о возможной потере точности.

Обработка списков в модели со статической типизацией также имеет свои особенности. Помимо необходимости определения размерности списков вводится ограничение на динамическое изменение размерности списков во время вычисления. Если на этапе типизации синтезатор определяет, что размерность списка изменяется и конечную размерность нельзя определить на этапе синтеза – генерируется ошибка синтеза.

2.2.1.1 Скалярные типы данных

Тип **bool** - представляет собой логический тип разрядностью 1 бит. Может принимать значение **true (1)** или **false (0)**. Синтаксис описания типа – ключевое слово **bool.**

Тип **bits** - представляет собой битовый вектор заданной разрядности. Описание типа bits имеет следующий синтаксис:

bits.*битовая разрядность*.

Тип **int** - представляет собой целое знаковое число заданной разрядности в дополнительном коде. Старший бит содержит знак числа. При преобразованиях с увеличением разрядности старшие биты заполняются знаковым разрядом. Описание типа int имеет следующий синтаксис:

int.*битовая разрядность*.

Тип **uint -** представляет собой целое беззнаковое число заданной разрядности. При преобразованиях с увеличением разрядности старшие биты заполняются нулем. Описание типа uint имеет следующий синтаксис:

uint.*битовая разрядность.*

При определении **пользовательских типов** синтаксис определения следующий:

type Name << **typedef** *скалярный тип с разрядностью;*

Пример:

type SmallInt << **typedef** int.16;

2.2.1.2 Векторные типы данных

Помимо скалярных типов в языке присутствуют векторные данные: списки данных, параллельные, асинхронные и задержанные списки. Параллельные списки известной размерности преобразуются согласно [71]. Для указания размерности параллельных списков и списков данных в язык вводится следующий синтаксис описания типов:

Param.*тип_списка.размерность.тип_данных*.

Тип списка может принимать следующие значения: datalist, parlist либо поле может отсутствовать для скалярных типов. Размерность списка является неотрицательным целым значением. Тип данных может быть любым скалярным или определенным пользователем типом. Неопределенное значение размерности или типа данных, вычисляемое в процессе синтеза, задается в соответствующем поле знаком **!**.

Пример:

Param.datalist.!.int

2.2.1.3 Преобразование типов

Преобразование типа bool в типы int, uint и bits происходит путем копирования значения bool в младший разряд. Все остальные разряды присваиваются нулю.

Преобразование типов int, uint и bits при одинаковой и большей разрядности результата приведено в таблице 1, при уменьшении разрядности результата – в таблице 2.

Таблица 1 – Преобразование типов при одинаковой или большей разрядности

из типа / **в тип**	**bits**	**uint**	**int**
bits	Копирование бит, старшие биты разницы инициализируются 0	Копирование бит, старшие биты разницы инициализируются 0	Копирование бит, старшие биты разницы инициализируются знаком
uint*	Копирование бит, старшие биты разницы инициализируются 0	Копирование бит, старшие биты разницы инициализируются 0	Копирование бит, старшие биты разницы инициализируются 0
int	Копирование бит, старшие биты разницы инициализируются знаком	Копирование бит, старшие биты разницы инициализируются знаком	Копирование бит, старшие биты разницы инициализируются знаком

*- конверсия из uint в int одинаковой разрядности приводит к потере старшего бита, так как в int старший бит знаковый.

Таблица 2 – Преобразование типов при уменьшении разрядности

из типа / **в тип**	**Bits**	**uint**	**int**
bits	Копирование младших бит	Копирование младших бит	Копирование младших бит
uint	Копирование младших бит	Копирование младших бит	Копирование младших бит
int*	Копирование младших бит	Копирование младших бит	Копирование младших бит

*-преобразование типов из int в uint и bits реализовано как в языках описания аппаратуры и не равно операции взятия модуля. Для этого сначала необходимо провести операцию вычисления модуля int типа.

2.2.1.4 Контроль и автоматическое назначение типов

Процесс вычисления типов данных строится на обходе информационного графа (ИГ) алгоритма. Процесс задания строгой типизации начинается с определения типа входного аргумента.

Для входного аргумента возможны два варианта: скалярный тип аргумента или список. В случае списка входных аргументов выбор конкретного значения в языке происходит с помощью операции интерпретации и целочисленной константы:

X1 << Param : 1.

Для определения и задания типов всех аргументов происходит поиск в ИГ всех операций интерпретации, ссылка на данные у которых содержит указатель на нулевую вершину и функцией является целочисленная константа. Нулевой вершиной в ИГ всегда является вершина аргумента. Для всех таких найденных вершин определяется тип выходного аргумента. Если ни одной такой вершины операции интерпретации не найдено – аргумент функции скалярное значение, тип аргумента записывается в вершину 0. Для задания типов аргументов используется отдельный файл типов с синтаксисом, приведенным выше. Это позволяет отделить привязку к конкретной целевой платформе от исходного архитектурно-независимого описания алгоритма на ФПП языке. Далее совершается нисходящий обход графа с вычислением и назначением типов для выходов всех вершин операций интерпретации, констант, списков и вершины возврата результата функции в соответствии с семантикой языка. Правила вычисления типов для разных операций приведены в таблице 3.

Таблица 3 – Правила вычисления типов данных на этапе синтеза

Операция	**Вход**			**Выход**		
	Вид	**Размер**	**Тип**	**Вид**	**Размер**	**Тип**
+,*,-	Список данных	2	int, uint	Атом	1	int, uint
		N	bool	Атом	1	bool
	Атом	1	bool	Атом	1	bool

Окончание таблицы 3

Операция	**Вход**			**Выход**		
	Вид	**Размер**	**Тип**	**Вид**	**Размер**	**Тип**
-	Атом	1	int,uint	Атом	1	int
/,%	Список данных	2	int, uint	Атом	1	int
=, !=, <, <=, >, >=	Список данных	2	int, uint,bool	Атом	1	bool
?	Список данных	N	bool	Параллельный список	N	bool
|	Список данных	N		Атом	1	uint
Целое число 1..N	Список данных	N	int, uint, bool	Атом	1	int, uint, bool
Булевское значение	Список данных	N	int, uint, bool	Список данных	N	int, uint, bool
	Атом	1		Атом	1	
Dup	Список данных	2	int, uint, bool	Список данных	N	int, uint, bool
#	Список данных	N	int, uint, bool	Список данных	N	int, uint, bool

2.2.2 Рекурсивные вычисления

Циклические вычисления в функциональных языках описывают с помощью рекурсии. Обычные циклы обладают ресурсными ограничениями. При реализации рекурсивных вычислений в неизменном виде также возникают проблемы, связанные с выделение памяти. Данная проблема присуща некоторым вычислительным архитектурам, в том числе платформе СБИС. Проблема

заключается в длительных («бесконечных») повторяющихся вычислениях, рекурсивные вызовы которых приводят к использованию огромного количества памяти. Стратегия асинхронного и массового параллелизма вносит свои особенности в решение этой проблемы.

Для практической реализации рекурсивных вычислений на платформе СБИС существует вариант преобразования рекурсии в итеративные схемы с использованием хвостовых рекурсий [69, 70]. Поскольку на платформе СБИС отсутствует динамическое выделение памяти, то данное ограничение делает невозможным реализацию рекурсивных вычислений с неизвестной на этапе трансляции глубиной рекурсии. Такое же ограничение на количество итераций цикла существуют и в описании циклических вычислений при HLS на базе императивных языков [15].

Таким образом, на платформе СБИС рекурсивные вычисления преобразуются в итеративные с использованием хвостовой рекурсии и заданием глубины рекурсии на этапе трансляции. Это позволяет преобразовать такие вычисления в конвейерную схему.

Для задания максимального числа итераций цикла (глубины рекурсии) предлагается использовать точное указание количества итераций на этапе описания алгоритма, либо указание разрядности переменной цикла, на основании которой, можно вычислить размерность цикла. Использование этих ограничений позволяет развернуть рекурсию в дерево вызовов функций для дальнейшего преобразования в конвейерную схему при переходе на платформу СБИС [81]. Рассмотрим преобразование рекурсии с известной глубиной в конвейерную схему (рисунок 4).

```
VecSum << funcdef Param {     //сложение элементов массива
  Len << Param:|;             //длина входного списка данных
  variant<< ((Len,2):[<,=,>]):?; //длина меньше, равна или больше 2

  (
    (Param:[([],signal,signal):variant]),  //меньше 2, возвращаем аргумент
    (Param:[(signal,+,signal):variant]),  //равна 2, возвращаем сумму двух элементов списка
    (Param:[(signal,signal,Alpha):variant]) //больше двух, передаем в функцию Alpha
```

```
 ):[variant]:[]   //фильтрация (выбор) результата
>>return
}
//
Alpha<< funcdef Param {
   Len    << Param:|;
   OddVec << Param:[(1,Len,2):..]; //выбираем четные элементы массива
   EvenVec<< Param:[(2,Len,2):..]; //нечетные

 ([OddVec,EvenVec]:VecSum):+>>return//параллельное сложение в VecSum обоих частей
}
```

Рисунок 4 – Рекурсивная функция сложения элементов массива на языке Пифагор

В примере реализована рекурсивная функция сложения элементов массива. В данном примере глубину рекурсии определяет переменная Len – длина входного списка аргумента. На этапе типизации длина может быть задана явно – путем указания длины списка, либо неявно – указанием разрядности переменной Len. Во втором случае будет определена максимальна возможная для данной разрядности переменной глубина рекурсии – 2^N-1, где N – разрядность переменной, определяющей глубину рекурсии. Допустим, что размер массива Param функции VecSum задан как: Param.datalist.8.int.16 - представляет массив из 8 значений целого типа разрядностью 16 бит. При разворачивании рекурсии получаем следующее дерево вызовов с соответствующими значениями переменной Len (в скобках) в каждом вызове (рисунок 5).

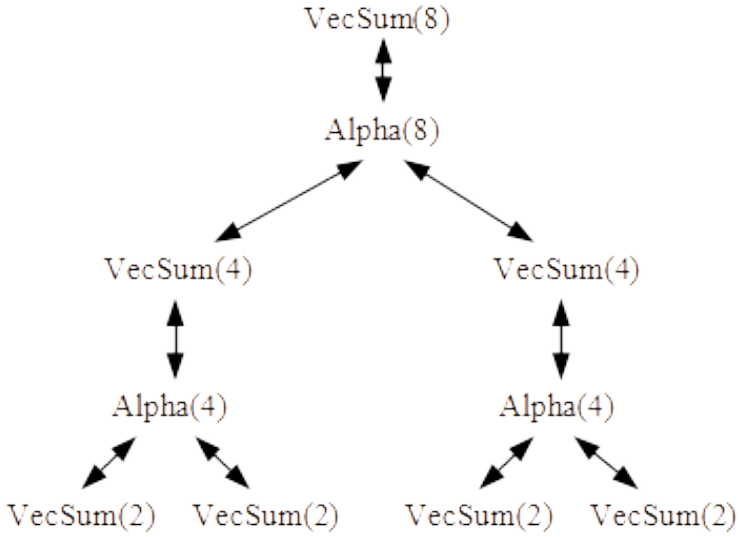


Рисунок 5 – Схема вызовов функций вычисления суммы элементов массива

На каждом уровне рекурсии (конвейера) на этапе синтеза уже известен реализуемый вариант, а неиспользуемые варианты могут быть удалены на этапе оптимизации. Если рассматривать реализацию данной схемы в виде конвейерной, то стадии конвейера на рисунке 5 расположены снизу вверх.

2.2.3 Обработка списков и задержанных вычислений

Для обработки списков данных и параллельных списков при переходе от описания алгоритма на ФПП – языке к платформе СБИС достаточно определить размерность списков на этапе трансляции. В ФПП языке параллельные списки можно преобразовывать с помощью различных эквивалентных преобразований. Эти преобразования позволяют упрощать параллельные списки во время трансляции и оптимизации. Известные на этапе трансляции размерности списков позволяют эффективно проводить оптимизирующие преобразования.

Первым вариантом оптимизации является раскрытие параллельного списка данных или функций. Применение функции к параллельному списку данных преобразуется в параллельный список результата. Каждый элемент списка формируется как исходный элемент списка данных, обработанный функцией:

$$[m_1, m_2, m_3, \ldots, m_n] : H \Rightarrow [m_1 : H, m_2 : H, m_3 : H \ldots, m_n : H] .$$

Аналогично преобразуется параллельный список функций и элемент данных:

$$m : [H1, \ldots, Hn] \Rightarrow [m : H1, \ldots, m : Hn] .$$

Согласно правилам языка параллельные списки могут содержать как данные, так и функции. Результатом операции будет список с размерностью, равной произведению размерностей исходных списков. Элементы в списке результата располагаются в порядке элементов сначала данных, потом функций:

$$[n_1, \ldots, n_n] : [f1, \ldots, fn] \rightarrow [n_1 : f1, \ldots, n_1 : fn, \ldots, n_n : fn].$$

В общем случае в параллельных списках данных и функций возможны вложенные параллельные списки. По правилам языка порядок их преобразований

не оговаривается. Вложенность списков при преобразованиях может быть сохранена. Результат функции над вложенными данными будет следующим:

[n1, [n2, [n3], n4, n5]]:f1 => [n1:f1, [n2, [n3], n4, n5]:f1] → [n1:f1, n2:f1, [n3, n4]:f1, n5:f1] → [n1:f1, n2:f1, n3:f1, n4:f1, n5:f1] .

В большинстве случаев вложенные параллельные списки преобразуются в результате в один линейный параллельный список. Например, если вложенный параллельный список оказывается в результате вычислений внутри списка данных:

([m1, [m2, [m3, m4]]]) => (m1, m2, m3, m4),

или при применении функции к такому списку:

[[n1, n2], [n3, [n4], n5]]: . => (n1, n2, n3, n4, n5).

Все выше приведенные преобразования позволяют избавиться от вложенных параллельных списков. При определении на этапе трансляции вложенных параллельных списков производятся оптимизирующие преобразования. В результате получается простой параллельный список:

[[n1, n2, [n3, n4], n5]] => [n1, n2, n3, n4, n5].

Еще одним вариантом оптимизации является замена параллельного списка размерности 1 на его элемент:

[a] → a.

Отложенные вычисления в исходной ФПП модели используются для выбора альтернативных вариантов в ветвлениях. Для выбора альтернативных вариантов вычислений на платформе СБИС используется сигнал “валидности” данных, который через схему выбора попадет на один из блоков альтернативных вариантов вычислений. Для поддержки реализации схем ветвлений без отложенных вычислений используется значение signal.

Рассмотрим процесс описания альтернативных вычислений без использования задержанных списков и преобразования таких вычислений при переходе на платформу СБИС.

Для программирования без задержанных списков используется реализация для каждого варианта ветвления функции с соответствующим кодом, который

обычно помещался в задержанный список при их использовании. Эти функции помещаются в списки данных внутри основной функции, использующей ветвления. Каждый список содержит одну функцию и signal в качестве остальных значений, при этом если функция является i-тым элементом какого-либо списка — то остальные списки хранят в качестве i-того элемента signal. Из каждого списка выбирается функция (случай) в зависимости от условия. Из формируемого списка результатов обработки каждого случая выбирается единственный нужный вариант путем повторного применения выбора варианта. Функция signal возвращает пустой список вне зависимости от своего аргумента.

Рассмотрим в качестве примера вспомогательную функцию вычисления суммы элементов вектора (Рисунок 6):

```
VecSum << funcdef Param {
  Len << Param:|;                    //длина входного списка данных
  variant<< ((Len,2):[<,=,>]):?;    //длина меньше, равна или больше 2
  (
    (Param : [([], signal, signal):variant]),  //меньше 2, возвращаем аргумент
    (Param : [(signal, +, signal):variant]),  //равна 2, возвращаем сумму двух элементов списка
    (Param : [(signal, signal, Alpha):variant]) //больше двух, передаем в функцию Alpha
  ):[variant]:[]   //фильтрация (выбор) результата
>>return
}
```

Рисунок 6 – Функция вычисления суммы элементов вектора

Значение variant в данном случае является сигналом выбора функции из функций +, Alpha или пустой функции пропуска аргумента, а также сигналом выбора выходного результата. При синтезе в схему СБИС данный сигнал используется как сигнал “валидности” входных данных для конкретной функции и как сигнал мультиплексора выходных значений. Получаемая для данной функции схема представлена на рисунке 7.

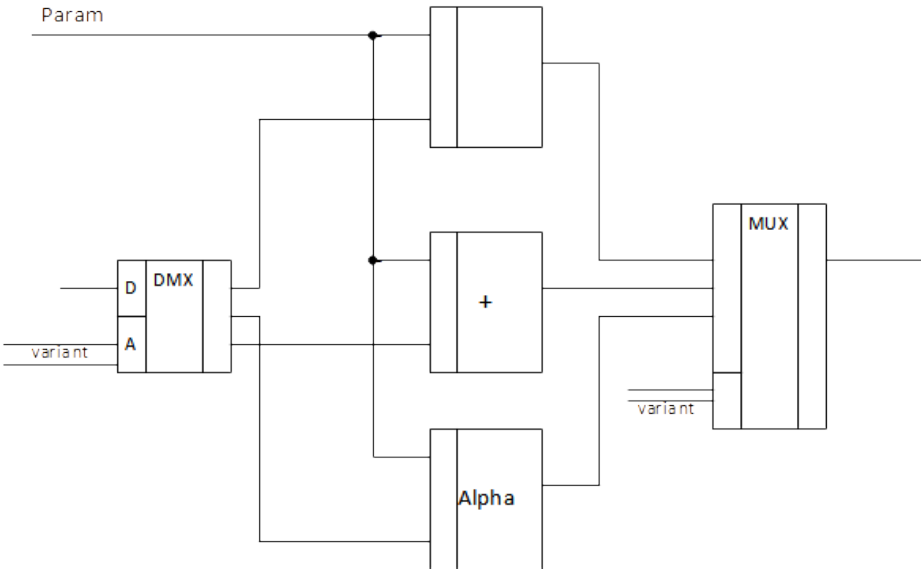


Рисунок 7 – Схема синтеза функции выбора варианта без использования задержанного списка

2.3 Язык ФПП программирования для синтеза СБИС

На основе ФПП модели разработан функциональный язык параллельного программирования «Пифагор» [41]. Язык «Пифагор» обеспечивает текстовое представление информационного графа программы. Описание языка приведено в [41, 42].

Учитывая изменения для платформы СБИС, вносимые в исходную ФПП модель, в качестве языка для разработки метода высокоуровневого синтеза можно использовать либо исходный язык “Пифагор”, либо учитывая все изменения, разработать отдельный проблемно-ориентированный язык. Так как исходный язык разрабатывался как язык переносимых параллельных программ, его использование на еще одной платформе параллельных вычислений укладывается в изначальную концепцию архитектурно-независимых параллельных вычислений.

На этапе разработки описания алгоритмов функционирования необходимо обеспечить поддержку в языке всех изменений, внесенных в ФПП модель для описания архитектуры и алгоритмов функционирования СБИС. Для сохранения исходного описания алгоритма переносимым, изменения в модели учитываются на этапе промежуточных представлений, используемых при переходе к архитектурно-зависимой реализации алгоритма. Изменения, связанные с

типизацией, определяются через отдельный файл ограничений. Обработка рекурсии и отложенных вычислений реализуется на этапе обработки промежуточного представления.

Для исполнения ФПП программ на различных параллельных архитектурах возможны несколько вариантов преобразования. Первый вариант преобразования - компиляция в машинный код соответствующей архитектуры. Второй вариант - трансляция в формат промежуточного представления, который в дальнейшем интерпретируется или подвергается дополнительным преобразованиям.

Для обеспечения эффективности эквивалентных преобразований и архитектурной независимости, для синтеза схемы СБИС предлагается использовать исходный код на ФПП языке, транслируемый в промежуточное представление. В языке “Пифагор” промежуточное представление включает в себя информационный граф (ИГ), задающий схему обработки данных и зависимости по данным и может включать управляющий граф (УГ), задающий управление вычислениями. В простейшем случае промежуточное представление включает в себя только информационный граф. В этом случае управляющий граф можно считать заданным неявно.

Для явного управления вычислениями необходимо описание управляющего графа в явном виде. Это позволяет реализовывать различные стратегии управления вычислениями путем преобразования управляющего графа при переходе к различным целевым архитектурам, таким образом, преобразовывая параллелизм с учетом требований СБИС - проекта.

В процессе трансляции программы формируется информационный граф. Управляющий граф программы генерируется статически или динамически. Динамический вариант управляющего графа получается непосредственно в процессе исполнения программы. Статический вариант генерируется при обходе информационного графа программы. Второй вариант для платформы СБИС является необходимым, так как в этом случае имеется возможность преобразования управляющего графа перед выполнением программы и преобразование параллелизма под конкретную целевую платформу.

При трансляции построение промежуточного представления ФПП программы осуществляется в два этапа. После синтаксического анализа исходного кода программы выполняется построение информационного графа. Информационный граф является исходными данными для построения управляющего графа. Для одного информационного графа можно построить множество управляющих графов: для последовательной, максимально-параллельной либо любой другой стратегии управления.

Для учета изменений, внесенных в модель, вводится дополнительный промежуточный слой, называемый HDL-графом. Данный слой получается путем преобразований информационного графа программы. Далее кратко описаны исходный информационный и управляющие графы программы. Подробное описание приведено в [44]. Дополнительный слой – HDL-граф – описан в главе 3.

2.3.1 Информационный граф

Информационный граф (ИГ) [44, 90] формируется на основе исходного кода программы на языке программирования Пифагор.

Описание информационного графа состоит из следующих элементов:

1) ребро графа. Ребро графа соединяет вершину приемника и источника;

2) задержанный список. Каждому задержанному списку присваивается номер. Номер списка представляет собой константу. Номер списка используется впоследствии для обработки списка;

3) константы данных. Для описания констант выделен отдельный раздел ИГ. Константа в ИГ заменяется ссылкой на описание внутри этого раздела.

Вершины ИГ могут быть следующих типов:

1) константа (const). Константа может быть строкой, символом или числом. Константа также может ссылаться на задержанный список. Содержимым вершины константы является ссылка на описание константы в соответствующем разделе ИГ;

2) вершина аргумента функции (arg). Содержимое вершины аргумента при трансляции неизвестно. В процессе исполнения программы аргумент преобразуется в константу;

3) вершина списка данных («(---)»). Используется для создания списка данных из атомарных значений. Количество входов данной вершины равно количеству элементов данных в списке;

4) вершина параллельного списка («[---]»). Используется для создания параллельного списка из входных данных. Количество входов и выходов вершины одинаково и равно количеству элементов списка;

5) вершина операции интерпретации («:»). Входами вершины являются ребра данных и функции. Количество ребер может быть от 1 до N в случае параллельного списка данных или функций;

6) вершина возврата результата (return). Используется для возврата значения из функции. Вершина имеет один вход.

2.3.2 Управляющий граф

Управляющий граф ФПП программы [44] является ациклическим графом, по которому распространяются управляющие сигналы. Сигналы, формируемые в каждой вершине управляющего графа, связаны с соответствующей вершиной информационного графа. Вершина управляющего графа определяет правила обработки управляющих сигналов. Передача сигнала из вершины УГ в вершину ИГ обеспечивает выполнение старта вычислений в соответствующей вершине ИГ.

Процесс построения УГ связан с созданием управляющей вершины для каждой из вершин ИГ. Связи УГ определяют пути передачи сигналов, информирующих о готовности данных. Сигнал внутри УГ задается парой (x, y), где x - номер вершины приемника сигнала, y - номер входа вершины, на который поступает сигнал. Логика обработки сигналов внутри вершины УГ определяется типом соответствующей вершины ИГ. В первоначальном виде число вершин в УГ совпадает с числом вершин в ИГ. В ходе оптимизации и изменения

параллелизма алгоритма может меняться количество вершин УГ и связи между ними.

Разделение логики управления и преобразования данных дает возможность преобразовать потоки управляющих сигналов, оптимизируя их под конкретную архитектуру. Выбор стратегии управления, задаваемой управляющим графом, может определяться как оптимизацией, связанной с избавлением от избыточных связей, так и архитектурой целевой платформы, на которой будет реализовываться алгоритм. Для изменения параллелизма алгоритма под конкретную платформу достаточно произвести преобразования УГ, без изменения ИГ.

Управляющий граф состоит из следующих типов вершин:

1) вершина константы;
2) вершина аргумента;
3) вершина результата;
4) вершина операции интерпретации;
5) вершина списка данных;
6) вершина параллельного списка;
7) вершина асинхронного списка.

Каждому типу вершины можно сопоставить конечный автомат, реализующий логику обработки управляющих сигналов.

Вершины УГ константы, аргумента и результата имеют один выход. Вершина константы не имеет входов, вершины аргумента и результата имеют по одному входу. Вершина УГ константы реализует простую логику работы – постоянная выдача сигнала на свой выход. Вершины аргумента и результата транслируют сигнал на свой выход при появлении сигнала на входе. Автомат вершин аргумента и результата представляет собой задержку входного сигнала.

Вершина УГ списка данных содержит N входов (где N – размерность списка) и один выход. Логика работы данной вершины следующая – автомат внутри вершины накапливает N входных сигналов и после этого формирует выходной сигнал.

Логика работы вершины УГ параллельного и асинхронного списка определяет логику обработки параллельных вычислений. Преобразование этих вершин позволяет преобразовывать параллелизм алгоритма под разные целевые платформы.

Элементы параллельного списка независимы друг от друга. Вершина имеет в общем случае N входов, определяемые размером списка, и M выходов. При трансляции по умолчанию значение M равно 1, что определяет последовательную логику обработки элементов параллельного списка. Появление каждого элемента списка вызывает формирование одного выходного сигнала – последовательную обработку одного элемента из списка.

Преобразование параллелизма в вершине УГ параллельного списка может происходить как путем изменения M - добавления выходных дуг, так и изменением логики работы вершины УГ параллельного списка. Возможные способы и алгоритмы преобразования параллелизма будут рассмотрены далее.

Вершина УГ обработки оператора интерпретации в общем случае N сигналов готовности данных и M сигналов готовности функции. Формирование пары сигналов готовности данных и функции вызывает формирование сигнала в соответствующую вершину РИГ - срабатывание оператора интерпретации. Количество операций, выполняемых параллельно в данной вершине оператора интерпретации, определяется как M*N.

Рассмотрим пример построения информационного и управляющего графа для программы умножения массива чисел на скаляр (рисунок 8).

```
Mult_VS << funcdef VS
{
        V << VS:1; // вектор
        S << VS:2; // скаляр
        X<< (S,V:|):dup; //вектор X из скаляров S длинной, одинаковой с вектором V
        return << (V,X):#:[]:*:(.)
};
```

Рисунок 8 – Программа умножения вектора на скаляр

Первая операция, X = (S,V:|):dup формирует вектор из скаляров S, длиной, равной длине вектора V. Операции # - транспонирование матрицы, и [] – преобразование в параллельный список – формируют массив вида [(v1,s),(v2,s) ….]. Далее следует попарное умножение и формирование результирующего вектора. Информационный и управляющий графы для данной программы приведены на рисунке 9.

Константы на графическом представлении УГ обозначаются эллипсом, списки и операции интерпретации – прямоугольником со скругленными углами. Цифры над дугами обозначают номер входа в следующей вершине для данной дуги.

2.4 Стратегии управления вычислениями и преобразование параллелизма

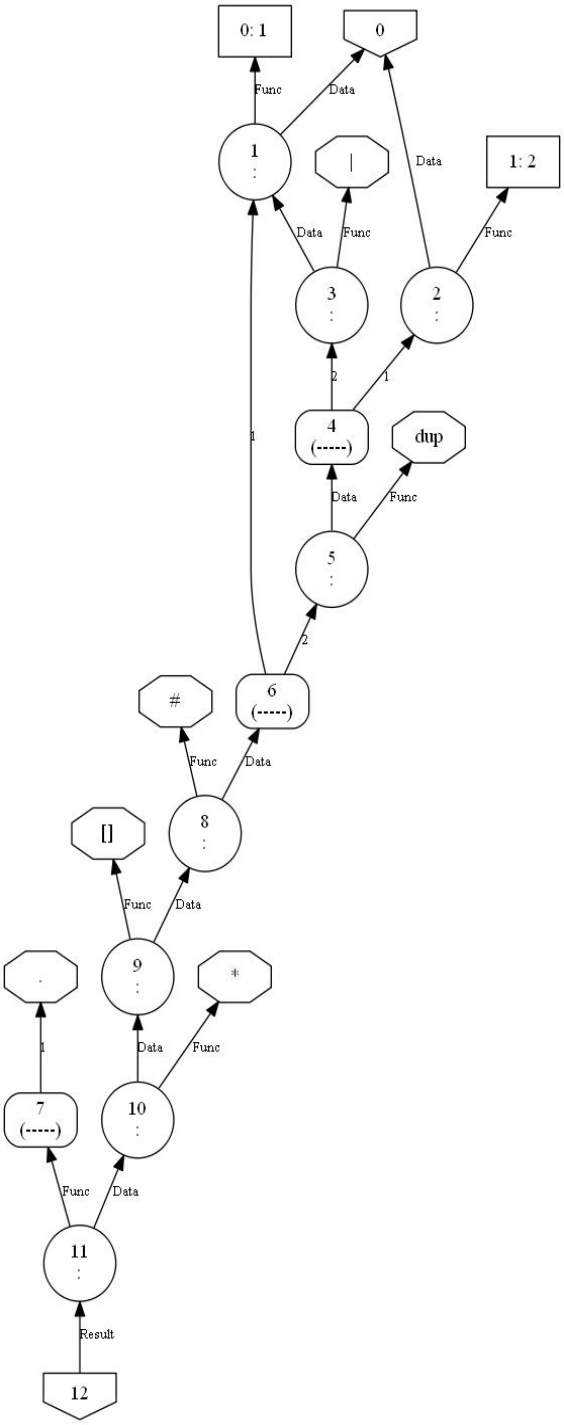
Помимо преобразований, применяемых к ИГ, и описанных выше, при переходе от модели с неограниченными ресурсами к реальным вычислительным системам необходимо выполнять преобразование максимального параллелизма, заложенного в исходный алгоритм, под конкретные ограничения целевой платформы.

Такие преобразования меняют степень параллелизма алгоритма – т. е. количество параллельных операций, выполняемых в различных частях алгоритма.

Крайними вариантами степени параллелизма реализации алгоритма являются полностью последовательное выполнение и выполнение алгоритма с той степенью параллелизма, которая заложена в исходное описание. Для изменения стратегии вычисления необходимо изменять управляющий граф. Все подобные преобразования никак не зависят от информационного графа и не изменяют его.

Начальный вариант УГ формируется транслятором языка Пифагор. Этот вариант УГ формируется с использованием стратегии управления по готовности данных – максимальной степени параллельности алгоритма, заложенной в его описание. Управление вычислительными ресурсами в таком варианте не задается.

Для перехода на платформу СБИС необходимо преобразовать информационный граф программы к конвейерной схеме. Конвейерная схема вычислений представляет собой ярусно-параллельную форму информационного графа. В исходном информационном графе может последовательно продвигаться несколько разметок графа, независимых друг от друга. В каждый момент времени очередной набор данных занимает одну стадию конвейера.



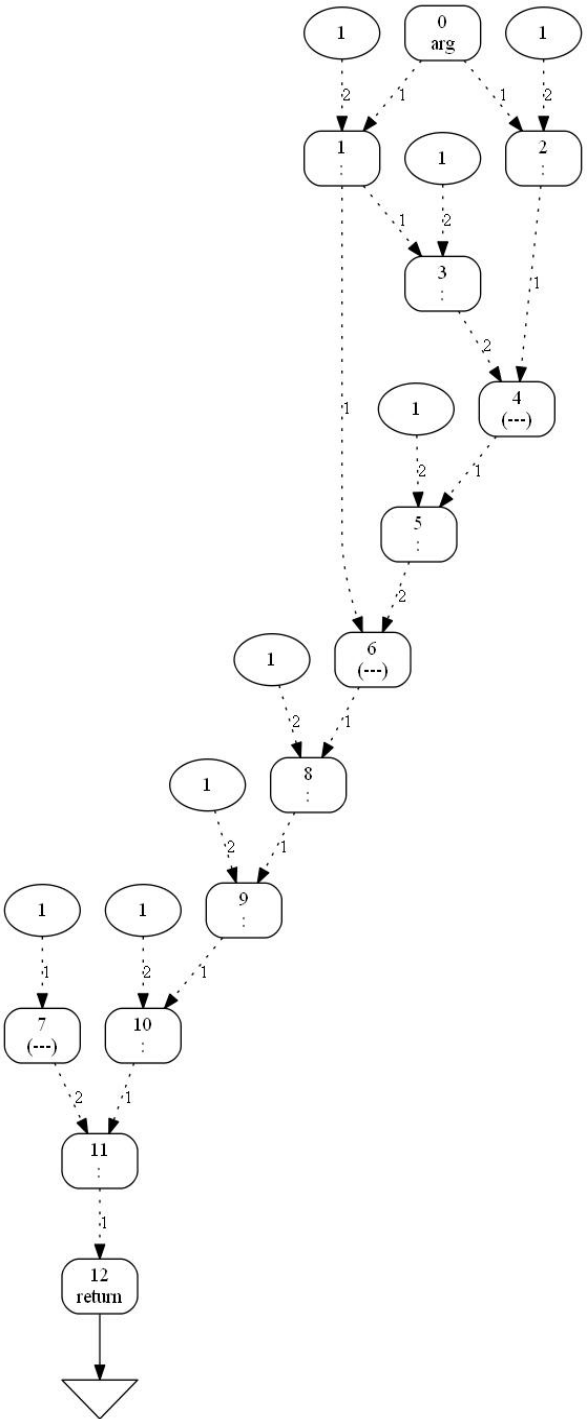


Рисунок 9 – Информационный (слева) и управляющий (справа) графы программы

Для решения задачи преобразования параллелизма под конкретные ресурсные ограничения необходимо:

- определение границ изменения параллелизма;
- реализация алгоритма изменения параллелизма;
- оценка полученного результата по ресурсным ограничениям.

Оценка степени изменения параллелизма предполагает расчет количества стадий конвейера для последовательной и максимально-параллельной схемы. Изменение степени параллелизма предполагает изменение количества стадий конвейера и связанное с этим изменение ресурса.

Рассмотрим алгоритмы управления вычислениями и преобразования параллелизма, реализуемые в методе высокоуровневого синтеза. Исходными данными являются управляющий граф и ресурсные ограничения, накладываемые целевой платформой.

Для оценки степени изменения параллелизма необходимо оценить крайние варианты по степени параллелизма, которые можно получить из исходного описания. Такими вариантами являются последовательное выполнение и максимально-параллельный вариант. Оценка последовательного и максимально параллельного варианта производится по двум метрикам: длине конвейера Lk (задержке схемы) и максимальному количеству параллельных операций в одной стадии конвейера Pk. Между этими двумя вариантами возможно преобразование параллелизма и получение промежуточным вариантов по задержке схемы и параллелизму (занимаемым ресурсам).

Информационный и управляющий графы промежуточного представления являются ацикличными ориентированными графами (Directed Acyclic Graph, DAG) [88, 89]. Для оценки задержки выполнения схемы необходимо найти критический путь в управляющем графе для последовательной схемы и максимально-параллельной схемы. Алгоритм, выполняющий оценку задержки схем, состоит из следующих шагов:

1) сгенерировать УГ для последовательного и максимально-параллельного варианта управления вычислениями;

2) провести оптимизирующие преобразования полученных графов;

3) вычислить критический путь графа.

Управляющий графы в данном алгоритме синтезируются по HDL информационному графу, так как HDL граф является оптимизированным и содержит типы данных. Учет типов данных позволяет точно оценить количество параллельных операций в параллельных участках схемы, таких как параллельные списки - их размерность будет известна. Для учета задержки выполнения при синтезе УГ каждой вершине присваивается метрика, равная задержке выполнения данной вершины. Для вершин интерпретации с параллельным списком размерности N в случае последовательного варианта метрика будет равна N, в случае максимально-параллельного варианта – 1. Вершинам интерпретации, содержащим вызов функции, также присваивается метрика, равная задержке обработки функции.

Для получения последовательного варианта необходимо при генерации УГ выполнить обход узлов информационного графа в последовательности, не нарушающей корректность вычислений. Максимально-параллельный вариант получается при генерации УГ для варианта по готовности данных (вариант, генерируемый по умолчанию) при условии одновременной готовности всех данных на входе функции.

Обобщая, алгоритм вычисления максимальной Lk_max и минимальной Lk_min длины конвейера схемы следующий:

1) генерация управляющего графа со стратегией управления по готовности данных на основе оптимизированного информационного HDL графа;

2) генерация последовательного управляющего графа по оптимизированному информационному HDL графу:

   a) установка сигнала готовности в вершину аргумента;

   b) выбор одного из двух вариантов:

i. выбор любой вершины интерпретации, у которой установлены все сигналы готовности на входе и генерация соответствующего типа вершины УГ;

ii. выбор всех вершин списка данных/параллельного списка, у которых установлены все сигналы готовности на входе и генерация соответствующего типа вершины УГ;

c) соединение текущей вершины/вершин УГ с предыдущей вершиной УГ;

d) повтор шагов b, с пока список необработанных вершин ИГ не пустой.

3) присвоение метрик вершинам полученных УГ:

a) вершине аргумента и результата присваивается метрика 1;

b) вершинам списка данных присваивается метрика 1;

c) вершинам интерпретации присваивается метрика:

i. для последовательного варианта:

1) в случае вызова функции – метрика равная задержке функции;

2) в случае параллельного списка аргумента или функций – суммарной метрике последовательного выполнения параллельного списка.

ii. для параллельного варианта:

1) в случае вызова функции – равная задержке функции или максимальной задержке из всех функций параллельного списка;

2) в остальных случаях метрика равна 1.

4) удаление вершин, не влияющих на расчет критического пути в УГ:

4.1) вершин локальных констант;

4.2) вершин внешних ссылок;

4.3) вершин констант с предопределенными функциями.

5) Вычисление длины критического пути [88] с учетом метрик для последовательного варианта Lk_max и максимально-параллельного Lk_min;

Для определения максимального количества параллельных операций Pk необходимо привести управляющий граф для максимально-параллельного

варианта к ярусно-параллельной форме (ЯПФ) с высотой, равной Lk_min и определить максимальную ширину ЯПФ [88].

Рассмотрим алгоритм вычисления максимальной и минимальной задержки схемы (длины конвейера) на примере. В качестве примера возьмем функцию “бабочки” алгоритма БПФ radix-2. Исходный код функции на языке Пифагор приведен на рисунке 10.

```
BPF << funcdef Arg {
  x11 << Arg:1;
  x12 << Arg:2;
  x21 << Arg:3;
  x22 << Arg:4;
  w1 << Arg:5;
  w2 << Arg:6;
  // комплексное умножение
  multx << [(x21,w1),(x22,w2),(x22,w1),(x21,w1)]:*;
  mult << ( (multx:1,multx:2):-,(multx:3,multx:4):+);
  mult1 << mult:1;
  mult2 << mult:2;
  (((x11,mult1):+,(x12,mult2):+), ((x11,mult1):-,(x12,mult2):-)) >>return
}
```

Рисунок 10 – Программа вычисления БПФ Radix-2

Исходный информационный граф программы приведен на рисунке 11.

В данном случае основным преобразованием при формировании HDL графа было удаление операций выбора данных из списка, так как в HDL данная операция не занимает ресурса. Соответствие положений элементов в списках на HDL графе обозначается X->Y, где X – положение элемента в исходном списке, Y – в списке результата.

Следующим шагом алгоритма является формирование УГ для последовательного и максимально-параллельного варианта управления вычислениями.

Управляющий граф для управления по готовности данных, из которого формируются все остальные варианты, приведен на рисунке 13. Приведя этот

граф к ярусно-параллельной форме, получим что задержка (длина конвейера) Lk_min равна 8.

Формирование УГ для последовательного варианта управления представлено в таблице 4 и графическое представление результата на рисунке 14.

На каждом шаге, согласно алгоритму, выбираются либо одна вершина интерпретации, которая может сработать на текущем шаге, либо все остальные вершины, которые также могут сработать на данном шаге.

Также при формировании УГ для оценки задержки схемы удаляются вершины констант (1), так как эти вершины не влияют на максимальный критический путь графа.

HDL граф, получаемый после преобразования из ИГ, приведенный на рисунке 11, показан на рисунке 12.

Таким образом, минимальная Lk_min и максимальная Lk_max длина конвейера этой схемы составляет 8 и 15 тактов. При этом максимальное значение количества параллельных операций Pk составляет 4 (Таблица 4, шаг 7).

Учитывая ресурсные ограничения конкретной аппаратной платформы алгоритм из этого примера можно преобразовать в любой вариант с задержкой от 8 до 15 тактов и с использованием параллельных арифметических операций от 1 до 4.

Таблица 4 – Шаги алгоритма формирования УГ максимально-параллельного управления

Шаг	**Список вершин**	**Вершины интерпретации**	**Остальные вершины**
1	7,8,9,10,11,12,15,16,19,20, 21,22,23,24,25,26,27,28,29,30		7,8,9,10
2	11,12,15,16,19,20, 21,22,23,24,25,26,27,28,29,30		11
3	12,15,16,19,20, 21,22,23,24,25,26,27,28,29,30	12	
4	15,16,19,20,21,22, 23,24,25,26,27,28,29,30		15,19
5	16,20,21,22,23,24,25, 26,27,28,29,30	16,20	

6	21,22,23,24,25, 26,27,28,29,30		21,23,25,27
7	22,24,26,28,29,30	22,24,26,28	
8	29,30		29

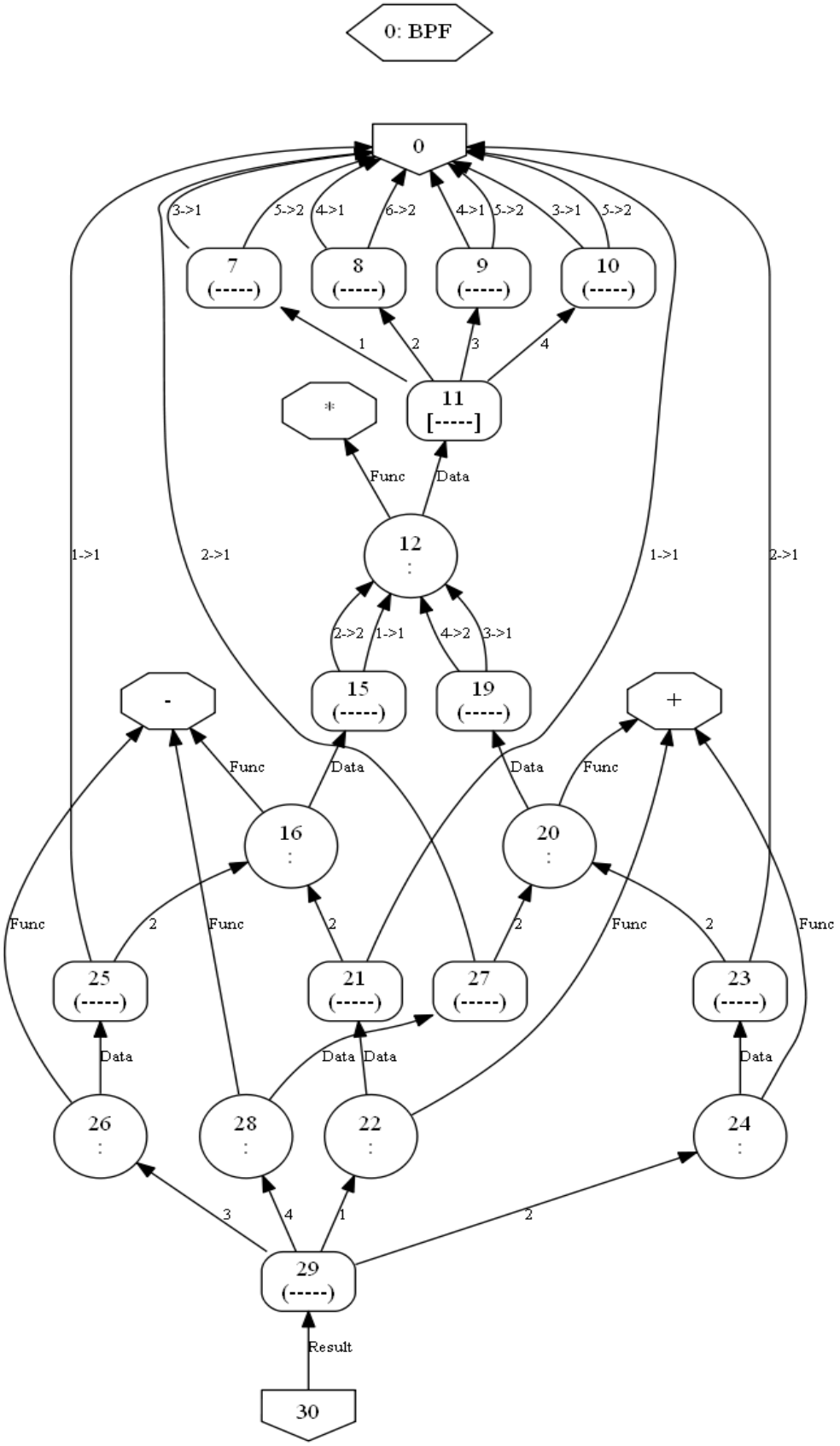


Рисунок 11 – Информационный граф алгоритма БПФ radix-2

Рисунок 12 – HDL граф алгоритма БПФ radix-2

Как показано в [72], характер обработки данных (параллелизм), можно условно разделить на 4 типа:

1) последовательный;
2) параллельно-независимый;
3) конвейерный;
4) параллельно-зависимый.

Для описания параллелизма в языке ФПП Пифагор используются 2 конструкции: параллельный и асинхронный список [75]. Все преобразования параллелизма в ФПП модели реализуются эквивалентными преобразованиями этих языковых конструкций.

Таблица 5 – Шаги алгоритма формирования УГ последовательного управления

Шаг	**Список вершин**	**Вершины интерпретации**	**Остальные вершины**
1	7,8,9,10,11,12,15,16,19,20, 21,22,23,24,25,26,27,28,29,30		7,8,9,10
2	11,12,15,16,19,20, 21,22,23,24,25,26,27,28,29,30		11
3	12,15,16,19,20, 21,22,23,24,25,26,27,28,29,30	12	
4	-//-	12	
5	-//-	12	
6	-//-	12	
7	15,16,19,20,21,22, 23,24,25,26,27,28,29,30		15,19
8	16,20,21,22,23,24,25, 26,27,28,29,30	16	
9	20,21,22,23,24,25, 26,27,28,29,30	20	
10	21,22,23,24,25, 26,27,28,29,30		21,23,25,27
11	22,24,26,28,29,30	22	
12	24,26,28,29,30	24	
13	26,28,29,30	26	

14	28,29,30	28	
15	29,30		29

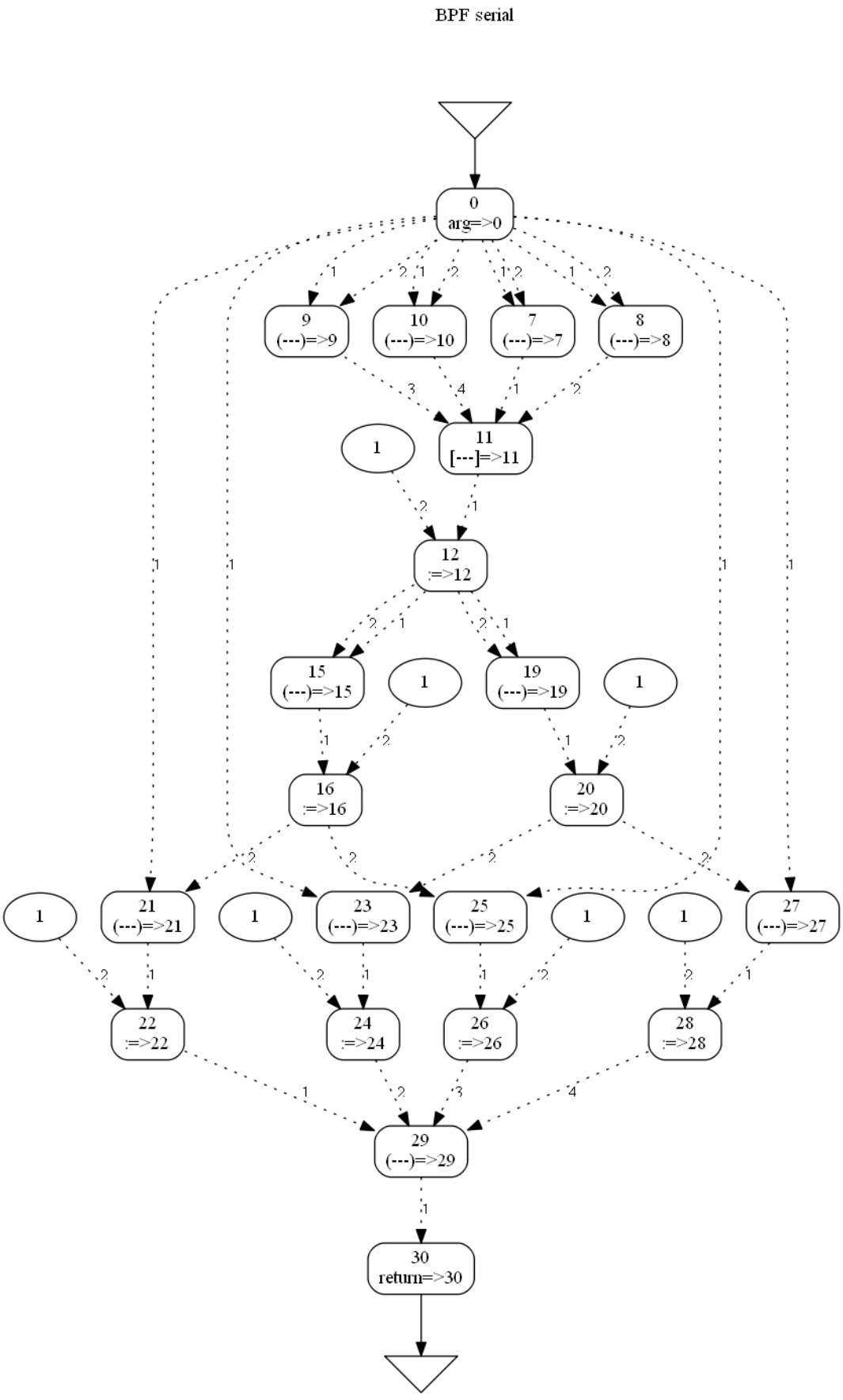


Рисунок 13 – УГ по готовности данных для HDL графа алгоритма БПФ radix-2

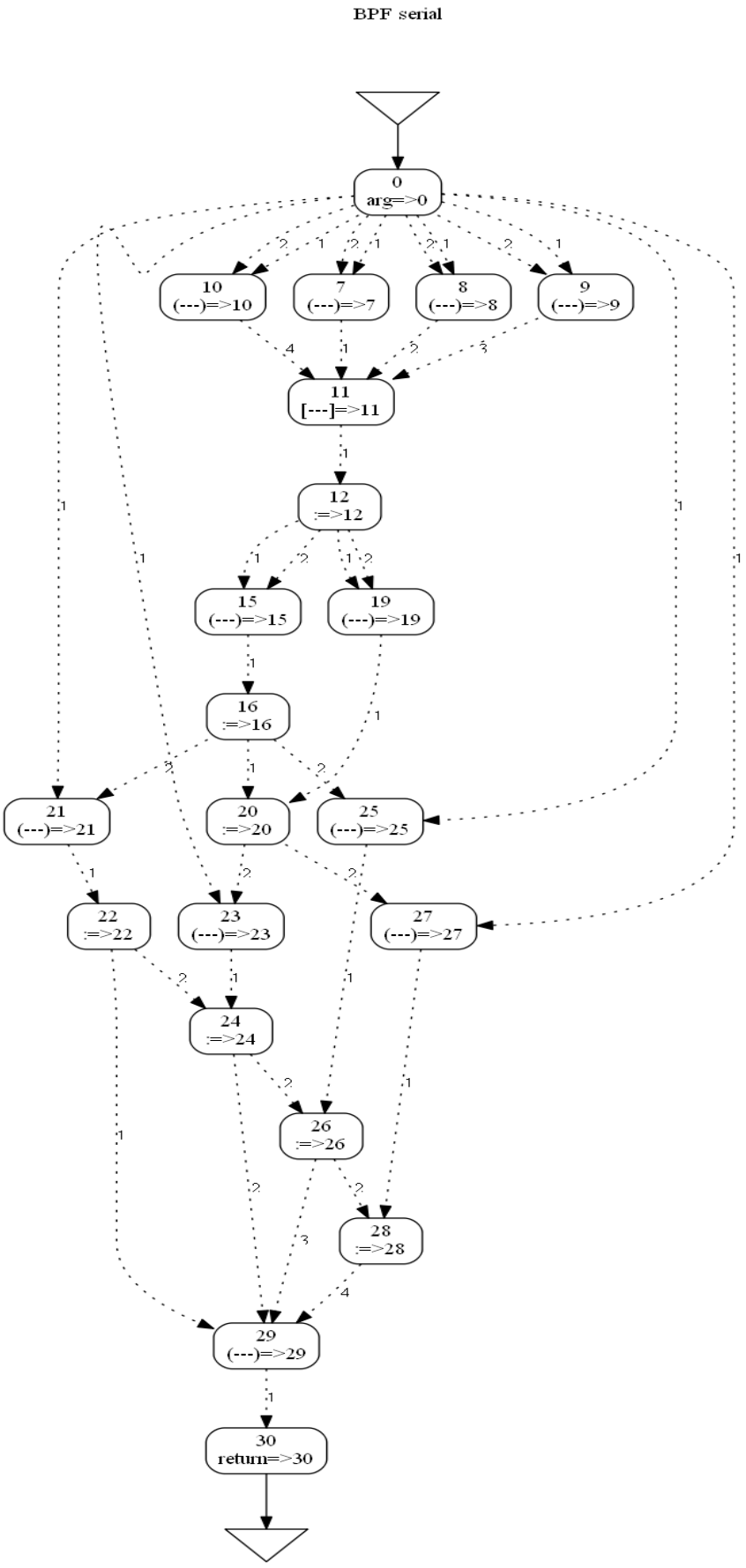


Рисунок 14 – УГ последовательного управления для алгоритма БПФ radix-2

2.4.1 Обобщенный алгоритм преобразования параллелизма

Рассматривая задачу преобразования параллелизма в общем виде можно выделить 3 варианта преобразования исходной максимально-параллельной схемы в зависимости от доступного ресурса целевой платформы. Под доступным ресурсом целевой платформы при этом понимается наименьший из ресурсов – памяти или вычислительного ресурса. Обозначим отношение доступного на платформе ресурса к ресурсу, требуемому для реализации в исходном максимально-параллельном виде, как P. Возможные варианты отношения следующие:

1) $P < 0.5$, возможна индукция(увеличение) количества схем;
2) $0.5 < P < 1$, ресурса достаточно для размещения 1 максимально-параллельного варианта схемы;
3) $P > 1$, необходима редукция параллелизма.

Первый вариант не требует преобразований максимально-параллельной схемы, при этом в нем возможно увеличение производительности путем размещения нескольких схем параллельно:

$$S = int(1/P).$$

Второй вариант схемы также не требует преобразований. Для третьего варианта схемы рассмотрим алгоритм редукции с использованием коэффициентов редукции, описанных в п.1.1/1.2. Алгоритм состоит из следующих шагов:

1) рассчитываем коэффициент редукции R;
2) редуцируем параллелизм схемы на R_{max};
3) для измененной схемы пересчитываем коэффициенты R, если они меньше 1 – завершение алгоритма;
4) 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 R_{ram} и $R_{выч}$;
5) если какой-либо из коэффициентов больше 1:

$R_{max}=R_{max}+1$

и возврат 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).

Выбор конкретного варианта реализации схемы из всех возможных осуществляется на этапе решения задачи выбора оптимального решения по заданным ограничениям - Design Space Exploration (DSE).

2.5 Выводы

1. В результате адаптации ФПП модели вычислений для описания СБИС предложены следующие изменения модели:
   a) введение статической системы типов;
   b) отказ от задержанных вычислений, их преобразование при переходе на платформу СБИС;
   c) преобразование рекурсии в итеративную схему с последующим переходом к конвейерной схеме;
   d) определение глубины рекурсии на этапе трансляции;
   e) методы преобразования параллельных списков;
   f) ограничение на списки данных неизвестной на этапе трансляции размерности.
2. Для реализации статической системы типов в язык добавлено расширение, включающее в себя скалярные и векторные типы данных, а также механизм преобразования (приведения) типов.
3. Для учета всех изменений, вносимых в ФПП модель и язык, добавляется дополнительный слой промежуточного представления – HDL-граф.
4. Для реализации метода сокращения параллелизма при переходе на платформы с разными ресурсными ограничениями предложены:

a. метод оценки максимальной и минимальной длины конвейера (степени параллелизма) задачи по информационному графу;
b. преобразование исходного информационного графа к ярусно-параллельной форме;
c. обобщенный алгоритм преобразования параллелизма.

3 Программные средства для метода высокоуровневого синтеза

3.1 Инструментальная поддержка процесса разработки СБИС

На рисунке 15 представлен функциональный состав разработанных инструментальных средств поддержки проектирования СБИС на основе предложенного метода высокоуровневого синтеза.

Для поддержки процесса высокоуровневого синтеза СБИС разработан комплект программных средств, позволяющих выполнять:

- преобразование исходного кода на ФПП языке в промежуточное представление в виде информационного и управляющего графа;

- выполнять оптимизацию информационного графа, что позволяет повысить эффективность программ на этапе высокоуровневого описания [81];

- проводить отладку и анализ ФПП кода во время выполнения, обеспечивая поиск ошибок и трассировку [78];

- осуществлять формальную верификацию ФПП программ [77].

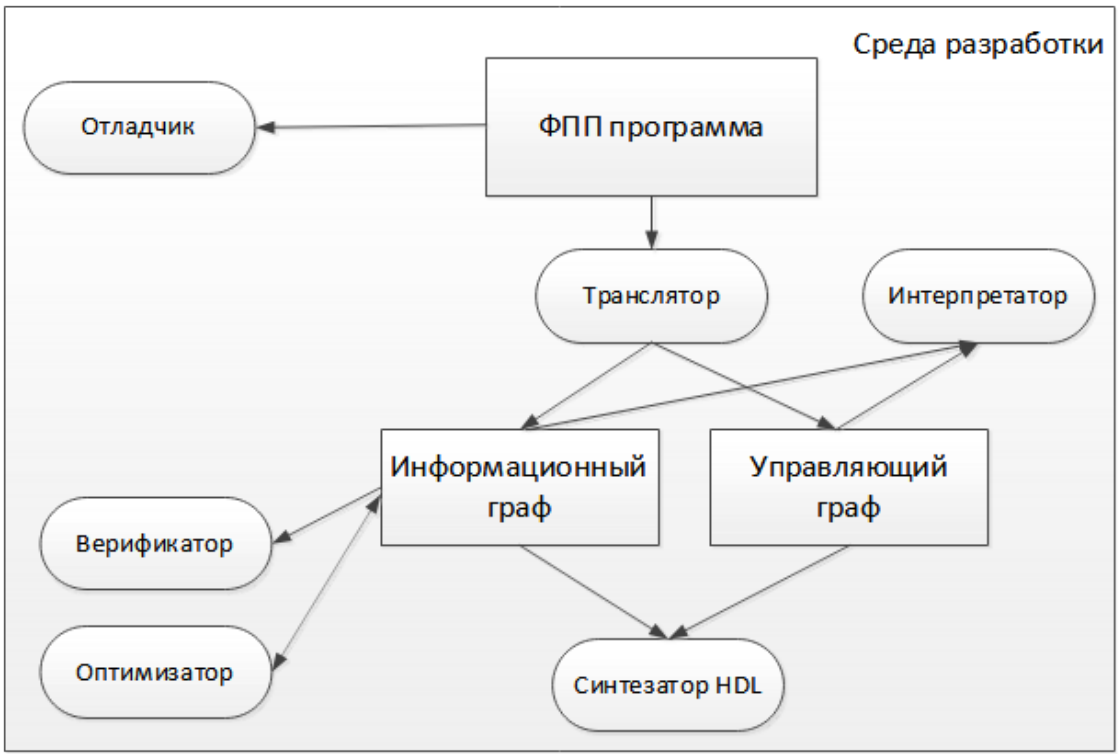


Рисунок 15 – Обобщенная структура инструментальных средств поддержки проектирования

Разработанные инструментальные средства позволяют сформировать набор отлаженных функций для их реализации в виде СБИС.

Состав инструментальных средств разработки функционирует под управлением интегрированной среды разработки (оболочки) предоставляющей пользователю информационные ресурсы для организации процесса высокоуровневого синтеза.

Транслятор с языка “Пифагор” предназначен для синтаксической проверки исходного текста и преобразование в промежуточное представление в виде информационного (ИГ) и управляющего (УГ) графа. Сформированный ИГ полностью задает информационный слой программы и частично данные – константные выражения. Полученный ИГ транслятор может сохранить в одном из трех видов – текстовом, двоичном и графическом. В состав транслятора входит также генератор УГ, который принимает на вход ИГ и строит по нему УГ. Созданный УГ также сохраняется в одном из трех видов – текстовом, двоичном и графическом. Промежуточное представление в виде ИГ и УГ является входным форматом для интерпретатора, оптимизатора, отладчика и синтезатора HDL.

Интерпретатор позволяет исполнить текст программы на ФПП языке. Входными данными для интерпретатора являются информационный и управляющий граф, а также аргумент функции верхнего уровня. Аргумент представляется в формате описания ИГ, для этого он обрабатывается транслятором. Интерпретатор представляет собой событийную машину, которая формирует управляющие сигналы для узлов ИГ и обрабатывает данные в соответствующих узлах ИГ.

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:

1) удаление неиспользуемого кода;
2) оптимизация повторяющихся вычислений;
3) непосредственная подстановка функций;

4) удаление повторяющегося кода;

5) оптимизация на основе эквивалентных преобразований алгебры ФПП модели.

Отметим, что эти преобразования аналогичны методам, используемым при оптимизации программ на прочих языках программирования и их промежуточных представлений. Кроме того, выполняется ряд специфичных для функционально-потоковой модели вычислений при переходе к платформе СБИС оптимизирующих преобразований, в том числе:

1) упрощение параллельных списков;

2) упрощение списков, размер которых известен при компиляции.

Часть оптимизирующих преобразований выполняется на этапе синтеза HDL, в соответствии с процедурами, изложенными в п. 3.3.5.

Для отладки функционально-потоковых программ реализован отладчик [77]. В отладчике реализовано 3 режима отладки: пошаговая отладка, отладка слоев и отладка ветвей.

Последовательный режим отладки реализован в режиме пошаговой отладке. Отсутствие зависимости по данным позволяет проводить последовательную отладку и получать корректный результат независимо от порядка выполнения параллельных участков программы. В параллельных ветвях информационного графа отсутствуют информационные связи - отсутствуют конфликты, такие как появление невычисленных вершин, блокировка в ожидании данных, возникновение конфликтов по данным и т.д.

Параллельным режимом отладки является режим отладки слоев. В этом режиме отладка выполняется по готовности данных. На каждом шаге отладки анализируются вершины информационного графа и выбираются для исполнения вершины с готовыми данными на входах. Режим отладки слоев аналогичен алгоритму работы интерпретатора. Выбранные на каждом шаге вершины вычисляются и помечаются как вычисленные. На следующем шаге для вычисления выбираются вершины, которые могут быть вычислены. Этот режим отладки позволяет увидеть количество тактов конвейера (тактов исполнения)

алгоритма и оценить степень параллелизма алгоритма (количество операторов). Такой подход позволяет контролировать порядок работы алгоритма в параллельной системе.

Режим отладки ветвей является последовательным режимом. В этом режиме выбирается и исполняется одна ветвь информационного графа. В выбранной ветви программы нет зависимостей по данным от других вершин ИГ и эта ветвь может быть выполнена последовательно.

Для трансформации промежуточных представлений (РИГ и УГ) ФПП программ в HDL описание разработан синтезатор HDL. При функционировании синтезатора, в соответствии с разработанным алгоритмом синтеза HDL, на первоначальном этапе осуществляется синтез графовых представлений, в том числе происходит вычисление константных выражений, задание и вычисление типов. На следующем этапе выполняется оптимизация полученных представлений посредством удаления вычисленных вершин. На заключительной стадии синтезируется описание системы на HDL-языке, в том числе происходит генерация входных и выходных портов, генерация таблицы имен (регистров), связывание входных портов с входными регистрами и генерация описания модуля на HDL в режиме обхода информационного графа в режиме “символьной” интерпретации. Далее будут подробно рассмотрены алгоритмы и этапы синтеза, реализованные в синтезаторе с ФПП на HDL, разработанного в рамках проводимых исследований.

3.2 Синтез описания СБИС

Синтезатор HDL выполняет функцию трансляции из промежуточного представления языка Пифагор (ИГ и УГ) в описание схемы на языках xHDL [76]. В качестве выходного языка может быть выбран Verilog или VHDL. Синтезатор представляет собой утилиту, входящую в состав интегрированной среды разработки.

Входными параметрами для синтезатора являются:

1) имя функции верхнего уровня на ФПП языке;

2) выходной HDL язык: Verilog или VHDL;

3) файл типизации и формата аргумента;

4) имя выходного файла с кодом на HDL языке.

Синтаксис файла типизации формата аргумента соответствует описанию типов, приведенных в п. 2.2.1. При загрузке функции верхнего уровня осуществляется компоновка с функциями, являющимися внешними ссылками в данной функции и всех функциях, вызываемых из них.

3.3 Этапы высокоуровневого синтеза

Метод высокоуровневого синтеза на основе ФПП подхода можно разделить на два уровня:

1) разработка исходного архитектурно-независимого описания алгоритма функционирования СБИС на ФПП языке;

2) реализация конкретного варианта архитектуры СБИС с использованием алгоритма на ФПП языке и проектных ограничений (ресурсных и размерностей данных).

Первый уровень, архитектурно-независимый, заканчивается получением промежуточного представления ФПП отлаженной программы в виде ИГ и УГ. Уровень архитектурно-зависимой разработки начинается с задания ограничений по ресурсам и типам данных и заканчивается получением описания алгоритма СБИС на HDL языке. При этом данный уровень может использоваться многократно для различных ограничений без возврата на предыдущий, архитектурно-независимый, уровень.

Уровень разработки архитектурно-независимого описания СБИС на основе ФПП представления исходных алгоритмов состоит из следующих этапов:

1) разработка алгоритмов функционирования СБИС на языке ФПП программирования. На данном этапе осуществляется высокоуровневое архитектурно-независимое представление исходных алгоритмов с максимальной степенью параллелизма;

2) тестирование программного кода. В соответствии с заданием на проектирование по программному коду происходит отладка алгоритмов функционирования СБИС без привязки к целевой платформе;

3) синтез промежуточного представления СБИС в виде информационного и управляющего графов. Выполняется синтез промежуточных представлений, происходит формирование промежуточного представления типов аргументов и констант в соответствии с полученным программным кодом.

Архитектурно-зависимый уровень разработки СБИС состоит из следующих этапов:

1. Типизация данных. Выполняется типизация данных согласно заданным описаниям типов.

2. Оптимизация промежуточных представлений. Выполняется оптимизация информационного и управляющего графов.

3. Синтез управляющего и информационного HDL-графа. Выполняется промежуточный синтез графов данных и управления из информационного и управляющего графа. Реализуется при необходимости организации переходов к соответствующему этапу традиционного HLS или организации совмещенных циклов традиционного и ФПП синтеза.

4. Решение задачи планирования и распределения ресурсов с изменением степени параллелизма реализации алгоритма, выбор схемы управления вычислениями.

5. Синтез схемы обработки данных и сигналов управления.

6. Синтез описания на HDL языке.

7. Синтез регистрово-вентильного представления СБИС.

При этом начальные и заключительные этапы: постановка задачи, формирование технического задания на проект и пакета спецификаций, синтез регистрово-вентильного представления СБИС полностью соответствуют традиционному маршруту проектирования. Рассмотрим каждый этап подробнее.

3.3.1 Разработка алгоритма на языке ФПП программирования

Первоначальное описание алгоритма, реализуемого на кристалле, осуществляется средствами интегрированной среды проектирования с использованием языка функционально-потокового параллельного программирования Пифагор.

На верхних уровнях иерархии при описании алгоритма СБИС можно не использовать типизацию данных. Наличие специальных конструкций языка и отсутствие типизации данных позволяет обеспечить полную архитектурную независимость проекта СБИС. Также описание алгоритма функционирования СБИС без привязки к типам и размерностям данных позволяет получить из одного описания множество вариантов реализаций.

3.3.2 Тестирование программного кода

Процесс верификации и тестирования программ на ФПП языке упрощен по сравнению с другими подходами и архитектурами параллельного программирования. В ФПП языке отсутствуют конфликты и зависимости по данным, что позволяет исключить анализ ресурсных конфликтов.

Для всех базовых функций языка определен набор правил их функционирования и преобразования. Используя данные правила выполняется анализ корректности программы. В результате производится формальная верификация программы [77, 78, 79].

Под формальной верификацией подразумевается доказательство соответствия программы её спецификации. Тестирование направлено на поиск ошибок, в то время как формальная верификация доказывает отсутствие ошибок в программе. Формальная верификация является строгим математическим доказательством соответствия программы её спецификации.

3.3.3 Синтез информационного и управляющего графов

Формируемый при трансляции ФПП программы информационный граф является описанием СБИС на ФПП языке, представленном в виде промежуточного формата. Различие в принципах представления основного и промежуточного формата для исходной программы заключается в описании связей. В программном коде описание происходит «сверху вниз», а в промежуточном, «графовом» представлении, осуществляется в реверсивном порядке, от функции к аргументам.

3.3.4 Типизация данных

После построения информационного графа происходит типизация, то есть определение всех типов аргументов и констант. Для типизации синтезатор использует отдельный формат файла, содержащий описание типов для аргумента данной функции. Использование отдельного от языка задания типов позволяет делать привязку к целевой платформе на этапе перехода к данной вычислительной платформе и сохранить исходное архитектурно-независимое описание алгоритма в неизменном виде. Правила назначения типов и алгоритмы типизации, реализуемые синтезатором, описаны в разделе “Контроль и автоматическое назначение типов на этапе синтеза”.

3.3.5 Оптимизация кода и промежуточного представления

На этапе оптимизации решаются задачи как общей оптимизации кода, возможной в ФПП языке, так и специфичных оптимизационных преобразований, свойственных при переходе к платформе СБИС. Оптимизационные преобразования, возможные в ФПП языке “Пифагор”, описаны в работах [81, 82].

Этап оптимизации состоит из следующих шагов:

1) вычисление константных выражений и операторов интерпретации, известных на этапе синтеза;

2) удаление вычисленных вершин;

3) преобразование рекурсии;

4) удаление неиспользуемых ветвлений;

5) преобразование параллельных списков.

После завершения предыдущего этапа, типизации, становятся известны размерности и типы данных аргумента и вершин. Это позволяет в случае, когда известны данные и функция на этапе трансляции, вычислить выходное значение таких вершин. К таким вершинам относятся:

1. оператор интерпретации с известными данными и со следующими операторами в качестве функций:

a) целые числа – выбор элемента из списка;

b) оператор определения длины списка | ;

c) арифметические операции +, -, *, /, %;

d) операции сравнения <, >, =, !=, >=, <=;

e) оператор определения положения истинных значений ?.

Вычисленное значение заменяется константной вершиной. После завершения вычисления константных выражений данные вершины удаляются из информационного графа. Это влечет за собой изменение связи в ИГ и соответствующее изменение управляющего графа.

Рассмотрим эти этапы оптимизации на примере из листинга 4 (графы на рисунке 13).

Допустим, что аргумент функции на этапе типизации был задан как список данных из 6 целочисленных значений плюс целочисленная константа:

((int, int, int, int, int, int), int).

На первом шаге произойдет вычисление вершин №1 и 2 – выбор частей аргумента (элементов списка аргумента). Вторым шагом вычисление вершины №3 – определение длины списка, равной 6. Третьим шагом сформируется список в вершине №4 – из целочисленной константы длины списка 6 и второго элемента аргумента. На четвертом шаге в вершине 5 сформируется список из 6 вторых элементов аргумента. Результат этих шагов показан на информационном графе

(рисунок 16). Для упрощения введено двойное обозначение связей над дугами вида x -> y, где x – номер элемента в списке вышестоящей вершины, y - номер позиции, куда помещается элемент в нижерасположенной вершине ИГ.

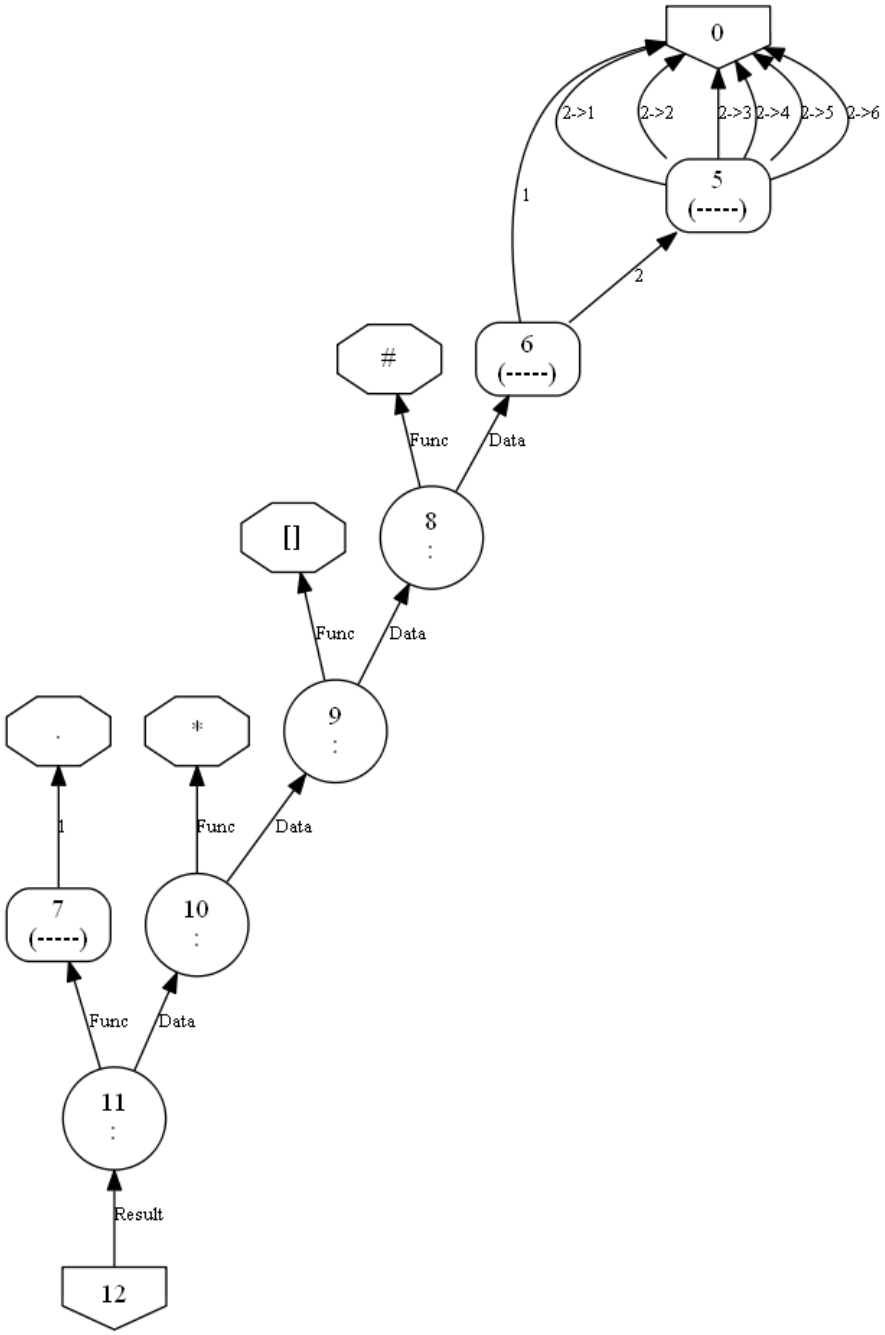


Рисунок 16 – Информационный граф после оптимизирующих преобразований

На следующем этапе будут вычислены вершины 6,8 и сформирован параллельный список 9 из пар элементов (рисунок 17). Этот граф и будет являться результатом этапа оптимизации информационного графа.

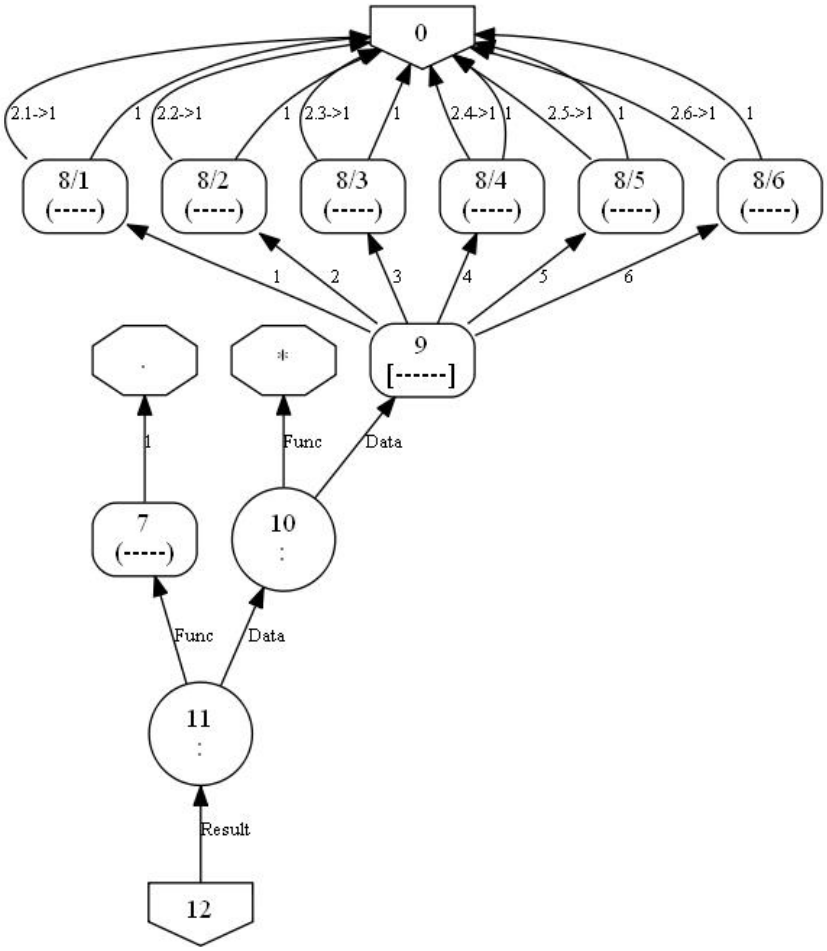


Рисунок 17 – Оптимизированный информационный граф

Соответствующие преобразования проводятся и над управляющим графом. При удалении вершины из информационного графа соответствующая вершина из управляющего графа также будет удалена с соединением связей между выше и нижестоящей вершиной относительно удаляемой. Помимо этого, после вычисления результата для вершин интерпретации данная вершина может менять тип с вершины интерпретации на тип вершины, соответствующий результату операции интерпретации этой вершины. В данном примере вершина №9 после оптимизации преобразуется в вершину параллельного списка.

Преобразования рекурсии и параллельных списков были рассмотрены в п.п. 2.2.2-2.2.3.

3.3.6 Синтез HDL-графа

Информационный и управляющий графы являются промежуточным представлением ФПП программы и не содержат архитектурно-зависимых элементов. При переходе к конкретной платформе СБИС происходит переход к архитектурно-зависимому описанию ФПП - программы. Для учета всех изменений, вносимых в исходный алгоритм, включая типы данных, вводится дополнительное промежуточное представление. Этот промежуточный слой будем называть HDL-графом.

В процессе синтеза исходное промежуточное представление ФПП языка (УГ и ИГ графы) преобразуются в управляющий и информационный HDL - граф. Такой вариант описания повторяет формат ИГ и УГ графов, используемый в промежуточном представлении ФПП языка, и позволяет в дальнейшем производить операции планирования вычислений, преобразования параллелизма используя только управляющий граф, без изменения информационного. Кроме того, графы в HDL формате не содержат циклов, так как в исходном описании на ФПП языке циклы отсутствуют.

Структура данных, соответствующая описанию HDL графа, приведена на рисунке 18.

Для HDL – графа введены следующие типы узлов:

1. узел оператора преобразования данных;

2) узел списка данных;

3) узел параллельного списка;

4) узел асинхронного списка;

5) узел внешней ссылки (вызова функции);

6) константы двух типов: «значения» и «операции»;

7) узел аргумента;

8) узел результата.

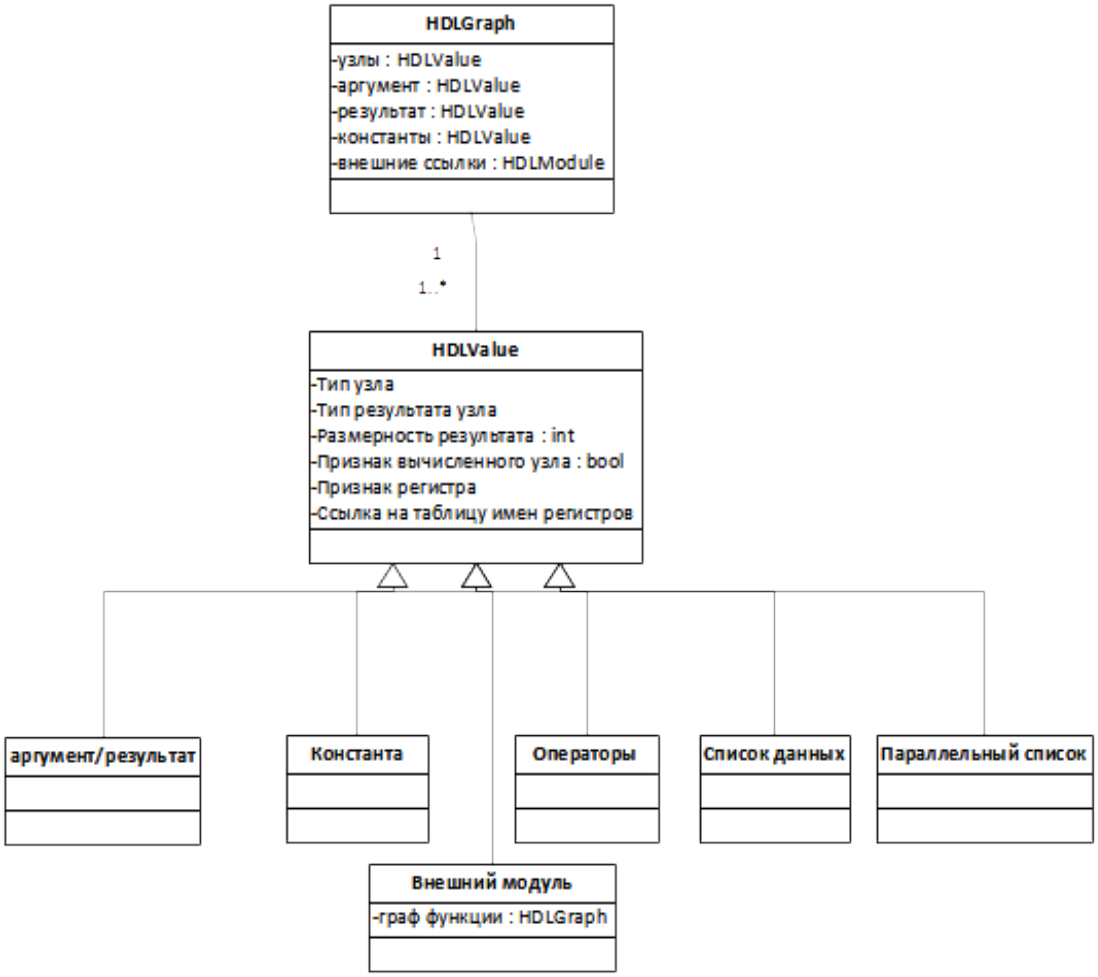


Рисунок 18 – Структура данных промежуточного представления HDL графа

Каждый тип узла содержит также данные о типе результата (целое, беззнаковое, булевское и т.д.) и его размерности, назначаемые на этапе автоматического назначения типов. Кроме того, некоторые типы узлов содержат ссылки на таблицу имен модуля. Под модулем в данном случае понимается один экземпляр функции, реализовываемый в виде модуля на СБИС.

Узел оператора преобразования данных является эквивалентом операции интерпретации языка ФПП. Узел содержит ссылки на узлы аргумента и функции. Узлами аргумента и функции может быть любой из типов узлов 2-7.

Узел внешней ссылки содержит имя функции и ссылку на HDL граф, реализующие эту функцию. Поле тип выходного значения определяет тип результата этой функции. Узлы списков (параллельных, асинхронных) содержат ссылки на таблицу регистров модуля.

Основным отличием HDL информационного графа от РИГ языка Пифагор является, помимо назначенных типов выходных значений каждой вершины, возможность указания связи между элементами списков связанных вершин. Это позволяет проводить оптимизационные преобразования путем вычисления операций выбора данных из списка.

Для синтеза HDL графа этапы, описанные в п. п. 2.13.4-2.13.5 формируют исходные данные. Для получения HDL графа необходимо также выполнить следующие операции:

1. синтез выходных и входных портов;

2. синтез таблицы регистров.

Для синтеза входных портов обрабатывается вершина аргумента. Для каждого элемента в этой вершине генерируется описание порта с соответствующей разрядностью, определенной на этапе типизации. Для выходных портов те же действия производятся для вершины результата.

Синтез таблицы регистров заключается в создании таблицы регистров. Для этого после этапа типизации производится обход графа с созданием для каждой вершины интерпретации, списка данных и параллельного списка записей в таблице регистров в соответствии с определенной разрядностью. Для списков соответственно создается несколько записей в таблице регистров в количестве, равном размеру списка. Каждая запись содержит имя регистра, генерируемое из номера вершины и номера элемента в списке для списков, разрядность регистра, тип знаковый или беззнаковый, а также ссылку на вершину, которой принадлежит данный регистр.

3.3.7 Синтез схемы обработки данных

На этапе синтеза схемы обработки данных по HDL графу генерируется описание на HDL языке. Для этого осуществляется обход HDL графа и синтез соответствующих каждой вершине элементов СБИС. Результат обработки данных помещается в регистр. Регистры для каждой вершины формируются при синтезе таблицы имен.

При синтезе схемы обработки данных по HDL графу соблюдаются следующие правила:

– одна вершина интерпретации – один такт конвейера (в общем случае);

– одна функция – один модуль;

– каждый вызов одной и той же функции – экземпляр модуля;

– вершины аргумента и результата – начальная и конечная стадия конвейера функции.

Процесс синтеза схемы обработки данных состоит из двух этапов:

1) преобразование информационного графа в ярусно-параллельную форму;

2) преобразование каждой вершины в соответствующую схему обработки данных СБИС.

Преобразование в ярусно-параллельную форму и алгоритмы преобразования параллелизма были рассмотрены в п. 2.4.

Рассмотрим процесс преобразования операций ФПП языка в элементы СБИС.

Арифметические операции

К таким операциям относятся сложение/вычитание, умножение. Операция деления не поддерживается. В общем случае генерируется сумматор/умножитель, осуществляющий операцию за 1 такт и записывающий результат в выходной регистр вершины оператора интерпретации.

Операции сравнения

Операции сравнения преобразуются в логическую функцию с N входами и 1 выходом, где N – суммарная разрядность сравниваемых данных. Результатом генерации операции сравнения является компаратор с соответствующей функцией.

Оператор ?

Данный оператор в языке выдает номера позиций в списке аргумента, где содержатся истинные значения. Так как в СБИС статическая разрядность данных, то размерность списка результата не изменяется. Для позиций, где в списке аргумента значения не истина, в списке результата подставляется значение 0.

Данный оператор реализуется в виде N параллельных сравнений (компараторов) и присвоения результата соответствующему регистру списка результата размерности N.

Пример:

(x1,x2,x3):?

Для x1 = true, x2 = false, x3 = true результатом будет список данных (1,0,3). На рисунке 19 представлена схема реализации оператора ?.

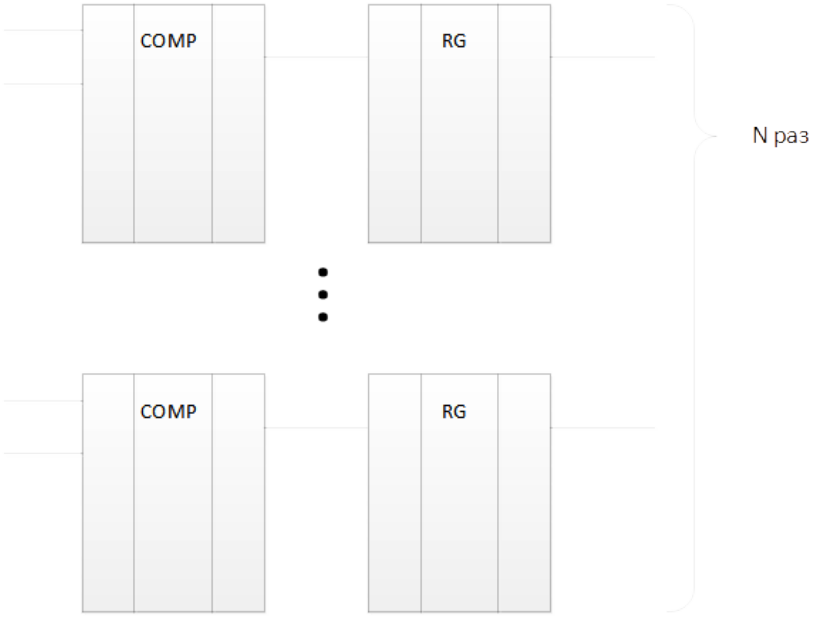


Рисунок 19 – Реализация оператора ?

Оператор dup

Данный оператор реализует преобразование входного значения A в список длины N путем повторения значения A N раз. Генерируемый элемент – набор регистров и синхронное присвоение каждому регистру первого элемента (A) аргумента. Ограничение реализации данного оператора на платформе СБИС – значение количества повторений N или размерность повторения должна быть известна или вычислена на этапе синтеза.

Оператор #

Данный оператор в языке отвечает за транспонирование списков. Так как размерности списков известны на этапе трансляции, данный оператор реализуется

путем присвоения выходным регистрам данного узла значений входных регистров с соответствующим изменением порядка входных значений.

Целое/булевское значение в качестве функции

Данный оператор в языке реализует выбор одного значения из списка. В СБИС для реализации данного оператора генерируется мультиплексор. Количество входов данных мультиплексора определяется как $\log_2 N$, где N – разрядность значения, используемого в качестве функции.

Вершина списка данных/оператор ()

Данный оператор в языке реализует формирование списка данных из атомарных значений. Так как атомарные значения формируются выше по графу функции и уже присвоены регистрам результата, то данный оператор реализуется в виде асинхронных присвоений. Этот оператор/вершина не занимает такт конвейера, длительность операции – 0 тактов.

Вызов внешней функции

Вызов внешней функции заключается в загрузке соответствующего информационного графа данной функции с передачей формата(типа) аргумента данной функции и проведение синтеза HDL графа данной функции. В результате возвращается формат результата, который используется для формирования регистров результата данной вершины интерпретации в вызывающей функции. Помимо этого, в таблицу внешних ссылок для вызывающей (верхней) функции добавляется ссылка на сгенерированный HDL граф вызываемой функции, которая потом используется для генерации HDL кода данной функции.

3.3.8 Синтез описания на HDL языке

Входными данными для синтеза кода на HDL являются HDL граф и соответствующий управляющий граф. Процесс синтеза состоит из следующих этапов:

1) генерация описания заголовочной части модуля;
2) генерация таблицы регистров модуля;
3) генерация кода модуля для информационного графа;

4) генерация кода для управления вычислениями.

Для генерации описания заголовочной части исходными данными служит таблица входных и выходных портов, полученная на этапе создания HDL графа (п. 3.3.6). Для каждого порта из таблицы генерируется описание в соответствии с выбранным HDL языком.

Для генерации таблицы регистров модуля используется таблица регистров модуля из HDL графа, а также из управляющего графа. В управляющем графе таблица регистров включает в себя регистры для сигналов управления стадиями конвейера информационного графа.

Для синтеза кода информационного графа на HDL языке осуществляется обход HDL графа. Для каждой вершины генерируются соответствующие конструкции HDL языка, описанные в п. 3.3.7 для соответствующих операций. В качестве входных данных для операции вершины графа используются регистры узлов графа (через ссылки на таблицу имен регистров), являющиеся выходными в узлах предшественниках. Результат операции записывается в регистры результата текущей вершины. Каждый уровень в HDL графе после приведения его к ярусно-параллельной форме представляет собой стадию конвейера, и каждая стадия конвейера запускается от соответствующего управляющего сигнала. Код каждой стадии конвейера заключается в блок, срабатывающий только при наличии сигнала управления:

```
if (cg1) begin
  код стадии конвейера
end
```

Схема управления представляет собой конвейерную схему, с количеством стадий равным количеству стадий в конвейере HDL графа. Результатом каждой стадии конвейера управляющего графа является сигнал, вызывающий срабатывание соответствующей стадии информационного. Внутри стадии расположен автомат, формирующий сигнал управления. Количество автоматов обработки сигналов управления соответствует количеству типов вершин информационного графа:

1) автомат аргумента;

2) автомат вершины обработки данных;

3) автомат списка данных;

4) автомат параллельного списка;

5) автомат результата.

Автомат результата представляет собой логическое “И” всех входных портов готовности данных модуля и задержки на 1 такт. В случае атомарного аргумента автомат вырождается в задержку входного сигнала готовности данных.

Автомат вершины обработки данных представляет собой автомат на три состояния:

- ожидание сигнала аргумента;

- ожидание сигнала функции;

- выдача сигнала управления.

Автомат реализуется как два регистра для хранения сигналов готовности аргумента и функции и логическое “И” формирования сигнала управления. Пример на языке Verilog:

```
assign cg3 = cg_func & cg_arg;
always @(posedge clk) begin
    cg_func <= cg1;
    cg_arg <= cg2;
    cg4 <= cg3;
end
```

здесь cg1, cg2 – сигналы готовности с предыдущих стадий, функции и аргумента соответственно, cg3 – сигнал разрешения работы текущей стадии и cg4 – сигнал готовности для следующей стадии.

Автомат списка данных представляет собой счетчик по количеству элементов списка данных. По достижении счетчиком максимума выдается управляющий сигнал готовности списка данных.

Автомат параллельного списка в реализации управляющего графа на HDL отсутствует, так как параллельные списки известной размерности на этапе

оптимизации и синтеза HDL графа раскрываются во множество отдельных вершин.

3.3.9 Синтез регистрово-вентильного представления СБИС

Данный этап выполняется с использованием стандартных средств разработки от производителя конкретной аппаратной платформы. Это могут быть синтезаторы RTL, входящие в состав среды разработки для ПЛИС (Xilinx XST, Altera IDE и пр.) либо других платформ (ASIC, БМК). Для синтеза используется HDL описание, полученное на предыдущем шаге.

3.4 Выводы по главе

1. Разработанный на основе ФПП подхода метод высокоуровневого синтеза СБИС позволяет разделить разработку на 2 уровня: архитектурно-независимый и архитектурно-зависимый. Архитектурно-независимый уровень заканчивается разработкой отлаженного алгоритма на ФПП языке и преобразованием ФПП программы в промежуточное представление.

2. Разработанный метод при изменении архитектурно-специфичных данных, таких как ресурсных ограничений, параллелизма реализации, типов и размерностей данных не требует возврата на архитектурно-независимый уровень. Исходный алгоритм на ФПП языке, в отличие от существующих методов высокоуровневого синтеза, остается неизменным.

3. Разработаны алгоритмы синтеза HDL описания СБИС из промежуточного представления ФПП программы.

4. Разработаны алгоритмы синтеза конструкций ФПП языка в схемы СБИС.

5. На основе алгоритмов синтеза разработан синтезатор с ФПП языка в HDL описание схемы СБИС. Синтезатор позволяет преобразовывать неизменное промежуточное представление алгоритма на ФПП языке в различные варианты схем СБИС с использованием различных размерностей и типов данных, различным уровнем параллелизма схемы.

4 Практические результаты и анализ архитектурных решений

4.1 Применение инструментальных средств для разработки СБИС

Первоначальное описание алгоритма, реализуемого на кристалле, осуществляется средствами интегрированной среды проектирования с использованием языка функционально-потокового параллельного программирования Пифагор. Транслятор с данного языка [44] обеспечивает преобразование разработанных функций, образующих ФПП программу, в набор реверсивных информационных графов (ИГ), описывающих информационные зависимости между выполняемыми операциями. Дополнительная утилита, используя ИГ в качестве исходных данных, формирует управляющий граф (УГ), который задает порядок выполнения вершин ИГ. При этом возможен выбор стратегии выполнения от последовательной до максимально-параллельной. Выбор стратегии управления вычислениями определяется разработчиком. Полученные ИГ и УГ вместе образуют программу, которая может быть выполнена на интерпретаторе [80], это позволяет осуществить проверку ее корректности до перехода к процессу синтеза RTL-представления.

Рассмотрим применение метода синтеза на примере сложно-функционального блока (IP core). При выполнении работ в рамках работ по ФЦП гос. контракт № 14.578.21.0116 “ Разработка архитектуры СБИС класса Система на кристалле для создания угломерного навигационного приемника” для разработки сложно-функциональных блоков использовался предложенный метод синтеза. В качестве примера рассмотрим блок дискретного преобразования Фурье из состава навигационного приемника. В рамках проекта в функции данного блока входило вычисление преобразования Фурье и вычисление средней амплитуды спектра. Необходимо было реализовать проект первоначально в базисе ПЛИС с дальнейшим переходом в базис базового матричного кристалла (БМК). При этом в базисе ПЛИС и БМК было доступно существенно разное

количество ресурсов и требовались 2 реализации одного алгоритма с различной степенью параллелизма и ресурсных ограничений.

В базисе ПЛИС, исходя из ресурсных ограничений и тактовой частоты, схема реализовывалась полностью параллельно, в базисе БМК ограничения по ресурсам позволяли реализовать вычисление каждого значения “бабочки” БПФ и сложение элементов массива последовательно.

На рисунке 20 представлена программа вычисления средней амплитуды спектра на ФПП языке.

```
MeanAbs << funcdef Arg {
  X <<  (Arg :[(1, Len , 1 ):..:[]] : ComplexNumsAbs);  // параллельно вычисляем модули
  (X : VecSum, X : |) :/ >> return;  // находим сумму модулей и делим на длину массива
}
ComplexNumsAbs << funcdef params {
  a << params:1;
  b << params:2;
  return << ((a,a):* , (b,b):* ):+:Sqrt ;
}
VecSum << funcdef Param {     //сложение элементов массива
  Len << Param:|;                    //длина входного списка данных
 variant<< ( (Len,2):<, (Len,2):=, (Len,2):> ):?; //длина меньше, равна или больше 2
  x1 <<  ([], . , . ):variant;
  x2 << ( . ,+, . ):variant;
  x3 << ( . , . ,Next):variant;

  (
    Param:x1,  //меньше 2, возвращаем аргумент
    Param:x2,  //равна 2, возвращаем сумму двух элементов списка
    Param:x3 //больше двух, передаем в функцию Alpha
  ):variant   //фильтрация (выбор) результата
  >>return
}
 Next << funcdef Param {
    Len    << Param:|;
    OddVec << Param:[(1,Len,2):..:[]]:(.); //выбираем четные элементы массива
    EvenVec<< Param:[(2,Len,2):..:[]]:(.); //нечетные

   (OddVec:VecSum, EvenVec:VecSum ):+ >> return//параллельное сложение в VecSum
}
```

Рисунок 20 – Код функции вычисления средней амплитуды спектра

На рисунке 21 представлен код функции вычисления БПФ.

```
fft<< funcdef Arg {
  // Arg =( (W0, W1 … W[]), (x[1],x[2],x[3],...,x[2^N]))
  X<< Arg : 2;
  Wn << Arg : 1;
  Len << X:|;
  LenW << Wn :| ;
  Half << (len,2):%; // длина половины вектора
    Odd  << X:[(1,Len,2):..:[]]:(.); //выбираем четные элементы массива отсчетов
    Even << X:[(2,Len,2):..:[]]:(.); //нечетные
    WnOdd<<Wn:[(1,LenW,2):..:[]]:(.);//выбираем четные элементы массива коэф. поворота
    WnEven << Wn:[(2,Len,W2):..:[]]:(.); //нечетные
    variant<< ( (Len,2):>, (Len,2):= ):?; //длина больше  или равна 2
    select <<  (. , . ):variant;
    Next << (.,fft):variant;
 (
    [ (WnOdd, Odd), (WnEven, Even) ] : Next, // разделение на N/2
    (Wn, (Odd, Even)) : select // 2 элемента
  ) :  variant : BF >> return
}
}
BF << funcdef Param {
    Wn << Param:1;
    M << (Wn, Param : 2 : 2): ComplexNumsMult;
    ( (Param : 2 : 1, M) : ComplexNumsSum,
      (Param : 2 : 1, M):ComplexNumsSub ) >> return;
}
ComplexNumsMult << funcdef params {
  a << params:1:1;
  b << params:1:2;
  c << params:2:1;
  d << params:2:2;
  return << ( ((a,c):* , (b,d):*):- , ((a,d):* , (b,c):*):+      );
}
ComplexNumsSum << funcdef params {
  a << params:1:1;
  b << params:1:2;
  c << params:2:1;
  d << params:2:2;
  return << ( (a,c):+ , (b,d):+  );
}
ComplexNumsSub << funcdef params {
  a << params:1:1;
  b << params:1:2;
  c << params:2:1;
  d << params:2:2;
  return << ( (a,c):- , (b,d):-  );
}
```

Рисунок 21 – Код функции вычисления БПФ

Функциями верхнего уровня являются функция вычисления БПФ – fft, и вычисление средней амплитуды спектра MeanAbs.

На первом этапе описание алгоритма транслируется в промежуточное представление. Используя промежуточное представление и отладчик исходный код на ФПП языке отлаживается и тестируется.

При переходе к этапу синтеза необходимо задать размерности и типы данных. В данном проекте размерность преобразования БПФ составляла 512 точек, разрядность одного отсчета 16 бит. Для функции fft тип данных определяется так:

(arg.datalist.512.int.16, arg.datalist.512.int.16).

Первый массив содержит константы коэффициентов поворота, второй - отсчеты сигнала.

Для функции вычисления среднего значения амплитуды спектра формат аргумента, следующий:

Arg.datalist.512.int.32.

На этапе оценки границ изменения параллелизма для функции fft происходит разворачивание рекурсии в конвейерную схему. При данной размерности глубина дерева рекурсии составит 9 стадий. На первой стадии вычисляются значения элементарных “бабочек” из пар значений, далее 4,8 .. 512. Для каждой “бабочки” необходимо выполнить 1 комплексное умножение и 2 сложения/вычитания. Комплексное умножение в максимально-параллельной форме занимает 2 стадии конвейера – 4 умножения на первой стадии и сложение/вычитание на второй. Комплексные вычитания/сложения занимают 1 такт. Итоговый максимально-параллельный вариант “бабочки” занимает 3 стадии конвейера для данного варианта. Итоговая минимальная длина конвейера составит 9*3 = 27 тактов.

Последовательная форма конвейера вычисления “бабочки” займет 10 тактов - 4 умножения/2 сложения для комплексного умножения, по 2 сложения/вычитания для вычисления комплексной суммы и разности. Итоговая длина конвейера для последовательной формы составит $N/2*\log_2 N$ вычислений

“бабочки” или 23040 тактов. Между этими крайними вариантами можно генерировать схемы без изменения исходного описания на ФПП языке.

С использованием данного метода были реализованы некоторые сложно-функциональные блоки из состава навигационного приемника [97, 102, 103].

4.2 Критерии сравнения

Для сравнения разработанного метода высокоуровневого синтеза с существующими методами предложена методика сравнения, реализован набор тестовых задач для всех сравниваемых методов и получена оценка результатов проектирования. Согласно доступным источникам в настоящее время не существует единой (стандартизированной) методики оценки сравнительных результатов HLS [86, 87], за исключением метода, предложенного в [85].

Для сравнения методов синтеза выделены следующие классы исследуемых параметров, применяемых при идентичных условиях тестирования:

- метрики получаемого результата разработки (быстродействие, объем, энергопотребление и т.д.);
- метрики процесса разработки (трудоемкость разработки/тестирования, время/затраты на перенос проекта с платформы на платформу).

Рассмотрим каждую группу метрик подробнее. Метрики, оценивающие полученный результат наиболее просты, но в тоже время требуют точного определения для исключения ошибок при сравнении. Выделены две группы метрик: быстродействие и занимаемый ресурс целевой платформы.

Быстродействие схемы определяет количественную меру результата, выдаваемого СБИС в единицу времени. К метрикам, определяющим быстродействие схемы, относят:

- максимальную тактовую частоту схемы;
- задержку схемы в тактах до выдачи результата (латентность).

По отдельности сравнение этих метрик у разных схем некорректно, так как при различных алгоритмах синтеза у конвейерных схем для одного и того же алгоритма могут быть получены результаты с более длинным конвейером

(большей латентностью в тактах) и большей тактовой частотой и наоборот. Поэтому для корректного сравнения предложено ввести композитную временную метрику от появления данных на входе в схему до выдачи результата, рассчитываемую по формуле:

$$Tw = Cl/Fmax$$

где Fmax – максимальная тактовая частота схемы,

Cl – задержка схемы в тактах (длина конвейера).

Не менее важным аспектом, влияющим на точность результата сравнения метрики по быстродействию, является аппаратная платформа. На различных платформах быстродействие схемы может отличатся. Для корректного сравнения необходима реализация различных методов синтеза на одной и той же платформе. Определено, что ввиду высокой трудоемкости, временных и финансовых затрат разработки СБИС сравнение методов синтеза целесообразно проводить на платформе ПЛИС. В рамках предложенного метода сравнения выбрана платформа ПЛИС от компании Xilinx. Для исключения влияния разницы в версиях кристаллов, для всех методов применялась одинаковая версия ПО и один конечный вариант кристалла ПЛИС. Результатом для сравнения методов являлось RTL описание на языке HDL, которое посредством САПР Xilinx Vivado доводилось до этапа топологии. Настройки САПР были идентичными для всех проектов участвовавших в тестировании. Это позволило исключить влияние этапа прототипирования на ПЛИС на результат.

Для определения максимальной тактовой частоты схемы использовался результат оценки частоты схемы, получаемой после этапа размещения и трассировки кристалла (Place and route) в Xilinx ISE/Vivado [91]. Для оценки задержки схемы использовалась симуляция RTL кода с оценкой задержки схемы в тактах. Для симуляции использовалось ПО Mentor Graphics ModelSim [92, 93].

Оценка метрики занимаемого ресурса также производилась в ресурсе ПЛИС, занимаемым прототипом схемы. Использовались следующие основные группы ресурсов:

1) логические ячейки;

2) триггеры;

3) встроенные специализированные блоки:

a) блочная память;

b) специализированные арифметические блоки (умножители DSP);

c) блоки ввода/вывода.

Определено, что при использовании одинаковой платформы и одной версии САПР для разных методов синтеза, прямое сравнение занимаемого ресурса по различным категориям может дать ошибку. В зависимости от степени оптимизации и использования специализированных ресурсов ПЛИС различными методами получаемые схемы могут быть реализованы путем перехода, например, от реализации умножителя на логических ячейках к использованию специализированного блока DSP. Для исключения данной ошибки сравнивали общую площадь схемы, занимаемую на кристалле.

Для оценки энергопотребления использовались численные значения из отчета Xilinx Vivado, получаемые после этапа топологической привязки к кристаллу. На данном этапе так же было выявлено наличие отклонений, поскольку результат синтеза различных схем посредством одной версии ПО и для одинакового кристалла ПЛИС может отличатся, за счет реализации одних и тех же участков схемы с использованием разных ресурсов с различным энергопотреблением (например логических ячеек вместо DSP и т.д.).

Для исключения данной ошибки в расчет принимались только идентичные результаты синтеза схем.

Помимо этого, также оценивался получаемый на новой платформе результат по метрикам быстродействия и занимаемого ресурса. Сохранение отношений между метриками быстродействия для различных методов синтеза после переноса проекта на новую платформу показывает хорошую переносимость проекта.

Метод сравнение переноса проекта на другую платформу производился путем переноса проекта и прототипирования на платформе ПЛИС фирмы Altera.

Как и в предыдущем случае, на платформе ПЛИС Altera также использовалась одинаковая версия ПО САПР и один целевой кристалл.

Сравнительная оценка трудоемкости процесса разработки проекта для различных методов синтеза представляет собой более сложную задачу. Прямое сравнение, например, по количеству строк кода, необходимому для описания проекта различными методами, некорректно, так как не учитывает различные языковые конструкции в разных методах, разное время разработки и часть алгоритма, реализуемого одной строкой кода в различных методах. Кроме этого оценка сроков и трудоемкости разработки всегда носит вероятностный характер. Поскольку данная оценка является вероятностным утверждением, означает, что для нее существует некоторое распределение вероятности, которое может быть как с высокой неопределенностью (менее точная оценка) так и с низкой (более точная оценка).

В разработке ПО для оценки трудоемкости применяется также метод функциональных метрик, который не зависит от используемого языка программирования. В основе метода лежит разбиение разрабатываемого ПО на некоторые функциональные элементарные процессы с присвоением им весовых коэффициентов в зависимости от сложности. Трудоемкость всего проекта оценивается как сумма всех элементарных метрик с учетом весового коэффициента. Для области СБИС такой метод оценки трудоемкости можно применить, если выразить схему в элементарных функциональных блоках, таких как память, мультиплексоры/демультиплексоры, арифметические блоки, автоматы состояний (FSM) и т.д.

4.3 Методы синтеза для сравнения

На основе анализа наиболее часто используемых в промышленной разработке СБИС (ПЛИС) методов разработки были выбраны следующие методы синтеза:

1) разработка проекта по традиционному маршруту проектирования с использованием описания проекта на поведенческом уровне на языке HDL;

2) высокоуровневый синтез на базе описания на С-подобном языке;

3) высокоуровневый синтез в среде MATLAB с последующим автоматическим синтезом проекта в HDL описание.

Традиционный маршрут проектирования СБИС, хотя и не относится к высокоуровневым методам проектирования, включен в сравнение как исходная точка для оценки производительности и ресурса, занимаемого схемой. В большинстве случаев данный метод показывает наибольшую производительность/наименьший ресурс при наибольших затратах на реализацию. По данному методу реализация всех тестовых задач велась на языке Verilog. Этот же HDL язык был выбран во всех остальных методах как итоговый для прототипирования результата на платформе СБИС. Такой подход позволил исключить возможные различия, вносимые разными языковыми конструкциями при представлении результата на разных HDL языках.

Высокоуровневый синтез на С-подобном языке включен в сравнительный обзор как стандарт де-факто в современных промышленных высокоуровневых методах разработки. Так как в качестве платформы прототипирования использовались ПЛИС фирмы Xilinx, то для сравнения был выбран высокоуровневый метод синтеза HLS [15].

Еще одним методом высокоуровневого синтеза был выбран метод разработки описания СБИС, поддерживаемый средой MATLAB [94]. Данный метод основан на описании алгоритма функционирования схемы СБИС на встроенном в среду m-языке с последующим полуавтоматическим преобразованием в описание на HDL языке. Вторым вариантом данного подхода является описание схемы в графическом виде в среде Simulink/MATLAB [95] и последующим синтезом в HDL. Разница в этих подходах состоит только в способе исходного описания схемы.

4.4 Тестовые задачи

Для тестовых задач, используемых при сравнении методов синтеза, можно выделить два класса алгоритмов: алгоритмы обработки данных и управления.

Также, как и в [85] типы задач взяты из пакета CHStonex. Список задач приведен в таблице 6.

Таблица 6 – Список тестовых задач

Номер п.п.	**Название**
1	Квадратный корень (модуль комплексного числа)
2	Умножение матриц
3	Сумма элементов массива (последовательно)
4	Сумма элементов массива (параллельно)
5	FIR фильтр №1
6	FIR фильтр №2

Фильтр №1 и №2 отличаются структурой конвейера. Фильтр №1 разработан с минимальной длиной конвейера (максимальным параллелизмом) и максимальным занимаемым ресурсом, фильтр №2 – с максимальной задержкой и минимальным ресурсом (последовательно). Различные варианты архитектуры для решения одной и той же задачи позволяют оценить затраты на изменение степени параллелизма при реализации задачи разными методами. Задачи №1,3 и 4 взяты из примеров компилятора высокоуровневого синтеза LegUp [85] и соответствуют тестовым примерам sra и loop. Размерности данных во всех примерах составляли 16 бит (если не указано иное). Размерности фильтров в примерах №5 и 6 составляли 16 коэффициентов. Размерность массива в примерах 3 и 4 аналогично с [85] составляла 32 элемента.

Для сравнения результаты были приведены к относительному масштабу, где для производительности за 100% принимался самый быстрый вариант, а для сравнения ресурсов за 100% принимался результат с наименьшим количеством занимаемого ресурса по категориям логических ячеек, триггеров и умножителей.

4.5 Результаты

Результаты реализации тестовых задач по метрикам группы производительности приведены в таблице 7.

Таблица 7 – Результаты реализации тестовых задач (производительность)

Задача	**Метод**	**Частота, МГц**	**Задержка схемы, тактов**	**Tw**	**LUT, шт**	**Триггеры, шт**	**DSP**
Модуль комплексного числа	MATLAB HDL Coder	152	1	0.0065	60	48	0
	HLS	124	1	0.008	496	67	0
	xHDL	377	2	0.0053	33	30	0
	ФПП	593	5	0.0084	37	108	0
Умножение матриц 4x4	MATLAB HDL Coder, параллельно	163	1	0.0061	528	10	64
	MATLAB HDL Coder, последовательно	250	16	0.064	1529	8096	4
	HLS, параллельная	155.4	2	0.0128	820	1173	56
	HLS, последовательно	269	20	0.0743	840	613	4
	xHDL, параллельно	172.4	1	0.0058	10	10	72
	xHDL, последовательно	562	19	0.0338	376	551	4
	ФПП, параллельно	704	4	0.00568	832	880	64
	ФПП, последовательно	691	25	0.0361		352	4

Окончание таблицы 7

Задача	**Метод**	**Частота, МГц**	**Задержка схемы, тактов**	**Tw**	**LUT, шт**	**Триггеры, шт**	**DSP**
Сумма элементов массива (параллельно)	MATLAB HDL Coder	187.16	1	0.005343	533	0	0
	HLS	101.5	1	0.00985	715	2	0
	xHDL	200	1	0.005	496	0	0
	ФПП	470	4	0.0052	499	485	0
Сумма элементов массива (последовательно)	MATLAB HDL Coder	380	31	0.0815	221	559	0
	HLS	714	32	0.0448	659	353	0
	xHDL	424.2	31	0.073	191	28	0
	ФПП	710	32	0.044	409	50	0
FIR фильтр (последовательно)	MATLAB HDL Coder	174	9	0.0517	200	323	1
	HLS	360	16	0.0444	720	1286	2
	xHDL	710	21	0.026	36	68	1
	ФПП	640	20	0.0305	48	259	1
FIR фильтр (параллельно)	MATLAB HDL Coder	87	2	0.023	148	272	8
	HLS	119.5	2	0.0167	957	900	28
	xHDL	710	7	0.00865	78	69	8
	ФПП	543	7	0.011	468	439	16

На рисунке 22 приведена сравнительная диаграмма производительности методов синтеза (числовые значения в %). На диаграмме видно, что реализации в MATLAB показывают сравнимую с реализацией на HDL производительность на 3 задачах из 7, метод HLS на 1 задаче из 7, ФПП метод – на 6 из 7 задач. При этом остальные задачи показывают производительность в ~1,5-2 раза меньше, чем реализация на HDL. Суммарно по производительности на всём наборе тестовых задач методы распределились в следующем порядке (в порядке убывания):

1) HDL;
2) ФПП;

3) MATLAB;

4) HLS.

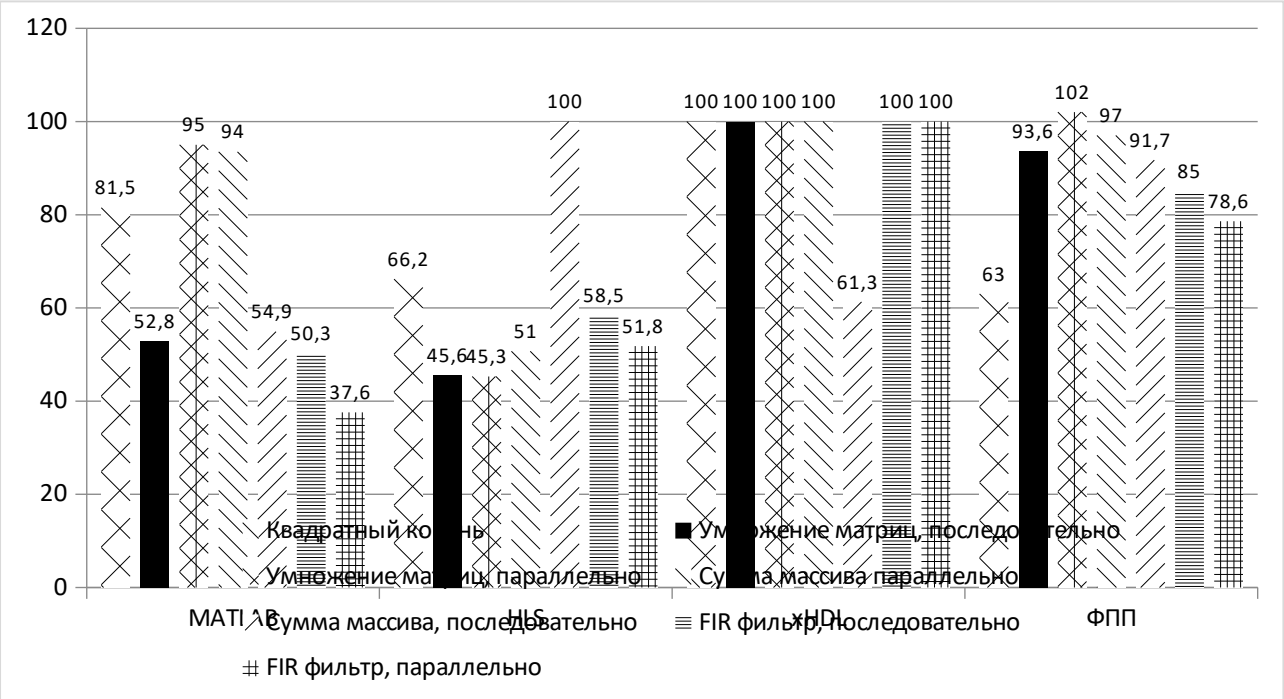


Рисунок 22 – Производительность методов на тестовых задачах

На рисунках 23, 24 и 25 приведены диаграммы занимаемого ресурса для разных методов по логическим ячейкам, триггерам и умножителям (DSP).

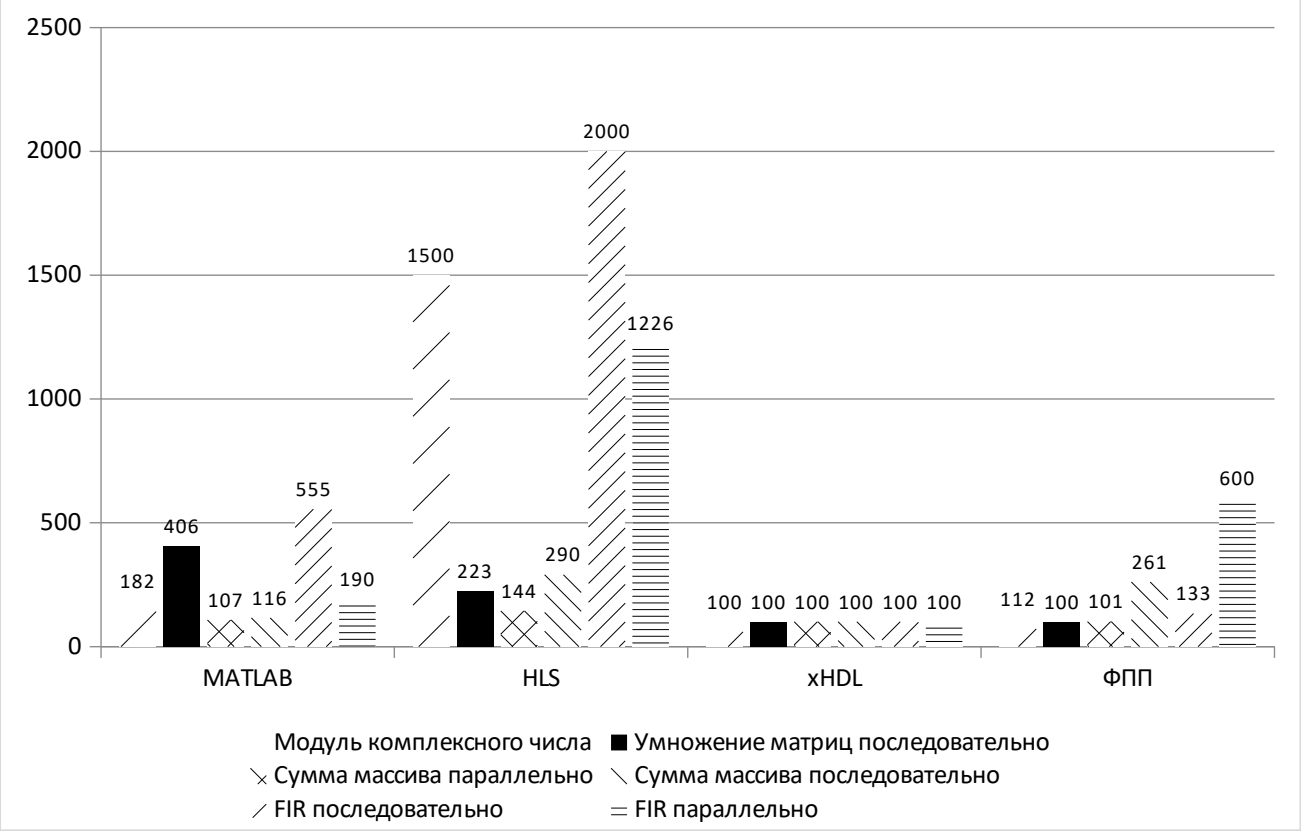


Рисунок 23 – Ресурс логических ячеек

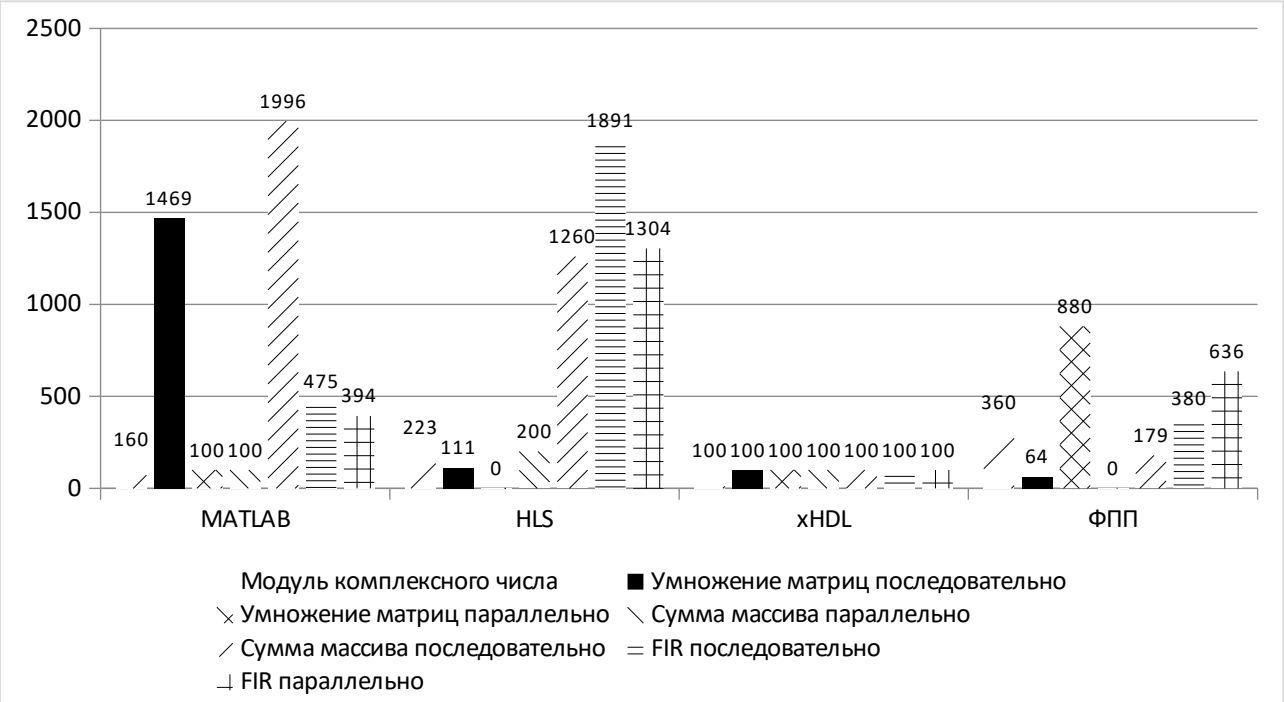


Рисунок 24 – Ресурс триггеров

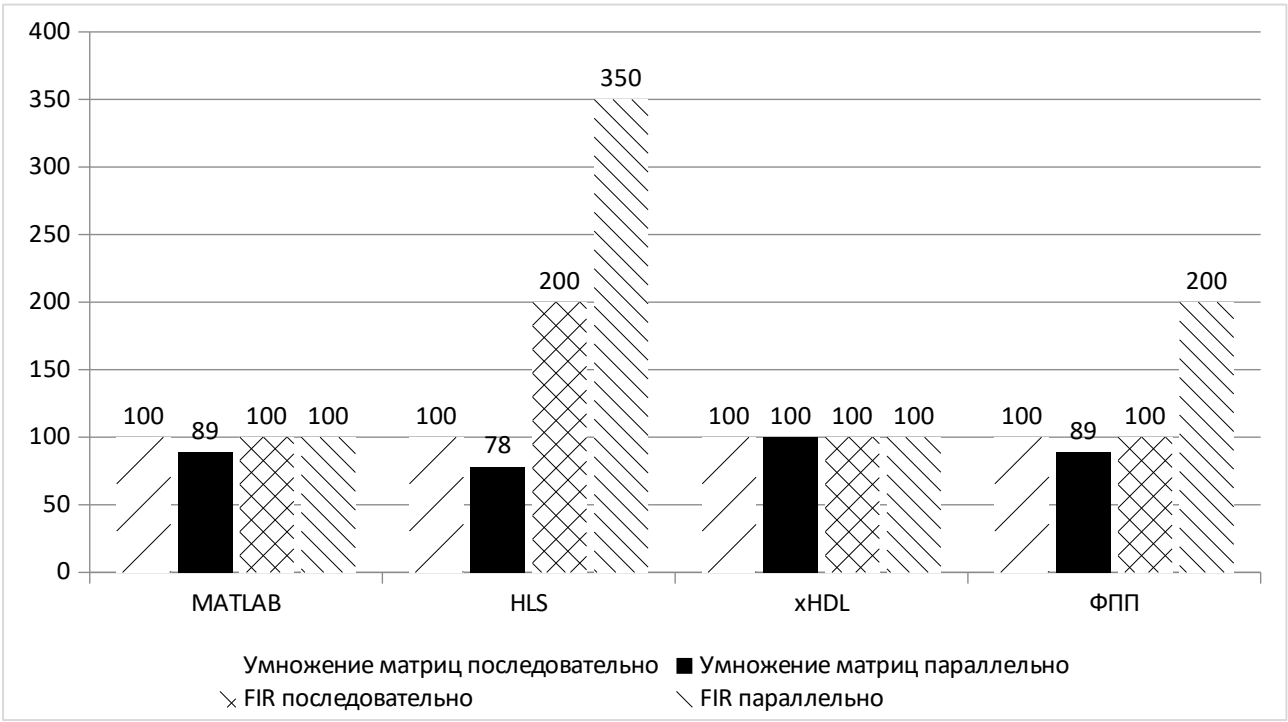


Рисунок 25 – Ресурс DSP

На диаграмме ресурса триггеров для сохранения читаемости диаграммы для двух задач (умножение матриц параллельно для HLS и сумма массива параллельно для ФПП) не приведен занимаемый ресурс, так как он многократно

превышает остальные варианты. На диаграмме ресурса DSP приведены не все задачи, так, как только часть задач использует умножители.

Из диаграмм видно, что полученные методом HLS решения занимают самый большой ресурс, многократно превышающий остальные (в 3 задачах по ресурсу логических ячеек, в 4 задачах по ресурсу триггеров и 1 задаче по умножителям). На данном этапе тестирования это связано с неэффективностью распараллеливания изначально последовательного описания алгоритма.

Реализации тестовых задач в среде MATLAB по занимаемому ресурсу логических ячеек и триггеров в целом отличается от HDL реализации в 2-4 раза. Исключение составили три задачи с последовательной реализацией (сумма массива, умножение матриц и FIR фильтр). Данные варианты были получены из общего варианта описания алгоритма путем изменения настроек HDL кодера MATLAB, таких Pipelining, Oversampling и Resource Sharing, до получения одинакового с другими реализациями использования ресурса DSP. Неоптимальное использование ресурса в последовательных вариантах показывается неэффективность преобразования параллелизма в MATLAB.

ФПП метод показывает сравнимое использование ресурса, при сравнимой с HDL реализациями производительностью. Исключение составили задачи умножение и сложение матриц параллельно по ресурсу триггеров. Анализ этих двух вариантов показал, что избыточное использование ресурса триггеров возникает из-за большего количества стадий конвейера в этих задачах (4 стадии в ФПП против 1 в HDL/MATLAB). Большее количество стадий конвейера возникает из-за изначального описания с максимальным параллелизмом на уровне операций. Например, в случае максимально-параллельного варианта при умножении матриц размерностью 4x4 вычисление одного элемента требует 4 умножения и 3 сложения. В варианте MATLAB все 7 операций выполняются за один такт, в варианте ФПП – за 3: умножения, сложения 2 пар и сложение результатов предыдущей стадии. В результате у ФПП варианта при большей тактовой частоте и большем количестве тактов метрика производительности сравнима с вариантами HDL/MATLAB, но ресурс триггеров используется больше.

Эти примеры показывают, что в данном случае эффективнее объединить стадии конвейера без потери производительности и выигрышем по ресурсу. Для оценки оптимальности по производительности реализации в HDL синтезатору с ФПП в HDL необходимо получение метрик производительности. Данная оптимизация может быть добавлена в синтезатор с ФПП на HDL при условии реализации передачи метрики производительности после синтеза HDL обратно в ФПП этап синтеза. Тогда на этапе синтеза с ФПП описания в HDL можно реализовать оптимизацию по данным критериям производительности.

С использование разработанного комплекта ПО производилась разработка 3 комплектов СБИС в рамках контрактов 14.578.21.0116, 02.G25.31.0202 СБИС угломерного навигационного приемника и 14.578.21.0021 СБИС бортового комплекса управления для малых космических аппаратов.

В рамках первых двух контрактов разрабатывалась СБИС цифровой обработки сигналов для первичной обработки сигналов угломерного навигационного приемника. СБИС первичной обработки сигналов угломерного навигационного приемника реализовывалась для 2 платформ: ПЛИС и БМК Алмаз-13. Исходное описание алгоритма было выполнено на ФПП языке с использованием предложенного высокоуровневого метода синтеза. С помощью разработанного ПО из одного исходного описания был произведен синтез 2 вариантов схемы под разные платформы: ПЛИС и БМК. Для платформы ПЛИС существовал используемый для отладки вариант описания СБИС приемника, выполненный ручным описанием на языке HDL, с которым производилось сравнение эффективности разработки с использованием разработанного метода. Результаты показали сравнимую производительность и ресурс СБИС (разница ~20%) при двукратной экономии времени за счет использования одного описания для обоих платформ ПЛИС и БМК.

В рамках проекта СБИС бортового комплекса управления для малых космических аппаратов разрабатывалось описание СБИС для 2 платформ: отладочной на базе ПЛИС и радиационно-стойкой СБИС. В результате оба

варианта СБИС были реализованы из одного высокоуровневого описания с уменьшением времени разработки в 2 раза.

4.6 Выводы по главе

1. Предлагаемый метод синтеза на основе ФПП описания показывает сравнимую с реализацией на HDL производительность, превосходя при этом наиболее распространенный метод HLS по эффективности использования ресурса.

2. Получение различных вариантов по степени параллелизма из одного описания в ФПП методе не сказывается на эффективности реализации по ресурсам.

3. Изменение степени параллелизма в методах на основе MATLAB и HLS в большинстве случаев приводит к резкому увеличению используемого ресурса.

4. Реализация вариантов задач “с нуля” с целью оптимизации в случае MATLAB/HLS приводит к большей эффективности, чем изменение степени параллелизма у исходного описания – т.е. переписывание исходного алгоритма для иного варианта параллелизма более эффективно в случае HLS/MATLAB.

5. Для дальнейшей оптимизации процесса синтеза из ФПП в HDL необходимо обеспечить передачу метрик производительности полученных архитектурных вариантов из HDL синтеза обратно в ФПП этап.

Заключение

В соответствии с паспортом специальности, результаты диссертационных исследований включают развитие теории параллельного программирования, создания и сопровождения программных средств поддержки проектирования цифровых интегральных схем и систем.

Областью диссертационных исследований являются модели, методы и алгоритмы проектирования и анализа параллельных программ и программных систем, их эквивалентных преобразований, верификации и тестирования.

Предложена модификация языка параллельного программирования, разработана программная система программирования, предложена семантика программ для описания алгоритмов функционирования СБИС.

Разработаны модели, методы, алгоритмы, модифицирован язык и реализованы программные инструменты для организации взаимодействия комплекта прикладных программ поддержки проектирования.

Предложены модели и методы создания программ на языке параллельного программирования «Пифагор» для параллельной обработки данных на базе однокристальных систем, разработаны инструментальные средства параллельного программирования.

Основным научным результатом диссертационных исследований является метод архитектурно-независимого высокоуровневого синтеза цифровых однокристальных систем, базирующийся на функционально-потоковой парадигме параллельных вычислений, позволяющий обеспечить переносимость и архитектурную независимость полученных решений, а также обеспечить максимально возможный охват вариантов решения задач с учетом требуемых спецификаций проекта.

Получены следующие результаты исследования:

1. На основе анализа моделей вычисления и языков программирования предложен метод синтеза функционально-потокового описания исходного

алгоритма с массовым параллелизмом. Разработаны изменения, вносимые в ФПП модель, с целью её адаптации для разработки СБИС, что позволяет обеспечить переносимость и архитектурную независимость проекта.

2. Разработаны алгоритмы изменения параллелизма исходного ФПП описания при переходе к различным вариантам реализации схемы на СБИС. Предложенные алгоритмы позволяют реализовывать схемы от полностью последовательного варианта до максимально-параллельного.

3. Предложены алгоритмы преобразования (синтеза) языковых конструкций и промежуточного представления ФПП языка в описание схемы СБИС на языках описания аппаратуры.

4. На основе разработанных алгоритмов синтеза создано ПО синтезатора с ФПП языка на языки HDL.

5. Предложена методика сравнительного анализа методов синтеза, определены метрики тестирования и получены результаты решения практических задач. Результаты сравнительного анализа показали, что предлагаемый метод синтеза обеспечивает архитектурную независимость, переносимость инвариантность решения задач и при этом показывает сравнимую с реализацией на HDL языках производительность конечных решений. При этом, получение различных вариантов по степени параллелизма из одного описания алгоритма на ФПП языке не сказывается на эффективности реализации.

6. Получены практические результаты реализации сложно-функциональных блоков и кристаллов СБИС на основе разработанного метода:

a) сложно-функциональный блок драйвера бортовой сети космического аппарата [98] из состава СБИС БА и СБИС КПА КА производства АО ИСС им. ак. М.Ф. Решетнева г. Железногорск;
b) сложно-функциональные блоки из состава СБИС спутникового модема ЯР-1040, пр-ва АО КБ «Искра» г. Красноярск [96,97,102,103];
c) СБИС К5540ТН014А на базе БМК «Алмаз-13» пр-ва ОА «НИИМЭ и Микрон» г. Зеленоград для навигационного оборудования пр-ва ФГУП «Радиосвязь» г. Красноярск.

Предложенные алгоритмы и программные решения создают основу для дальнейшего развития метода высокоуровневого синтеза СБИС.

По мнению автора целесообразно продолжить работу в направлении разработки алгоритмов преобразования параллелизма и изменений стратегий управления вычислениями (преобразования управляющего графа) при переходе к архитектурным вариантам. В разрабатываемых алгоритмах необходимо реализовать учет метрик производительности (обмен метриками между ФПП и HDL этапами синтеза), получаемых после синтеза архитектурного решения из HDL описания. Это позволит учесть метрики производительности схемы на этапе преобразования параллелизма, что дополнительно увеличит эффективность алгоритма синтеза.

Список сокращений

СБИС – сверхбольшая интегральная схема;

САПР – система автоматизированного проектирования и разработки;

HDL – hardware description language, язык описания аппаратуры;

DSE – design space exploration, задача поиска оптимального архитектурного решения СБИС при заданных ограничениях;

EDA - Electronic design automation, автоматизация разработки электроники;

EDIF - Electronic Design Interchange Format, стандарт формата описания схем и нетлистов;

ESL - Entire System Level;

RTL – register-transfer level, уровень регистровых передач;

HLS – High-level synthesis, высокоуровневый синтез;

PMS – processor, memory, switch, процессор, память, порт;

ФПП – функционально-параллельный подход;

ПЛИС – программируемая логическая интегральная схема;

ИС – интегральная схема;

ИГ – информационный граф;

РИГ – реверсивный информационный граф;

УГ – управляющий граф;

ЯПФ – ярусно-параллельная форма графа;

CFG – Control flow graph, управляющий граф;

DFG – Data flow graph, информационный граф потока данных;

ПО – программное обеспечение;

БА – Бортовая аппаратура;

КА – Космический аппарат;

ГЛОНАСС –спутниковая навигационная система РФ;

GPS – спутниковая навигационная система США;

DVB – стандарт спутниковой связи и вещания;

БМК – Базовый матричный кристалл;

SW – программный (модуль);

HW – аппаратный (модуль);

ПО – программное обеспечение;

ООП – объектно-ориентированное программирование;

DAG - Directed Acyclic Graph, ацикличный ориентированный граф;

ЯПФ - ярусно-параллельная форма;

БПФ – быстрое преобразование Фурье;

DSP – блок арифметических операций на ПЛИС.

Список литературы

1. Xin Wu. 3D-Ie Technologies and 3D FPGA // IEEE 2015 International 3D Systems Integration Conference [Электронный ресурс] URL: https://www.xilinx.com/publications/white-papers/3d-ic-in-3d-fpgas.pdf (дата обращения: 29.01.2019)

2. И.И. Левин, В.А. Гудков. Расширение языка высокого уровня COLAMO для программирования реконфигурируемых вычислительных систем на уровне логических ячеек ПЛИС // Вестник компьютерных и информационных технологий. – М.: Машиностроение, 2010. - №12. – С.10-17.

3. Каляев И. А., Дордопуло А. И., Левин И. И., Гудков В. А., Гуленок А. А. Технология программирования вычислительных систем гибридного типа // Вычислительные технологии. - 2016. - Т. 21, № 3. - С. 33-44. ISSN 1560-7534.

4. IEEE Std 1364-2001. IEEE Standard Verilog Hardware Description Language: введен впервые: дата введения 28.09.2001 / разработан IEEE Computer Society. – New York, NY 10016-5997, USA, 2001. – 791 с.

5. IEEE Std 1076-2008. IEEE Standard VHDL Language Reference Manual : введен впервые : дата введения 26.01.2009 / разработан IEEE Computer Society. – New York, NY 10016-5997, USA, 2009. – 620 с.

6. IEEE 1800-2009. IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language : введен впервые : дата введения 11.12.2009 / разработан IEEE Computer Society. – New York, NY 10016-5997, USA, 2009. – 1285 с.

7. Steven M. Rubin. Computer Aids for VLSI Design. // R. L. Ranch Press – 2009 – 318 с.

8. Алехин, В.А. SystemC. Моделирование электронных систем : Учебное пособие для ВУЗов / В.А. Алехин. – Москва : Горячая линия – Телеком, 2018. – 320 с. ISBN 978-5-9912-0722-5.

9. Стешенко В., Руткевич А., Гладкова Е., Шишкин Г., Воронков Д. Проектирование СБИС типа «Система на кристалле». Маршрут проектирования. Синтез схемы // Электронные компоненты. - №1. – 2009.

10. David Dahan. Synthesis Techniques Using Synopsys' ACS. [Электронный ресурс] URL: https://www.eetimes.com/document.asp?doc_id=1215017 (дата обращения: 29.03.2019)

11. Е. Перельройзен. Проектируем на VHDL. // Солон-Пресс – 2004 – 448 с.

12. Стешенко В.Б. Основы HDL Verilog как средства проектирования цифровых устройств. // Попова Т.В., Малашевич Д.Б. – Москва, МИЭТ – 2006 – 136 с.

13. Patrick Lee. Introduction to Physical Integration and Tapeout in VLSIs. // Lulu.com - 2010 — p. 59.

14. IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis. – IEEE 1076.6-2004

15. Vivado Design Suite User Guide. High-Level Synthesis. UG902 – Xilinx – 2015. [Электронный ресурс] URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ ug902-vivado-high-level-synthesis.pdf (дата обращения: 23.03.2021).

16. RG. Handel-C Language Reference Manual : иллюстрированное научное издание / RG ; Celoxica Limited, 2005. – p. 348.

17. Satnam, Sing. Lava and Jbits: From HDL to bitstream in seconds / S. Singh, P. James-Roxby // 9th IEEE Symp. Field-Programmable Custom Computing Machines – 2001. URL: https://caxapa.ru/thumbs/441669/download.pdf (дата обращения 12.10.2021).

18. Michaelson, G. Compiling Hume down to gates / Michaelson G., Serot J. // Draft Proceedings of 11th International Symposium on Trends in Functional Programming, Madrid. – 2012. URL: https://www.researchgate.net/publication/228964514_Compiling_Hume_down_to_gates (дата обращения 12.10.2022).

19. Alves, C. Erlang inspired Hardware / Alves C., Ferreira P., Ferreira C. // International Conference on Field Programmable Logic and Applications / IEEE Xplore. – Milan, Italy, – 2010. – P. 244–246. – DOI: 10.1109/FPL.2010.56

20. W. Chen. The VLSI Handbook // CRC Press – 2019 – 1788 p.

21. Николич, Б. Цифровые интегральные схемы. Методология проектирования / Б. Николич, Ж. М. Рабаи, А. Чандракасан. - Москва - ООО «ИД Вильямс», 2016. - 912 с.

22. Parnell, K. Programmable Logic Design Quick Start Hand Book / K. Parnell, N. Mentha. Xilinx Corp. - 2004. – 192 p.

23. V. Y. Sarge. Evaluating Simulink HDL coder as a framework for flexible and modular hardware description. // Master's thesis, Massachusetts of Technology, 2018.

24. Xin Cai. Model-Based Design for Software Defined Radio on an FPGA / Xin Cai; Mingda Zhou; Xinming Huang // IEEE Access – 2017.

25. Колесников Е.И. Обзор маршрутов проектирования прикладного программного обеспечения для ПЛИС/ASIC/SoC на основе языков C/C++// Мат. VI Всероссийской межвузовской конференции молодых ученых. СПб: ИТМО – 2009 - С. 175.

26. Bailey B., Martin G., Piziali A. ESL Design and Verification // ESL Des. Verif. 2010. 462 p.

27. Непомнящий О.В., Алекминский С.Ю. Проблемы верификации проекта при сквозном проектировании вычислительных систем на кристалле // Нано- и микросистемная техника. М.: Новые технологии. - 2010 - №9(122). - с. 4-7

28. Б.Я. Штейнберг. Автоматизация разработки программ для параллельных вычислительных систем с распределенной памятью / Б.Я. Штейнберг // Доклады пятой международной конференции «Параллельные вычисления и задачи управления» - Москва, 2010 - с. 767-786.

29. Канышев, В.И. Инженерная микроэлектроника / В.И. Канышев, И.А. Шагурин // Chip-News. – 2006. – № 9(112). – С. 51.

30. J. Bhasker, A. SystemC Primer, Second Edition // Star Galaxy Publishing, 2010. 320 p.

31. Денис Дж.Б., Фоссин Дж.Б., Линдерман Дж.П. Схемы потоков данных // В кн. Теория программирования. Ч 2. Новосибирск: ВЦ СО АН СССР, 1972 г. С. 7-43.

32. Jinian Bian. A hierarchical CDFG as Intermediate Representation for Hardware/Software Codesign / Jinian Bian, Quang Wu, Yunfeng Wang, Weimin Wu.// Communications, Circuits and systems and West Sino Expositions, IEEE 2002 International Conference – 2002 – V.2 - pp.1429-1432.

33. Ejnioui A. Control and data flow graph extraction for high-level synthesis / Namballa R., Ranganathan N. // VLSI Proceedings. IEEE Computer society Annual Symposium on - 2004 - pp. 187-192.

34. Zhongda Yuan. Automatic enhanced CDFG generation based on runtime instrumentation / Zhongda Yuan, Yuchun Ma, Jinian Bian // Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on – 2013.

35. Amellal S. Scheduling of a control and data flow graph / Amellal S., Kaminska B. // IEEE Int. Symp. on Circuits and Systems – 1993 - vol. 3 - pp. 1666–1669

36. Бен-Ари М. Языки программирования. Практический сравнительный анализ. /Бен-Ари М. – Москва, изд. Мир - 2000.

37. Robert Sebesta. Concepts of Programming Languages (11th Edition) – Pearson - 2015 - 800 p.

38. Т. Пратт, М. Зелковиц. Языки программирования: разработка и реализация. 4-е изд. СПб.-Питер, 2003

39.Гордиенко, А. П. Функциональное программирование. / А.П. Гордиенко. – М. : Кнорус, 2022. – 278 с. ISBN: 978-5-406-08432-8.

40. Мараховский В. Б., Розенблюм Л. Я., Яковлев А. В. Моделирование параллельных процессов. Сети Петри. Курс для системных архитекторов, программистов, системных аналитиков, проектировщиков сложных систем управления. - Санкт-Петербург: Профессиональная литература, АйТи-Подготовка, 2014. - 400 с.

41. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ. // Вычислительные технологии, № 1 (10) – 2005 - С. 71-89.

42. Описание языка Пифагор. [Электронный ресурс] URL: http://softcraft.ru/sysprog/lang/ (дата обращения: 29.03.2019).

43. Рыженко И.Н. Технология архитектурно-независимого, высокоуровневого синтеза сверхбольших интегральных схем / Непомнящий О.В., Шайдуров В.В., Легалов А.И., Рыженко И.Н. // Доклады АН ВШ РФ. Новосибирск, НГТУ – 2014, - Т. 57. № 3. – С.35-39 (ВАК, ISSN 1727-2769)

44. Легалов А.И., Васильев В.С., Матковский И.В., Ушакова М.С. Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ. //Труды ИСП РАН, том 29, вып. 5, 2017 г. С. 165-184.

45. Anthony Danalis. Dataflow-based Task Execution through PaRSEC for HPC / G. Bosilca, A. Bouteiller, M. Faverge, T. Herault, J. Dongarra //Innovative Computing Laboratory University of Tennessee – MCSoC - 2014 – [Электронный ресурс] URL: http://xev.arch.is.tohoku.ac.jp/LHAM2014/file/LHAM2014-4-Anthony-Danalis.pdf (дата обращения: 9.03.2019)

46. Jack Dongarra. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability / Jack Dongarra., G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault // IEEE Computing in Science and Engineering. IEEE Xplore. – 2013. – Vol. 15, № 6. P.p. 36 – 45.

47. Jack Dongarra. PTG: An Abstraction for Unhindered Parallelism. / A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, J. Dongarra // Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing. IEEE Xplore. – 2014. – p.p. 21 – 30. – DOI 10.1109/WOLFHPC.2014.8.

48. B. Essink. Using FPGAs as Fine-Grained Static Dataflow Machines /Bas Klein Essink - University of Twente – 2014 – [Электронный ресурс]

URL:http://referaat.cs.utwente.nl/conference /21/paper/7441/ using-fpgas-as-fine-grained-static-dataflow-machines.pdf (дата обращения: 22.03.2019)

49. J. Johansen. Design and Implementation of a Novel Dataflow Model and an Intermediate Representation Language for High Level Synthesis on Field Programmable Gate Arrays /AALBorg Universitet [Электронный ресурс] URL: http://projekter.aau.dk/projekter/files/71270246/main.pdf (дата обращения: 22.03.2019)

50. Ab Al Hadi Bin ab Rahman. Optimizing Dataflow Programs for Hardware Synthesis / Ph.D Thesis – Lausanne-2013 [Электронный ресурс] URL: https://infoscience.epfl.ch/record/195762/files/EPFL_TH6059.pdf (дата обращения: 21.03.2019)

51. J. A. N. Lee. Computer Semantics / Van Nostrand Reinhold Company - 1972.

52. R. E. Frankel. Beyond register transfer: an algebraic approach for architectural description / R. E. Frankel, S. W. Smoliar // Proceedings of the 4th International Conference on Computer Hardware Description Languages – 1979 - pp. 1–5.

53. M. Sheeran. μFP, a language for VLSI design // Proceedings of the Conference Record of the ACM Symposium on LISP and Functional Programming (LISP ’84) - Austin, Texas, USA -1984 - pp. 104–112.

54. S. D. Johnson. Synthesis of Digital Designs from Recursion Equations. // The MIT Press - Cambridge, Mass, USA - 1983.

55. M. D. Ercegovac. A functional language for description and design of digital systems: sequential constructs / F. Meshkinpour, M. D. Ercegovac //Proceedings of the 22nd ACM/IEEE Conference on Design Automation - ACM Press – 1985 - pp. 238 – 244.

56. M. D. Ercegovac. A high-level language approach to custom chip layout design / M. D. Ercegovac, T. Lang // Technical Report MICRO Project Reports 1982-83, University of California, Berkeley - 1982.

57. J. O'Donnell. Hydra: hardware description in a functional language using recursion equations and high order combining forms // Proceedings of the Fusion of Hardware Design and Verification - NorthHolland, Amsterdam - 1988 - pp. 309–328

58. M. Aagaard. HML: a hardware description language based on standard ML / M. Aagaard, J. O'Leary, M. H. Linderman, M. Leeser //Proceedings of the IFIP Conference on Hardware Description Languages and Their Applications (CHDL '93) – 1993 - no. A-32 - pp. 327–334.

59. M. Leeser. HML: an innovative hardware description language and its translation to VHDL / Y. Li, M. Leeser // Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '95) – 1995 - pp. 691–696.

60. A. Mycroft. A statically allocated parallel functional language / A. Mycroft, R. Sharp // Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP '00) - Springer, 2000 - vol. 1853 of Lecture Notes in Computer Science - pp. 37–48.

61. A. Mycroft. Hardware/software co-design using functional languages / A. Mycroft, R. Sharp //Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01) - 2001 - pp. 236–251.

62. B. Cook. Microprocessor specification in hawk / B. Cook, J. Launchbury, J. Matthews // Proceedings of the International Conference on Computer Languages (FTH '98) – 1998 - pp. 90–101.

63. Christiaan Baaij. Using Rewriting to Synthesize Functional Languages to Digital Circuits / Christiaan Baaij, Jan Kuper. // TFP 2013: Revised Selected Papers of the 14th International Symposium on Trends in Functional Programming – V. 8322 – 2013 – pp. 17-33.

64. J. Grundy. A reflective functional language for hardware design and theorem proving / J. Grundy, T. Melham, J. O'Leary // Tech. Rep. PRG-RR-03-16, Oxford Univerity, Computing Laboratory - 2003.

65. C.-J. H. Seger. Practical formal verification in microprocessor design / R. B. Jones, J. W. O'Leary, C.-J. H. Seger, M. D. Aagaard // IEEE Design and Test of Computers – 2001 - vol. 18, no. 4 - pp. 16–25.

66. Hawkins, Tom. HDCaml Home Page. 2006. [Электронный ресурс] URL: http://www.funhdl.org/wiki/doku.php/hdcaml (дата обращения: 2.02.2018)

67. J. Grundy. A reflective functional language for hardware design and theorem proving / T. Melham, and J. O'Leary //Journal of Functional Programming, v. 16(2) – 2006 – p. 157-196.

68. M. Bernardo. Formal Methods for Hardware Verification. / M. Bernardo A. Cimatti. //6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, v. 3965 – 2006 - Springer-Verlag.

69. Легалов А.И., Непомнящий О.В., Матковский И.В., Кропачева М.С. Преобразование хвостовых рекурсий в функционально-потоковых параллельных программах. / Моделирование и анализ информационных систем. Том 19, No 4. - 2012. С. 48-58.

70. Васильев В.С., Легалов А.И., Постников А.И. Особенности преобразования хвостовой рекурсии в функционально-потоковом языке параллельного программирования // Системы. Методы. Технологии. 2013. №3(17). С.106-111.

71. Оптимизация параллельных списков функционально-потокового языка программирования «Пифагор». Васильев В.С., Рыженко И.Н., Матковский И.В. Системы. Методы. Технологии. 2014. № 3 (23). С. 102-107.

72. Левин И.И. Подход к архитектурно-независимому программированию вычислительных систем на основе аспектно-ориентированного языка SET@L. / И.И. Левин, А.И. Дордопуло, И.В. Писаренко, А.К. Мельников // Известия ЮФУ. Технические науки – 2018 – 3 – С. 46 –

73. И.Н. Рыженко. Метод архитектурно-независимого высокоуровневого синтеза. /О.В. Непомнящий, И.Н. Рыженко, А.И. Легалов // Известия ЮФУ. Технические науки – 2018 – 8 – С. 36 – 47.

74. Легалов А.И. Функционально-потоковое параллельное программирование при асинхронно поступающих данных. / Легалов А.И., Редькин А.В., Матковский И.В. // Параллельные вычислительные технологии (ПаВТ'2009) : Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). — Челябинск: Изд. ЮУрГУ, – 2009. – С.573-578.

75. Легалов А.И. Использование асинхронно поступающих данных в потоковой модели вычислений. Третья сибирская школа-семинар по параллельным вычислениям. Сб. Докладов, Томск. Изд-во Томского ун-та, 2006, С. 113-120

76. Свидетельство о государственной регистрации ПО для ЭВМ № 2015619175 “Программа синтеза описания схем на языках описания аппаратуры HDL с языка функционально-параллельного программирования «Пифагор»”. /Комаров А.А., Рыженко И.Н., Непомнящий О.В.

77. Удалова Ю.В. Отладка и верификация функционально-потоковых параллельных программ: дис. канд. технич. наук: 05.13.11/ Удалова Юлия Владимировна – Н., 2015 – 140с.

78. Удалова Ю.В., Легалов А.И., Сиротинина Н.Ю. Средства отладки функционально-потоковых параллельных программ. /Доклады АН ВШ РФ. — 2008. — Т. 10, № 1. — С. 96-105.

79. Удалова Ю.В., Легалов А.И. Верификация функционально-потоковых параллельных программ методом индуктивных утверждений. / Доклады АН ВШ РФ, №2-3(23-24), 2014. С. 125-132.

80. Легалов А.И., Технологические аспекты создания, преобразования и выполнения функционально-потоковых параллельных программ: Научный сервис в сети Интернет: все грани параллелизма // Матковский И.В., Кропачева М.С., Удалова Ю.В., Васильев В.М. / Новороссийск – 2013 – с. 443-447.

81. Васильев В.С. Оптимизация программ функционально-потокового языка Пифагор: Перспективы развития информационных технологий №20 -2014 – с.7-14.

82. Васильев В.С., Легалов А.И. Оптимизация инварианта цикла в языке Пифагор. // Моделирование и анализ информационных систем. – 2018 - №25(4) - с.347-357.

83. Alexander I. Legalov. High-Level Design Flows for VLSI Circuit. / Oleg V. Nepomnyashchy, Alexander I. Legalov and Natalia J. Sirotinina. // Journal of Siberian federal university. Engineering & Technologies – 2014 - Vol. 7, No.6 - pp. 674-684.

84. Razvan Nane, A Survey and Evaluation of FPGA High-Level Synthesis Tools/ Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems – 2015 – Vol. X, No. Y.

85. LegUp High-Level Synthesis – [Электронный ресурс] URL: http://legup.eecg.utoronto.ca/download.php (дата обращения: 22.03.2019)

86. Holland B., Vacas M., Aggarwal V., De Ville R., Troxel I., and George A. D., "Survey of C-based Application Mapping Tools for Reconfigurable Computing" – [Электронный ресурс] URL: http://klabs.org/mapld05/presento/215_holland_p.ppt (дата обращения: 22.03.2019)

87. El-Araby E., Taher M., Abouellail M., El-Ghazawi T., and Newby G. B., “Comparative analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study” / Taher M., Abouellail M., El-Ghazawi T., and Newby G. B. // 2007 3rd Southern Conference on Programmable Logic.

88. Кристофидес Н. Теория графов. Алгоритмический подход. - М.: Мир - 1978 - 432 с.

89. Новиков Ф.А. Дискретная математика для программистов. – СПБ.: ПИТЕР - 2009 – 384 с.

90. Methods and algorithms for a high-level synthesis of the very-large-scale integration/ Oleg Nepomnyashchy, Alexandr Legalov, Valery Tyapkin, Igor Ryzhenko, Vladimir Shaydurov. (Методы и алгоритмы высокоуровневого синтеза сверх больших интегральных схем) // WSEAS Transactions on Computers ISSN / E-ISSN: 1109-2750 / 2224-2872, Volume 15, 2016, Art. #22, pp. 239-247c.

91. Vivado Design Suite User Guide Implementation 2018. [Электронный ресурс] URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug904-vivado-implementation.pdf (дата обращения: 2.02.2018)

92. И. Каршенбойм. Краткий курс HDL.Часть 8. Моделирование в ModelSim SE // Компоненты и технологии – 2008 - №11- с.139-144.

93. ModelSim SE Tutorial [Электронный ресурс] URL: http://www.cs.colby.edu/courses/S15/cs232/labs/lab01/modelsim_se/_tut.pdf (дата обращения: 11.01.2019)

94. Генерация VHDL и Verilog-кода для проектов на ПЛИС и ASIC. [Электронный ресурс] URL: https://matlab.ru/products/hdl-coder/HDL-coder-ru.pdf (дата обращения: 11.01.2019)

95. Using Simulink to Deploy a MATLAB Algorithm on an FPGA or ASIC. [Электронный ресурс] URL: https://www.mathworks.com/videos/using-simulink-to-deploy-a-matlab-algorithm-on-an-fpga-or-asic-1484667134277.html (дата обращения: 11.01.2019)

Список публикаций по теме диссертации

96. Рыженко И.Н. Проблемы и решения проектирования специализированных бортовых вычислительных систем/ Непомнящий О.В., Андреев А.С., Комаров А.А., Рыженко И.Н., Хныкин А.В. // Интеллект и наука: Труды XIII Международной научной конференции. Железногорск: 2013, – С. 38-40.

97. Увеличение максимальной корректируемой ошибки при реализации алгоритма Фитца. / Комаров А.А., Рыженко И.Н., Андреев А.С., Леонова А.В. // Решетневские чтения. Красноярск. – 2015 - Т. 1. № 19. С. 234-236.

98. Methods and algorithms for a high-level synthesis of the very-large-scale integration. Oleg Nepomnyashchy, Alexandr Legalov, Valery Tyapkin, Igor Ryzhenko, Vladimir Shaydurov. (Методы и алгоритмы высокоуровневого синтеза сверх больших интегральных схем) WSEAS Transactions on Computers ISSN / E-ISSN: 1109-2750 / 2224-2872, Volume 15, 2016, Art. #22, pp. 239-247

99. Технология архитектурно-независимого, высокоуровневого синтеза сверхбольших интегральных схем / Непомнящий О.В., Шайдуров В.В., Легалов А.И., Рыженко И.Н. // Доклады АН ВШ РФ. Новосибирск, НГТУ – 2014, - Т. 57. № 3. – С.35-39.

100. Оптимизация параллельных списков функционально-потокового языка программирования «Пифагор» Васильев В.С., Рыженко И.Н., Матковский И.В. Системы. Методы. Технологии. 2014. № 3 (23). С. 102-107, ВАК.

101. Рыженко И.Н. Методы и средства определения частотной ошибки сигнала спутниковой связи в режиме реального времени / Непомнящий О.В., Хабаров В.А., Рыженко И.Н., Комаров А.А. // Известия вузов. Приборостроение. С.Петербург. Спб ИТМО – 2014, - Т. 57. № 3. – С.35-39. (ВАК, ISSN 0021-3454).

102. Непомнящий О.В. Многомерное пространственное кодирование видеоданных в каналах спутниковой связи для малых космических аппаратов / Непомнящий О.В., Митюков В.А., Рыженко И.Н., Хабаров В.А. / Наукоемкие технологии 2015. Т.16. № 3. С. 66-70. (ВАК, ISSN 1999-8465).

103. The VLSI High-Level Synthesis for Building Onboard Spacecraft Control Systems O.V. Nepomnyashchiy, I.N. Ryjenko, V.V. Shaydurov, N.Y. Sirotinina and A.I. Postnikov, Proceedings of the Scientific-Practical Conference "Research and Development - 2016". Springer, Cham.

104. Методы, алгоритмы и программные инструменты архитектурно независимого высокоуровневого синтеза однокристальных цифровых систем. Непомнящий О.В., Рыженко И.Н., Легалов А.И. В сборнике: Суперкомпьютерные технологии (СКТ-2018) Материалы 5-й Всероссийской научно-технической конференции. 2018. С. 104-109.

105. Microprogram control of the switching voltage regulator with the minimum duration of transient phenomena, Yablonskiy A.P., Latyshev R.A., Komarov A.A., Ryzhenko I.N., EUROPEAN RESEARCH, сборник статей победителей VIII международной научно-практической конференции. 2017. С. 73-76.

106. И.Н. Рыженко. Метод архитектурно-независимого высокоуровневого синтеза. /О.В. Непомнящий, И.Н. Рыженко, А.И. Легалов // Известия ЮФУ. Технические науки – 2018 – 8 – С. 36 – 47.

107. I.N. Ryzhenko. Carrier compensation mode implementation in satellite communication channels / I.N. Ryzhenko, A.E. Lutsenko, O.G. Varygin, O. V. Nepomnyashchiy // IEEE: 2019 International Siberian Conference on Control and Communications (SIBCON). Tomsk, 2019, DOI: 10.1109/SIBCON.2019.8729665

108. И.Н. Рыженко. Метод высокоуровневого синтеза и программный инструментарий для описания алгоритмов функционирования СБИС //О.В. Непомнящий, И.Н. Рыженко / Программная инженерия, г. Москва. - 2020 - 1 - с. 34-39.

109. Результаты сравнения методов высокоуровневого синтеза СБИС. Фундаментальные основы инновационного развития науки и образования. Непомнящий О.В., Рыженко И.Н. Сборник статей IX Международной научно-практической конференции, Пенза - 2020г. - с. 46-51.

110. А. И. Легалов, Непомнящий О. В., Рыженко И. Н., Шайдуров В. В. Методы преобразования параллелизма в процессе высокоуровневого синтеза СБИС // Моделирование и анализ информационных систем. 2022. Т. 29. № 1. С. 60-72.

Приложение А Зарегистрированные результаты интеллектуальной деятельности

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

2. Свидетельство о государственной регистрации ПО для ЭВМ № 2016662259 “Сложно-функциональный блок генераторов псевдослучайной последовательности”, Непомнящий О.В., Рыженко И.Н. и др., 3.11.2016.

3. Свидетельство о государственной регистрации ПО для ЭВМ № 2016619714 “Сложно-функциональный блок понижающего сумматора-ограничителя”, Непомнящий О.В., Комаров А.А., Рыженко И.Н. и др., 26.08.2016

4. Свидетельство о государственной регистрации ПО для ЭВМ № 2016619836 “Сложно-функциональный блок интерфейсов вычислительного узла”, Непомнящий О.В., Комаров А.А., Рыженко И.Н. и др., 31.08.2016.

5. Свидетельство о государственной регистрации ПО для ЭВМ № 2015614726 «Программное обеспечение спутникового модема «ЯР-1040», Комаров А.А., Рыженко И.Н., Андреев А. С. и др., 27.04.2015.

6. Свидетельство о государственной регистрации ПО для ЭВМ № 2015619175 “Программа синтеза описания схем на языках описания аппаратуры HDL с языка функционально-параллельного программирования «Пифагор»”, Комаров А.А., Рыженко И.Н., Непомнящий О.В. , 26.08.2015.

7. Свидетельство о государственной регистрации ПО для ЭВМ № 2014660589 “Приёмник спутникового сигнала стандарта DVB-S/S2”, Комаров А.А., Рыженко И.Н., Андреев А. С. и др., 10.10.2014.

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

Приложение Б Акты внедрения

Акционерное общество
«ИНФОРМАЦИОННЫЕ СПУТНИКОВЫЕ СИСТЕМЫ»
имени академика М.Ф. Решетнёва»

РЕШЕТНЕВ
АО «ИСС»

ул. Ленина, д. 52, г. Железногорск, ЗАТО Железногорск, Красноярский край, Российская Федерация, 662972
Тел. (3919) 76-40-02, 72-24-39, Факс (3919) 72-26-35, 75-61-46, e-mail: office@iss-reshetnev.ru, http: //www.iss-reshetnev.ru
ОГРН 1082452000290, ИНН 2452034898

УТВЕРЖДАЮ
Заместитель генерального конструктора
по электрическому проектированию и
системам управления КА

_____С.Г. Кочура
«___» _____2021 г.

М.П.

АКТ
внедрения результатов диссертационных исследований

Настоящим актом подтверждается, что результаты диссертационных исследований по теме: Методы, алгоритмы и программные инструменты архитектурно-независимого высокоуровневого синтеза однокристальных цифровых схем
(тема диссертации)

выполненной Рыженко Игорем Николаевичем.
(Ф.И.О Аспиранта/исследователя - автора)

использованы в: Акционерное Общество «Информационные спутниковые системы» имени академика М. Ф. Решетнёва».
(Полное наименования предприятия)

1. Вид внедренных результатов: Комплекты сложно-функциональных блоков СБИС в составе приборов гражданского назначения: 751.2513-0 БУ БРК, 765.1512-0-01 БУ БКУ, ЕАМ5.1526-0-01 БУН.
2. Форма внедрения: Экспериментальное внедрение при выполнении НИР и СЧ ОКР
3. Новизна результатов научно-исследовательских работ:
качественно новые
(принципиально новые, качественно новые, модификация)

4. Опытно-промышленная проверка в ОКР/СЧ ОКР/НИР: по Гос. Контракту № 14.578.21.0021и СЧ ОКР вн. №№ 20717, 20718.
5. Научно-технический эффект: Предположительное сокращение сроков выполнения работ, сокращение собственного энергопотребления, сокращение требуемой площади кристалла за счет переносимости полученного кода и возможности рассмотрения модельного ряда для вариантов полученных решений.

Заместитель начальника отдела
проектирования и испытаний РЭА _____ И.Н. Тульский

СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ | SIBERIAN FEDERAL UNIVERSITY

МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет»

660041, Красноярский край,
г. Красноярск, проспект Свободный, д. 79
телефон: (391) 244-82-13, тел./факс: (391) 244-86-25
http://www.sfu-kras.ru, e-mail: office@sfu-kras.ru

ОКПО 02067876; ОГРН 1022402137460;
ИНН/КПП 2463011853/246301001

12.05.2022 № 4792

на № _____ от _____

УТВЕРЖДАЮ
Ректор ФГАОУ ВО СФУ
_____ М. В. Румянцев
«___» _____ 2022 г.

АКТ о внедрении

в учебный процесс в ФГАОУ ВО Сибирском федеральном университете результатов кандидатской диссертационной работы Рыженко Игоря Николаевича «Методы, алгоритмы и программные инструменты архитектурно-независимого высокоуровневого синтеза однокристальных цифровых схем»

Настоящим подтверждаем, что результаты диссертационного исследования Рыженко И.Н. на тему «Методы, алгоритмы и программные инструменты архитектурно-независимого высокоуровневого синтеза однокристальных цифровых схем», включающие метод высокоуровневого синтеза описания СБИС и инструментальное средство, обеспечивающее синтез описания СБИС из высокоуровневого описания на функционально-потоковом языке (свидетельство о государственной регистрации программы для ЭВМ №2015619175) обладают актуальностью, предоставляют научный и практический интерес и внедрены в учебный процесс на кафедре вычислительной техники и кафедре высокопроизводительных вычислений Института космических и информационных технологий СФУ. Эти материалы используются при подготовке магистров по направлению 09.04.01 – «Информатика и вычислительная техника», при изучении дисциплин «Программируемые логические интегральные схемы», «Микропроцессорные системы», «Микроконтроллеры и система на кристалле» и при выполнении выпускных квалификационных работ.

И. о. директора института космических и информационных технологий СФУ, к.т.н. Д. В. Капулин

Заведующий кафедрой вычислительной техники, к.т.н. О. В. Непомнящий

Заведующий кафедрой высокопроизводительных вычислений, к.т.н. Д. А. Кузьмин