ГЛАВА 9

ПРОЕКТИРОВАНИЕ НА ОСНОВЕ ЯЗЫКОВ ОПИСАНИЯ АППАРАТНЫХ СРЕДСТВ

Закат схемотехнического проектирования

С увеличением размеров и сложности заказных микросхем к концу 80-х наметился спад интереса к схемотехническим подходам проектирования. Визуализация, ввод, отладка, поддержка устройства на уровне вентилей стали трудным и неэффективным занятием, если в устройстве используется более 5000 логических вентилей, размещенных на нескольких страниц.

Кроме того, ввод схемы большого устройства на уровне вентилей часто приводил к ошибкам и являлся чрезвычайно трудоемким делом. Поэтому некоторые поставщики САПР электронных систем приступили к разработке средств и методов проектирования, основанных на языках описания аппаратных средств или *HDL (Hardware Description Language)*.

История развития HDL-проектирования

Идея, положенная в основу языков описания аппаратных средств, как это не удивительно, заключается в том, что с их помощью можно описать работу аппаратных средств. В более широком понимании, термин «аппаратный» используется для описания любых физических узлов и компонентов электронной системы, включая микросхемы, печатные платы, системные блоки, кабели, и даже гайки и болты, с помощью которых собрано изделие. Однако в контексте языков HDL понятие «аппаратный» имеет отношение только к электронным узлам, т. е. компонентам и проводникам, микросхем и печатных плат. HDL может также использоваться для ограниченного представления кабелей и соединителей, связывающих вместе печатные платы.

На заре развития электроники почти каждый разработчик САПР изобретал свою версию языка HDL. Одни из них представляли собой аналоговые версии HDL, так как описывали работу аналоговых систем, в то время как другие концентрировали своё внимание на описании цифровой функциональности. Нас интересуют только версии HDL, которые применяются в процессе проектирования ПЛИС и заказных микросхем.

Уровни абстракции

Наиболее известные версии «цифровых» языков HDL будут рассмотрены в этой главе, но не сейчас. А сейчас давайте сосредоточимся на том, как некий абстрактный «цифровой» язык описания аппаратных средств (HDL) используется в проектировании цифровых микросхем. В первую очередь следует обратить внимание на то, что функциональные возможности цифровых микросхем могут быть представлены на различных уровнях абстракции. Эти уровни в большей или меньшей степени поддерживаются различными версиями HDL (**Рис. 9.1**).

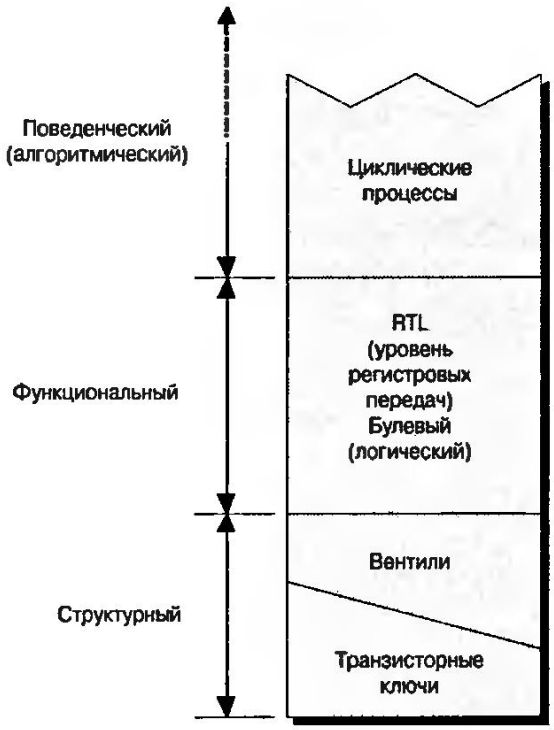

Рис. 9.1. Уровни абстракции

Самым низким уровнем абстракции цифровых HDL является *уровень транзисторных ключей*, который определяется способностью описывать схему в виде таблицы соединений транзисторных ключей. Выше него находится *уровень вентилей*, который описывает схему в виде таблицы соединений простых логических вентилей и функций. Поэтому первые версии форматов таблиц соединений логических вентилей, формируемые с помощью программ ввода принципиальных схем, о которых говорилось в предыдущей главе, представляли собой, по существу простейшие версии языков описания аппаратных средств.

Оба уровня, транзисторных ключей и логических вентилей, можно отнести к *структурному* представлению устройства. Однако следует обратить внимание на то, что слово «структурный» имеет несколько значений, так как оно может также относиться к иерархической таблице соединений блоков устройства, где содержание каждого блока определено на любом уровне абстракции, изображенном на **Рис. 9.1**.

Следующий, более сложный уровень HDL, определяется способностью поддерживать *функциональные* представления устройства, которые включают в себя набор конструктивов. Нижняя граница этого уровня определяется способностью описывать функцию с помощью булевых (логических) выражений. Например, если представить, что имеется набор сигналов, которые называются Y, SELECT, DATA-A и DATA-B, то можно описать функциональность простого 2:1 мультиплексора с помощью следующего булевого выражения:

```
Y = (SELECT & DATA-A) | (!SELECT & DATA-B);
```

Это выражение записано с помощью обычного синтаксиса, который не поддерживается ни одной конкретной версией HDL и используется только в качестве примера. Как уже отмечалось, символ «**&**» представляет собой логическую функцию **И**, символ «**|**» соответствует логической функции ИЛИ, а символ «$\bar{}$» отображает функцию НЕ (*см.* гл. 3).

Функциональный уровень абстракции включает в себя представления *уровня регистровых передач* (RTL — *register transfer level)*. Термин RTL подразумевает большое количество различных проявлений. Са-

мый простой способ понять его основную концепцию — представить устройство в виде совокупности регистров, связанных между собой элементами комбинационной логики. Эти регистры часто управляются с помощью общего синхросигнала. Допустим, что мы располагаем двумя сигналами, называемыми CLOCK и CONTROL, а также набором регистров, обозначенных как REGA, REGB, REGC и REGD. Тогда выражение RTL может иметь следующий вид:

```
when CLOCK rises
            if CONTROL == "1"
                  then REGA = REGB & REGC;
                  else REGA = REGB | REGD;
            end if;
end when;
```

В этом случае команды *when, rises, if, then, else* и им подобные являются ключевыми словами, семантика которых определяется версией языка HDL. Этот пример записан с помощью обычного синтаксиса, который не поддерживается ни одной конкретной версией HDL и используется только в качестве примера.

Высшим уровнем абстракции считается *поведенческий*, который поддерживается современными версиями HDL и обозначает возможность описывать поведение схемы, используя абстрактные логические структуры, например циклы и процессы. Этот уровень также подразумевает использование в выражениях алгоритмических элементов, таких как сумматоры и умножители. Например:

```
Y = (DATA-A + DATA-B) * DATA-C;
```

Имеется еще и *системный* уровень абстракций (он не показан на **Рис. 9.1**), который характеризует конструктивы, предназначенные для приложений системного уровня проектирования, но о них мы поговорим немного позже.

Многие ранние цифровые версии HDL понимали только структурные описания устройств в форме таблиц соединений вентилей или транзисторных ключей. Другие представители этого семейства языков, например ABEL, CUPL или PALASM, использовались для описания функциональности программируемых логических устройств (ПЛУ). Эти языки поддерживали различные уровни функционального уровня абстракции, такие как булевы выражения, текстовые таблицы истинности, текстовые описания конечных автоматов (*см.* гл. 3).

Следующее поколение языков HDL, которые изначально предназначались для систем логического моделирования, поддерживало более сложные уровни абстракции, например RTL и некоторые поведенческие конструкции (имеются в виду языковые конструкции, описывающие поведенческие аспекты системы или устройства). Как будет показано ниже, эти версии языков описания аппаратных средств стали основой для первых по-настоящему HDL-ориентированных технологий проектирования.

1847 и 1854 гг. *Самоучка Джордж Буль (George Boole) внес значительный вклад в ряд областей математики, но был увековечен благодаря двум своим работам, в которых он представил логические выражения в математической форме. Теперь эта форма называется булевой алгеброй.*

***В* 1938 году** *Клод Шеннон (Claude Shannon) опубликовал статью, в которой показал, как булевы концепции могут быть использованы для реализации функций переключения в электронных цепях.*

Простые HDL-методы проектирования заказных микросхем

Ключевой особенностью *HDL-проектирования* заказных микросхем является использование в технологическом цикле проектирования методов логического синтеза, средства автоматизации которых начали появляться на рынке в середине 80-х годов. Эти средства могли воспринимать RTL-описания устройств вместе с набором временных

ограничений. Временные ограничения описывались в файле, который содержал выражения, описывающие работу устройства, дополненные строками вида «максимальная задержка распространения сигнала от входа *A* до выхода *B* должна быть не больше, чем *N* наносекунд». На самом деле формат этой строки выглядел более сжато и скучно.

Средства логического синтеза автоматически конвертировали RTL-описания устройств в набор регистров и булевых выражений, попутно реализуя различные процедуры минимизации и оптимизации, в том числе оптимизацию по временным задержкам и по площади используемого места на кристалле. После этого средства синтеза генерировали таблицы соединений вентилей, которые должны были соответствовать, и соответствовали, начальным временным ограничениям **(Рис. 9.2)**.

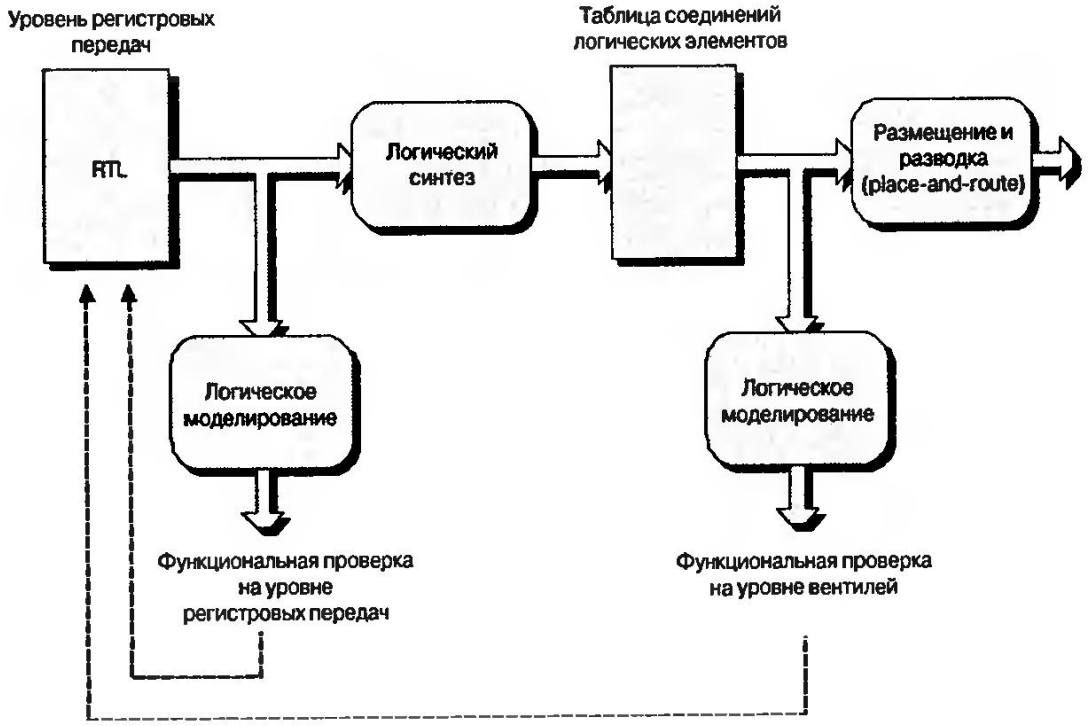

Рис. 9.2. Цикл HDL-проектирования заказных микросхем (простой)

Подход обладал рядом преимуществ. Во-первых, существенно повысилась производительность труда разработчиков, так как стало намного проще определять, понимать, обсуждать и отлаживать функциональность устройств, описание которых выполнено на уровне регистровых передач, чего нельзя было осуществить, работая с пачкой чертежей принципиальных схем. Во-вторых, средства логического моделирования работали с RTL-моделями намного быстрее, чем со схемами соединений вентилей.

Незначительная проблема возникала при использовании систем логического моделирования и заключалась в том, что хотя эти средства и позволяли работать с моделями, описанными на высоких уровнях абстракции, с использованием поведенческих конструкций, но первые версии средств синтеза воспринимали описание функциональности устройства только до уровня регистровых передач. Вследствие этого инженеры были вынуждены выбирать для работы «синтезопригодные» подмножества HDL.

После генерации средствами синтеза таблицы соединений вентилей дальнейшая процедура проектирования повторяла схемотехнические методы, рассмотренные в предыдущей главе. Таблица соединений

вновь подвергалась процедуре моделирования, чтобы убедиться в её функциональной достоверности, а также применялась для формирования результатов временного анализа, основанного на значениях параметров проводников и других элементов схемы. Затем таблица соединений использовалась в качестве исходных данных для работы утилит размещения и разводки элементов, после которого выполнялся более точный временной анализ с использованием полученных значений сопротивлений и ёмкостей.

Простые HDL-методы проектирования ПЛИС

Потребовалось какое-то время, прежде чем HDL-проектирование расцвело пышным цветом среди поклонников заказных микросхем. В те времена разработчики всё ещё только осваивали принципы ПЛИС. В связи с этим лишь только в начале 90-х в мир ПЛИС пришли технологии HDL-проектирования, в частности средства логического синтеза **(Рис. 9.3)**.

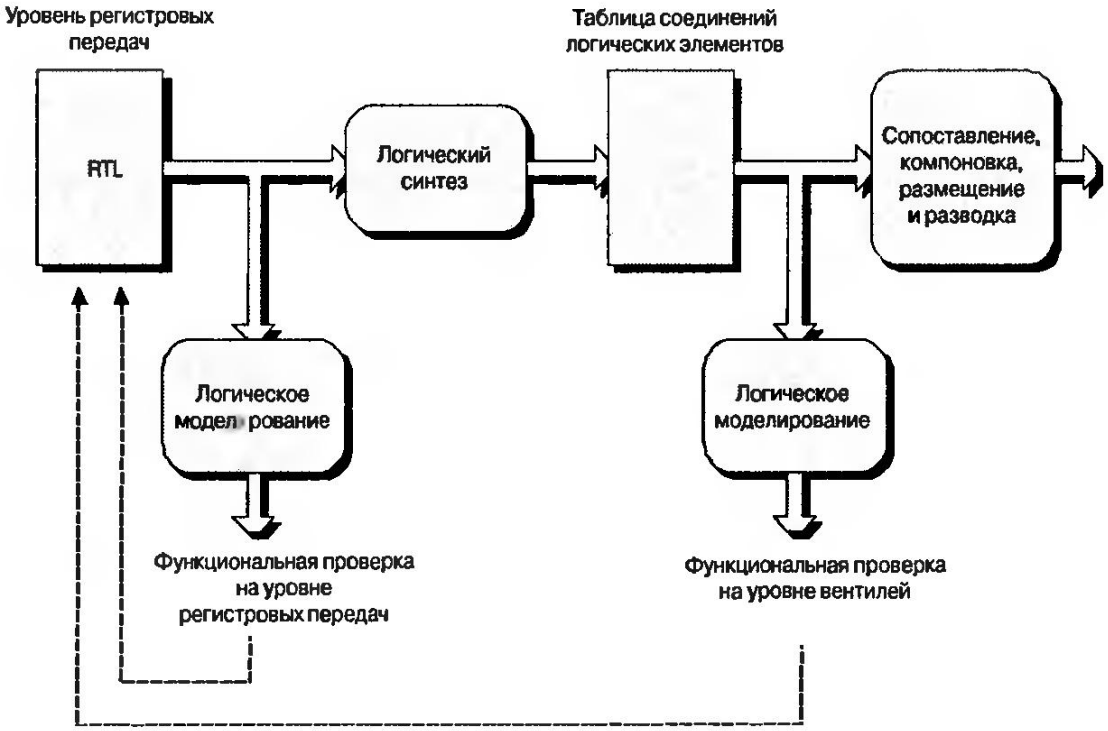

Рис. 9.3. Цикл HDL-проектирования ПЛИС (простой)

Аналогично случаю с заказными микросхемами после генерации таблицы соединений вентилей средствами синтеза, технология проектирования сильно напоминала схемотехнические методы создания ПЛИС, описанные в гл. 8. Таблица соединений вновь подвергалась процедуре моделирования, чтобы убедиться в её функциональной достоверности, а затем подвергалась временному анализу на основе оценок параметров проводников и других элементов схемы. После этого таблица соединений обрабатывалась средствами сопоставления, компоновки, размещения и разводки, и вновь проводился более точный временной анализ, основанный на реальных, т. е. физических, параметрах устройства.

Архитектурное проектирование ПЛИС

1878 г. *Англия. Джозеф Сван (Sir Joseph Wilson Swan) изобрел электрическую лампу накаливания.*

Главная проблема ранних подходов к проектированию ПЛИС заключалась в том, что их средства логического синтеза вышли из мира заказных микросхем. Другими словами, эти средства «мыслили» в понятиях простейших вентилей и регистров. Это значит, что они формировали таблицу соединения вентилей и передавали её программным модулям поставщиков ПЛИС для выполнения операций сопоставления, компоновки, размещения и разводки.

Примерно в 1994 году средства синтеза «вооружились знаниями» о различных принципах построения (архитектурах) ПЛИС. Это позволило им самостоятельно выполнять операции сопоставления и, в некоторой степени, компоновки, а также формировать таблицу соединений логических блоков и таблиц соответствия. Впоследствии эта таблица соединений могла использоваться средствами размещения и разводки из инструментария, предоставляемого поставщиками ПЛИС. Главное преимущество этого метода заключалось в том, что средства синтеза располагали лучшими методами оценки временных параметров и занимаемой площади, что обеспечивало более высокое качество результатов. На практике проектирование ПЛИС с помощью архитектурных методов синтеза производилось на 15...20% быстрее, чем при использовании традиционных (на уровне вентилей) средств синтеза.

Логический и физический синтез

Здесь мы несколько опережаем события, так как именно сейчас как нельзя уместно упомянуть о содержимом данной части книги. Первые версии программ логического синтеза, которые появились в середине 80-х годов, предназначались для проектирования заказных микросхем. В этих устройствах задержка на вентилях, намного превышала задержку, вносимую проводниками, соединяющими эти вентили. Кроме того, количество этих вентилей было относительно мало (по современным меркам), а для их управления применялась относительно низкая тактовая частота. Поэтому первые версии программ логического синтеза могли использовать относительно простые алгоритмы для оценки задержки сигнала на проводниках. При этом полученные оценки несущественно отличались от действительных значений, получаемых после размещения и разводки элементов.

Спустя годы, сложность и размеры (выраженные в количестве вентилей) заказных микросхем значительно увеличились. Одновременно с этим размеры структур на кремниевом кристалле уменьшались, что привело к следующим последствиям:

- эффекты, вызванные задержками, в общем случае становилось более сложными;
- задержки, на проводниках, начали перевешивать задержки на вентилях.

К середине 90-х, в связи с ростом размеров заказных микросхем, также существенно усложнились эффекты, вызванные задержкой. Сложность этих эффектов существенно превышала уровень, на который рассчитывались первые средства логического синтеза. В результате это привело к тому, что оценки задержек, полученных с помощью средств логического синтеза, очень плохо соотносились с данными, полученными после размещения и разводки элементов. В свою очередь это значит, что достижение *состояния временного соответствия*

1878 г. *Англия. Вильям Крукс (William Crookes) разработал электронную «трубку Крукса».*

(доводка устройства для достижения заданных временных характеристик) становилась всё более трудной и длительной задачей.

Поэтому, примерно в 1996 году, в процессе проектирования специализированных микросхем начали применять средства *физического синтеза*. Способы работы физического синтеза более подробно будут рассмотрены в гл. 19. Здесь лишь заметим, что в процессе, физического синтеза производится начальное размещение логических вентилей и функций. Основываясь на результатах размещениях, средства временного анализа позволяют получить более точные временные оценки.

В конечном счете, средства физического синтеза формировали таблицу соединений размещенных (но не разведенных) вентилей. Средства физической реализации (размещения и разводки) заказных микросхем использовали эту начальную информацию о размещении в качестве стартовой точки, от которой выполнялась локальная (точная) оптимизация размещения, после чего производилась разводка элементов. В результате применения средств физического синтеза существенно повысилась точность оценок задержек. В свою очередь это значило, что достижение временного соответствия стало менее дорогостоящим.

Ну, а что там с ПЛИС? Размеры и сложность этих устройств также увеличивались на протяжении 90-х годов. К концу прошлого тысячелетия проектировщики ПЛИС столкнулись со значительными проблемами, связанными с временным соответствием. Поэтому, примерно в 2000 году, поставщики САПР электронных устройств начали предлагать ПЛИС-ориентированные средства физического синтеза, которые могли выдавать сопоставленную, скомпонованную и размещенную таблицу соединений КЛБ и таблиц соответствия (LUT/CLB-level netlist). В этом случае средства физической реализации ПЛИС (модули размещения и разводки — place-and-route) используют эту информацию о начальном размещении в качестве стартовой точки, от которой выполняется процедура локальной (точной) оптимизации размещения, после чего производится детальная разводка элементов устройства.

Средства графического ввода продолжают жить

Когда появились первые методы проектирования, основанные на языках описания аппаратных средств (HDL), многие предрекали средствам *графического ввода* и визуализации незавидную участь: в скором времени они должны были исчезнуть навсегда. И правда, одно время многих инженеров прямо распирало от гордости при мысли, что они используют текстовые редакторы, подобные *VI*[1] (сокращенно от Visual Interface) или EMACS, в качестве единственного средства описания принципиальной схемы.

Но, как говорится, лучше один раз увидеть, чем сто раз услышать. Другими словами, одна картинка может заменить тысячи слов, и средства графического ввода и визуализации остались популярными на многих этапах проектирования. Например, редакторы схем широко применяются для ввода структурных схем, представляющих собой совокупность соединенных вместе блоков высокого уровня. На основании этой схемы система может автоматически создать скелет текстового HDL-описания с объявленными названиями блоков, входов и выходов. И наоборот, пользователь может самостоятельно создать скелет текстового HDL-описания устройства, а затем система на основе этого описания автоматически построит структурную схему устройства.

---

[1] Редактор *VI* — чрезвычайно мощный, но трудный для понимания редактор.

С точки зрения пользователя, щелчок мышью на одном из этих блоков структурной схемы может автоматически открыть HDL-редактор. В качестве такого редактора может выступать простой текстовый редактор, запускаемый из командной строки, подобный VI, или можно использовать более сложный специализированный HDL-редактор со встроенной функцией выделения различными цветами ключевых слова языка, автоматическим завершением выражений и другие.

Кроме того, щелкнув мышью на одном из таких блоков современных систем проектирования можно сделать выбор между описанием и просмотром содержимого этого блока, блоков нижнего уровня, исходного HDL-кода, графической диаграммы состояний, используемой для описания конечных автоматов, графической блок-схемы и так далее. Для графических представлений, подобных графическим диаграммам состояний и блок-схемам, существует возможность автоматически генерировать их описания на уровне регистровых передач (**Рис. 9.4**).

1878 г. *Ирландия. Денис Редмонд (Denis Redmond) продемонстрировал эффект ввода изображений с помощью фотоэлемента, изготовленного из селена.*

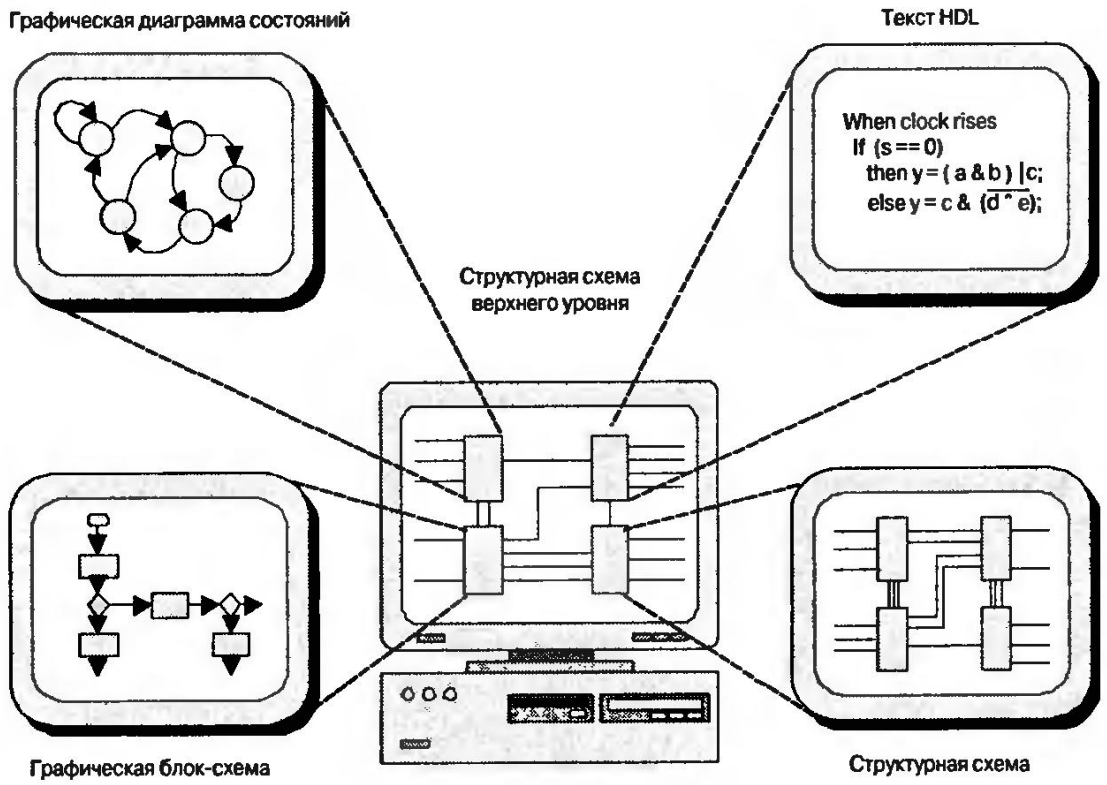

Рис. 9.4. Многоуровневая система ввода схем

Инженеры в процессе разработки довольно часто используют табулированный файл, содержащий информацию о внешних входах и выходах устройства. При этом блок-схема верхнего уровня устройства и табулированный файл будут напрямую связаны одними и теми же данными, и на самом деле будут представлять различные интерпретации этих данных. При изменении любой из этих интерпретаций система автоматически модифицирует все связанные с этой модификацией данные и немедленно отобразит внесённые изменения в других интерпретациях.

Типы языков HDL

Для определенного круга людей жизнь могла быть предельно простой, если существовала бы только одна версия языка HDL. Но никто не может сказать, что жизнь — простая штука. Как уже упоминалось, на заре развития цифровых микросхем, где-то в 70-х годах, каждый

разработчик HDL-средств проектирования обычно испытывал непреодолимое желание заодно создать собственный язык описания аппаратных средств. И не удивительно, что в результате в этой области возникла такая неразбериха. Даже невозможно себе вообразить весь ужас этой ситуации. Решением проблемы мог бы стать промышленный стандарт языка HDL, но где же его было взять?

Verilog HDL

В середине 80-х Фил Морби (Phil Moorby), один из создателей знаменитой системы логического моделирования HILO, разработал новую версию языка HDL под названием Verilog. В 1985 году компания Gateway Design Automation, в которой работал Фил Морби, выпустила этот язык на рынок вместе с сопутствующей его системой логического моделирования, называемой Verilog-XL.

Verilog и Verilog-XL поддерживали весьма прогрессивную концепцию, называемую *интерфейсом языка программирования* или *PLI Verilog* (*Programming Language Interface*). В общем виде эта концепция известна как *интерфейс прикладного программирования* или *API (Application Programming Interface)*. API представляет собой библиотеку программных функций, которая позволяет внешнему программному обеспечению передавать данные в приложение и получать из него выходные результаты. Другими словами, функции API Verilog позволили пользователям расширить функциональность этого языка и поставляемого с ним пакета логического моделирования.

В качестве простого примера рассмотрим ситуацию, когда инженер разрабатывает схему, в которой должен использоваться существующий модуль, выполняющий математические функции, например быстрое преобразование Фурье (БПФ). Представление этой функции, выполненное на языке Verilog, потребует довольно много времени для работы системы моделирования, что может вызвать задержку разработки последующих модулей устройства. В таком случае разработчики системы могут создать модель этой функции на языке программирования C, которая будет моделироваться примерно в 1000 раз быстрее, чем её Verilog-эквивалент. Такая модель должна содержать структуры PLI, через которые осуществляется связь со средой моделирования. Впоследствии подобные модели могут вызываться из Verilogописания других модулей устройства при помощи обращения к соответствующим функциям PLI, которые обеспечивают двунаправленную передачу данных между главной схемой (представленной в коде Verilog) и БПФ (описанным на языке C).

Другая, действительно полезная особенность, связанная с Verilog и Verilog-XL, заключалась в возможности получать информацию о временных параметрах устройства из внешнего текстового файла, сформированного в SDF-формате (*Standart Delay Format — формат стандартных задержек*). Это свойство позволяло некоторым средствам, например приложению, выполняющему временной анализ после процедуры размещения и разводки, создавать SDF-файл, который затем мог быть использован средствами моделирования для обеспечения более точных результатов.

Как язык программирования, Verilog обладал достаточной мощью при работе на структурном (вентили и ключи) уровне абстракции (особенно в отношении моделирования задержек), отличался большой мощью на функциональном (булевы выражения и RTL) уровне абстракции и поддерживал некоторые поведенческие (алгоритмические) конструкции (**Рис. 9.5**).

Рис. 9.5. Уровни абстракции (Verilog)

1879 г. *Америка. Томас Эдисон (Thomas Alva Edison) разработал лампу накаливания (на год позже, чем Джозеф Сван из Англии).*

В 1989 году компания Cadence Design Systems приобрела компанию Gateway Design Automation и заодно язык Verilog, а также систему моделирования Verilog-XL. По наиболее вероятному сценарию того времени в результате этой процедуры язык Verilog должен был просто сменить своего владельца. Однако неожиданно для всех в 1990 году компания Cadence представила спецификации VerilogHDL, VerilogPLI и VerilogSDF для открытого использования.

Это было довольно смелое решение, так как после этого любой желающий мог развивать средства моделирования, основанные на языке Verilog, тем самым становясь потенциальным конкурентом Cadence. Причина такой щедрости этой компании заключалась в растущей популярности языка VHDL. Другой причиной открытия Verilog для публичного использования было то, что огромное количество компаний, развивающих средства проектирования на основе HDL, такие как средства логического синтеза, теперь ощутили достоинства использования языка Verilog.

С появлением единого представителя, который мог быть использован средствами моделирования, синтеза и так далее, для многих разработчиков жизнь значительно упростилась. Однако не стоит забывать, что Verilog изначально разрабатывался совместно со средствами моделирования, а средства синтеза при этом находились в стороне. Это значило, что при создании устройства, инженеры были вынуждены использовать *подкласс средств синтеза* используемого ими языка, который примерно можно было определить, как некую совокупность языковых конструкций, понятную и поддерживаемую приложением синтеза, выбранным пользователем.

Официальные выпуски Verilog включали в себя документы, называемые *справочным руководством по языку* или *LRM (Language Reference Manual)*, которые описывали *синтаксис* и *семантику* языка. В данном случае термин *синтаксис* относится к грамматике языка и определяет порядок слов и символов в выражениях, а *семантика* определяет смысл этих слов и символов, и отношения между понятиями, которые они описывают.

1879 г. *Англия. Вильям Крукс (William Crookes) установил, что катодные лучи в трубке могут быть отрицательно заряженными частицами.*

В идеале справочные страницы LRM должны довольно строго определять понятия языка, исключая какие-либо неверные или двусмысленные толкования. В действительности, однако, Verilog LRM допускает некоторую неопределенность. Эти неопределенности возникали в выражениях вида «если управляющий сигнал на триггере переходит в неактивное состояние, а в это же время происходит переключение синхросигнала, то какой из этих сигналов сработает первым?» Это приводило к тому, что в различных версиях средств моделирования могли генерироваться различные результаты, что зачастую и приводило пользователя в замешательство.

Verilog быстро стал весьма популярным. Отдельные компании начали развивать язык в разных направлениях, что привело к определенным сложностям. В связи с этим в 1991 году была создана некоммерческая организация Open Verilog International (OVI), которая выполняла функции координации и стандартизации Verilog HDL и Verilog PLI и пользовалась поддержкой ведущих поставщиков САПР электронных систем.

После этого популярность языка Verilog стала возрастать по экспоненте, и в конечном итоге OVI подала запрос в Институт инженеров по электротехнике и радиоэлектроники (IEEE) о создании группы поддержки языка Verilog. Такой комитет был сформирован в 1993 году и стал известным под названием IEEE 1364. Май 1995 года оказался свидетелем первого официального выпуска IEEE Verilog, официальное название которого звучит так: 1364-1995, а неофициальное так: Verilog 95.

В 2001 году в этот стандарт были внесены некоторые изменения, поэтому новую версию часто называют Verilog 2001 (или Verilog 2K1). Во время написания этой книги комитет IEEE 1364 усиленно работал над следующей версией Verilog 2005, а весь конструкторский мир, затаив дыхание, ожидал выхода нового релиза.

VHDL и VITAL

В 1980 году Министерство обороны США приступило к разработке программы *сверхбыстрых схем с высоким уровнем интеграции* или *VHSIC (Very High Speed Integrated Circuit)*. Эта программа предназначалась для развития современных технологий изготовления цифровых микросхем. Программа предназначалась также для решения проблемы воспроизведения микросхем и печатных плат военного оборудования, которое длительное время находилось в эксплуатации. Эта проблема возникла в связи с отсутствием подробной документации на составные части данного оборудования.

Кроме того, различные части системы зачастую разрабатывались и тестировались с помощью разнотипных и несовместимых средств разработки и моделирования.

Для решения этих проблем в ходе реализации проекта, в 1981 году, был разработан новый язык аппаратных средств, названный *VHSIC HDL*, или кратко *VHDL*. Примечательно, что на ранних стадиях разработки этого языка в процесс были вовлечены промышленные компании. В 1983 году ряд компаний-производителей, включая Intermetrics, IBM и Texas Instruments получили контракт на развитие языка VHDL, первая версия которого вышла в 1985 году.

Чтобы получить признание промышленности, в 1986 году Министерство обороны передало права на язык VHDL Институту инженеров по электротехнике и радиоэлектроники (IEEE). После выполнения некоторых модификаций, предназначенных для решения ряда про-

блем, в 1987 году VHDL вышел в качестве официального стандарта IEEE 1076. Последующие модифицированные версии этого языка выходили в 1993 и 1999 годах.

Язык VHDL отличался большой мощностью на функциональном (булевы выражения и RTL) и поведенческом (алгоритмическом) уровнях абстракции, а также поддерживал некоторые конструкции системного уровня. Однако он не оправдал своих возможностей при работе на структурном (вентили и ключи) уровне абстракции, особенно в части моделирования задержек.

Изначально язык VHDL не поддерживал библиотеку, аналогичную Verilog PLI. Современные системы моделирования располагают собственными методами обмена данными, например, такими как *ModelSim foreign language interface* или *FLI* (инородный языковый интерфейс). Будем надеяться, что подобные средства, в конечном счете, станут стандартом.

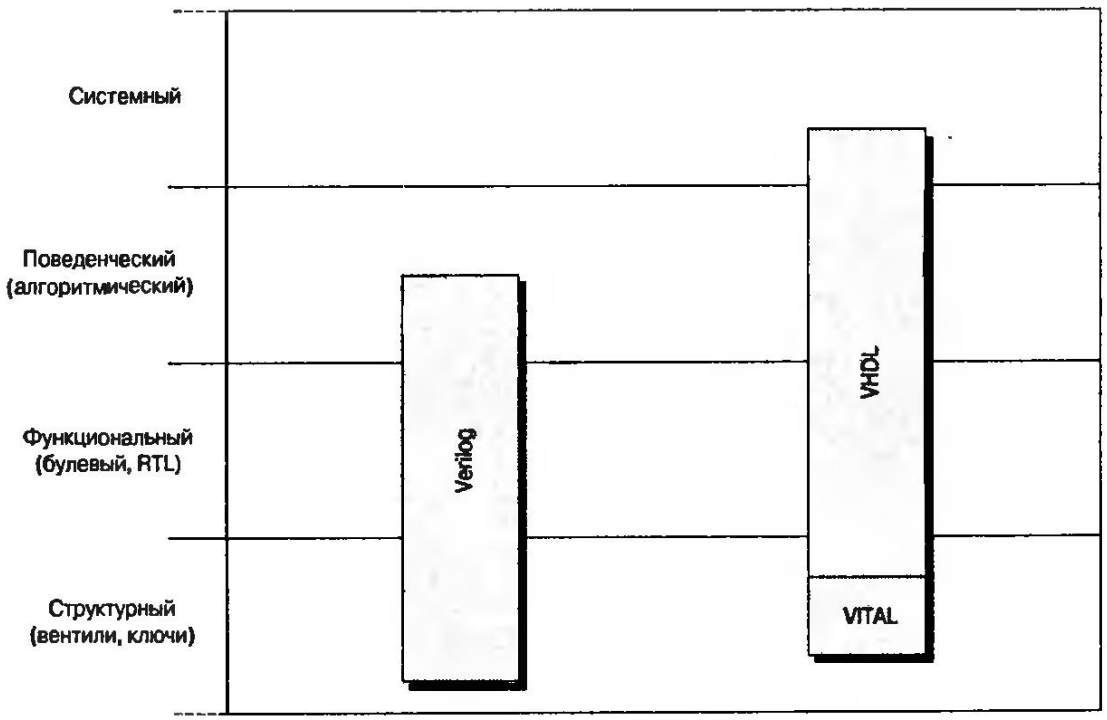

Рис. 9.6. Уровни абстракции: Verilog и VHDL

Очень скоро стало очевидно, что VHDL обладает недостаточной временной точностью для использования в качестве завершающего моделирующего средства. По этой причине в 1992 году на конференции по САПР электронных систем была представлена VHDL-библиотека начального уровня для заказных микросхем или *VITAL (VHDL Initiative toward ASIC Libraries)*. Библиотека VITAL являлась попыткой повышения возможностей языка VHDL для применения его в системах временного моделирования при проектировании ПЛИС и заказных микросхем. В итоге в её состав вошла библиотека простых функций, используемых в ПЛИС и заказных микросхемах, и модуль обратной корректировки информации о задержке. Механизм задержки основывался на тех же форматах, что и язык Verilog (**Рис. 9.6**).

1880 г. *Америка. Александр Белл запатентовал оптическую телефонную систему, названную фотофоном.*

Проектирование с применением нескольких языков

Давным-давно для описания целого устройства, как правило, использовался один язык описания аппаратных средств: Verilog или VHDL. По мере увеличения размеров и сложности устройств для разработки различных частей устройства стали привлекаться разные команды разработчиков. Эти команды могли быть из разных компаний или даже проживать в разных странах. Не удивительно, что в процессе разработки разные группы использовали разные языки проектирования.

Закон Мэрфи гласит: если неприятность может случиться, она обязательно случается. Капитан и инженер Эдварду Мэрфи (Edward Murphy) работал на авиабазе Эдвардс в 1949 г.

Вместе с тем, все чаще используются уже готовые блоки или блоки интеллектуальной собственности, приобретенные у сторонних разработчиков. И очень часто, как по законам Мэрфи, если в процессе разработки использовался один язык, приобретённый блок интеллектуальной собственности наверняка оказывался написанным на другом языке.

В начале 90-х началась так называемая война HDL, когда сторонники одного языка Verilog или VHDL предсказывали неминуемую кончину другого языка... Но годы идут, а оба языка по-прежнему существуют и развиваются. В конечном итоге поставщики САПР электронных устройств стали поддерживать многоязычные программные продукты, в число которых входили модули логического моделирования, синтеза и другие средства, которые могли работать с устройствами, состоящими из блоков, описанных с помощью и VHDL, и Verilog.

UDL/I

Как уже отмечалось, язык Verilog изначально создавался для решения задач моделирования. Аналогично VHDL создавался как язык описания с учетом необходимости моделирования. В результате оба этих языка можно было использовать для описания и последующего моделирования систем, но они не были предназначены для синтеза.

Для решения этих проблем в 1990 году Японская ассоциация развития электронной промышленности (JEIDA) представила собственную версию языка HDL, которая называлась *унифицированный язык разработки интегральных микросхем* (или *UDL/I — Unified design language for integrated circuits*).

Главным преимуществом этого языка являлось то, что он изначально создавался как для обеспечения моделирования, так и для обеспечения синтеза. Окружение UDL/I включало в себя системы моделирования и синтеза, к тому же он распространялся бесплатно (в том числе и с исходными кодами). Однако когда UDL/I вышел в свет, языки Verilog и VHDL уже стали настолько популярными, что UDL/I не вызвал особого интереса у инженеров за пределами Японии.

Superlog и SystemVerilog

В 1997 году с появлением компании Co-Design Automation ситуация в сфере языков описания аппаратных средств стала накаляться. Разработчики компании усердно корпели над созданием продукта под названием Verilog on steroids, который впоследствии назвали Superlog.

Superlog был забавной штукой, сочетающий простоту языка Verilog и мощь языка программирования C. Он содержал такие вещи, как временная логика, возможность верификации сложных схем, динамические библиотеки API и концепцию *утверждений* — ключевое понятие

стратегии формальной верификации, известной также как *model checking* (проверка модели). (VHDL также поддерживал простые структуры утверждений, но оригинальная версия языка Verilog не могла этим похвастаться.)

У Superlog было две проблемы, во-первых, он был совершенно обособленным языком, во-вторых, он был намного сложнее, чем Verilog 95 и Verilog 2001. Это потребовало от поставщиков САПР электронных систем дополнительных усилий для его поддержки.

Тем временем, пока многие пребывали в раздумьях о том, что их ожидает в будущем, группа OVI наладила отношения с аналогичной организацией по поддержке языка VHDL, которая называлась VHDL International, чтобы сформировать новую ассоциацию, которая теперь называется Accellera. Функциями этой организации были: определение новых форматов и стандартов, их развитие, поддержка новых методов проектирования, основанных на этих форматах и стандартах.

Летом 2002 года Accellera представила спецификацию на новый гибридный язык, названный SystemVerilog 3.0. (Только не спрашивайте меня о версиях 1.0 и 2.0.) Новый язык отличался большими возможностями, чем существующие версии Verilog, и превзошел нашумевшую когда-то реализацию языка Superlog. Действительно, SystemVerilog 3.0 поддерживал большинство форматов и структуру языка Superlog, предоставленных компанией Co-Design Automation. Кроме того, он поддерживал *утверждения* и возможности расширенного синтеза. Наконец-то свершилось то, о чем все давно мечтали. SystemVerilog 3.0 являлся также стандартом компании Accellera, что способствовало его быстрому и широкому распространению.

В 2002 году Co-Design Automation была приобретена компанией Synopsys, которая продолжила политику передачи логической структуры языка Superlog в SystemVerilog, но теперь уже никто не говорил, что Superlog — независимый язык. После некоторых раздумий все основные поставщики САПР электронных систем официально приняли SystemVerilog и расширили свои продукты для поддержки этого языка в той или иной степени в зависимости от области применения и предъявленных требований. Летом 2003 года появился SystemVerilog 3.1, и вслед за ним, в начале 2004 года, вышла следующая версия 3.1a, в которую было добавлено несколько дополнительных возможностей и исправлены некоторые ошибки. Тем временем, Институт инженеров по электротехнике и радиоэлектроники (IEEE) начал работу над новой версией Verilog 2005. Для предотвращения раскола между Verilog 2005 и SystemVerilog. компания Accellera обещала к лету 2004 передать свои права на SystemVerilog институту IEEE.

1880 г. *Франция. Пьер и Жак Кюри (Pierre Currie, Jacques Currie) открыли пьезоэлектричество.*

SystemC

Давайте рассмотрим язык SystemC, к которому одни разработчики относятся с большой любовью, а другие просто терпеть не могут. Язык SystemC можно использовать для описания схем устройств на уровне абстракции RTL (*см.* гл. 11). Моделирование этих описаний может быть выполнено в 5...10 раз быстрее, чем моделирование эквивалентных описаний на Verilog или VHDL, а средства синтеза способны преобразовывать SystemC RTL в таблицы соединений вентилей.

Весомым аргументом в пользу языка SystemC является то, что SystemC обеспечивает более естественную среду для совместного программно-аппаратного проектирования и проверки. В качестве аргумента против SystemC выступал тот факт, что большинство разработ-

SystemC может поддерживать более высокие уровни абстракции, чем RTL, но их рассмотрение выходит за рамки этой главы; более подробно они будут описаны в гл. 11.

1881 г. *Алан Маркуонд (Alan Marquand) изобрёл графический метод представления логических проблем.*

чиков хоть и были хорошо знакомы с языками Verilog или VHDL, но очень плохо разбирались в объектно-ориентированных особенностях языка SystemC. Кроме того, в большинстве современных средств синтеза реализованы многолетние инженерные наработки по переводу текстов Verilog или VHDL в таблицы соединений вентилей. В отличие от них средства синтеза на основе языка SystemC намного проще, чем традиционные методы.

В действительности SystemC больше подходит для проектирования на системном уровне, чем на уровне регистровых передач (RTL). Поскольку SystemC становится популярным в странах Азии и Европы, споры между SystemC, SystemVerilog и VHDL будут продолжаться еще очень долго.

Информация к размышлению

Ужасы проектирования

У большинства программистов опускаются руки при виде кода, написанного другим программистом. При этом они начинают ворчать по поводу комментариев, логики и других вещей, и нельзя сказать, что они пребывают от этого занятия в восторге.

Они просто не ведают, как им повезло, что они не видели RTL-кода, который описывает работу какого-либо устройства. Печально, но большинство схем, описанных на RTL, почти не понятны для других инженеров. В идеале RTL-описание схемы устройства можно было бы читать как книгу, начиная с «оглавления» в виде описания структуры устройства, переходя к содержанию в виде логического описания, разбитого на «главы», т. е. логические блоки устройства, и большого количества «комментариев», поясняющих структуру и принцип действия устройства.

Следует иметь в виду, что стиль кодирования может повлиять на производительность устройства. Обычно это замечание относится в большей степени к ПЛИС, чем к заказным микросхемам. Причина заключается в том, что логически эквивалентные, но разные RTL-операторы могут выдать различные результаты, разные результаты могут получиться также при использовании разных средств проектирования.

Поставщики ПЛИС и САПР по возможности снабжают своих клиентов большим количеством документации рекомендательного характера по стилям кодирования, ориентированным соответственно на собственные чипы и средства разработки. Ниже вниманию читателя предлагается ряд общих положений, которые могут оказаться полезными в большинстве ситуаций, возникающих в процессе разработки устройств.

Последовательные и параллельные мультиплексоры

При создании RTL-кода полезно иметь представление, как будут вести себя средства синтеза в определённых ситуациях. Например, оператор *if-then-else (если-то-иначе)*, как правило, реализуется с помощью мультиплексоров вида 2:1. Самое интересное начинается при использовании нескольких, вложенных друг в друга операторов if-then-else, которые средствами синтеза будут реализованы в виде структуры с приоритетом. Допустим, что ранее мы уже описали сигналы Y, A, B, C, D и SEL (сигнал выбора) и используем их в многоуровневом операторе if-then-else (**Рис. 9.7**).

1883 г. *Вильям Хаммер (William Hammer) и Томас Эдисон (Thomas Alva Edison) открыли эффект Эдисона.*

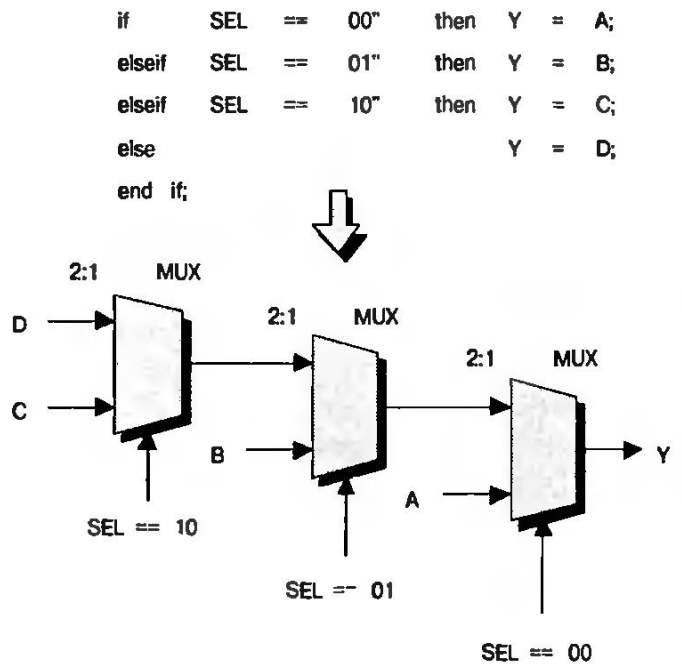

Рис. 9.7. Синтез вложенных операторов if-then-else

Как и раньше, использованный в этом случае синтаксис является общим и не принадлежит ни к одному из описываемых в этой книге языков (**Рис. 9.7**). В рассмотренном примере самый первый оператор *if-then-else* будет выполняться быстрее, чем все остальные, а самый последний может не уложиться в отведенный временной бюджет. Несмотря на это, в некоторых ПЛИС время прохождения сигналов через подобную структуру будет меньше, чем при использовании оператора *case* (оператор выбора). Реализация рассмотренного примера с помощью оператора *case* потребует использования мультиплексора вида 4:1, в котором время распространения сигналов от всех входов до выхода будет примерно одинаково (**Рис. 9.8**).

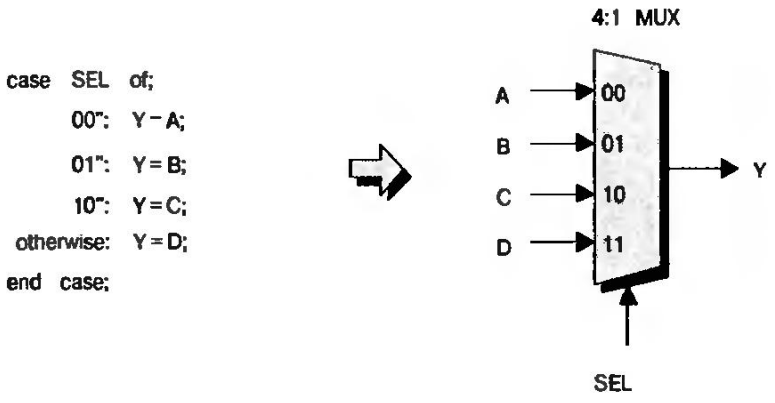

Рис. 9.8. Синтез оператора case

Остерегайтесь защёлок!

В инженерной практике настоятельно рекомендуется избегать использования защёлок в ПЛИС, кроме случаев, когда они действительно необходимы. Также следует остерегаться случаев, когда при использовании операторов *if-then-else* вы пренебрегаете конструкцией «*else*», так как большинство средств синтеза в этом месте будут вставлять защёлку.

1884 г. *Германия. Пол Нипков (Paul Gottleib Nipkow) использовал вращающийся диск для сканирования, передачи и воспроизведения изображений.*

Рационально используйте константы

В обычных схемах из всех сложных операторов наиболее часто используются сумматоры. В некоторых случаях разработчики устройств на основе заказных микросхем используют комбинации полусумматоров и полных сумматоров. Этот подход весьма эффективен в устройствах, например, представляющих собой массив вентилей, но, как правило, плохо подходит для ПЛИС.

При использовании сумматоров с константами можно провести некоторую модификацию кода, которая позволит более эффективно расходовать ресурсы устройства. Например, выражение «A + 2» может быть реализовано более эффективно в виде «A + 1 с переносом», а выражение «A – 2» как «A – 1 с переносом».

Аналогично, при использовании умножителей выражение «A × 2» более эффективно может быть реализовано в виде выражения «A SHL 1» (что значит «A сдвигается влево на один бит»), а выражение «A × 3» лучше будет представить в виде «(A SHL 1) + A».

В самом деле, небольшие преобразования могут иметь большое значение при реализации ПЛИС. Например, замена «A × 9» выражением «(A SHL 3) + A» в итоге сократит занимаемую этим выражением площадь на кристалле на 40%.

Распределение ресурсов

Распределение ресурсов представляет собой метод оптимизации, который позволяет использовать один функциональный блок, например сумматор или компаратор, для реализации нескольких операторов.

Если распределение ресурсов не используется, каждая RTL-операция реализуется с помощью собственной логики. Такой подход способствует повышению производительности устройства, но требует использования большего количества вентилей, т. е. потребляет больше места на кремниевом кристалле. Если же используется распределение ресурсов, то сокращается количество требуемых вентилей, но, как правило, это приводит к снижению производительности. Рассмотрим, например, выражение, приведенное на **Рис. 9.9**.

Для читателей нетехнического склада ума заметим, что знак «>» обозначает компаратор, т. е. устройство, которое сравнивает два числа и определяет, какое из них больше; знак «+» обозначает сумматор, а клиновидный блок представляет собой мультиплексор 2:1, который на выход пропускает одно из двух значений в зависимости от состояния на выходе компаратора.

Значения частот, указанных на **Рис. 9.9**, представляют интерес только для сравнительного анализа, так как эти значения будут зависеть от типа архитектуры ПЛИС, а также от технологии изготовления микросхем.

Перечисленные ниже операторы могут быть объединены вместе с такими же операторами или с другими однотипными, записанными с ними в одной строке:

```
*
+ -
> < >= <=
```

Например, оператор «+» может быть объединен с другими операторами «+» или с операторами «–», а оператор «×» может быть объединен только с такими же операторами «×».

1886 г. *Льюис Кэррол (Lewis Carrol) опубликовал работу, в которой изложил диаграммные методы логических представлений в Теории Игр.*

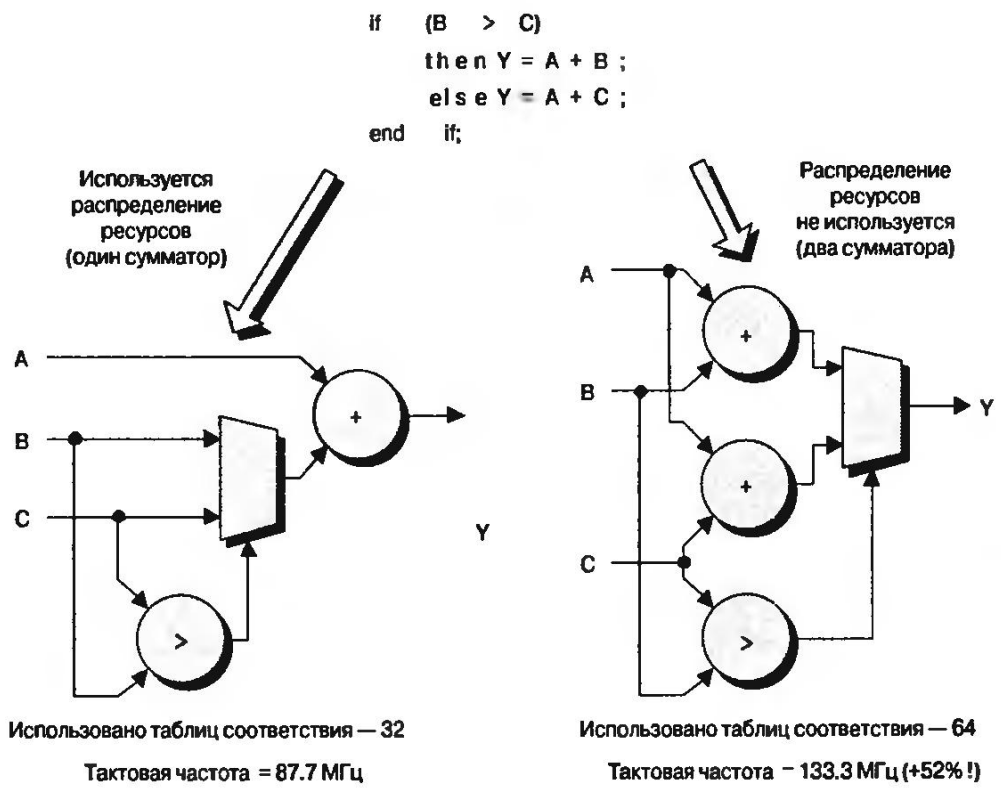

Рис. 9.9. Распределение ресурсов

Также необходимо проверить, разрешено или запрещено по-умолчанию использование распределения ресурсов в вашем приложении синтеза. И в завершение: распределение ресурсов может существенно упростить разводку элементов в заказных микросхемах, а в случае ПЛИС, напротив, может привести к возникновению ряда проблем при разводке.

Последнее, но не менее важное

Внутренние шины с тремя состояниями обычно работают довольно медленно, и в большинстве ПЛИС следует избегать их использования, если только нет полной уверенности в их необходимости. По возможности используйте буферы с тремя состояниями исключительно на верхнем уровне устройства. Если все-таки они будут использоваться, а применяемая ПЛИС не поддерживает вентили с тремя состояниями, большинство современных средств синтеза обеспечивают автоматическую реализацию третьего состояния с помощью мультиплексоров. В основном производятся преобразования буферов с тремя состояниями, описанными с помощью RTL-кода, в соответствующую логику на основе таблиц соответствия.

Причиной проблем синхронизации также могут быть двунаправленные буферы, поэтому при их использовании следует быть предельно осторожными.