УДК 004.315

МАРШРУТЫ И МЕТОДЫ ВЫСОКОУРОВНЕВОГО СИНТЕЗА СВЕРХ БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ

Непомнящий О.В.,
Кочан И.В., Кондратьев К.В., Митюков В.А.
Сибирский Федеральный университет

Рассмотрены маршруты и методы синтеза однокристальных вычислительных систем. Изложены проблемы традиционных методов проектирования. Предложена технология архитектурно-независимого проектирования вычислительных систем на кристалле.

Ключевые слова: интегральная схема, высокоуровневое проектирование, верификация, алгоритм.

При разработке однокристальных вычислительных систем, маршрут проектирования описывает все этапы создания проекта – от выработки концепции до производства конечных изделий.

Наиболее распространенным является традиционный маршрут нисходящего проектирования [1], с предварительной абстракцией проекта на системном уровне (ESL – Entire System Level). Известны также технологии восходящего проектирования, применяемые, в основном, для больших проектов. Например, компания Synopsys, предлагает технологию Automated Chip Synthesis (ACS, автоматизированный синтез кристаллов) или RTL Budgeting (RTL – Register Transfer Level, уровень регистровых передач), которая заключается в предварительном определении временных и других ограничений для каждой составной части проекта. Здесь, для каждой части в отдельности синтезируются регистровая и вентильная структуры, после чего осуществляется переход к конструкторскому проектированию. Благодаря декомпозиции время синтеза уменьшается в 5-10 раз по сравнению с продолжительностью традиционного нисходящего проектирования [1].

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

- уровень структурных модулей: процессор, память, готовый модуль и др.
- регистровый уровень, на котором задаются межрегистровые связи и описываются мелкие функциональные блоки;
- уровень логических вентилей;
- схемотехнический (транзисторный) уровень.

Традиционный маршрут проектирования основывается на представлении проекта на одном из языков описания аппаратуры (HDL – hardware description language) и представляет собой последовательный спуск по означенным уровням.

При таком подходе изначально разрабатывается техническое задание на проектирование, в том числе формируются пакеты спецификаций проекта, на основе которых происходит разработка поведенческой модели, учитывающей сложные условия эксплуатации. В качестве исходных спецификаций используются тексты программ на универсальных языках программирования (C/C++), описания на проблемно-ориентированных алгоритмических языках (например, SystemC). Также могут рассматриваться модели отдельных узлов системы, представленные на функциональном уровне (например, с помощью Matlab-Simulink), готовые программные (SW – software), аппаратные (HW – hardware) блоки на языках описания аппаратуры (VHDL/Verilog).

Далее осуществляется проработка и создание модели проекта на функционально-алгоритмическом уровне. На этом этапе формируется описание алгоритмов взаимодействия системы с внешним окружением и происходит переход к структуре будущей системы посредством компоновки высокоуровневого представления SW/HW модулей в единый вычислительный комплекс и его верификации.

При компоновке и верификации используют системы, позволяющие выполнять одновременную высокоуровневую отладку на различных языках и совместное логическое моделирование, например, Active-HDL, ModelSim и Matlab/Simulink.

В результате разработчики получают отлаженное структурное описание системы на уровне процессор-память-порт (PMS – processor, memory, switch).

Основные достоинства применяемого на данном этапе инструментария – это полная абстрагируемость от конечной реализации, возможность быстрой модификации и моделирования. К основным недостаткам следует отнести:

- отсутствие средств описания параллелизма;
- семантический разрыв между исходным управлением вычислениями в программе и тем, как оно будет организовано на уровне микросхемы;
- невозможность без полного или частичного, ручного или полуавтоматического перепроектирования преобразовывать проект в программы на языке описания аппаратуры для представления на уровне регистровых передач;
- необходимость приобретения или разработки собственных библиотек для используемой среды проектирования под конкретный целевой кристалл или платформу.

На следующем этапе происходит формирование функционально законченной системы в виде компоновочного плана кристалла. В недавнем прошлом методики описания

проектируемой системы на функционально-логическом уровне иерархи носили узкоспециализированный характер. Инструментарий представлял собой отдельные программные продукты, поставляемые на рынок сторонними корпорациями, либо узкоспециализированные системы, предназначенные для внутреннего применения конкретной компании разработчика или производителя. В настоящее время применяемые на данном этапе системы автоматизации проектирования, например, Quartus от компании Altera, или Xilinx ISE от Xilinx, имеют все требуемые средства для эффективного управления проектом, начиная с разработки структуры системы.

Основной этап традиционного маршрута – этап блочного синтеза, в процессе которого выполняется распределение операций алгоритма по временным тактам и по функциональным блокам аппаратуры, выбирается тип памяти.

Результатом блочного синтеза при традиционном проектировании является компоновочный план кристалла, соответствующий логике функционирования системы – первый шаг к представлению проекта на RTL.

На следующем этапе определяется структура схемы на RTL, в том числе типы блоков (комбинационные или последовательные), реализуются распараллеливание и конвейеризация вычислений. После этого производится окончательное моделирование работы устройства на уровне передачи информации между регистрами.

В заключение, производится компиляция проекта для реализации в конечном кристалле. Полученное RTL-представление на языке описания аппаратуры преобразуется в вентильную структуру. Результат данного этапа – модель вентильного уровня (gate level). После этого выполняется тестирование полученной модели, при котором проверяются временные соотношения сигналов, и производится окончательная трассировка на физическом уровне [2].

На основании вышеизложенного отметим основные проблемы традиционного маршрута проектирования:

1. Наличие семантического разрыва между представлениями проекта на системном и RTL уровнях.
2. Изначальное разделение программной и аппаратной составляющих с последовательным итерационным проектированием и раздельным моделированием каждой.
3. Частичное или полное ручное преобразование проекта при переходах от уровня к уровню с жесткой привязкой применяемых библиотек к выбранной целевой реализации.
4. Ручная разработка программного обеспечения для разработанной аппаратной части.

5. Ограниченные возможности анализа альтернативных вариантов разрабатываемой системы

Причины означенных проблем кроются в существенной разнице между уровнями абстракции при функциональном и RTL-описаниях проекта. При разработке системы на верхних, абстрактных, уровнях не учитывается целый ряд параметров функционирования системы (например, временных), в то время как при описании на HDL-языке они зачастую выходят на первый план. Поэтому встает проблема перехода от описания на алгоритмическом языке к описанию на языке представления аппаратуры.

При таком подходе система разрабатывается под изначально определенную архитектуру и функциональный состав встроенных элементов СБИС специального назначения на основе стандартных или рекомендованных производителем библиотек, используемых для конкретной реализации. Если библиотеки разрабатываются проектировщиком, то нет гарантии, что при изготовлении сверхбольших интегральных схем (СБИС) они окажутся реализуемыми у производителя. Причем даже для разработки систем на кристаллах от одного и того же производителя может потребоваться использование различных систем проектирования. Например, система проектирования QUARTUS от компании Altera не поддерживают семейства простых программируемых логических интегральных схем (ПЛИС) MAX и Flex первых поколений, а система проектирования MAX II Plus этой же компании не предназначена для работы со старшими семействами ПЛИС ACEX, Mercury, APEX, Excalibur и Cyclone. Для большинства систем на базе ПЛИС Xilinx применяется бесплатный пакет ISE WebPack, но для старших семейств ПЛИС этой компании придется приобретать полную версию системы проектирования Xilinx ISE.

При традиционном проектировании любая прикладная программа, начиная с функционального уровня представления, будет ориентирована на способ и метод ее применения, конкретную архитектуру и платформу.

Применяемые на сегодняшний день для системного представления в традиционном маршруте проектирования алгоритмические языки, например, C/C++ или платформа MatLab, могут использоваться на верхнем уровне абстракции. На этом уровне происходит общая алгоритмизация системы, а отработка машинных алгоритмов еще невозможна. Существующие языковые средства описания аппаратуры (HDL-языки) применяются на более низком уровне, где система имеет привязку к конкретной архитектуре. Именно это и обуславливает наличие семантического разрыва между представлением СБИС на верхних уровнях абстракции и воплощением в кристалл.

При реализации спецификаций используются алгоритмические языки системного уровня, базирующиеся на представлении на языке С и имеющие встроенные типы данных HDL языков – SystemC и HandelC. Эти языки создавались специально как С-подобные языки

описания аппаратуры, поэтому уровень их абстрагируемости от RTL недостаточно высок. С другой стороны, код на этих языках моделируется быстрее, чем, например, описание на языке VHDL или Verilog. Тем не менее, при моделировании и отладке системы происходит жесткая привязка к конкретному архитектурному решению.

Средства синтеза для этих языков достаточно громоздки, и для того, чтобы осуществлять эффективную трансляцию в RTL-код, которая выполняется вручную, требуется высокая квалификация инженера и серьезное освоение инструментов верификации и транслирования.

Дальнейшее развитие традиционного метода на основе добавления функциональных моделей на верхнем уровне абстракции или попытки их анализа на RTL-уровне привели к появлению понятия проектирования на абстрактном системном уровне (Electronic System Level, ESL), которое активно развивается с 2004 года и поддерживается всеми ведущими производителями ПЛИС, в первую очередь компанией Xilinx.

Под ESL понимают любой уровень абстракции выше уровня регистровых передач. Языки проектирования, применяемые на этом уровне, ближе по синтаксису и семантике к языку ANSI C, чем к языкам HDL. 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. Кроме того, для каждой модели имеется банк графических образов.

При таком подходе на верхнем уровне абстракции к традиционному этапу описания и формирования пакета спецификаций добавляют два дополнительных уровня [2]: уровень сообщений (Message Level) и уровень передач (Transaction Level). При этом спецификация системы описывается на уровне объектных диаграмм.

В отличие от традиционного маршрута, где разделение на аппаратную и программную части происходит изначально, при ESL проектировании на начальных этапах создания системы такого разделения не происходит. На этом этапе формулируется задача на проектирование, описывается состав системы в виде функциональных блоков и связей между ними, а также происходит отладка их взаимодействия с целью проверки разработанного функционального состава на соответствие поставленной задаче. В этом случае разработка, отладка и верификация алгоритмов, решающих заданную прикладную задачу, полностью отделены от последующего процесса переложения этих алгоритмов на архитектуру радиационно-стойкой ПЛИС.

После создания модели переходят к формированию архитектуры системы. В ходе этого процесса сформированные на предыдущем этапе блоки описываются на стандартных 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.

К сожалению, нет гарантии, что все возможные варианты реализации проекта будут рассмотрены. Однако отладка на самом верхнем уровне позволяет существенно сэкономить общее время проектирования, хотя и требует высокой квалификации проблемно-ориентированных разработчиков.

После достижения приемлемого результата происходит ручная декомпозиция разработанной системы на программную и аппаратную части с целью привязки к конечной типовой аппаратной (микропроцессор) и программной платформе реализации. 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. Для этого каждый блок вновь представляется на SystemC/SystemVerylog и генерируется пакет тестов для проверки аппаратной части системы. 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 в иерархии для полной или частичной декомпозиции проекта с целью достижения требуемых параметров.

На основании полученных результатов моделирования и симуляции анализируется несколько вариантов реализации с целью определения оптимального разделения на аппаратную и программную части.

Результатом проектирования системы на ESL является отлаженная на функциональном и алгоритмическом уровне высокоуровневая модель с четко выраженными программными и аппаратными модулями, представленными на алгоритмических языках.

Дальнейший переход на RTL уровень осуществляется либо вручную, путем переписывания программного кода на язык HDL, либо полуавтоматически, например, из SystemC генерируется Verilog/VHDL-код.

Полная автоматизация перехода на RTL практически невозможна, исключение составляет синтезируемый код SystemC. Результатом такого синтеза будет внутренняя микроархитектура СБИС в виде “черного ящика”. Отметим, что при преобразовании, выполненном вручную, выигрыш в размере файла прошивки для кристалла по сравнению с файлом, полученным при помощи технологии логического синтеза на основе поведенческой RTL-модели, составляет не более 10% [4].

Дальнейшее проектирование полностью соответствует традиционному маршруту.

Отметим, что средства ESL позволяют:

1. Осуществить «интеллектуальное» сегментирование ПЛИС и автоматическое преобразование программных функций в эквивалентные функции аппаратных средств.

2. Опробовать значительно больше вариантов реализации системы, чем при традиционном проектировании, экспериментально проверить различные стратегии сегментирования разрабатываемой системы на программные и аппаратные компоненты и выполнить быстрый сравнительный анализ вариантов с целью достижения оптимального компромисса между производительностью и размерами ПЛИС. В результате зачастую удается получить более высокие характеристики системы за время, меньшее, чем при использовании традиционных методов преобразования в RTL-код.

Работа на более высоком уровне абстракции позволяет разработчикам, имеющим опыт программирования на универсальных и проблемно-ориентированных языках программирования, быстрее воплотить свои идеи в аппаратуре без привлечения опытного конструктора аппаратных средств. Наибольший эффект от применения ESL достигается при реализации приложений с большим объемом вычислений и множеством циклов.

Несмотря на преимущества ESL перед традиционными подходами, нерешенным остается ряд проблем, характерных для обеих методик. В частности, для получения оптимального по соотношению производительность/занимаемое на кристалле место требуется выполнить анализ множества вариантов реализации проекта. При этом сложно заранее предсказать такие характеристики системы, как занимаемую площадь на кристалле, энергопотребление и т.д. На каждой стадии проектирования требуется обязательная верификация системы, то есть необходима разработка и генерация пакета тестов для каждого этапа. При этом нет гарантии, что будут рассмотрены все возможные варианты реализации и выбран оптимальный из них.

В процессе работы над проектом сложно вносить изменения. На алгоритмическом уровне модификации просты, но при переходе на RTL описание вновь может потребоваться перекомпоновка всего проекта и его повторная верификация.

При проектировании на уровнях ниже RTL остается нерешенным весь комплекс проблем традиционного проектирования связанных с жесткой привязкой к целевой платформе реализации.

На основании вышеизложенного отметим, что при проектировании сложных однокристальных систем требуется новый, архитектурно независимый подход, с одной стороны, обеспечивающий максимальное абстрагирование исходных алгоритмов от архитектуры целевого кристалла, с другой стороны, реализующий механизм перехода на RTL уровень с параллельной верификацией при проектировании без возврата к предыдущим уровням создания проекта.

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 ПЛИС без возврата к предыдущим уровням иерархии.

В связи с этим актуальна задача поиска таких методов описания параллелизма исходных алгоритмов, которые в дальнейшем могли бы только «сжиматься» с учетом специфики ресурсных ограничений.

Языки описания аппаратуры на нижнем уровне – это языки, ориентированные на описание графа, состоящего из узлов-элементов, обменивающихся данными и связей между ними, обеспечивающих перетекание этих данных. Поэтому и на верхнем уровне целесообразно формировать описание на уровне потоков данных, а не использовать императивный стиль. Следовательно, при реализации данного подхода необходимо ориентироваться не на традиционное программирование с применением императивных языков, а использования функционально-потоковое представление программ. Кроме того, верификацию системных алгоритмов необходимо выполнять на верхнем, формальном, уровне абстракции, в то время как сам проект необходимо верифицировать на функционально-потоковом уровне, и затем, автоматически генерировать RTL-код и из RTL-кода посредством инструментов RTL-синтеза получать готовое вентильное описание. Полностью исключать стадию RTL-проектирования из маршрута нельзя по следующим причинам. Во-первых, подключаемые блоки интеллектуальной собственности (Intellectual Property, IP-блоки) предоставляются в виде RTL-описания. Во-вторых, дополнительная верификация и отладка проекта существующими средствами идет намного быстрее на RTL уровне, нежели на вентильном. В-третьих, корректировка проекта на вентильном уровне намного сложнее, чем на уровне RTL. И наконец, анализ потребляемой мощности, предварительный анализ трассы и встраиваемые модули тестовой логики сегодня доступны только на RTL-уровне.

В связи с изложенным представляет интерес замена существующих методов высокоуровневого (прикладного) описания решаемой задачи на языковые и инструментальные средства, обеспечивающие поддержку архитектурно-независимого параллельного программирования.

Для поддержки архитектурно-независимой разработки параллельных программ в [6] предложена функционально-потоковая парадигма, положенная в основу языка Пифагор. Предлагаемый язык обеспечивает описание разнообразных алгоритмов и имеет развитые средства по представлению параллелизма программ и данных. Использование неявного управления по готовности данных позволяет описывать максимальный параллелизм решаемой задачи. Архитектурная независимость используемой в языке модели вычислений базируется на следующих принципах.

Считается, что виртуальная машина, предназначенная для выполнения функционально-потоковых параллельных программ, имеет неограниченные вычислительные ресурсы. Это позволяет выделять для каждой операции новый вычислительный ресурс.

Организация циклов обеспечивается за счет рекурсии. Это позволяет избавиться от ресурсных ограничений, присущих циклическим фрагментам. Наличие асинхронных списков в сочетании с рекурсией позволяет формировать алгоритмы с динамически изменяемым параллелизмом.

Вместе с тем следует отметить, что используемые в языке Пифагор конструкции должны, в конечном итоге, отображаться на архитектуры реальных параллельных вычислительных систем. Формирование подобного отображения является нетривиальной задачей. Аналогичные проблемы возникают и при преобразовании функционально-потоковых параллельных программ в топологические структуры ПЛИС. Решение данной задачи позволит повысить эффективность разработки заказных интегральных схем.

Для решения целевой задачи осуществляется разработка программы на подмножестве функционально-потокового языка параллельного программирования Пифагор. Подмножество языка выбирается в соответствии со спецификой задач, решаемых на ПЛИС и особенностями реализации аппаратно-программного обеспечения на кристалле.

Применение данного языка позволяет повысить эффективность отладки и верификации за счет ресурсно-независимого описания параллелизма [7]. При этом возможно использование инструментальных средств, разработанных как для обычных компьютеров, так и для параллельных вычислительных систем с различной архитектурой.

В ходе трансляции программы строятся промежуточные структуры данных, которые в дальнейшем могут быть связаны с различными блоками, реализуемыми на ПЛИС.

Среди таких промежуточных структур следует выделить:

- реверсивный информационный граф, описывающий зависимости операций программы от формируемых аргументов;
- управляющий граф, определяющий передачу управления от одной выполняемой операции другой (в зависимости от особенностей разработанной программы, передачи управляющих сигналов могут происходить параллельно);
- слой данных, отвечающий за хранение значений;
- слой автоматов, реагирующих на поступающие значения и изменяющих состояния выполняемых операций.

Порождаемые в ходе трансляции промежуточные структуры могут использоваться для формирования топологии интегральных схем, создание которых может происходить по различным траекториями и вполне согласуется с существующими маршрутами. В частности, на первых этапах вполне возможно преобразование функционально-потоковых параллельных программ в программы, написанные на языках программирования, поддерживающих ESL, что позволит достаточно быстро осуществить внедрение предлагаемой технологии за счет использования применяемых инструментальных средств, библиотек и маршрутов проектирования СБИС. Помимо этого также возможен непосредственный переход от функционально-потоковых параллельных программ к архитектурно-зависимым представлениям на соответствующих языках.

В дальнейшем возможно движение по двум направлениям. С одной стороны можно ориентироваться на реализацию окончательной топологии с использованием наработанных процессорных модулей. Также возможен вариант, когда для различных промежуточных структур функционально-потоковой модели вычислений разрабатываются свои модули, из которых и будет складываться топология кристалла.

Функционально-потоковая модель параллельных вычислений может быть поддержана следующими компонентами:

- вычислительными модулями, обеспечивающими выполнение операций в узлах информационного графа (посредством этих элементов обеспечивается выполнение арифметических и логических операций, манипуляции со структурами данных);
- модулями, предназначенными для формирования и передачи управляющих сигналов между узлами информационного графа программы;
- модулями, реализующими автоматы управления состояниями для узлов информационного графа различного типа.

Анализ темпа поступления данных позволяет на уровне функционально-потоковой параллельной программы определить количество вычислительных узлов (сумматоров, умножителей и т.д.), которое позволит эффективно справляться с вычислениями в реальном времени.

Заключение

В работе предложен новый подход к формированию топологии ПЛИС, основанный на использовании функционально-потоковой парадигмы, обеспечивающей архитектурно независимое описание реализуемых параллельных алгоритмов. Такой подход позволяет осуществлять описание системы на алгоритмическом уровне без привязки к конкретной реализации и разделения на программную и аппаратную составляющие, но с учетом изначального параллелизма решаемой задачи. Наряду с отсутствием явного описания параллелизма облегчается отладка и верификация параллельных программ. Разработка инструментальных средств, обеспечивающих непосредственную трансляцию с рассматриваемого языка в языковые средства, используемые при проектировании ПЛИС, позволит повысить эффективность процесса разработки.

Список литературы

[1] *Рабаи Жан М.., Чандракасан А., Николич Б.* Цифровые интегральные схемы. Методология проектирования. М.: ООО «ИД Вильямс», 2007. 912 с.

[2] *Колесников Е.И.* // Мат. VI Всероссийской межвузовской конференции молодых ученых. СПб: ИТМО, 2009. С. 175.

[3] *Шагурин И.А., Канышев В.И.* // Chip-News. Инженерная микроэлектроника. 2006. № 9(112). С. 51.

[4] *Непомнящий О.В., Алекминский С.Ю.* // Нано- и микросистемная техника. М.: Новые технологии. 2010. №9(122). С. 4.

[5] *Densmore D., Passerone R., Sangiovanni-Vincentelli A.* // IEEE Design and Test of Computers, 2006. Vol. 23. No. 5. P. 359.

[6] *Легалов А. И.* // ИВТ СОРАН, Вычислительные технологии. 2005. № 1 (10). С. 71.

[7] *Легалов А. И., Сиротинина Н. Ю., Удалова Ю. В.* // Journal of Siberian Federal University. Engineering & Technologies. 2011. 2 (2011 4). С. 213.