ДОКЛАДЫ АН ВШ РФ
2014 январь–март № 1 (22)

ТЕХНИЧЕСКИЕ НАУКИ

УДК 004.272.44

ТЕХНОЛОГИЯ АРХИТЕКТУРНО-НЕЗАВИСИМОГО, ВЫСОКОУРОВНЕВОГО СИНТЕЗА СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ

А.И. Легалов, О.В. Непомнящий, И.Н. Рыженко
Сибирский федеральный университет

Изложены результаты анализа методов и маршрутов, обеспечивающих поддержку процесса проектирования однокристальных систем с реконфигурируемой архитектурой. Выделены характерные особенности проектирования однокристальных систем методами нисходящего проектирования и высокоуровневого синтеза. Сформулирована задача повышения эффективности разработки СБИС на основе технологий архитектурно-независимого проектирования. Рассмотрены язык и метод построения аппаратной модели СБИС на основе функционально-потокового подхода. На основании рассмотренной аппаратной модели вычислений предложен подход к разработке архитектурно-независимого представления СБИС.

Применение архитектурно-независимого описания алгоритмов и использование параллелизма на уровне операций в совокупности с потоковой моделью параллельных вычислений на уровне языка позволило разработать принципиально новый маршрут проектирования СБИС. Разработаны методы проектирования цифровых однокристальных систем на основе функционально-потокового подхода для данного маршрута. Отличительной чертой предлагаемого маршрута проектирования является построение промежуточных структур данных, описывающих программу на функционально-потоковом языке. Предложенный подход позволяет ускорить переход от функционально-потокового представления исходных алгоритмов к регистрово-вентильному представлению СБИС. При этом верификация архитектуры СБИС осуществляется на этапе формального описания, до перехода к синтезу системы. Рассмотрены основные особенности и ограничения синтеза функций языка при переходе к регистрово-вентильному представлению.

Ключевые слова: параллельные вычисления, потоки данных, функциональное программирование, система на кристалле, алгоритм, высокоуровневый синтез.

1. Постановка задачи

Разработка СБИС в настоящее время осуществляется с использованием двух основных маршрутов проектирования: традиционного, базирующегося на низкоуровневом представлении исходных алгоритмов с использованием языков описания аппаратуры (HDL – hardware description language), и высокоуровневого, описывающего исходный проект на системном уровне (ESL – Entire System Level).

При традиционном представлении в структуре СБИС выделяют следующие уровни: функциональный, структурный и топологии (геометрии) микросхемы [1]. Причем каждый уровень представления разбивается на дополнительные подуровни. Например, в структуре СБИС можно выделить:

– функциональный уровень структурных модулей: процессор, память, готовый модуль;
– регистровый уровень, на котором задаются межрегистровые связи и описываются мелкие функциональные блоки (RTL – Register transfer level);
– уровень логических вентилей;
– схемотехнический (транзисторный) уровень.

Традиционный маршрут проектирования основывается на представлении проекта на одном из языков описания аппаратуры [2] и представляет собой последовательный спуск по означенным уровням с одновременным перемещением по оси иерархии других областей.

При реализации сложно-функциональных СБИС проектирование на низком уровне неприемлемо в силу целого ряда причин, основными из которых следует считать:

– наличие семантического разрыва между представлениями проекта на системном уровне и уровне регистровых передач;

– изначальное разделение программной и аппаратной составляющих для систем на кристалле (SoC – System on Chip) с последовательным итерационным проектированием и раздельным моделированием каждой;

– частичное или полное ручное преобразование проекта при переходе от уровня к уровню;

– жесткая привязка применяемых библиотек к выбранной целевой платформе или к конкретному кристаллу;

– ограниченные возможности анализа альтернативных вариантов разрабатываемой системы в сжатые сроки, диктуемые экономической целесообразностью проекта.

Попытки решения означенных проблем привели к введению в традиционный маршрут проектирования высокоуровневых программных средств, которые ранее использовались либо с традиционными вычислительными системами, либо были ориентированы на решение совершенно других задач, например MathLab или LabView [3]. Однако использование встроенных средств перехода к описанию аппаратуры не дает должного эффекта. В случае высокой сложности проекта конечная реализация получается громоздкой, не соответствует заданным временным ограничениям и зачастую не удовлетворяет требованиям к информационной емкости при формировании архитектуры целевого кристалла.

Дальнейшее развитие традиционного метода базировалось на основе добавления функциональных моделей на верхнем уровне абстракции или попытках их анализа на RTL-уровне, что привело к появлению проектирования на абстрактном системном уровне. При таком подходе осуществляется проектирование «сверху вниз», основанное на имитационном моделировании, при котором в разрабатываемой модели четко выделены структура, связи между компонентами и способы обмена информацией.

В отличие от традиционного маршрута, где разделение на аппаратную и программную части происходит на верхних уровнях проекта, при ESL-проектиовании такого разделения не происходит. На начальном этапе формулируется задача на проектирование, описывается состав системы в виде функциональных блоков и связей между ними, происходит отладка их взаимодействия с целью проверки разработанного функционального состава на соответствие поставленной задаче. После создания модели переходят к формированию архитектуры системы. В ходе этого процесса сформированные на предыдущем этапе блоки описываются на императивных языках программирования, причем также без разделения на аппаратную и программную части. Для разработанных блоков генерируются тестовые воздействия, фрагменты отладочного программного обеспечения и тесты для будущих аппаратных модулей. На основании тестов и разработанного программного обеспечения происходит детальная верификация полученной программной модели. При обнаружении в ходе верификации ошибок или несоответствия поставленной задаче происходит возврат на первоначальный уровень разработки, декомпозиция или разработка новых блоков спецификаций. Этот процесс повторяется до полного устранения ошибок и несоответствий.

После достижения приемлемого результата происходит ручная декомпозиция разработанной системы на программную и аппаратную части с целью привязки к конечной аппаратной и программной платформе реализации. Однако при таком

подходе полученные типовые структуры не обеспечивают поддержку параллелизма решаемой задачи, хотя локальные задачи для той или иной структуры могут решаться параллельно внутри нее.

В ходе заключительной стадии проектирования на ESL-уровне выполняется совместная интегрированная отладка программного и аппаратного обеспечения (HW/SW – CoVerification) с целью проверки их системной совместимости. На данном этапе разработки используется симуляция системных блоков, при которой каждый из них может быть раскрыт по дереву иерархии до вентильного представления с целью выявления несоответствий требуемых параметров спецификации заданию на проектирование. И вновь возможен возврат к предыдущим уровням иерархии для полной или частичной декомпозиции проекта с целью достижения требуемых параметров [4].

Несмотря на преимущества ESL перед традиционными подходами, нерешенным остается ряд проблем, характерных для обеих методик. Например, для достижения оптимальных значений производительности и необходимой площади кристалла требуется выполнить анализ множества вариантов реализации. На алгоритмическом уровне модификации просты, но при переходе на RTL-описание вновь может потребоваться перекомпоновка всего проекта и его повторная верификация. Кроме того, применяемые на верхнем уровне проектирования языки (C, HandelC, SystemC)являются преимущественно императивными языками программирования и не имеют развитой поддержки параллелизма, в то время как реализация схемы на кристалле представляет собой параллельную потоковую схему обработки данных, т. е. по существу является параллельной вычислительной системой. При их использовании процесс перехода от описания на высокоуровневом языке к описанию на языке HDL по сути представляет собой переход от пошаговой императивной модели вычисления к описанию в виде графа потока данных. Такой переход выполняется либо автоматически средствами разработки, либо осуществляется в полуавтоматическом режиме под управлением разработчика (HLS – high-level synthesis) [5].

Отсутствие описания параллелизма на верхнем уровне представления приводит к сложности перехода к представлению на RTL. При ESL-проектировании автоматический переход от высокоуровневого описания к описанию на RTL практически невозможен ввиду высокой сложности алгоритмов автоматического распараллеливания, которые не имеют приемлемого решения [6].

Таким образом, при проектировании цифровых схем требуется архитектурно-независимый подход, обеспечивающий максимальное абстрагирование исходных алгоритмов от архитектуры целевого кристалла и реализующий механизм перехода на RTL-уровень с поддержкой сквозной верификации и параллелизма на уровне операций. Необходим такой способ представления алгоритмов на самом верхнем уровне иерархии, который при последующем нисходящем проектировании обеспечивал бы сохранение параллелизма, задаваемого на самом верхнем уровне, и допускал сквозную верификацию в ходе формирования топологии кристалла без возврата к предыдущим уровням иерархии.

2. Обоснование и описание метода решения

Любая цифровая схема при реализации на кристалле является набором блоков, соединенных между собой. Связи между блоками можно разделить на два типа: линии данных и линии управления. Каждый блок может быть как элементарным, так и составным. Цифровые схемы работают в синхронном потоковом режиме, т. е. в зависимости от состояния сигналов управления блок обрабатывает входные данные. Следовательно, цифровая схема на кристалле реализует граф потока данных, где узлами являются блоки, а ребрами – сигналы данных и управления. Бло-

ки в цифровой схеме работают параллельно. Зависимость между ними определяется только сигналами управления, которые определяют готовность данных на его входах.

При использовании высокоуровневых языков процесс перехода к описанию на языке HDL по сути представляет собой переход от пошаговой императивной модели вычисления к описанию в виде графа потока данных. Отсутствие в ESL описания параллелизма на верхнем уровне представления приводит к сложности перехода к представлению на RTL. Так как процесс контролируется и выполняется не в автоматическом режиме, а непосредственно разработчиком, требуются значительные временные затраты для реализации системы, что при высокой сложности проекта будет экономически нецелесообразным.

Особенностью архитектурно-независимого синтеза СБИС является переход от высокоуровневого представления исходных алгоритмов к RTL на основе разделения управляющего графа и графа данных (Control and Data Flow Graph) [6, 7]. Такой подход использует одинаковую модель вычислений на всех уровнях разработки, а язык и метод разработки верхнего уровня ориентирован на описание параллельных программ, управляемых по готовности данных.

Идея применения функционального подхода к разработке цифровых схем представлена на текущий момент в работах по четырем языкам: Lava, Hume, F#(Kiwi) и Erlang [8–10]. Однако большинство работ в данной области не получило широкого распространения и, как правило, не выходит за рамки академических проектов. Кроме того, известные языки и методы обладают одним общим и существенным недостатком – отсутствием поддержки параллелизма на уровне элементарных (базовых) операций и модели вычислений.

Для эффективного решения поставленной задачи требуется использование модели вычислений с максимальным параллелизмом, которая позволяет описывать алгоритмы функционирования СБИС без привязки к конечной реализации. В этом случае требуется переход от традиционного представления к использованию параллелизма на уровне операций. Для этого необходимо использовать соответствующую модель вычислений и разработанный на ее основе язык программирования. В работе [11] предложены функционально-потоковая модель параллельных вычислений и язык программирования «Пифагор», использование которых позволяет по-иному взглянуть на проектирование сверхбольших интегральных схем.

Архитектурная независимость предложенной модели вычислений базируется на следующем принципе: считается, что виртуальная машина, предназначенная для выполнения функционально-потоковых (ФП) параллельных программ, имеет неограниченные вычислительные ресурсы, при этом циклические конструкции реализуются за счет рекурсивных вызовов [12, 13]. Это позволяет выделять для каждой операции новый вычислительный ресурс. Процесс перехода от архитектурно-независимого описания параллелизма к конкретной вычислительной системе, в нашем случае цифровой схеме на кристалле, представляет собой редукцию параллелизма под имеющиеся вычислительные ресурсы.

Авторами предложен маршрут проектирования, согласно которому в ходе трансляции с ФП языка строятся следующие промежуточные представления и структуры данных:

– информационный граф (ИГ), описывающий функциональные преобразования данных;

– управляющий граф (УГ), определяющий передачу управления между выполняемыми функциями (передачи управляющих сигналов могут происходить параллельно, что определяется особенностью разрабатываемой программы);

– слой данных, отвечающий за хранение промежуточных значений;

– слой автоматов, реагирующих на поступающие значения сигналов и изменяющих состояния выполняемых операций.

Эти промежуточные структуры используются для перехода к описанию схемы на RTL. При этом одновременно решается задача эффективного сокращения параллелизма под имеющиеся вычислительные ресурсы кристалла согласно заданным ограничениям.

Модель цифровой схемы представляет собой граф с блоками обработки данных, сигналами передачи управления между ними и схемами формирования сигналов, определяющих готовность данных. Аналогом этих частей являются ИГ и УГ ФП модели соответственно. Процесс перехода от ФП модели к описанию цифровой схемы на кристалле в общем виде будет представлять собой синтез схемы для ИГ, а также схемы для УГ, выходы которой соединяются с входами управления схемы ИГ. Одновременно с этим выполняется редукция параллелизма под имеющиеся вычислительные ресурсы кристалла и формальная верификация ФП описания проектируемой системы. Возможность оптимизационных преобразований ФП параллельной программы позволяет оптимизировать СБИС до перехода к архитектурному описанию системы, что не выполняется при использовании существующих методов разработки.

В процессе трансляции с языка функционально-потокового параллельного программирования формируется промежуточная двухуровневая структура, содержащая информационный и управляющий графы. Информационный уровень описывается графом, задающим операции над данными в узлах и связи между ними. Управляющий слой так же описывается графом и задает управление вычислениями. К управляющему уровню относятся также слой конечных автоматов, задающих управление различными операциями языка и обрабатывающий сигналы готовности данных, которые распространяются по УГ.

Согласно предлагаемой методике трансляции графового представления ФП программ в RTL, на этапе трансляции идет построение модели УГ на базе процессорных ядер с заданной программой управления, в то время как ИГ реализуется в виде конечных аппаратных примитивов – специализированных функций для ФП языка. Следует упомянуть, что в отдельных случаях возможно построение как ИГ, так и УГ как при помощи программно-процессорных ядер, так и на основе конечных автоматов, реализуемых на кристалле. По окончании построения происходит генерация полученных моделей в HDL-описание цифровой схемы на кристалле.

Информационный граф задает операции, совершаемые над данными. Реализация данного уровня в виде схемы на кристалле предполагает три стадии:

– представление операций (функций) языка в виде базовых вычислительных блоков – формирование списка требуемых базовых компонент для реализации системы на кристалле;

– реализация интерфейса управления для УГ – формирование дерева межблочных соединений для базовых блоков;

– организация межблочных соединений схемы в соответствии со связями информационного графа.

3. Формирование промежуточного представления информационного графа

При анализе ИГ основной задачей является выделение фаз обработки данных: приведение графа к ярусно-параллельной форме (ЯПФ), оптимизация и редукция графа согласно требуемым критериям параллелизма, а также ограничениям согласно заданию на проектирование. В результате анализа получается ряд промежуточных представлений ИГ согласно фазам обработки потока данных. Одно из таких представлений будет использоваться для последующего синтеза и многокритериального анализа однокристальной системы.

Для достижения максимальной эффективности распараллеливания с учетом ресурсных ограничений кристалла на этапе построения промежуточного слоя происходит оптимизация и формальная верификация фаз. Формируется динамический перечень необходимых и достаточных требований к конечной реализации, основными из которых являются:

– внутренняя тактовая частота кристалла на корневом выходе дерева внутренней частотной шины (Core frequency), влияющая на задержку исполнения фазы (такта) графа;

– потребляемая мощность, которая косвенно оценивается по количеству задействованных элементарных логических блоков (вентилей);

– требуемая площадь кристалла, выраженная в суммарном значении площади элементарных логических блоков.

На основе этих ограничений в автоматическом режиме происходит разбиение графа на фазы исполнения с параллельной оптимизацией согласно заданным критериям. По результатам преобразования возможен экспертный анализ, изменение критериев и повторная итерация. То есть возможно множественное преобразование одного представления ИГ программы в промежуточное представление с минимальными временными затратами, для последующего синтеза в RTL.

Процесс преобразования графа в этом случае делится на две стадии: планирование и выделение ресурсов.

На стадии планирования при анализе графа решается, какие операции будут выполняться параллельно и одновременно (сокращение параллелизма); здесь же происходит выделение фаз исполнения – построение ярусно параллельной формы (ЯПФ) графа. На стадии выделения ресурсов каждой функции и операторам ФП языка присваиваются соответствующие аппаратные функции, а также формируются аппаратные ресурсы для хранения промежуточных результатов в виде требуемого количества регистров.

В результате формируется отчет о распределении ресурсов кристалла, занимаемой площади и потребляемой мощности на различных заданных частотах функционирования. Далее происходит привязка к целевой платформе для реализации однокристальной вычислительной системы. Такая платформа может быть абстрактной, представляющей собой перечень сформированных требований к кристаллу для реализации в виде заказной (ASIC – application-specific integrated circuit) или полузаказной (ASSP – Application-specific standard product) микросхемы, или выбираться в автоматическом режиме из имеющегося банка рекомендованных к применению кристаллов для реализации в виде многократно-программируемой СБИС – FPGA.

При этом для каждой функции (узла ИГ) определяется ее тип и задержка выполнения, выраженная в машинных тактах (фазах опорной тактовой частоты), которая впоследствии будет использована для расчетов и анализа эффективности получаемых моделей графа. В этом случае для каждой операции определяется ее тип t, который задается двумя параметрами: занимаемый ресурс R (ti –> $\{R_1, R_2,...\}$) и время исполнения в тактах N. Для каждого преобразования ИГ выполняется оценка $\{R, N\}$ и по результатам базового преобразования осуществляется цикл эквивалентных преобразований ИГ путем редукции параллелизма – например, в сторону повышения производительности или уменьшения занимаемой полезной площади кристалла.

4. Методы преобразования модели вычислений с максимальным параллелизмом в целевую модель вычислений

Рассмотрим подробнее реализацию стадии планирования при преобразовании представления ФП программы в описание схемы на кристалле. Основной задачей, решаемой на данном этапе, является сокращение параллелизма в соответствии с

имеющимися ресурсами целевого кристалла. Рассмотрим операторы и структуры языка, задающие параллелизм, и возможные методы их преобразования при трансляции в цифровую схему. Параллелизм, заложенный в ФП модель вычислений, можно условно разделить на два класса:

– параллелизм между разными операциями (вершинами) в информационном графе;

– параллелизм внутри одной вершины графа, заданный конструкциями языка, такими как параллельный и асинхронный список.

Преобразование параллелизма, заложенного в информационном графе модели, осуществляется путем разбиения графа на ярусы и перехода к ярусно-параллельной форме представления. При проведении данных преобразований учитываются ограничения, наложенные разработчиком на реализуемую схему.

Рассмотрим подробнее возможные преобразования параллелизма, задаваемые параллельным и асинхронным списком. Преобразование параллелизма, задаваемого параллельным списком, также осуществляется с учетом наложенных ограничений. Например, конструкция $[x_1, x_2, x_3, x_4, x_5, x_6]$: f из параллельного списка данных с 6 элементами $x_1, x_2, x_3, x_4, x_5, x_6$ и одной функции f может быть преобразована в различные варианты схемы на кристалле с разной степенью параллелизма. Схема, обеспечивающая последовательное вычисление функции f, состоит из одного блока, который обрабатывает аргументы x_1–x_6 за шесть временных тактов. Для параллельных вычислений используются 6 одинаковых блоков, выполняющих одновременные вычисления за один временной такт. Возможны также промежуточные варианты. Синтезатор в процессе синтеза, в зависимости от наложенных для данного варианта синтеза ограничений (скорость, потребляемая мощность, занимаемый ресурс), реализует тот или иной вариант схемы. Такое преобразование также комбинируется в процессе синтеза с разбиением на ярусы исполнения всего информационного графа.

Асинхронный список [14] asynch(g_1, g_2, … g_n) представляет собой совокупность данных, поступающих в произвольной последовательности. По мере готовности данных к ним применяется соответствующая операция. Данная конструкция языка, в зависимости от временных соотношений между операциями поступления и обработки данных, может интерпретироваться как набор альтернативных алгоритмов с разной степенью параллелизма, описывающих одну и ту же задачу. В зависимости от того, какое количество вычислительных блоков используется в схеме, реализуется схема с разной степенью параллелизма. Задавая различные ограничения на этапе синтеза схемы, можно получать все возможные варианты реализации потоковых схем обработки данных с различной степенью параллелизма.

В разрабатываемой методологии проектирования цифровых схем изменение степени параллелизма путем задания соотношения между скоростью поступления данных и временем их обработки реализовано посредством генерации схем потоковой цифровой обработки сигналов из готовых блоков (IP core). Задавая частоту следования данных, можно менять степень параллелизма системы и количество используемых ресурсов при генерации схемы для данного блока. Например, реализуя фильтр с 4 умножениями и сложениями и задав отношение скорости данных к скорости обработки 1:1, получим фильтр на 7 блоках (4 умножения и 3 – древовидное сложение), задав соотношение 1:4 – 4 блока.

Таким образом, использование асинхронных списков и различных вариантов их трансляции при переходе от функционально-потокового параллельного (ФПП) представления к RTL-описанию схемы позволяет реализовать предложенную методику перехода на любой схеме и получить все множество вариантов реализации потоковых схем с разной степенью параллелизма, используя одно исходное опи-

сание на ФП языке параллельного программирования. В процессе трансляции необходимо только задать ограничения на скорость поступления данных (соотношение между скоростью поступления и скоростью обработки).

5. Преобразование предопределенных функций ФПП программ

Рассмотрим основные особенности и ограничения синтеза функций языка при переходе к RTL-представлению. Арифметические операторы *, +, –, % и / при переходе к схеме на кристалле могут быть реализованы для целых чисел, чисел с фиксированной точкой и булевых значений. Разрядность данных будет задана типами переменных. Для чисел с фиксированной точкой задается разрядность целой и дробной части. Операции сравнения =, !=, <, <=, > и >=, в отличие от базового ФПП языка, допускают в качестве аргументов числовые (целочисленные и с фиксированной точкой), булевы и символьные значения. Операции сравнения для функций и типов не используются.

Операция ?, формирующая номера позиций истинных булевых значений, в ФПП языке приводит к изменению размерности списка, например:

(true, false, true, false, false, true):? ⇒[1, 3, 6]

Для того чтобы избежать изменения размерности при переходе к RTL-представлению, неиспользуемые значения устанавливаются в –1, пример:

(true, false, true, false, false, true):? ⇒[1, 3, 6, –1, –1, –1]

Данная операция реализуется с использованием нескольких мультиплексоров, на вход выбора каждого из которых будет подан один из элементов списка булевых аргументов. На выходе каждого *i*-го мультиплексора будет сформировано число *i*, если на вход подано значение true. В противном случае на выходе будет сформирована –1.

Операция # используется для транспонирования элементов списка. При синтезе схемы данная функция реализуется как блок коммутации входов на выходы.

Операция диапазона .. (две точки) формирует список данных из заданных граничных значений и шага, например:

(–3,5; 2,0; 1,5):.. ⇒ (–3,5; –2,0; –0,5; 1,0)

При определенных на этапе компиляции размерности выходного списка, граничных значениях и шаге данный блок синтезируется в блок констант.

Использование целочисленной константы или булевой переменной в качестве функции над списком известной на этапе компиляции размерности преобразуется в мультиплексор с двумя (булево значение) или *N* входами (целое число от 1 до *N*). К схеме с целочисленным значением, если оно неизвестно на этапе компиляции, добавляется схема проверки на выход за границы диапазона длины списка.

Выводы

Несмотря на значительные успехи в области традиционного проектирования СБИС и наличие перспективных направлений высокоуровневой разработки, существует комплекс нерешенных проблем, решение которых позволит повысить возможности соответствующих инструментов при разработке систем параллельной обработки данных и систем с реконфигурируемой архитектурой. Предлагаемая технология архитектурно-независимого синтеза СБИС, в отличие от известных традиционных методов и технологий высокоуровневого ESL-синтеза, позволяет осуществлять первоначальное проектирование на уровне описания параллельного вычислительного процесса с переносом полученной модели на целевой кристалл. Использование функционально-потоковой модели вычислений, поддержки параллелизма на уровне операций и параллельной потоковой модели на всех стадиях процесса высокоуровневого проектирования СБИС позволяет выйти на качественно новый уровень проектирования однокристальных систем.

ЛИТЕРАТУРА

[1] **Grout I.** Digital Systems Design with FPGAs and CPLDs // *Elsevier ltd*. – Burlington, MA01803, USA. – 2008. – 724 p.

[2] **Greaves D., Singh S.** Designing Application Specific Circuits with Concurrent C# Programs. Formal Methods and Models for Codesign (MEMOCODE) // *8th IEEE/ACM International Conference.* – July, 2010. – Pp. 21–30.

[3] **Строгонов А.** Проектирование цифровых фильтров в системе MATLAB/Simulink и САПР ПЛИС Quartus. Компоненты и технологии. – СПб.: Файнстрит. – 2008. – № 6. – С. 13–17.

[4] **Greaves David J., Singh S.** Exploiting System-Level Concurrency Abstractions for Hardware Descriptions // *TechReport MSR-TR-2009-48.* Microsoft Corporation. – April, 2009.

[5] **Singh S.** Designing Reconfigurable Systems in Lava // *17th International Conference on VLSI Design.* – 2004. – Pp. 299–306.

[6] **Amellal S., Kaminska B.** Scheduling of a control and data flow graph // *In IEEE Int. Symp. on Circuits and Systems.* – Vol. 3. – May, 1993. – Pp. 1666–1669.

[7] **Namballa R., Ranganathan N.** Control and Data Flow Graph Extraction for High-Level Synthesis // *In IEEE Int. Symp. On VLSI.* – Feb, 2004. – Pp. 187–192.

[8] **Abdallah Al Zain, Vanderbauwhede W., Michaelson G.** Hume to FPGA // *In Draft Proceedings of 10th International Symposium on Trends in Functionsl Programming* (*TFP10), University of Oklahoma.* – Oklahoma, USA. – 2010. – Pp. 151–164.

[9] **Sérot J., Michaelson G.** Compiling Hume down to gates // *Draft Proceedings of 11th International Symposium on Trends in Functional Programming.* – May, 2011. – Pp. 191–226.

[10] **Ferreira P., Ferreira C., Alves C.** Erlang inspired Hardware // *International Conference on Field Programmable Logic and Applications.* – August, 2010. – Pp. 244–246.

[11] **Легалов А.И.** Функциональный язык для создания архитектурно-независимых параллельных программ // *Вычислительные технологии*, ИВТ СО РАН. – 2005. – № 1 (10). – С. 71–89.

[12] **Legalov A.I., Nepomnyaschy O.V., Matkovsky I.V., Kropacheva M.S.** Tail Recursion Transformation in Functional Dataflow Parallel Programs // *AutomaticControlandComputerSciences.* – 2013. – Vol. 47. – No. 7. – Pp. 366–372.

[13] **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*. Сб. докладов. – Я.: ЯрГУ, 2012. – Т. 19. – № 4. – С. 48–58.

[14] **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.

THE TECHNOLOGY OF ARCHITECTURE-INDEPENDENT HIGH-LEVEL VLSI SYNTHESIS

Legalov A.I., Nepomnyaschy O.V., Rizhenko I.N.
Siberian Federal University ,Krasnoyarsk, Russia

The article presents the results of the analysis of the methods and design flows supporting the design of a single-chip computing system with a reconfigurable architecture. Some distinctive features of the development of a single-chip computing system with the use of a top-down design and a high-level synthesis are revealed. The problem of increasing the effectiveness of the VLSI design process based on the architecture-independent design technology is defined. A functional dataflow model of parallel computing, the language and method of constructing a VLSI hardware model based on the functional dataflow approach are considered. An original approach to the development of an architecture-independent VLSI representation based on the proposed hardware computation model is suggested. The application of an architecture-independent description of algorithms and the use of parallelism at the level of operations in conjunction with the dataflow model of parallel computing make it possible to develop a new type of the VLSI design flow. Methods for designing a digital single-chip computing system on the basis of the functional dataflow approach have been developed for this design flow. An outstanding feature of the proposed

design flow is the construction of intermediate data structures which describe the program in the functional dataflow language. The proposed approach allows an efficient conversion of the initial functional representation of dataflow algorithms into the register-gate representation of VLSI. The verification of the VLSI architecture is carried out at the formal description stage before proceeding to the system synthesis. The main features and limitations of the language functions synthesis in the transition to the register-gate representation are considered.

Keywords: parallel computing; data flows; functional programming; system-on-chip; algorithm; high-level synthesis.

REFERENCES

[1] **Grout I.** Digital Systems Design with FPGAs and CPLDs. Burlington (MA), Elsevier ltd., 2008, 724 p.

[2] **Greaves D., Singh S.** Designing Application Specific Circuits with Concurrent C# Programs. Formal Methods and Models for Codesign (MEMOCODE). *Proc. of the8th IEEE/ACM International Conference*, 2010, pp. 21–30.

[3] **Strogonov A.** Proektirovanie cifrovyh fil'trov v sisteme MATLAB/Simulink i SAPR PLIS Quartus [Design of digital filters in the MATLAB / Simulink and FPGA CAD Quartus].*Komponenty i tehnologii*, 2008, no. 6. pp. 13–17.

[4] **Greaves D.J., Singh S.** Exploiting System-Level Concurrency Abstractions for Hardware Descriptions. TechReport MSR-TR-2009-48. Microsoft Corporation. April 2009.

[5] **Singh S.** Designing Reconfigurable Systems in Lava. *Proc. of the 17th International Conference on VLSI Design*, 2004, pp. 299–306.

[6] **Amellal S., Kaminska B.** Scheduling of a control and data flow graph. *In IEEE Int. Symp. on Circuits and Systems*, 1993, vol. 3, pp. 1666–1669.

[7] **Namballa R., Ranganathan N.** Control and Data Flow Graph Extraction for High-Level Synthesis. *In IEEE Int. Symp. on VLSI*, 2004, pp. 187–192.

[8] **Abdallah Al Zain, Wim Vanderbauwhede, Michaelson G.** Hume to FPGA. *In Draft Proceedings of 10th International Symposium on Trends in Functionsl Programming* (*TFP10*), University of Oklahoma, Oklahoma, USA, 2010, pp. 151–164.

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

[10] **Ferreira P., Ferreira C., Alves C.** Erlang inspired Hardware. *International Conference on Field Programmable Logic and Applications*. August, 2010, pp. 244–246.

[11] **Legalov A.I.** Funkcionalny yazuk dlya sozdaniya arhitekturno-nezavisimyh parallelhyh program. [Functional language for architecture-independent programming]. *Vychislitelnye tehnologii – Computing Technologies*, 2005, no. 1(10), pp. 71–89.

[12] **Legalov A.I., Nepomnyaschy O.V., Matkovsky I.V., Kropacheva M.S.** Tail Recursion Transformation in Functional Dataflow Parallel Programs. *Automatic Control and Computer Sciences*, 2013, vol. 47, no. 7, pp. 366–372.

[13] **Legalov A.I.** Nepomnyaschy O.V., Matkovsky I.V., Kropacheva M.S. Preobrazovanie hvostovyh recursiy v funkcionalno-potocovyh parallelhyh programmah [Tail Recursion Transformation in Functional Dataflow Parallel Programs]. *Modelirovanie I analiz informacionnyh system* [*Proc.* "*Modeling and analysis of information systems*"]. Yaroslavl, 2012, vol. 19, no. 4, pp. 48–58.

[14] **Legalov A.I.** Ispolzovanie asinhronno postupayshih dannyh v potokovoi modeli vychislenyi [Using asynchronous data in functional data-flow model calculations]. *Tretya sibirskaya shkola-seminar po parallelnym vichisleniam* [*Proc. 3th Scool-Symp.* "*Parallel calculations*"]. Tomsk, 2006, pp. 113–120.

СВЕДЕНИЯ ОБ АВТОРАХ

Легалов Александр Иванович – родился в 1956 году, д-р техн. наук, профессор, заведующий кафедрой вычислительной техники Сибирского федерального университета. Область научных интересов: модели параллельных вычислений и методы параллельного программирования, системное программирование, технологии программирования, разработка трансляторов и языков программирования. Опубликовано 160 научных работ. (Адрес: 660041, Россия, Красноярск, пр. Свободный, 79. Email: legalov@mail.ru)

Legalov Alexander Ivanovich (b. 1956) – Doctor of Science (Eng.), Professor, Head of Computer Engineering Department of the Siberian Federal University. His research interests are currently focused on parallel computing model and parallel programming techniques, system programming, software engineering, compilers and programming languages development. He is author of 160 scientific papers (Address: 79, Svobodny Av., Krasnoyarsk, 660041, Russia. Email: legalov@mail.ru)

Непомнящий Олег Владимирович – родился в 1968 году, канд. техн. наук, доцент, руководитель НУЛ МПС ИКИТ Сибирского федерального университета. Область научных интересов: однокристальные вычислительные системы, теория и практика параллельных вычислений, вычислительные системы ответственного применения. Опубликовано 117 научных работ. (Адрес: 660074, Россия, Красноярск, ул. ак. Киренского, 79)

Nepomnyaschy Oleg Vladimirovich (b. 1968) – PhD (Eng.), Associate Professor, Head of Scientific and Educational Laboratory of Microprocessor Systems in the Institute of Space and Information Technologies in Siberian Federal University. His research interests are currently focused on single-chip computer systems, theory and practice of parallel computing, responsible use computer systems. He is author of 117 scientific papers. (Address: 79, Kirenskogo St., Krasnoyarsk, 660074, Russia)

Рыженко Игорь Николаевич – родился в 1981 году, окончил Сибирский федеральный университет (СФУ), с 2012 года аспирант кафедры вычислительной техники СФУ. Область научных интересов: цифровая обработка сигналов, технологии высокоуровневого синтеза СБИС. (Адрес: 660074, Россия, Красноярск, Борисова, 30)

Rizhenko Igor Nikolaevich (b. 1981) – graduated from the Siberian Federal University (SFU), Post-graduate Student of Computer Engineering Department of the SFU. Area of research: digital signal processing, high-level VLSI synthesis technology. (Address: 30, Borisova St., Krasnoyarsk, 660074, Russia)

Статья поступила 11 февраля 2014 г.
Received 11 Feb. 2014

To Reference:
Legalov A.I., Nepomnyaschy O.V., Rizhenko I.N. Tekhnologiya arkhitekturno-nezavisimogo, vysokourovnevogo sinteza sverkhbol'shikh integral'nykh skhem [The technology of architecture-independent high-level VLSI synthesis]. *Doklady Akademii Nauk Vysshei Shkoly Rossiiskoi Federatsii* [*Reports of Russian Higher Education Academy of Sciences*], 2014, no. 1(22), pp. 93–103. (in Russ.).