ГЛАВА 19

СРЕДСТВА МОДЕЛИРОВАНИЯ, СИНТЕЗА, ВЕРИФИКАЦИИ И РЕАЛИЗАЦИИ

Введение

Для ввода, верификации, синтеза и реализации своих проектов инженерам-разработчикам обычно приходится использовать огромное количество разнообразных средств. Даже краткое описание всех этих средств вылилось бы не в одну книгу[1]. Поэтому мы ограничимся рассмотрением лишь наиболее важных из них в контексте разработки устройств на основе ПЛИС, включая:

- средства моделирования на основе циклов, событий и другие;
- средства синтеза логического/HDL и физического;
- средства общей верификации;
- средства формальной верификации;
- другие средства.

Типы моделирования

Событийное моделирование

Сегодня логическое моделирование является одним из главных инструментов проверки, или верификации, в арсенале инженера. Наиболее распространенной формой логического моделирования является *событийное* (event-driven) *моделирование*. Средства событийного моделирования воспринимают окружающий мир как последовательность дискретных событий. В качестве примера рассмотрим очень простую схему, показанную на **Рис. 19.1.** Как следует из рисунка, схема включает вентиль ИЛИ, буферный вентиль BUF пару инверторов НЕ; сигнал с вентиля ИЛИ поступает на буферный вентиль BUF. Давайте посмотрим, что произойдет в схеме при изменении сигнала на одном из ее входов.

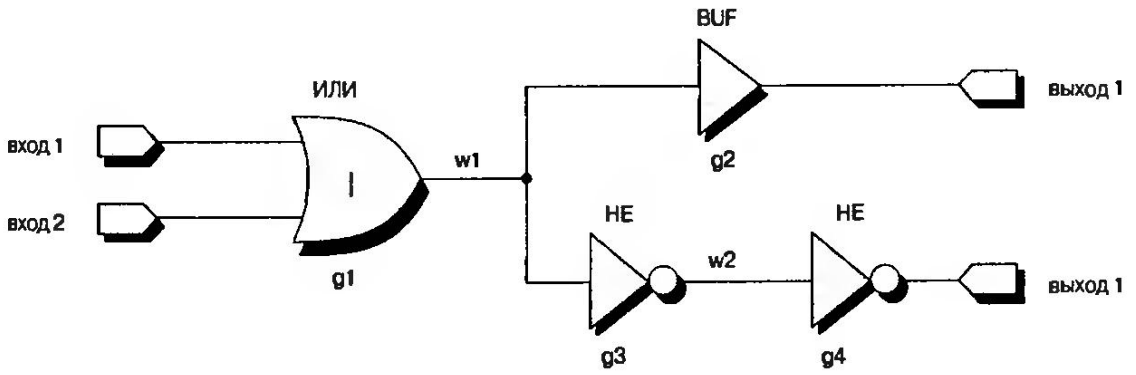


Рис. 19.1. Пример простой схемы

[1] Я был бы безмерно счастлив написать такую книгу, если бы нашел спонсора.

Для упрощения примера допустим, что вентили НЕ имеют задержку 5 пикосекунд, буферный логический элемент 10 пикосекунд, а логический элемент ИЛИ 15 пикосекунд (**Рис. 19.2**).

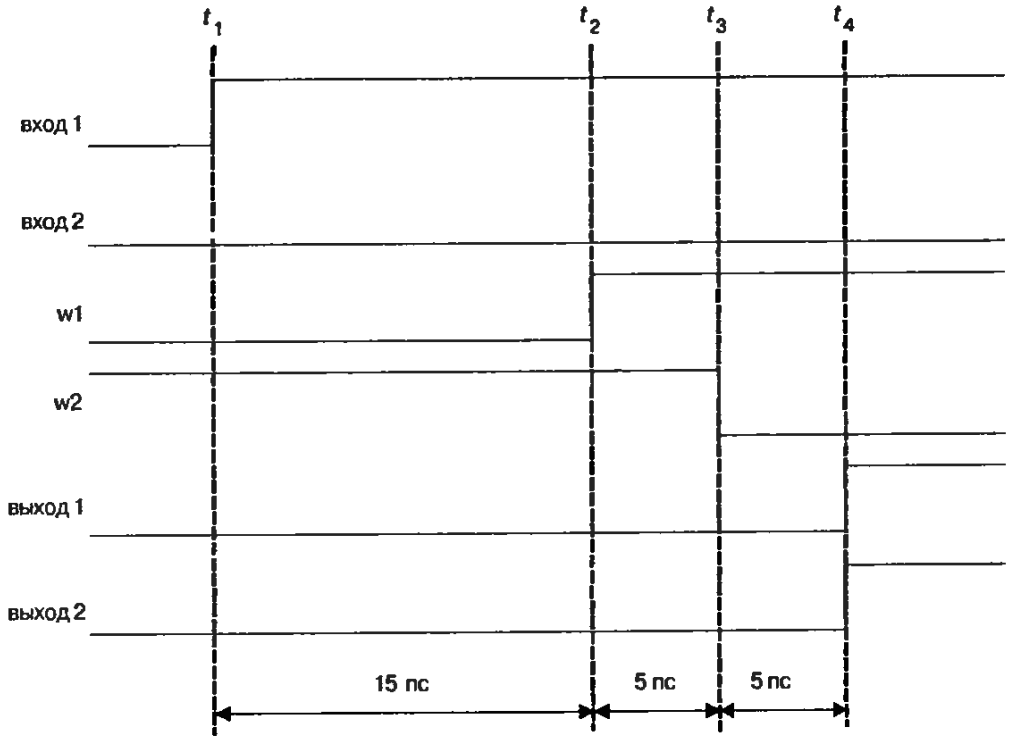


Рис. 19.2. Результаты событийного моделирования

Внутри системы моделирования находится так называемое *событийное колесо, или колесо событий*, на которое помещаются события, которые будут «происходить» в будущем. Когда на входе *вход 1* имеет место первое событие в период времени t_1, система моделирования отслеживает, с чем соединен этот вход: в данном случае он соединён с логическим элементом ИЛИ. Затем система моделирования с учётом информации о задержке этого элемента планирует соответствующее событие на его выходе, а именно: через 15 пикосекунд в момент времени t_2 на шине *w1* состоится переход с уровня логического 0 на уровень 1.

После этого система моделирования проверяет, нужно ли выполнять ещё какие-либо действия в текущий момент времени t_1, и обращается к колесу событий, чтобы выяснить, что должно произойти дальше. В нашем примере следующим событием оказалось как раз то, что мы только что запланировали на время t_2, т. е. фронт импульса, или переход с уровня 0 на уровень 1, на шине *w1*. Во время выполнения этого действия система моделирования также отслеживает, с чем соединена шина *w1*: в нашем случае к этой шине подсоединены буферный элемент BUF, обозначенный как *g2*, и инвертирующий логический элемент НЕ, обозначенный как *g3*.

Так как логический элемент НЕ (*g3*) имеет задержку 5 пикосекунд, система моделирования планирует на выходе этого элемента, т. е. на шине *w2*, на момент времени t_3 (через 5 пикосекунд) спад импульса (переход с уровня логической 1 на уровень 0). Аналогично, поскольку буферный элемент BUF (*g2*) имеет задержку 10 пикосекунд, система моделирования планирует фронт импульса (переход с уровня 0 на уровень 1) на выходе *выход 1* на время t_4 (через 10 пикосекунд). И так далее, пока не будут выполнены все события, запущенные фронтом импульса на *входе 1*.

Преимущество такого событийного подхода состоит в том, что системы моделирования на его основе могут использоваться для пред-

ставления почти всех видов устройств, включая синхронные и асинхронные схемы, комбинаторные цепи обратной связи и другие. Эти системы моделирования обеспечивают хорошую визуализацию, упрощая тем самым процесс отладки, и могут оценивать влияние запаздывающих коротких импульсов и выбросов, которые трудно выявить с помощью других методов. Существенным недостатком этих систем моделирования является то, что они требуют значительных вычислительных ресурсов и, соответственно, отличаются медлительностью.

1928 г. *Джон Байрд (John Logie Baird) продемонстрировал работу электронной системы цветного телевидения.*

1928 г. *Джон Байрд разработал видеодиск для записи телевизионных программ.*

Краткий обзор систем событийного моделирования

Как уже обсуждалось в гл. 8, первые цифровые системы событийного моделирования (конец 60-х — начало 70-х) основывались на концепции *базовых элементов моделирования*. Как минимум, эти базовые элементы включали логические вентили, такие как буферы, НЕ, И, И-НЕ, ИЛИ, ИСКЛЮЧАЮЩЕЕ НЕ-ИЛИ, а также некоторые типы буферных элементов с тремя состояниями. Одни системы моделирования в качестве базовых элементов содержали набор регистров и защёлок, другие для реализации этих же функций предусматривали создание подсхем из набора более примитивных логических вентилей.

В то время функциональность устройств обычно описывалась с помощью стандартных текстовых редакторов в виде таблицы соединений вентилей. Аналогично сами тесты описывались в виде текстовых (табличных) файлов задающих воздействий. Система моделирования воспринимала для обработки таблицу соединений и таблицу задающих воздействий, а также управляющие файлы и команды из командной строки. По таблице соединений вентилей в памяти компьютера создавалась модель устройства, затем к этой моделям прикладывались задающие воздействия из соответствующего файла, и в конце формировались результаты в виде текстового (табличного) файла (**Рис. 19.3**).

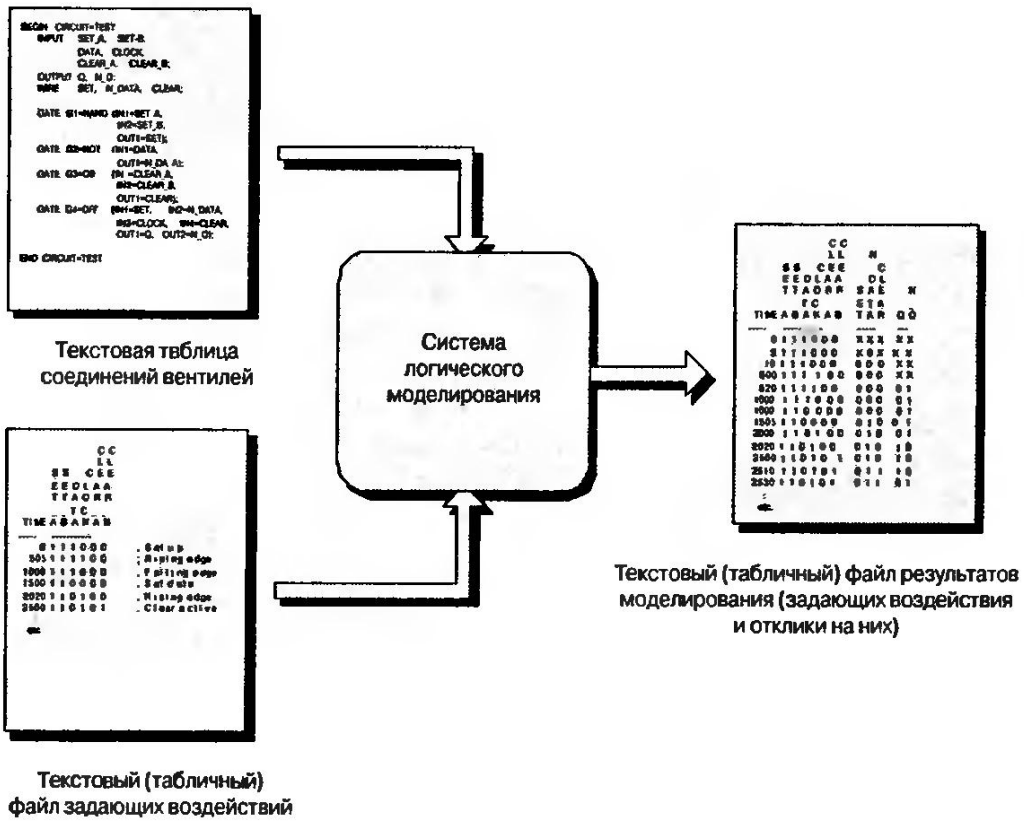


Рис. 19.3. Работа системы логического моделирования

1929 г. *Джозеф Шик (Joseph Schick) разработал электрическую бритву.*

Со временем всё стало несколько сложнее. Сначала для создания таблицы соединений вентилей стали использоваться средства ввода принципиальных схем. Затем использовались специальные средства, которые считывали готовые текстовые файлы и представляли полученную информацию в графическом виде. В некоторых случаях эти средства использовались также для описания в графическом виде задающих воздействий и последующей генерации соответствующего табличного файла.

После этого разработчики цифровых систем моделирования взялись за более сложные языки, которые могли бы описывать логические функции на более высоких уровнях абстракции, таких как уровень регистровых передач (*RTL — register transfer level*). Хорошим примером такого языка был *GHDL (GenRad Hardware Description Language — язык описания аппаратных средств GenRad)*, используемый системой моделирования System HILO.

Получили также развитие и более сложные языки описания испытательных стендов, как, например *GWDL (GenRad WaveformDescription Language — язык описания формы сигнала GenRad)*. Языки этого типа могли поддерживать сложные структуры, такие как циклы, а также имели доступ к текущему состоянию схемы и, соответственно, могли изменять наборы тестов по принципу: «Если на этом выходе находится уровень логического нуля, приступайте к тесту Б, в противном случае начинайте тест В».

Эти языки в какой-то степени опережали свое время. Например, полезное свойство языка GWDL заключалось в том, что кроме определения входного сигнала, например «вход A = 0», можно было определять ожидаемый выходной отклик, например «выход Y = = 1». При этом наличие одного знака равенства означало подачу воздействия на вход системы, а пара знаков равенства соответствовала ожидаемому отклику. Если после этого использовался специальный оператор STROBE, имитатор проверял, совпадает или нет реальный отклик (со схемы) с ожидаемым откликом (заданным формой сигнала) и выдавал предупреждение, если эти сигналы отличались.

Шло время, и начали появляться промышленные стандарты языков HDL, такие как Verilog и VHDL. Их преимущество заключалось в том, что один и тот же язык мог использоваться для описания и функциональности схемы, и испытательных стендов[1)].

Стали появляться и форматы стандартных файлов для вывода результатов моделирования, например *VCD-формат* (Value Change Dump — дамп изменения значений). Эти форматы помогали сторонним компаниям САПР электронных систем создавать сложные графические средства наблюдения за формой сигнала, а также средства анализа, которые могли работать с выходными данными многочисленных систем моделирования. Следует упомянуть и новый *FSDB-формат* (Fast Signal Database™ — база данных быстрых сигналов) компании Novas Software (www.novas.com), который создаёт файлы гораздо меньших размеров, чем VCD, и в то же время позволяет осуществлять быстрый поиск информации.

Были и другие инновации, например *SDF-формат* (standard delay format — формат стандартных задержек), который помогал сторонним компаниям САПР разрабатывать сложные средства временного анализа. Эти средства были способны производить оценку схемы, созда-

---

[1)] Главный разработчик языка Verilog Фил Морби (Phil Moorby) был также одним из разработчиков первой версии языка и среды моделирования HILO.

вать временные отчеты, выделять потенциальные проблемы и выводить SDF-файлы, которые могли быть использованы для проведения более точного временного моделирования

Логические величины и системы логических величин

Подавляющее большинство современных цифровых электронных систем основываются на *двоичной логике*, которая использует цифры, называемыми *битами*. Другими словами, логические элементы используют два разных напряжения для представления двоичных значений 0 и 1 или логических (булевых) значений вида *true* и *false* (соответственно *истина* и *ложь*). Были проведены некоторые эксперименты по использованию *троичной логики*, основанной на трех различных логических уровнях, значения которых называются *тритами*. Однако до сих пор эта технология не нашла коммерческого и промышленного применения. Чему я несказанно рад.

Опять я отвлекся. Минимальный набор логических значений, необходимых для описания работы двоичного логического элемента, состоит из чисел 0 и 1. Затем, чтобы как-то представить неизвестные величины, для их обозначения обычно используется символ «*X*». Эти неизвестные величины могут использоваться для представления разных состояний, например для обозначения содержания неинициализированного регистра или для выделения конфликта, возникающего, когда два логических элемента управляют одной и той же шиной с противоположными логическими значениями. Для обозначения *высокоимпедансного состояния*, которое может появиться на выходах логических элементов с тремя состояниями, обычно используется символ «*Z*».

В языках описания аппаратных средств вместо символа «*X*» обычно используются обозначения «?» или «–». Неопределённое значение не может быть установлено на выходе в качестве входных воздействий для других входов. Вместо этого используется форма записи, в которой описывается, как входы реагируют на различные комбинации сигналов.

Но состояния 0, 1, *X* и *Z* — только верхушка айсберга. Более продвинутые логические системы моделирования могут связывать различные воздействия с выходами разных логических элементов. Они могут также содержать средства разрешения и описания ситуаций, в которых несколько логических элементов управляются одним и тем же проводником с различными логическими значениями разных воздействий. Ну, разумеется, исключительно ради разнообразия, языки VHDL и Verilog работают с такими ситуациями по-разному.

Цифровые системы логического моделирования, такие как *векторное произведение* и *метод интервальных значений*, и различные аспекты неизвестной переменной *X* более детально рассмотрены в моей книге *Designus Maximus Unleashed (Banned in Alabama)*, ISBN 0-7506-9089-5.

Моделирование с использованием разных языков

При существовании двух стандартных промышленных языков, например, VHDL и Verilog, рано или поздно возникнет проблема, смысл которой в том, что через некоторое время у пользователя окажутся две части устройства, описанные на разных языках. Вероятно, все собственные разработки компании будут написаны на том языке, которому она отдает предпочтение. Тем не менее, проблема может возникнуть, если компания захочет использовать рабочий код, написанный на другом языке. Та же ситуация может повториться, если компания примет решение о приобретении блоков интеллектуальной собственности у стороннего разработчика, который может работать только с

тем языком, который компания в текущий момент не использует. Возможна также ситуация, когда одна компания объединится с другой для совместного проекта, причем обе компании давно занимаются проектными разработками и используют несопоставимые языки.

Подобная ситуация приводит к необходимости прибегнуть к помощи концепции *моделирования на разных языках*, которая исторически имеет несколько разновидностей. Одна из первых методик заключалась в переводе исходного кода с «чужого» языка на «свой». Но этот подход отличался низкой эффективностью, так как разные языки поддерживали разные логические состояния и разные языковые конструкции, и даже похожие операторы языка имели разную семантику. В результате при моделировании переведенный с другого языка проект редко вел себя так, как ожидалось, поэтому этот метод нечасто используется в наши дни.

Другая методика подразумевала наличие двух систем моделирования — для языка VHDL и для Verilog и предполагала совместное моделирование двух подсистем с помощью двуядерной системы. В этом случае производительность моделирования, к сожалению, находилась на низком уровне, так как одна из подсистем останавливалась, дожидаясь, пока другая закончит свою работу. Поэтому и этот метод редко используется в наши дни.

Оптимальное решение описанной проблемы предполагает одноядерную систему моделирования, которая поддерживает описания устройств, представленных в виде комбинации (или смеси) блоков на языках VHDL и Verilog. Все компании САПР электронных систем располагают собственными версиями таких средств, а некоторые из них переходят границы всех смелых предположений, сделанных в прошлом, поддерживая множество языков, например: VHDL, Verilog, SystemVerilog, SystemC и PSL[1)].

Альтернативные форматы описания задержек

В моделях, которые разрабатываются для использования в системе событийного моделирования, выбор формата представления задержек зависит от возможностей моделирования задержек самой системой моделирования и от выбора места (и средств) в процессе разработки, в котором будет выполняться временной анализ.

Очень часто *статический временной анализ* (или *STA — static timing analysis*) выполняется отдельно от моделирования (подробно обсуждается позже в этой главе). В этом случае логические вентили и более сложные операторы могут быть промоделированы с нулевой задержкой, т. е. масштаб времени — 0 или с единичной задержкой, т. е. масштаб времени — 1. Здесь термин *масштаб времени* относится к наименьшему сегменту времени, распознаваемому системой моделирования.

В другом случае можно связать более сложные типы задержек с логическими вентилями и с другими более сложными операторами для использования в системе моделирования. При этом сначала необходимо отделить задержку фронта импульса от задержки спада на выходе логического вентиля или другого оператора. Исторически сложилось так, что задержка фронта импульса (при переходе с 0 в 1) часто обозначается как LH (сокращенно от low-to-high — от низкого к высокому

[1)] Хорошим примером одноядерного решения такого типа может служить программный продукт ModelSim® от компании Mentor Graphics (www.mentor.com).

уровню). Соответственно задержка спада (при переходе с 1 в 0) обозначается как HL (high-to-low — от высокого к низкому уровню).

Рассмотрим, например, что произойдет, если подать 12-пикосекундный (12 пс) положительный (0-1-0) импульс на вход простого буферного логического элемента с задержкой LH = 5 пс и HL = 8 пс (**Рис. 19.4**).

1929 г. *В Британии заработала технологическая линия по производству механических телевизионных систем.*

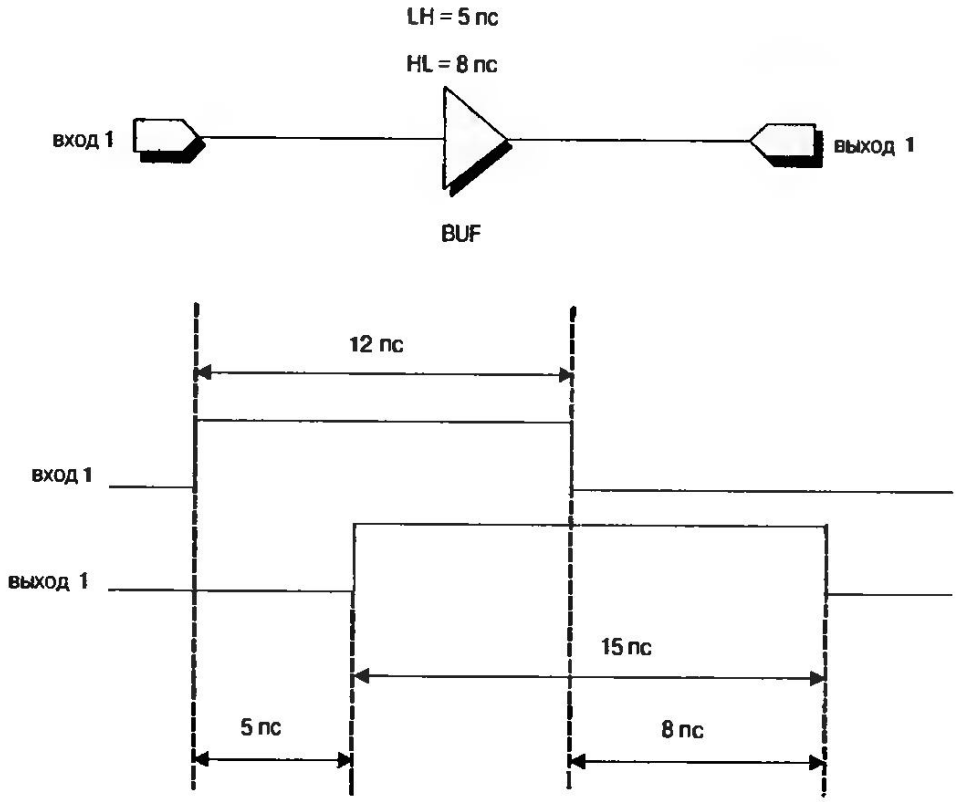


Рис. 19.4. Различные LH- и HL-задержки

Не удивительно, что напряжение на выходе логического элемента возрастает через 5 пс после приложения к входу фронта импульса, и падает через 8 пс после снятия импульса на входе. Интересно, что из-за несбалансированных задержек 12-пикосекундный импульс увеличился до 15 пс на выходе логического элемента, причем дополнительные 3 пс отражают разницу между значениями LH и HL. Аналогично, если отрицательный импульс длительностью 12 пс (1-0-1) будет подан на вход этого логического элемента, соответствующий импульс на выходе уменьшится до 9 пс. Читатель, попробуйте самостоятельно изобразить этот процесс на листе бумаге.

Кроме задержек LH и HL, системы моделирования поддерживают минимальные, номинальные и максимальные (мин:ном:макс) значения для каждой задержки. Рассмотрим, например, положительный импульс длительностью 16 пс, который подается на вход буферного логического элемента с задержками фронта и спада соответственно, 6:8:10 пс и 7:9:11 пс (**Рис. 19.5**).

Диапазон величин задаётся для работы в разных рабочих условиях, например при перепадах окружающей температуры или напряжения питания. Таким же образом охватывают изменения в производственном процессе, так как некоторые микросхемы могут работать немного быстрее или медленнее, чем другие того же класса. Аналогично логические вентили микросхем из одной области (например, в заказных микросхемах или в ПЛИС) могут включаться быстрее или медленнее, чем идентичные логические вентили из другой области.

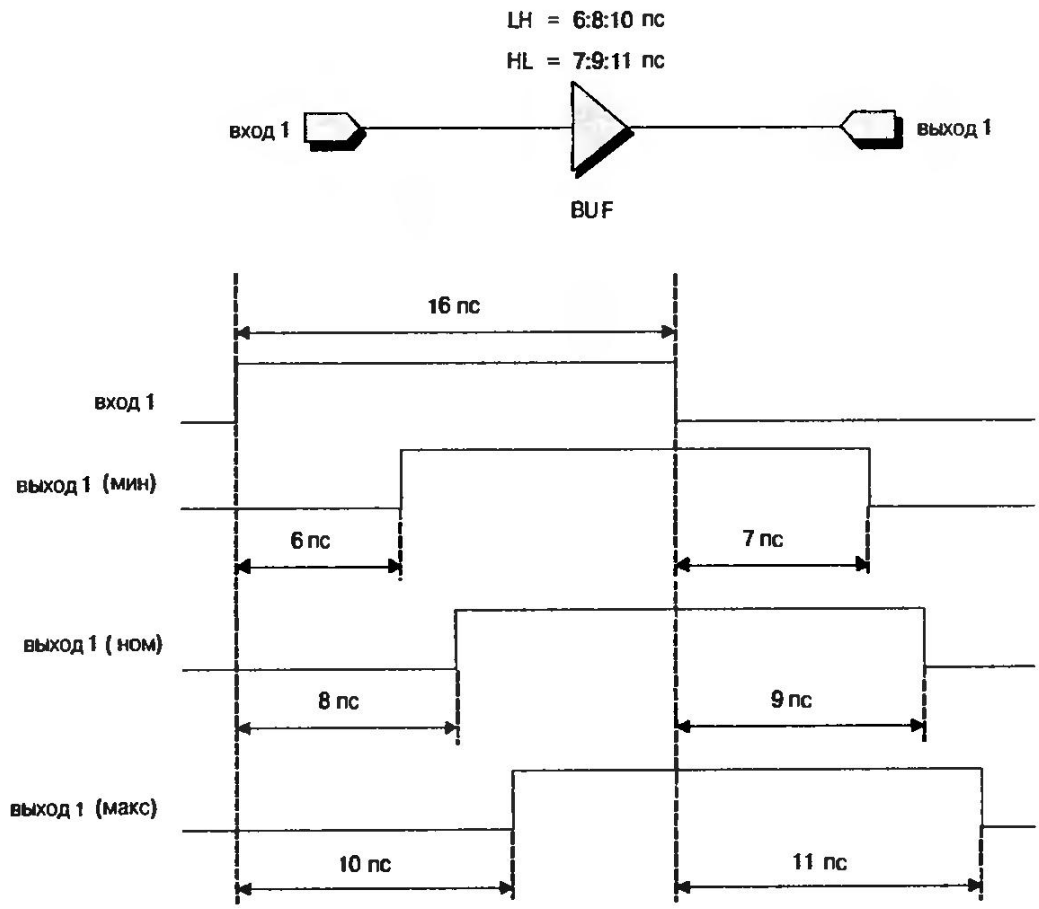


Рис. 19.5. Поддержка формата описания задержек вида мин:ном:макс

Раньше все задержки распространения сигнала от входа до выхода микросхемы у многовходовых вентилей были одинаковыми. Например, в случае 3-входового вентиля И с входами *a*, *b* и *c*, и выходом *y* любые задержки вида LH и HL при прохождении по пути *a*-*y*, *b*-*y* и *c*-*y* были одинаковыми. Изначально это обстоятельство не вызывало никаких проблем, поскольку таким же способом задержки описывались и в соответствующих справочниках. Однако со временем в справочниках стали определять задержки индивидуально для каждого пути прохождения сигнала, и в результате возникла необходимость модернизировать системы моделирования для поддержки этих возможностей.

ТТЛ-логика реализуется с помощью соединённых определённым образом биполярных транзисторов.

Давайте рассмотрим, что произойдет, если к входу логического вентиля (или более сложной функции) приложить короткий импульс. Под «коротким» мы будем подразумевать импульс, длительность которого меньше времени задержки прохождения сигнала через вентиль. Следует заметить, что первые логические элементы в основном использовались в простых микросхемах *транзисторно-транзисторной логики (TTL)*, которые располагались на печатной плате. Эти микросхемы имели свойство поглощать (или отбрасывать) поступающие на них короткие импульсы, что и имитировали системы моделирования. Описания этих процессов стали называть *моделью инерционной задержки*. В качестве простого примера давайте рассмотрим два положительных импульса длительностью 8 пс и 4 пс, приложенных к буферному логическому элементу, у которого время задержки по фронту и по спаду в форме мин:ном:макс равно 6 пс **(Рис. 19.6)**.

1929 г. *Начало экспериментов по изучению цветного электронного телевидения.*

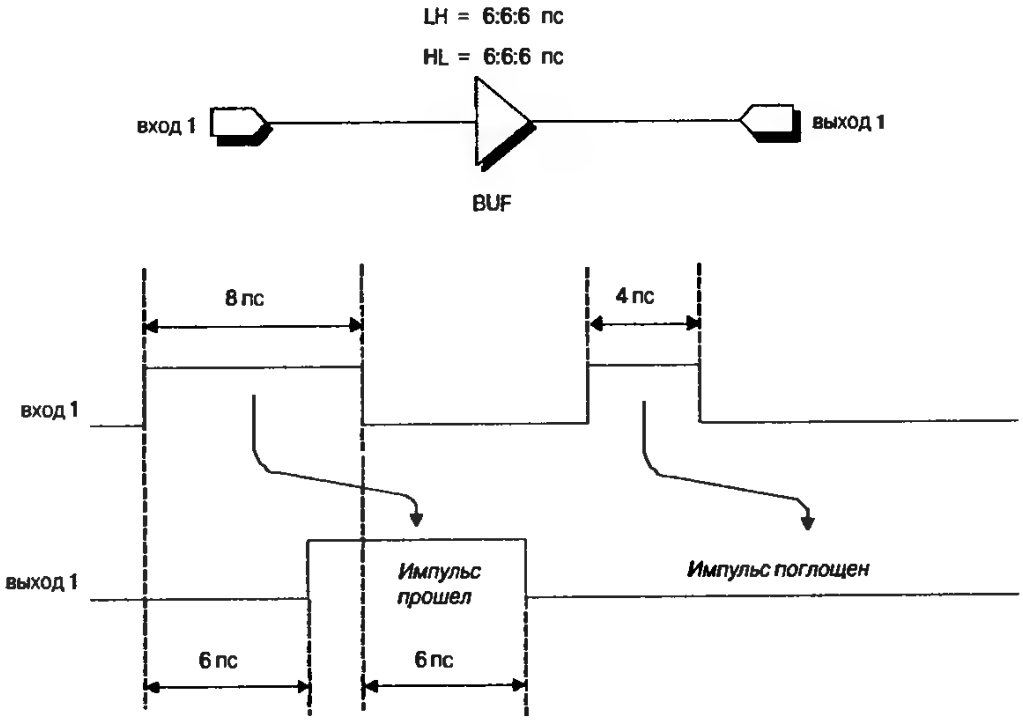


Рис. 19.6. Модель инерционной задержки отбрасывает импульс, длительность которого меньше, чем время задержки вентиля

В отличие от рассмотренного примера вентили, реализованные с помощью более поздних технологий, таких как *эмиттерно-связанная логика (или ECL — emitter-coupled logic)*[1], пропускали импульсы, длительность которых была меньше задержки прохождения через них сигнала. Для работы с такими элементами некоторые системы моделирования могли функционировать в режиме, названном *моделью транспортной задержки*. Давайте ещё раз рассмотрим два положительных импульса длительностью 8 пс и 4 пс, приложенных к буферному логическому элементу, у которого время задержки по фронту и по спаду в форме мин:ном:макс равно 6 пс (**Рис. 19.7**).

В случае моделей инерционной и транспортной задержки возникает проблема, связанная с тем, что они могут применяться лишь в экстремальных ситуациях. В связи с этим разработчики некоторых систем моделирования начали проводить эксперименты с более сложными короткоимпульсными методиками, такими как *модель трёхдиапазонной задержки*[2]. В этом случае каждая задержка может определяться двумя значениями, скажем *r* для описания отбрасывания сигнала и *p* для описания пропускания сигнала, которые задаются в процентах от общей задержки. Например, рассмотрим буферный вентиль, у которого все задержки в форме мин:ном:макс равны 6 пс, а значения *r* и *p* соответственно 33% и 66% (**Рис. 19.8**).

Любые, поданные на вход вентиля импульсы, длительность которых больше или равна значению *p*, будут пропускаться этим вентилем; все импульсы, длительность которых меньше, чем значение *r*, будут

---

[1] ЭСЛ-логика также реализуется с помощью биполярных транзисторов, но соединённых по другой схеме, чем ТТЛ. Логические элементы ЭСЛ работают быстрее своих ТТЛ-аналогов, но и потребляют больше энергии (следовательно, выделяют больше тепла).

[2] Система моделирования *System HILO* компании GenRad начала использовать модель 3-диапазонной задержки незадолго до ее исчезновения с лица Земли.

1929 г. *Начали работу первые коммерческие службы связи (для пассажиров) между кораблями и берегом.*

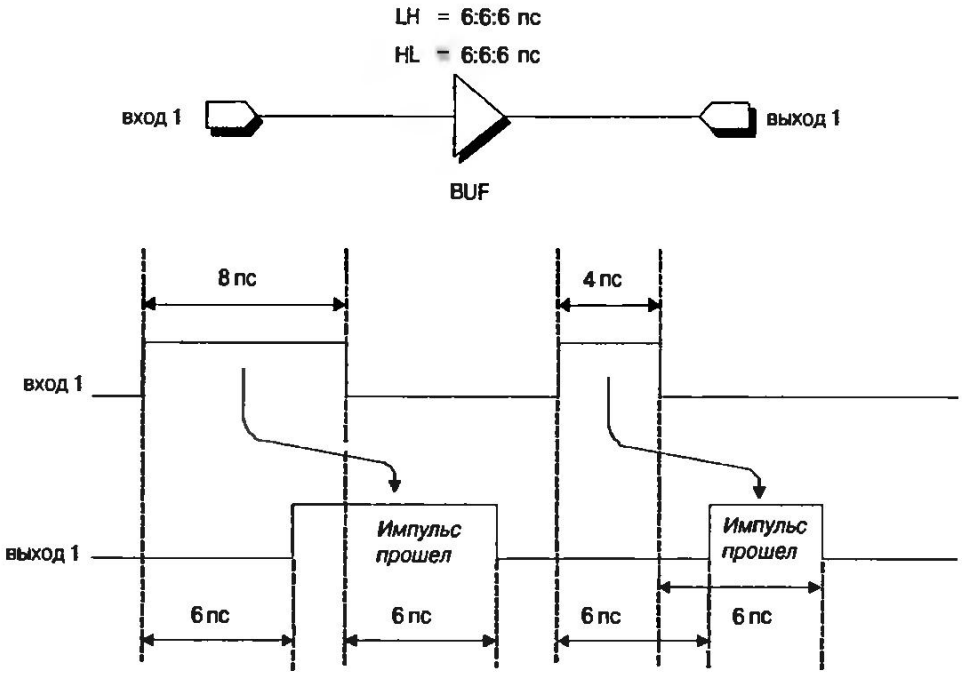


Рис. 19.7. Модель транспортной задержки пропускает импульсы любой длительности

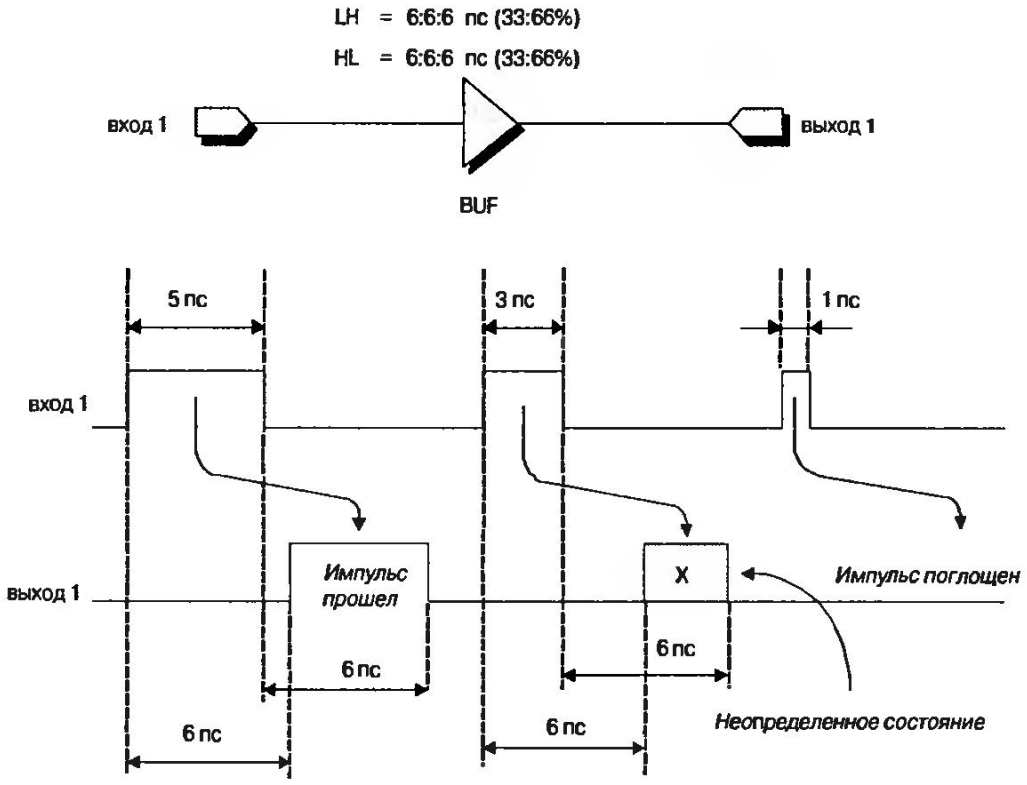


Рис. 19.8. Модель инерционной задержки отбрасывает импульс, длительность которого меньше, чем время задержки вентиля

полностью отбрасываться (поглощаться); импульсы, длительность которых находится между этими двумя экстремумами, будут пропускаться, как импульсы с неизвестными величинами *X* с тем, чтобы показать, что их состояние неопределенно, так как мы не знаем, будут ли они пропущены через логический элемент при работе в реальном устройстве. (Установка значений *r* и *p* на уровень 100% эквивалентна использованию модели инерционной задержки, а их установка на уровень 0% приводит к модели транспортной задержки.)

Системы циклового моделирования

Кроме событийного моделирования, возможно также цикловое (cycle-based) моделирование. Оно очень хорошо подходит для моделирования конвейерных устройств, в которых «островки» комбинационной логики расположены между блоками регистров (**Рис. 19.9**).

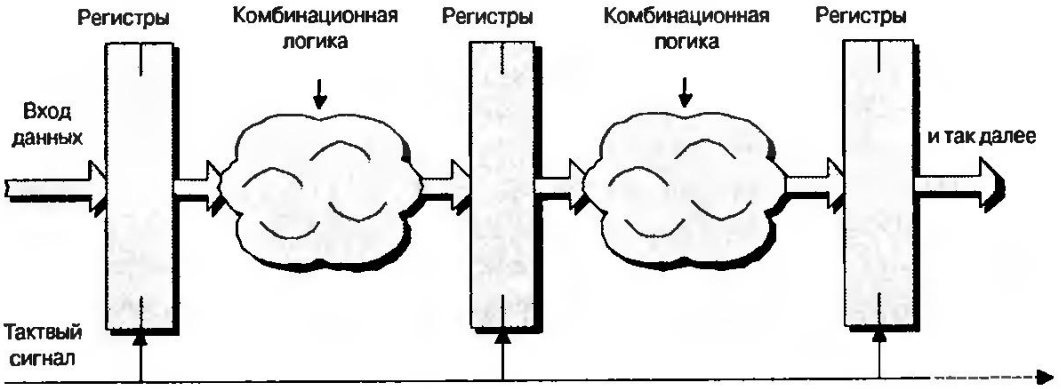


Рис. 19.9. Простое конвейерное устройство

В этом случае система циклового моделирования будет выдавать любую временную информацию, связанную с вентилями комбинационной логики, а также будет преобразовывать эту логику в последовательность логических (булевых) операций, которые могут быть реализованы с помощью микропроцессорных команд.

При наличии схемы, которая работает соответствующим образом, системы циклового моделирования демонстрируют лучшее быстродействие по сравнению со своими событийными аналогами. Недостаток таких систем заключается в том, что они обычно работают лишь с логическими величинами 0 и 1 (значения *X*, *Z* и другие выражения не поддерживаются). Кроме того, эти системы не могут моделировать работу асинхронных логических устройств или комбинационных цепей обратной связи.

В наше время цикловые системы моделирования в чистом виде почти не используются. Однако путем определённых расширений некоторые системы событийного моделирования приобрели смешанные свойства. В этом случае, если дать указание такой системе работать в режиме максимальной производительности, а не в режиме максимальной временной точности, она автоматически обработает некоторые части схемы с помощью методов событийного моделирования, а оставшуюся часть с помощью методов циклового моделирования.

Выбор оптимальной системы логического моделирования!

Выбор системы моделирования, как и всего остального при разработке чего-то нового, это выбор сбалансированного решения. Если вы работаете в небольшой начинающей компании, для которой при выборе средств разработки решающим является их стоимость, переходите сразу к гл. 25, в которой описаны методы разработки с использованием средств с открытым исходным кодом.

Первым делом необходимо определиться, нужно ли вам использовать нескольких языков в пределах одного устройства. Если у вас небольшая начинающая фирма, можно ограничиться только одним языком, но помните, что какие-нибудь блоки интеллектуальной

1929 г. *Германия. Осуществлена запись звука на пластиковую ленту с магнитным покрытием.*

собственности, которые вы решите приобрести в будущем, возможно, не смогут работать с этим языком. Неплохо было бы начать работать с языками VHDL, Verilog или SystemVerilog, а, если возможно, еще и оперировать с SystemC вместе с одним или несколькими языками формальной верификации. Тогда, по всей видимости, ваша позиция будет достаточно сильной в течение определённого времени.

В общем случае, наиболее важным критерием для большинства является производительность. Возникает вопрос: как наиболее точно определить производительность? Пожалуй, единственный способ заключается в разработке собственного тестового устройства и его моделировании на нескольких системах. Создание хорошего теста — нетривиальная задача, но этот способ лучше, чем использовать разработку, предлагаемую производителем САПР. Подобный проект будет реализован так, чтобы в наилучшем свете продемонстрировать решение своего разработчика и поставить подножку конкурентам.

Однако, кроме производительности, есть еще не менее важные показатели. Вам также понадобится хорошее интерактивное средство отладки, чтобы после обнаружения проблемы можно было бы остановить моделирование и проверить свое устройство. В этом отношении не все системы моделирования одинаковы и разные средства имеют разные уровни возможностей. В некоторых случаях, даже если имитатор позволяет выполнить задуманное, для этого иногда приходится встать на уши. Поэтому целесообразно воспользоваться некоторой уловкой, которая заключается в следующем: после прохождения собственного образцового теста создать такую же схему, но с заложенным в неё заранее известным дефектом и посмотреть, насколько легко и быстро будет обнаружен и локализован заложенный дефект. На самом деле, некоторые системы моделирования, которые обладают необходимой производительностью, настолько плохо справляются с подобным заданием, что придётся воспользоваться средствами сторонних разработчиков для проведения дополнительного анализа после завершения моделирования[1)].

Внимания заслуживает и разрядность системы моделирования. Средства, поставляемые большими компаниями САПР, как правило, не имеют ограничений по разрядности, а вот системы моделирования небольших разработчиков могут основываться на 32-битном коде. Естественно, если вы собираетесь работать с небольшими проектами, скажем не более 500000 вентилей, то, вероятно, подойдут системы моделирования, поставляемые производителями ПЛИС. Обычно это упрощенные версии средств от крупных разработчиков САПР.

Разумеется, кроме вышеперечисленных, у вас будут собственные критерии оценки средств моделирования, например по качеству кодового покрытия или анализу производительности. Когда-то эти возможности были в компетенции специализированных средств сторонних разработчиков, но теперь большинство мощных систем моделирования позволяют оценить на некотором уровне кодовое покрытие и провести анализ производительности непосредственно в среде моделирования. При этом разные системы моделирования предлагают различные наборы дополнительных функциональных возможностей.

[1)] Лидирующее положение в этой области занимает компания Novas Software Inc. (www.novas.com) со своими средствам анализа Debussy® и Verdi™.

Средства синтеза. Логический/HDL и физический синтез

1929 г. *На автомобиль впервые установлен радиоприёмник.*

Технология логического/HDL-синтеза

Традиционные средства *логического синтеза* появились примерно в первой половине 80-х. В наши дни эти средства иногда называют *технологией HDL-синтеза*.

Роль первых средств логического/HDL-синтеза заключалась в формировании таблицы соединений вентилей на основе заранее составленного RTL описания заказной микросхемы и соответствующих временных ограничений. Во время этого процесса приложение синтеза выполняло различные процедуры минимизации и оптимизации, включая оптимизацию по быстродействию и по площади, занимаемой на поверхности кристалла.

В середине 90-х средства синтеза были расширены для поддержки архитектуры устройств ПЛИС. Такие приложения могли формировать таблицу соединений КЛБ/таблиц соответствия, которая затем передавалась программному обеспечению размещения и разводки, предоставляемому поставщиком ПЛИС (**Рис. 19.10**).

Рис. 19.10. Традиционный логический/HDL-синтез

На практике, устройства, на основе ПЛИС, созданные средствами архитектурного синтеза, были на 15...20% быстрее, чем их аналоги, созданные с использованием средств традиционного синтеза на уровне вентилей.

Технология физического синтеза

У традиционного логического/HDL-синтеза была одна проблема, которая заключалась в том, что он был разработан в то время, когда задержка распространения сигнала в основном определялась задержкой вентилей, а задержки на проводниках были сравнительно малыми. Это значит, что средства синтеза могли использовать простые модели нагрузки на шину для оценки влияния задержек на проводниках. Такие модели включали параметры ненагруженной шины, ёмкость шины при нагрузке в виде одного логического элемента — *x* пикофарад (пФ), ёмкость шины при нагрузке двумя логическими элементами — *y* пФ и т. д. По этим данным средства синтеза могли оценивать задержку, связанную с каждым проводником, как функцию её загрузки и сопротивления вентиля, подключенного к этому проводнику.

1930 г. *Ваневар Буш (Vannevar Bush) сконструировал аналоговый компьютер, названный Differential Analizer (дифференциальный анализатор).*

1933 г. *Эдвин Армстронг (Edwin Howard Armstrong) разработал новую систему радиосвязи: широкополосную частотную модуляцию.*

Эта методика соответствовала уровню изготовления устройств того времени, которые реализовывались по мультимикронным технологиям и содержали, по сегодняшним меркам, относительно небольшое количество вентилей. В сравнении с ними, современные устройства могут содержать десятки миллионов вентилей, и размеры этих элементов уходят в область глубокого субмикронна. Другими словами, задержки сигнала на проводниках теперь могут составлять до 80% полной задержки. Следовательно, использование традиционных средств логического/HDL-синтеза для оценки временных параметров современных устройств может привести к таким данным, которые будут иметь мало общего с действительностью. При этом достичь состояния временного соответствия будет почти невозможно.

По этой причине при разработке заказных интегральных микросхем (ASIC), примерно с 1996 года стали использовать физический синтез, который в 2000 или 2001 году стал применяться и для ПЛИС. Конечно, существует множество различных определений в отношении того, что именно обозначает термин *физический синтез*. Основная идея состоит в том, чтобы до начала синтеза использовать физическую информацию об устройстве, но что это означает на самом деле? Например, некоторые компании добавили возможности интерактивного планирования компоновки в качестве первого этапа алгоритма синтеза, и классифицировали это как *физический синтез*. Однако для большинства инженеров физический синтез означает получение информации связанной с физическим расположением различных логических элементов в схеме, использование этой информации для точной оценки задержек на проводниках, использование этих задержек для уточненного размещения элементов и выполнения других оптимизаций. Интересно заметить, что физический синтез начинается с выполнения первого этапа, в котором используется алгоритм традиционного логического/HDL-синтеза (**Рис. 19.11**).

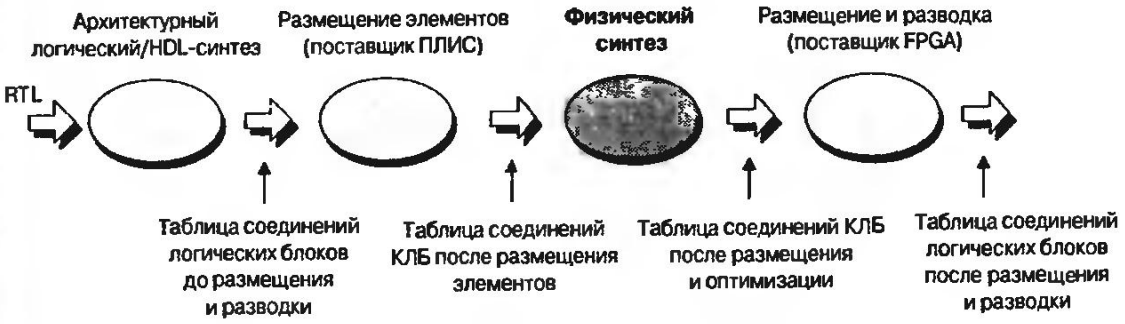


Рис. 19.11. Физический синтез

Коррекция временных параметров, репликация и повторный синтез

В инженерной практике существует ряд терминов, которые можно услышать, когда речь о физическом синтезе, например *коррекция временных параметров*, *репликация*, *повторный синтез*[1]. Коррекция временных параметров основана на концепции балансировки между положительным и отрицательным резервом времени в масштабах всего устройства. Под положительным резервом будем подразумевать неко-

[1] Эти термины могут применяться в традиционном логическом/HDL-синтезе, но они более действенны, когда используются в контексте физического синтеза.

торую величину задержки, которая остается в запасе, а под отрицательным некоторую задержку, которая превышает допустимое значение.

Рассмотрим конвейер, у которого тактовая частота выбрана таким образом, что максимальная задержка между регистрами составляет величину 15 пикосекунд. Допустим, что для этого устройства сложилась ситуация, которая показана на **Рис. 19.12, а**. В этом случае максимальное время распространения сигнала в первом логическом блоке составит 10 пс. Другими словами, положительный резерв времени составит 5 пс, а во втором логическом блоке максимальное время прохождения сигнала будет равно 20 пс, т. е. величина отрицательного резерва составит величину 5 пс.

1934 г. *Половина домов в США оснащены радиоприёмниками.*

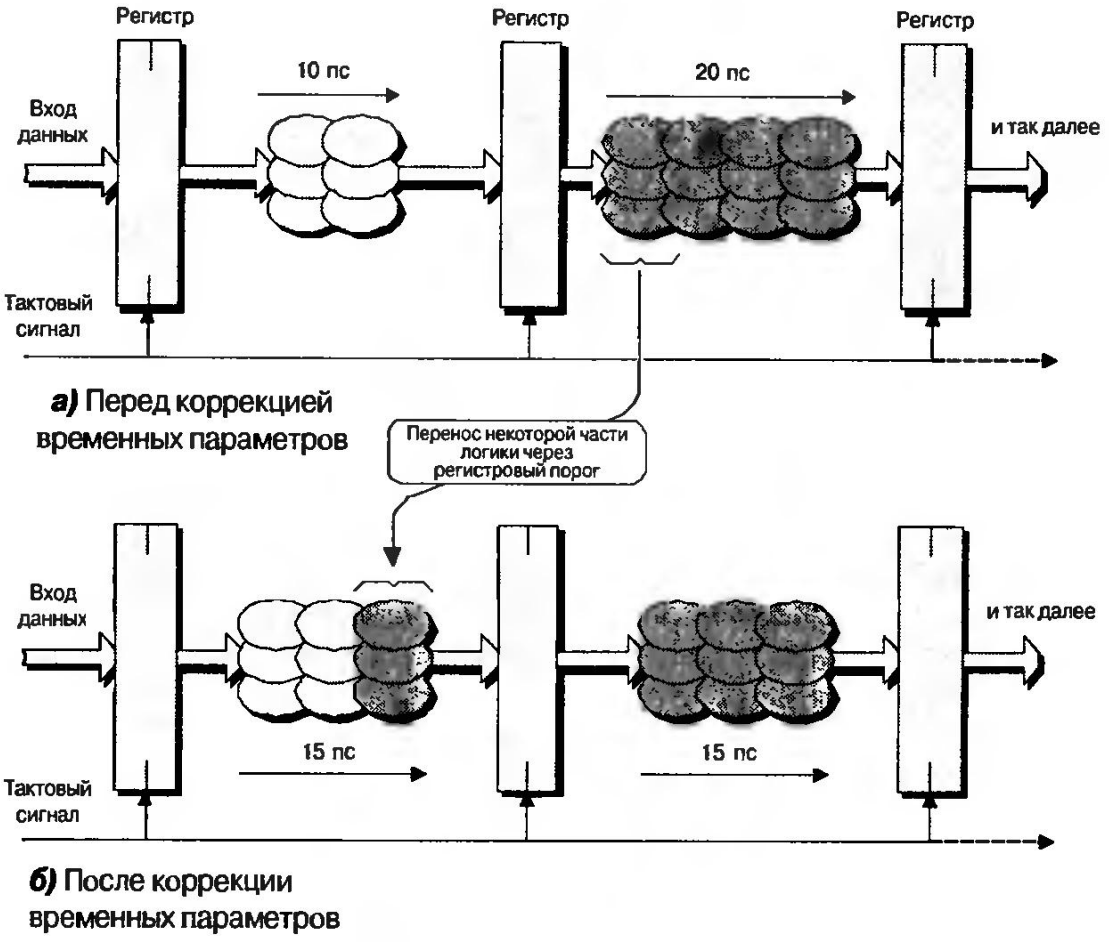


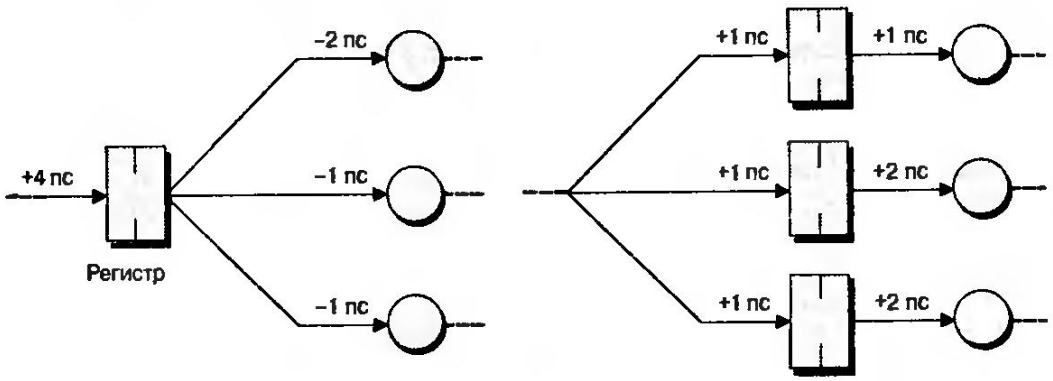
Рис. 19.12. Коррекция временных параметров

После расчёта значений временных параметров логических блоков, в том числе с учётом задержки, вызванной разводкой элементов, часть логики перебрасывается через регистровый порог, тем самым, изымая резервы времени из первого блока и передавая их во второй (**Рис. 19.12, б**). Коррекция временных параметров очень часто используется при проектировании устройств на основе ПЛИС, поскольку в них реализовано очень большое количество регистров.

Процедура репликации аналогична коррекции временных параметров с той лишь разницей, что в этом случае производится разрыв длинных внутренних соединений. Например, имеется регистр с положительным резервом времени 4 пс. Допустим, что к выходу этого регистра подключено три блока, каждый из которых имеет отрицательный временной резерв (**Рис. 19.13, а**).

С помощью репликации регистра и расположения его копий в непосредственной близости к их нагрузке, можно перераспределить временные резервы и сделать все части устройства рабочими (**Рис. 19.13, б**).

1935 г. *Разработан полностью электронный телевизор метрового диапазона волн.*



а) Перед репликацией *б)* После репликации

Рис. 19.13. Репликация

И в завершении замечу, что концепция повторного синтеза основана на большом разнообразии способов реализации и размещения различных функций. При повторном синтезе используется информация о физическом размещении элементов, после чего выполняется локальная оптимизация на критических участках устройства с помощью операций логической реструктуризации, повторной кластеризации, и, возможно, путем исключения вентилей и проводников.

Выбор оптимального средства синтеза!

Неужели вы надеетесь получить здесь ответ на этот вопрос? В реальном мире возможности различных машин синтеза вместе со связанными с ними особенностями, такими как автоинтерактивное компоновочное планирование, меняются почти ежедневно, и разные производители постоянно обходят друг друга в этой гонке.

Существует также мнение, что разные системы работают лучше или хуже с разными архитектурами ПЛИС от разных производителей. Одним из критериев поиска системы синтеза является способность (или отсутствие таковой) автоматически выполнять из исходного кода логический синтез некоторых узлов, например элементов системы синхронизации и встроенных функций, или обеспечивать целостность файлов без необходимости создавать соответствующие описания в явном виде. Однако, в конце концов, придется самому сделать свой выбор, когда наступит время оценивать и сортировать различные предложения. Пожалуйста, не стесняйтесь и сообщите мне о своем решении по адресу max@techbites.com.

Временной анализ. Статический и динамический

Статический временной анализ

В наши дни наиболее распространенной формой временного анализа является статический анализ. Концептуально этот метод довольно прост, хотя на практике, как обычно, все оказывается гораздо сложнее, чем казалось на первый взгляд.

По сушеству, временной анализатор суммирует значения задержек всех логических элементов и проводников и формирует общие значения задержек от входа до выхода микросхемы для каждого канала прохождения сигнала. В устройствах, в которых используются конвейеры, анализатор подсчитывает значение задержек между соседними группами регистров.

Перед размещением элементов и разводкой соединений временной анализатор может оценивать значения задержек, вызванных проводниками устройства. После размещения и разводки для достижения более точных результатов анализатор также будет использовать полученные значения паразитных параметров, т. е. сопротивлений и ёмкостей физических проводников. В результате будет сформирован отчёт с указанием всех путей прохождения сигнала, задержка на которых не соответствует заданным временным ограничениям, а также с предупреждениями о потенциальных временных проблемах, например о нарушении режимов установки и хранения данных, связанных с сигналами, поступающими на вход регистров и защёлок.

Статический временной анализ хорошо подходит для классических синхронизированных схем и конвейерных архитектур. Главное преимущество такого анализа состоит в том, что он достаточно быстрый, для него не требуются дополнительные средства тестирования, и он тщательно проверяет каждый возможный путь прохождения сигнала. Вместе с тем, статические временные анализаторы жульничают при обнаружении ложных путей распространения сигнала, которые никогда не будут использованы в процессе штатной работы устройства. Кроме того, эти средства плохо работают со схемами, в которых используются защёлки, асинхронные цепи и комбинационные обратные связи.

Статистический статический временной анализ

Статический временной анализ составляет основу современных методов проектирования устройств на основе ПЛИС и заказных микросхем (ASIC). Однако при работе с устройствами, изготовленными с помощью последних технологических процессов, в работе этого метода стали возникать некоторые проблемы. Во время написания этой книги для изготовления микросхем использовался технологический процесс 90-нм, а к 2007-му году ожидается переход к 45-нм процессу.

Как уже упоминалось, при использовании современных кремниевых кристаллов, задержка, вызванная распространением сигнала через внутренние соединения, преобладает над задержкой вентилей, особенно в ПЛИС. В свою очередь, задержки внутренних соединений зависят от значений паразитных ёмкостей, сопротивлений и индуктивностей, которые определяются топологией и формой используемых проводников.

Проблема заключается в том, что при реализации современных технологических процессов фотолитографическим способом практически невозможно обеспечить точную форму проводника. Поэтому вместо квадратов и прямоугольников приходится использовать окружности и эллипсоиды. Размеры проводников, например их ширина, теперь настолько малы, что даже небольшие изменения в процессе травления вызывают отклонения, которые хоть и невелики, но достаточно существенны по сравнению с шириной проводника. Эти изменения становятся ещё более значимыми, так как в высокочастотных устройствах вступает в силу так называемый *поверхностный эффект*, смысл которого заключается в том, что высокочастотные сигналы передаются только через внешнюю поверхность проводника. Кроме того, существуют отклонения в вертикальной плоскости поперечного сечения дорожки, обусловленные процессами *химико-механического полирования* или другими подобными процедурами.

В конечном счете, все эти особенности существенно затрудняют определение точных значений временных задержек на внутренних соединениях. Конечно, можно воспользоваться традиционным инженерным методом резервирования параметров, используя наихудшие оценки, но практика чрезмерного консерватизма в сфере проектирования приводит к значительному снижению производительности кристалла и, как следствие, является не очень удачным выбором в условиях жесткой конкуренции на рынке. В реальной жизни разброс геометрических размеров может привести к значительному расширению распределения вероятности ошибки, и в наихудшем случае устройство может оказаться более медленным, чем при использовании даже более ранних технологических процессов!

Потенциальным решением этого вопроса может быть концепция *статистического статического временного анализатора* (или *SSTA — statistical static timing analyzer*). Этот метод основан на формировании функции вероятности задержки для каждого сигнала каждого сегмента проводника, затем, на основе которых, формируются общие функции распределения задержки сигналов, проходящих через все части устройства. Во время написания этой книги не существовало коммерческих средств статистического статического временного анализа, но некоторые компании — разработчики САПР и ряд учебных заведений занимаются проработкой этого вопроса.

Динамический временной анализ

Другая форма контроля временных параметров, известная как *динамический временной анализ*, или DTA — *dynamic timing analysis*, в наши дни не очень-то популярна и упомянута здесь для полноты обзора рассматриваемых средств. Эта форма проверки основана на использовании системы событийного моделирования, и в процессе работы использует набор тестов. В отличие от стандартной системы событийного моделирования, которая использует одно из значений задержки, т. е. минимальное (мин), номинальное (ном) или максимальное (макс), для каждого пути прохождения сигнала, динамический анализатор работает с парой значений задержки, т. е. мин:ном, ном:макс или мин:макс). Например, рассмотрим, как две системы моделирования оценят работу простого буферного вентиля (**Рис. 19.14**).

В случае стандартной системы моделирования изменение сигнала на входе вентиля вызовет событие, которое должно быть спланировано на некоторое время в будущем, а случае временного анализатора, использующего пару мин:макс, изменение сигнала на выходе логического элемента начнёт происходить после истечения времени минимальной задержки и закончится только после достижения значения максимальной задержки.

Неопределенность между этими двумя значениями отличается от неизвестного состояния *X*, так как известно, что будет происходить переход с уровня 0 на уровень 1 или с 1 в 0, но при этом неизвестно время перехода. В связи с этим вводятся два новых состояния, называемых «перешёл в 1, но не знаю когда» и «перешёл в 0, но не знаю когда»[1)].

Динамический временной анализ позволяет обнаруживать неочевидные потенциальные проблемы, которые почти невозможно опре-

---

[1)] Динамический временной анализ более подробно описан в моей книге «Designus Maximus Unleashed (Banned in Alabama)», ISBN 0-7506-9089.

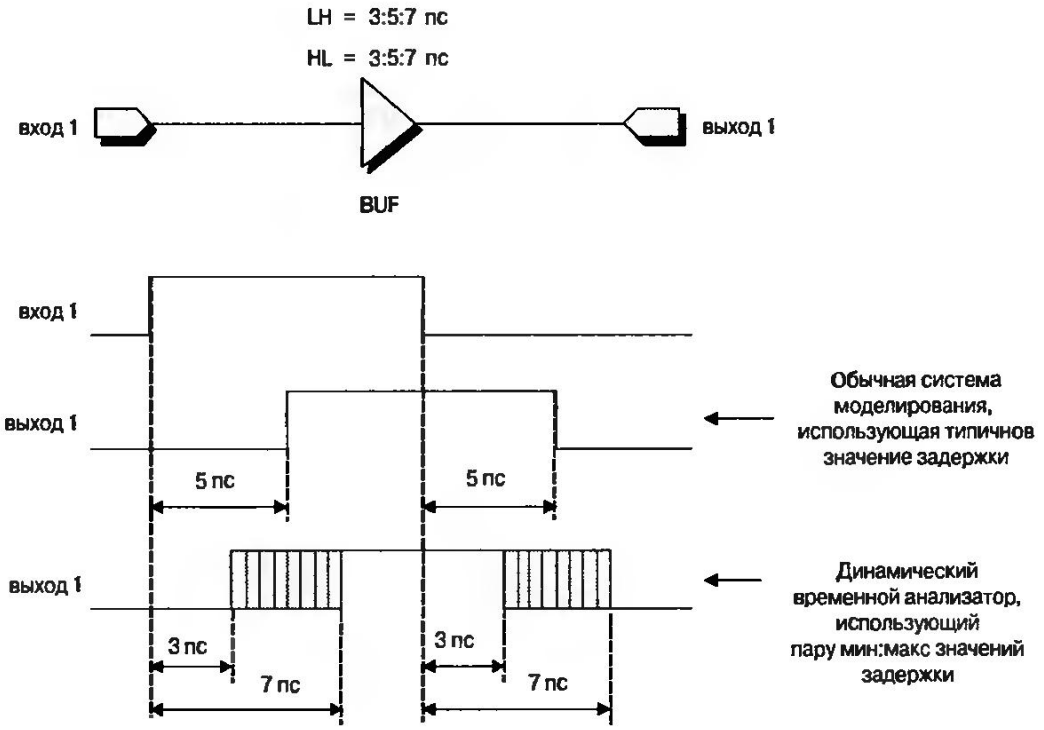


Рис. 19.14. Стандартная система событийного моделирования и динамический временной анализатор

делить другими формами временного анализа. К сожалению, эти средства настолько переполнены вычислениями, что в наши дни они не очень популярны, но кто знает, что день грядущий нам готовит?

Общая верификация

Верификация блоков интеллектуальной собственности

С увеличением сложности устройств все больше ресурсов и времени уходит на проверку их функциональности. Такая проверка включает в себя реализацию проверочной среды, создание набора тестов, выполнение логического моделирования, анализ результатов для выявления и выделения проблем и так далее. На практике, на контроль современных высокотехнологичных однокристальных устройств, ASIC и ПЛИС, может потребоваться свыше 70% времени, потраченного с момента разработки начальной концепции и до окончательной реализации.

Помочь в решении этой проблемы может *верификация блоков интеллектуальной собственности (верификация IP)*. Идея состоит в том, что устройство, которое на этапе верификации называют *проверяемым устройством*, обычно взаимодействует с окружающим миром, используя стандартные интерфейсы и протоколы. Кроме того, проверяемое устройство обычно общается с микропроцессорами, арбитрами, контроллерами, периферийными устройствами и т. д.

В большинстве случае для функциональной проверки используется стандартная система событийного моделирования. Один из способов тестирования проверяемого устройства заключается в создании набора тестов, которые точно описывают сигналы на уровне битовых значений, поступающих на вход устройства и появляющихся на его выходах. Однако в связи со сложностью современных протоколов, применяемых при работе интерфейсов и шин, разработать подобные тесты для большинства устройств просто невозможно.

Другой метод тестирования предполагает использование RTL-моделей всех внешних устройств, формирующих систему. Однако многие из этих устройств запатентованы, и их RTL-модели могут быть недоступны для использования. Более того, моделирование всей системы с использованием полнофункциональных моделей всех процессоров и устройств ввода/вывода может потребовать значительных временных и вычислительных ресурсов.

В качестве решения проблемы можно применить верификацию блоков IP в форме *функциональных моделей шины ФМШ*, или *BFM (bus functional model)*, для представления процессоров и блоков ввода/вывода, формирующих тестируемую систему (**Рис. 19.15**)[1].

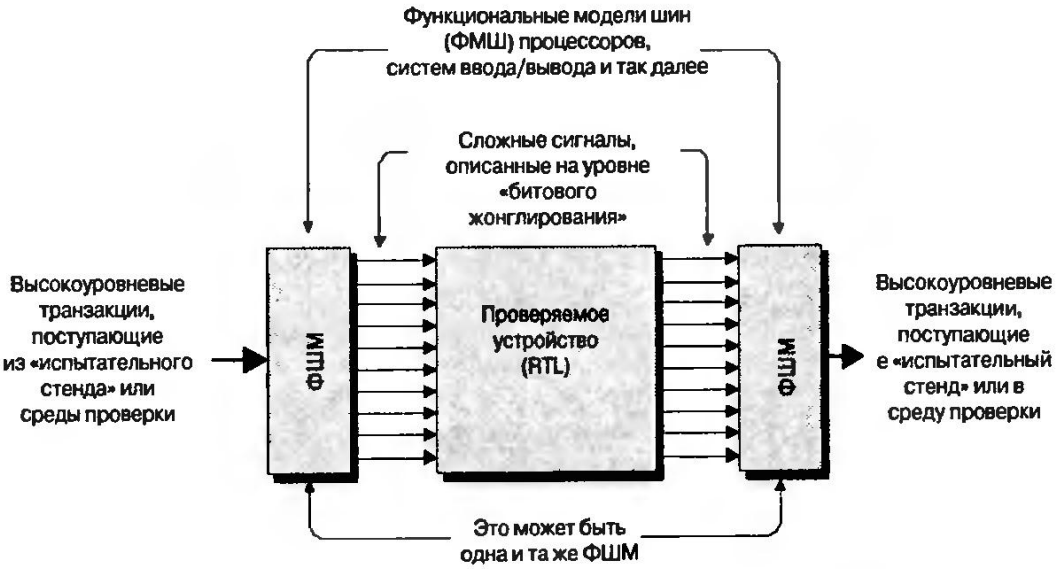


Рис. 19.15. Применение проверки IP в форме ФМШ

Функциональная модель шины не копирует полную функциональность представляемого устройства, вместо этого она эмулирует способ его работы на уровне интерфейса шины путем генерации и приёма транзакций. В контексте рассматриваемого материала термин *транзакция* относится к высокоуровневому событию на шине, например к выполнению цикла чтения или записи. Среда проверки, или испытательный стенд, могут выдавать команды для ФМШ на выполнение транзакций, например чтения содержимого памяти. После этого ФМШ генерирует сложный сигнал, состоящий из битов («битовое жонглирование»[2]), поступающий на интерфейс проверяемого устройства.

Аналогично, когда проверяемое устройство отвечает на задающее воздействие каким-либо сложным сигналом, другая ФМШ, или это может быть та же модель, интерпретирует эти сигналы и переводит их в соответствующую высокоуровневую транзакцию.

Хотя ФМШ намного меньше и проще, и, следовательно, процедура моделирования выполняется намного быстрее, чем полнофункциональные модели представляемых ими устройств, они отнюдь не тривиальны. Например, сложные ФМШ, которые часто создаются в виде потактных, побитовых C/C++ моделей, могут включать в себя внутреннюю кэш-память (вместе с возможностью её инициализа-

---

[1] Примером очень сложного средства верификации блоков IP может служить TransEDA PLC (www.transeda.com).

[2] Битовое жонглирование — программирование на уровне машинных кодов с манипулированием битами, флагами, полубайтами и прочими элементами данных, размером меньше одного байта. — *Прим. пер.*

ции), внутренние буферы, конфигурационные регистры, очереди с отложенной записью и т. д. ФМШ могут также обеспечивать огромный набор параметров, которые позволяют производить низкоуровневое управление такими элементами, как адресная синхронизация, состояния ожидания данных для различных устройств памяти и другими.

1935 г. *В продажу поступила звукозаписывающая лента.*

Среда верификации и разработка тестов

Когда я в молодости начинал работу с системами моделирования, мы разрабатывали для них тестовые вектора, задающие воздействия и реакции на них, в виде табличных текстовых ASCII-файлов, содержащих логические значения 0 и 1 (или шестнадцатеричные значения, если повезёт). В то время устройства, которые мы пытались тестировать, были существенно проще, чем современные монстры. В переводе на человеческий язык эти тесты могли бы быть представлены в виде следующих строк:

- При значении времени 1000 переведите сигнал сброса в активное состояние.
- При значении времени 2000 переведите сигнал сброса в неактивное состояние.
- При значении времени 2500 удостоверьтесь, что на 8-битной шине данных установлено значение 00000000.
- При значении времени ... и так далее.

Со временем устройства усложнялись, и с появлением языков высокого уровня средства их верификации стали более сложными. Языки высокого уровня использовались в системах моделирования для формирования задающих воздействий и ожидаемых откликов, а также для поддержки различных особенностей, таких как циклы и возможности переключать тесты в зависимости от состояния выходов. (Например, если на шине состояния установлено значение 010, тогда переходим к тесту *xy*.) На определённом этапе инженеры начали называть эти тесты *испытательным стендом*.

Если быть более точным, понятие «испытательный стенд» на самом деле относится к инфраструктуре, поддерживающей выполнение этих тестов.

В настоящее время многие современные устройства настолько сложны, что почти невозможно создать адекватный тест вручную. Это обстоятельство послужило поводом для применения сложных сред тестирования и языков высокого уровня. Возможно, самым сложным из языков, известных как *языки верификации аппаратного обеспечения* (*HVL — hardware verification language*), является специализированный продукт под названием *e* от компании Verisity Design (www.verisity.com)[1].

Возможно, что вас удивило это название. На самом деле *e* ничего не обозначает, хотя поначалу у разработчиков было намерение отразить идею его подобия английскому *(english)* языку. При желании можно использовать *e* для написания специализированных тестов, но обычно в таком качестве он применяется только в особых случаях. Вместо этого концепция *e*, которую можно рассматривать как смесь языков C, Verilog и отчасти Pascal, больше подходит для описания допустимых диапазонов и порядка следования входных значений, а так-

[1] Поскольку промышленность с большим подозрением относится ко всем патентованным языкам, компания Verisity Design работает с институтом IEEE над тем, чтобы сделать *e* стандартным промышленным языком. Во время написания этой книги институтом IEEE была образована рабочая группа P1647 и издано справочное руководство по этому языку.

же стратегий верификации высокого уровня. Эти описания воспринимаются средой моделирования для управления непосредственно процессом моделирования.

Необходимо отметить, что первой и единственной во время написания этой книги средой верификации, полностью использующей мощь языка *e*, являлся пакет Specman Elite® компании Verisity. Это пакет можно рассматривать как совокупность компилятора и системы событийного моделирования, которая связывает и управляет используемые вами стандартные событийные HDL-системы моделирования. Пакет Specman с помощью программ на языке *e* на лету генерирует задающие воздействия, и затем подаёт их на вход устройства через используемую HDL-систему моделирования. Пакет производит также наблюдение за результатами и функциональным покрытием моделирования, и по результатам этого наблюдения динамически перестраивает последующие задающие воздействия для тестирования неохваченных участков устройства.

Анализ результатов моделирования

Почти все современные системы моделирования оснащены средствами графического просмотра сигнала, которые могут быть использованы для интерактивного отображения результатов моделирования во время работы системы моделирования или для последующего просмотра этих результатов, сохранённых в файлах VCD-формата (value change dump — дамп изменения значений). Грустно признавать, но некоторые из этих средств не настолько эффективны, как хотелось бы, когда дело доходит до анализа полученной информации и выявления проблем. В этом случае, возможно, следовало бы воспользоваться средствами сторонних разработчиков.

В вопросах классического анализа формы сигнала, средств отладки и отображения информации одним из наиболее известных производителей является компания Novas Software Inc. (www.novas.com) со своим пакетом Debussy®. Другое, заслуживающее внимания средство под названием Verdi™ поддерживает чрезвычайно прогрессивные и мощные методы выделения, визуализации, анализа и отладки устройств в течение большого количества тактовых импульсов.

Формальная верификация

Несмотря на то что большие компьютерные компании и производители микросхем, такие как IBM, Intel и Motorola, десятилетиями разрабатывают и используют свои средства формальной верификации (примерно с середины 80-х), для большинства людей эта область тестирования всё же является новой. Особенно это касается ПЛИС, где применение формальной верификации отстаёт от заказных микросхем. Следует заметить, что формальная верификация может быть настолько мощным средством, что всё больше и больше людей начинают воспринимать её всерьез.

Основная проблема формальной верификации состоит в том, что она ещё не вышла на уровень повсеместного использования, поэтому находится много желающих, которые с радостью бросаются в эту область и теряются в массе различных направлений. Кроме отсутствия стандартов, здесь существует множество предложений, от которых просто голова идёт кругом. При этом почти каждый пытается внести свою лепту в существующую неразбериху, тем самым ещё больше осложняя ситуацию. Например, если попросить 20 поставщиков САПР дать определение терминам «*утверждение*» и «*свойство*» и указать их отличия, вы сойдете с ума из-за диаметрально противоположных ответов[1)].

Попытка распутать этот клубок окажется трудноразрешимой задачей, если вообще разрешимой. Однако, как любил говаривать мой дед,

---

[1)] Я убедился в этом на собственном горьком опыте.

у страха глаза велики, поэтому давайте ещё раз попытаемся приоткрыть завесу и описать формальную верификацию понятным для всех способом.

Средства формальной верификации изначально разрабатывались большими компьютерными компаниями и производителями микросхем для своего внутреннего пользования. Одним из первых коммерческих средств такого типа был программный пакет проверки на эквивалентность Design VERIFYer, разработанный в 1993 г. компанией Chrysalis Symbolic Design Inc. Средства проверки на эквивалентность изначально разрабатывались большими компаниями для своего внутреннего применения и первая коммерческая утилита проверки моделей также была разработана компанией Chrysalis в 1996 и получила название Design inSIGHT.

Особенности формальной верификации

Еще совсем недавно большинство разработчиков рассматривали термин *формальная верификация* как синоним *проверки на эквивалентность*. В рамках рассматриваемого материала проверка на эквивалентность представляет собой средство, использующее формальные, строгие математические методы сравнения двух разных представлений устройства, скажем RTL-описания и таблицы соединений вентилей. Такая проверка проводится для определения, имеют эти представления одинаковую функциональность от входов до выходов или нет.

В сущности, проверка на эквивалентность может рассматриваться как подвид формальной верификации, называемый *верификация модели (Model checking)*[1]. Эта проверка относится к методам, используемым для анализа пространственного состояния системы с целью проверки достоверности её определенных характеристик, которые обычно описываются в виде *утверждений*.

И в завершении этих рассуждений уточню: под формальной верификацией мы будем понимать именно верификацию модели. Также замечу, что существует и другая категория формальной верификации, известная как *автоматизированные рассуждения*, которые для тестирования устройств используют логику, больше напоминающую формальное математическое доказательство, и применение этого вида тестирования имеет свои характерные особенности.

Что такое формальная верификация, и чем она хороша

Предположим, что имеется устройство, состоящее из нескольких блоков, и в текущий момент мы работаем с одним из этих блоков, который выполняет некоторую специфичную функцию. Кроме RTL-представления, определяющего функциональность этого блока, с ним можно связать одно или несколько утверждений или свойств. Эти утверждения/свойства могут быть связаны с сигналами интерфейса этого блока либо с сигналами или регистрами внутри блока.

Очень простое утверждение/свойство может соответствовать строкам вида: «Сигналы А и Б никогда не будут активными одновре-

---

[1] Термин Model Checking (проверка модели или верификация модели) получил широкое распространение в 90-х годах двадцатого столетия и обозначает проверку систем алгоритмическими методами на формальное соответствие спецификации. Книга, целиком посвященная этой концепции, была переведена и издана на русском языке в 2002 г. (Кларк Э.М. и др. «Верификация моделей программ: Model Checking»). — *Прим. ред.*

1935 г. *Англия. Демонстрация первого радара.*

1936 г. *Эксперт в области эффективности Август Дворак (August Dvorak) изобрёл новый тип клавиатуры, названной клавиатурой Дворака, в которой символы располагались по частоте их встречаемости.*

менно». Но эти выражения могут также расширяться до чрезвычайно сложных конструкций уровня транзакций, например: «При получении команды на запись от шины PCI соответствующая команда записи в память вида *xxxx* должна быть выдана в течение от 5 до 36 тактов».

Таким образом, утверждения/свойства позволяют описывать поведение контролируемой по времени системы в формальной и строгой форме. Это обеспечивает точное и всестороннее представление о предназначении устройства. (А теперь попробуйте повторить это скороговоркой.) Кроме того, утверждения/свойства могут использоваться для описания как ожидаемых, так и запрещенных сценариев поведения устройства.

Тот факт, что утверждения/свойства могут читать как человек, так и машина, делает их идеальными для описания выполняемых спецификаций, но на этом их применение не ограничивается.

Давайте вернемся к рассмотрению очень простого утверждения/свойства вида: «Сигналы А и Б никогда не будут активными одновременно». Здесь должен «прозвучать» термин *верификация на основе утверждений*, который постоянно присутствует в разговорах о моделировании, статической формальной верификации и динамической формальной верификации. При *статической формальной верификации* соответствующее средство считывает функциональное описание устройства, обычно на уровне абстракции регистровых передач (RTL), и затем полностью анализирует его логику, чтобы гарантировать, что именно это частное условие никогда не наступит. При *динамической формальной верификации* соответствующим образом расширенная система логического моделирования будет работать до определенного момента, затем прервется и автоматически запустит соответствующее средство формальной верификации.

Разумеется, утверждения и свойства могут быть связаны с устройством на любом уровне, т. е. могут относиться к отдельным блокам, к нескольким соединённым по какому-либо интерфейсу блокам или ко всей системе. Эта особенность предусматривает важную процедуру, которая называется *повторная верификация*. До появления формальной верификации повторная верификация использовалась довольно редко. Например, при продаже ядро IP обычно снабжается соответствующими средствами тестирования, которые анализируют сигналы ввода/вывода непосредственно на его входах и выходах. Эти средства позволяют проверять отдельное ядро, но как только ядро будет интегрировано в устройство, поставляемые с ним средства тестирования становятся бесполезными.

Теперь рассмотрим ядро IP, которое оснащено набором предопределённых утверждений/свойств вида: «Сигнал А никогда не будет осуществлять переход из 0 в 1 в течение трех тактов после активации сигнала Б». Эти утверждения и свойства обеспечивают прекрасный механизм взаимодействия от разработчика IP до конечного пользователя. Более того, утверждения и свойства остаются в силе и могут быть восприняты средствами проверки даже после встраивания ядра IP в разрабатываемое устройство.

Что касается утверждений и свойств, связанных с внешними входами и выходами системы, среда верификации может использовать их для автоматической генерации задающих воздействий, подаваемых на вход системы. Кроме того, можно использовать утверждения и свойства для анализа расширенного кода и функционально покрытия, чтобы гарантировать выполнение характерной последовательности действий или выполнения ряда условий.

Термины и определения

Получив общее представление о верификации модели (Model checking), можно поговорить о терминах и определениях. Честно говоря, эти понятия пока представляют собой мало исследованную область и были тщательно отобраны в процессе общения со многими людьми. Другими словами, была сделана попытка отделить зерна от плевел.

- *Свойства и утверждения* — термин *свойство* пришел из области верификации модели (Model checking) и означает характерное функциональное поведение устройства, которое желательно (формально) проверить, например: «После запроса мы ожидаем ответа в течение 10 тактов».
  Термин *утверждение* пришел из области моделирования и означает специфическое функциональное поведение устройства, которое желательно наблюдать в процессе моделирования, и сигнализирует об ошибке, если такое утверждение «срабатывает».
  В наши дни при использовании формальных методов и средств моделирования в унифицированных средах термины «свойство» и «утверждение» взаимозаменяемые, т. е. свойство может выступать в качестве утверждения и наоборот. В общем случае под понятием «свойство/утверждение» мы понимаем формулирование специфических атрибутов, связанных с допустимым устройством или объектом. Таким образом, свойства/утверждения могут использоваться в качестве средств проверки и (или) наблюдения или в качестве объектов формальных доказательств, а часто для выявления и локализации недопустимого поведения устройства.
- *Ограничения* — термин пришел из области верификации модели (Model checking). В процессе формальной верификации модели устройства рассматриваются все возможные допустимые входные комбинации. Следовательно, часто возникает необходимость ограничивать входные воздействия по каким-либо признакам, в противном случае средства проверки будут сигнализировать об ошибочных отказах и нарушениях свойств устройства, которые обычно не возникают в реальных устройствах. Как и свойства, *ограничения* также могут быть простыми и сложными. В некоторых случаях ограничения могут быть интерпретированы как свойства, которые подлежат доказательству. Например, входное ограничение первого модуля может быть также свойством выходного сигнала второго модуля, при этом выход второго модуля подключен к входу первого. Поэтому свойства и ограничения могут иметь двойственную природу.
  Термин *ограничение* также используется в системах ограниченного стохастического моделирования, и в этом случае ограничение обычно применяется для обозначения диапазона значений, которые могут быть использованы для управления шиной.
- *События* — отчасти напоминают утверждения и свойства, и в общем случае могут рассматриваться как подмножество утверждений или свойств. Однако если утверждения и свойства обычно используются для выделения недопустимого поведения устройства, события могут использоваться для определения требуемого поведения устройства в целях обеспечения анализа функционального охвата. В некоторых случаях утверждения/ свойства могут состоять из последовательности

1936 г. *Америка. Психолог Бенджамин Бурак (Benjamin Burack) сконструировал первую электрическую логическую машину (но информацию о ней не публиковал вплоть до 1949 г.).*

1936 г. *Осуществлён первый электронный синтез речи.*

событий. События могут также использоваться для определения интервала, внутри которого должно проверяться утверждение/свойство. Например, «после *а*, *б*, *в* мы предполагаем *г*, пока не произошло *д*», где *а*, *б*, *в* и *д* — события, а *г* — поведение устройства, которое надо контролировать. Отслеживание имеющих место событий и утверждений/свойств позволяет получать количественные данные о том, какие «тупиковые» ситуации и другие характерные состояния устройства были проверены. Статистика по событиям и утверждениям/свойствам может быть использована для определения функционального охвата показателей устройства.

- *Процедурный* — этот термин употребляется по отношению к утверждениям, свойствам, событиям и ограничениям, описанным в контексте исполняемого процесса или набора последовательных выражений, например, в виде процесса VHDL или блока языка Verilog (поэтому их иногда называют внутриконтекстными свойствами/утверждениями). В этом случае утверждение/свойство является встроенным в логику устройства, и в дальнейшем будет оцениваться исходя из образа действия набора последовательных операторов.
- *Декларативный* — относится к утверждениям/свойствам, событиям и ограничениям, которые существуют в структурном контексте устройства и оцениваются вместе со всеми другими структурными элементами. Например, это может быть модуль, принимающий формы структурного элемента. Вместе с тем, декларативные утверждения/свойства **всегда** «активны/включены», а их процедурные аналоги активируются только при выполнении определённой части HDL-кода.
- *Прагмы* — сокращение словосочетания *«прагматическая информация»*; относится к специальным директивам в виде псевдокомментариев, которые могут интерпретироваться и использоваться различными анализаторами, компиляторами и другими средствами. (Это универсальный термин, и прагма-методы широко используются не только при формальной верификации.)

Альтернативные методы описания утверждений/свойств

Вот с этого места и начинается полная неразбериха, потому что, как будет показано, существует множество способов реализации утверждений/свойств, описывающих устройство.

- *Специальные языки* — в этом случае используются формальные языки, которые разработаны специально для максимально эффективного описания утверждений/свойств. Языки этого типа, к которым относятся Sugar, PSL и OVA, отличаются широкими возможностями при создании сложных регулярных и временных выражений, и с помощью очень компактного кода позволяют описывать сложные поведенческие алгоритмы. Эти языки часто используются для определения утверждений/свойств, которые располагаются в отдельных файлах и не входят в главное HDL-описание устройства. Файлы могут быть доступными в процессе компилирования, а также реализовываться в описательной форме. Кроме того, анализаторы кода, компиляторы и системы моделирования могут быть дополнены возможностями поддержки выражений, написанных на

специальных языках. Это позволяет напрямую встраивать их в строки HDL-кода или в прагмы (определение «прагмы» смотрите в предыдущем подразделе). В подобных случаях выражения могут быть реализованы в декларативной или процедурной форме.

- *Специальные операторы языка HDL* — изначально язык VHDL поддерживал простые операторы контроля, которые проверяли значения булевых выражений и отображали определённые пользователем текстовые строки, если выражение принимало значение *false (ложь)*. Изначально Verilog не поддерживал операторов проверки, но его последующая реализация SystemVerilog была дополнена такой возможностью. Недостаток данного подхода заключается в относительной простоте операторов контроля (в сравнении со специальными языками утверждений/свойств). Кроме того, эти операторы плохо подходят для описания сложных временных последовательностей (хотя для этих целей SystemVerilog в некоторой степени превосходит VHDL).
- *Модели, написанные на HDL и вызываемые из HDL* — эта концепция основана на доступе к библиотекам внутренних и внешних моделей. Эти модели представляют собой утверждения/свойства, использующие стандартные выражения языка HDL, и могут быть представлены в устройстве как любые другие блоки. Однако эти блоки будут свернуты прагмами синтеза вида «Вкл/Выкл», чтобы быть уверенным, что они не будут реализованы физически. Хорошим примером этого метода может служить *открытая библиотека средств проверки (OVL)*, разработанная компанией Accellera (www.accellera.org).
- *Модели, созданные на HDL с доступом через прагмы* — эта концепция аналогична концепции предыдущего метода, так как включает библиотеки моделей, представляющие собой утверждения и свойства, использующие стандартные HDL-выражения. Однако вместо того чтобы вызывать эти модели напрямую из главного кода устройства, обращение к ним производится с помощью прагм. Хорошим примером этой методики является библиотека CheckerWare® компании 0-In Design Automation (www.0-In.com). Например, рассмотрим устройство, содержащую строку кода Verilog:

  reg [5:0] STATE_VAR; //0in one_hot.

  Левая часть этого выражения объявляет 6-битный регистр, названный STATE_VAR, который, допустим, будет использоваться для хранения переменных состояния, используемых в конечных автоматах. Тем временем, правая часть, которая обозначена как «0in one_hot», является прагмой. Большинство средств будут воспринимать эту прагму как комментарий, и игнорировать её, но средства компании 0-In будут использовать её для вызова соответствующей «one_hot» модели (которая является схемой прямого кодирования) свойств/утверждений из библиотеки CheckerWare. Замечу, что при использовании данного выражения не требуется определять переменную, тактовый сигнал или ширину шины этого утверждений, так как информация такого рода собирается автоматически. Кроме того, в зависимости от положения прагмы в коде, она может реализовываться в декларативной или процедурной форме.

1936 г. *Начато применение люминесцентного освещения.*

1936 г. *Олимпиада в Мюнхене впервые транслировалась по телевидению.*

1937 г. *Джордж Стибиц (George Robert Stibitz) сотрудник лаборатории Bell Labs, сконструировал простой, работающий на реле, цифровой калькулятор под названием «Model K».*

Статическая и динамическая формальная верификация

Данная тема может оказаться несколько трудной для восприятия, поэтому будем погружаться в нее постепенно. Во-первых, в среде моделирования можно использовать утверждения/свойства. В том случае, когда эти свойства и утверждения означают, что «сигналы A и B никогда не будут активными одновременно», и когда такая недопустимая ситуация всё-таки имеет место во время моделирования, на экране появится соответствующее предупреждение и данное событие будет зарегистрировано в журнале.

Альтернативой системам моделирования может служить *статическая формальная верификация*. Эти средства отличаются высокой точностью и позволяют проверять всё пространство состояний устройства без применения систем моделирования. Их недостаток кроется в том, что обычно они могут использоваться только для небольших частей устройства, так как пространство состояний с увеличением сложности возрастает по экспоненте и можно очень быстро дойти до так называемого *разрыва пространства состояний*. В отличие от статической формальной верификации системы логического моделирования, которые также могут быть использованы для проверки утверждений, могут охватывать огромные устройства, но для их работы требуются задающие воздействия и применять их можно не в каждой ситуации.

Например, можно использовать моделирование до тех пор, пока не будет достигнуто определённое тупиковое состояние, затем сделать паузу и автоматически вызвать алгоритм статической формальной верификации для более тщательного обследования этого состояния. В контексте рассматриваемого материала понятия «тупиковое» состояние и тупиковая ситуация характеризуют трудновыполнимые и труднодоступные функциональные состояния устройства. После оценки «тупикового» состояния управление автоматически вернётся к системе моделирования для выполнения дальнейшей работы. Эта комбинация системы моделирования и традиционной статической формальной верификации называется *динамической формальной верификацией* .

В качестве простого примера применения динамической формальной верификации рассмотрим память типа FIFO (очередь вида «первым пришёл, первым вышел»), в которой состояния «заполнена» и «пустая» могут считаться тупиковыми ситуациями. Для заполнения такой очереди потребуется много тактов, поэтому состояния «заполнена» можно достичь с помощью системы моделирования. Но точная оценка утверждений/свойств, связанных с этими тупиковыми ситуациями, например факт того, что в память FIFO невозможно записать какие-либо данные, пока не освободится место, лучше всего достигается с помощью статических методов.

И еще хороший пример такого средства динамического верификации предоставляет компания 0-In. Библиотека моделей CheckerWare содержит подробные описания тупиковых ситуаций. Если в процессе моделирования достигается положение тупикового состояния, система останавливается, и для детального анализа этого случая используется средство статической верификации.

Обзор других языков

Содержимое этого подраздела при неаккуратном подходе может вызвать некоторую путаницу, поэтому давайте будем аккуратными. Начнём обзор с продукта под названием Vera®, который появился на

свет в компании Sun Microsystems в начале 90-х. Впоследствии, где-то в середине 90-х, он был предоставлен компании Systems Science Corporation, которая в 1998 году была приобретена компанией Synopsys.

По сути Vera есть не что иное, как среда полной верификации, похожая, но, возможно, не такая сложная, как язык или среда верификации под названием *e*. Среда Vera поддерживает возможности формирования наборов тестов и утверждений, и компания Synopsys продвигает ее как функционально законченный продукт с интеграцией в свою систему логического моделирования. Позже, по заявкам пользователей, Synopsys открыла свой продукт для стороннего использования под марками OpenVera™ и OpenVera Assertions (или OVA[1]).

Примерно в это же время язык SystemVerilog после очередной модификации обзавелся оператором assert (утверждение). Тем временем, благодаря возрастающему интересу к технологии формальной верификации один из комитетов по стандартам компании Acceller занялся поиском языка формальной верификации, который можно было бы принять в качестве производственного стандарта. При этом рассматривались несколько вариантов, включая OVA, но в 2002 году комитет, наконец, отдал предпочтение языку Sugar от IBM. Забавно, что через какое-то время компания Synopsys подарила OVA одному из комитетов Accellera, ответственному за язык SystemVerilog (это была другая комиссия, которая оценивала формальные свойства языков).

Кроме того, одному из комитетов компании Accellera удалось стать ответственным за так называемую *библиотеку открытой проверки* (или *OVL — open verification liblary*), которая является библиотекой моделей утверждений и свойств для языков VHDL и Verilog 2K1.

В связи с упомянутыми событиями, теперь операторы assert поддерживаются языками VHDL и SystemVerilog, библиотекой моделей OVL, языком утверждений OVA и специализированным языком описания свойств PSL (property specification language), который представляет собой версию языка Sugar компании IBM, переработанную компанией Accellera (**Рис. 19.16**)[2]. Преимущество языка PSL заключается в том, что он достаточно самостоятельный и может использоваться независимо от языков, применяемых для представления функциональности устройства. К его недостаткам можно отнести то, что он не похож на языки описания аппаратных средств, к примеру, на VHDL, Verilog, C/C++ и другие, с которыми хорошо знакомы разработчики. Необходимо также упомянуть о появлении различных версий языка PSL, таких как Verilog PSL, VHDL PSL, SystemC PSL и других. В этих версиях его синтаксис отличается от оригинала и соответствует целевому языку, а вот семантика идентична исходной версии.

---

[1] Изначально пакет OVA был позаимствован из технологии моделирования компании Synopsys. Позже было принято решение о его расширении для поддержки средств формальной верификации на основе свойств. Поэтому Synopsys начала сотрудничать с компанией Intel, чтобы воспользоваться опытом её сотрудников в сфере формальной верификации, работавших с языком внутреннего использования на основе утверждений под названием ForSpec. В результате появился пакет OVA 2.0, который объединил в себе мощные модули статической и динамической формальной верификации.

[2] Не следует считать, что PSL и Sugar — это один комбинированный язык. Есть язык PSL, а есть Sugar, и это не одно и то же. PSL представляет собой стандарт компании Accellera, а Sugar — язык, используемый компанией IBM.

1937 г. *Разработаны принципы импульсно-кодовой модуляции (ИКМ) для цифровой радиопередачи сигналов.*

1938 г. *Американец Клод Шеннон (Claude E. Shannon) опубликовал статью, основанную на его работах в Массачусетском технологическом институте, в которой описывались принципы построения цифровых схем на основе булевой алгебры.*

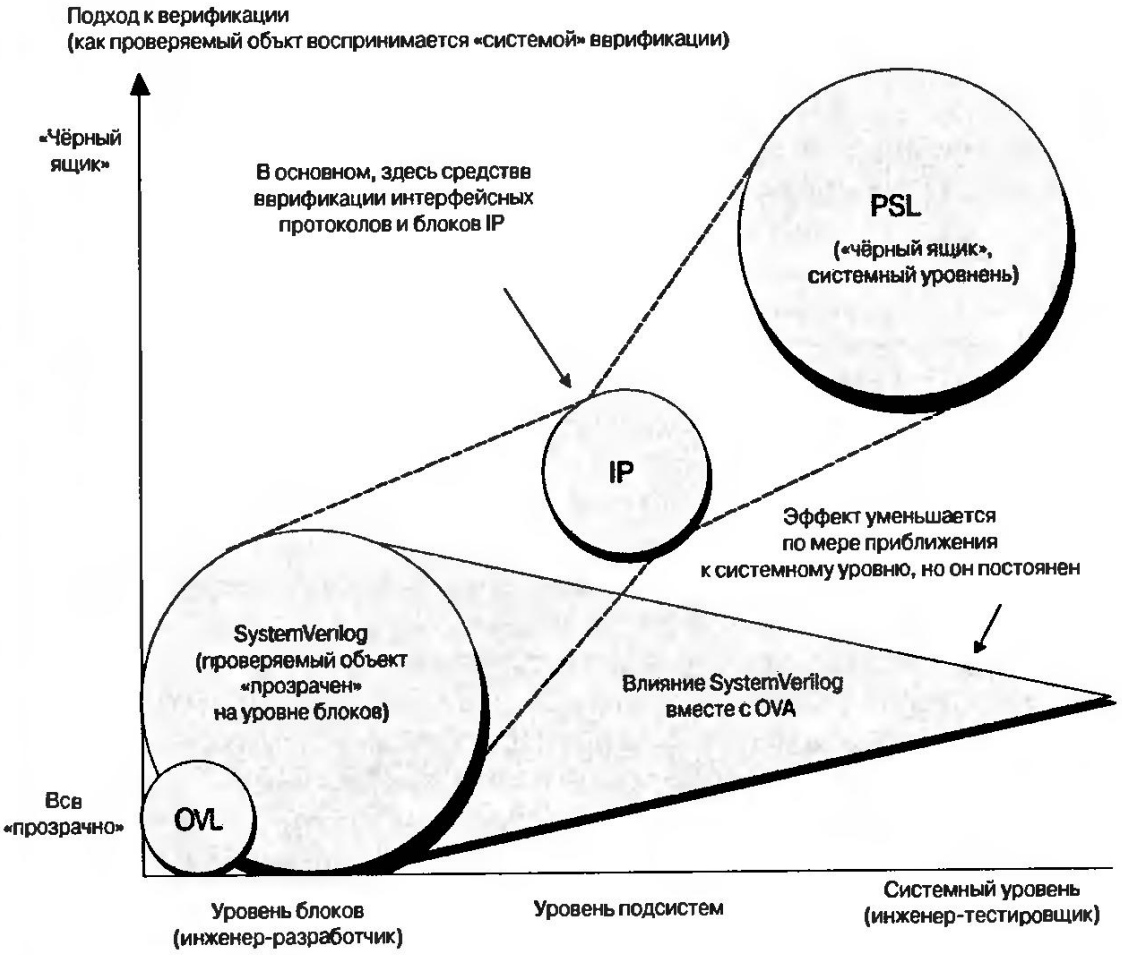


Рис. 19.16. Попытка окинуть все одним взглядом

Рис. 19.16 отражает только одну точку зрения, и не каждый с ней согласится: кто-то будет считать его блестящим разрешением излишне запутанной ситуации, а кто-то в лучшем случае посчитает его грубым упрощением или в худшем — полной чепухой.

Разное

Преобразование из HDL в C

Как уже обсуждалось в гл. 11, в наши дни наблюдается тенденция описывать устройства на высших уровнях абстракции, например, с помощью языка C/C++. Кроме более лёгких архитектурных исследований, высокоуровневые (поведенческие и (или) алгоритмические) модели, описанные на языке C/C++ могут моделироваться в сотни и тысячи раз быстрее, чем их аналоги, созданные в HDL-коде.

Несмотря на всё вышесказанное, многие инженеры продолжают работать с привычными для них RTL-описаниями. При этом надо иметь в виду, что в случае моделирования однокристальной системы со встроенным микропроцессорным ядром, памятью, периферией и другой логикой, описанной на уровне регистровых передач, пользователю крупно повезет, если скорость моделирования будет составлять пару герц (что составляет пару тактов системы синхронизации за каждую секунду реального времени).

Чтобы решить эту проблему, некоторые компании САПР электронных устройств начали предлагать способы преобразования пользовательских «эталонных» RTL-моделей в высокоскоростные аналоги, которые могут достичь скорости моделирования в несколько кило-

герц[1]. Этого достаточно, чтобы программные модули выполнялись на представленном (виртуальном) аппаратном обеспечении за единицы миллисекунд реального времени. В свою очередь, это обстоятельство позволяет тестировать критичное базовое программное обеспечение, например драйверы, модули диагностики, встроенные микрокоды, вследствие чего проверка системы будет выполняться быстрее, чем при использовании традиционных методов.

Кодовое покрытие

Не так давно средства оценки кодового покрытия, или анализа структуры кода, разрабатывались сторонними производителями САПР электронных систем. Однако сегодня эта возможность считается настолько важной, что все большие компании встраивают средства оценки кодового покрытия в свои среды верификации/моделирования, но, естественно, набор возможностей у таких средств сильно меняется в зависимости от предложения.

Не удивляйтесь, если узнаете о существовании разновидностей кодового покрытия. Их типы в порядке возрастания сложности приведены в нижеследующем списке:

- *Базовое кодовое покрытие (Basic code coverage)* — просто анализ строк исходного кода, т. е. сколько раз каждая строка исходного кода выполнялась.
- *Покрытие ветвей исходного кода (Brunch coverage)* — анализируются выражения, подобные оператору *if-then-else*: сколько раз выполнялась часть *then*, и сколько раз *else*.
- *Покрытие условий (Condition coverage)* — относится к операторам условий, записанных в виде *«if (a OR b == TRUE) then ...»*. В этом случае речь идет о том, сколько раз выполнялся оператор *then* вследствие того, что переменной *a* было присвоено значение *TRUE*, и сколько раз из-за того, что это значение соответствовало переменной *b*.
- *Покрытие выражений (Expression coverage)* — в этом случае имеем выражения вида *«a = (b AND c) OR d»*, а именно, речь идет обо всех возможных комбинациях входных значений, какие из них инициируют изменение выходного сигнала и какие переменные не были протестированы.
- *Покрытие состояний (State coverage)* — означает анализ конечных автоматов и определение, какие состояния были ими задействованы, а какие нет, также какие защитные условия и пути между этими состояниями были задействованы, а какие нет и так далее. Вы можете получить эту информацию из анализа структуры строк, но должны читать «между строк».
- *Функциональное покрытие (Functional coverage)* — анализу подвергаются события на уровне транзакций (например, транзакции чтения или записи в память) и особые комбинации и перестановки этих событий.
- *Покрытие утверждений/свойств (Assertion/property coverage)* — относится к среде верификации, которая может собирать, организовывать и делать доступными для анализа результаты рабо-

1938 г. *Аргентина. Выходец из Болгарии Лазро Биро (Lazro Biro) разработал и получил патент на первую шариковую ручку.*

1938 г. *Германия. Конрад Зюс (Konrad Zuse) сконструировал первый работающий механический цифровой компьютер, названный ZI.*

1938 г. *Джон Байрд (John Logie Baird) продемонстрировал работу цветного телевидения.*

1938 г. *Америка. Трансляция радиоспектакля «Война миров» стала причиной распространившейся паники.*

---

[1] Интересным решением является транслятор VTOC™ (из языка Verilog в C) от компании Tenison Technology Ltd. (www.tenison.com). Также интересны продукты SPEEDCompiler™ и DesignPlayer™ компании Carbon Design Systems Inc. (www.carbondesignsystems.com).

1938 г. *Телевизионный сигнал может быть записан на пленку и подвержен редактированию.*

1938 г. *Вальтер Шоттки (Walter Schottky) открыл существование дырок в зонной структуре полупроводников и объяснил принципы построения диодов на основе технологии металл-полупроводник.*

ты различных событийных, формальных статических и динамических машин верификации на основе свойств и утверждений. Эта форма анализа на самом деле может быть разделена на две части: *анализ на уровне спецификации* и *анализ на уровне реализации*. В данном контексте, анализ на уровне спецификации измеряет активность средств верификации по отношению к элементам высокоуровневым функциональным или макроархитектурным определениям. К ним относятся поведение систем ввода/вывода устройства, типы допустимых транзакций (включая взаимоотношение между собой транзакций разного типа) и требуемые преобразования данных. Для сравнения, анализ на уровне реализации измеряет активность средств верификации по отношению к микроархитектурным деталям реального воплощения. К ним относятся инженерные решения, встроенные в RTL, которые являются результатом специфичных тупиковых ситуаций, например, ситуации, связанные с опустошением и переполнением буфера FIFO. Подобные детали реализации редко видны на уровне спецификации.

Анализ производительности

И последней, важной характеристикой современных систем верификации является возможность выполнять *анализ производительности*. Это свойство относится к некому анализу и формированию отчётов о том, где и сколько времени система моделирования затратила на свою работу. Сформированный таким образом отчёт позволяет сфокусироваться на высокоактивных областях устройства и, следовательно, достичь высоких показателей производительности системы.