ГЛАВА 11

ПРОЕКТИРОВАНИЕ НА ОСНОВЕ C/C++ И ДРУГИХ ЯЗЫКОВ

Проблемы использования HDL-языков

В этой главе рассматриваются только общие вопросы проектирования с применением языков C/C++. Рассмотрение других вопросов, например, связанных с квантованием (замена числа с плавающей точкой на число с фиксированной точкой), более подробно рассматриваются в гл. 12.

При использовании традиционных методов проектирования ПЛИС, основанных на HDL-языках, которые были рассмотрены в гл. 9, создание устройства начинается с того, что архитекторы совместно с разработчиками создают концепцию и определяют функции системы. На этом уровне вырабатываются *макроархитектурные* определения, например разделение устройства на программные и аппаратные компоненты (*см.* гл. 13).

Итоговая спецификация передаётся разработчикам аппаратных модулей системы, которые начинают, свою часть процесса с решения задач *микроархитектурных определений*, например детализации управляющих структур, шин и элементов тракта передачи данных. Эти микроархитектурные определения, которые зачастую появляются на белой доске во время совещания инженеров в результате мозговой атаки, могут включать формирование определённых операций в параллельных и последовательных, конвейерных и неконвейерных частях устройства с использованием общих ресурсов, например две операции выполняются на одном умножителе и используют выделенные ресурсы, и без таковых.

В конечном итоге все функции устройства описываются терминами уровня регистровых передач (RTL) на VHDL или Verilog. После проверки устройства с помощью средств моделирования RTL-представление трансформируется в структурную таблицу соединений, пригодную для промышленных средств размещения и разводки элементов устройства (**Рис. 11.1**).

При реализации ПЛИС таблица соединений конфигурируемых логических блоков/таблиц соответствия может быть представлена в EDIF, VHDL или Verilog-версиях в зависимости от поставщика микросхем.

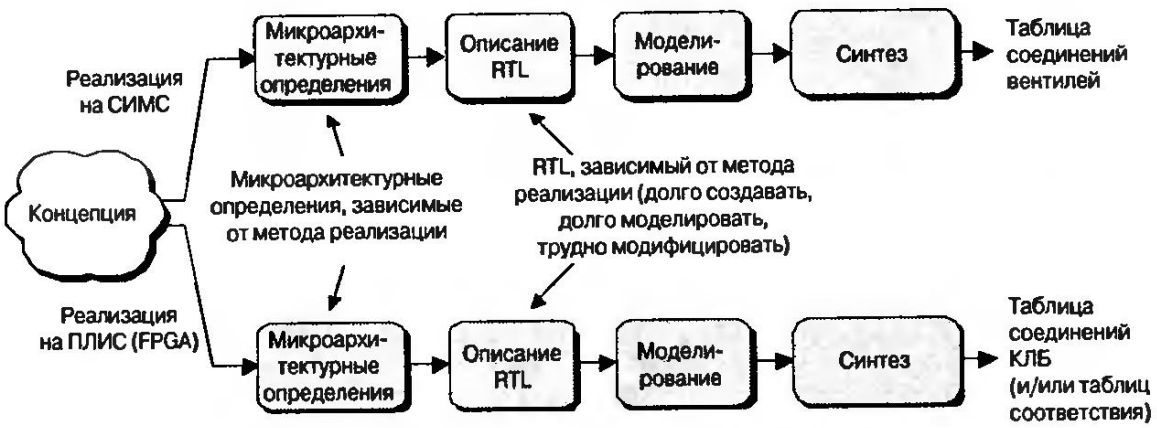

Рис. 11.1. Традиционное проектирование на основе HDL-языков (упрощенно)

Во время написания этой книги около 95% всех методов проектирования заказных микросхем или ПЛИС основывались на языках программирования VHDL или Verilog. Однако при работе с этими языками возникает ряд проблем:

- *Большие временные затраты на описание RTL* — несмотря на то что языки VHDL и Verilog предназначены для представления аппаратного обеспечения, описание функциональности устройства с их помощью занимает много времени.
- *Большие временные затраты на проверку RTL* — использование моделирования для проверки больших устройств, представленных в виде *RTL*-кода, требует больших вычислительных и временных затрат.
- *Трудности с оценкой альтернативных реализаций* — для выполнения ряда оценок альтернативных микроархитектурных реализаций используют модификацию и повторную проверку *RTL*, выполнение которых является весьма затруднительным и трудоемким. Это означает, что команда разработчиков ограничена в количестве оценок различных вариантов устройства, что может стать результатом не вполне оптимальной реализации.
- *Трудности внесения изменений в спецификацию устройства* — если в ходе реализации проекта потребуется внести изменения в технические требования к устройству, то свертывание этих изменений в *RTL* и выполнение необходимых перепроверок может оказаться весьма неприятным и трудоемким процессом. Для некоторых прикладных областей это замечание является существенным, например, для сферы беспроводной передачи данных, что связано с постоянным развитием и изменением стандартов и протоколов передачи данных.
- *Особенности реализации RTL* — реализация устройства на ПЛИС обычно требует иного стиля кодирования RTL, чем при использовании заказных микросхем (смотри также главы 7, 9 и 18). Другими словами, может оказаться чрезвычайно трудно перенести сложную схему, представленную в RTL-коде, из одной технологии в другую. Это замечание относится к случаям переноса существующих устройств, выполненных на заказных микросхемах, в их эквиваленты на основе ПЛИС, или к случаям реализации устройств в ПЛИС в качестве прототипов для последующей реализации устройств на заказных микросхемах. Чтобы прояснить ситуацию, скажу, что информация о реализации устройства жестко зашита в RTL, который вследствие этого становится зависимым от реализации конкретного устройства. Важно понять, что эта особенность выходит за рамки грубого сравнения заказных микросхем и ПЛИС, согласно которому RTL-код, предназначенный для ПЛИС устройства, не подходит для оптимальной реализации заказной микросхемы, и наоборот. Даже в устройствах, предназначенных для решения одних и тех же задач, набор алгоритмов, используемых для обработки данных, может потребовать ряд различных микроархитектурных реализаций в зависимости от целевого назначения устройства. Если пойти до конца, скажу, что одни и те же RTL могут использоваться для реализации как заказных микросхем, так и ПЛИС. Этот подход применяется для предотвращения функциональных ошибок в RTL при переносе кода из одной области в другую, но за это приходится платить. Это значит что, если код, изначально предназначенный для ПЛИС, используется для реа-

1895 г. *Америка. Абонентские телефоны начали использоваться в здании городского совета города Милуоки.*

лизации заказной микросхемы ASIC, в результате такого переноса заказная микросхема обычно будет занимать больше места на кристалле и будет потреблять энергии больше, чем устройство, изначально разработанное для ASIC. Аналогично, если код, изначально предназначенный для ASIC применяется для реализации ПЛИС, конечное устройство, как правило, потребует существенно больше ресурсов, чем при использовании RTL, созданного специально для FPGA.

- *RTL не является идеальным средством для аппаратно-программного проектирования* — к однокристальным устройствам, как правило, относят такие, которые содержат микропроцессорные ядра. Независимо от способа реализации этих устройств, т. е. с помощью заказной микросхемы или ПЛИС, современные однокристальные системы тяготеют к увеличению доли программных компонентов. Кроме того, в связи с участившимися случаями повторного использования аппаратной части таких устройств, как правило, требуется одновременно проверять программные и аппаратные части, чтобы полностью подтвердить достоверность таких компонентов, как системная диагностика, операционная система реального времени, драйверы устройств, встроенные программные приложения. В общем случае довольно-таки сложно проверить (промоделировать) аппаратную часть, описываемую языком VHDL или Verilog, вместе с программной частью, написанной на языке C/C++ или ассемблера.

Системы реального времени работают по принципу не только **что** выполнить, но и **когда** это должно быть выполнено.

Один из методов решения описанных выше проблем заключается в первоначальном описании устройства на более высоком уровне абстракции, чем тот, которого можно достичь при использовании языка VHDL или Verilog. Первый такой уровень достигается при использовании какой-либо формы записи языка C/C++, и, следовательно, вновь возникает не простая ситуация, поскольку имеются различные версии этого языка, например, SystemC, расширенный C/C++ и немодифицированный (чистый) C/C++.

C, C++ и другие версии

Прежде всего необходимо рассмотреть несколько понятий, чтобы убедиться, что все мы говорим на одном и том же языке. Во-первых, существует огромное количество языков программирования, но, за исключением специальных областей, наиболее часто используется традиционный язык C и его объектно-ориентированный потомок C++. В своем повествовании я буду ссылаться на них, пользуясь общим названием C/C++.

Во-вторых, и это очень важно, по определению операторы в языках, подобных C/C++, выполняются *последовательно*. Например, допустим, что мы уже описали три переменных целого типа, обозначенных как a, b, c. Тогда нижеприведенные операторы

```
a = 6;      /* Оператор в программе C/C++ */
b = 2;      /* Оператор в программе C/C++ */
c = 9;      /* Оператор в программе C/C++ */
```

будут выполняться, как и полагается, один за другим. Однако такая последовательность имеет определенные последствия. Например, если предположить, что в программе появятся операторы вида

```
a = b;      /* Оператор в программе C/C++ */
b = a;      /* Оператор в программе C/C++ */
```

то переменной a, которой изначально было присвоено значение 6, будет присвоено значение, которое в настоящее время хранится в переменной b, т. е. 2.; переменной b, которой в начале было присвоено значение 2, будет присвоено значение, которое в настоящее время хранится в переменной a, т. е. 2. Другими словами, в результате выполнения этих выражений переменным a и b будет присвоено одинаковое значение.

1895 г. *Германия. Вильгельм Конрад Рентген (Wilhelm Kohnrad Roentgen) открыл рентгеновские лучи.*

Последовательное выполнение команд в языках программирования отражает ход мысли разработчиков программного обеспечения. Однако разработчики аппаратных средств смотрят на мир по-другому. Допустим, что в аппаратной части устройства содержится два регистра a и b, на которые подается общий тактовый сигнал. Предположим, что в эти регистры изначально загружены значениями соответственно 6 и 2 и в некоторой части HDL-кода находятся следующие операторы:

```
a = b;      /* Выражение в коде VHDL или Verilog */
b = a;      /* Выражение в коде VHDL или Verilog */
```

Как обычно, приведенный здесь синтаксис не соответствует языкам VHDL или Verilog: эта общепринятая форма записи приведена здесь только для наглядности рассматриваемого примера. При этом разработчики аппаратных средств ожидают, что оба этих оператора будут выполняться одновременно. Это значит, что в регистр a, в который изначально было записано число 6, будет загружено значение, которое в настоящее время хранится в регистре b, т. е. 2, и одновременно в регистр b, в который изначально было записано число 2, будет загружено значение, которое в настоящее время хранится в регистре a, т. е. 6. В результате выполнения этих операторов регистры a и b обменяются своим содержанием.

Как обычно, всё наши рассуждения несколько упрощены по сравнению с практикой. Справедливо будет заметить, что HDL-операторы будут выполняться одновременно по умолчанию. Однако это правило можно отменить с помощью некоторых методов, например, метода блокирующего присваивания. Таким образом, по определению подсистема моделирования на основе RTL будет выполнять приведённые выше операторы одновременно; аналогично средства синтеза на основе RTL буду синтезировать аппаратную часть устройства, которая также будет выполнять эти два действия одновременно. В отличии от них, если явно не задано иное , операторы языка C/C++ будут выполняться последовательно.

Проектирование на языке SystemC

Что такое SystemC и его история

Перед тем, как приступить к обсуждению методов проектирования, основанных на языке SystemC, целесообразно кратко рассмотреть, что собой представляет этот язык, поскольку при обсуждении этого вопроса обычно возникает некоторая неразбериха, в том числе и у автора.

Язык SystemC находится под покровительством независимой некоммерческой организации OSCI (Open SystemC Initiative), которая объединяет различные компании, университеты и независимых разработчиков и продвигает SystemC как стандарт с открытым исходным кодом, предназначенный для проектирования на системном уровне.

Код SystemC вместе со встроенной системой моделирования и средой проектирования можно найти на сайте www.systemc.org.

1895 г. *Россия. Александр Попов сконструировал приёмник природных электромагнитных волн, предназначенный для предупреждений о грозе.*

SystemC 1.0

Основная концепция языка SystemC заключается в том, что он является программным продуктом с открытым кодом, в который любой желающий может внести свой вклад. В качестве аналогичного примера можно привести операционную систему Linux, которая в самом начале своего развития ничего особенного собой не представляла. Однако, основываясь на открытом коде и заручившись поддержкой многих людей, Linux в конечном итоге стала настоящей *операционной системой (ОС)*, обладающей достаточным потенциалом, чтобы бросить вызов компании Microsoft.

Относительно недокументированный открытый программный продукт SystemC пустился в свободное плавание где-то в 2000 году. Это был объектно-ориентированный язык программирования, который поддерживал библиотеки классов, что упрощало реализацию параллельных процессов (ситуации, когда несколько операций обрабатываются параллельно), синхронизации и выводов входа/выхода. Используя библиотеку классов, инженеры могли описывать устройства с помощью абстракций, выполненных на уровне регистровых передач.

Реализация подходов с использованием языка SystemC давала инженерам ряд преимуществ. Одно из них заключалось в том, что применение этих методов сделало возможным создание среды программно-аппаратного кодирования. Другое преимущество состояло в том, что моделирование устройства, описание которого было реализовано с помощью абстракций уровня регистровых передач на языке SystemC, могло быть выполнено в 5...10 раз быстрее, чем при использовании языков VHDL или Verilog[1]. В то же время использование этого языка имело ряд неудобств и в частности требовало существенно больше времени и сил для создания RTL-описания устройства. Кроме того, ощущалась нехватка программных средств, позволяющих синтезировать таблицу соединений элементов устройства из кода, написанного на SystemC.

SystemC 2.0

В 2002 году появилась следующая версия языка — SystemC 2.0. Эта версия отличалась от предыдущей наличием некоторых высокоуровневых схем, предназначенных для моделирования, например — очереди FIFO. Это устройство представляет собой форму памяти, которая может принимать и впоследствии выдавать в обработку последовательность слов данных по принципу «первым пришел, первым вышел».

Версия 2.0 также поддерживала ряд поведенческих, алгоритмических и моделирующих возможностей, например концепции транзакций и каналов (которые используются для абстрактного описания процесса передачи данных между блоками). Чтобы представить перспективу применения этих нововведений, давайте сначала рассмотрим использование первоначальной версии языка SystemC 1.0 на простом примере. Допустим, что имеются две функции *f(x)* и *g(x)*, которые требуется связать между собой (**Рис. 11.2**).

В случае схемы, представленной на **Рис. 11.2** интерфейс между блоками должен быть определен на физическом уровне, т. е. на уровне

---

[1] Это значение существенно зависит от схемы устройства; в действительности, некоторые модели, созданные с помощью языка SystemC, обрабатываются не быстрее, чем их HDL-аналоги.

1897 г. *Англия. Гульельмо Маркони (Guglielmo Marconi) передал сообщение в коде Морзе через Бристольский канал.*

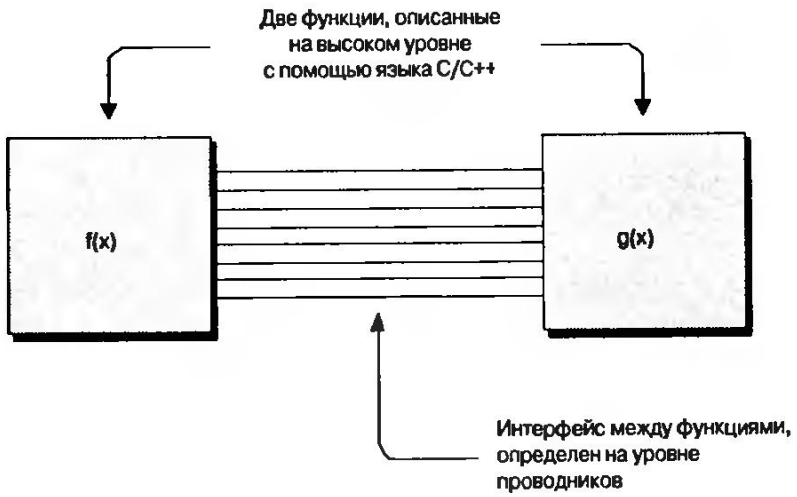

Рис. 11.2. Связь двух функций с помощью SystemC 1.0

проводников, связывающих их между собой. Проблемы, связанные с этим методом, дают о себе знать на первых стадиях проектирования, так как на этом этапе уже должны быть определены все детали реализации, в частности ширина шины. Это усложняло процедуру внесения изменений в схему при попытке попробовать различные варианты архитектуры устройства. Однако эта проблема перестала существовать с приходом SystemC 2.0, который позволил выделить в явном виде абстрактные интерфейсы между этими блоками (**Рис. 11.3**).

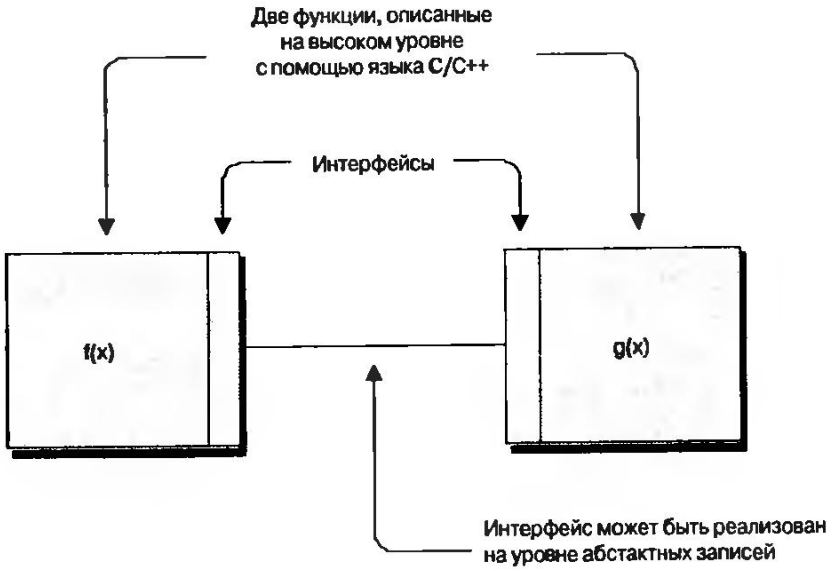

Рис. 11.3. Связь двух функций в SystemC 2.0

Теперь связь между блоками можно было описать с помощью абстрактных записей, и на ранних стадиях проектирования устройства не заботиться о том, каким образом данные из точки *а* передаются в точку *б*, а лишь ограничиться констатацией факта такой передачи.

Абстрактные интерфейсы облегчают формирование архитектурных оценок на ранних этапах проектирования. Начало формирования архитектуры можно совместить с началом процесса усовершенствования интерфейсов путем применения высокоуровневых устройств, таких как очереди FIFO. При использовании таких очередей должны быть установлены параметры, как, например, ширина и глубина очереди, а также ряд характеристик, таких как «блокирующая запись», «неблокирующее чтение», и порядок работы при опустошении и пол-

1897 г. *Англия. Г. Маркони установил первую станцию Маркони на острове Уайт и передал сигнал через 22 км на побережье Англии.*

ном заполнении очереди. На последующих этапах логические интерфейсы могут быть полностью заменены специализированными интерфейсами, позволяющими связать функциональные блоки на физическом уровне.

Уровни абстракции

К сожалению, определение уровней абстракции несколько расплывчатое, размытое. Очевидно, это связано с тем, что каждый стремится внести свою лепту в это определение и придать этому определению только ему понятный смысл. Для начала рассмотрим описание различных уровней абстракции языка SystemC **(Рис. 11.4)**.

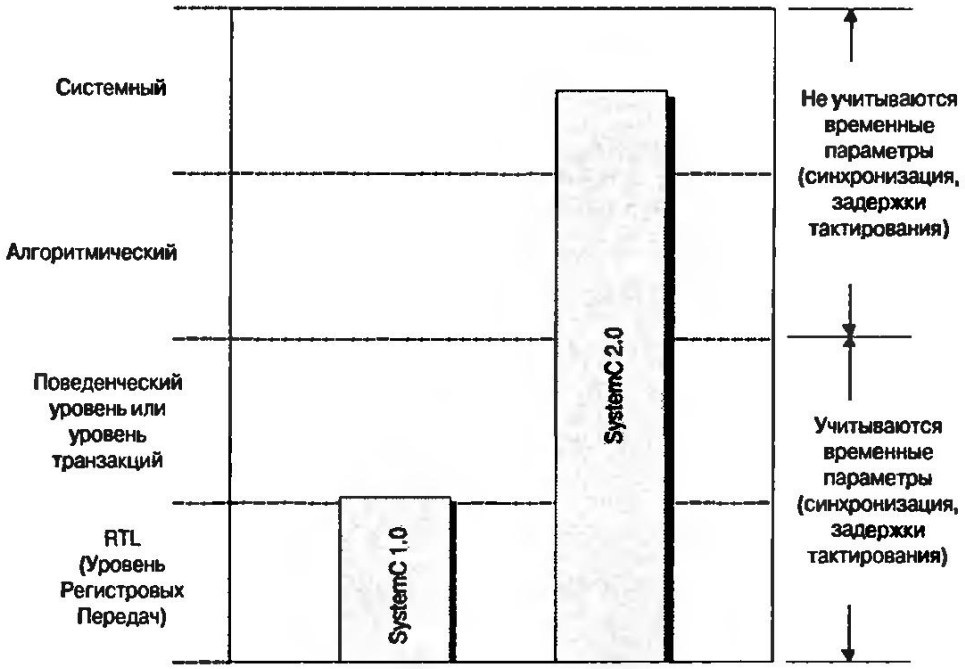

Рис. 11.4. Уровни абстракции языка SystemC

Неразбериха возникает из-за того, что в процессе проектирования язык SystemC может использоваться по-разному. Так, например, он может «замещать» языки VHDL/Verilog, когда речь идет об описания уровня регистровых передач. В другом случае он может оказаться единственным языком, который можно использовать для описания устройства на системном уровне, для алгоритмического и архитектурного анализа, поведенческого проектирования и создания испытательных стендов с целью верификации.

Примером неразберихи и недоразумений может служить поведенческий синтез, в котором, безусловно, проявляются аспекты как алгоритмического, так и транзакционного уровней. В последнем случае следует быть аккуратным при определении транзакций.

Альтернативные методы проектирования на основе языка SystemC

Это достаточно сложная тема, поскольку она имеет много различных аспектов. Например, многие современные устройства начинают свою жизнь в виде набора сложных алгоритмов. В этом случае разработку устройства можно начать с реализации этих алгоритмов на языке C/C++ и полученный код скомпилировать в форму, которая может быть использована для проверки работоспособности алгоритмов. При

этом моделирование может выполняться в тысячи раз быстрее, чем аналогичная проверка, выполненная на уровне регистровых передач.

В случае использования HDL-методов проектирования, рассмотренных в гл. 9, представление алгоритмов с языка C/C++ может быть экспортировано в RTL-код языка VHDL/Verilog. При этом код языка C/C++ обычно продолжают использовать в качестве *золотой модели*, которая может быть связана с системой моделирования RTL и работать параллельно с RTL-моделью. Результаты работы обеих моделей затем сравниваются для того, чтобы гарантировать их функциональную эквивалентность.

Еще один способ применения языка SystemC для проектирования устройств заключается в постепенной модификации первоначального кода C/C++ путем добавления в него возможностей для обеспечения параллелизма, синхронизации, определения контактов и так далее, т. е. для модификации кода до уровня, когда он может быть подвержен поведенческому синтезу.

А можно пойти и по такому пути, как создание первоначального описания устройства с помощью конструкций системного, алгоритмического уровня или уровня транзакций языка SystemC, которые могут быть использованы для верификации устройства на высоком уровне абстракций. Это описание может быть постепенно модифицировано и переведено на более низкий уровень регистровых передач SystemC или уровень поведенческого синтеза.

Не зависимо от того, какой из описанных выше способов формирования кода используется, допустим, что имеется описание устройства на языке C/C++, которое подходит для RTL или поведенческого синтеза на основе SystemC. В этом случае существует два подхода к последовательному проектированию. Первый из них состоит в использовании средств синтеза языка SystemC для непосредственной генерации таблицы соединений. Второй подход заключается в автоматической трансляции из SystemC в RTL VHDL/Verilog и последующем использовании традиционных средств RTL-синтеза.

Поклонники первого подхода утверждают, что прямой синтез таблицы соединений средствами SystemC предлагает более быстрое, наглядное и эффективное решение. Приверженцы второго подхода настаивают на том, что лучше транслировать исходный код C/C++ в RTL VHDL/Verilog. Кроме того, RTL предоставляет разработчикам визуализацию кода устройства, также на этом уровне легко проводить интеграцию различных блоков, в том числе блоков интеллектуальной собственности сторонних разработчиков. Не говоря уже о том, что средства синтеза VHDL/Verilog отличаются мощностью и глубиной проработки по сравнению со средствами SystemC. Но я вновь отвлекся.

Оба подхода могут быть с успехом применены как для ПЛИС, так и для заказных микросхем (**Рис. 11.5**).

Первые версии программ синтеза на основе кода SystemC преимущественно использовались для заказных микросхем, так как эти средства довольно неэффективно работали со специфичными для ПЛИС блоками, такими как: встроенное ОЗУ, встроенные умножители, и другими. Последние версии этих программ намного лучше справляются с блоками ПЛИС, но их уровень сложности постоянно меняется, и продвинутому пользователю я настоятельно рекомендую до оформления заказа провести тщательный анализ этих средств.

На **Рис. 11.5** показано использование зависимого от реализации кода SystemC в процессе проектирования ПЛИС и заказных микросхем. Как только начнется процесс кодирования на уровне регистро-

1901 г. *Хьюберт Бут (Hubert Booth) изобрёл пылесос.*

1901 г. *Г. Маркони послал радиосигнал через Атлантику.*

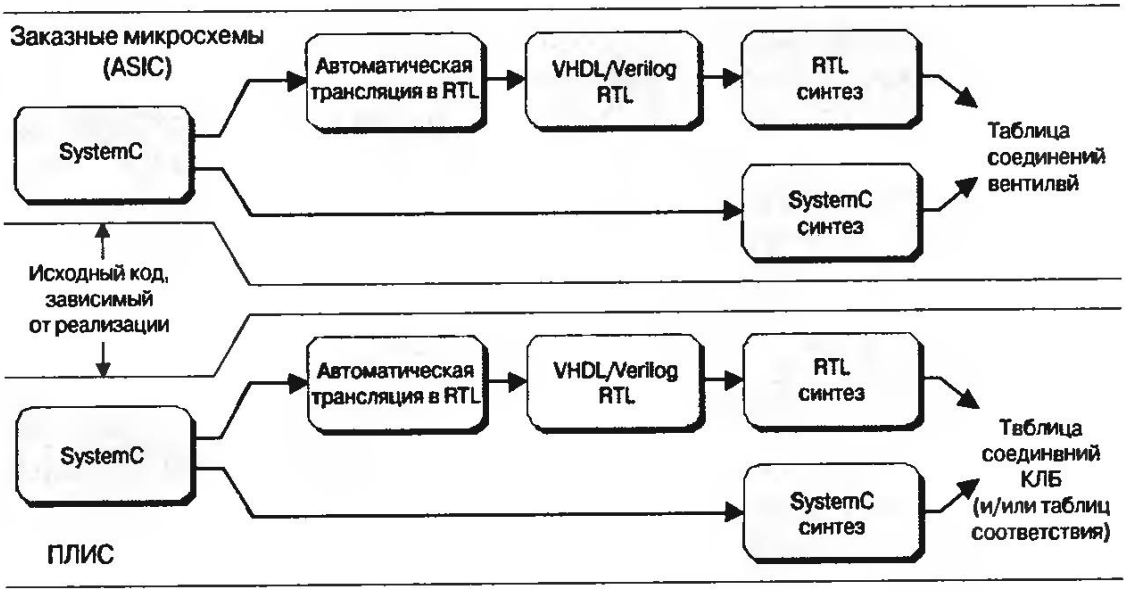

Рис. 11.5. Последовательность проектирования на основе языка SystemC

вых передач и будет добавлено понятие синхронизации, независимо от используемого языка программирования, будь-то VHDL, Verilog или C/C++, для достижения оптимальной реализации устройства код должен зависеть от специфики архитектурной реализации.

В этом случае один и тот же код SystemC может использоваться как при построении ПЛИС, так и при построении заказных микросхем (ASIC), но за такую универсальность придётся платить. Если такой код SystemC изначально предназначался для создания ПЛИС, а использовался для реализации заказной микросхемы, изготовленная таким способом схема будет занимать больше места на поверхности кристалла и потреблять энергии больше, чем устройство, изготовленное с использованием кода, ориентированного на заказную микросхему. Аналогично, если код, изначально предназначенный для реализации заказных микросхем, использовался для создания ПЛИС, изготовленное таким образом устройство будет потреблять значительно больше ресурсов, чем устройство, изготовленное по коду, ориентированному на ПЛИС. Все описанные здесь накладные расходы при нецелевом использовании кода возникают вследствие жесткого аппаратного кодирования микроархитектурных определений на уровне исходного кода.

Принять иль не принять

Разработчики, занимающиеся созданием ПЛИС, делятся на две группы: на сторонников и противников языка SystemC. Большинство специалистов утверждают, что SystemC 2.0 весьма перспективен и обладает непревзойдёнными возможностями проектирования. Эти возможности уже реализованы в языке SystemVerilog, но еще далеко не все. Также нужно отметить, что SystemC может быть очень полезен для верификации и высокоуровневого системного моделирования, но в полной мере эти возможности пока не востребованы реальными методами проектирования.

Большинство специалистов, занимающихся проектированием устройств, на достаточно хорошем уровне владеют навыками написания кода на языке C, но многие из них имеют весьма смутное представление об объектно-ориентированных аспектах C++. Следовательно, требуя от некоторых разработчиков, чтобы они пользовались языком

SystemC, мы, с одной стороны, стремимся вручить им мощное средство, а с другой стороны, толкаем в мир, который им, мягко говоря, не симпатичен или не совсем понятен.

Некоторые инженеры говорят, что хотя SystemC относительно тяжело использовать для написания кода вручную и также тяжело производить синтез устройства (что делает его отчасти грубым языком спецификаций), но он предоставляет мощную основу для моделирования на разных уровнях абстракции с применением различных языков.

В то время, когда я писал эту книгу, среди компаний, занимающихся поддержкой и развитием языка SystemC в Соединённых Штатах, наметился некий спад продвижения своих продуктов. Однако этот язык нашёл своих приверженцев в Европе и Азии. Что ожидает SystemC в будущем? Поживем, увидим.

1902 г. *Америка. Миллар Хутчинсон (Millar Hutchinson) изобрёл электрический слуховой аппарат.*

Проектирование на расширенном C/C++

Что такое расширенный C/C++

В стандартной версии языка C/C++ существует два способа расширения его функциональности. Первый из них заключается во включении в «чистый» C/C++ код специальных комментариев, которые называют *комментированными директивами* или *псевдокомментариями (прагмами)*. Эти комментарии распознаются и интерпретируются программами синтаксического анализа или парсерами, препроцессорами, компиляторами и другими средствами, а затем используются в процессе создания кода или изменения метода его построения[1]. Существенным недостатком этого метода является тот факт, что системы моделирования, построенные на основе C/C++, требуют использования собственных компиляторов. Это обстоятельство ограничивает набор применяемых для расширения опций, реализация которых возможно лишь в том случае, когда поставщики САПР электронных систем поддерживают стандарты расширения языка.

Еще один способ расширения возможностей языка C/C++ заключается в добавлении в язык специальных ключевых слов и выражений. Этот метод пользуется большой популярностью, и в настоящее время во всем мире существует огромное количество различных вариантов языковых модификаций C/C++ для разных сфер применения. Главным недостатком этого расширения, так же как и в предыдущем методе, является то, что средства моделирования для своей работы требуют применения собственных компиляторов, и если в исходном коде будут содержаться непонятные для них конструкции, это приведет к краху системы моделирования. Общее решение этих проблем заключается в том, что новые ключевые слова и выражения заключаются в стандартные директивы препроцессора *#ifdef*. Таким образом, препроцессор при необходимости может пропускать непонятные для него новые конструкции. Этот метод не отличается элегантностью, но он работает.

При описании аппаратной функциональности для ПЛИС и заказных микросхем часто возникает необходимость расширить стандартные возможности языка C/C++ для поддержки таких понятий, как

---

[1] Примером такой формы расширения языка C/C++ может служить *технология проверки условий* компании 0-In Design Automation (www.0in.com). В качестве другого примера можно упомянуть компанию Future Design Automation (www.future-da.com), которая использует упомянутые здесь методы в системах RTL-синтеза.

1902 г. *Роберт Бош (Robert Bosch) изобрёл свечу зажигания.*

синхронизация, тактовый сигнал, выводы, параллелизм, и *общие (выделенные) ресурсы*[1].

Допустим, что мы располагаем начальной моделью системы, описанной на чистом C/C++. Первым делом необходимо расширить её, добавив выражения, описывающие систему формирования синхроимпульсов, а также описать с помощью выражений интерфейсы для определения контактов ввода/вывода. После этого с помощью соответствующего средства синтеза вы можете создать реализацию своего устройства. Поскольку C/C++ по своей природе является языком последовательного выполнения, полученная в результате процедуры синтеза аппаратная реализация может оказаться чрезвычайно медленной и неэффективной, если только в ней не будут реализованы принципы параллелизма.

Допустим, например, что в описании устройства, выполненного с помощью языка C/C++, находятся следующие операторы:

```
a = 6;      /* Стандартный оператор C/C++ */
b = 2;      /* Стандартный оператор C/C++ */
c = 9;      /* Стандартный оператор C/C++ */
d = a + b;  /* Стандартный оператор C/C++ */
и так далее
```

По умолчанию каждый символ «=» рассматривается средствами синтеза как один период тактового сигнала. Следовательно, если приведенный выше код рассматривать в том виде, как он приведён, система синтеза будет генерировать аппаратное обеспечение следующим образом: на первом такте в регистр *a* будет загружаться число 6, на следующем такте в регистр *b* будет загружено число 2, на следующем такте в регистр *c* загрузится число 9 и так далее. Другими словами, с точки зрения стандартов аппаратного обеспечения, такое устройство будет чрезвычайно медленным.

Конечно, большинство средств синтеза поддерживают реализацию параллелизма в виде, приемлемом для реализации рассмотренного выше примера, но и они могут не справиться с более сложными задачами. В этом случае в процесс синтеза необходимо вмешательство человека для проведения некоторого анализа сложившейся ситуации. Несмотря на простоту рассмотренного примера, будем работать с ним в пределах этой главы. Расширенная версия языка C/C++ должна поддерживать ключевые слова «*parallel*», или «*par*» (параллельный), и «*sequential*», или «*seq*» (последовательный), для формирования инструкций средствам синтеза на последовательную или параллельную реализацию операторов. Например:

```
parallel;       /* Оператор расширенной версии C/C++ */
a = 6;          /* Оператор стандартной версии C/C++ */
b = 2;          /* Оператор стандартной версии C/C++ */
c = 9;          /* Оператор стандартной версии C/C++ */
sequential;     /* Оператор расширенной версии C/C++ */
d = a + b;      /* Оператор стандартной версии C/C++ */
и так далее
```

---

[1] Ярым приверженцем подобной формы расширения языка C/C++ для ПЛИС и заказных микросхем является компания Celoxica (www.celoxica.com), поставляющая на рынок средства ввода описания схемы, моделирования и синтеза на основе языка Handel-C.

В этом случае ключевое слово *«parallel»* указывает средствам синтеза, что следующие за ним выражения могут выполняться одновременно, а ключевое слово *«sequential»* подразумевает, что последующие выражения должны реализовываться поочередно. Конечно, эти выражения размещаются в соответствии с потребностями.

Гораздо более сложная задача возникает при необходимости реализации циклов, причём уровень сложности зависит от способа реализации цикла. Если рассмотреть цикл вида *«для i = 1 до 10 с шагом увеличения 1 выполнить операции xxxx, yyyy и zzzz»*, то в некоторых случаях его можно просто заменить выражениями параллельного или последовательного выполнения. Если же потребуется реализация более изысканных задач, инженеры, вероятно, будут вынуждены полностью переписать этот участок кода.

Возможно также, что необходимо будет добавить в код конструкции *«share», чтобы* обеспечить общий доступ к ресурсам, выражение *«channel», чтобы обеспечить* совместное использование сигналов между выражениями, а также другие ключевые слова.

1902 г. *Кабель, проложенный через Тихий океан, связал Канаду и Австралию.*

Как отмечалось ранее, средства моделирования и другие утилиты проектирования, не поддерживающие новых ключевых слов и выражений, прекращают свою работу, если встречают их в исходном коде. Одно из решений заключается в охвате этих выражений с помощью директив #ifdef, которые позволяют при необходимости отказаться от их использования. Однако это значит, что средства синтеза и моделирования будут по-разному смотреть на схему, что не очень хорошо. Другое решение заключается в использовании патентованных систем моделирования, но они могут быть менее мощными или несовместимыми с имеющимися у пользователя технологиями моделирования.

Расширенный стандарт языка C/C++

Очевидно, что во многих областях один и тот же результат может быть достигнут разными способами. Если жизненный путь устройства начинается с разработки набора алгоритмов, описывающих его работу, процесс создания устройства целесообразно начать с составления описания этих алгоритмов на языке C или C++. После необходимых проверок написанная на языке C/C++ модель может постепенно дополняться выражениями, реализующими тактовые сигналы, интерфейсы ввода-вывода, параллелизм, систему синхронизации и общий доступ к ресурсам, тем самым доводя её до уровня, приемлемого для работы средств синтеза. Можно пойти другим путём, изначально реализовав описание устройства с помощью расширенной версии языка C/C++.

Независимо от типа используемого варианта допустим, что мы уже располагаем описанием устройства на расширенной версии языка C/C++ на уровне, пригодном для использования средствами синтеза. В этом случае, снова повторюсь, существую два основных метода синтеза устройств. Один из них заключается в автоматическом переводе кода, выполненного на расширенном C/C++, на языки Verilog или VHDL на уровне абстракций регистровых передач с последующем применением обычных средств RTL синтеза. Второй метод предусматривает использование собственных средств синтеза на основе расширенной версии языка C/C++.

И ещё раз повторюсь: некоторые утверждают, что прямой синтез из кода C/C++ в таблицу соединений элементов предлагает более простое и эффективное решение. В то же время существует другое мнение,

1902 г. *На кораблях Военно-морских сил (ВМС) США установили радиотелефоны.*

1904 г. *Джон Флеминг (John Ambrose Fleming) изобрёл вакуумный диодный детектор.*

1904 г. *Разработана первая фотоэлектрическая ячейка.*

согласно которому уровень регистровых передач языков Verilog/VHDL наиболее адаптирован для интеграции различных устройств, к тому же современные технологии RTL-синтеза отличаются мощностью и глубиной проработки.

Оба подхода находят своё применение при реализации как ПЛИС, так и заказных микросхем (**Рис. 11.6**). Первые приложения синтеза на основе расширенной версии языка C/C++ использовались преимущественно при проектировании заказных микросхем (ASIC). Объяснить это обстоятельство можно тем, что ранние версии программного обеспечения не могли эффективно работать с ПЛИС-модулями, такими как встроенные блоки ОЗУ, встроенные умножители, и другими. Последние версии этих утилит намного лучше справляются с поставленной задачей, но, как обычно, будущему пользователю настоятельно рекомендую тщательно проверить и оценить то или иное средство, прежде чем оплатить счет.

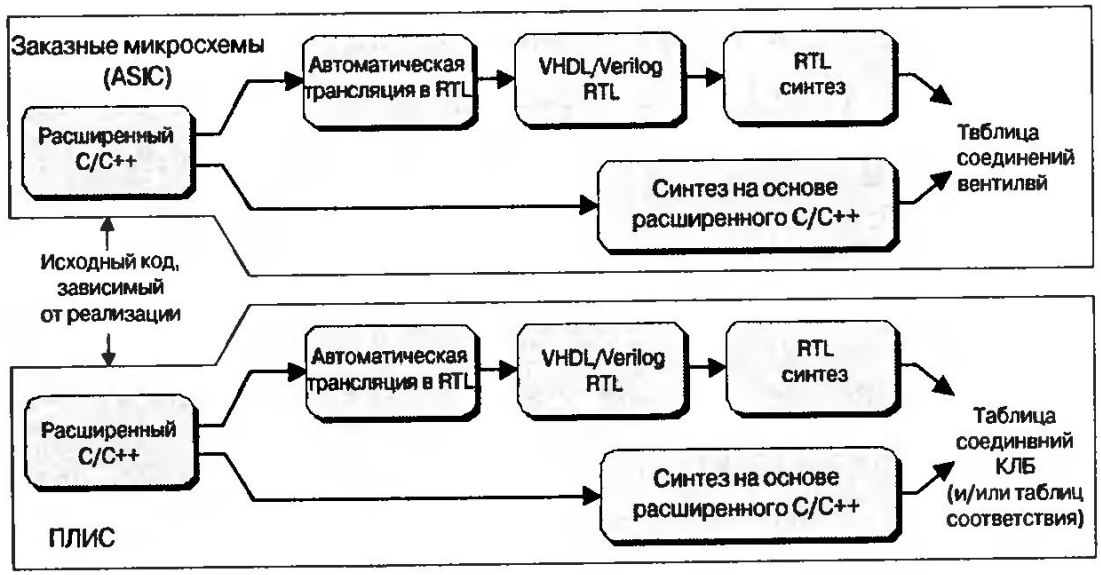

Рис. 11.6. Последовательность проектирования, на основе расширенной версии языка C/C++

Рис. 11.6 демонстрирует использование зависимого от реализации кода при проектировании ПЛИС и заказных микросхем (ASIC), что требуется для реализации оптимального устройства. На практике один и тот же код может использоваться как для построения ПЛИС, так и для создания заказной микросхемы, но, как обычно, при нецелевом использовании кода придётся расплачиваться эффективностью построения устройства.

Проектирование на немодифицированном C/C++

И в завершении этой главы рассмотрим методы проектирования, связанные с применением немодифицированного языка[1)] C/C++. На самом деле термин *немодифицированный,* или *чистый,* относится к промышленному стандарту языка C/C++, который минимально расширен типами данных, подобных применяемым в SystemC. Это позволяет задавать количество битов данных, отводимых для переменных и констант.

[1)] Во время написания этой книги, возможно, самым лучшим примером использования немодифицированного C/C++ в средствах проектирования стал программный продукт «*Precision C Synthesis*» компании Mentor (www.mentor.com). Также интересный программный комплекс *SPARK C-to-VHDL synthesis* разработан в Центре встроенных компьютерных систем Университета Калифорнии (www.cecs.uci.edu/~spark).

Несмотря на то что методы проектирования с применением немодифицированной версии языка C/C++ используются сравнительно недавно, их использование, как правило, имеет ряд преимуществ по сравнению с применением других C-подобных или традиционных Verilog/VHDL подходов:

- *Создать код на чистом C/C++ можно быстро и эффективно*: при использовании немодифицированного C/C++ для составления описания устройства полученный код не учитывает временных параметров и получается более компактным и простым для понимания, чем выполненные с помощью языков SystemC или расширенного C/C++, хотя и они на один-два порядка меньше, чем их аналоги, выполненные на уровне регистровых передач.
- *Верифицировать C/C++ код можно быстро и эффективно*: проверка модели без учета временных параметров, описанной на чистом C/C++, будет производиться намного быстрее, чем аналогичная процедура с моделью, реализованной с помощью SystemC или с помошью расширенного C/C++, и в 100...10000 раз быстрее, чем при использовании RTL-кода. В настоящее время модели, описанные на чистом C/C++ очень широко используются системными разработчиками для проверки достоверности алгоритмов и систем.
- *Оценить альтернативные варианты устройства можно быстро и эффективно*: при использовании моделей без учета временных параметров, созданных на чистом C/C++, очень удобно проводить модификацию и повторную проверку устройства, что является необходимым условием для оценки альтернативных микроархитектурных реализаций. Такое свойство облегчает команде разработчиков поиск наилучшего микроархитектурного решения. Это в свою очередь позволяет синтезировать более быстрые и компактные устройства, чем при использовании традиционных методов ручного RTL-кодирования.
- *Относительно просто вносить изменения в спецификации*: если в ходе проектирования в исходную спецификацию на устройство внесены некоторые изменения, то при использовании моделей без учета временных параметров, описанных с помощью чистого C/C++, можно достаточно просто реализовать и оценить влияние этих изменений на работу устройства, и затем продублировать изменения при окончательной его реализации.

Как уже упоминалось в этой главе, наиболее существенная проблема проектирования, основанного на языках SystemC и расширенном C/C++, заключается в том, что информация о реализации устройства жестко прописывается в модели. Вследствие чего модель становится зависимой от способа реализации.

Ключевым аспектом, разработки устройств на основе чистой, т. е. без учета временных параметров, версии языка C/C++ является то, что код, представляемый средствам синтеза, может быть написан людьми, заранее не представляющими себе способ аппаратной реализации или архитектуру устройства. Это значит, что C/C++-код, который системные инженеры создают сегодня, является идеальным источником для такой формы синтеза. Единственная модификация, которая обычно требуется для того чтобы модель, описанная на чистом C/C++, стала пригодной для средств синтеза, заключается в добавлении в исходный код простого специального комментария, который бу-

дет обозначать начало описания функциональной части устройства; всё, что концептуально находится до этой точки, рассматривается как часть средств тестирования.

В отличие от встраивания информации в исходный код, таким образом, ограничивая его целевой реализацией, эти же данные можно реализовать с помощью пользовательского управления алгоритмом синтеза (**Рис. 11.7**).

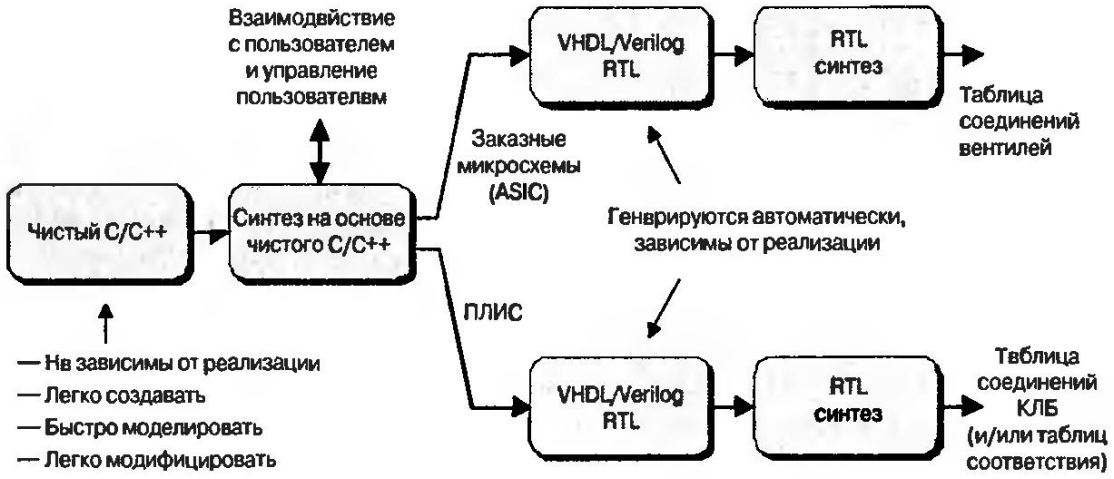

Рис. 11.7. Последовательность проектирования при использовании чистого C/C++

После обработки исходного кода средством синтеза, пользователь может использовать его для формирования оптимальных микроархитектурных соотношений и оценить их размеры и скорость. Средство синтеза анализирует код, определяет в нём различные логические конструкции и операторы вместе со связанными с ними данными и участками памяти, а также по возможности автоматически обеспечивает параллелизм. Средства синтеза также поддерживают графический интерфейс, который позволяет пользователю определять способы управления различными элементами. Например, интерфейс позволяет пользователю связывать порты с регистрами или блоками памяти; определять логические конструкции, такие как циклы, и позволяет устанавливать для каждого из них (при необходимости) режим полного или частичного интерпретирования; также позволяет определять, следует ли конвейеризировать циклы и другие структурные элементы. Кроме того, он помогает формировать распределение ресурсов по специфичным объектам и многое другое.

Средство синтеза на лету формирует оценки производимых действий и сообщает о размерах/площади и латентности в терминах тактовых импульсов и задержек ввода/вывода, или в пропускной способности канала, т. е. время/количество тактов для конвейеризованных устройств. Пользовательская конфигурация для каждого возможного сценария может быть именована, сохранена и при необходимости повторно использована. Иначе, было бы почти невозможно своевременно найти оптимальные соотношения, используя традиционные методы ручного RTL-кодирования.

Тот факт, что чистый, т. е. без учета временных параметров, исходный код C/C++, используемый средствами синтеза, не содержит какой-либо информации о реализации устройства и что вся эта информация обеспечивается за счёт управления средством синтеза, позволя-

ет легко перенастраивать исходный код на разные микроархитектуры и разные технологии реализации.

Как только пользовательские оценки будут сформированы, при нажатии кнопки **Go** («Начать») средство синтеза сгенерирует соответствующий RTL VHDL код. Этот код может быть использован средствами традиционного логического синтеза или приложениями физического синтеза для генерирования таблицы соединений, используемой средствами реализации (размещения и разводки).

Можно также синтезировать таблицу соединений вентилей непосредственно из исходного C/C++-кода (этот метод не показан на **Рис. 11.7**). Однако промежуточный RTL-код предоставляет разработчикам определенную зону комфорта, так как на этом этапе можно провести проверки и убедиться в правильности результатов, достигнутых в ходе трансляции из C/C++ в RTL.

Создание промежуточного RTL весьма полезно и с той точки зрения, что на этом уровне абстракции разработчики аппаратуры производят соединение различных функциональных блоков, из которых состоит устройство. Большая часть модулей современных устройств обычно представлена в виде отдельных блоков интеллектуальной собственности, выполненных на уровне регистровых передач. Это значит, что промежуточный RTL-код весьма полезен при разработке, так как позволяет производить интеграцию и проверку всей аппаратной части системы **Рис. 11.7**. Разработчики также могут воспользоваться всеми преимуществами существующих средств RTL-синтеза, которые очень хорошо продуманны, устойчивы к ошибкам и понятны пользователю.

1904 г. *Разработана первая ультрафиолетовая лампа.*

1904 г. *Разработан телефонный автоответчик.*

Уровни абстракций синтеза

Фундаментальное отличие различных подходов проектирования, основанных на языке C/C++ и рассматриваемых в этой главе, определяется уровнями абстракций, которые они поддерживают. Так, хотя SystemC предусматривает возможность моделирования на системном, алгоритмическом и транзакционном уровнях, его средства синтеза работают на относительно низком уровне абстракции. Аналогично, хотя описания на расширенном C/C++ более близки к чистому C/C++, чем SystemC, и их моделирование выполняется быстрее, их «синтезопригодность» далека от идеала.

Этот недостаток абстрактности синтеза становится причиной того, что описания устройств, проектируемых на SystemC и расширенном C/C++, т. е. описания с учетом временных параметров, зависят от способа реализации. Это, в свою очередь; усложняет разработку и модификацию устройств, а также значительно снижает их гибкость в оценке возможных вариантов реализации и перенастройки их на другие технологии (**Рис. 11.8**).

В отличие от них последнее поколение средств синтеза на основе чистого C/C++ поддерживает высокий уровень абстракции. Не зависящие от конкретной реализации модели C/C++ являются очень компактными, быстро и легко создаются и модифицируются. Используя само по себе средство синтеза, пользователь может легко и быстро формировать оценки возможных вариантов и перестраивать схему устройства на другие технологии реализации. В итоге проектирование на основе чистого C/C++ может значительно повысить скорость реализации и увеличить гибкость процесса проектирования по сравнению с проектированием на основе других версий языка C/C++.

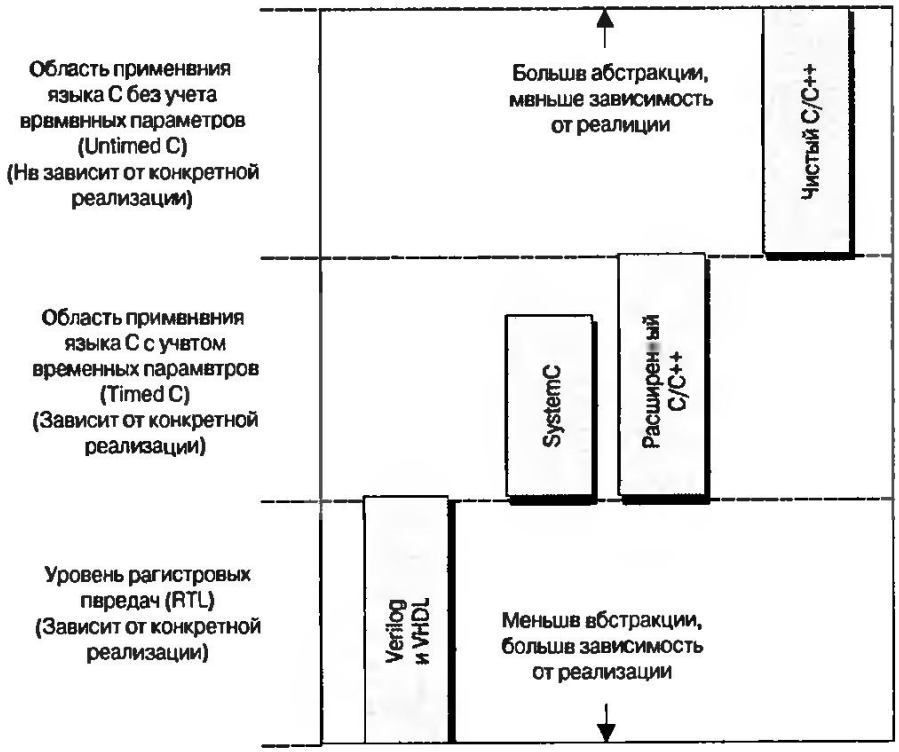

Рис. 11.8. Уровни абстракции синтеза языка C/C++

Системы мультиязычного проектирование и тестирования

Подводя итог, необходимо заметить, что ряд поставщиков САПР электронных систем выпускают системы многоязычного проектирования и тестирования, которые могут выполнять совместное моделирование устройств, описанных на разных уровнях абстракции.

В ряде случаев этот процесс может представлять собой вызов средствами моделирования Verilog связанных C/C++ моделей через его *интерфейс языка программирования (PLI — Programming Language Interface)* либо вызов из средств моделирования VHDL таких же функций через *интерфейс незнакомых слов (FLI — Foreign Language Interface)*. Аналогично, вы можете найти окружение для языка SystemC с возможностью работать с блоками, представленными в виде кода Verilog или VHDL.

Существуют очень сложные программные продукты, которые в графическом блочном редакторе отображают главные функциональные узлы устройства, при этом каждый узел может быть представлен с помощью языков:

- VHDL;
- Verilog;
- SystemVerilog;
- SystemC;
- Handel-C;
- Чистый C/C++.

Верхний уровень схемы может быть представлен традиционным кодом HDL, который вызывает другие модули, описанные на различных версиях HDL или на версия C/C++ (одной или нескольких). Верхний уровень схемы может быть также представлен одной из разновидностей языка C/C++, из которой происходит вызов субмодулей, написанных на других языках.

При таком подходе работа с описаниями модулей, написанных на языках VHDL, Verilog и SystemVerilog, обычно выполняется с помощью

одноядерного алгоритма моделирования. Затем этот алгоритм вместе с другим соответствующим алгоритмом одной из разновидностей языка C/C++ производит моделирование всего устройства. Кроме того, такие системы проектирования содержат встроенные отладчики исходного кода, который поддерживает различные версии языка C/C++; позволяют создавать тестирующие стенды, используя любой язык; поддерживают средства, такие как графический экран формы сигнала, или осциллограф, который способен отображать сигналы и переменные, связанные с любым из блоков вне зависимости от используемого языка[1)].

Для программиста, который по тем или иным причинам перестает заниматься своим делом, т. е. разработкой программного обеспечения, и начинает заниматься разработкой программных модулей или аппаратно-программным проектированием и тестированием, язык SystemC является своеобразной палочкой-выручалочкой.

В действительности, состав различных мультиязычных систем проектирования и тестирования меняется почти каждую неделю, поэтому перед выбором той или иной системы (среды) проектирования вам необходимо провести определённый анализ.

Всем противникам языка SystemC по секрету признаюсь, что этот язык рассматривается только применительно к проектированию ПЛИС. В этой сфере набор средств SystemC, используемых для получения действительной реализации устройств, пока не отличается сильной проработкой и сложностью. Однако приложения проверки и моделирования на системном уровне языка SystemC отличаются высокой эффективностью. Многие эксперты используют связку SystemC-SystemVerilog, при которой на системном уровне работают с SystemC, а затем с помощью SystemVerilog «конкретизируют» содержимое уровня реализации.

При разработке представления схемы с помощью одной из рассмотренных здесь версий языка C/C++ довольно часто на этом же языке создаются и наборы тестов. Эти тесты, как правило, используют языковые конструкции, которые не понятны многим низкоуровневым средствам, таким как преобразователь кода из C/C++ в RTL. В этом случае, как и раньше, придётся вручную переносить тестовые воздействия из C/C++ в VHDL/Verilog, с тем чтобы запустить их на системе моделирования RTL. Одним из преимуществ смешанных систем проектирования и тестирования заключается в том, что можно использовать тестовые воздействия языка C/C++ при работе средств моделирования на уровнях абстракции RTL и вентилей. Не исключено, что кое-что придётся подкорректировать, но это всё же намного легче, чем начинать всё с нуля.

[1)] Хорошим примером мультиязычных средств моделирования и контроля для ПЛИС и в меньшей степени для специализированных микросхем — могут служить продукты компании Aldec Inc. (www.aldec.com). Другим хорошим представителем средств этого класса может выступать пакет ModelSim® компании Mentor, который включает собственный модуль поддержки языка SystemC, таким образом, позволяя одноядерной системе моделирования работать с модулями VHDL, Verilog и SystemC.