ГЛАВА 24

СРЕДСТВА ПРОЕКТИРОВАНИЯ НЕЗАВИСИМЫХ РАЗРАБОТЧИКОВ

Введение

Когда дело доходит до средств проектирования — систем логического моделирования, средств синтеза и так далее, мы обычно обращаемся к большим компаниям, разрабатывающим полный набор САПР электронных систем, а также к небольшим фирмам, специализирующимся на средствах для отдельных этапов проектирования, или к производителям самих ПЛИС.

Однако не следует забывать о тех, кто работает с продуктами с открытым исходным кодом (смотрите также гл. 25). Более того, небольшие консалтинговые компании часто тратят довольно много времени и усилий на создание собственных средств, предназначенных для их внутренних проектов. Порой эти инструменты становятся настолько полезными, что они становятся доступны всему миру. В этой главе мы кратко рассмотрим такие средства.

ParaCore Architect

Компания Dillon Engineering (www.dilloneng.com) предлагает разнообразные услуги по разработке определяемых заказчиком алгоритмов ЦОС на основе ПЛИС, а также приложений по широкополосной цифровой обработке сигналов и изображений в реальном масштабе времени.

К концу 90-х инженеры этой компании осознали, что они постоянно заново изобретают и реализовывают такие средства, как библиотеки функций с плавающей точкой, ядра свёртки, процессоры БПФ и так далее. Поэтому для облегчения своей жизни они разработали средство под названием ParaCore Architect™, которое представляет собой набор блоков интеллектуальной собственности.

Процесс проектирования начинается с написания на языке программирования Python исходных файлов, содержащих высоко параметризованные описания устройства на очень высоком уровне абстракции (более подробно язык Python будет рассматриваться в гл. 25). ParaCore Architect обрабатывает это описание, комбинирует его со значениями пользовательских параметров, и затем генерирует эквивалентное представление устройства на языке HDL, потактную C/C++ модель для ускорения процесса моделирования, и соответствующий набор тестов (**Рис. 24.1**).

Полученный таким образом RTL-код гарантировано подойдет для использования в любой среде моделирования или средствах синтеза, поэтому нет необходимости запускать какую-либо программу для его проверки. Вся прелесть этого высокопараметризованного описания заключается в том, что его чрезвычайно легко использовать в новых приложениях или других устройствах.

1967 г. *Америка. Джек Килби (Jack Kilby), работая в компании Texas Instruments, изобрёл первый ручной электронный калькулятор.*

Рис. 24.1. ParaCore Architect генерирует C/C++ и RTL-представления, а также набор тестов

Реализация модулей для обработки данных с плавающей точкой

В качестве простого примера использования ParaCore Architect можно рассмотреть некоторые ПЛИС, в которые встраивают микропроцессорные ядра. К сожалению, эти ядра, как правило, не комплектуются соответствующим модулем для выполнения операций с плавающей точкой. Это означает, что если проектировщики захотят выполнять операции с плавающей точкой в соответствующем проекте, то сделать это они смогут программным (что довольно медленно) или аппаратным путём. В последнем случае на реализацию подобного решения понадобится затратить много усилий, которые лучше было бы потратить на разработку хорошей функциональной части устройства.

Для решения этой проблемы можно использовать одно из конструктивных описаний ParaCore Architect, которое позволит сгенерировать соответствующее ядро обработки данных с плавающей точкой. При этом могут использоваться различные параметры для определения необходимой точности порядка и мантиссы чисел, количества ступеней используемого конвейера, необходимости обработки особых случаев чисел с плавающей точкой, например, бесконечности (для некоторых приложений эти особые случаи не требуются), типа используемого микропроцессорного ядра и т. д.

Реализация функций БПФ

Хорошим примером демонстрации мощности пакета ParaCore Architect может служить его использование для генерации ядра, выполняющего функции быстрого преобразования Фурье (БПФ). Наименьший вычислительный элемент, используемый для создания функций БПФ, называется «**бабочка**», который состоит из комплексного умножения, комплексного сложения и комплексного вычитания (**Рис. 24.2**).

В свою очередь, операция комплексного умножения требует четырех простых умножителей и двух простых сумматоров, а комплексное сложение и комплексное вычитание требуют по два простых сумматора. Таким образом, для реализации каждой «бабочки» потребуется четыре простых умножителя и шесть простых сумматоров.

Одно реальное приложение обработки изображений, использующее подобное ядро, должно обрабатывать двухмерный массив размером 2048×2048 точек БПФ, обновляющийся с частотой 120 кадров в

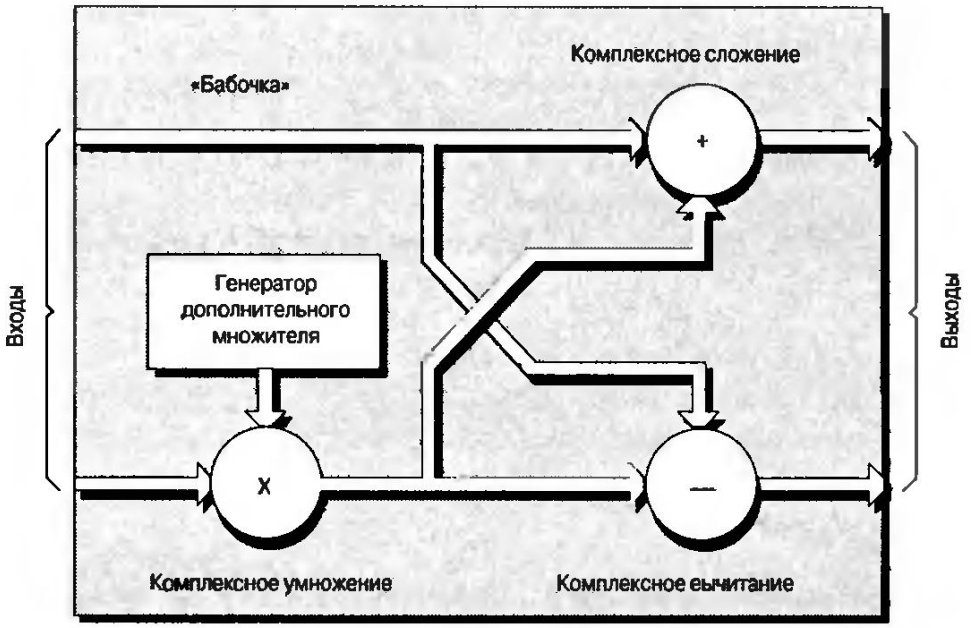

Рис. 24.2. «Бабочка» — наименьший вычислительный элемент БПФ

секунду. Для обработки одного столбца из 2048 пикселей понадобится 11256 «бабочек», организованных в одиннадцать рангов, где выходы «бабочек» первого ранга поступают на вход второго ранга и т. д. Таким образом, для обработки одного столбца потребуется 45025 простых умножителей и 67536 простых сумматоров. Чтобы произвести БПФ для всего кадра размером 2048×2048 точек, необходимо повторить этот процесс для каждого из 2048 рядов, формирующих кадр. Это значит, что для достижения частоты обновления в 120 кадров в секунду обработка каждого ряда должна завершаться за 4 микросекунды. (Это значит, что на каждое простое умножение отводится 90 пикосекунд, а на каждое простое сложение — 60 пикосекунд.)

Давайте рассмотрим работу 11256 «бабочек», требуемых для обработки 2048 точек БПФ. Если бы время выполнения этой операции не было бы решающим фактором, то для её решения можно было бы использовать сравнительно небольшие ПЛИС, например Xilinx Virtex-II XC2V40 с четырьмя блоками умножения для создания одной «бабочки» (четыре простых умножителя и шесть простых сумматоров), и затем выполнить все операции «бабочки» с помощью этой функции. В итоге для обработки каждых 2048 точек БПФ уходило бы 90 микросекунд. Хотя это очень хорошее значение, но оно не подходит для приложения обработки изображений, в котором на эту операцию отводится всего 4 микросекунды.

Самый легкий способ увеличения скорости работы этого алгоритма заключается в увеличении количества «бабочек» в аппаратной части и в выполнении большего количества параллельных вычислений. При использовании микросхемы Xilinx XC2V6000, которая содержит шесть миллионов системных логических элементов, 144 18-битных умножителя и 144 блока оперативной памяти объёмом 18 килобит, можно обрабатывать все 2048×2048 точек со скоростью 120 кадров в секунду.

Всё дело заключается в том, чтобы получить все эти устройства потребуется установить только один параметр в пакете ParaCore Architect, который определяет количество «бабочек», подлежащих реализации в аппаратной части устройства.

В другом примере, если мы решили изменить длину БПФ с 2048 на 1024 точки, тогда нам необходимо поменять значение только одного параметра, который вызовет реконфигурацию всех необходимых

1969 г. *«Человек на Луне» передал первый радиосигнал.*

структур, в том числе произведёт изменение размеров оперативной памяти, используемой для хранения внутренних результатов. Также может использоваться ещё один параметр для выбора формата представления чисел — с фиксированной точкой или с плавающей точкой (в последнем случае задействуются ещё два параметра для определения размера экспоненты и мантиссы).

В начале 2002 сотрудники Dillon Engineering использовали пакет ParaCore Architect для создания, возможно, самого быстрого в мире (на тот момент) процессора БПФ. Этот процессор впоследствии нашел применение во многих областях, например, в программе SETI для поиска внеземных цивилизаций, где он использовался для обработки огромного количества данных, поступающих с радиотелескопов!

Web-интерфейс

Пожалуй, самым приятным моментом является то, что компания Dillon Engineering открыла для своих клиентов доступ к пакету ParaCore Architect через сеть Интернет. При разработке функций, таких как БПФ и ей подобных, часто хочется достичь какого-либо компромисса, например, узнать, сколько надо сохранить бит для одной точки. Теперь клиенты компании Dillon Engineering могут посетить сайт www.dilloneng.com, выбрать интересующий их тип ядра, задать параметры и нажать кнопку «Go» для создания эквивалентных C/C++ и HDL-моделей, а также набора соответствующих тестов.

Язык проектирования системы Confluence

Как и большинство разработчиков, меня бросает в дрожь, когда я сталкиваюсь с ещё одним языком программирования или разработки аппаратных средств, но компания Launchbird Design Systems (www.launchbird.com) предложила свой язык проектирования систем под названием *Confluence*, а также соответствующий компилятор *Confluence Compiler*, который стоит рассмотреть более подробно.

Трудно охватить вниманием все многочисленные грани языка Confluence, но мы все же попытаемся это сделать. Во-первых, Confluence является невероятно компактным языком, который может использоваться для создания описаний как аппаратных средств, так и встраиваемого программного обеспечения. При работе с аппаратной частью эти описания обрабатываются компилятором Confluence, который генерирует соответствующее RTL-описание на языках VHDL или Verilog (**Рис. 24.3**).

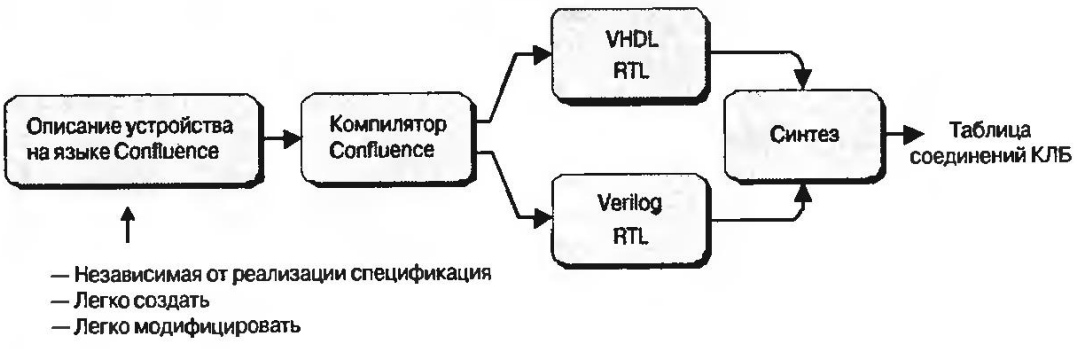

Рис. 24.3. Использование компилятора языка Confluence

Чтобы лучше понять этот процесс, можно представить, что один из языков HDL (например, VHDL или Verilog) используется для описа-

ния отдельных схем, а язык Confluence — для описания алгоритмов, которые могут сгенерировать целый класс схем. Смысл такого подхода заключается в том, что с помощью Confluence вы можете выразить больше, затрачивая на это существенно меньше строк кода (таким образом можно уменьшить исходный код от 3-х до 10-ти раз, что сократит время, затрачиваемое на разработку и верификацию устройства). Также, в результате работы компилятора получится «гарантированно чистый» RTL-код, который не содержит ошибок и некорректных схемных решений.

Если рассуждать, оперируя терминами программирования, язык Confluence поддерживает рекурсию, типы данных высшего порядка, лексический анализ и прозрачность ссылок (более чем достаточно, чтобы привести в восторг любого разработчика).

1970 г. *Америка. Боб Меткалф (Bob Metcalf) и Дэвид Боггс (David Boggs) в исследовательском центре в Пало-Альто разработали описание Ethernet.*

1970 г. *Америка. Компания Fairchild представила первую 256-битную микросхему статического ОЗУ, которая получила название 4100.*

Простой пример

В качестве простого примера давайте рассмотрим компонент языка Confluence, который позволяет описать последовательное (каскадное) соединение любого количества любых элементов с одним входом и одним выходом:

```
component Cascade +Stages +SisoComp +Input -Output
  is
    if Stages <= 0
      Output <- Input
    else
      Output <- {Cascade (Stages - 1) SisoComp
        {SisoComp Input $} $}
    end
end
```

Хотя люди, не связанные с программированием, могут отнестись к этому тексту с опаской, но на самом деле ничего страшного здесь нет. В первой строке описывается новый компонент, который мы решили назвать Cascade, с которым связано четыре параметра: Stages (количество необходимых стадий, т. е. количество элементов в последовательной цепочке), SisoComp (название элемента, из которого вы решили создать последовательную цепочку), Input (название входного сигнала или сигналов, если используется шина) и Output (название выходного сигнала или сигналов, если используется шина).

Заметим, что к ключевым словам языка в этой строке относятся только «component» и «is», а выражения «Stages», «SisoComp», «Input» и «Output» являются определяемыми пользователем именами переменных. Знаки «+» и «—» в этой строке отображают связь пользовательских переменных с портами ввода и вывода соответственно.

Более того, когда мы говорили, что этот компонент описывает последовательное соединение любых элементов с одним входом и одним выходом, то на самом деле имелось в виду, что переменные Input и Output могут представлять собой многобитные шины. Фактически, эти сигналы даже не должны быть двоичными векторами, они могут быть списками битовых векторов или списками списков битовых векторов (или любым другим подобным типом данных).

В качестве простого примера использования нашего нового компонента под названием Cascade давайте, допустим, что нам по какой-то причине необходимо связать вместе 1024 вентиля НЕ (не спраши-

вайте зачем) таким образом, чтобы выходной сигнал с первого вентиля поступал на вход второго, со второго на третий и т. д. В этом случае мы могли бы эту процедуру описать с помощью одной строки, которая вызывает наш компонент Cascade и передает ему соответствующие параметры:

```
{Cascade 1024 ('~') Input Output}
```

В этом случае компилятор языка Confluence понимает, что символ «~» обозначает простейший логический элемент отрицания (НЕ).

В качестве более интересного примера давайте рассмотрим ситуацию, когда мы хотим организовать каскад из шестнадцати 8-битных регистров, в котором выходы первого регистра соединялись бы с входами второго, выходы второго — с входами третьего и т. д. В этом случае нам необходимо сначала описать компонент под названием, например, Reg8, представляющий собой 8-битный регистр, а затем использовать компонент Cascade, который и будет 16 раз тиражировать этот регистр:

```
component Reg8 +A -X is
  {VectorReg 8 A X}
end
{Cascade 16 Reg8 Input Output}
```

Здорово, да? Но будет ещё лучше! А как насчет возведения в квадрат значений сигналов, и так четыре раза подряд, да ещё с конвейерным регистром между соседними стадиями? Мы можем легко и быстро представить эту процедуру следующим образом:

```
component RegisteredPowerOfTwo +A -X is
  {Delay 1 (A '*' A) X}
end
{Cascade 4 RegisteredPowerOfTwo Input Output}
```

Как мы можем увидеть, наш компонент Cascade представляет хороший пример рекурсии и использования типов данных высокого уровня, что составляет две основные характеристики функционального программирования, обеспечивающие высокие уровни абстракции и повышающие возможность повторного использования ранее разработанного кода.

Очень радует тот факт, что переменная-элемент SisoComp не подразумевает, что порты ввода и вывода должны быть одинаковой ширины. Фактически эта переменная может быть связана с любой определяемой пользователем функцией, и даже может получать на входе описание компонент, а после обработки выводить другой компонент, или она может получить на входе систему, и затем выводить другую систему. Не существует никаких ограничений, чтобы элемент SisoComp работал не только с двоичными векторами, но также и с целыми числами, переменными, списками, компонентами, системами или любыми другими типами данных языка Confluence.

И в завершающем примере элемент SisoComp может использоваться для соединения двоичного вектора с самим собой, тем самым, удваивая количество бит. Для того, чтобы проиллюстрировать этот процесс, давайте допустим, что мы создаем новый компонент под названием SelfConcat:

```
component SelfConcat +A -X is
  X = A '++' A
end
```

где «++» обозначает оператор соединения (конкатенации). Когда SelfConcat используется вместе с компонентом Cascade, тогда двоичный вектор возрастает в два раза на каждом этапе. Например, допустим, что мы начинаем с 2-битного вектора, установленного в значение **01** и помещаем SelfConcat в Cascade:

```
{Cascade 4 SelfConcat '01' Output}
```

В этом случае на выходе мы получим 32-битный вектор со значением **01010101010101010101010101010101**.

Безусловно, язык VHDL всегда поддерживал операторы генерирования, такая же возможность была добавлена в Verilog, начиная с версии 2K1, но из языка Confluence эти операторы словно ветром сдуло.

1970 г. *Америка. Компания Intel анонсировала первую 1024-битную микросхему динамического ОЗУ, которая получила название 1103.*

Но подождите, есть кое-что ещё

Как было сказано раньше, трудно охватить вниманием все возможности языка Confluence. Возможно, лучше всего это сделать с помощью наглядного примера (**Рис. 24.4**).

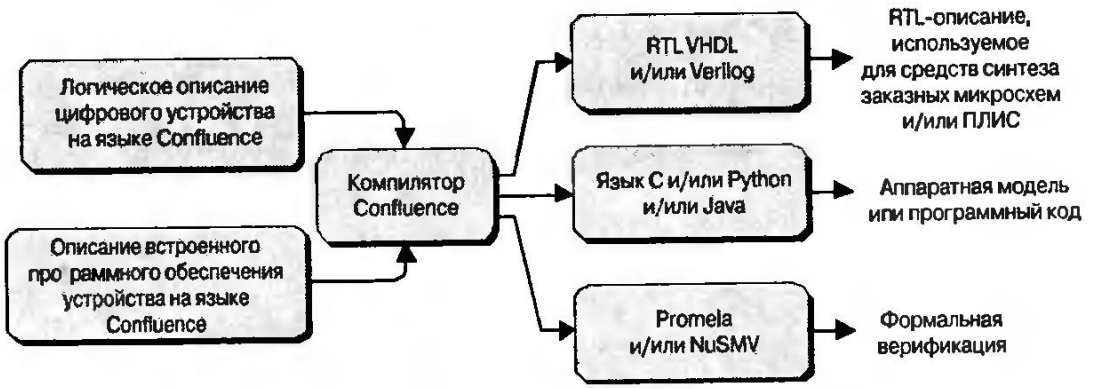

Рис. 24.4. Более полное представление возможностей компилятора языка Confluence

Компилятор может принимать для обработки исходные тексты языка Confluence, описывающие аппаратные элементы схемы устройства, или фрагменты встроенного программного обеспечения устройства, написанные на этом же языке. Если источником является описаниие аппаратных элементов, нужно дать команду компилятору Confluence генерировать RTL-код на языке VHDL или Verilog, чтобы впоследствии использовать его для моделирования и синтеза.

Также можно использовать компилятор Confluence для создания эквивалентных исходных текстов на языках ANSI C, Python или Java (более подробно о языке Python *см.* гл. 25). Если при этом ваш исходный текст описывает работу аппаратного обеспечения, тогда полученные выходные описания могут выступать в качестве потактных или битовых высокопроизводительных моделей, которые можно включить в собственную среду верификации. Аналогично, если исходный текст представляет собой встроенное программное обеспечение, то на выходе получим исполняемый код для использования в среде совместной верификации аппаратного и программного обеспечения.

1970 г. *Для хранения компьютерных данных начали применяться флоппи-диски (8.5 дюймов).*

1970 г. *Исследования компании Corning Glass привели к появлению первого коммерчески-используемого оптического кабеля.*

И последнее, но не менее важное, компилятор языка Confluence может создавать описания на языках PROMELA или NuSMV, которые используются в средствах формальной верификации SPIN и NuSMV соответственно (формальная верификация рассматривалась нами в гл. 19, а PROMELA, SPIN и NuSMV будут рассмотрены в гл. 25).

Бесплатная копия

Если вы посетите сайт компании Launchbird (www.launchbird.com), то там вы найдете большое количество примеров исходного текста, написанных на языке Confluence. Особенно приятно то, что каждый может загрузить с этого сайта и использовать совершенно бесплатно одну неограниченную лицензию. За последующие лицензии придется выложить определённую сумму (версии для учебных заведений выпускаются бесплатно), но цены часто меняются, поэтому последнюю информацию по этому вопросу лучше узнать непосредственно у компании Launchbird.

Но действительно радует то, что вы остаетесь полноправным собственником всех своих разработок, выполненных на одной бесплатной лицензии (под которую попадают все исходные коды на языке Confluence и полученные из них тексты на языках VHDL, C или Verilog) и можете делать с ними всё, что пожелаете, в том числе их можно и продать.

У вас есть средство?

Если вам приходилось сталкиваться с полезными средствами или утилитами для проектирования от небольших фирм-разработчиков, или если вы сами создаёте средства такого типа, то, пожалуйста, не стесняйтесь и свяжитесь со мной по электронной почте max@techbites.com, и, возможно, информация о таких средствах появится в следующих изданиях этой книги или в выходящем раз в два месяца разделе «Max Bytes» на сайте www.eedesign.com.