Порождение модели микроархитектуры из модели макроархитектуры

М.Н.Смирнов
smn@tepkom.ru
Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

Аннотация

В статье рассматривается способ автоматической генерации модели микроархитектуры по описанию макроархитектуры, реализованный в рамках проекта PADLA, посвященного технологии разработки программно-аппаратных систем. При генерации на вход поступает формальное описание макроархитектуры целевого процессора и порождается описание микроархитектуры целевого процессора в виде файлов на языке VHDL [7]. Процесс генерации является настраиваемым и не требующим вмешательства пользователя. Полученное в процессе генерации описание микроархитектуры предполагается использовать для последующей организации проекта в любом из средств контрукторской разработки с целью аппаратной реализации устройства.

Введение

Возможность автоматической генерации микроархитектуры в рамках задачи разработки программно-аппаратных систем позволяет получить практическую реализацию разрабатываемого устройства в виде представления его архитектуры на одном из языков описания цифровых устройств. С помощью такого представления становится возможной симуляция поведения устройства на аппаратном уровне.

Для решения вопроса автоматической генерации описания микроархитектуры в процессе создания нового устройства, рас-

сматриваемого в данной статье, предполагается использование методологии быстрого прототипирования, описанной в статье [1]. Этой методологией предусматриваются следующие стадии разработки:

- стадия извлечения формального описания макроархитектуры (подробно это понятие обсуждается ниже) устройства из программы на языке высокого уровня;
- стадия генерации средств разработки (ассемблера, дизассемблера, симулятора, линкера, загрузчика и т.д.) по формальному описанию макроархитектуры;
- стадия генерации и оптимизации микроархитектуры (порождение спецификации на одном из языков описания цифровых устройств, моделирование в средствах конструкторской разработки и т.д.).

Последующее изложение построено следующим образом: в разделе "Обзор других работ по данной тематике" будет дано сравнение с другими работами на данную тему; далее, прежде чем говорить о генерации описания микроархитектуры, в разделе "Описание макроархитектуры с точки зрения микроархитектуры" будет кратко освещен используемый язык описания макроархитектуры и особенно те его элементы, которые важны с точки зрения микроархитектуры. В разделе "Генерация микроархитектуры по описанию макроархитектуры" будет рассматриваться собственно решение задачи автоматической генерации модели микроархитектуры по описанию макроархитектуры.

1 Обзор других работ по данной тематике

Различные аспекты задачи автоматической генерации микроархитектуры изложены во множестве работ, рассматривающих разработку программно-аппаратных систем. В качестве источника информации о языках описания архитектур можно упомянуть [8], где проводится краткий сравнительный анализ распространенных языков описания машинных архитектур (Architecture Description Language, ADL). В обзоре выделено три группы языков: языки структурного уровня, позволяющие описывать компоненты архитектуры и связи между ними; языки

поведенческого уровня, предназначенные для описания синтаксиса, машинного представления и семантики инструкций; смешанные языки, объединяющие возможности первых двух классов. Используемый нами язык PADLA относится к третьей группе по этой классификации.

Среди языков описания архитектур нужно упомянуть язык nML [5], который является наиболее близким к используемому нами языку PADLA. В работе [6] обсуждаются проблемы, возникающие при попытке описания на языке nML таких сложных аспектов архитектуры процессоров, как отложенные переходы/присваивания, прерывания, работа со счетчиком инструкций. В работе [4] рассматривается генерация аппаратной модели на основе описания на языке nML. Описывается подход, применяющийся при трансформировании описания набора инструкций на языке nML в аппаратную модель. Данный подход опирается на эффективное представление данных и концепцию унификации поведенческой и кодирующей моделей.

Дальнейшим развитием языка nML стал язык Sim-nML [9]. В данное время ведутся работы, посвященные генерации описания микроархитектуры на языке Verilog по спецификациям на этом языке, но никакой доступной информации по этому поводу обнаружить не удалось.

Следует также упомянуть такой подход к созданию архитектуры, как параметризации некоторой шаблонной архитектуры. Для реализации такого подхода существуют средства формального описания, например используемый в проекте Trimaran язык HMDES и его расширения для описания набора инструкций [3]. Такой подход, конечно, не допускает большой свободы в определении компонент архитектуры и набора инструкций, но он менее ресурсоемок и проще в реализации.

2 Описание макроархитектуры с точки зрения микроархитектуры

Под *макроархитектурой* понимается представление целевого устройства с точки зрения разработчика компилятора. В рамках данной статьи важно, что описание макроархитектуры содержит информацию, касающуюся семантики каждой из инструк-

ций, входящих в систему команд устройства. Эта информация ложится в основу описания микроархитектуры.

В состав макроархитектуры входят ресурсы (память, регистры) и набор команд различных форматов. Описание макроархитектуры представляет собой текстовый файл, разбитый на секции. Структура секций отражает приведённую выше классификацию.

В секции `resource` описаны такие ресурсы, как регистры и различные виды памяти. Каждому ресурсу сопутствуют семантические атрибуты, одним из которых является разрядность. Отдельно выделен ресурс, содержащий код программы. Он помечен специальным признаком.

В секции `type` описаны возможные типы параметров инструкций, которые различаются своей разрядностью.

В секции `format` содержится описание форматов инструкций. Тут могут содержаться стандартные форматы для наборов однотипных команд (например таких, как сложение и вычитание, или различные виды переходов и т.д.).

В секции `asm` содержится набор правил, позволяющих группировать элементы описания синтаксиса инструкций. Это позволяет устанавливать единый синтаксис для управляемых общей семантикой элементов синтаксиса ассемблера, например для режимов адресации.

В последней секции содержится описание машинных инструкций. Инструкция в данном описании представлена своим форматом, в котором может быть указана семантика, синтаксис и бинарное представление.

Описание бинарного представления инструкции требуется для организации дешифрации инструкций в описании микроархитектуры. Описание семантики инструкции используется для генерации описания микроархитектуры устройства, выполняющего данную инструкцию. Из совокупности сгенерированных описаний всех перечисленных инструкций, а также из блока дешифрации инструкций, описания ресурсов и описания интерфейсов и состоит генерируемая микроархитектура.

На рисунке 1 приведён пример описания “игрушечной” макроархитектуры. Архитектура, описанная в примере, имеет следующие компоненты: 512 регистров размерностью 3 байта, адресация которых производится 9-битным адресом; указатель те-

```
resource
  r(0..511):24;
  PC:16 ip (r);

type
  opnd: 8;
  reg: 9;
  imm(opnd);
  arif(reg, reg);

instruction j:
  imm (X) ->
    action {PC <- (PC + (sext(X,10)))}
    code   {0b0000010000000000  X}
    repr   {"l.j", X}

instruction addinsn:
  arith (i, j) ->
    action {PC <- (PC + 1:16)[:16]; r(i) <- (r(i) + r(j))[:24]}
    code   {0b000010 i j}
    repr   {"l.add" , i, j}
```

Рис. 1: Пример описания архитектуры

кущей команды в памяти; инструкцию `j` (безусловный переход), имеющую один (непосредственно адресуемый) операнд (смещение); инструкцию `arith` (сложение двух регистров), имеющую 2 операнда (номера регистров по 9 бит).

Грамматику и подробные комментарии к описанию макроархитектуры можно найти в [2].

3 Генерация микроархитектуры по описанию макроархитектуры

При генерации микроархитектуры решается задача получения описания проектируемого устройства по описанию его макроархитектуры. Входным является внутреннее представление макроархитектуры, которое содержит описание семантики инструкций, описание ресурсов системы и бинарное представление инструкций.

При генерации микроархитектуры по описанию макроархи-

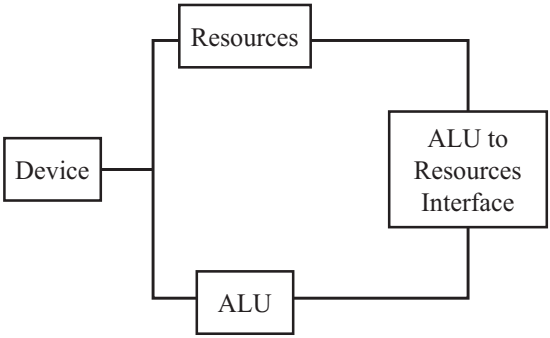

Рис. 2: Структура устройства (ресурсы вместе с вычислительными блоками)

тектуры решается нетривиальная задача перехода от описания устройства на уровне системы команд к описанию устройства на уровне функциональных блоков, реализующих данную систему команд. Сложность задачи заключается в том, что в описании макроархитектуры информация о функциональных блоках устройства содержится в крайне урезанном виде или не содержится вовсе. Всю информацию о функциональных блоках приходится извлекать из описания макроархитектуры в ходе процедуры ее анализа. Такой подход потребовал формализации определенного промежуточного представления устройства, содержащего результаты анализа, который позднее будут использованы для генерации описания микроархитектуры. Данное промежуточное представление представляет собой иерархическую структуру, отображающую на верхних уровнях иерархии разбиение устройства на функциональные блоки и, на нижних уровнях иерархии, функции, выполняемые каждым блоком.

Результаты генерации зависят, кроме описания макроархитектуры, еще и от параметров генерации. В частности, можно выбрать, где размещать ресурсы устройства — в одной связке с вычислительными блоками или нет. Также можно указать, требуется ли генерировать тестовый интерфейс для устройства. Тестовый интерфейс требуется для моделирования поведения устройства с помощью любого из средств контрукторской разработки.

Извлечение содержащейся в описании макроархитектуры ин-

формации происходит в несколько проходов. При первом проходе извлекается информация об организации микроархитектуры устройства на верхнем уровне, на основе которой определяется декомпозиция устройства на отдельные функциональные блоки. Эта информация отображается в виде дерева структуры устройства. В случае размещения ресурсов вместе с вычислительными блоками дерево структуры устройства выглядит так, как показано на рисунке 2. В случае размещения ресурсов отдельно от вычислительных блоков и добавлении тестового интерфейса дерево структуры устройства выглядит как на рисунке 3.

При втором проходе происходит извлечение информации, опредеделяющей микроархитектуру устройства на нижнем уровне. По результатам этих двух проходов происходит генерация микроархитектуры всего устройства.

В дальнейшем изложении структура блоков будет описываться с точки зрения наиболее полного набора генерируемых блоков (ресурсы отдельно от вычислительных блоков и есть тестовый интерфейс).

3.1 Предварительный анализ макроархитектуры с целью определения свойств микроархитектуры

При предварительном анализе макроархитектуры, осуществляющемся при первом просмотре описания макроархитектуры, происходит извлечение информации о таких основополагающих свойствах архитектуры устройства, как:

- длина командного слова устройства;
- количество, размер и размещение регистров;
- количество, размер, типы и размещение памятей;
- интерфейсы между вычислительными блоками и блоком ресурсов, если он есть;
- интерфейсы между вычислительными блоками и блоком тестового интерфейса, если он есть.

Исходной информацией для определения длины командного слова служит формат команды, который, кроме всего прочего, определяет и ее длину. Размер максимальной по длине команды

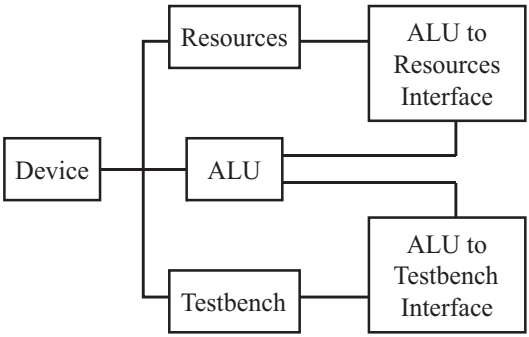

Рис. 3: Структура устройства (ресурсы отдельно от вычислительных блоков, есть тестовый интерфейс)

устройства и является длиной командного слова. В приведенном выше примере макроархитектуры устройства длина командного слова составляет 8 бит, такую длину имеют все его команды.

Информация, касающаяся регистров устройства, извлекается из описания ресурсов, содержащегося в описании макроархитектуры. В описании ресурсов непосредственно присутствуют размер и количество, размещение же указывается параметром при генерации. Так, в примере "игрушечной" макроархитектуры ресурс `r` описывает блок из 8 регистров, каждый размерностью в один байт. А ресурс `PSW` описывает регистр размером в байт, который имеет вид структуры из 6 полей.

Что касается памятей устройства, то их размер и количество также извлекаются из описания ресурсов. Что касается типа, то в данный момент под типом понимается только количество портов чтения и записи, которое имеет тот или иной тип памяти. Количество портов памяти определяется числом возможных одновременных (в одном машинном такте) обращений к ней для чтения или записи, причем операции чтения и записи рассматриваются независимо. Информация о количестве одновременных операций чтения или записи для каждой из памятей устройства в явном виде в описании макроархитектуры не содержится и ее приходится извлекать из семантики команд устройства. В примере "игрушечной" макроархитектуры ресурс `memory` является памятью из 256 слов по байту, имеющей один порт для чтения и один порт для записи.

На основе информации о виде и размещении ресурсов определяются интерфейсы между блоками, содержащими ресурсы (блоки ресурсов и тестового интерфейса), и блоками, их использующими (вычислительные блоки).

3.2 Микроархитектура устройства на верхнем уровне

В зависимости от параметров генерации описание микроархитектуры может состоять как из одного блока (рис. 2), так и из нескольких блоков (вычислительного блока, блока ресурсов, блока интерфейсов между блоками ресурсов и устройства, блока отладочного интерфейса устройства и блока интерфейсов между блоками отладочного интерфейса устройства и самого устройства) (рис. 3).

Разделение на блоки зависит от выбранной модели описания устройства, которая предполагает наличие таких компонентов, как процессор и память. Кроме этих компонент возможна генерация еще и тестового интерфейса.

Вычислительный блок представляет собой собственно процессор устройства. Он декодирует и выполняет команды, подкачивая из памяти командные слова и операнды. Этот блок может содержать внутри себя регистры любого размера, если это задано параметрами при генерации. Другие свойства этого блока берутся из макроархитектуры и не параметризуются при генерации.

Блок ресурсов содержит регистры и память, определенные при генерации как внешние. В нем производится декодирование адресов, приходящих из вычислительного блока и обработка запросов на чтение и запись.

Эти два вышеописанных блока связываются вместе в одно устройство блоком интерфейсов между ними, который стоит выше в иерархии блоков. Блок интерфейсов описывает связи между данными блоками.

Блок тестового интерфейса подобен блоку ресурсов, который он подменяет в случае моделирования устройства на инструментальной машине. Этот блок осуществляет те же самые функции, что и блок ресурсов, с тем отличием, что имеется возможность первоначального задания содержимого любого расположенного в нем ресурса из соответствующего файла.

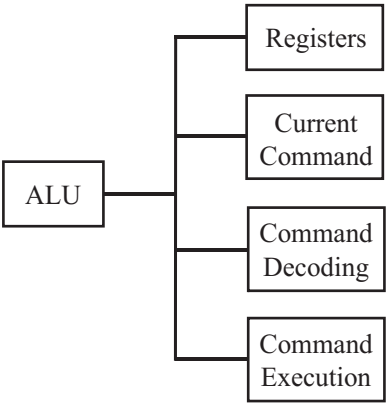

Рис. 4: Измененная структура программы на VHDL: блок вычислений

Интерфейс между блоком тестового интерфейса и вычислительным блоком обеспечивается соответствующим блоком интефейсов.

3.3 Микроархитектура устройства на нижнем уровне

Выше уже перечислялись основные блоки, из которых состоит описание микроархитектура устройства. Каждый из таких блоков, в свою очередь, состоит из ряда подблоков, или процессов, каждый из которых выполняет одну из задач, возложенных на блок.

Разбиение блока на процессы производится на основе выбранной модели описания цифровых устройств с программным управление и не зависит от макроархитектуры устройства. Макроархитектура влияет только на интерфейсы и содержимое процессов.

На приведенной (рис. 4) схеме видно, что вычислительный блок содержит четыре основных процесса. Первый из них описывает регистры и память, расположенные внутри него. Далее следует процесс, отвечающий за выборку кода команды из памяти и формирование командного слова. Следующий процесс отвечает за декодирование инструкций, выборку из памяти операндов команд и выполнение команд. Последний процесс занимается тем, что формирует адреса и запросы к памяти для чте-

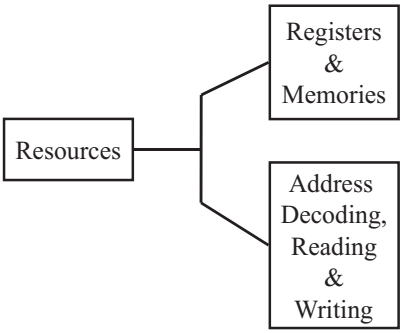

Рис. 5: Измененная структура программы на VHDL: блок ресурсов

ния или записи операндов. В случае размещения ресурсов внутри вычислительного блока формирование адресов и запросов к ресурсам не выносится в отдельный процесс, а находится в процессе, отвечающем за выполнение команд.

Процесс дешифрации команд основывается на бинарном представлении команды. Каждая команда имеет фиксированные поля (код команды) и переменные поля, относящиеся к операндам команды. При генерации дешифратора для каждой команды формируется список ее фиксированных полей и их значений. Дешифратор производит перебор команд, сопоставляя значения их фиксированных полей с содержимым текущего командного слова. Перебор организован таким образом, что сперва проверяются команды с меньшим суммарным размером фиксированных полей. Это позволяет избежать неправильного декодирования в случае команд разной длины.

На рисунке 5 изображен блок ресурсов, содержащий два процесса. Первый из них отвечает за описание регистров и памятей, расположенных в данном блоке, а второй — за декодирование адресов, по которым производится обращение из вычислительного блока и выполнения чтения или записи по этим адресам.

Подобно блоку ресурсов организован и блок отладочного интерфейса устройства, схема которого приведена на рисунке 6. Единственное его отличие в том, что во втором процессе производится обращение не к реальным ресурсам, а к их модели, как это требуется при отладке.

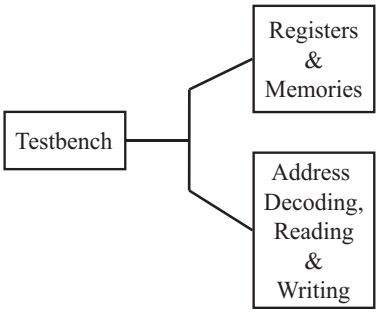

Рис. 6: Измененная структура программы на VHDL: блок отладочного интерфейса

Существует два блока интерфейсов. Первый отвечает за связь между вычислительным блоком и блоком ресурсов, а второй — за связь вычислительного блока с блоком отладочного интерфейса. Оба этих блока интерфейсов практически одинаковы, что вызвано идентичностью интерфейсов блоков ресурсов и тестового интерфейса. Оба блока интерфейсов содержат описание интерфейса вычислительного блока, описание интерфейса блока ресурсов или блока тестового интерфейса, а также связи между этими двумя интерфейсами. Эти связи и определяют реальное соединение вычислительного блока с одним из двух блоков, ресурсов или отладочного интерфейса. Кроме того, как и любой другой блок, интерфейсные блоки имеют свой собственный интерфейс, описанный в них.

Заключение

В данной статье рассматривался метод генерации описания микроархитектуры из описания макроархитектуры, реализованный в рамках проекта PADLA, посвященного разработке программно-аппаратных систем. Как говорилось ранее, на данной стадии развития метода поддерживается генерация только RISC-подобных архитектур с одним потоком инструкций и регистровой памятью. Одним из наиболее приоритетных путей развития метода является путь исследования возможностей поддержки разных типов памяти, а также генерации более сложных

архитектур, таких, как:

- CISC-подобные архитектуры;
- векторные архитектуры;
- конвейерные архитектуры;
- скалярные архитектуры;
- VLIW-подобные архитектуры.

Предполагается вставить в метод поддержку следующих свойств архитектуры проектируемого устройства:

- развитие возможности порождать интерфейсы для работы с различными типами памяти (а не только регистровой);
- поддержка более сложных, чем RISC, инструкций;
- параллельность операций внутри одной инструкции;
- поддержка параллельного выполнения одинаковых, а затем и разнородных инструкций;
- поддержка конвейерного выполнения одинаковых, а затем и разнородных инструкций;
- поддержка архитектур с микропрограммным управлением.

В настоящее время идет работа в этих направлениях.

Список литературы

[1] Булычев Д.Ю. Разработка программно-аппаратных систем на основе описания макроархитектуры // Наст. сборник. — С. 7-22.

[2] Булычев Д.Ю. Язык описания макроархитектуры для технологии совместной программо-аппаратной разработки // Наст. сборник. — С. 23-48.

[3] Aditya S., Mahlke S., Rau R. Code Size Minimization and Retargetable Assembly for Custom EPIC and VLIW Instruction Formats // ACM Transactions on Design Automation of Electronic Systems. — 2000. — P. 752-773.

[4] Fauth A., Freericks M., Knoll A. Generation of Hardware Machine Models from Instruction Set Descriptions. — http://citeseer.nj.nec.com/fauth93generation.html.

[5] Fauth A., van Praet J., Freericks M. Describing Instruction Set Processors Using nML. // Proceedings of European Design and Test Conference. — 1995. — P. 503-507.

[6] Hartoog M. e.a. Generation of Software Tools from Processor Descriptions for Hardware/Software Codesign // Design and Automation Conference. — 1997. — P. 303-306.

[7] IEEE Standard VHDL Language reference Manual: IEEE Std 1076-2002. — IEEE Standards, 2002. — 306 p.

[8] Quin W. A Survey of Architecture Description Languages. http://campuscgi.princeton.edu/~znhuang/mescal/ppt/27.pdf.

[9] Rajesh V. A Generic Approach To Performance Modeling and its Aplication to Simulator Generator: Master Thesis. — Kanpur: 1998.

[10] Ramsey N., Fernandez M.L. The New Jersey Machine-Code Toolkit. // USENIX Technical Conference. — 1995. — P. 289-302.