Серия «Библиотека профессионала»

Е. З. Перельройзен

Проектируем на VHDL

Москва
СОЛОН-Пресс
2016

Серия «Библиотека профессионала»

Е. З. Перельройзен

Проектируем на VHDL

Москва
СОЛОН-Пресс
2016

УДК 681.3
ББК 32.973.26-018.2
П27

Е. З. Перельройзен

П27 **Проектируем на VHDL** — М.: СОЛОН-Пресс, 2016. — 448 с.: ил. — (Серия «Библиотека профессионала»).

ISBN 5-98003-113-8

Книга посвящена проектированию цифровых систем с помощью языка описания аппаратуры VHDL (VHSIC Hardware Description Language).

Первая часть книги описывает процесс проектирования на языках описания аппаратуры.

Во второй части книги рассматривается работа с VHDL в различных средах проектирования: ModelSim (Mentor Graphics), Active HDL (Aldec), OrCAD (Cadence), Warp (Cypress Semiconductor), Foundation Series (Xilinx) и Symphony (Symphony EDA).

Третья часть книги содержит VHDL-модели ряда комбинационных и последовательностных цифровых схем.

Предполагается знакомство читателя с основами программирования и проектирования цифровых устройств.

Книга написана на основе преподавания курса языка VHDL и его приложений к моделированию цифровых систем в Еврейском университете (Иерусалим), Хайфском университете и филиале английского университета Ковентри в Израиле.

ISBN 5-98003-113-8

Предисловие автора

Весной 1992 г. автор держал в руках только что вышедшую из печати первую книгу о языке описания аппаратуры VHDL на русском языке. Это был перевод книги Джеймса Армстронга (см. [2р] в списке литературы на русском языке). Автор, воспитанный с 1976 г. на языках описания аппаратуры советских систем проектирования МОДЭПС, РАПИРА и КОНДИЦИЯ, был восхищен красотой нового для него языка. Однако в России 1992 г. мало у кого была возможность работать с VHDL, его компиляторами и симуляторами. Такая возможность появилась у автора лишь в 1997 г. А с 2001 г. и по настоящее время автор преподает курс языка VHDL и его приложений к моделированию цифровых систем в Еврейском университете (Иерусалим), Хайфском университете и филиале английского университета Ковентри в Израиле. Данная книга является кратким изложением этого курса.

Книга состоит из трех частей. Первая часть (главы 1—3) описывает процесс проектирования цифровых систем на языках описания аппаратуры, а также особенности языка VHDL. Вторая часть (главы 4—9) посвящена работе с VHDL в различных средах моделирования и синтеза: ModelSim, Active HDL, OrCAD, Warp, Foundation Series и Symphony. Третья часть (главы 10—12) содержит VHDL-модели ряда комбинационных и последовательностных цифровых схем, а также «учебного» микропроцессора.

Библиография к книге содержит два раздельных списка литературы: на русском и английском языках. Здесь приведены все известные автору книги на русском языке о VHDL, а также те книги на английском языке, которые использовались автором при подготовке учебных курсов и данной книги.

Автор выражает свою благодарность жене Гале и дочери Лизе за возможность работать над книгой сколько ему вздумается, а также старому белому коту Боре, который, разлегшись на письменном столе, дарил автору бесценный заряд положительных эмоций.

Автор будет благодарен читателям за замечания по содержанию книги, которые он просит направлять в адрес издательства.

Евгений Перельройзен
г. Герцелия, Израиль
Август 2003 г.

Часть 1. Концепции проектирования цифровых систем

Глава 1. Современные методологии проектирования цифровых систем с использованием языков описания аппаратуры

1.1. Языки описания аппаратуры

Языки описания аппаратуры (ЯОА, HDLs — Hardware Description Languages) изменили мир проектирования в электронике цифровых систем: появилась возможность запрограммировать в виде интегральных схем многие тысячи вентилей (Gates) и триггеров (FFs — Flip Flops) с помощью персональных компьютеров в течении нескольких минут (4a, 5a, 9a, 12a, 15a, 16a, 41a, 42a, 47a, 61a, 62a, 65a, 2p).

Это означает, что возможно автоматически сгенерировать файл для программирования микросхем (ПЛИС) из описания на ЯОА. Такой метод называется быстрым макетированием (Rapid Prototyping). Он может быть использован для производства малых серий микросхем.

Описание аппаратуры (Hardware Description) — это, согласно словарю (15p), однозначный метод описания межэлементных соединений и работы электрической и электронной частей аппаратных средств вычислительной техники.

Написание HDL-кода вместо использования схемотехнических компонентов (Schematic Components), например, логических вентилей, является в настоящее время магистральным путем в области проектирования цифровых систем.

20—25 лет назад проект моделировался (верифицировался) путем создания физического прототипа (Physical Prototype), обычно смонтированного на плате с использованием проволочного монтажа (Wire Wrapped On A Circuit Board). Так начинал и автор, начав работать инженером в 1976 году.

Генератор сигналов (Signal Generator) использовался для генераций входных тестовых сигналов, а осциллоскоп (Oscilloscope) — для отображения выходных сигналов прототипа (рис. 1.1).

Прототип здесь подобен «черному ящику» (Black Box) с входными и выходными сигналами (I/Q Signals), которые отображаются на экране осциллоскопа.

Со временем, с помощью логического анализатора (Logic Analyzer) стало возможным запоминать «длинные» временные диаграммы, которые нельзя просмотреть на экране осциллоскопа без «прокрутки» изображения.

Затем развитие методов проектирования пошло по направлению создания более сложных прототипов: возникла проблема верификации внутренних узлов\внутренних соединений (Internal Nodes\Interconnections) в цифровых схемах. Решение этой проблемы пришло со следующим поколением компьютерных инструментов проектирования (Design Computer Tools).

С появлением ЯОА стало возможным реализовать в виде программного обеспечения:

- интегрированную модель прототипа (Integrated Prototype Model);
- генератор тестовых сигналов;
- осциллоскоп и логический анализатор.

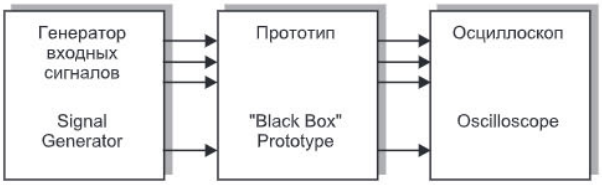


Рис. 1.1. Верификация с прототипом в виде «черного ящика» (Verification with «block box» prototype)

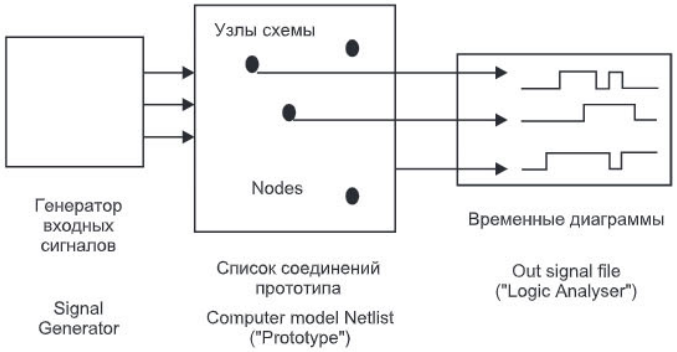


Рис. 1.2. Верификация с помощью компьютерной модели (schematic)

При этом программно реализованные генератор, осциллоскоп и анализатор составляют систему моделирования или симулятор (Simulation System, Simulator) (рис. 1.2). Прототип представляется в компьютере своей моделью на одном из ЯОА.

Вся информация о каждом узле проекта для всего периода моделирования сохраняется в памяти компьютера в виде специальной базы данных (Time Base) в интервале от Start time до Finish time.

Вначале, информация о сигналах содержала только логические уровни 0 или 1. Лишь позднее были добавлены другие состояния сигналов, например X (Undefined — неопределенное состояние) и Z (High Impedance — состояние с высоким сопротивлением). Кроме того, стало возможным проводить моделирование при различных температурах и производственных условиях.

Работа с ЯОА означает не просто написание кода. Это означает также:

- иерархичность подхода (Building Hierarchies);
- возможность проектирования с помощью библиотеки компонентов (Component Library).

В настоящей книге рассматривается проектирование цифровых устройств с помощью одного, самого популярного, языка описания аппаратуры — VHDL.

Аббревиатура VHDL означает VHSIC HDL, где, в свою очередь, VHSIC — это: Very High Speed Integrated Circuit — высокоскоростные интегральные схемы (так

называлась программа поддержки Министерства обороны С ША (US Departament of Defense — DoD) исследований в области высокоэффективной интегральной электроники).

1.2. Почему стоит использовать VHDL?

Существуют пара десятков ЯОА высокого и низкого уровня (см. 1.9). Так, например, к ЯОА высокого уровня, помимо VHDL, относится язык Verilog HDL.

Очень многие специалисты убеждены, что VHDL более предпочтителен, так как он уже более 15 лет является стандартом Института инженеров по электронике и электричеству (IEEE — Institute of Electronic and ElectricaL Engineers). Именно поэтому этот язык лидирует в промышленном и академических мирах, деля рынок поровну с Verilog HDL лишь в области логического синтеза проектов на ПЛИС.

VHDL поначалу был создан для написания спецификаций проектов (Project Specification) посредством создания моделей будущих систем — VHDL Modeling. Затем он был применен для верификации этих моделей путем моделирования — VHDL Simulation. Таким образом, с самого начала VHDL является языком документирования и моделирования, позволяющим точно задавать и имитировать поведение цифровых систем:

- параллельность функционирования;
- временные ограничения;
- синхронизация тактовыми сигналами;
- логические действия систем и временные характеристики.

Вместе с этим, VHDL очень подходит для эффективного проектирования. Проектирование на языке VHDL (VHDL Design) имеет преимущество по сравнению с традиционным схемотехническим проектированием (Traditional Schematic Design).

VHDL поддерживает среду для цифрового проектирования (Digital Development Environment) (во второй части книги будут рассмотрены для примера шесть таких сред), а также различные методы проектирования (Design Methods, Development Methods):

- сверху вниз (Top Down Design);
- снизу вверх (Bottom Up Design);
- смешанное проектирование (Mixed Design).

Для многих современных электронных изделий время жизни (Life Time) составляет порядка десяти лет. В течение этого периода они также модифицируются (приобретают новые функции и переводятся на новые технологии).

Поэтому разумно использовать проектирование на VHDL, которое не зависит от конкретной технологии (Technology Independent). В этом случае можно перейти на новую технологию, используя автоматические инструменты (Automatic Tools).

Министерство обороны США инициировало разработку VHDL в начале 80-х годов, так как нуждалось в стандартном методе (Standartized Method) описания электронных цифровых систем.

VHDL был стандартизован в 1987 году институтом IEEE. Соответствующее справочное руководство называлось IEEE VHDL Language Reference Manual Draft Standard yersion 1076/B и получило статус стандарта IEEE в декабре 1987 года — IEEE 1076 — 1987.

VHDL уже стандартизован и для аналоговой электроники — получено расши-рение VHDL для описания аналоговых цепей (Analog Extension) — VHDL-AMS.

VHDL имеет большое сходство с языком Ада. Это объясняется тем, что компания Intermetrics, которой Пентагон поручил специфицировать новый язык, имела большой опыт работы именно с языком Ада.

Особенности VHDL с самого начала и по настоящее время таковы [75a]:

- проектируемое устройство иерархически разбивается на составные части (компоненты);
- каждый компонент имеет четко очерченный интерфейс (для его соединения с другими компонентами) и точное функциональное описание для моделирования его поведения (подробно об этом в гл. 2);
- функциональное описание может быть основано либо на структуре, либо на алгоритме, которыми определяется функционирование данного компонента (см. гл. 2).

Описание функционирования посредством алгоритма (Behavioral Description) на ранних этапах проектирования делает возможным верификацию компонента более высокого уровня иерархии. На более поздних этапах проектирования алгоритмическое описание можно заменить структурной схемой (Structural Description) (см. гл. 2).

Наряду с этими заимствованными чертами, VHDL получил также только ему присущие черты: параллельность (Concurrency) и время (Time), которые являются специфическими для электронных систем.

Следующий стандарт, VHDL-93, не содержал значительных изменений по сравнению с VHDL-87: в него было включено несколько дополнительных команд, атрибутов и ключевых слов — преимущественно для целей построения моделей (VHDL Modeling).

VHDL поддерживает способность к модификации (Modifiability), удобен для создания иерархичных структур и легок для чтения, так как при создании VHDL были взаимствованы принципы структурного программирования из языков Ада и Паскаль: задать интерфейс схемного компонента, а его внутреннюю структуру скрыть [4a, 5a, 6a, 7a, 8a, 47a, 57a, 58a, 65a].

Еще раз позволим себе повторить: большой успех VHDL обусловлен тем фактом, что он был давно стандартизован, и эта работа по стандартизации не прекращается и по сей день.

1.3. Иерархия проекта на VHDL

Иерархические структуры (Hierarchies, Block Diagrams) описываются с помощью так называемого структурного VHDL (Structural VHDL), а также с помощью подпрограмм (Subprograms): процедур (Procedures) и функций (Functions). Структурный VHDL представляет собой средство описания структурных, иерархических моделей и технику для работы с блочными диаграммами.

Многие системы проектирования (CAD — Computer Aided Design, САПР) поддерживают графический ввод проекта (Graphical Design Entry), который транслируется автоматически в операторы структурного VHDL.

VHDL также поддерживает параллельные и последовательные конструкции (Concurrent and Sequential Constructions (Statements)), а также большое количество других разнообразных вещей (от описания требований спецификаций до вентильного описания (Gate Description)).

Иерархия, как известно, является средством сокращения трудоемкости (сложности) проектирования. Сложные проекты нуждаются в механизме повышения их «прозрачности» для разработчика (Reducing Complexity), так как трудно разобраться в проекте, содержащем сотни и тысячи компонентов.

Использование иерархии (Design Hierarchy) не означает, что проект становится менее сложным (иногда, наоборот, в целом он становится еще более сложным), однако он становится более легким для понимания разработчиками.

Существуют несколько механизмов для облегчения понимания проектов:

- языковые абстракции (Language Abstractions), которые используются для описания сложных вещей, избегая описания мелких деталей;
- иерархия проектирования, которая использует компоненты для скрытия мелких деталей по методу «черного ящика» (Black Box Principle), который означает, что только входы/выходы компонента видны на определенном иерархическом уровне;
- функции и процедуры, как важная часть языка VHDL, служат для сдерживания роста сложности проектирования.

Разработчик решает, как много внутренних уровней иерархии будет в проекте. Создание иерархической структуры осуществляется с помощью компонент, содержащих другие компоненты.

Иерархическая структура является ниспадающим деревом. На ветвях этого дерева находятся листья компонентов (Leaf Components), написанных с использованием последовательных и параллельных операторов VHDL-кода (Sequential and Concurrent Statements). Структурные же компоненты, состоящие из других компонентов, пишутся на структурном VHDL (Structural VHDL-code).

На рис. 1.3 приведена иерархия проектирования для некоторого проекта. Она показывает, какие компоненты содержит этот проект и как эти компоненты структурированы.

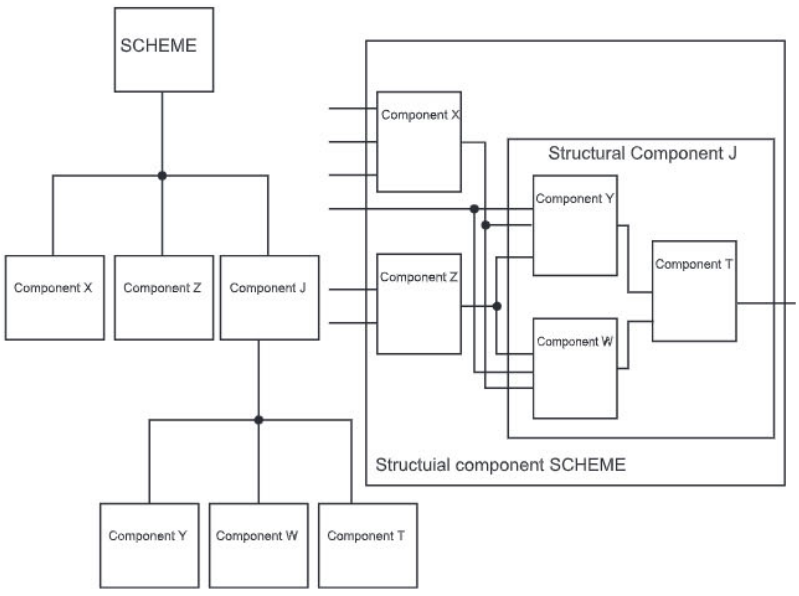


Рис. 1.3. Иерархия проекта (Design Hierarchy)

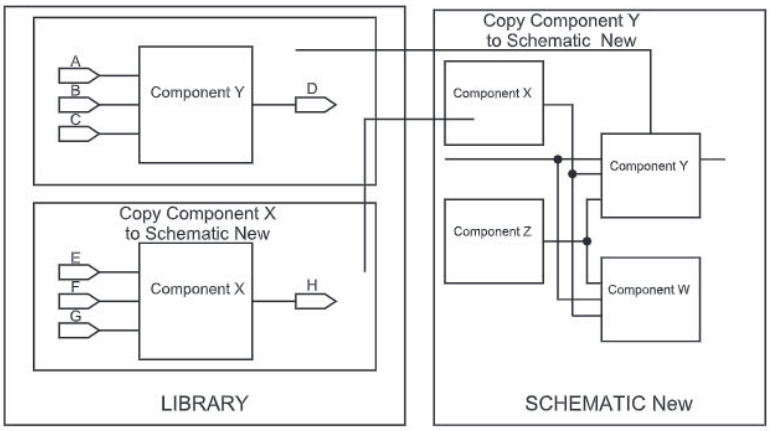


Рис. 1.4. Копирование компонентов из библиотеки в создаваемую новую схему (Schematic New)

Структурное описание компонента содержит интерфейс и сведения о том, какие компоненты существуют на следующем, более низком уровне иерархии проекта.

Работа по проектированию обычно стартует с определения интерфейса (Entity) для корня проекта (Root, Top Design). Затем компонент делится на несколько новых подкомпонентов (Subcomponents) с межсоединениями (Interconnections). Новая структура называется архитектурой (Architecture).

Не существует комерчески реализуемого инструмента, который поддерживает процесс разбиения компонентов на подкомпоненты (Partioning). Общее правило, однако, таково: разбиение должно осуществляться так, чтобы интерфейс между компонентами становился как можно проще. С другой стороны, подкомпоненты не должны становиться чрезмерно малыми, так как объем написанного VHDL-кода для данной иерархии при этом непрерывно увеличивается (5а, 6а, 47а, 58а, 65а).

1.4. Описание электронных компонентов

Использование компонента является центральной концепцией языка VHDL. Компоненты используются, в числе прочего, также и для построения библиотек (Component Libraries), содержащих модели микропроцессоров (Microprocessors), схем специального применения (Special User Circuits) и т. д.

Если верифицированные компоненты (Good Components) разработаны и содержатся в библиотеке, становится возможным копировать их столько раз, сколько нужно. Таким образом осуществляется их многократное использование (Reusable Components) (рис. 1.4). Такое копирование называется созданием экземпляров (копий) компонента.

VHDL является объектно-базирующимся языком (Object-Based Language). Разница между ним и объектно-ориентированными языками (Obje ct-Oriented Languages) заключается в том, что VHDL не имеет механизма наследования (Inheritance).

Для VHDL характерно использование компонент с настраиваемыми параметрами (Generic Components) и реализация множественных экземпляров компонент (Component Instantiation).

Компоненты с настраиваемыми параметрами являются компонентами, которые модифицируются перед реализацией экземпляров. Например, такой компонент может копироваться с различными размерностями входных и выходных сигналов.

Внутренняя структура компонента может быть скрыта от проектировщика по принципу «черного ящика». В определенных случаях абсолютно не нужно знать, как структурирован компонент. Разработчика обычно интересуют лишь входы и выходы компонента, спецификация его функций и время доступа (Access Time). Разработчики используют в качестве «черных ящиков» ПЛИС (FPGAs, PLDs) или серийные микросхемы, например, 74 LSXX.

Библиотеки могут содержать и более сложные компоненты (как, например, контроллеры, процессоры и коммуникационные схемы), которые могут состоять, в свою очередь, из других компонентов.

Существуют фирмы, продающие стандартные VHDL-компоненты, являющиеся моделями стандартных микросхем, для реализации процесса проектирования. Это означает, что модели печатных плат (Printed Circuit Boards — PCBs) со стандартными микросхемами могут быть верифицированы с использованием компьютерного моделирования (Computer Simulation).

В случае традиционного схематического проектирования разработчик должен постоянно осуществлять проверку вручную выполнения специфических технологических требований (Technology-Specific Factors), таких как:

- временные ограничения (Timing);
- занимаемая площадь (Area);
- выбор компонентов (Component Choise);
- потребляемая мощность (Driving Strength);
- разветвление по выходу (Fan-Out).

Проектирование же VHDL-компонентов является технологически независимым (Technology Independent) или относительно технологически независимым (More-Or-Less Technology Independent). Технологически зависимая же часть проекта выполняется на более позднем этапе в автоматизированном режиме (см. п. 1.6).

Таким образом, одним из самых больших достоинств проектирования на VHDL является то, что разработчик может сосредоточиться на функциях проекта, т. е. на реализации требований спецификации (Requirement Design Specification), и избавлен от необходимости уделять внимание факторам, которые не влияют на функции проекта.

VHDL-компонент содержит две основные части (рис. 1.5):

- описание интерфейса («сущность») — Entity (Port Declaration), — которое описывает взаимосвязи между компонентом и средой его «обитания» (функционирования);
- архитектура (архитектурное тело) — Architecture (Architectural Body), — описывающая поведение компонента с функциональной или структурной точки зрения (Behavioral or Structural Description) относительно входов и выходов.

Имя Entity (Entity Name) является и именем самого VHDL-компонента [5а, 6а, 47а, 58а, 65а].

1.5. Абстракции языка VHDL (VHDL language abstractions)

VHDL богат языковыми конструкциями и может использоваться для описания различных уровней абстракций (Different Abstraction Levels) — от функционального к полностью вентильному описанию (Gate Description). Уровни абстракций являются средством сокрытия избыточных деталей (Conclaling Details). Уровни абстракции языка VHDL не надо смешивать с уровнями абстракций проектирования (построения моделей — см. гл. 2). Оба типа абстракций позволяют разработчику бороться со сложностью процесса проектирования.

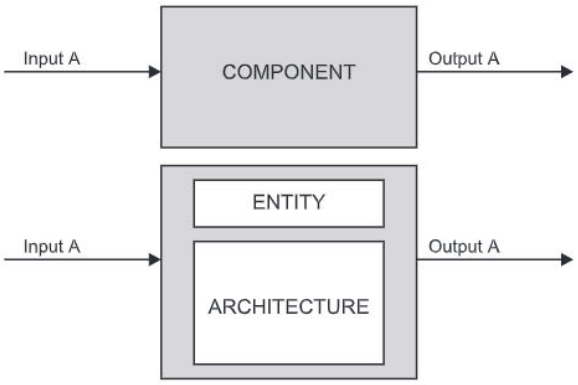


Рис. 1.5. Части компонента (VHDL Component): Entity и Architecture

Если разработчик хочет сложить два числа (A = B + C), он может использовать один из трех следующих методов описания:

- использование оператора «+» языка VHDL, т. е. a<=b+c (это означает, что сигнал A принимает значение, которое вычисляется как сумма двух других сигналов B и C);
- пректирование сумматора (Adder) на вентильном уровне;
- проектирование сумматора на уровне размещения элементов в толще полупроводниковой структуры.

Этот пример показывает, как определенная функция может быть реализована на трех различных уровнях абстракции.

На рис. 1.6 приведены различные уровни абстракции для языка VHDL:

- функциональный (Functional Level);
- поведенческий (Behavioral Level);
- регистровых передач (Register Transfer Level-RTL);
- вентильный (логический) (Gate Level, Logic Level).

Функциональный уровень используется для описания поведенческого (алгоритмического) уровня функциональных моделей, а также для описания уровня процессор — память — коммутатор структурных моделей. На функциональном уровне не используется временная информация.

Поведение (Behavior) во времени описывается на поведенческом уровне. Он используется для описания тех же уровней функциональных и структурных моделей, которые упоминались выше.

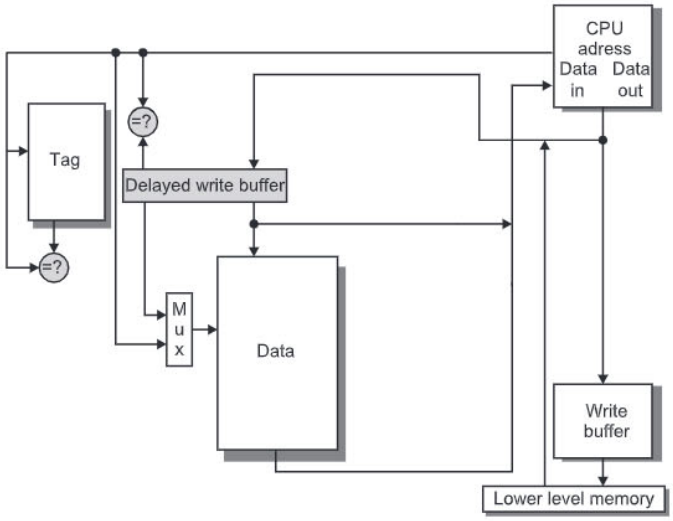


Рис. 1.6,а. Различные уровни абстракций VHDL. Функциональный уровень (Functional Level)

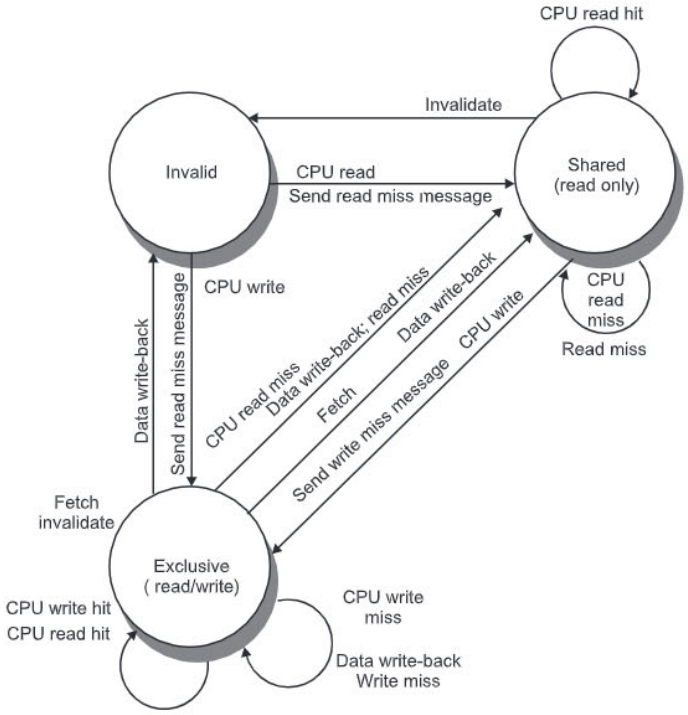


Рис. 1.6,б. Различные уровни абстракций VHDL. Поведенческий уровень (Behavioral Level)

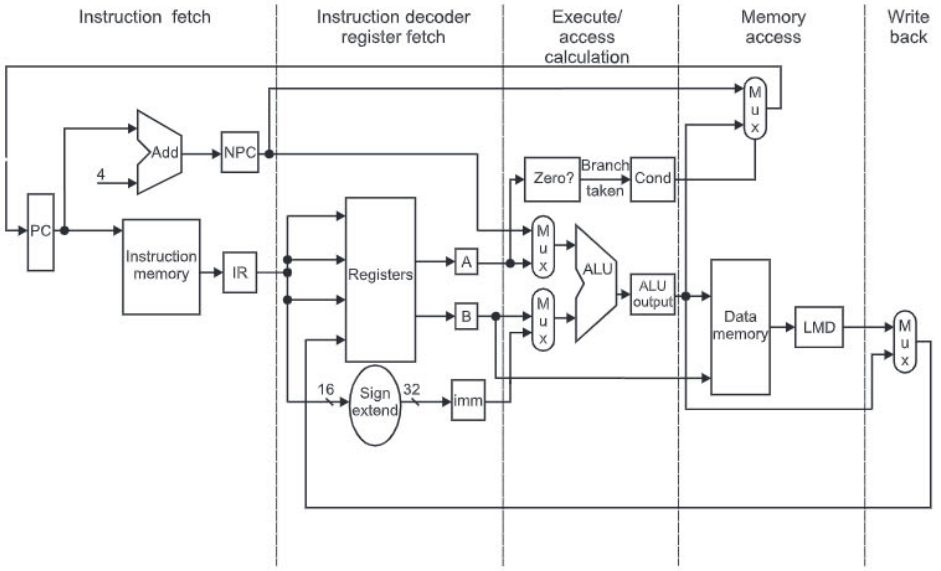


Рис. 1.6,в. Различные уровни абстракций VHDL. Уровень регистровых передач (RTL Level)

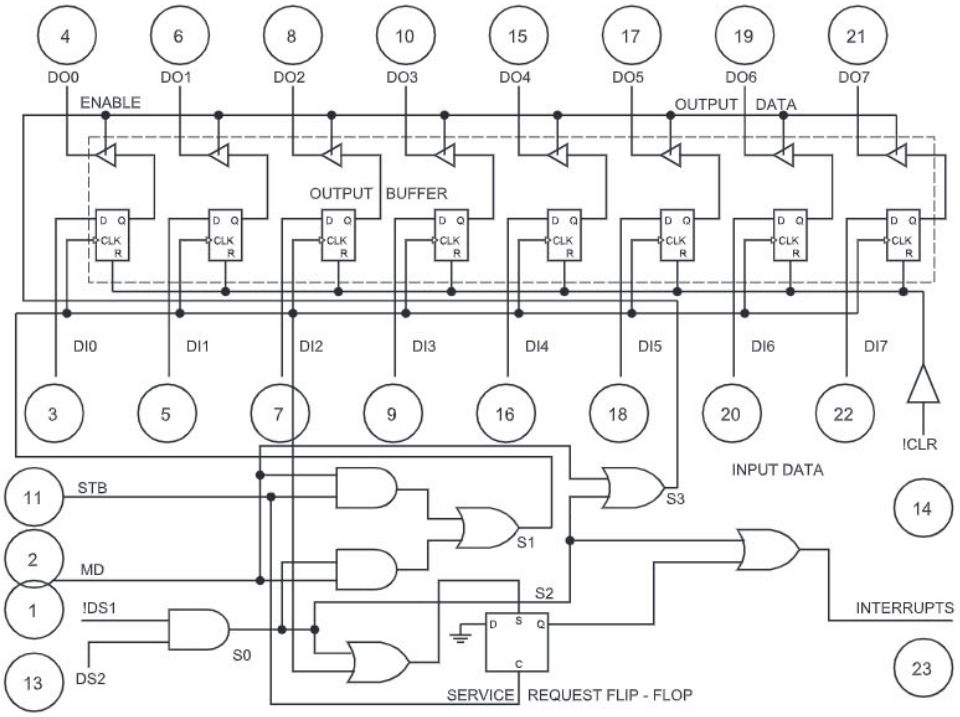


Рис. 1.6,г. Различные уровни абстракций VHDL. Вентильный уровень (Gate Level)

На этом уровне, как правило, не определена структура системы. Поведенческая модель (Behavioral Model) представляет собой совокупность функциональных моделей (Functional Modules) с интерфейсом между ними. К этим модулям приписываются определенные функции и временные соотношения (Time Relations). Преимущество моделей этого уровня состоит в том, что они могут быть быстро построены для выполнения процессов моделирования и верификации.

В определенных случаях на этом уровне может быть определена и структура системы, если речь идет о структурных моделях уровня процессор — память — коммутатор (PMS: Processor — Memory — Switch) (см. гл. 2).

Проблемы появляются в случае, если функции системы разделяют определенные ресурсы. В этом случае необходимо перейти с уровня PMS на RTL-уровень или скомпоновать функции, использующие те же самые ресурсы, вместе.

Входная информация этого уровня используется для выполнения процесса синтеза с помощью инструментов (инструментальных систем), называемых Behavioral Synthesis Tools.

На этом же уровне рассматриваются модели уровня интегральных схем [5a, 2p]. Модель уровня интегральной схемы (ИС) является поведенческой моделью, в которой задержки цепей точно отображаются без использования описания более низкого уровня:

1. Модели уровня ИС точно моделируют временные параметры входных/выходных сигналов.

2. Модели уровня ИС реализуются как последовательность микроопераций, закодированных на ЯОА. Каждая ИС является базовым примитивом (т. е. представляет собой лист дерева проекта).

3. Рассматриваемый метод моделирования на уровне БИС/СБИС (VLSI) оперирует моделями, содержащими десятки и сотни тысяч логических вентилей.

4. Модели уровня ИС строятся без описаний вентильного или регистрового уровня. В качестве входной информации для построения таких моделей используются:

- текстовые описания,
- структурные схемы,
- спецификации и временные диаграммы,
- таблицы и диаграммы состояний.

Отсюда вытекает важное практическое следствие: на ранних этапах проектирования разработчик может проверить характеристики компонентов системы во взаимосвязи, несмотря на то что их низкоуровневая реализация неизвестна (модель вентильного уровня обычно является собственностью изготовителя ИС). Однако известна целевая технология (Target Technology) — семейство микросхем для реализации проекта), позволяющая произвести оценку временных задержек этих моделей.

5. Модель уровня ИС представляет собой множество спецификаций БИС/СБИС.

Входные спецификации описывают:

- время установления сигналов,
- время удержания сигналов,
- минимальные длительности импульсов для входных сигналов ИС.

Выходные спецификации описывают моменты времени, в которых происходит переключение выходных сигналов.

И, наконец, внутренние спецификации описывают требования к внутренней памяти ИС, задержки для внутренних сигнальных цепей и микрооперации ИС.

Уровень регистровых передач (RTL) содержит язык, который описывает поведение или структуру системы с помощью:

- регистров (Registers),
- тракта передачи данных (Data Path),
- асинхронных и синхронных цифровых автоматов, как моделей устройства управления (Asynchronous and Synchronous State Machines for Control Unit Modeling).

Вентильный (или логический) уровень представляет собой описание с помощью булевой алгебры, или вентильной схемы. Он предназначен для описания функциональных моделей в виде сети логических вентилей.

Транзисторный уровень (Transistor Level) (рис. 1.7, *а*), содержащий модели транзисторов, емкостей и сопротивлений, а также уровень геометрического проектирования в толще кремния (Layont Level — рис. 1.7, *б*), который содержит модели описания физических процессов, не поддерживаются конструкциями VHDL.

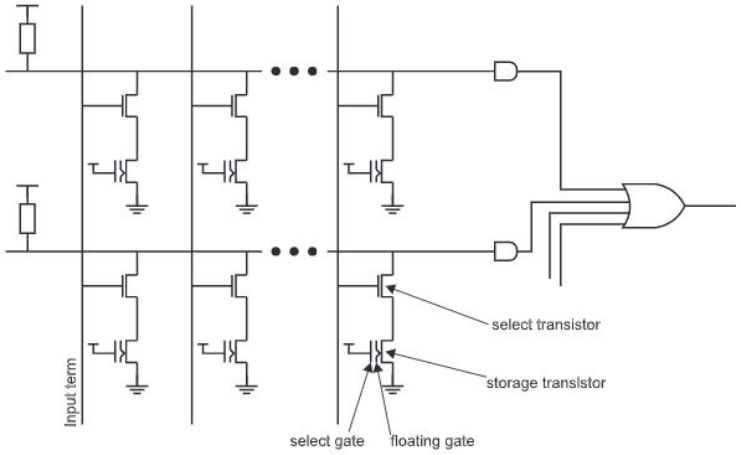


Рис. 1.7,а. Транзисторный уровень (Transistor Level)

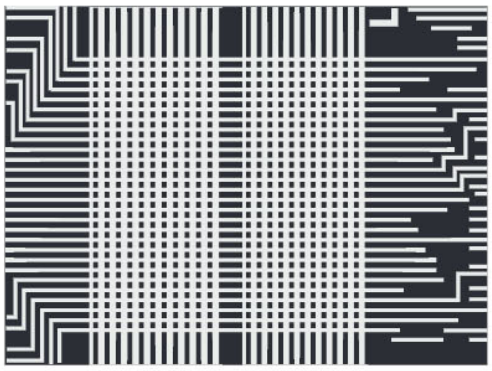


Рис. 1.7,б. Уровень геометрического проектирования в толще кремния (Layout Level)

Использование различных уровней абстракции подобно использованию различных языков программирования для процессоров:

- микрокод (Microcode),
- машинный код (Machine Code),
- Ассемблер (Assembler),
- C, C++, VB, Ada.

Если надо написать программу, отвечающую требованию очень короткого времени исполнения, то используют Ассемблер. Если же надо написать очень сложную программу, то ее пишут на C, C++ или Ada.

Подобным же образом, если требуется короткое время разработки VHDL-модели (Short Development Time),выбирается высокий уровень абстракции (Hign Abstraction Level). На практике из описаний на уровне RTL и, частично, на уровне Behavioral Level могут быть автоматически синтезированы описания на вентильном уровне. Таким образом, VHDL может быть использован для построения моделей от самого высокого уровня абстракций до самого низкого уровня, включая вентильный. Поэтому язык VHDL очень хорош для разработки ПЛИС: ASICs (Application Specific Integrated Circuits) и FPGAs (Field Programmable Gate Arrays), от функционального до вентильного уровня, а также для построения разнообразных моделей на стадии верификации проекта (Design Verification).

В случае проектирования блоков (печатных плат с микросхемами) — PCB Design — на печатной плате обычно находятся несколько ASICs (или FPGAs) вместе с микропроцессором (Microprocessor) и его инфраструктурой. В этом случае моделировать проект на вентильном уровне обычно невозможно, так как при этом время моделирования оказывается очень большим. Поэтому компоненты PCBs должны быть описаны на уровне не ниже регистровых передач (RTL) для достижения приемлемого времени моделирования [5a, 6a, 47a, 58a, 65a].

1.6. Процесс проектирования с использованием VHDL

Фаза разработки (проектирования) — Development Phase — является для изделия первой фазой его жизненного цикла (Life Cycle) (рис. 1.8, *а*). В течение этой фазы изделие специфицируется, проектируется и верифицируется. Общая модель разработки называется «водопад» (Waterfall Model). Она стартует со спецификации и ведет через определенные этапы к созданию функционирующего прототипа (Functioning Prototype). Через все эти этапы надо пройти при проектировании на основе любого ЯОА. Поток информации в процессе проектирования приведен на рис. 1.8, *б*, а подробная структура отдельных этапов — на рис. 1.8, *в*.

Этап анализа (Analysis) состоит в написании спецификации, которая составляется на VHDL или на обычном языке. Цель разработки спецификации состоит в ответе на следующий вопрос: что должно быть сделано? Если спецификация написана на VHDL, она верифицируется с помощью симулятора (VHDL Simulator).

Этап собственно проектирования (Design) означает трансформацию спецификации в архитектуру проекта и ее описание на VHDL (VHDL Code). Здесь возникает осознание основного подхода и функций отдельных блоков на уровне иерархической блок-схемы. VHDL служит основой для определения компонент (блоков): их интерфейсов и деталей внутреннего устройства, а также для их детализации в дальнейшем.

Этапы проектирования (Development Phase)	Результат этапа (Result)	Документация (Documentation)
Анализ (Analysis)	Спецификация (Specification)	Описывает требования к проекту
Проектирование (Design)	Модель на языке VHDL (VHDL code)	Описывает собственно процесс проектирования
Выбор технологии (Technology mapping)	Список соединений принципиальной схемы (Netlist)	
Макетирование (Prototyping)	Технологически-зависимая модель проекта (Prototype)	Описывает функционирование прототипа

Рис. 1.8,а. Поток проектирования (Flow Design)

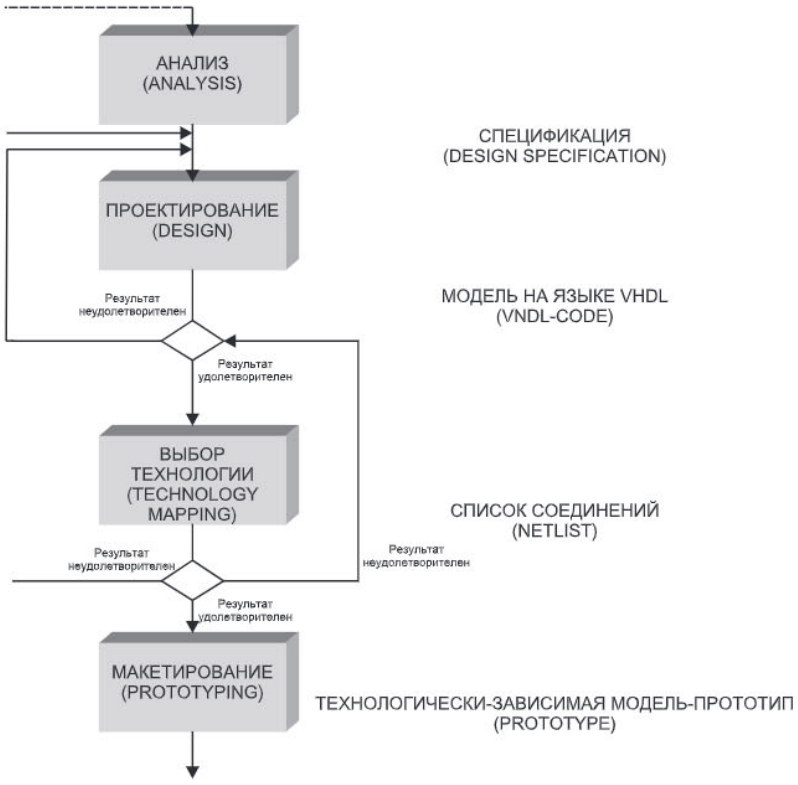


Рис. 1.8,б. Схема потока процесса проектирования с использованием VHDL

Этот этап начинается с определения архитектуры проекта в виде блочных диаграмм (Block Diagrams). Когда архитектура проекта готова, различные компоненты-блоки описываются с помощью VHDL или копируются из библиотеки готовых компонент.

В большинстве случаев среда, в которой выполняется проектирование, включает специализированный текстовый редактор HDL (VHDL) Text Editor, который облегчает работу: он обычно содержит автоматическое выделение ключевых слов VHDL, автоматический отступ от начала строки, встроенные шаблоны часто

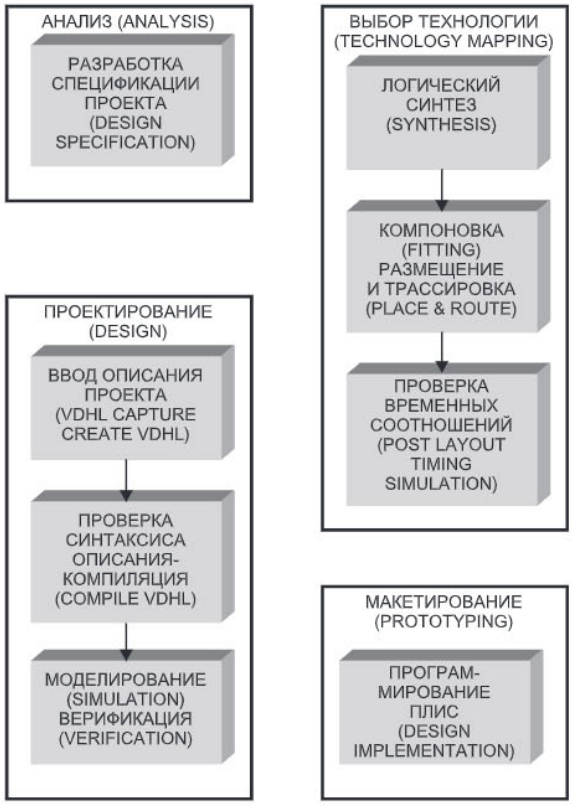


Рис. 1.8,в. Подробная структура этапов проектирования с использованием VHDL

встречающихся программных структур, встроенную проверку синтеза и упрощенный доступ к компилятору.

Компилятор (VHDL Compiler) анализирует текст (VHDL-Code) на отсутствие в нем синтаксических ошибок и проверяет совместимость программы с другими модулями, на которые имеются ссылки. Кроме того, компилятор подготавливает информацию для моделирующей программы-симулятора.

Симулятор позволяет задавать входные сигналы и подавать их на входы разрабатываемой VHDL-модели, наблюдая ее выходные реакции.

Моделирование является одной из ступеней более крупного этапа, называемого верификацией (Verification). Для большого проекта одинаково важны как стадия написания, так и стадия тестирования в широком диапазоне условий для проверки правильности функционирования проекта. При функциональной верификации (Functional Verification) логика работы проекта изучается независимо от временных соотношений (задержки вентилей и другие временные параметры считаются равными нулю).

Функции проекта верифицируются с помощью системы моделирования, и если результаты согласуются со спецификацией, разработчик переходит к следующему этапу. Главным вопросом этапа проектирования (Design) является следующий вопрос: как должны быть спроектированы архитектура проекта и ее отдельные компоненты?

Следующим этапом является выбор технологии (Technology Mapping). Характер действий здесь сильно зависит от технологии изготовления ПЛИС для реализации проекта. Выбор технологии представляется следующими параметрами:

- цена (Cost),
- производительность (Performance),
- поставщик (Supply) и т. д.

Этот этап в настоящее время в основном автоматизирован.

Процесс логического синтеза (Synthesis) является преобразованием описания проекта на VHDL (Project VHDL Description) в набор компонент, которые можно будет образовать в среде выбранной ПЛИС.

В случае ПЛИС типа CPLD (Complex Programmable Logic Device) инструмент синтеза (Synthesis Tool) реализует проект двухуровневыми схемами «суммы произведений».

В случае специализированных ПЛИС (FPGAs, ASICs) результатом работы инструмента синтеза является список вентилей и их соединений между собой. При этом разработчик задает ограничения (Constraints), характерные для данной технологии (например, максимальное число логических уровней, нагрузочная способность логических буферов), тем самым помогая работе инструмента синтеза.

Как уже отмечалось, в процессе синтеза генерируется список соединений, который зависит от выбранной технологии (Technology Dependent Netlist). Этот список соединений является входной информацией (входным файлом) для других инструментов, например, для инструментов ATPG (Automatic Test Pattern Generation).

Временные ограничения (Time Constraints) описываются в формате, который читается с помощью используемых инструментов синтеза. Если система синтеза не способна соблюсти эти ограничения, вновь повторяется полностью или частично данный этап проектирования.

Всего лишь несколько минут достаточно на программирование FPGA. Работа с FPGAs имеет определенное преимущество, так как FPGAs уже протестированы изготовителями. Если же, например, мы имеем дело с массивом вентилей, то понадобится несколько недель на окончание работы над проектом, так как должны быть разработаны тестовые векторы (Test Vectors, Test Patterns), которые в данном случае являются составной частью проекта.

Процесс компоновки (Fitting), выполняющийся соответствующей программой (Fitter), отображает синтезированные компоненты на имеющиеся в ПЛИС ресурсы.

Для CPLD (Complex PLD, Complex Programmable Logic Device) выражения типа «суммы произведений» приписываются определенным физическим элементам И-ИЛИ.

Для специализированных ПЛИС (ASICs, FPGAs) производится раскладка вентилей в нужной конфигурации и соединения их соответствующими трассами с учетом физических ограничений в кристалле ПЛИС. Этот процесс называется размещением и трассировкой (Place & Route). Здесь у разработчика проекта есть возможность задать дополнительные ограничения на размещение компонентов в кристалле ПЛИС и назначение выводов ПЛИС для внешних входов и выходов проекта.

Далее выполняется этап проверки временных соотношений (Timing Verification): работа проекта исследуется с учетом предполагаемых задержек. Главный вопрос в случае использования последовательностной логики на базе триггеров состоит в следующем: удовлетворяются ли требования по времени установления сигналов и их удержания?

Только на этом этапе можно с определенной точностью найти реальные задержки в проекте, обусловленные длиной соединений, величиной нагрузки и другими факторами. Используются те же самые условия тестирования, что и при функциональной верификации, однако здесь в эти условия подставляются величины предполагаемых задержек.

Если результат синтеза, компоновки или размещения и трассировки не может быть реализован с помощью выбранной ПЛИС или не удовлетворяет временным требованиям, приходится вернуться назад, а иногда даже пересмотреть сам подход к проекту в целом (штриховая линия на рис. 1.8, *а*).

На последнем этапе проектирования разработчик получает физический прототип (Prototype), который он сравнивает со спецификацией проекта. Если функционирование прототипа удовлетворяет требованиям спецификации, это значит, что проект выполнен. Такое сравнение называется подтверждением (Validation) [47a, 58a, 65a, 75a]. Далее подробно остановимся на вопросах верификации и синтеза.

1.7. Верификация проекта

Выполнение процесса моделирования является эффективным средством верификации проекта.

Компьютерная модель (Computer Model) обладает определенной изысканностью (Refinements) по сравнению с физическим прототипом (Physical Prototype). Моделирование наихудшего случая (Worst Case) для параметров проекта и температурного диапазона обеспечивает лучшую верификацию по сравнению с построением прототипа (в приемлемое время на тестирование!).

С помощью симулятора и временного анализатора возможно моделировать различные временные случаи (Timing Cases):

- Worst Case:
  - низкое напряжение (4.5 v);
  - высокая температура (+125 °C);
  - замедленные характеристики процессов;
- Typical Case:
  - нормальное напряжение (5v);
  - нормальная температура (+25 °C);
  - нормальные характеристики процессов;
- Best Case:
  - высокое напряжение (5.5v);
  - низкая температура (−55 °C);
  - быстрые характеристики процессов.

Моделирование на поведенческом уровне является функциональной верификацией проекта (Verifying Functionality, Functional Verification). Для этой же цели используются и RTL-модели.

Временные ограничения для следующего уровня моделирования (Timing Verification) определяются в процессе RTL-синтеза (RTL Synthesis), однако только физическая модель обеспечивает хороший анализ поведения в реальном времени (Actual Time Behavior). При этом возможно также вычислить потребление мощности (Power Consumption).

Выполнение процесса моделирования требует все больше времени, так как только таким образом можно повысить точность моделей (Accuracy of Models). Это

означает, что большие проекты не могут быть промоделированы на уровне геометрического проектирования (Layout Level).

Время выполнения моделирования (Simulation Time) также возрастает, когда выполняются проверки:

- времени установки (Setup Tests),
- времени удержания (Hold Tests),
- бросков питания (Spike Tests).

Кроме того, время выполнения моделирования зависит от длины входных воздействий. Поэтому функциональная верификация выполняется на самом высоком уровне описания, какой только возможен (Behavioral Level или RTL Level).

Верификация временных параметров (Timing Verification), включая проверку времени установки и удержания, выполняется эффективно и быстро с помощью временного анализатора (Timing Analyser) на вентильном уровне.

Так как размер проектов постоянно растет, то и время выполнения моделирования для них также драматически растет. Решением этой проблемы является применение аппаратных ускорителей (Hardware Accelerators). Эти специально разработанные блоки могут выполнять процесс моделирования на несколько порядков быстрее, чем обычные компьютерные средства. Вместе с тем аппаратные ускорители также на несколько порядков дороже обыкновенных компьютеров.

Инструменты для формальной верификации также используются для верификации электронных цифровых систем. Формальная верификация может быть использована для обнаружения тупиковых ситуаций (Dead Locks), а также для проверки, являются ли два различных описания идентичными.

VHDL-код для цифровых компонентов и их систем верифицируется с помощью симулятора, который исполняет этот код на соответствующих наборах входных сигналов и генерирует сигналы моделирования в виде временных диаграмм (Signal Diagrams) и сообщений об ошибках (Error Messages).

Входные сигналы задаются с помощью VHDL или на внутреннем языке симулятора (Simulator's Language). Наборы выходных сигналов являются частью испытательной программы (Test Bench). Структура ее приведена на рис. 1.9.

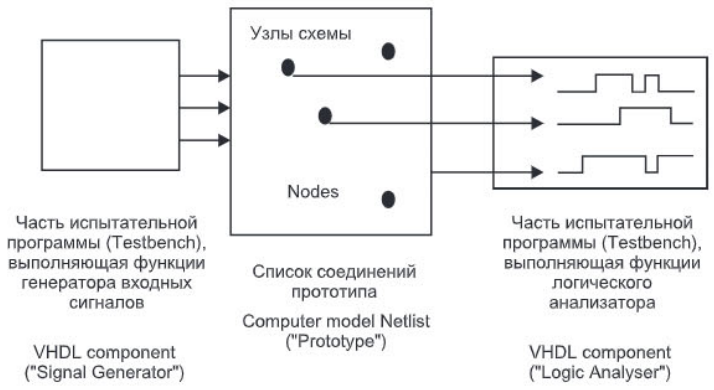


Рис. 1.9. Верификация с помощью испытательной программы Test Bench на компьютере

Сам проект (Prototype) не обязательно должен быть представлен здесь своим VHDL-кодом, альтернативой этому может быть представление в виде принципиальной схемы (Schematics, Net). Испытательная же программа пишется на VHDL,

что позволяет переносить этот VHDL-код между различными системами проектирования (см. гл. 4—9 ч. 2).

Верификация с помощью VHDL также обеспечивает возможность построения менеджера ошибок (Error Manager) в каждом компоненте (рис. 1.10). Это означает, что каждый компонент модели имеет секцию кода, который проверяет сигналы интерфейса (Interface Signals) во времени и использование действительных адресов (Valid Addresses). На этом пути могут быть выявлены неверные адреса и определено время доступа (Access Times).

Компоненты RAM, например, могут проверять время доступа и убедиться в том, что сигналы READ и WRITE не могут быть активными одновременно. Если они не удовлетворяют этому требованию, симулятор выдает сообщение об ошибках. В общем, компоненты обязаны «разговаривать» и сообщать симулятору, что они находятся в некорректном (нерабочем) состоянии на этапе верификации.

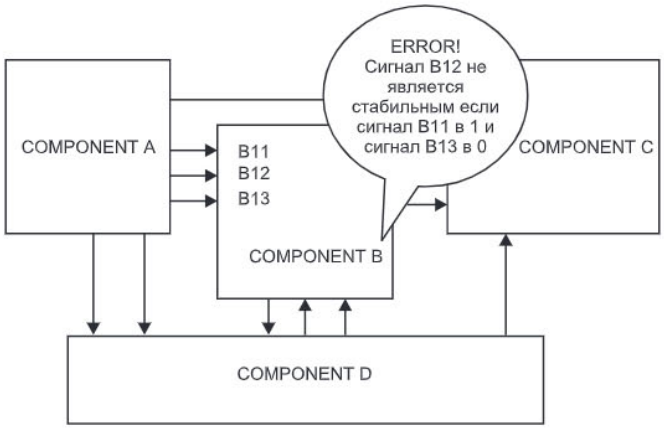


Рис. 1.10. Обнаружение ошибки компонентом (Fault-handling in a VHDL-component)

Уровень абстракции (Abstraction Level)	Описание (Description)	Единица времени моделирования (Time Granularity)
Поведенческий уровень (Behavioral Level)	Поведение объекта проектирования согласно спецификации (Behavioral Description)	Микросекунда (Microsecond)
Проектирование (Register Transfer Level)	Язык межрегистровых передач (VHDL code)	Цикл тактовых импульсов (Clock Cycles)
Выбор технологии (Logic (Gate) Level)	Булевы уравнения (Boolen equations)	Наносекунда (Nanosecond)

Рис. 1.11. Выбор единицы времени моделирования на различных уровнях абстракции

В настоящее время существуют симуляторы смешанного уровня (Mix Level Simulators), которые могут поддерживать модели на различных уровнях абстракций и ЯОА.

Например, вначале весь проект моделируется на самом высоком поведенческом уровне (Highest Behavioral Level) для верификации спецификации пректа.

Затем определенные части проекта моделируются на RTL-уровне. И наконец, еще меньшая часть проекта моделируется на вентильном уровне с целью выверить задержки вентилей (Gate Delays), соединений между вентилями (Interconnect Delays) и подачи входных сигналов (Load Delays).

Некоторые симуляторы могут выполнять процесс моделирования, используя VHDL-код, на трех следующих уровнях абстракций:

- Behavioral Level,
- RTL Level,
- Gate (Logic) Level

на базе результатов размещения и трассировки с задержками межсоединений (Basis of Layout) стандартных компонент. Применение таких симуляторов сокращает общее время моделирования проекта.

Выбор единицы времени моделирования на различных уровнях абстракций иллюстрируется рис. 1.11.

Общие замечания по САПР, включающих различные симуляторы, приведены в п. 1.10. Подробное описание работы с некоторыми симуляторами приведено в части 2 (гл. 4—9):

- глава 4: симулятор системы ModelSim (фирма Model Technology, подразделение Mentor Graphics) [44a—46a, 58a];
- глава 5: симулятор системы ActiveHDL (фирма Aldec) [1a, 2a];
- глава 6: симулятор системы OrCAD (фирма Cadence Design Systems) [48a, 49a, 11p, 12p];
- глава 7: симулятор системы Warp (фирма Cypress Semiconductor) [76a, 77a, 66a];
- глава 8: симулятор системы Foundation Series (фирма Xilinx) [78a—80a, 14p];
- глава 9: симулятор системы Symphony (фирма Symphony EDA) [74a].

1.8. Процесс синтеза с использованием VHDL (VHDL Synthesis)

Язык VHDL и его возможности моделирования были сами по себе нововведениями, однако скачок популярности и признания полезности VHDL произошел с появлением коммерческих инструментов синтеза (VHDL Synthesis Tools). Теперь с помощью VHDL стало возможным разрабатывать и синтезировать проекты в широком диапазоне (от простой комбинационной схемы до микропроцессорной системы на одном кристалле [3a, 13a, 14a, 50a, 58a, 60a, 65a, 66a, 68a, 69a].

Использование программного обеспечения для синтеза (Synthesis Software) означает, что проектировщик устраняется от непосредственного участия в процессах трансляции и минимизации VHDL-кода, а также от проверки соответствия временным ограничениям (Time Constraints).

Существуют несколько различных видов синтеза:

- Logic Synthesis: трансляция (и минимизация) булевых функций в вентильную схему;
- RTL Synthesis: примерно то же самое, что и логический синтез, но в этом случае используются «последовательные» конструкции языка (Sequential Language Constructions) , которые транслируется в схему, содержащую не только вентили, но и триггеры и представляющую собой цифровой автомат;

- Behavioral Synthesis: может использовать один и тот же схемный компонент (Hardware Component) для более чем одной последовательной конструкции языка.

Логический синтез хорошо описан в литературе, поэтому начнем с синтеза на RTL-уровне.

Пусть имеется следующий фрагмент VHDL-кода, который подлежит синтезу:

```
process (select, input1, input2)
begin
      if select = ' 1 ' then
         output < = input2 ;
      else
         output < = input1 ;
      end if
end process ;
```

Этот VHDL-код является независимым от технологии (Technology-Independent). Он легок для прочтения. Перед нами описание двухвходового мультиплексора: если сигнал управления выбором (select) равен 1, тогда выходному сигналу output присваивается величина входного сигнала input2, в противном случае сигналу output присваивается величина входного сигнала input1.

Данный VHDL-код является входной информацией (Input Data) для инструмента синтеза (рис. 1.12). Временные ограничения содержатся в другом файле, который также является входной информацией для данного инструмента синтеза.

Технологическая библиотека (Technology Library) описывает технологию для реализации этого VHDL-кода. Например, пусть используется простая команда для трансляции VHDL-файла в принципиальную схему с вентилями и триггерами инструмента синтеза View Logic`s Synthesis Tool (фирмы View Logic):

```
vhdldes vhdl_fil -tec = XC400.
```

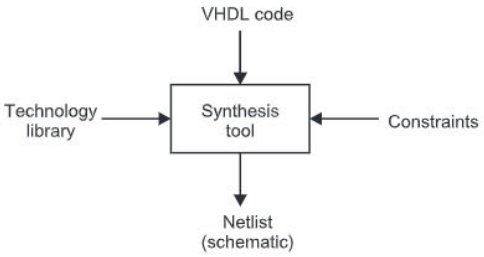


Рис. 1.12. Использование инструмента синтеза (Synthesis Tool)

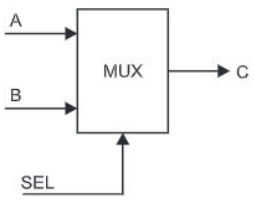


Рис. 1.13. Результат синтеза — двухвходовой мультиплексор

При этом XC4000 является указателем, который передает инструменту синтеза следующую информацию (сам код XC4000 означает Xilinx 4000 Family; другие семейства приборов содержатся в других библиотеках):

- потребляемую мощность для схемы,
- скорость функционирования,
- размер поверхности микросхемы (в мм).

Результатом синтеза этого VHDL-кода является двухвходовый мультиплексор (рис. 1.13). Он является технологически зависимым: если будет выбрана другая

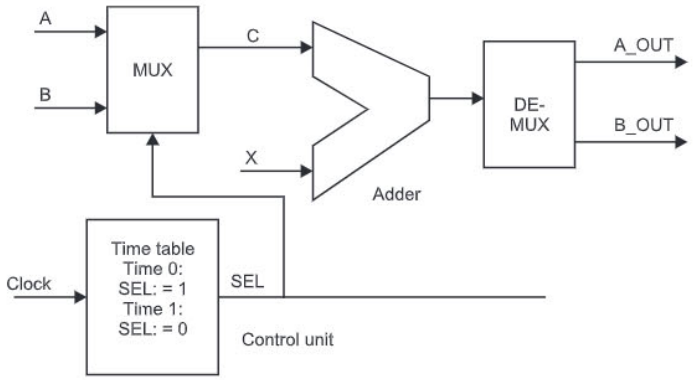


Рис. 1.14. Результат синтеза на поведенческом уровне (Behavioral Synthesis)

технология, результат может содержать вместо ячейки мультиплексора несколько вентилей, выполняющих ту же функцию.

Процесс синтеза может быть сравнен с компиляцией, которая компилирует входной код в машинный код: VHDL-код транслируется в принципиальную схему (Schematics) с вентилями и триггерами в выбранной технологии.

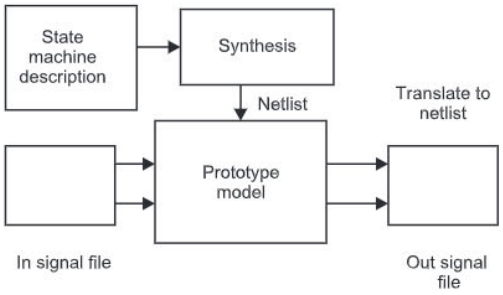


Рис. 1.15. Процесс логического синтеза (Logic Synthesis)

В языке VHDL существует множество различных языковых конструкций, которые успешно синтезируются на этом RTL-уровне, например:

- Generic Components (Использование параметров в архитектурном теле; при этом сами параметры определяются в описании интерфейса entity);
- Instantiation Component Statements (создание экземпляров базовых компонент);
- Structural VHDL Statements.

Когда имеем дело с большими проектами, существуют избыточные функции (компоненты) в целом ряде мест схемы: сумматоры, мультиплексоры и т. п. В аппаратуре тяжело разделить функции между различными «пользователями», однако существуют инструменты, которые решают эту задачу автоматическим образом.

Процесс разделения ресурсов (функций) вместо использования избыточных аппаратных единиц (Redundant Hardware Units (Functions)) называется Behavioral Synthesis (Синтез на поведенческом уравне). Такой синтез выполняется с временными таблицами (Time Tables), которые определяют, когда различные пользователи могут использовать общий ресурс (Shared Component).

На рис. 1.14 показано, как один сумматор может находиться в совместном пользовании двух пользователей A и B. Устройство управления (Control Unit) осуществляет подключение лишь одного пользователя к сумматору в каждый момент времени.

Возможно, таков будущий путь построения «аппаратных» операционных систем (Hardware Operating Systems) для поддержки функционирования аппаратуры

системы (Hardware Units). Тем самым будет стерта разница между аппаратурой и программным обеспечением системы.

Процесс синтеза выполняется между соседними уровнями процесса проектирования изделий, например между RTL-уровнем и вентильным уровнем. При этом VHDL-код транслируется в вентили и триггеры, образующи-ми структуру минимизированного проекта. При таком синтезе нужна дополнительная информация об используемой технологии. Объем такой информации возрастает с каждым дополнительным переходом между уровнями абстракций. Каждый переход (Transition Synthesis) генерирует дополнительную информацию, вследствие этого, сложность проектирования (Design Complexity) возрастает.

Проектирование принципиальной схемы по данной системе булевых функций занимает много времени. До эры САПР (CAD/CAM) проектировщик строил вентильную схему и проводил вентильную минимизацию вручную.

Эта проблема была решена разработкой инструментов автоматической трансляции текстового файла, описывающего булевые уравнения, в вентильную схему (после минимизации). Такие инструменты стали первыми инструментами логического синтеза (Logic Synthesis) (рис. 1.15).

Первые коммерческие инструменты логического синтеза (Logic Synthesis Tools) предназначались для программирования простых программируемых схем (Programmable Circuits), содержащих только вентили. Затем были разработаны программируемые схемы, содержащие триггеры (PLDs — Programmable Logic Devices), и новые инструменты для их программирования.

Тут же обнаружились проблемы с совершенно новыми семействами микросхем, появляющимися и исчезающими с большой скоростью. Если микросхема исчезала с рынка, проектировщик вынужден был переработать описание всего используемого кода, так как каждый инструмент мог работать лишь с одним семейством микросхем. Поэтому стали разрабатывать новое поколение инструментов синтеза, которые могли работать с несколькими семействами микросхем.

Язык VHDL давно стандартизован, и это сделало возможным переносить VHDL-код между различными средами разработки и моделирования. До сих пор, однако, нельзя перемещать код, предназначенный для синтеза, между произвольными системами. VHDL не имеет стандарта для синтеза. Если разработчик решает использовать для синтеза некоторое подмножество VHDL, код может стать перемещаемым между некоторыми произвольными системами, однако это никем не гарантируется.

Мощные инструменты синтеза автоматически генерируют сегодня проекты, которые позволяют эффективно тестировать себя (Testable Designs). Например, инструмент синтеза автоматическим образом, без участия разработчика, добавляет к проекту сдвиговый регистр (Shift Register) для сканирования состояний внутренних узлов схемы проекта (Scan Path Techniques). Такие регистры доступны для загрузки и чтения с первичных выводов микросхем (Primary Inputs, Primary Outputs). Подобные инструменты синтеза генерируют также файлы текстовых наборов (Test Files), которые позднее загружаются в память аппаратуры контроля — тестера (Automate Test Equipment — ATE) на заводе изготовителя этой микросхемы. При этом, например, для описания JTAG-интерфейса БИС/СБИС используются ЯОА, являющиеся подмножествами VHDL [38a, 63a, 11a]: BSDL(Boundary Scan Description Language) и HSDL (Hierarchy Scan Description Language). Необходимо отметить, однако, что до сих пор нет стандарта для файлов тестовых наборов, предназначенных для использования в тестерах.

1.9. Другие языки описания аппаратуры

Формализованные текстовые описания с начала 70-х годов прошлого века применяются при проектировании цифровых устройств. Эти первые ЯОА предусматривали определение набора микроопераций и описание функционирования в форме микропрограмм (являясь языками уровня регистровых передач — RTL Level) [15р, 22р].

Приведем несколько примеров таких первых ЯОА:

- ЛЯПАС (логический язык для представления алгоритмов синтеза релейных устройств) (1966) [8р];
- языки АЛГОРИТМ и СТРУКТУРА (1970) [6р];
- CDL (Computer Description Language) (1975) [19р];
- AHPL (A Hardware Programming Language) (1975) [15р];
- ADLIB (A Desing Language for Indicating Behaviour) (1979) [21а];
- Ф-язык (1979) [9р];
- ISPS (Instruction Set Processor Specification) (1979) [47а];
- SLIDE (Structural Language for Interface Description and Evaluation) (1981) [55а];
- OODE (Object Oriented Description Environment for computer hardware) (1983) [47а];
- CONLAN (CONsensus LANguage) (1983) [59а];
- BORIS (Block — Oriented Interacting Simulation system) (1984) [47а];
- HILL (HIgh hierarchical Layout Language) (1984) [47а].

Как правило, используют простую классификацию для современных ЯОА, которые отличаются от своих предшественников (упомянутых выше) тем, что, кроме описательных функций, ориентированы на машинное моделирование и синтез (компиляцию верифицированного описания в аппаратуру):

– ЯОА высокого уровня (High-Level Design Languages), которые являются технологически независимыми (VHDL, Verilog HDL)
– ЯОА низкого уровня (Low-Level Design Languages) (ABEL, AHDL, CUPL, PALASM и др.), зависящие от конкретной технологии (ПЛИС конкретной фирмы изготовителя) и предназначенные для описания дискретных устройств и их синтеза.

1.9.1. Verilog HDL

Язык Verilog HDL (или просто Verilog) был предложен в 1984 году фирмой Gateway Design в качестве собственного ЯОА и средства моделирования. Автор [18р, 75а] полагает, что имя языка образуется следующим образом: VERIfy LOGic. Фирма Gateway Design в 1989 году была куплена гигантом Cadence Design Systems. Примерно в это же время фирмой Synopsys были созданы первые программные инструменты синтеза на основе языка Verilog. Эти факторы привели к широкому распространению Verilog: в настоящее время VHDL и Verilog делят примерно поровну рынок логического синтеза [39а, 54а, 68а, 70а, 7р]. Синтаксис Verilog имеет своим началом язык C, в отличие от VHDL, который, как уже упоминалось, берет свое начало в языке Ada.

В качестве примера использования Verilog приведем описание 4-входового мультиплексора (Verilog Description Multiplexer) на этом языке [54а]. (Наш муль-

типлексор имеет 4 входа: i0, i1, i2, i3; один выход: out и два управляющих сигнала: s0, s1.)

```
// Module 4-to-1 Multiplexer
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0)
//Port Declarations
output out
input  i0, i1, i2. i3
input  s1, s0
//Internal Wire Declarations
wire s1n, s0n
wire y0, y1, y2, y3
//Gate Instantiations
//Greate s1n and s0n Signals
not (s1n, s1);
not (s0n, s0)
//3- input AND Gates Instantiated
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0 );
and (y2, i2, s1 , s0n);
and (y3, i3, s1 , s0 );
//4 - input OR Gate Instantiated
or (out, y0, y1, y2, y3);
endmodule
```

Более подробно с Verilog можно познакомиться в книгах [39a, 54a, 70a, 75a, 7p, 13p, 14p, 18p].

1.9.2. ABEL

Язык ABEL (Advanced Boolean Equation Language) был создан фирмой Data I/O Corporation, чтобы позволить разработчику задавать логические функции, подлежащие реализации в ПЛИС [75a, 56a, 69a, 18p].

ABEL допускает задание функций в виде:

- таблиц истинности (Truth Tables),
- вложенных операторов IF,
- любых алгебраических соотношений.

Программа на языке ABEL представляет собой текстовый файл, содержащий следующие элементы:

- документирования, в том числе имя программы и комментарии;
- объявления, задающие входные и выходные логические функции, подлежащие реализации;
- тестовые векторы, которые задают ожидаемые значения логических функций для некоторых комбинаций элементов.

Типичная структура программы на языке ABEL выглядит следующим образом:

```
module module_name
title  string
device ID device device_Type;
     pin declarations
     other declarations
```

```
equations
     equations
test vectors
      test vectors
end module _ name
```

Компилятор с языка ABEL (ABEL Compiler, ABEL Language Processor) обрабатывает эти форматы и минимизирует результирующие соотношения так, чтобы сделать их соответствующими, если возможно, структуре имеющихся ПЛИС. Он транслирует текстовый файл описания в схему соединения для определенной «физической» ПЛИС.

В качестве примера использования ABEL приведем описание 4-входового мультиплексора (4:1 MUX — эквивалент микросхемы LS153 TTL-логики) на этом языке.

```
module MUX4
title '4:1 Mux`
My Device device `P16L8';
@ALTERNATE
"INPUTS
A,B, /P1G1,/P1,G2          pin 17,18,1,6        "LS153 pins 14,2,1,15
P1CO, P1C1,P1C2,P1C3       pin  2,3,4,5         "LS153 pins 6,5,4,3
P2C0, P2C1,P2C2, P2C3      pin 7,8,9,11         "LS153 pins 10,11,12,13
"outputs
P1Y, P2Y                    pin19,12            "LS153 pins7,9
equations
P1Y=P1G*(/B*/A*P1C0+/B*A*P1C1+B*/A*P1C2+B*A*P1C3)
P2Y=P2G*(/B*/A*P2C0+/B*A*P2C1+B*/A*P2C2+B*A*P2C3)
end MUX4
```

(в этом коде используются следующие обозначения: / — логическое отрицание; " — комментарий; * — логическое умножение; + — логическое сложение).

1.9.3. AHDL

Язык Altera HDL (AHDL) был создан фирмой Altera в 1983 году и представляет собой эффективное средство для:

- поведенческого и структурного описания проектируемого устройства;
- документирования проекта.

В отличие от ЯОА высокого уровня (VHDL, Verilog), он более прост и предназначен для проектирования определенных ПЛИС фирмы Altera. Однако он содержит набор инструкций весьма высокого уровня.

В целях экономии места автор не намерен приводить примеры использования AHDL, так как он уже хорошо описан в российской технической литературе [1р, 13р, 14р, 7р].

1.9.4. CUPL

Язык CUPL (Universal Compiler for Programmable Logic) был создан фирмой Logical Devices, Inc (www. logical-devices.com.). Он является языком проектирования ПЛИС (PLD Design Language) фирмы Atmel [13а, 69а].

На рис. 1.16 приведен пример работы с пятой версией этого языка (CUPL 5,0) (компилятор WinCupl Ver. 5.2.16, среда моделирования WinSim Ver. 5.1.30).

CUPL-код описывает цифровые автоматы (машины состояний). Для этой цели используются различные операторы присваивания (Different State Machine Assignment Statements) или специальные таблицы истинности.

Преимуществами данного языка являются:

- отсутствие необходимости вводить детальное описание декодирования состояний — это делается автоматически;
- выполнение изменений в структуре модели осуществляется простым текстовым редактированием.

Приведем пример CUPL-кода, описывающего последовательную схему (Sequential Logic) — цифровой автомат с двумя состояниями:

```
sequence Bay Bridge Toll Plaza {
        present red
            if car next grun out go ; / * conditional synchronous output* /
            default next red; / * default next state * /
        present green / * unconditional next state * /
next red;
```

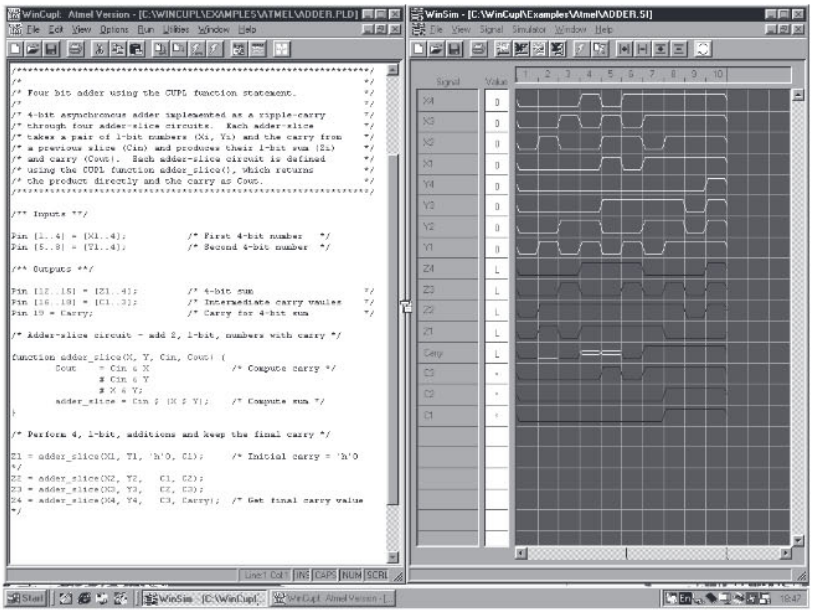


Рис. 1.16. Работа с компилятором WinCupl (слева) и средой моделирования WinSim (справа)

А вот пример описания состояний с помощью таблицы истинности на языке CUPL:

```
field input [in 1…0]
field output [out 3…0]
table input => output {00=> 01; 01=>02; 10=>04; 11=>08}
```

И, наконец, пример CUPL-кода для 16-разрядного сдвигающего регистра с последовательным выходом (Ser_out) и параллельным входом (DO…D15) — 16-bit Loadable Shift Register, предназначенного для реализации в ПЛИС F1500a:

```
/*Inputs*/
pin 43 = CLK;
pin 1 = Load;
pin 2 = RES;
pin [41…36,34…31,29…24]=[DO…15]
/*outputs*/
pin 21 = ser_out;
/*Q Nodes*/
pinnode = [Q1…15];
Field Qnodes = [Q15…1,ser_out];      /*Q nodes*/
Field Qshift   = ['b'0,Q15…1];       /* Shifting input*/
Field Din      = [D15…0];            /*Data input*/
/*equations*/
Q nodes.ck = CLK;
Q nodes.ar = RES;
Q nodes.d = Load & Din               /* Load in the data*/
            #! Load & Q Shift;       /*Shifts the data*/
```

1.9.5. PALASM

Язык PALASM (PAL (Programmable Array Logic) Assembler) является языком проектирования ПЛИС (PLD Desing Language), разработанным фирмой AMD/MMI [69a].

Следующий пример описания сдвигающего регистра ShiftReg показывает основные особенности второй версии языка PALASM:

```
TITLE
CHIP
CK/LD D0 D1 D2 D3 D4 D5 D6 D7 CURS GND NC REV
        Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 /RST VCC
STRING Load          'LD* /REV*/CURS* RST';
STRING Load Inv      'LD* REV * /CURS * RST';
STRING Shift         '/LD*/CURS * / RST';
Eqvations
/Q0: = / D0 * load + D0 * load inv: +: / Q1 * Shift + RST;
/Q1: = / D1 * load + D1 * load inv  +: / Q2 * Shift + RST;
/Q2: = / D2 * load + D2 * load inv: +: / Q3 * Shift + RST;
/Q3: = / D3 * load + D3 * load inv: +: / Q4 * Shift + RST;
/Q4: = / D4 * load + D4 * load inv: +: / Q5 * Shift + RST;
/Q5: = / D5 * load + D5 * load inv: +: / Q6 * Shift + RST;
/Q6: = / D6 * load + D6 * load inv: +: / Q7 * Shift + RST;
/Q7: = / D7 * load + D7 * load inv: +:          Shift + RST;
```

(здесь:+: означает логическую операцию XOR.)

Авторы [7p] предлагают более подробную классификацию ЯОА. Критерием отнесения языка к определенному уровню является степень абстракции используемых в языке конструкций. Рассматриваются четыре следующих уровня языка [7p]:

- языки реализации (таблицы соединений);
- машинно-ориентированные языки (простые ассемблеры PALASM, PLDASM [17р]; микроассемблеры: ABEL, AHDL);
- процедурно-ориентированные языки (VHDL, Verilog);
- объектно-ориентированные языки (Hardware-C).

1.10. Системы автоматизированного проектирования на основе языков описания аппаратуры

Как отмечается в работе [7р], в настоящее время происходит кооперация усилий различных фирм при разработке САПР. Каждая фирма, участвующая в разработке новой САПР, реализует тот этап проектирования (фрагмент САПР), в котором она является признанным лидером. Примером кооперационной САПР для продукции фирмы Xilinx является система Foundation Series (работа в этой системе рассмотрена в гл. 8) [7р, 14р]. Данная САПР разработана фирмами Xilinx, Aldec и Synopsys.

Foundation Series позволяет использовать ПЛИС в качестве элементной. базы для построения «систем на кристалле» (System — On — Chip, SOC). Для сложных систем на кристалле, содержащем миллионы логических вентилей, данная САПР обеспечивает коллективную работу над проектом, как в локальной сети, так и в Интернете (технология Internet Team Design — ITD) [14р].

Основой системы служит оболочка Foundation Project Manager (разработка фирмы Aldec). Описание проекта на ЯОА выполняется с помощью специального редактора HDL Editor и средства поддержки Language Assistant. Для ввода описания проекта (цифрового автомата) в графической форме используется графический редактор State Editor (о работе с ним см. гл. 8) (все эти компоненты также разработаны фирмой Aldec).

В данной системе можно работать с ЯОА ABEL, VHDL u Verilog; для целей синтеза используются синтезируемые подмножества VHDL и Verilog. Кроме того, AHDL-файлы могут быть преобразованы в VHDL- или Verilog-файлы.

В качестве инструмента синтеза в системе используются FPGA Express Synthesis фирмы Synopsys.

Функциональное моделирование и верификация (Functional Simulation, Verification) на этапе проектирования (см. п. 1.6) с возможностью задания входных воздействий в интерактивном графическом режиме, а также проверка временных соотношений (Timing Simulation) (временные ограничения для проекта задаются с помощью Express Constraints Editor) на этапе выбора технологии (см. п. 1.6) выполняются с помощью симулятора Logic Simulator.

На стадии макетирования (п. 1.6) программирование ПЛИС (Design Implementation) запускается из оболочки Foundation Project Manager. За этим процессом можно наблюдать в окне Flow Engine.

После окончания проектирования на PC система позволяет провести аппаратную верификацию проекта с использованием средства Xchecker и отладочной платы [14р].

Ко всему вышесказанному следует добавить, что система содержит еще два важных компонента [14р, 7р]. Средство Logi BLOX служит для создания описания модулей (счетчиков, регистров, памяти, мультиплексоров). Оно загружается из текстового редактора HDL Editor по команде Tools/Logi BLOX. Генератор логических ядер (CORE Generator) генерирует ядра (Logi CORE), представляющие собой фун-

кциональные параметризированные блоки системного уровня для применения в области DSP (Digital Signal Processing). Этот генератор также вызывается из HDL Editor по команде Tools/CORE Generator.

Foundation Series реализована в форме единой САПР. Возможна также организация «кооперативной» САПР в форме нескольких взаимосвязанных САПР с единым информационным потоком проектирования. Создание программ-оболочек, осуществляющих подобную интеграцию, упрощается с использованием системы программирования Tcl/Tk (Tool Command Language/Tool Kit), служащей для разработки сценариев на языке Tcl в среде графического интерфейса пользователя (GUI) Tk [7p].

На примере Foundation Series видно изменение отношения к программному обеспечению САПР ПЛИС [14p, 7p].До конца прошлого века основным средством описания проекта являлся ввод схемы с помощью графических редакторов или описания на ЯОА с использованием схематических или текстовых (на ЯОА) моделей стандартных ИС, СИС, сгруппированных в библиотеки. В настоящее время актуальным является использование ЯОА для программирования ПЛИС. В современных САПР поддерживаютя как стандартизованные ЯОА (VHDL, Verilog), так и ЯОА компаний-производителей ПЛИС (AHDL, ABEL и др.).

Еще одно свидетельство этому — программное обеспечение фирмы Mentor Graphics, содержащее инструмент синтеза Leonardo Spectrum (разработан фирмой Exemplar Logic, подразделением Mentor Graphics), который поддерживает ПЛИС основных производителей и позволяет осуществить проектирование заказных микросхем (ASIC). Для проектирования FPGAs в составе данного ПО находится также инструмент синтеза Precision Synthesis и пакет FPGA Advantage, включающий как составную часть Leonardo Spectrum, и позволяющий проектировать также и ASICs.

Второй формой кооперации при создании САПР [7p] является использование отдельных популярных компонентов (в различных модификациях) в САПР других фирм.

Так, симулятор Model Sim (разработчик — фирма Model Technology, подразделение Mentor Graphics) помимо использования в ПО Mentor Graphics, используется также в САПРах фирм Xilinx, Altera (работа с данным симулятором рассматривается в гл. 4). Синтезатор Leonardo Spectrum помимо использования в САПР Mentor Graphics используется, например, также в САПР Altera (вместе с Modelsim, MAX+Plus II, Quartus II).

Инструменты синтеза фирм View Logic, Mentor Graphics и Synopsys составляют львиную долю всех инструментов синтеза, используемых в индустрии и образовании.

Инструмент синтеза Work View Office (фирма View Logic) является PC-ориентированной системой (Win 95/98/NT), вместе с тем, с ней можно работать так же и на рабочих станциях (Workstations). Синтезаторы фирмы Synopsys и синтезатор Autologic-2 фирмы Mentor Graphics работают только на рабочих станциях. Они более дорогие и мощные по сравнению с Work View Office.

На протяжении 90-х годов синтезаторы Synopsys лидировали на рынке VHDL Synthesis. В настоящее время усиливается конкуренция со стороны Mentor Graphics во всем, что касается синтеза.

Синтезаторы двух упомянутых фирм поддерживают то же самое подмножество языка VHDL (Subset of VHDL), содержащее до 99% конструкций языка. Поэтому VHDL -код может перемещаться между САПРами этих двух фирм [65a].

Процесс синтеза рассматривается вкратце в этой книге в гл. 7 (система Warp) и 8 (Foundation Series).

Третьей формой кооперации в создании современных САПР [7р] является включение фирмами-производителями ПЛИС в свои САПР оценочных версий пакетов сторонних фирм.

Так, фирма Atmel поставляет САПР IDS (Integrated Development System) версии 7.2, которая включает оценочные версии Leonardo Spectrum и ModelSim [7р].

Фирма Altera рекомендует для эффективного синтеза проектов использовать синтезаторы FPGA Express или Leonardo Spectrum, а для эффективного моделирования — ModelSim [14р].

Для удобства проектирования цифровых автоматов многие современные САПР используют графические редакторы, описывающие цифровые автоматы в виде диаграммы состояний (см. подробно об этом в гл. 8). Кроме Foundation Series, использующей такой редактор, отметим здесь графические редакторы фирмы Mentor Graphics (Renoir, HDL Designer Series) [7р], а также систему StateCAD (разработчик — фирма Visual Software Solutions, Inc) [14р, 7р], входящей в пакет Work ViewOffice.

StateCAD предназначена для создания описаний цифровых автоматов на ЯОА, пригодных для реализации на ПЛИС. Входной информацией при этом является диаграмма состояний автомата, вычерчиваемая проектировщиком [14р] (рис. 1.17).

StateCAD включает несколько мастеров (Wizards) (FSM, Logic, Design, Optimization), средство просмотра и редактирования файлов на VHDL, Verilog, ABEL, AHDL или ANSI-C — HDL Browser, а также генератор тестов State Bench и средство поведенческой верификации проектов (Behavioral Verification) [14р]. Подобный графический редактор (FSM Editor системы Xilinx Foudation Series) описывается в гл. 8.

Как уже упоминалось выше, в данной книге (гл. 4—9) подробно рассмотрена работа шести известных симуляторов, работающих с VHDL. Не рассматри- вается в настоящей книге система MAX+Plus II фирмы Altera, так как она достаточно хорошо описана в российской технической литературе [1р, 13р, 14р]. Здесь можно найти сведения о редакторах системы (Graphic Editor, Symbol Editor, Text Editor, Waveform Editor и Floorplan Editor), компиляторе (MAX + Plus II Compiler), средствах верификации проекта (MAX + Plus II Simulator, Timing Analyzer) и программирования ПЛИС (MAX + Plus II Programmer).

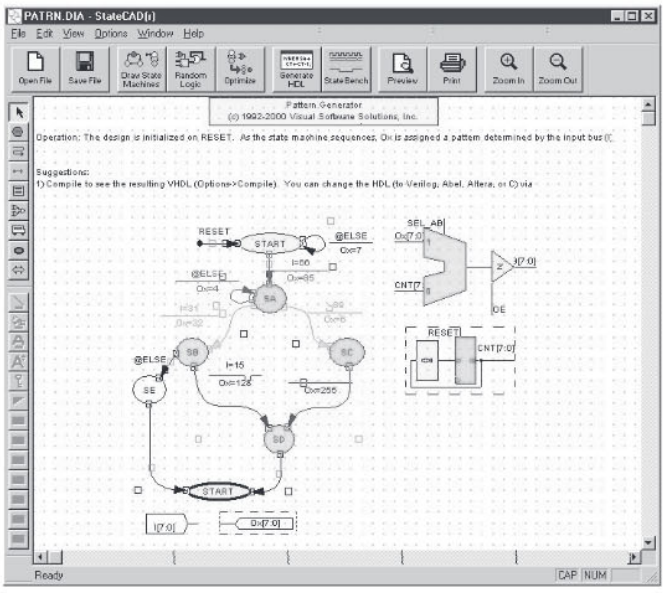


Рис. 1.17. Главное окно StateCAD

Глава 2. Фундаментальные концепции языка VHDL

2.1. Моделирование цифровых систем

Вначале, основываясь на требованиях спецификации (Specification) к системе, разрабатывают абстрактную структуру системы (Abstract Structure), которая соответствует этим требованиям. Затем можно выполнить декомпозицию этой структуры в совокупность компонентов (Components), взаимодействующих между собой для выполнения требуемых функций. Результатом этого процесса является модель системы с иерархической структурой (Hierarchically Composed System), построенная из примитивных элементов (Primitive Elements).

Преимуществом этой методологии моделирования является то, что каждая подсистема (Subsystem) может быть разработана независимо от остальных подсистем (компонент). Прежде чем разрабатывать детальную структуру подсистем, их рассматривают как некоторые абстракции. Поэтому на любом этапе процессов моделирования и проектирования можно уделять основное внимание только текущему «фокусу» проектирования (Current Focus of Design), не обременяя себя массой второстепенных в текущий момент времени деталей.

Термин «модель» означает наше понимание системы. Модель представляет, как правило,только актуальную на данном этапе информацию. Таким образом, для той же самой системы можно построить множество моделей, каждая из которых будет актуальна для определенного контекста.

Существует несколько важных причин для построения моделей систем [6a, 47a, 58a, 62a].

Первая причина состоит в том, что первым шагом на пути создания некоторой системы является написание технических требований (спецификации). Сама эта спецификация является первой моделью будущей системы.

Второй причиной использования формализованных моделей является необходимость дать пользователю будущей системы представление о ней. Проектировщик не всегда предвидит все возможные пути использования системы и не способен пронумеровать все способы ее поведения. В случае, если разработчик системы передаст ее пользователю и модель данной системы, пользователь сможет проверить ее на требуемом наборе входных сигналов и определить, каково поведение системы в данном контексте. Таким образом, формализованная модель является бесценным инструментом документирования системы.

Третья причина использования моделей заключается в желании сделать возможным тестирование (Testing) и верификацию (Verification) проекта (Design) с помощью использования процесса моделирования. Можно промоделировать поведение модели на определенном множестве входных тестовых сигналов (Test Inputs) и получить результирующие выходные реакции системы. Согласно принятой методологии проектирования, можно спроектировать вентильные схемы для подсистем, для каждой из которых есть собственная модель поведения. Затем можно промоделировать подсистемы на соответствующих множествах входных тестовых сигналов и сравнить выходные реакции (Outputs) с реакциями на предыдущем уровне моделирования. Если они оказались тождественными, это означает, что

составная система (Composite System) отвечает требованиям спецификации для проверенных случаев. В противном случае неизбежна ревизия проекта.

Можно продолжать этот процесс до тех пор, пока не достигнем нижнего уровня иерархии проекта (Design Hierarchy), где компонентами являются реальные устройства (Devices), чье поведение известно априори. Когда же проект реализован, входные тестовые сигналы и выходные реакции, полученные в процессе моделирования (Outputs from Simulation), могут быть использованы для верификации функций физических схем. Такой подход к тестированию и верификации предполагает, что входные воздействия покрывают все состояния, в которых используется произведенная схема. Формирование тестового покрытия (Test Coverage) является сложной проблемой и представляет собой до настоящего времени арену активных исследований.

Четвертая причина использования моделей заключается в желании осуществить формальную верификацию корректности проекта. Такая верификация требует математического обоснования (Mathematical Statement) требуемой функции системы. Это обоснование может быть выражено в нотации формальной логической системы, такой как временная логика (Temporal Logic). Формальная верификация требует также математического определения конструкций языка для написания моделей (Modeling Language). Процесс формальной верификации использует правила логического вывода (Inference) в рамках определенной логической системы для доказательства того, что проект обладает требуемыми функциями. Формальная верификация также является областью интенсивных исследований.

Пятой, очень важной, **причиной** использования моделей является возможность осуществления процесса автоматического синтеза вентильных схем (Automatic Synthesis). В случае, если мы можем формально описать требуемые функции схемы, есть возможность транслировать это описание (спецификацию) в вентильную схему, которая выполняет требования данной спецификации.

Преимуществом этого подхода является то, что стоимость проекта снижается, а инженеры освобождаются для изучения альтернатив проектирования вместо погружения в рутинную работу. Кроме того, автоматическая трансляция из спецификации в реализацию схемы приводит к уменьшению числа ошибок в проекте. Все это ведет к достижению максимальной надежности проектирования (Maximum Design Reliability) при минимальной стоимости (Minimum Cost) и минимальном времени проектирования (Minimum Design Time).

2.2. Области и уровни построения моделей

Различные модели систем можно классифицировать на три области (Modeling Domains) [6а] (рис. 2.1):

- область функциональных моделей (Functional Domain);
- область структурных моделей (Structural Domain);
- область геометрических моделей (Geometric Domain).

На этом рисунке области представлены тремя независимыми осями, а уровни абстракции представлены в виде концентрических кругов, пересекающих каждую из осей.

Область функциональных моделей содержит модели функций, выполняемых системами. Это область наиболее абстрактных моделей-описаний, поскольку здесь не рассматриваются конкретные реализации функций системы (рис. 2.2).

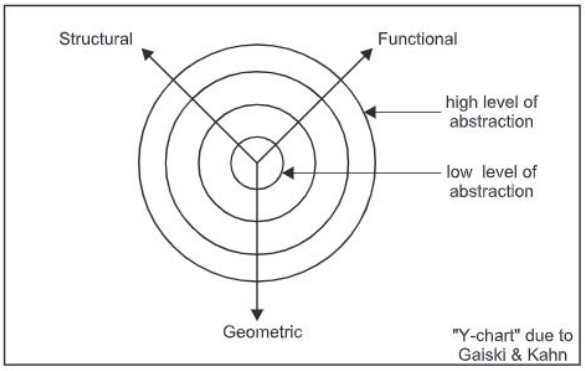


Рис. 2.1. Области и уровни абстракций (построения моделей)

В этой области используются следующие уровни абстракции:
- поведенческий (алгоритмический) (Algorithm);
- языка межрегистровых передач (Register-Transfer Language);
- описаний в виде булевых уравнений (Boolean Equation);
- описаний в виде дифференциальных уравнений (Differential Equation).

Область структурных моделей содержит информацию о структуре систем, о том, как системы скомпонованы из взаимосвязанных подсистем (рис. 2.3).

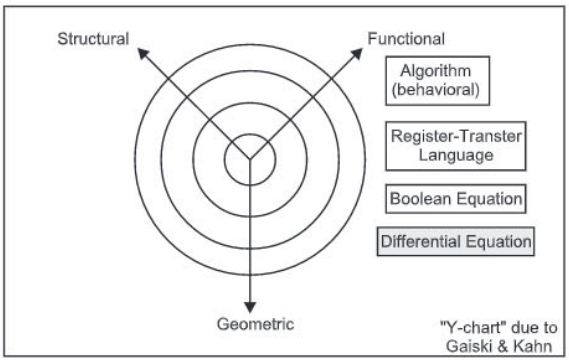


Рис. 2.2. Уровни описаний-абстракций в области функциональных моделей

Уровни абстракций в области структурных моделей:
- процессор-память-коммутатор (Processor-Memory-Switch);
- регистровых передач (Register-Transfer);
- вентильный (Gate);
- транзисторный (Transistor).

Область геометрических моделей содержит информацию о размещении элементов системы на физическом уровне (рис. 2.4).Уровни описаний-абстракций в этой области:
- многогранников (Polygons) для каждого слоя маски интегральной схемы;
- топологических схем (Sticks);
- стандартных ячеек (Standard Cells);
- физического размещения на подложке (Floor Plan).

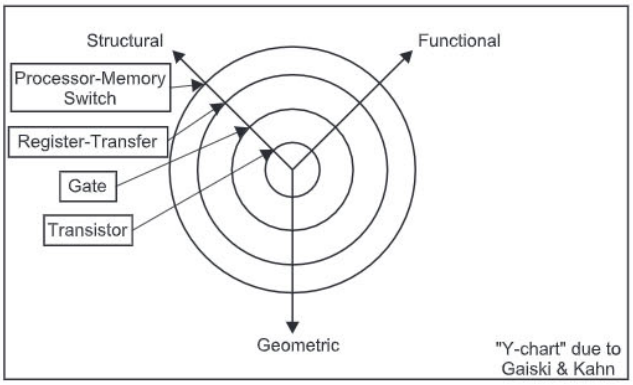


Рис. 2.3. Уровни описаний-абстракций в области структурных моделей

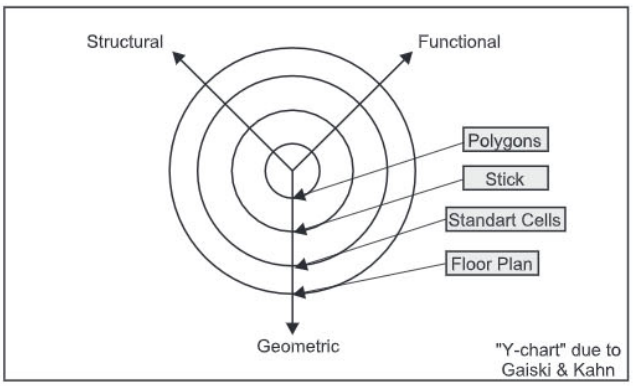


Рис. 2.4. Уровни описаний-абстракций в области геометрических моделей

В области функциональных моделей на наиболее абстрактном уровне функция целой системы может быть описана в терминах алгоритма, подобно алгоритму для компьютерной программы.

В области структурных моделей на высшем уровне абстракции структура системы может быть описана как разнообразные соединения следующих компонент:

- процессоры (Processors);
- модули памяти (Memories);
- коммутаторы (Switches);
- устройства ввода-вывода (I/O devices).

Этот уровень часто называют уровнем Processor-Memory-Switch (PMS) Level. На рис. 2.5 приведена структурная модель контроллера на этом уровне.

Данная модель содержит процессор, соединенный через коммутатор с модулем памяти и контроллерами для ввода входных данных (Data Inputs) и вывода выходной информации (Display Outputs) на устройства отображения.

В области геометрических моделей на верхнем уровне абстракции система представляется в виде схемы размещения компонент контроллера внутри VLSI-схемы (Floor Plan). Здесь (рис. 2.6) можно увидеть, как компоненты структурной модели контроллера размещаются на полупроводниковой пластине (Silicon Die).

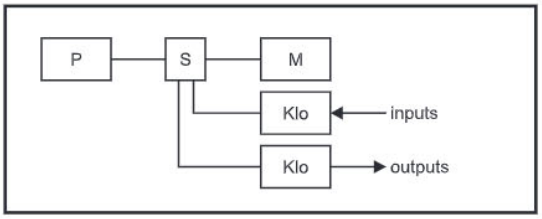


Рис. 2.5. Структурная схема контроллера: P — процессор; M — модуль памяти; S — коммутатор; Kio — контроллеры ввода/вывода

Следующий уровень абстракции в построении моделей соответствует второму кольцу на рис. 2.3 и описывает систему в терминах структурных единиц хранения и преобразования данных (Data Storage and Transformation). В структурной области этот уровень называется Register-Transfer Level (RTL). Модели RTL состоят из двух частей:

- тракт передачи данных (Data Path);
- устройство управления (Control Section).

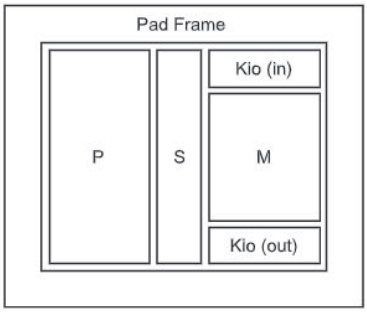


Рис. 2.6. Размещение структурных компонент на полупроводниковой пластине

Тракт передачи данных содержит регистры хранения данных (Data Storage Register). Данные передаются между ними, проходя через разнообразные устройства преобразования (Transformation Units — например, сумматоры, умножители и т.п.). Устройство управления упорядочивает выполнение операций компонентами тракта передачи данных. Структурная модель процессора, входящего в наш контроллер, на RTL-уровне приведена на рис. 2.7.

В функциональной области для создания моделей на этом уровне используется специальный язык межрегистровых передач (Register-Transfer Language) для спецификации операций системы. Хранение данных описывается с помощью регистровых переменных, а преобразования данных описываются с помощью арифметических и логических операторов.

Например, RTL-модель процессора может включать следующее описание:

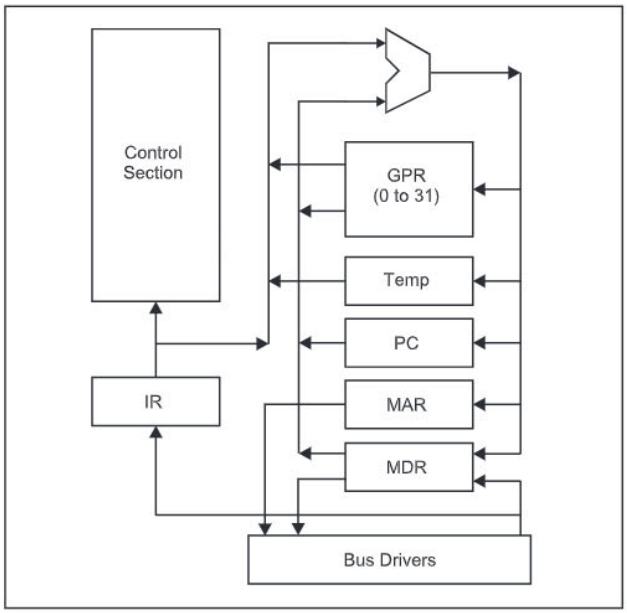


Рис. 2.7. Структурная модель процессора

```
MAR <- PC, memory_read <- 1
PC <- PC + 1
wait until ready = 1
IR <- memory_data
memory_read <- 0
```

Этот фрагмент кода описывает операции доставки (Fetching) инструкций из памяти. Содержание счетчика команд (PC Register) передается в Memory Address Register (MAR), и сигнал memory_read устанавливается в «1». Затем величина счетчика команд увеличивается на единицу. Далее, в момент, когда вход ready для модуля памяти memory устанавливается в «1», данные из памяти (memory_data) передаются на регистр команд (IR- Instruction Register). В заключение, сигнал (чтение из памяти) устанавливается в «0», отмечая конец цикла чтения.

В геометрической области вид модели зависит от физической среды. В нашем примере стандартные библиотечные ячейки (Standard Library Cell) могут быть использованы для реализации регистров и устройств преобразования информации (Data Transformation Units). Эти ячейки размещаются в полупроводниковой пластине кристалла.

Третьим уровнем абстракции, приведенным на рис. 2.3, является логическийуровень (Logic Level). На этом уровне моделируется структура, состоящая из взаимосвязанных вентилей. Для описания функций такой системы используются булевы уравнения (Boolean Equations) или таблицы истинности (Ttruth Tables). В физической среде заказной интегральной микросхемы (Custom Integrated Circuit) геометрическая структура может моделироваться с использованием виртуальной решетки (Virtual grid) или топологической схемы (Sticks).

На наиболее детализированном уровне абстракции можно моделировать транзисторную структуру микросхемы. На функциональном уровне используются дифференциальные уравнения (Differential Equations), оперирующие напряжениями и токами в цепях микросхемы. На геометрическом уровне используются многогранники (Polygons) для каждого слоя маски интегральной схемы. Необходимо отметить, что современные инструменты проектирования (Design Tools) позволяют автоматически транслировать описания верхних уровней на нижний, поэтому проектировщику нет необходимости работать на нижнем уровне абстракции.

Выше были рассмотрены различные виды моделей, которые используются для представления различных уровней функциональной, структурной и физической компоновки системы (Arrangement of a System). Существуют также различные представления этих моделей. Например, одной из форм представления структурной модели является ее схемотехническая реализация (Circuit Schematic). Здесь графические символы используются для представления подсистем, которые соединяются между собой посредством линий, представляющих проводные соединения (Wires). Эта графическая форма предпочтительна для проектировщиков. Однако та же самая структурная информация может быть представлена текстуально в форме списка соединений (Netlist).

В области построения функциональных моделей обычно используется текстовая форма представления. При этом для описания операций системы используются обычно языки спецификаций, которые базируются на формальных математических методах, таких, как временная логика или абстрактные цифровые автоматы (Abstract State Machines).

Другие формы представления служат для моделирования систем в целях тестирования и верификации и базируются на обычных языках программирования.

Последняя группа форм представления ориентируется на синтез аппаратуры (Hardware Synthesis) и обычно имеет сокращенный набор программных конструкций, так как далеко не все программные конструкции обычных языков программирования можно применить для описания аппаратуры.

Языками спецификаций, моделирования и синтеза аппаратуры, именуемыми также языками написания моделей аппаратуры (Modeling Languages), являются рассмотренные в первой главе языки описания аппаратуры (HDLs).

VHDL включает возможности для описания функций и структуры на различных уровнях абстракции. VHDL предназначен для разработки спецификации, выполнения процесса моделирования, а также используется для синтеза аппаратуры при условии, что используется определенное подмножество этого языка, чьи программные конструкции транслируются в соответствующие вентильные схемы [6a, 47a, 58a, 62a].

2.3. Интерфейс и архитектура

Любая модель на языке VHDL (VHDL-компонент), как мы видели в главе 1, содержит две основные части:

- описание интерфейса-сущность (Entity);
- описание его функционирования-архитектура (Architecture).

Например, если у VHDL-компонента один входной сигнал **a_input** и один выходной сигнал **b_output**, то описание интерфейса принимает следующий вид (рис. 2.8). Здесь ключевые слова **in**, **out** задают режимы работы портов компонента. Существуют еще несколько возможных режимов, например, режим **inout** используется в случае двунаправленных сигналов-портов (Bidirectional Signals).

```
-----------------------------------------------
-- 2-INPUT VHDL COMPONENT
-----------------------------------------------
ENTITY vhdl_component IS
 PORT( SIGNAL a_in : IN BIT;
 SIGNAL b_out : OUT BIT);
END vhdl_component;
```

Рис. 2.8

Ключевое слово **signal** слева от имени сигнала не несет дополнительной информации и может быть опущено: любой вход или выход в описании интерфейса по умолчанию является сигналом. Сигнал — одно из основных понятий языка VHDL, о котором мы не раз будем говорить далее. Ключевое слово **in** и имя **entity** после ключевого слова **end** также могут быть опущены. Стандарт языка VHDL-93 позволяет писать словосочетание **end entity** вместо **end**.

Заметим, что язык VHDL не делает различия между большими (**upper_case**) и малыми буквами (**lower_case**), а комментарии в языке VHDL задаются с помощью двойного тире (**double dash**). Это означает, что остальная часть строки является комментарием (**commentary**).

Интерфейс (Entity) может быть связан с несколькими архитектурами (архитектурными телами — Architectural Bodies), содержащими описания поведения компонента на различных уровнях абстракции.

Архитектура содержит две основные части:

1) часть, содержащую описания (декларации) (Architecture Declarative Part);
Эта часть располагается перед первым ключевым словом **begin** внутри архитектурного тела и может содержать:
   – описания типов данных (Data Types);
   – функции и процедуры (Subprograms);
   – компоненты более низкого уровня иерархии (Components);
   – описания сигналов (Signal Declarations);
2) часть, содержащую исполняемые операторы VHDL-кода (Architecture Statement Part).

Сейчас напишем VHDL-код для компонента **NAND** с двумя входами (рис. 2.9). Используя терминологию языка VHDL, будем называть этот компонент **NAND2 Design Entity**. Как мы уже знаем, **Design Entity** — это описание интерфейса для определенного объекта проекта. Входы и выходы такого интерфейса будем называть портами (Ports). На рис. 2.9 приведено описание на языке VHDL (VHDL Description) для интерфейса нашего компонента (**NAND2**). Это второй пример написания Entity Declaration. Такая декларация вводит в рассмотрение имя объекта проекта (Entity) и список его входных и выходных портов (Input and Output Ports), а также тип данных для каждого порта (например, **bit** означает, что сигнал в данном порту может принимать лишь два значения: логического нуля и логической единицы).

```
-----------------------------------------------
-- VHDL DESCRIPTION FOR 2-INPUT NAND
-----------------------------------------------
ENTITY nand2 IS
PORT(a,b : IN BIT;
c : OUT BIT);
END nand2;
ARCHITECTURE vhdl_code OF nand2 IS
BEGIN
c <= NOT(a AND b);
END vhdl_code;
```

Рис. 2.9

Внутри архитектуры этой простой модели используется лишь один оператор, который вычисляет новое значение для выходного сигнала.

Концепции построения более сложных VHDL-моделей рассмотрим далее на двух примерах [6а, 58а]:
   – описание RS-триггера (**RSFF** — RS Flip Flop);
   – описание 4-разрядного регистра (**REG4** — Four-Bit Register).

2.4. Поведенческое архитектурное тело

Мы можем написать поведенческую архитектуру (Behavioural Architecture), которая описывает функции объекта в абстрактном стиле, т. е. не затрагивая вопроса о структуре будущей аппаратуры, сосредоточась лишь на описании требуемых от объекта функций.

Такая поведенческая архитектура содержит только операторы процессов (Process Statements), которые содержат совокупность операторов, выполняющихся по-

следовательно. Эти операторы называются последовательными операторами (Sequential Statements) — они подобны аналогичным структурам в обычных языках программирования. Эти операторы предназначены для:

- вычисления выражений (Evaluating Expressions);
- присваивания значений переменным (Assigning Values to Variables);
- условного выполнения (Conditional Execution);
- повторного выполнения (Repeated Execution);
- вызова подпрограмм (Subprogram Calls).

В VHDL существуют также параллельные операторы (Parallel Statements), которые являются уникальными, присущими только языкам описания аппаратуры. Примером такого оператора является оператор присваивания значений сигналам (Signal Assignment). Он внешне похож на оператор присваивания значений переменным (Variable Assignment), но отличается способом исполнения (это будет подробно обсуждаться далее). Для параллельных операторов порядок их выполнения не совпадает с порядком их написания внутри архитектурного тела.

Рассмотрим поведенческое архитектурное тело для триггера **RSFF**, выполненное в стиле потока данных (Data Flow Style) (рис. 2.10). Эта архитектура использует лишь два оператора присваивания значения сигналам для двух выходов триггера (Q, QB). С помощью ключевого слова **after** и соответствующих временных значений (2 ns, 2 ns) вводятся инерционные задержки (delays) для выходных реакций **RSFF** [58a].

```
-----------------------------------------------
-- Entities needed to make rsff compile
-----------------------------------------------
ENTITY rsff IS
 PORT ( set, reset : IN BIT;
 q, qb : INOUT BIT);
END rsff;
-----------------------------------------------
ARCHITECTURE behave OF rsff IS
BEGIN
 q <= NOT( qb AND set ) AFTER 2 ns;
 qb <= NOT( q AND reset ) AFTER 2 ns;
END behave;
-----------------------------------------------
```

Рис. 2.10

На рис. 2.11 приведено поведенческое архитектурное тело для триггера **RSFF**. Оно выполнено в чисто поведенческом стиле (Pure Behavioral Style). Эта архитектура использует **процесс** (еще одно основное понятие языка VHDL). Процесс начинается с ключевого слова **begin** и заканчивается ключевым словом **end process**. Только внутри процесса разрешено использовать последовательные операторы (каким является, в частности, и используемый здесь оператор **if else**).

В данном случае используется процесс вместе с его списком чувствительности (Sensitivity List). Этот список содержит последовательность имен сигналов. Использование такого списка приводит к тому, что данный процесс активизируется системой моделирования (симулятором) (Simulation System, Simulator), т. е. начинает выполняться только в случае, если налицо изменение значения по крайней мере для одного сигнала из этого списка.

```
-----------------------------------------------
-- Entities needed to make rsff compile
-----------------------------------------------
ENTITY rsff IS
 PORT ( set, reset : IN BIT;
 q, qb : INOUT BIT);
END rsff;
-----------------------------------------------
ARCHITECTURE sequential OF rsff IS
BEGIN
 PROCESS (set, reset )
 BEGIN
 IF set = '1' AND reset = '0' THEN
 q <= '0' AFTER 2 ns;
 qb <= '1' AFTER 4 ns;
 ELSIF set = '0' AND reset = '1' THEN
 q <= '1' AFTER 4 ns;
 qb <='0' AFTER 2 ns;
 ELSIF set = '0' AND reset = '0' THEN
 q <= '1' AFTER 2 ns;
 qb <= '1' AFTER 2 ns;
 END IF;
 END PROCESS;
END sequential;
```

Рис. 2.11

Еще одно чисто поведенческое архитектурное тело для **RSFF**, которое использует процесс без списка чувствительности, приведено на рис. 2.12. Вместо списка чувствительности здесь используется оператор **wait** (в конструкции **wait on**). Обратим внимание на то, что в данном случае этот оператор находится в конце процесса. Такое применение оператора **wait** является полностью эквивалентной заменой списка чувствительности: процесс ждет изменения значения по крайней мере одного из сигналов, перечисленных после ключевых слов **wait on**.

```
-----------------------------------------------
-- Entities needed to make rsff compile
-----------------------------------------------
ENTITY rsff IS
 PORT ( set, reset : IN BIT;
 q, qb : INOUT BIT);
END rsff;
-----------------------------------------------
ARCHITECTURE sequential1 OF rsff IS
BEGIN
 PROCESS
 BEGIN
 IF set = '1' AND reset = '0' THEN
 q <= '0' AFTER 2 ns;
 qb <= '1' AFTER 4 ns;
 ELSIF set = '0' AND reset = '1' THEN
```

```
 q <= '1' AFTER 4 ns;
 qb <='0' AFTER 2 ns;
 ELSIF set = '0' AND reset = '0' THEN
 q <= '1' AFTER 2 ns;
 qb <= '1' AFTER 2 ns;
 END IF;
 WAIT ON set, reset;
 END PROCESS;
END sequential1;
```

Рис. 2.12

Если же мы напишем оператор **wait** в начале процесса, это приведет к неверным результатам. Такова архитектура, приведенная на рис. 2.13. Ниже будут приведены временные диаграммы процессов моделирования для различных архитектурных тел триггера **RSFF**, и можно будет видеть справедливость этого замечания.

```
-----------------------------------------------
-- Entities needed to make rsff compile
-----------------------------------------------
ENTITY rsff IS
 PORT ( set, reset : IN BIT;
 q, qb : INOUT BIT);
END rsff;
-----------------------------------------------
ARCHITECTURE sequential2 OF rsff IS
BEGIN
 PROCESS
 BEGIN
 WAIT ON set, reset;
 IF set = '1' AND reset = '0' THEN
 q <= '0' AFTER 2 ns;
 qb <= '1' AFTER 4 ns;
 ELSIF set = '0' AND reset = '1' THEN
 q <= '1' AFTER 4 ns;
 qb <='0' AFTER 2 ns;
 ELSIF set = '0' AND reset = '0' THEN
 q <= '1' AFTER 2 ns;
 qb <= '1' AFTER 2 ns;
 END IF;
 END PROCESS;
END sequential2;
```

Рис. 2.13

Перейдем теперь к построению поведенческого архитектурного тела для 4-разрядного регистра (**REG4**) [6а]. Описание этого регистра приведено на рис. 2.15. В этом архитектурном теле содержимое после ключевого слова **begin** включает один процесс, который описывает поведение регистра **REG4**. Описание процесса начинается с имени процесса (**storage**) и заканчивается ключевыми словами **end process**.

```
ENTITY reg4 IS
 port ( d0, d1, d2, d3, en, clk : in bit;
 q0, q1, q2, q3 : out bit );
END reg4 ;
```

Рис. 2.14

```
ARCHITECTURE behav OF reg4 IS
BEGIN
 storage : PROCESS
 VARIABLE stored_d0, stored_d1, stored_d2, stored_d3 : BIT;
 BEGIN
 IF en = '1' AND clk = '1' THEN
 stored_d0 := d0;
 stored_d1 := d1;
 stored_d2 := d2;
 stored_d3 := d3;
 END IF;
 q0 <= stored_d0 AFTER 5 ns;
 q1 <= stored_d1 AFTER 5 ns;
 q2 <= stored_d2 AFTER 5 ns;
 q3 <= stored_d3 AFTER 5 ns;
 WAIT ON d0, d1, d2, d3, en, clk;
 END PROCESS;
END behav;
```

Рис. 2.15

Этот процесс определяет последовательность действий, которые должен выполнять моделируемый объект.Эти действия управляют изменением величин сигналов интерфейса объекта с помощью операторов присваивания значений сигналам (Signal Assignment Statements).

Когда начинается процесс моделирования, все сигналы устанавливаются в состояние логического нуля, а сам процесс активизируется. Переменные процесса (Process's Variables) также инициализируются нулями. Первый оператор процесса проверяет значения сигналов **en** и **clk** интерфейса **REG4** entity. Если оба эти сигнала находятся в состоянии логической единицы, то выполняются операторы между ключевыми словами **then** и **end if**: тем самым переменные процесса получают значения входных значений интерфейса.

После условного оператора **if** в теле процесса располагаются четыре оператора присвоения значений сигналам. При этом новые значения сигналов на выходах интерфейса **REG4** entity (q1, q2, q3, q4) устанавливаюся через 5 ns (так как именно с помощью ключевого слова **after** задаются инерциональные задержки — Inertial Delay).

Когда все эти операторы выполнены, процесс достигает оператора ожидания (Wait Statement) и становится неактивным. Процесс будет находиться в этом состоянии ожидания до тех пор, пока хотя бы один из сигналов, перечисленных в операторе **wait (d0, d1, d2, d3, en, clk)**, не изменит своего состояния.

В момент, когда произойдет изменение сигнала хотя бы на одном из вышеупомянутых входных портов, опреаторы процесса начнут выполняться вновь, стартуя от ключевого слова **begin**, и весь цикл повторится вновь. Заметим, что пока процесс находится в состоянии ожидания, величины переменных процесса не теряются — таким образом процесс хранит состояние системы.

2.5. Структурное архитектурное тело

Альтернативным путем описания реализации объекта **RSFF** entity и **REG4** entity является композиция его подсистем. Можно дать структурное описание (Structural Description) для реализации объекта проекта (Entity's Implementation). Таким структурным описанием является структурное архитектурное тело (Structural Architecture Body).

На рис. 2.16 показано, как может быть скомпонован объект **RSFF** из двух двухвходовых вентилей **NAND2**.

```
-----------------------------------------------
-- Entities needed to make rsff compile
-----------------------------------------------
entity nand2 is
 port (a, b: in bit; c : out bit);
end nand2;
architecture behave of nand2 is
begin
 c <= not(a and b) after 1 ns;
end behave;
-----------------------------------------------
ENTITY rsff IS
 PORT ( set, reset : IN BIT;
 q, qb : INOUT BIT);
END rsff;
-----------------------------------------------
ARCHITECTURE netlist OF rsff IS
 COMPONENT nand2
 PORT ( a, b : IN BIT;
 c : OUT BIT);
 END COMPONENT;
BEGIN
 U1: nand2
 PORT MAP (set, qb, q);
 U2: nand2
 PORT MAP (reset, q, qb);
END netlist;
-----------------------------------------------
```

Рис. 2.16

Вначале необходимо описать интерфейсы и архитектурные тела для всех подсистем (компонент). На рис. 2.16 (вверху) приводится описание интерфейса для **NAND2** и соответствующее архитектурное тело **behave**.

Сама структура **RSFF**, приведенная на рис. 2.16, описывается на VHDL структурным архитектурным телом **netlist**. Еще один вариант структурного архитектурного тела для **RSFF**, использующего другую модель базового компонента — **mynand**, приведен на рис. 2.17 (**netlist1**).

```
-----------------------------------------------
-- Entities needed to make rsff compile
-----------------------------------------------
entity mynand is
```

```
 port (a, b: in bit; c : out bit);
end mynand;
architecture version1 of mynand is
begin
 c <= not(a and b) after 3 ns;
end version1;
-------------------------------------------------
ARCHITECTURE netlist1 OF rsff IS
 COMPONENT mynand
 PORT ( a, b : IN BIT;
 c : OUT BIT);
 END COMPONENT;
BEGIN
 U1: mynand
 PORT MAP (set, qb, q);
 U2: mynand
 PORT MAP (reset, q, qb);
END netlist1;
-------------------------------------------------
```

Рис. 2.17

Описывая объект **REG4** на VHDL, также необходимо описать архитектурные тела для всех подсистем (компонент) на рис. 2.18.

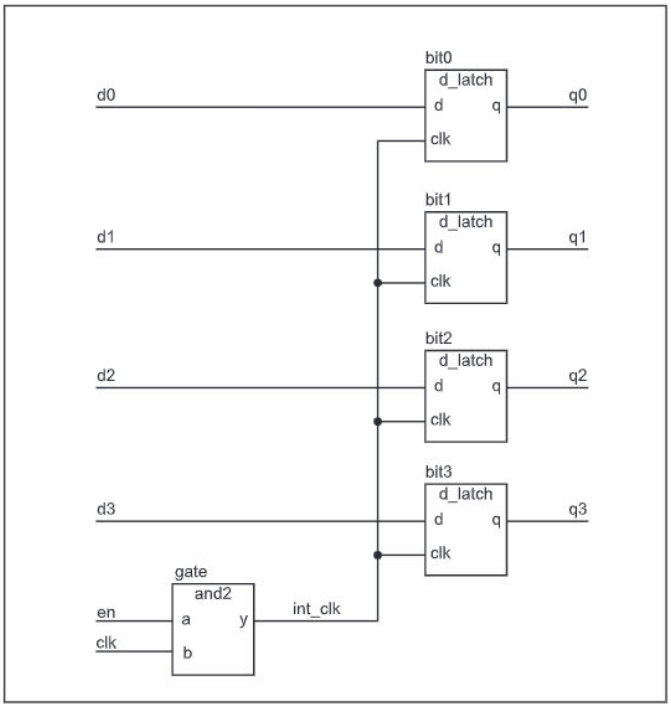


Рис. 2.18. Структура 4-разрядного регистра

На рис. 2.19 приведено описание интерфейса для **DFF** и соответствующее архитектурное тело **basic** для **NAND2**.

```
entity d_latch is
 port ( d, clk : in bit; q : out bit );
end d_latch;
architecture basic of d_latch is
begin
 latch_behavior: process
 begin
 if clk = '1' then
 q <= d ; -- after 2 ns;
 end if;
 wait on clk, d;
 end process latch_behavior;
end basic;
-----------------------------------------
entity and2 is
 port ( a, b : in bit; y : out bit );
end and2;
architecture basic of and2 is
begin
 and2_behavior : process
 begin
 y <= a and b ; -- after 2 ns;
 wait on a, b;
 end process ;
end basic;
```

Рис. 2.19

В структурном архитектурном теле **struct** перед ключевым словом **begin** располагается описание внутренних для архитектуры сигналов (Internal Signals), служащих для соединения между собой базовых компонент (Signal Declarations). Так, например, архитектура struct использует внутренний сигнал int_clk, имеющий тип данных bit (в общем же случае сигналы языка VHDL (VHDL Signals) могут быть декларированы с более сложными типами данных). Внутри архитектурного тела порты интерфейса также могут быть интерпретированы как сигналы.

В этой же области архитектуры (до ключевого слова **begin**) располагаются описания базовых компонент, из которых она строится. Так, архитектуры **netlist** и netlist1 содержат описания базовых компонент **nand2** и **mynand** соответственно. Архитектура **struct** содержит описания сразу двух базовых компонент (**d_latch** и **and2**), из которых строится регистр **REG4**.

Во второй части архитектурного тела находится совокупность реализаций компонент (Component Instances) регистра **REG4** entity. Каждая реализация является копией объекта (entity), представляющего подсистему и использующего соответствующее архитектурное тело.

Ключевые слова **port map** специфицируют межсоединения портов каждой реализации компонента с внутренними сигналами архитектуры и сигналами интерфейса основного объекта, который построен на базе этих компонент.

В архитектурах **netlist** и **netlist1** создаются по два экземпляра компонента (**nand2** и **mynand** соответственно) с именами U1 и U2.

Архитектура **struct** содержит 4 реализации компонента **d_latch** (с именами **bit0**, **bit1**, **bit2**, **bit3**) и одну реализацию компонента **and2** (с именем **gate**).

Например, **bit0** — это реализация **d_latch entity**, для которой порт **d** соединен с сигналом **d0**, порт **q** соединен с сигналом **q0**, а порт **clk** — с сигналом **int_clk**.

```
ARCHITECTURE struct OF reg4 IS
 COMPONENT d_latch
 PORT(d,clk: IN bit; q: OUT bit);
 END COMPONENT;
 COMPONENT and2
 PORT(a,b: IN bit; y: OUT bit);
 END COMPONENT;
 SIGNAL int_clk : bit;
BEGIN bit0 : d_latch
 PORT MAP (d0, int_clk, q0);
 bit1 : d_latch
 PORT MAP (d1, int_clk, q1);
 bit2 : d_latch
 PORT MAP (d2, int_clk, q2);
 bit3 : d_latch
 PORT MAP (d3, int_clk, q3);
 gate : and2
 PORT MAP (en, clk, int_clk);
END struct;
```

Рис. 2.20

На рис. 2.21—2.25 приведены результаты моделирования для вышеприведенных моделей **RSFF** и **REG4** симулятором **ModelSim**.

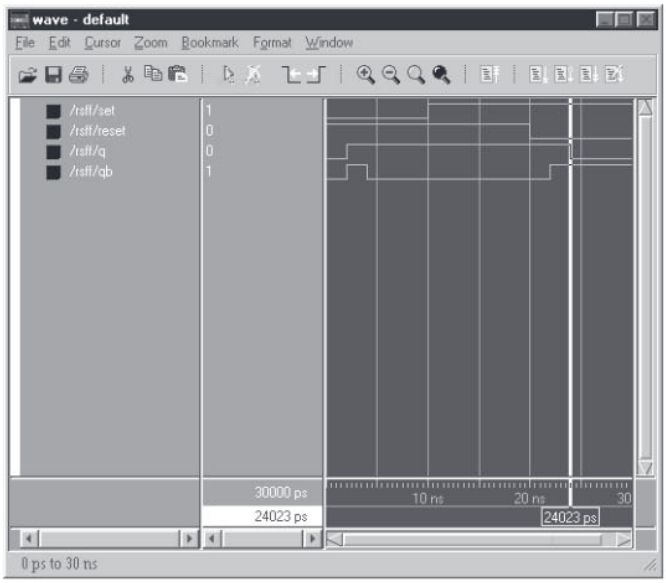


Рис. 2.21. Результаты моделирования для модели behave_RSFF

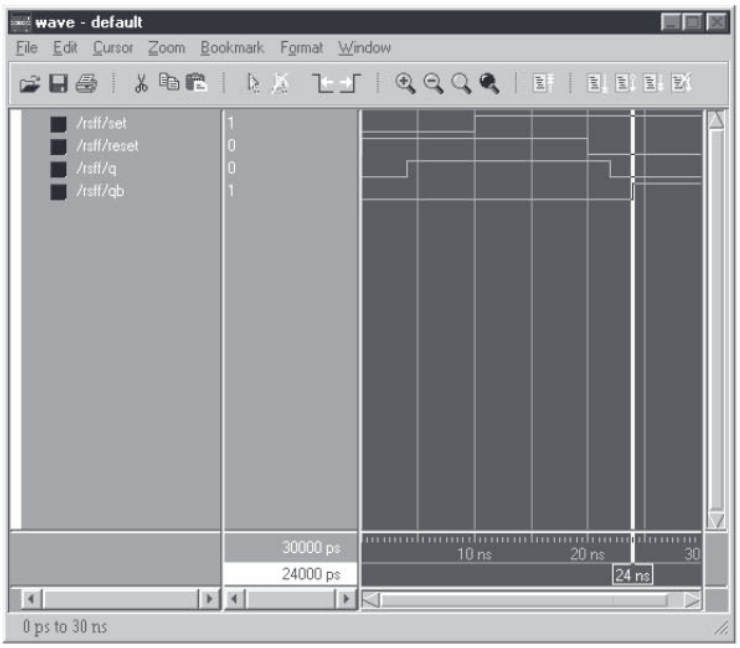


Рис. 2.22. Результаты моделирования для модели sequential_RSFF

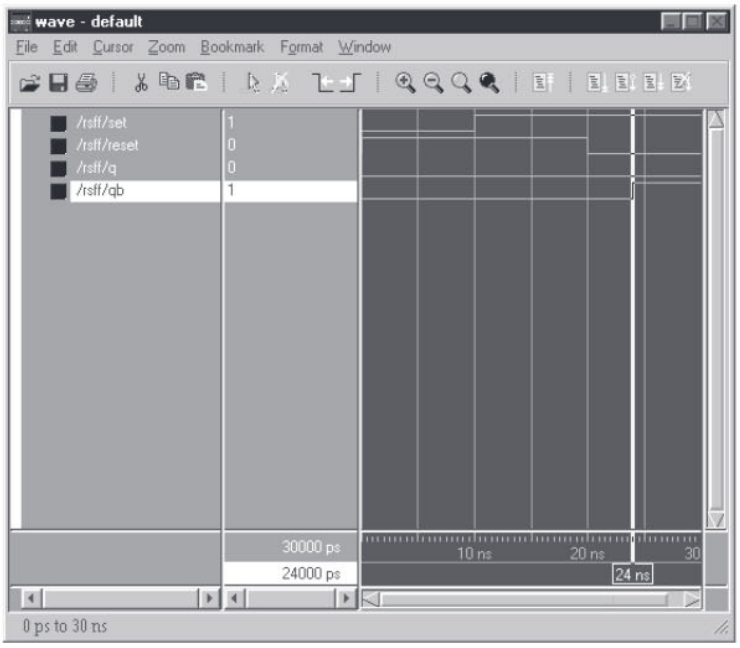


Рис. 2.23. Результаты моделирования для модели sequential2_RSFF

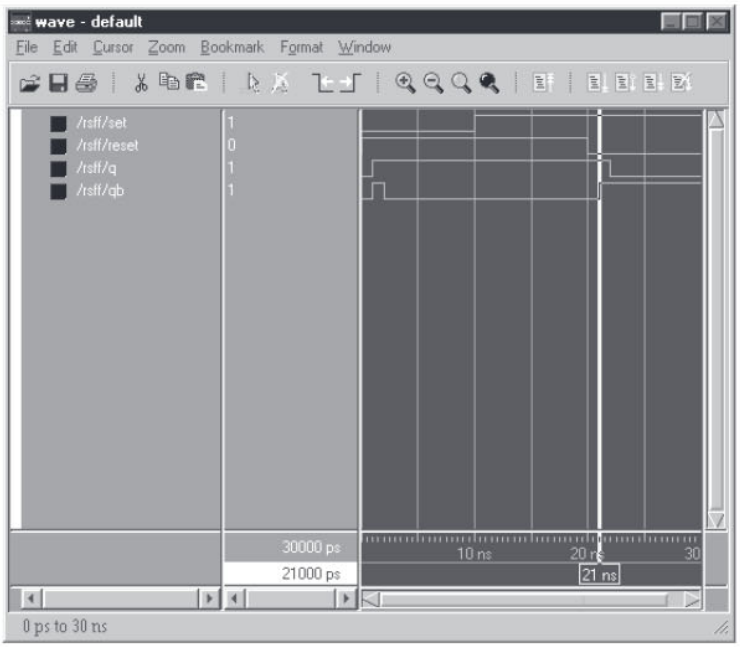


Рис. 2.24. Результаты моделирования для модели netlist_RSFF

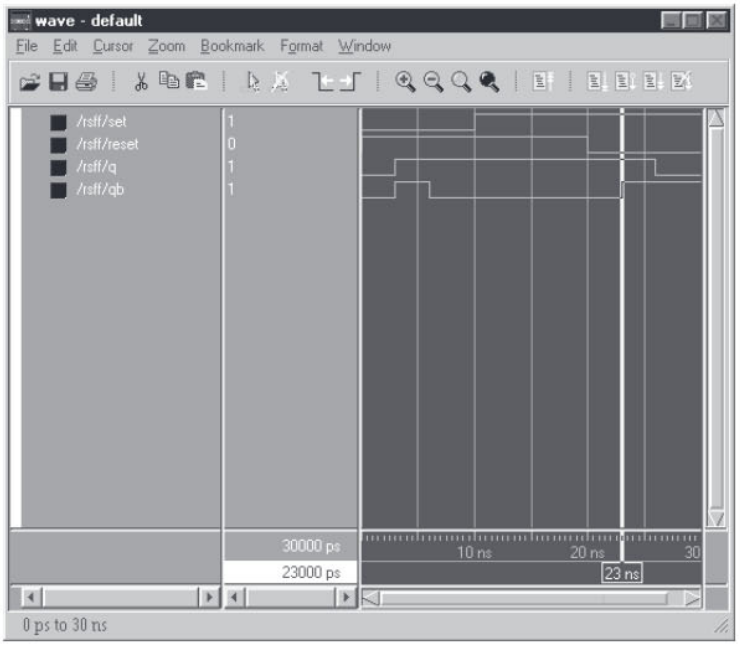


Рис. 2.25. Результаты моделирования для модели netlist1_RSFF

2.6. Смешанные структурно-поведенческие модели

Модели могут быть не обязательно чисто структурными или чисто поведенческими (Purely Structural or Purely Behavioral). Часто полезно специфицировать модель, содержащую как взаимосвязанные копии компонент (Interconnected Component Instances), так и процессы. Сигналы используются для взаимосвязи копий компонент и процессов. Сигнал может быть ассоциирован с портом копии компонента и устанавливаться в определенное состояние внутри некоторого процесса.

Можно написать такую гибридную модель (Hybrid Model), включающую как копии компонент, так и операторы процессов в архитектурном теле [6a, 58a, 77a]. Эти операторы называются параллельными (Concurrent Statements), так как соответствующие процессы выполняются параллельно в ходе моделирования.

Подобная модель приведена на рис. 2.27. Эта модель описывает параллельный сумматор (Serial Adder), состоящий из комбинационного блока combinational logic (1-Bit Full Adder) и D-триггера (DFF) (рис. 2.26).

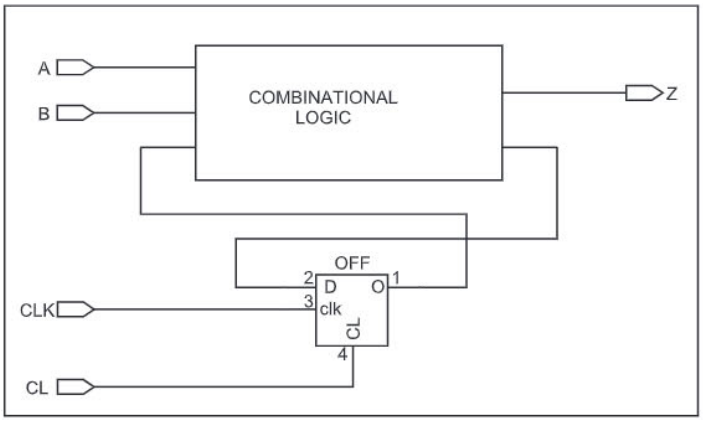


Рис. 2.26. Последовательный сумматор (Serial Adder)

```
-------------------------------------
-- VHDL DESCRIPTION FOR SERIAL ADDER
-------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY serial_adder IS
 PORT(a,b,clk,cl : IN std_logic;
 z : OUT std_logic);
END serial_adder;
ARCHITECTURE struct OF serial_adder IS
 COMPONENT combinational_logic
 PORT(a,b,c_in : IN std_logic;
 z,carry : OUT std_logic);
 END COMPONENT;
 PROCEDURE dff (SIGNAL d,clk,cl : IN std_logic;
 SIGNAL q : OUT std_logic) IS
```

```
 BEGIN
 IF (cl = '0') THEN
 q <= '0' AFTER 5 ns;
 ELSIF (clk'event AND clk = '1') THEN
 q <= d AFTER 5 ns;
 END IF;
 END dff;
 SIGNAL c1,c2 : std_logic;
BEGIN
 U1: combinational_logic PORT MAP(a => a, b => b, c_in => c1,
 z => z, carry => c2);
 PROCESS
 BEGIN
 dff(clk => clk, cl => c1, d => c2, q => c1);
 WAIT ON clk, cl, c2;
 END PROCESS;
END struct;
```

Рис. 2.27

Полный сумматор имеет структурное архитектурное тело, а D-триггер описывается поведенческой моделью внутри архитектуры последовательного сумматора. Таким образом, у нашего последовательного сумматора смешанное структурно поведенческое архитектурное тело (Mixed Structural-Behavioral Architectural Body).

2.7. Тесты

Выше было отмечено, что выполнение тестирования посредством моделирования является одной из важных причин построения моделей. Часто для тестирования модели, написанной на языке VHDL (VHDL model), используется другая модель, включающая в себя тестируемую модель (Enclosing model). Такая модель называется тестом (Test Bench). Это имя привнесено по аналогии с тестом реального оборудования (Real Hardware Test Bench), в котором тестируемый прибор принимает входные воздействия от входного генератора сигналов (Signal Generator) и параллельно обследуется с помощью специальных зондов (Signal Probes) [6a, 47a, 58a, 62a].

VHDL Test Bench содержит архитектурное тело, содержащее тестируемый компонент и процессы, генерирующие последовательности значений для сигналов, подсоединенных к входам этого тестируемого компонента (рис. 2.28).

Архитектурное тело может также содержать процессы, которые контролируют выходные реакции тестируемого компонента. При этом используются средства отображения (Monitoring Facilities) симулятора для наблюдения выходов.

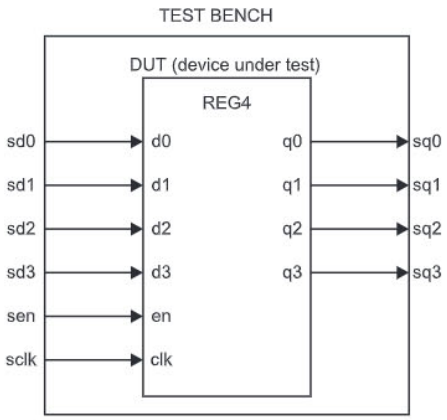


Рис. 2.28. Структура испытательной программы (Test Bench)

Test Bench для поведенческой модели (реализации) (Behavioral Implementation) регистра **REG4** приведен на рис. 2.29. Здесь описание интерфейса (Entity Declaration) не содержит списка портов (Port List), так как **Test Bench** является полностью самодостаточным (автономным, замкнутым — Self-Contained).

```
-------------------------------------------------------------
-- TEST BENCH EXAMPLE
-------------------------------------------------------------
ENTITY testbench_example IS
END testbench_example;
ARCHITECTURE test_reg4 OF testbench_example IS
COMPONENT reg4
 PORT ( d0, d1, d2, d3, en, clk : in BIT;
 q0, q1, q2, q3 : out BIT);
END COMPONENT;
 SIGNAL sd0, sd1, sd2, sd3, sen, sclk, sq0, sq1, sq2, sq3 : BIT;
BEGIN
 dut : reg4
 PORT MAP ( sd0, sd1, sd2, sd3, sen, sclk, sq0, sq1, sq2, sq3 );
 stimulus : PROCESS
 BEGIN
 sen <= '0'; sclk <= '0';
FOR i IN 0 TO 5 LOOP
 WAIT FOR 20 ns;
 sd0 <= '1'; sd1 <= '1'; sd2 <= '1'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sd0 <= '0'; sd1 <= '1'; sd2 <= '1'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sd0 <= '1'; sd1 <= '0'; sd2 <= '1'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '1';
END LOOP;
 WAIT;
 END PROCESS;
END test_reg4;
```

Рис. 2.29

Архитектурное тело содержит сигналы, подсоединенные к входным и выходным портам тестируемого компонента **DUT** (Device Under Test). Процесс с меткой **stimulus** обеспечивает последовательность тестовых значений (Test Values) на входах тестируемого компонента (Input Signals) с помощью совокупности операторов присваивания значений для сигналов с вкраплениями операторов ожидания. Каждый оператор ожидания специфицирует паузу в 20 ns, в течение которой регистр **REG4** устанавливает на своих выходах новые значения сигналов.

Можно использовать систему моделирования (Simulator) для наблюдения этих значений сигналов **q0**, **q1**, **q2**, **q3**. Так выполняется верификация регистра **REG4** с целью убедиться, что он функционирует корректно. Когда все входные воздействия сгенерированы, процесс моделирования завершается.

На рис. 2.30 приведен еще один вариант теста с использованием оператора ASSERT для выдачи диагностических сообщений внутри процесса с меткой **stimulus** (это будет подробно рассмотрено далее).

```
LIBRARY IEEE ;
USE IEEE.std_logic_1164.ALL ;
USE std.textio.ALL ;
USE IEEE.std_logic_textio.ALL ;
ENTITY test_bench IS
END test_bench;
ARCHITECTURE test_reg4 OF test_bench IS
COMPONENT reg4
 PORT ( d0, d1, d2, d3, en, clk : in BIT; q0, q1, q2, q3 : out
BIT);
END COMPONENT;
 SIGNAL sd0, sd1, sd2, sd3, sen, sclk, sq0, sq1, sq2, sq3 : BIT;
BEGIN
 dut : reg4
 PORT MAP ( sd0, sd1, sd2, sd3, sen, sclk, sq0, sq1, sq2, sq3 );
 stimulus : PROCESS
 BEGIN
 sen <= '0'; sclk <= '0';
FOR i IN 0 TO 5 LOOP
 WAIT FOR 20 ns;
 sd0 <= '1'; sd1 <= '1'; sd2 <= '1'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sd0 <= '0'; sd1 <= '1'; sd2 <= '1'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
 ASSERT ((sq0='1')AND(sq1='1')AND(sq2='1')AND(sq3='1'))
 REPORT "D_LATCH REACTION IS NOT GOOD!"
 SEVERITY ERROR;
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sd0 <= '1'; sd1 <= '0'; sd2 <= '1'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
```

```
 ASSERT ((sq0='0')AND(sq1='1')AND(sq2='1')AND(sq3='1'))
 REPORT "D_LATCH REACTION IS NOT GOOD!"
 SEVERITY ERROR;
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sd0 <= '1'; sd1 <= '1'; sd2 <= '0'; sd3 <= '1';
 sen <= '1'; sclk <= '1';
 ASSERT ((sq0='1')AND(sq1='0')AND(sq2='1')AND(sq3='1'))
 REPORT "D_LATCH REACTION IS NOT GOOD!"
 SEVERITY ERROR;
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sd0 <= '1'; sd1 <= '1'; sd2 <= '1'; sd3 <= '0';
 sen <= '1'; sclk <= '1';
 ASSERT ((sq0='1')AND(sq1='1')AND(sq2='0')AND(sq3='1'))
 REPORT "D_LATCH REACTION IS NOT GOOD!"
 SEVERITY ERROR;
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '0';
 WAIT FOR 20 ns;
 sen <= '1'; sclk <= '1';
 WAIT FOR 20 ns;
 ASSERT (false)
 REPORT "END OF CYCLE"
 SEVERITY NOTE;
END LOOP;
 WAIT;
 END PROCESS;
END test_reg4;
```

Рис. 2.30

Результаты моделирования, полученные с помощью этой испытательной программы, для регистра **REG4** приведены на рис. 2.31.

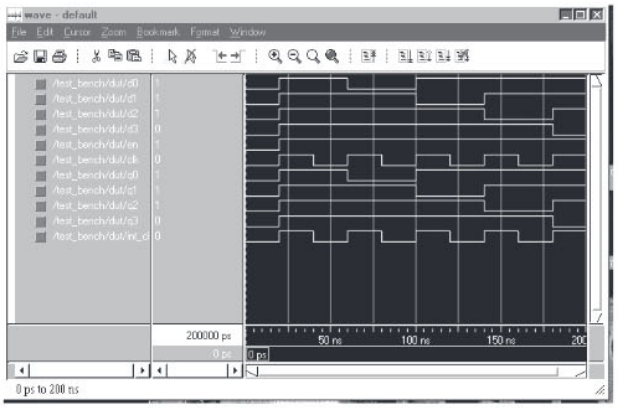


Рис. 2.31. Результаты моделирования для регистра REG4

Подобным же образом построена испытательная программа для триггера **RSFF** (рис. 2.32). Здесь дополнительно к упомянутой выше испытательной программе для **REG4** используется механизм конфигураций (Configurations). Этот механизм подробно будет описан далее, здесь же только отметим, что с помощью конфигурации **RSFFcon1** выбирается архитектура **пе-**

tlist1, конфигурация **RSFFcon2** определяет использование архитектуры **sequential**, а конфигурация **RSFFcon3** — архитектуры **netlist**.

```
LIBRARY IEEE ;
USE IEEE.std_logic_1164.ALL ;
USE std.textio.all ;
USE IEEE.std_logic_textio.all ;
ENTITY rsff_tb IS
 -- no need for any ports
END rsff_tb ;
CONFIGURATION rsffcon1 OF rsff IS
 FOR netlist1
 FOR U1,U2: mynand USE ENTITY WORK.mynand(version1);
 END FOR;
 END FOR;
END rsffcon1;
CONFIGURATION rsffcon2 OF rsff IS
 FOR sequential
 END FOR;
END rsffcon2;
CONFIGURATION rsffcon3 OF rsff IS
 FOR netlist
 FOR U1,U2: nand2 USE ENTITY WORK.nand2(behave);
 END FOR;
 END FOR;
END rsffcon3;
ARCHITECTURE arch OF rsff_tb IS
 COMPONENT rsff
 PORT ( set, reset : IN BIT ;
 q, qb : INOUT BIT) ;
 END COMPONENT ;
 SIGNAL sset : BIT := '1' ;
 SIGNAL sreset : BIT := '0' ;
 SIGNAL sq : BIT ;
 SIGNAL sqb : BIT ;
 CONSTANT period : time := 50 ns ;
-- FOR dut:rsff USE CONFIGURATION WORK.rsffcon1;
-- FOR dut:rsff USE CONFIGURATION WORK.rsffcon2;
 FOR dut:rsff USE CONFIGURATION WORK.rsffcon3;
BEGIN
 dut:rsff
 PORT MAP(
 set => sset,
 reset => sreset,
 q => sq,
 qb => sqb) ;
 sset <=NOT sset after period/2;
 sreset<=NOT sreset after period/2;
END arch;
```

Рис. 2.32

Соответствующие результаты моделирования для архитектур триггера **RSFF** приведены на рис. 2.33.

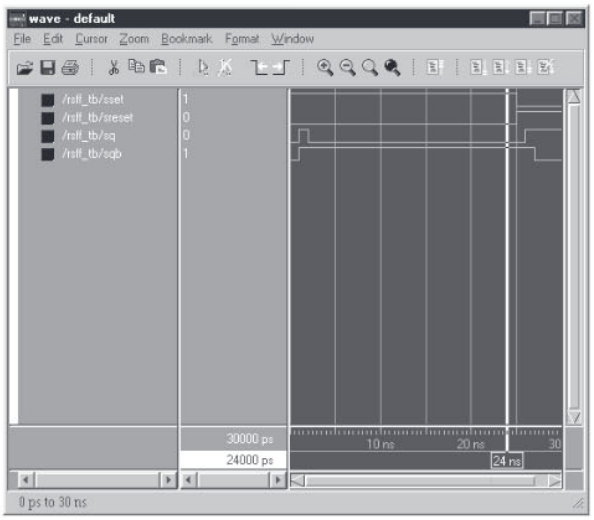


Рис. 2.33,а. Результаты моделирования для архитектуры netlist триггера RSFF

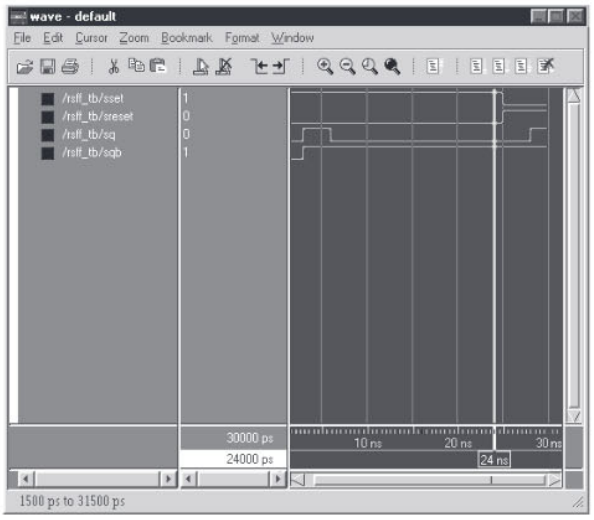


Рис. 2.33,б. Результаты моделирования для архитектуры netlist1 триггера RSFF

2.8. Анализ, уточнение и исполнение моделей

Одной из главных причин написания моделей систем является их пригодность для выполнения процесса моделирования. Построение модели содержит три этапа [6а]:

- анализ модели (Analysis);
- тщательная проработка и уточнение модели (Elaboration);
- исполнение модели (Execution).

Два первых этапа используются также для второго важного применения моделей: выполнения процесса логического синтеза (Logic Synthesis).

На этапе анализа описание на языке VHDL проверяется на наличие разнообразных ошибок. Как и большинство обычных языков программирования, VHDL имеет жестко определенные синтаксис (Syntax) и семантику (Semantics).

Синтаксисом называется набор грамматических правил (Set of Grammatical Rules), которые определяют написание модели. Семантические правила (Rules of Semantics) определяют смысловое содержание программы (Meaning of a Program). Например, можно выполнить операцию сложения с двумя числами, но не с двумя процессами.

В течение этапа анализа проверяется описание модели и локализуются синтаксические и семантические ошибки. При этом нет необходимости анализировать всю модель системы немедленно. Вместо этого есть возможность анализировать единицы проектирования (Design Units), такие как описания интерфейсов и архитектурные тела, порознь. Если аналитик не находит ошибок в данной единице проектирования, он создает промежуточное представление (Intermediate Representation) этой единицы и сохраняет его в библиотеке. Точный механизм выполнения этой процедуры варьируется в различных инструментах моделирования и синтеза на языке VHDL (VHDL Tools).

Второй этап моделирования — тщательная проработка и уточнение модели — заключается в анализе иерархии проекта (Design Hierarchy) и создании всех объектов проекта, определенных в декларациях.

Окончательным продуктом (Ultimate Product) этого этапа (Design Elaboration) является совокупность операторов присвоения значений сигналам и процессов, извлеченных из архитектурных тел единиц проектов и помещенных в общую очередь для выполнения моделирования (третий этап — Execution).

Уточнение модели начинается на верхнем уровне модели (Top Level of a Model). Для **entity** самого верхнего уровня выбирается соответствующее архитектурное тело. Оно содержит операторы присвоения значений сигналам, операторы процессов и экземпляры компонент.

Каждый экземпляр компонента, в свою очередь, также имеет описание интерфейса и архитектурное тело. Это архитектурное тело точно так же состоит из сигналов, процессов и экземпляров других компонент более низкого иерархического уровня. Экземпляры этих сигналов и процессов создаются в соответствии с реализацией компонента, и затем операция уточнения повторяется для реализаций подкомпонент (sub-component instances). Второй этап завершается, когда достигнут нижний уровень иерархии для всех составных компонент проекта, т. е. все новые экземпляры компонент имеют чисто поведенческие архитектурные тела, не содержащие подкомпонент. Такие архитектуры непосредственно предназначены для моделирования. На рис. 2.34 показано, как процесс уточнения (elaboration) выполняется для модели регистра **REG4**.

Как только создан экземпляр процесса, создаются и инициализируются его переменные. Каждый экземпляр процесса соответствует определенному экземпляру компонента.

Третьим этапом моделирования является исполнение модели (Execution of the Model). При этом продвижение времени (Passage of Time) моделируется дискретными шагами (Discrete Steps), зависящими от того, когда происходят события. Поэтому используется термин моделирование дискретных событий (Discrete Event Simulation).

В определенный момент времени моделирования (Simulation Time) некоторый процесс может быть активизирован изменением значения сигнала, к которому он «чувствителен» (т. е. данный сигнал находится в теле этого процесса). Упомянутый процесс вновь начинает выполняться и может устанавливать новые значения для своих сигналов в более поздние моменты времени моделирования. Такая процедура называется планированием перехода сигнала из одного состояния в другое (Scheduling a Transaction). Если новая величина сигнала отлична от предыдущей, принято говорить, что происходит событие (Event), и вслед за этим другие процессы, чувствительные к значениям данного сигнала, активизируются.

Исполнение модели начинается с фазы инициализации (Initialization Phase). Здесь каждый сигнал получает начальное значение (Initial Value), зависящее от типа данных для данного сигнала. Время моделирования устанавливается в ноль, а затем каждый экземпляр процесса активизируется и выполняются его последовательные операторы.

Обычно процесс включает, по крайней мере, один оператор присвоения значений для сигнала. Такие операторы используются для планирования переходов сигналов из одного состояния в другое через определенные отрезки времени моделирования. Выполнение процесса продолжается до достижения им оператора ожидания, который переводит данный процесс в состояние ожидания. Процесс (не содержащий операторов ожидания) также переводится в состояние ожидания, когда выполнены все его операторы, а новых изменений значений сигналов, к которым он чувствителен, нет.

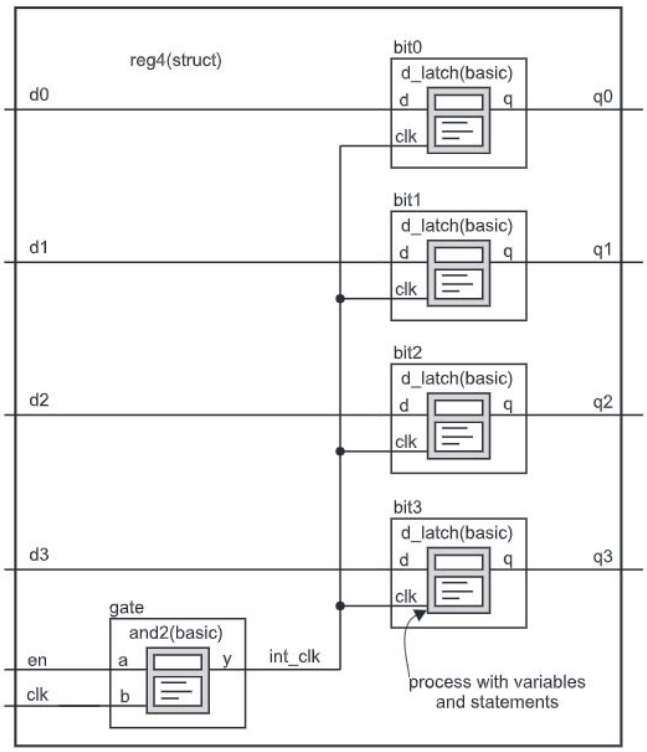


Рис. 2.34. Процесс уточнения модели (Model Elaboration) для регистра REG4

В течение цикла моделирования (Simulation Cycle) время моделирования продвигается посредством выполнения транзакций (Transaction) сигналов — их переходов в другие состояния в следующие моменты времени моделирования.

Когда все процессы переведены в состояние ожидания, цикл моделирования вновь повторяется. В случае же, если процесс моделирования достиг этапа, на котором уже не планируются будущие трансакции, это означает, что он завершается.

Глава 3. Краткое описание языка VHDL

3.1. Ресурсы VHDL как программной системы

Ресурсы языка VHDL как программной системы включают:

– средства для описания проектируемых систем во времени;
– средства для структурного описания проектируемых систем.

Ресурсы VHDL для описания проектируемых систем во времени предназначены для следующих целей (рис. 3.1):

– представление параллельных процессов, протекающих в функционирующей аппаратуре;
– представление задержек компонентов аппаратуры.

Средства для представления параллельных процессов (Parallel Process Representation) включают (рис. 3.1, *а*):

– параллельные операторы для работы с сигналами (Parallel (or Concurrent) Statements for Signals);
– VHDL-процессы (VHDL Processes);
– охраняемые блоки и охранные выражения (Guarded Blocks and Guard Expressions;
– оператор ожидания событий (Wait Statement).

Рис. 3.1,а. Ресурсы VHDL для описания проектируемых систем во времени

Рис. 3.1,б. Ресурсы VHDL для описания проектируемых систем во времени. Параллельные операторы для работы с сигналами

Параллельные операторы для работы с сигналами предназначены прежде всего для описания поведния комбинационной логики (Combinational Logic Description) (рис. 3.1, *б*). К таким операторам относятся:

- оператор безусловного присваивания для сигналов (Signal Assignment Statement);
- оператор условного присваивания для сигналов (Conditional Signal Assignment Statement);
- оператор присваивания по выбору для сигналов (Selected Signal Assignment Statement);
- оператор параллельного вызова подпрограмм (Concurrent Subprogram Call Statement);
- оператор блока (Block Statement);
- Statement);
- параллельный оператор проверки (Assert Statement).

Рис. 3.1,в. Ресурсы VHDL для описания проектируемых систем во времени. Процессы

Базовым элементом описания на языке VHDL является блок [2р]. Блоком называется ограниченный фрагмент текста, содержащий раздел описания (Declarative Part) и исполняемый раздел (Executive Part). Таким образом, и само архитектурное тело является блоком. В свою очередь, внутри архитектурного тела могут существовать другие внутренние блоки. Возможно любое число уровней такой вложенности. Блочная структура соответствует иерархическому представлению структуры проекта и позволяет для определенного блока установить условия защиты. Если условие защиты приобретает значение ИСТИНА, то это станет разрешением выполнения определенных внутренних операторов с охранными выражениями. Охраняемые блоки (Guarded Blocks) и охранные выражения (Guard Expressions) — мощное средство моделирования последовательностной логики во времени.

Еще одним важным инструментом моделирования на VHDL является VHDL-процесс (рис. 3.1, *б*). В случае, если какой-либо сигнал из списка чувствительности процесса или из списка аргументов эквивалентного оператора Wait (Wait Statement) меняет свое значение, процесс активизируется и выполняются операторы блока этого процесса. Это основной инструмент представления параллельных операций при моделировании цифровых систем.

VHDL-процессы преимущественно используются для описания поведения последовательностной логики (Sequential Logic Description) во времени (рис. 3.1, *б*). Внутри процесса, наряду с параллельными операторами, например оператором

присваивания для сигналов (Signal Assignment Statement), используются следующие последовательные операторы:

- оператор присваивания для переменных (Variable Assignment Statement);
- операторы условий и выбора (If Statement, Case Statement);
- оператор повторения (Loop Statement) и др.

Структурное описание (Structural Description) создается с помощью следующих средств (рис. 3.2):

- оператор создания копий базовых компонент (Component Instantiation Statement);
- оператор условной генерации копий компонент итеративным способом (Generate Statement);
- использование параметров настройки (Generics);
- средств описания конфигураций (Configuration Declaration).

Рис. 3.2. Ресурсы VHDL для структурного описания проектируемых систем

3.2. Краткое описание синтаксиса языка VHDL

Текст на языке VHDL представляет собой последовательность раздельных лексических элементов-лексем (Lexical Elements). Смежные лексемы отделяются разделителями (которые, в свою очередь, также являются лексемами — специальными символами), концами строк и знаками форматирования.

Лексемами являются:

- комментарии (Comments),
- идентификаторы (Identifiers),
- зарезервированные слова (Reserved Words),
- специальные символы (Special Symbols),
- числа (Numbers),
- символы (Characters),
- строки (Strings),
- битовые строки (Bit Strings).

Комментарии располагаются вслед за двумя смежными знаками тире (dashes) и до конца данной строки. Например:

```
a<= b + c; -- signal a calculation.
```

Идентификаторы называют понятия, необходимые для конкретной разработки программы. Идентификаторы конструируются из букв и цифр (возможно, соединенных символами подчеркивания) и должны отличаться от зарезервированных слов языка. Идентификаторы не могут содержать пробелов и располагаются более чем на одной строке исходного текста. Примерами корректно написанных идентификаторов являются: B, Y1, decoder, Data_Result.

А вот несколько некорректных идентификаторов:

next@value — использован некорректный символ @;
1bit_alu — идентификатор начинается с цифры;
_BIT — идентификатор начинается со знака подчеркивания;
dlatch__enable — использованы два смежных знака подчеркивания.

Список зарезервированных слов языка VHDL приведен на рис. 3.3.

abs	disconnect	label	port	srl
access	downto	library	postponed	subtype
after		linkage	procedure	
alias	else	literal	process	then
all	elsif	loop	protected	to
and	end		pure	transport
architecture	entity	map		type
array	exit	mod	range	
assert			record	unaffected
attribute	file	nand	register	units
	for	new	reject	until
begin	function	next	rem	use
block		nor	report	
body	generate	not	return	variable
buffer	generic	null	rol	
bus	group		ror	wait
	guarded	of		when
case		on	select	while
component	if	open	severity	with
configuration	impure	or	shared	
constant	in	others	signal	xnor
	inertial	out	sla	xor
	inout		sll	
	is	package	sra	

Рис. 3.3. Зарезервированные слова языка VHDL

Специальные символы содержат один или два знака:

« » # & ‘ ( ) * + — , . / : ; < = > [ ] /
=> ** := /= >= <= .

Числа записываются в VHDL-коде в виде целочисленных (Integer Literals) или вещественных литералов (Real Literals):

48 0 237 — десятичные целые литералы
2#10101101# — двоичный целый литерал
16#DE# — шестнадцатиричный целый литерал

S2E2	— целый литерал в экспоненциальной форме
3.1423	— вещественный литерал
31.3E-4	— вещественный литерал в экспоненциальной форме
838_123	— целый литерал со знаками подчеркивания для удобства чтения.

Символьным литералом (Character Literal) является любой печатный символ, заключенный в апострофы: ‘A‘ ‘b‘ ‘,‘ и т. д.

Строка (строчный литерал — String Literal) представляет собой последовательность символов, записанную в кавычках:

“computer”
“ZZZZ”
“”..

Примеры битовых строк:

B”01110010
B”0111_0010
b”0”

0”727	— эквивалент B “111_010_111 в восмеричной системе исчисления;
X”FF”	— эквивалент B “1111_1111” в шестнадцатиричной системе счисления.

Описание синтаксиса традиционно осуществляется с помощью формул Бэкуса—Наора (Extended Backus—Naur Form — EBNF). Мы не будем в тексте изложения пользоваться этой нотацией (формальное описание синтаксиса языка VHDL приведено в приложении).

Языки программирования предусматривают различные типы данных. Поскольку VHDL используется для описания аппаратуры в самых разных вариантах, средства типизации данных приобретают здесь очень важное значение. Например, они дают возможность представлять состояние памяти как массив битов, целое число или мнемонический код.

VHDL-язык со строгой типизацией. Это означает, что типы всех значений данных, используемых в программе, контролируются на согласованность их использования.

Введение типов — очень мощное средство выявления логических ошибок в написанной программе. Кроме того, оно обеспечивает значительную поддержку при последующем сопровождении программы.

Каждый элемент данных имеет конкретный тип, определяющий свойства тех значений, которыми этот элемент данных может обладать. Как и все языки со строгой типизацией, VHDL требует, чтобы программист указывал тип каждого встречающегося в программе элемента данных для возможности контроля его использования.

Компилятор VHDL гарантирует, что использование каждого элемента данных согласуется с набором его свойств, а когда программисту потребуется изменить программу, тип поможет ему понять назначение этого элемента данных. Непредумышленное смешение типов в одной операции будет восприниматься как ошибка. Средства строгого контроля типов позволяют уточнить намерения разработчика.

Тип (Type) характеризуется набором значений и набором операций, применяемых к этим значениям. Определением типа является языковая конструкция, с по-

мощью которой вводится тип. Подтип (SubType) является подмножеством значений данного типа. Подтипы конструируются посредством указания имени существующего типа и, когда это необходимо, дополнительных ограничений.

Классификация типов языка VHDL приведена на рис. 3.4.

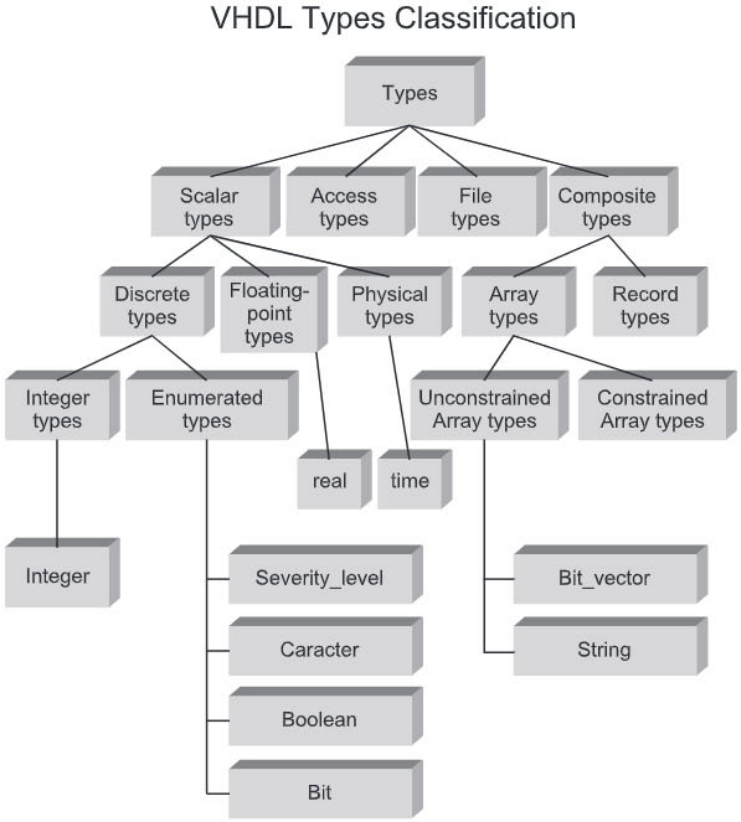


Рис. 3.4. Классификация типов языка VHDL

В языке VHDL три объекта данных:

- переменные (Variables),
- сигналы (Signals),
- константы (Constans).

Каждый объект имеет уникальное имя, и каждое имя обозначает уникальный объект. Каждое имя допустимо в определенной части программы (где о нем говорят, что оно видимо).

При появлении некоторого понятия в программе с ним связывается идентификатор. Этот идентификатор является локальным именем понятия и может использоваться везде, где это описание видимо (в операторах, следующих за описанием). Имена, не являющиеся простыми идентификаторами, образуются с помощью существительных имен, за которыми следуют индексы компонентов (для массивов), постфиксы (для компонентов записей) или квалификаторы (для предопределенных атрибутов) (см. далее).

Любые значения одного и того же скалярного типа можно сравнивать, при этом результатом станет значение истинности. Отношение записывается с помощью одной из операций отношения:

$$> < = /= >= <=$$

К значениям истинности могут быть применены логические операции not, and, or, xor в их обычном смысле.

При описании перечислимого типа его значения записываются в определенном порядке (см. далее), поэтому для любого определенного значения можно вычислить предыдущее или последующее значение, используя соответствующие атрибуты (см. п. 3.5).

Логические операции можно применять к массивам одинакового типа, если типом компонентов является boolean. Результатом, при этом, является другой массив того же типа, чье значение формируется с помощью последнего применения указанной операции к компонентам операндов.

Выражения образуются с помощью операндов и операций. Выражение может состоять из значения литерала или же из имени сигнала переменной или константы. В общем случае оно представляет собой один или несколько операндов, соединенных операциями.

Операция (Operator) является операцией типа, имеющей один или два операнда. В свою очередь, операция типа (Operation) представляет собой элементарное действие, связанное с одним или несколькими типами. Оно неявно выражается при описании этого типа, либо является подпрограммой, которая имеет параметр или результат этого типа.

Унарную операцию записывают перед операндом, бинарную — между двумя операндами.

Операция может быть описана как функция. Многие операции неявно описываются с помощью описания типа (например, большинство описаний типа подразумевает неявное описание операции сравнения на равенство для значений этого типа).

Ограничений на размер выражений нет, но выражения, размер которых превышает одну строку исходного текста, трудны для чтения и понимания.

Вычисление выражения заключается в вычислении операндов и применении соответствующих операций в порядке, определенном правилами приоритетов операций. Эти правила выбраны так, чтобы отобразить основную математическую практику и соглашения, принятые в других языках программирования.

Операнды, входящие в состав выражения, могут быть следующими:

- литералы, задающие значения скалярных объектов (к ним относятся числа или литералы перечисления);
- агрегаты, задающие значения составных объектов (записей и массивов);
- имена объектов (сигналов, переменных и констант), которые могут быть скалярными или составными;
- вызовы функций, в которых значения вычисляются посредством выполнения подпрограмм (значения могут быть скалярными или составными, а функция может иметь параметры, которые задаются как переменные, сигналы или выражения);
- квалифицированные выражения (атрибуты);
- подвыражения, т. е. выражения, заключенные в круглые скобки.

В выражениях могут использоваться следующие операции (рис. 3.5):

- операция возведения в степень (**);
- операция получения абсолютного значения (abs);
- операция умножения и деления (*, /, mod, rem);
- унарные (одноместные) операции (+, −, not);
- операции сложения и вычитания (+, −);
- операция конкатенации (&);
- операции сдвига (sll, srl, sla, sra, rol, ror);
- операции отношения (=, /=, <, <=, >, >=) и
- логические операции (and, or, nand, nor, xor, xnor).

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата операции
**	exponentiation	integer or floating-point	integer	same as left operand
abs	absolute value		numeric	same as operand
not	negation		bit, boolean or 1-D array of bit or boolean	same as operand
*	multiplication	integer or floating-point physical integer or floating-point	same as left operand integer or floating-point physical	same as operands same as left operand same as right operand
/	division	integer or floating-point physical physical	same as left operand integer or floating-point same as left operand	same as operands same as left operand universal integer
mod	modulo	integer	same as left operand	same as operands
rem	remainder	integer	same as left operand	same as operands
+	identity		numeric	same as operand

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата операции
-	negation		numeric	same as operand
+	addition	numeric	same as left operand	same as operands
-	subtraction	numeric	same as left operand	same as operands
&	concatenation	1-D array 1-D array element type of right operand element type of result	same as left operand element type of left operand 1-D array element type of result	same as operands same as left operand same as right operand 1-D array
sll	shift-left logical	1-D array of bit or boolean	integer	same as left operand
srl	shift-right logical			
sla	shift-left arithmetic			
sra	shift-right arithmetic			
rol	rotate left			
ror	rotate right			
=	equality	any except file	same as left operand	boolean
/=	inequality			
<	less then	scalar or 1-D array of any discrete type	same as left operand	boolean
<=	less then or equal			
>	greater than			
>=	greater than or equal			
and	logical and	bit, boolean or 1-D array of bit or boolean	same as left operand	same as operands

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата операции
or	logical or			
nand	negative logical and			
nor	negative logical or			
xor	exclusive or			
xnor	negated exclusive or			

Рис. 3.5. Операции языка VHDL

Отметим, что есть лишь три унарные операции, все остальные являются бинарными (двухместными). Операции «+», «−» могут быть унарными и бинарными, в зависимости от их расположения в выражениях. В соответствии с правилом строгой типизации, операции имеют вполне конкретные типы операндов, а результат применения операции также имеет известный тип.

Операция конкатенации выполняет «склеивание» (конкатенацию) двух или более одномерных массивов с совпадающим типом компонент (рис. 3.6).

Результатом операции является массив, содержащий определенные компоненты исходных массивов.

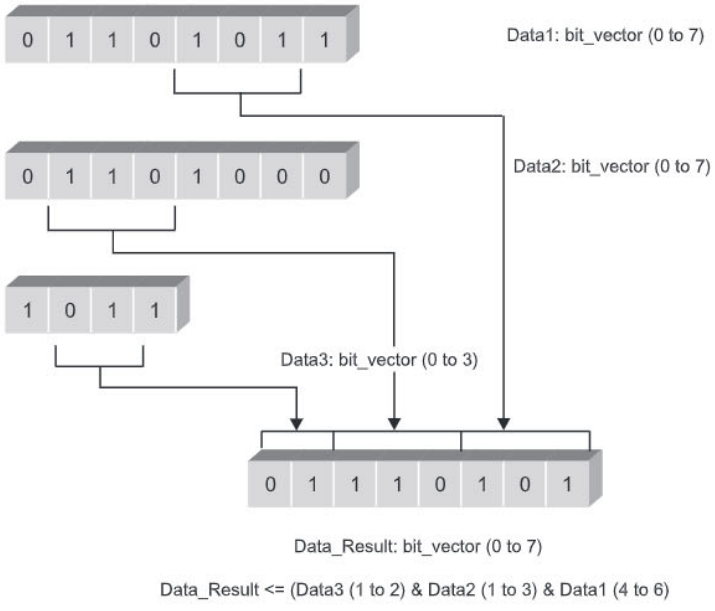


Рис. 3.6. Операция конкатенации (Concatenation)

Смысл операций сдвига поясняет рис. 3.7.

Выражение, как было уже сказано, состоит из одного или большего количества операндов, которым, возможно, предшествует унарная операция.

Таблица операций показывает приоритет операций (рис. 3.5): чем выше операция расположена в таблице, тем выше ее приоритет. При вычислении выражения применимы следующие правила:

- Для операнда, с обеих сторон окруженного операциями разного приоритета, в первую очередь, применяется операция, имеющая более высокий приоритет. Если операции имеют одинаковый приоритет, то первой применяется левая операция.
- Бинарная операция имеет операнды с обеих сторон. Оба операнда вычисляются (в любом порядке) до применения операции.
- Унарная операция применяется к операнду, стоящему справа, учитывая правило 1.

На рис. 3.8 приводятся примеры работы с объектами данных VHDL — сигналами, переменными и константами. Каждый объект данных имеет определенный тип, который указывается при описании объекта. В конкретной части программы объект данных вводится своим описанием, определяющим имя объекта, его тип или подтип и, возможно, его начальное значение.

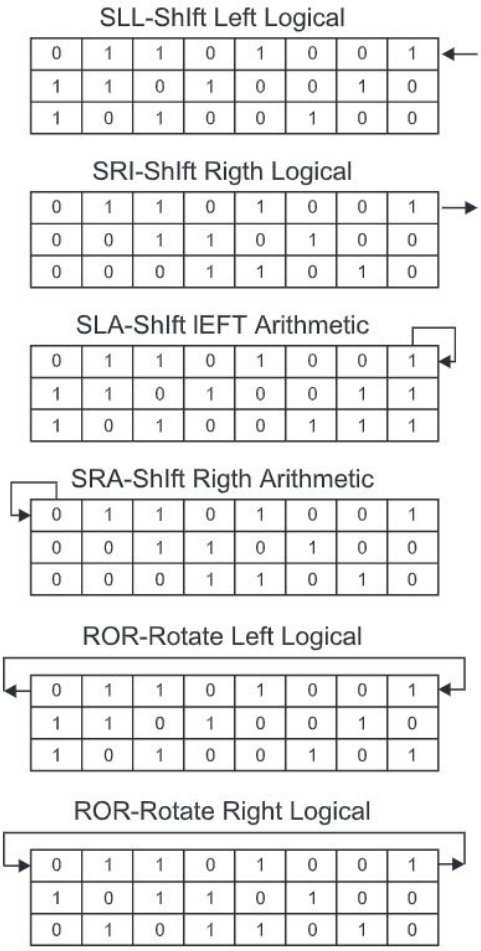


Рис. 3.7. Операции сдвига в языке VHDL

```
-------------------------------------------------------------
-- DATA OBJECTS
-------------------------------------------------------------
-- SIGNALS
-------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
PACKAGE SIGNAL_DECLARATION IS
  TYPE BUS_TYPE IS ARRAY(0 to 7) OF STD_LOGIC;
  SIGNAL vcc    : STD_LOGIC := '1';
  SIGNAL ground : STD_LOGIC := '0';
  FUNCTION transform_function( a : IN BUS_TYPE) RETURN BUS_TYPE;
END SIGNAL_DECLARATION;
USE WORK.SIGNAL_DECLARATION.ALL;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY MY_DESIGN is
  PORT( data_in  : IN  BUS_TYPE;
        data_out : OUT BUS_TYPE);
  SIGNAL sys_clk : STD_LOGIC := '1';
END MY_DESIGN;
ARCHITECTURE DATA_FLOW OF MY_DESIGN IS
  SIGNAL int_bus            : BUS_TYPE;
  CONSTANT disconnect_value : BUS_TYPE
           := ('X','X','X','X','X','X','X','X');

BEGIN
  int_bus <= data_in WHEN sys_clk = '1'
                     ELSE int_bus;
  data_out <= transform_function(int_bus) WHEN sys_clk = '0'
                                          ELSE disconnect_value;
  sys_clk <= NOT(sys_clk) after 50 ns;

END DATA_FLOW;
-----------------------------------------------------------------
-- CONSTANTS
-----------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY AND2 IS
  GENERIC(rise,fall : time; load : integer);
  PORT(A, B : IN  std_logic;
          C : OUT std_logic);
END AND2;
ARCHITECTURE test OF AND2 IS
  SIGNAL      tmp : std_logic;
  CONSTANT rdelay : time := rise + (load * 2 ns);
  CONSTANT fdelay : time := fall + (load * 1 ns);

BEGIN
  tmp <= A AND B;
  C <= tmp AFTER rdelay WHEN tmp = '1' ELSE
       tmp AFTER fdelay;
END test;
-----------------------------------------------------------------
-- VARIABLES
-----------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY AND3 IS
  PORT ( A, B, C : IN  std_logic;
         Q       : OUT std_logic);
END AND3;
ARCHITECTURE AND3 OF AND3 IS
```

```
BEGIN
  PROCESS(A, B, C)
    VARIABLE state : std_logic;
    VARIABLE delay : time;
  BEGIN
    state := A AND B AND C;
    IF state = '1' THEN
      delay := 2 ns;
    ELSIF state = '0' THEN
      delay := 1 ns;
    ELSE
      delay := 1.5 ns;
    END IF;
    q <= state AFTER delay;
  END PROCESS;
END AND3;
-------------------------------------------------------------------
```

Рис. 3.8. Примеры работы с обьектами данных VHDL

Приведем теперь ряд примеров использования типов языка VHDL (рис. 3.4).

Простейшие типы данных являются скалярными. Скалярный тип (Scalar Type) представляет собой либо дискретный тип (Discrete Type), либо вещественный тип (Real Type). Значения скалярного типа не имеют компонентов и всегда упорядочены.

Дискретными типами являются перечислимый тип (Enumerated Type) и целый тип (Integer Type). Дискретные типы используются для перечисления состояний сигналов и переменных, индексирования и управления повторением в операторах цикла, а также в выборах операторов выбора и вариантах записи.

К перечисленным типам относятся:

- битовый тип (Bit Type), включающий значения логической единицы и логического нуля;
- булевый тип (Boolean Type), включающий значения истинности (False, True);
- символьный тип (Character Type);
- тип, включающий уровни значимости собщений, генерируемых оператором проверки Assert Statement (Error, Failure, Warning, Note) — Severity-level Type).

Вещественный тип (Real Type) включает значения, которые представляют собой приближения вещественных чисел.

К скалярным типам относится еще один физический тип (Physical Type) (время), так необходимый при моделировании цифровых систем.

Составной тип данных (Composite Type) содержит значения, которые имеют компоненты. Существуют две разновидности составного типа:

- индексируемый тип (Array Type) и
- именуемый тип (Record Type).

Значения индексируемого типа (массива) состоят из компонентов одного и того же типа (или подтипа). Каждый компонент однозначно идентифицируется индексом (для одномерного массива) или последовательностью индексов (для многомерного массива). Каждый индекс должен быть значением дискретного типа (целого или перечислимого) и должен принадлежать требуемому диапазону

индексов. Если необходимо, чтобы компоненты массива были записями, то вначале описывается тип компонента. Тип компонента может быть ограничен либо с помощью промежуточного описания подтипа (Unconstrained Array Type), либо непосредственным указанием ограничения на тип компонента в определе-нии массива (Constrained Array Type). Ограничения задаются с помощью диапазона (Range) — упорядоченного набора значений скалярного типа с нижней и верхней границами этих значений.

Типы неограниченных массивов, именуемых строками (String) и битовыми векторами (Bit-vector), задаются следующим образом:

```
type string is array (positive range )
                of character;
type bit_vector is array (natural range)
                of bit;
```

Почти все полезные значения данных являются структурными. Обычно они содержат ряд различных компонент, каждая из которых имеет свой собственный тип. Это является причиной для введения именуемого типа (Record Type). Значение этого типа состоит из компонент, которые обычно бывают различных типов и подтипов. Для каждого компонента значения записи в определении именуемого типа задается идентификатор, который однозначно задает компонент в записи.

Компонент записи, в свою очередь, сам может быть записью. Записи могут быть компонентами массива, а компонентами записи — массивы.

Значение ссылочного типа (Access Type) — ссылочное значение — это либо пустое значение, либо значение, указывающее объект. Значение указываемого объекта может быть прочитано или изменено через ссылочное значение. Определение ссылочного типа задает тип объектов, на которые указывают значения ссылочного типа.

Файловый тип данных (File Type) служит для моделирования файловой подсистемы моделируемых цифровых подсистем.

Целый ряд примеров использования типов данных приведен на рис. 3.9—3.19.

```
------------------------------------------------------------
-- SUBTYPE EXAMPLE
------------------------------------------------------------
PACKAGE mux_types IS
  SUBTYPE eightval IS INTEGER RANGE 0 TO 7; -- line 1
END mux_types;
USE WORK.mux_types.ALL;
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY mux8 IS
  PORT(I0, I1, I2, I3, I4, I5,
       I6, I7: IN std_logic;
       sel : IN eightval; -- line 2
       q   : OUT std_logic);
END mux8;
ARCHITECTURE mux8 OF mux8 IS
BEGIN
  WITH sel SELECT   -- line 3
      Q <= I0 AFTER 10 ns WHEN 0, -- line 4
```

```
           I1 AFTER 10 ns WHEN 1, -- line 5
           I2 AFTER 10 ns WHEN 2, -- line 6
           I3 AFTER 10 ns WHEN 3, -- line 7
           I4 AFTER 10 ns WHEN 4, -- line 8
           I5 AFTER 10 ns WHEN 5, -- line 9
           I6 AFTER 10 ns WHEN 6, -- line 10
           I7 AFTER 10 ns WHEN 7; -- line 11
END mux8;
```

Рис. 3.9. Пример использования подтипа данных

```
-------------------------------------------------------------
-- INTEGER
-------------------------------------------------------------
ENTITY INTEGER_USAGE IS
  PORT (X : IN BIT);
END INTEGER_USAGE;
ARCHITECTURE ARCH_INTEGER_USAGE OF INTEGER_USAGE IS
  SUBTYPE INT_TYPE IS INTEGER;
BEGIN
  PROCESS(X)
    VARIABLE A : INTEGER;
    VARIABLE B : int_type;
  BEGIN
    A :=  1;   -- OK
    B := -1;   -- OK
--  A := 1.0;  -- ERROR
  END PROCESS;
END ARCH_INTEGER_USAGE;
-------------------------------------------------------------
```

Рис. 3.10. Пример использования целого типа INTEGER TYPE

```
-------------------------------------------------------------
-- ENUMERATED
-------------------------------------------------------------
PACKAGE INSTRUCTION IS
  TYPE instruction IS ( add, sub, lda, ldb, sta, stb, outa, xfr );
END INSTRUCTION;
USE WORK.INSTRUCTION.ALL;
ENTITY MICROPROCESSOR IS
  PORT (instr : IN    instruction;
        addr  : IN    INTEGER;
        data  : INOUT INTEGER);
END MICROPROCESSOR;
ARCHITECTURE ENUMERATED_USAGE OF MICROPROCESSOR IS
BEGIN
  PROCESS(instr)
    TYPE regtype IS ARRAY(0 TO 255) OF INTEGER;
    VARIABLE a, b : INTEGER;
    VARIABLE reg  : regtype;
  BEGIN
```

```
                          -- select instruction to execute
    CASE instr is
      WHEN lda =>
        a := data;        -- load a accumulator
      WHEN ldb =>
        b := data;        -- load b accumulator
      WHEN add =>
        a := a + b;       -- add accumulators
      WHEN sub =>
        a := a - b;       -- subtract accumulators
      WHEN sta =>
        reg(addr) := a;   -- put a accum in reg array
      WHEN stb =>
        reg(addr) := b;   -- put b accum in reg array
      WHEN outa =>
        data <= a;        -- output a accum

      WHEN xfr =>         -- transfer b to a
        a := b;
    END CASE;
  END PROCESS;
END ENUMERATED_USAGE;
-------------------------------------------------------------
```

Рис. 3.11. Пример использования перечислимого типа ENUMERATED TYPE

```
-------------------------------------------------------------
-- BIT,BIT_VECTOR
-------------------------------------------------------------
SIGNAL X: BIT;
  IF X = '1' THEN
    STATE <= IDLE;
  ELSE
    STATE <= START;
  END IF;

-------------------------------------------------------------
```

Рис. 3.12. Пример использования битового типа BIT TYPE

```
-------------------------------------------------------------

SIGNAL X: BOOLEAN;
  IF X = TRUE THEN
    STATE <= IDLE;
  ELSE
    STATE <= START;
  END IF;

-------------------------------------------------------------
```

Рис. 3.13. Пример использования булевого типа BOOLEAN TYPE

```
------------------------------------------------------------
-- DATA TYPES AND SUBTYPES
------------------------------------------------------------
-- STD_LOGIC,STD_ULOGIC,STD_LOGIC_VECTOR,STD_ULOGIC_VECTOR
------------------------------------------------------------
-- VALUES ARE: 'U'- UNINITIALIZED
--             'X'- FORCING UNKNOWN
--             '0'- FORCING 0
--             '1'- FORCING 1
--             'Z'- HIGH IMPEDANCE
--             'W'- WEAK UNKNOWN
--             'L'- WEAK 0
--             'H'- WEAK 1
--             '-'- DON'T CARE
------------------------------------------------------------
-- STD_LOGIC
------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
SIGNAL DATA_OUT,DATA_IN,ENABLE: STD_LOGIC;
   DATA_OUT <= DATA_IN WHEN ENABLE = '1' ELSE 'Z';

------------------------------------------------------------
-- STD_ULOGIC
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
SIGNAL DATA_OUT,DATA_IN,ENABLE: STD_ULOGIC;
   DATA_OUT <= DATA_IN WHEN ENABLE = '1' ELSE 'Z';
------------------------------------------------------------
-- STD_LOGIC_VECTOR,STD_ULOGIC_VECTOR
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
SIGNAL MUX: STD_LOGIC_VECTOR(15 DOWNTO 0);
IF STATE = ADDRESS THEN
  MUX <= RAM_ADDRESS;
ELSE
  MUX <= (OTHERS => 'Z');
END IF;
```

Рис. 3.14. Пример использования типов данных для моделирования многозначной логики

```
------------------------------------------------------------
-- REAL
------------------------------------------------------------
ENTITY REAL_USAGE IS
  PORT (X : IN BIT);
END REAL_USAGE;
ARCHITECTURE ARCH_REAL_USAGE OF REAL_USAGE IS
  SUBTYPE REAL_TYPE IS REAL;
  SIGNAL  C : REAL;
```

```
BEGIN
  PROCESS(X)
    VARIABLE A : REAL;
    VARIABLE B : REAL_TYPE;
  BEGIN
    A := 2.5E-17;    -- OK
    C <= -5.3E3;     -- OK
--  B := -1;         -- ERROR
--  A := 2 NS;       -- ERROR
  END PROCESS;
END ARCH_REAL_USAGE;
-------------------------------------------------------------
```

Рис. 3.15. Пример использования вещественного типа REAL TYPE

```
-------------------------------------------------------------
-- PHYSICAL
-------------------------------------------------------------
-- TYPE TIME IS RANGE <IMPLEMENTATION DEFINED>
-- UNITS
--   fs;
--   ps  = 1000 fs;
--   ns  = 1000 fs;
--   us  = 1000 fs;
--   ms  = 1000 fs;
--   sec = 1000 fs;
--   min = 60 sec;
--   hr  = 60 min;
-- END UNITS;
-- THIS TYPE IS DEFINED IN THE STANDARD PACKAGE
PACKAGE PHYSICAL_USAGE IS
  TYPE current IS RANGE 0 TO 1000000000
    UNITS
      na;            -- nano amps
      ua = 1000 na;  -- micro amps
      ma = 1000 ua;  -- milli amps
      a  = 1000 ma;  -- amps
    END UNITS;

  TYPE load_factor IS (small, med, big );

END PHYSICAL_USAGE;
USE WORK.PHYSICAL_USAGE.ALL;
ENTITY delay_calc IS
  PORT ( out_current : OUT current;
         load        : IN  load_factor;
         delay       : OUT time);
END delay_calc;
ARCHITECTURE delay_calc OF delay_calc IS
BEGIN
  delay       <= 10 ns WHEN (load = small) ELSE
```

```
                20 ns WHEN (load = med) ELSE
                30 ns WHEN (load = big) ELSE
                10 ns;
  out_current <= 100 ua WHEN (load = small)ELSE
                1 ma   WHEN (load = med) ELSE
                10 ma  WHEN (load = big) ELSE
                100 ua;
END delay_calc;
-------------------------------------------------------------
-- PHYSICAL
-------------------------------------------------------------
```

Рис. 3.16. Пример использования физического типа PHYSICAL TYPE

```
-------------------------------------------------------------
-- COMPOSITE TYPE -- EXAMPLE 1
-------------------------------------------------------------
PACKAGE array_example IS
  TYPE data_bus IS ARRAY(0 TO 31) OF BIT;
  TYPE small_bus IS ARRAY(0 TO 7) OF BIT;
END array_example;
USE WORK.array_example.ALL;
ENTITY extract IS
  PORT (data : IN data_bus;
        start : IN INTEGER;
        data_out : OUT small_bus);
END extract;
ARCHITECTURE test OF extract IS
BEGIN
  PROCESS(data, start)
  BEGIN
    FOR i IN 0 TO 7 LOOP
      data_out(i) <= data(i + start);
    END LOOP;
  END PROCESS;
END test;
-------------------------------------------------------------
-- COMPOSITE TYPE -- EXAMPLE 2
-------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
PACKAGE memory IS
  CONSTANT width   : INTEGER := 3;
  CONSTANT memsize : INTEGER := 7;
  TYPE data_out IS ARRAY(0 TO width) OF std_logic;
  TYPE mem_data IS ARRAY(0 TO memsize) OF data_out;
END memory;
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE WORK.memory.ALL;
ENTITY rom IS
```

```
  PORT( addr : IN INTEGER;
        data : OUT data_out;
        cs   : IN std_logic);
END rom;
ARCHITECTURE basic OF rom IS
  CONSTANT z_state : data_out := ('Z', 'Z', 'Z', 'Z');
  CONSTANT x_state : data_out := ('X', 'X', 'X', 'X');
  CONSTANT rom_data : mem_data :=
    ( ( '0', '0', '0', '0'),
    ( '0', '0', '0', '1'),
    ( '0', '0', '1', '0'),
    ( '0', '0', '1', '1'),
    ( '0', '1', '0', '0'),
    ( '0', '1', '0', '1'),
    ( '0', '1', '1', '0'),
    ( '0', '1', '1', '1') );
BEGIN
  ASSERT addr <= memsize
    REPORT "addr out of range"
    SEVERITY ERROR;
  data <= rom_data(addr) AFTER 10 ns WHEN cs = '1' ELSE
          z_state AFTER 20 ns WHEN cs = '0' ELSE
          x_state AFTER 10 ns;
END basic;
-------------------------------------------------------------
-- COMPOSITE TYPE -- EXAMPLE 3
-------------------------------------------------------------
TYPE optype IS ( add, sub, mpy, div, jmp );
TYPE instruction IS
  RECORD
    opcode : optype;
    src    : INTEGER;
    dst    : INTEGER;
  END RECORD;
-------------------------------------------------------------
-- COMPOSITE TYPE -- EXAMPLE 4
-------------------------------------------------------------
architecture behave of test8 is
  signal x : bit;
begin
PROCESS(X)
  VARIABLE inst : instruction;
  VARIABLE source, dest : INTEGER;
  VARIABLE operator : optype;
BEGIN
  source := inst.src;        -- Ok line 1
  dest   := inst.src;        -- Ok line 2
--  source := inst.opcode;   -- error line 3
  operator := inst.opcode;   -- Ok line 4
  inst.src := dest;          -- Ok line 5
```

```
  inst.dst := dest;          -- Ok line 6
  inst := (add, dest, 2);    -- Ok line 7
--  inst := (source);        -- error line 8
END PROCESS;
-------------------------------------------------------------
end behave;
```

Рис. 3.17. Пример использования составных типов COMPOSITE TYPES

```
-------------------------------------------------------------
--- ACCESS TYPE
-------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use work.test9.all;
entity test10 is
end test10;
architecture behave of test10 is
  signal x : bit;
begin
PROCESS(X)
  VARIABLE packet : data_packet;
BEGIN
  packet.addr.key := 5;        -- Ok line 1
  packet.addr := (10, 20);     -- Ok line 2
  packet.data(0) := ('0', '0', '0', '0');  -- Ok line 3
 -- packet.data(10)(4) := '1';   -- error line 4
  packet.data(10)(0) := '1';   -- Ok line 5
END PROCESS;
PROCESS(X)
  TYPE fifo_element_t IS ARRAY(0 TO 3)
    OF std_logic; -- line 1
  TYPE fifo_el_access IS
    ACCESS fifo_element_t; -- line 2
  VARIABLE fifo_ptr : fifo_el_access := NULL; -- line 3
  VARIABLE temp_ptr : fifo_el_access := NULL; -- line 4
BEGIN
  temp_ptr := new fifo_element_t; -- Ok line 5
  temp_ptr.ALL := ('0', '1', '0', '1');-- Ok line 6
  temp_ptr.ALL := ('0', '0', '0', '0');--Ok line 7
  temp_ptr.ALL(0) := '0';         -- Ok line 8
  fifo_ptr := temp_ptr;           -- Ok line 9
  fifo_ptr.ALL := temp_ptr.ALL;   -- Ok line 10
END PROCESS;
end behave;
```

Рис. 3.18. Пример использования ссылочного типа ACCESS TYPE

```
-------------------------------------------------------
-- FILE TYPE EXAMPLE
-------------------------------------------------------
LIBRARY IEEE;
```

```
USE IEEE.std_logic_1164.ALL;
ENTITY rom IS
  PORT(addr : IN INTEGER;
       cs   : IN std_logic;
       data : OUT INTEGER);
END rom;
ARCHITECTURE rom OF rom IS
BEGIN
  PROCESS(addr, cs)
    VARIABLE rom_init : BOOLEAN := FALSE;     -- line 1
    TYPE rom_data_file_t IS FILE OF INTEGER; -- line 2
    FILE rom_data_file : rom_data_file_t IS IN
        "/doug/dlp/test1.dat";                 --line 3
    TYPE dtype IS ARRAY(0 TO 63) OF INTEGER;
    VARIABLE rom_data : dtype; -- line 4
    VARIABLE i : INTEGER := 0; -- line 5
  BEGIN
    IF (rom_init = false) THEN   -- line 6
      WHILE NOT ENDFILE(rom_data_file) -- line 7
                AND (i < 64) LOOP
        READ(rom_data_file, rom_data(i)); -- line 8
        i := i + 1;           -- line 9
      END LOOP;
      rom_init := true;       -- line 10
    END IF;
    IF (cs = '1') THEN        -- line 11
      data <= rom_data(addr); -- line 12
    ELSE
      data <= -1;             -- line 13
    END IF;
  END PROCESS;
END rom;
```

Рис. 3.19. Пример использования файлового типа FILE TYPE

Каждый оператор языка VHDL определяет действие, которое должно быть выполнено. Последовательность операторов определяет набор действий, которые должны быть выполнены один за другим в порядке написания операторов (для последовательных операторов — Sequential Statements) или в порядке их активизации (Concurrent (Parallel) Statements).

Операторы делятся на простые и составные. Составные содержат вложенные последовательности операторов (которые также могут быть простыми или составными). Некоторые операторы (например, операторы процессов, операторы генерации) могут быть помечены и таким образом получить имя.

Если составной оператор содержит другой составной оператор, то говорят, что этот другой вложен. Составные операторы заключаются в «скобки» — ключевые слова, обозначающие начало и конец внутренней последовательности операторов составного оператора.

Составные операторы включают в себя либо критерии, по которым выбираются для выполнения последовательности операторов (условный оператор, оператор

выбора), либо условия повторения оператора (оператор цикла), либо охранные выражения (оператор блока).

Новые значения сообщаются переменным или сигналам (объектам данных, значения которых могут быть изменены) с помощью операторов присваивания (Assignment Statements). Первоначальное значение при этом полностью теряется. Переменная или сигнал могут быть любого типа (включая запись или массив). Оператор присваивания определяет слева переменную или сигнал, а справа — выражение, значение которого обязано быть новым значением этой переменной или этого сигнала. Это новое значение должно быть того же типа и удовлетворять всем ограничениям.

В VHDL-программе требуется варьирование выполняемых действий в зависимости от ситуации. Каждая из альтернатив представляет собой последовательность операторов, подключаемых в зависимости от условия.

Есть два основных метода выбора одного из альтернативных действий: по условию и по значению переменной или сигнала. Соответственно этому имеются:

- условный оператор для проверки условий (If-Then-Else Statement);
- оператор выбора по значению некоторой величины (Case-When Statement, With-Select Statement).

Выбор по условию:

При этом, в условном операторе задается условие, которое должно проверяться и служить основой для выбора последовательности операторов после ключевого слова then для выполнения в случае истинности условия. Если условие ложно, может выполняться другая последовательность операторов, которые могут включать проверку других условий.

Таким образом проверяется набор условий для поочередного выбора соответствующих операторов для исполнения при нахождении первого истинного условия и конечных операторов для исполнения, если все условия ложны. В общем случае, условный оператор состоит из набора условий для разбора их в заданном порядке. Первое истинное условие определяет, какая именно последовательность операторов должна быть выполнена. Если ни одно условие не является истинным, может быть выполнена последняя последовательность операторов.

Ключевые слова, используемые для выделения этих частей оператора, if — в начале, перед первым условием, elsif — перед каждым последующим условием (которых может быть несколько или одно). За каждым условием следует ключевое слово then и соответствующая последовательность операторов. Весь оператор заканчивается всегда ключевым словом end if.

Выбор по значению:

В операторе выбора по значению задаются выражения дискретного типа (целого или перечислимого) и их ожидаемые значения вместе с последовательностями операторов. Последовательность операторов выполняется в том случае, если выражение принимает связанное с данной последовательностью значение. Для каждой альтернативы выбора может быть назначено несколько значений, но значения для разных альтернатив выбора должны отличаться.

Повторение:

Некоторые операторы могут выполняться повторно. Единица повторения записывается последовательностью операторов и образует тело цикла, которое может начинаться условием прекращения повторений.

Простой цикл loop...end loop объединяет операторы для повторения без каких-либо условий завершения.

Управление повторением может осуществляться как перебором, так и проверкой логического условия. Способ управления задается в спецификации повторения перед циклом. Независимо от этого, во время выполнения цикла некоторое условие может привести к выходу из цикла.

Управление по перебору задается после ключевого слова for.

Управление по условию задается после ключевого слова while. Условие проверяется перед началом каждого повторения цикла; при этом может оказаться, что тело цикла не должно выполняться вообще.

Примеры использования последовательных операторов VHDL приведены в п. 3.3, а параллельных операторов VHDL — в п. 3.4.

3.3. Последовательные операторы VHDL

Как видно из рис. 3.20, *а*, к последовательным операторам языка VHDL относятся следующие операторы:

- Variable Assignment Statement,
- If-Then-Else, Case-When, For-Loop, While-Loop Statements,
- Subprogram Call Statement.

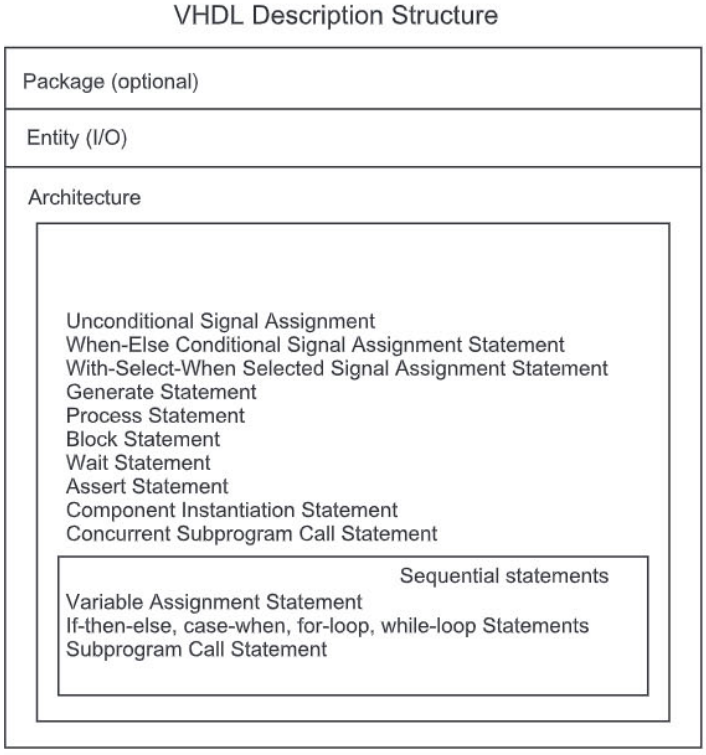


Рис. 3.20,а. Структура VHDL-описания

Примеры использования данных операторов приведены на рис. 3.20.

```
-------------------------------------------------
-- SEQUENTIAL STATEMENTS
-------------------------------------------------
-- PROCESS STATEMENT
-------------------------------------------------
EX_PROCESS:
PROCESS(RESET,CLOCK)
  CONSTANT NULLS : STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
BEGIN
  WAIT UNTIL CLOCK = '1';
    IF (RESET = '1') THEN
      Q <= NULLS;
    ELSIF (EN = '1') THEN
      Q <= DATA;
    ELSE
      Q <= Q;
    END IF;
END EX_PROCESS;
-------------------------------------------------
-- IF-THEN-ELSE STATEMENT
-------------------------------------------------
    IF (RESET = '1') THEN
      Q <= NULLS;
    ELSIF (EN = '1') THEN
      Q <= DATA;
    ELSE
      Q <= Q;
    END IF;

-------------------------------------------------
-- CASE-WHEN STATEMENT
-------------------------------------------------

CASE INPUT IS
  WHEN "0000" =>
    OUT1 <= DATA1;
  WHEN "0010" =>
    OUT1 <= DATA2;
  WHEN OTHERS =>
    OUT1 <= DATA1;
END CASE;
-------------------------------------------------
-- FOR-LOOP STATEMENT
-------------------------------------------------
EX_FOR_LOOP:
FOR I IN 7 DOWN TO 0 LOOP
  IF DATA(I) = '1' THEN
     OUT1(I):= '0';
```

```
  END IF;
END LOOP EX_FOR_LOOP;
-------------------------------------------------
-- WHILE-LOOP STATEMENT
-------------------------------------------------
COUNTER := 8;
EX_WHILE_LOOP:
WHILE (COUNTER>0) LOOP
  DATA_RESULT <= DATA_RESULT + DATA_IN;
  COUNTER := COUNTER - 1;
END LOOP EX_WHILE_LOOP;

-------------------------------------------------
```

Рис. 3.20. Примеры использования последовательных операторов VHDL

3.4. Параллельные операторы VHDL

Согласно тому же рис. 3.20, *а*, к последовательным операторам языка VHDL относятся следующие операторы:

- Unconditional Signal Assignment,
- When-Else Conditional Signal Assignment Statement,
- With-Select-When Selected Signal Assignment Statement,
- Generate Statement,
- Process Statement,
- Block Statement,
- Wait Statement,
- Assert Statement,
- Component Instantiation Statement,
- Concurrent Subprogram Call Statement.

Примеры использования параллельных операторов приведены на рис. 3.21.

```
-------------------------------------------------
-- CONCURRENT STATEMENTS
-------------------------------------------------
-- UNCONDITIONAL SIGNAL ASSIGNMENT
-------------------------------------------------
Y  <=(A AND B) OR C;
Y1 <= A XOR B XOR C;
Y2 <= A OR ((NOT B) AND C);
Y3 <= A AND  B AND  C;
Y4 <= A NAND B NAND C;
Y5 <= A NOR  B NOR  C;
-------------------------------------------------
-- WHEN-ELSE CONDITIONAL SIGNAL ASSIGNMENT
-------------------------------------------------
X  <= '1' WHEN A = 1 ELSE '0';
Y  <=  R  WHEN STATE = IDLE ELSE
       S WHEN STATE = STATE0 ELSE
```

```
           T WHEN STATE = STATE1 ELSE
           U WHEN OTHERS;

-------------------------------------------------
-- WITH-SELECT-WHEN SELECTED SIGNAL ASSIGNMENT
-------------------------------------------------
ARCHITECTURE ARCH_FSM OF FSM IS
  TYPE STATE_TYPE IS (STATE0,STATE1,STATE2,STATE3,STATE4)
  SIGNAL STATE: STATE_TYPE;
  SIGNAL A,B  : STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN
  WITH STATE SELECT
         OUT1 <= "00010010" WHEN STATE2|STATE3;
                     A      WHEN STATE1;
                     B      WHEN OTHERS;

END ARCH_FSM;

-------------------------------------------------
-- GENERATE SCHEME FOR COMPONENT INSTANTIATION
-- OR EQUATION
-------------------------------------------------
GEN_LABEL1:  FOR I IN 0 TO 15 GENERATE
             REG16: REGISTER_16
                    PORT MAP(CLOCK,RESET,ENABLE,
                             DATA_IN(I),DATA_OUT(I));
             END GENERATE GEN_LABEL1;

GEN_LABEL2:  FOR J IN 0 TO 15 GENERATE
               Z(J) <= X(J) AND Y(J);
             END GENERATE GEN_LABEL2;
-------------------------------------------------

-- PROCESS STATEMENT
-------------------------------------------------
EX_PROCESS:
PROCESS(RESET,CLOCK)
  CONSTANT NULLS : STD_LOGIC_VECTOR(3 DOWNTO 0):= "0000";
BEGIN
  WAIT UNTIL CLOCK = '1';
    IF (RESET = '1') THEN
      Q <= NULLS;
    ELSIF (EN = '1') THEN
      Q <= DATA;
    ELSE
      Q <= Q;
    END IF;
END PROCESS;
```

```
-------------------------------------------------
-- BLOCK STATEMENT
-------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY cpu IS
  PORT( clock : IN std_logic;
        addr  : OUT std_logic_vector(0 to 3);
        data  : INOUT std_logic_vector(0 to 3);
        interrupt : IN std_logic;
        reset : IN std_logic);
END cpu;
ARCHITECTURE fragment OF cpu IS
  COMPONENT int_reg
    PORT( data : IN std_logic;
          regclock : IN std_logic;
          data_out : OUT std_logic);
  END COMPONENT;
  COMPONENT alu
    PORT( a, b : IN std_logic;
          c, carry : OUT std_logic);
  END COMPONENT;
  SIGNAL a, c, carry : std_logic_vector(0 TO 3);
BEGIN
  reg_array : BLOCK
  BEGIN
    R1 : int_reg
      PORT MAP( data(0), clock, data(0));
    R2 : int_reg
      PORT MAP( data(1), clock, data(1));
    R3 : int_reg
      PORT MAP( data(2), clock, data(2));
    R4 : int_reg
      PORT MAP( data(3), clock, data(3));
  END BLOCK reg_array;
  shifter : BLOCK
  BEGIN
    A1 : alu
      PORT MAP( a(0), data(0), c(0), carry(0));
    A2 : alu
      PORT MAP( a(1), data(1), c(1), carry(1));
    A3 : alu
      PORT MAP( a(2), data(2), c(2), carry(2));
    A4 : alu
      PORT MAP( a(3), data(3), c(3), carry(3));
    shift_reg : BLOCK
    BEGIN
      R1 : int_reg
        PORT MAP( a(0), clock, a(1));
    END BLOCK shift_reg;
```

```
  END BLOCK shifter;
END fragment;
-------------------------------------------------
-- WAIT STATEMENT
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity test is
  port(clk, trigger1, trigger2, trigger3 : in std_logic);
end test;
-------------------------------------------------------
ARCHITECTURE test OF test IS
  TYPE t_int is (int1, int2, int3, int4, int5 );
  SIGNAL int, intsig1, intsig2, intsig3 : t_int;
  SIGNAL lock_out : BOOLEAN;
BEGIN
  int1_proc: PROCESS
  BEGIN
    --
    --
    --
    WAIT ON trigger1;  -- outside trigger signal
    WAIT UNTIL clk = '1';
      IF NOT(lock_out) THEN
        intsig1 <= int1;
      END IF;
  END PROCESS;
  int2_proc: PROCESS
  BEGIN
    --
    --
    --
    WAIT ON trigger2;  -- outside trigger signal
    WAIT UNTIL clk = '1';
      IF NOT(lock_out) THEN
        intsig2 <=int2;
      END IF;
  END PROCESS;
  int3_proc: PROCESS
  BEGIN
    --
    --
    --
    WAIT ON trigger3;-- outside trigger signal
    WAIT UNTIL clk = '1';
      IF NOT(lock_out) THEN
        intsig3 <=int3;
      END IF;
  END PROCESS;
  int <=intsig1 WHEN NOT(intsig1'QUIET) ELSE
```

```
        intsig2 WHEN NOT(intsig2'QUIET) ELSE
        intsig3 WHEN NOT(intsig3'QUIET) ELSE
        int;
  int_handle : PROCESS
  BEGIN
    WAIT ON int'TRANSACTION;-- described next
    lock_out <= TRUE;
    WAIT FOR 10 ns;
    CASE int IS
      WHEN int1 =>
        --
        --
      WHEN int2 =>
        --
        --
      WHEN int3 =>
        --
        --
      WHEN int4 =>
        --
        --
      WHEN int5 =>
        --
        --
    END CASE;
    lock_out <= false;
  END PROCESS;
END test;
-------------------------------------------------
-- ASSERT STATEMENT
-------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff IS
  PORT( d, clk : IN std_logic;
        q : OUT std_logic);
END dff;
ARCHITECTURE dff OF dff IS
BEGIN
  PROCESS(clk)
  BEGIN
    IF ( clk = '1') AND ( clk'EVENT ) THEN
      q <= d;
    END IF;
  END PROCESS;
END dff;
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY dff IS
  GENERIC ( setup_time, hold_time : TIME );
```

```
  PORT( d, clk : IN std_logic;
        q : OUT std_logic);
BEGIN
  setup_check : PROCESS ( clk )
  BEGIN
    IF ( clk = '1' ) and ( clk'EVENT ) THEN
      ASSERT ( d'LAST_EVENT >= setup_time )
        REPORT "setup violation"
        SEVERITY ERROR;
    END IF;
  END PROCESS;
END dff;
ARCHITECTURE dff_behave OF dff IS
BEGIN
  dff_process : PROCESS ( clk )
  BEGIN
    IF ( clk = '1' ) AND ( clk'EVENT ) THEN
      q <= d;
    END IF;
  END PROCESS;
END dff_behave;
-------------------------------------------------
-- COMPONENT INSTANTIATION STATEMENT
-------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY shift IS
  PORT( a, clk : IN std_logic;
        b :  OUT std_logic);
END shift;
ARCHITECTURE gen_shift OF shift IS
  COMPONENT dff
    PORT( d, clk : IN std_logic;
          q : OUT std_logic);
  END COMPONENT;

  SIGNAL z : std_logic_vector( 0 TO 4 );
BEGIN
  z(0) <= a;

  g1 : FOR i IN 0 TO 3 GENERATE
    dffx : dff PORT MAP( z(i), clk, z(i + 1));
  END GENERATE;

  b <= z(4);
END gen_shift;
ARCHITECTURE long_way_shift OF shift IS
  COMPONENT dff
    PORT( d, clk : IN  std_logic;
```

```
          q      : OUT std_logic);
  END COMPONENT;

  SIGNAL z : std_logic_vector( 0 TO 4 );
BEGIN
  z(0) <= a;

  dff1: dff PORT MAP( z(0), clk, z(1) );
  dff2: dff PORT MAP( z(1), clk, z(2) );
  dff3: dff PORT MAP( z(2), clk, z(3) );
  dff4: dff PORT MAP( z(3), clk, z(4) );

  b <= z(4);
END long_way_shift;
-------------------------------------------------
-- CONCURRENT SUBPROGRAM CALL STATEMENT
-------------------------------------------------
ENTITY async_state_machine IS
  PORT(X     : IN    bit;
       Y1,Y2 : INOUT bit);
END async_state_machine;
USE INTVAL.ALL;
ARCHITECTURE rom_model OF async_state_machine IS
  TYPE TRUE_TABLE IS ARRAY(0 TO 7,0 TO 1) OF bit;
  VARIABLE mem : TRUE_TABLE:=(('0','0'),('1','1'),('0','0'),('1','1')
                          ('0','1'),('0','1'),('1','0'),('1','0'));
BEGIN
  PROCESS(X,Y1,Y2)
  BEGIN
     Y1 <= MEM(INTVAL(X&Y1&Y2),0) AFTER PROPAGATE_DELAY;
     Y2 <= MEM(INTVAL(X&Y1&Y2),1) AFTER PROPAGATE_DELAY;
  END PROCESS;
END rom_model;
-------------------------------------------------
```

Рис. 3.21. Примеры использования параллельных операторов VHDL

3.5. Модели задержек на VHDL

В VHDL рассматриваются задержки электронных компонентов двух видов:

- инерционные задержки (Inertial Delay);
- транспортные задержки (Transport Delay).

По умолчанию (в отсутствие ключевого слова Transport) перед нами всегда инерционная задержка. Такова, например, задержка буфера со входом a и выходом b архитектурном теле buf (рис. 3.22). Величина задержки, как мы уже знаем, задается с помощью ключевого слова after.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```
ENTITY buf IS
  PORT ( a : IN std_logic   ;
         b : OUT std_logic );
END buf;
ARCHITECTURE buf OF buf IS
BEGIN
  b <= a AFTER 10 ns;
END buf;
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY delay_line IS
  PORT ( a : IN std_logic   ;
         b : OUT std_logic );
END delay_line;
ARCHITECTURE delay_line OF delay_line IS
BEGIN
  b <= TRANSPORT a AFTER 10 ns;
END delay_line;
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY delay_line_tb IS
---------------------------
END delay_line_tb;
ARCHITECTURE delay_line_tb OF delay_line_tb IS
        COMPONENT delay_line
             PORT (a : IN std_logic;
                   b : OUT std_logic);
        END COMPONENT;
        SIGNAL  sa     : std_logic := '0';
        SIGNAL  sb     : std_logic       ;
BEGIN
  dut : delay_line
        PORT MAP(
             a  => sa,
             b  => sb);
  sa <= '1' AFTER  5 ns,
        '0' AFTER  8 ns,
        '1' AFTER 10 ns,
        '0' AFTER 25 ns,
        '1' AFTER 28 ns,
        '0' AFTER 30 ns,
        '1' AFTER 45 ns,
        '0' AFTER 48 ns;
END delay_line_tb;
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY buf_tb IS
```

```
---------------------------
END buf_tb;
ARCHITECTURE buf_tb OF buf_tb IS
        COMPONENT buf
             PORT (a : IN std_logic;
                   b : OUT std_logic);
        END COMPONENT;
        SIGNAL  sa     : std_logic := '0';
        SIGNAL  sb     : std_logic       ;
BEGIN
  dut : buf
        PORT MAP(
             a  => sa,
             b  => sb);
  sa <= '1' AFTER  5 ns,
        '0' AFTER  8 ns,
        '1' AFTER 10 ns,
        '0' AFTER 25 ns,
        '1' AFTER 28 ns,
        '0' AFTER 30 ns,
        '1' AFTER 45 ns,
        '0' AFTER 48 ns;
END buf_tb;
-------------------------------------------------------
```

Рис. 3.22. Примеры использования инерциальных и транспортных задержек в VHDL

В случае инерционной задержки электронный компонент работает как фильтр: на выход проходят импульсы, чья длительность превышает величину инерционной задержки. В противном случае импульс «съедается». Этот моделируемый случай близок к реальному поведению цифровой схемы, поэтому на последних стадиях проектирования применяется именно этот вид задержек.

У буфера (внутри архитектурного тела delay-line) с входом a и выходом b налицо транспортная задержка, о чем свидетельствует ключевое слово transport. При этом на выход проходят импульсы любой длительности. Это идеальная ситуация, далекая от реальной действительности. Однако на первых стадиях проектирования использование этого вида задержек удобно для моделирования функционирования схем без учета таких временных эффектов, как фильтрация сигналов.

Моделирование эффектов от применения обоих видов задержек осуществляется с помощью испытательных программ buf_tb и delay_line_tb соответственно.

Результат применения инерционной задержки приведен на рис. 3.23, *а* результат применения транспортной задержки — на рис. 3.24.

В случае, если мы не задаем задержки электронных компонентов с помощью ключевого компонента after, VHDL-симулятор использует в своих вычислениях механизм дельта-задержки.

Покажем, что использование механизма является инструментом предотвращения неоднозначности вычислений реакции электронных компонентов в узлах цифровой схемы [58 a].

Пусть у нас имеется цепь, приведенная на рис. 3.25. У цепи три входа (A, E и Clock), один выход (F), четыре электронных компонента (вентили NOT, NAND,

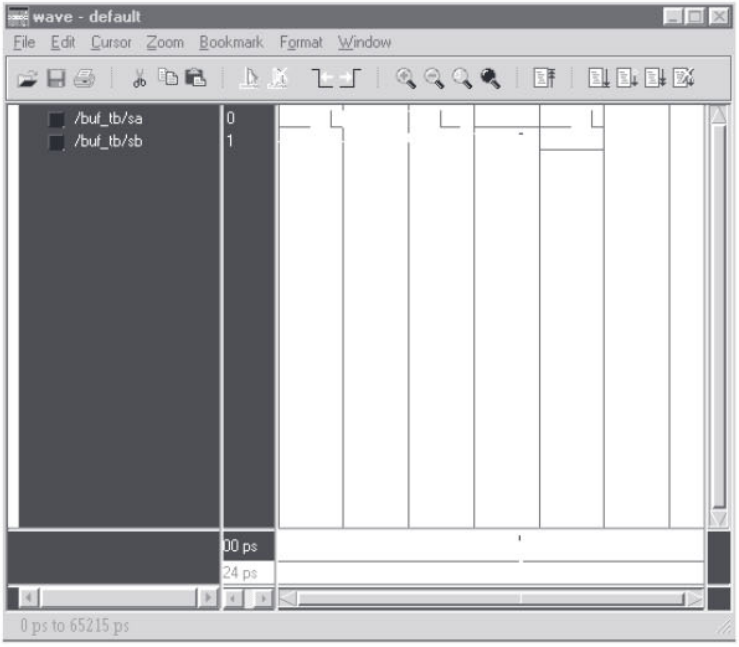


Рис. 3.23. Результаты моделирования задержек (инерционная задержка)

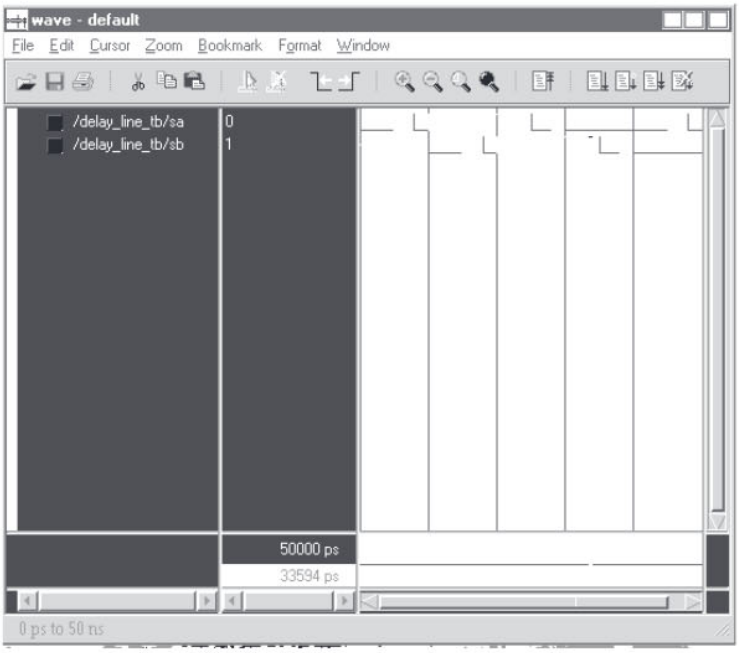


Рис. 3.24. Результаты моделирования задержек (транспортная задержка)

AND и D-триггер) и три внутренних узла (B, C, D). Положим, нам необходимо найти реакцию схемы в узле D, при условии, что на входе A происходит переход значения сигнала из 1 в 0 в момент t = 5 ns, а на входе Clock — постоянный уровень логической единицы.

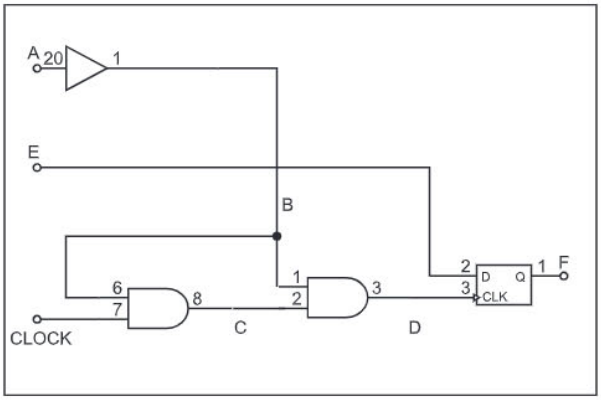


Рис. 3.25,а. Цепь для моделирования механизма дельта-задержки

Вначале вычисляется реакция вентиля AND	**Вначале вычисляется реакция вентиля NAND**
Реакция инвертора B <= 1	Реакция инвертора B <= 1
Реакция вентиля AND (C=1) D <= 1	Реакция вентиля NAND C <= 0
Реакция вентиля NAND C <= 0	Реакция вентиля AND D <= 0
Реакция вентиля AND (C=0) D <= 0	

Рис. 3.25,б. Вычисление реакции цепи в узле D по двум разным маршрутам в отсутствии механизма дельта-задержки

Время моделирования	**Дельта-задержка**	**Вычисление реакции цепи в узле D с помощью механизма дельта-задержки**
5 ns	1*delta	Реакция инвертора (A=0) B <= 1
	2*delta	Реакция вентилей AND, NAND D <= 1 C <= 0

Время моделирования	Дельта-задержка	Вычисление реакции цепи в узле D с помощью механизма дельта-задержки
	3*delta	Реакция вентиля AND D <= 0
6 ns		

Рис. 3.25,в. Вычисление реакции цепи в узле D с помощью механизма дельта-задержки

При отсутствии механизма дельта-задержки результат вычислений в узле зависит от маршрута вычислений (рис. 3.25, *а*). В случае, если вначале вычисляется реакция вентиля AND, мы убеждаемся, что в узле D возникает узкий импульс в момент t = 5ns. Если же вначале будет вычисляться реакция вентиля NAND, мы получим в узле D уровень логического нуля в течение всего интервала моделирования.

При вычислении же реакции цепи в узле D с помощью механизма дельта-задержки (рис. 3.25, *б*) мы достигаем однозначного результата (узкого импульса в интервале от 5ns до 5ns + 3* delta). Дельта-задержка (delta) является бесконечно малой задержкой и используется симулятором для достижения однозначности вычисления без отображения этих дельта-задержек в окне временных диаграмм.

В подтверждение этого опишем данную цепь (entity deltadelay, architecture deltadelay) на VHDL и составим соответствующую испытательную программу (delta_delay_tb) (рис. 3.26). Результатом моделирования является узкий импульс в момент времени t = 5ns (рис. 3.27).

```
-------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY deltadelay IS
  PORT ( a, clock : IN  std_logic  ;
         d        : OUT std_logic );
END deltadelay ;
ARCHITECTURE deltadelay OF deltadelay IS
      SIGNAL b,c : std_logic;
BEGIN
  b <= NOT(a);
  c <= NOT(clock AND b);
  d <= c AND b;
END deltadelay;
-------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY delta_delay_tb IS
-------------------------------------------------
END delta_delay_tb;
ARCHITECTURE delta_delay_tb OF delta_delay_tb IS
        COMPONENT deltadelay
             PORT (a,clock : IN std_logic;
```

```
                d       : OUT std_logic);
        END COMPONENT;
        SIGNAL  sa     : std_logic := '1';
        SIGNAL  sclock : std_logic       ;
        SIGNAL  sd     : std_logic       ;
BEGIN
  dut : deltadelay
        PORT MAP(
             a      => sa,
             clock  => sclock,
             d      => sd);
  sa      <= '0' AFTER  5 ns;
  sclock  <= '1';
END delta_delay_tb;
-------------------------------------------------
```

Рис. 3.26. Пример моделирования механизма дельта-задержки

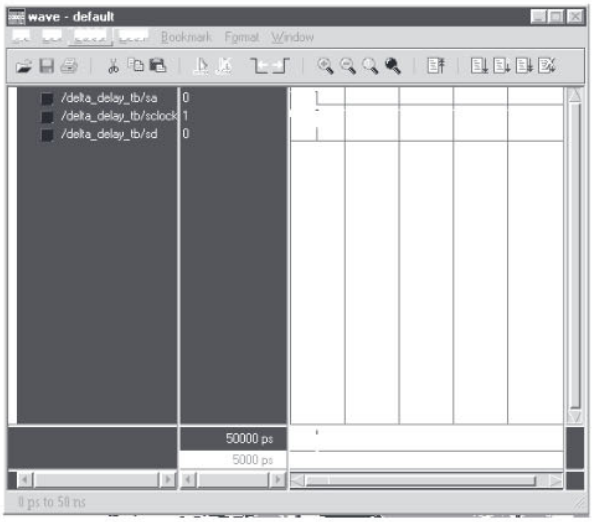


Рис. 3.27. Результаты моделирования цепи с помощью механизма дельта-задержки

3.6. Атрибуты

Атрибуты — это характеристики (квалификаторы), связанные с именованными элементами VHDL. Различают предопределенные атрибуты и атрибуты пользователя (Predefined Attributes и UserDefined Attributes). Здесь мы будем обсуждать лишь предопределенные атрибуты. Заметим лишь, что определенные пользователем атрибуты играют роль в подмножествах языка VHDL — языках BSDL и HSDL [11a, 63a], предназначенных для описания JTAG-интерфейса.

На рис. 3.28, *а* приведены атрибуты массивов, а на рис. 3.28, *в* — атрибуты сигналов, играющие особенно важную роль при моделировании цифровых систем на VHDL.

Вид атрибута	Вычисляемое значение	Пример
T ' base	Базовый тип данных	natural ' base -> integer
T ' left	Левая граница значений T	integer ' left -> -2147483647 bit ' left -> '0'
T ' right	Правая граница значений T	integer ' right -> 2147483647 bit ' right -> '1'
T ' low	Нижняя граница значений T	TYPE bit4 is 15 downto 0 bit4 ' low -> 0
T ' high	Верхняя граница значений T	TYPE bit4 is 15 downto 0 bit4 ' high -> 15
T ' pos(X)	Позиция значения X в наборе значений T	TYPE state IS (A, B, C, D); state ' pos (B) -> 1
T ' val(N)	Значение элемента в пози- ции N набора значений T	TYPE state IS (A, B, C, D); state ' val (2) -> C
T ' succ(X)	Значение в наборе значений T, на одну позицию большее X	TYPE state IS (A, B, C, D); state ' succ (A) -> B
T ' pred(X)	Значение в наборе значений T, на одну позицию меньшее X	TYPE state IS (A, B, C, D); state ' pred (C) -> B
T ' leftof(X)	Значение в наборе значений T, записанное в позиции слева от X	TYPE state IS (A, B, C, D); state ' leftof (C) -> B
T ' rightof(X)	Значение в наборе значений T, записанное в позиции справа от X	TYPE state IS (A, B, C, D); state ' rightof (C) -> D

Рис. 3.28,а. Атрибуты типов данных в — атрибуты сигналов

Вид атрибута	Вычисляемое значение	Пример
A ' left (N)	Левая граница диапазона индексов N-координаты массива A	TYPE mem IS ARRAY (0 TO 15, 7 DOWN 0) mem ' left (1) -> 0 mem ' left (2) -> 7
A ' right (N)	Правая граница диапазона индексов N-координаты массива A	mem ' right (1) -> 15 mem ' right (2) -> 0
A ' high (N)	Верхняя граница диапазона индексов N-координаты массива A	mem ' high (1) -> 15 mem ' high (2) -> 7
A ' low(N)	Нижняя граница диапазона индексов N-координаты массива A	mem ' low (1) -> 0 mem ' low (1) -> 0
A ' range(N)	Диапазон индексов N-координаты массива A	mem ' range (1) -> 0 to 15 mem ' range (2) -> 7 downto 0
A ' reverse_range(N)	Обратный диапазон индексов N-координаты массива A	mem ' reverse_range (1) -> 15 downto 0 mem ' reverse_range (2) -> 0 to 7
A ' length(N)	Длина диапазона индексов N-координаты массива A	mem ' length (1) -> 16 mem ' length (2) -> 8

Рис. 3.28,б. Атрибуты массивов

Вид атрибута	Вычисляемое значение	Пример
S ' delayed (t)	Значение сигнала, существовавшее на время t перед вычислением атрибута	carry ' delayed (20 ns)
S ' stable (t)	ИСТИНА, если не происходило изменение сигнала в течение времени t	clock ' stable (3 ns)
S ' transaction	ИСТИНА, если происходит очередное присвоение значения сигналу	data ' transaction
S ' event	ИСТИНА, если происходит изменение сигнала	data ' event
S ' active	ИСТИНА, если присвоение сигналу выполнено, но текущее значение еще не изменено (не окончен временной интервал, заданный выражением after)	reset ' active
S ' quiet	S ' quiet = not S ' active	carry ' quiet
S ' last_event	Время от последнего изменения сигнала до момента вычисления этого атрибута	reset ' last_event
S ' last_active	Время от последнего присвоения значения сигналу до момента вычисления этого атрибута	comput ' last_active
S ' last_value	Последнее присвоенное данному сигналу значение	comput ' last_value

Рис. 3.28,в. Атрибуты сигналов

Приведем пример работы с атрибутами. На рис. 3.29, *а* приведена VHDL-программа, вычисляющая четыре атрибута для сигнала s: s'event, s'last_value, s'stable (ns) и s'delayed (10ns). На рис. 3.29, *б* приведены результаты моделирования этой программы. Отсюда видно, что сигнал q2 всегда хранит предыдущее значение сигнала s; сигнал q3 показывает, что сигнал s стабилен в течение не менее 3 ns лишь в интервалах от 0 до 5ns и от 10ns и далее. Сигнал q1 устанавливается в состояние логической единицы в момент первого изменения значения сигнала s. На рис. 3.29, в приведена еще одна VHDL-программа, отличающаяся наличием сброса для сигнала q1. Из результатов моделирования на рис. 3.29, *г* видно, что сигнал q1 теперь способен сигнализировать о каждом изменении значения сигнала s.

```
-----------------------------------------------
-- ATTRIBUTES
-----------------------------------------------
entity attributes is
    port(q1 : out boolean;
         q2 : out bit;
         q3 : out boolean;
         q4 : out bit);
end attributes;
architecture arch of attributes is
    signal s : bit;
begin
    s <= '0' after 2 ns,
         '1' after 5 ns,
         '0' after 6 ns,
```

```
         '1' after 8 ns;
    q1 <= s'event;
    q2 <= s'last_value;
    q3 <= s'stable(3 ns);
    q4 <= s'delayed(10 ns);
end arch;
```

Рис. 3.29,а. Пример работы с атрибутами

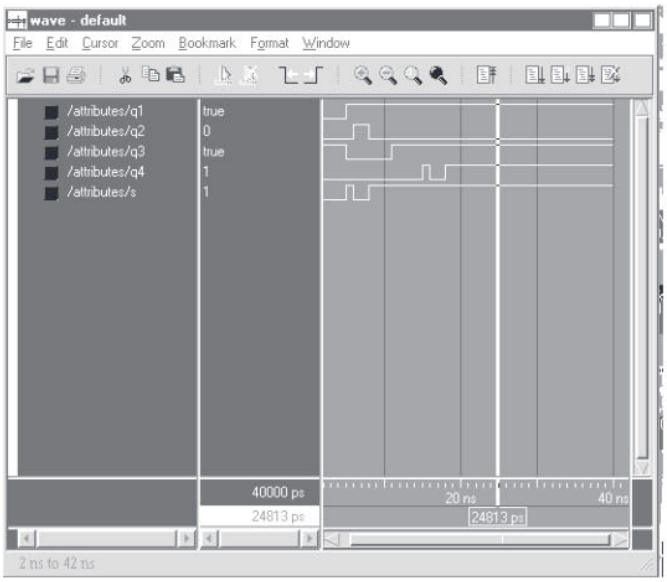


Рис. 3.29,б. Результаты моделирования сигналов, определяемых с помощью атрибутов сигналов

```
-------------------------------------------------
-- ATTRIBUTES WITH RESET FOR S'EVENT
-------------------------------------------------
entity attributes1 is
    port(q1 : out boolean;
         q2 : out bit;
         q3 : out boolean;
         q4 : out bit);
end attributes1;
architecture arch1 of attributes1 is
    signal s : bit;
    signal s1: boolean;
begin
     s <= '0' after 2 ns,
          '1' after 5 ns,
          '0' after 6 ns,
          '1' after 8 ns;
    s1 <= s'event;
    q1 <= s'event, false after 500 ps;
```

```
    q2 <= s'last_value;
    q3 <= s'stable(3 ns);
    q4 <= s'delayed(10 ns);
end arch1;
```

Рис. 3.29,в. Еще один пример работы с атрибутами

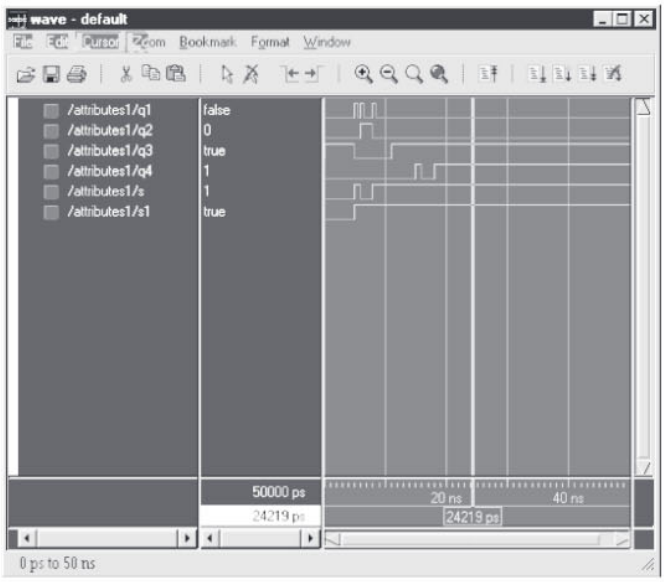


Рис. 3.29,г. Результаты моделирования сигналов, определяемых с помощью атрибутов сигналов (после введения сброса для q1)

3.7. Функции и процедуры

Использование программ — хорошо известный мощный инструмент программирования. Подпрограммы языка VHDL делятся на процедуры и функции.

Выполнение процедуры заключается в реализации некоторых действий над объектами и вызывается с помощью оператора вызова процедуры.

Функция определяет последовательность действий и возвращает значение, называемое результатом. Вызов функции, в отличие от вызова процедуры, должен использоваться в выражении.

Подпрограмма задается в виде описания подпрограммы, которое определяет ее имя, формальные параметры и (для функции) ее результат, и тело подпрограммы, которое определяет последовательность действий. В вызове подпрограммы задаются физические параметры, которые сопоставляются с формальными параметрами. Используемые в вызове подпрограммы фактические параметры задаются либо в порядке следования соответствующих формальных параметров в описании (спецификации) подпрограммы, либо связыванием фактических параметров с именами формальных параметров.

Параметр (Parameter) — это одно из именованных понятий, связанных с подпрограммами и используемых для связи с соответствующим типом подпрограмм. Формальный параметр — это идентификатор, используемый для обозначения

имени понятия в теле. Фактический параметр — частное понятие, связываемое с соответствущим формальным параметром при вызове подпрограммы.

Примеры VHDL-функций приведены на рис. 3.30, *а* примеры использования подобных функций — на рис. 3.31.

```
-----------------------------------------------------
-- FUNCTIONS
-----------------------------------------------------
-- BOOLEAN_TO_BIT
-- IN:     BOOLEAN
-- RETURN: BIT
-----------------------------------------------------
FUNCTION BL_TO_BIT(A : BOOLEAN) RETURN BIT IS
  BEGIN
    IF A THEN
       RETURN '1';
    ELSE
       RETURN '0';
    END IF;
END BL_TO_BIT;
-----------------------------------------------------
-- BITVECTOR_TO_INTEGER
-- IN:     BIT_VECTOR
-- RETURN: INTEGER
-----------------------------------------------------
FUNCTION BV_TO_INT(BV : BIT_VECTOR) RETURN INTEGER IS
  VARIABLE RESULT, ABIT : INTEGER := 0;
  VARIABLE COUNTER      : INTEGER := 0;
BEGIN
  BITS : FOR I IN BV'LOW TO BV'HIGH LOOP
  ABIT := 0;
    IF ((BV(I) = '1')) THEN
       ABIT := 2**(I - BV'LOW);
    END IF;
       RESULT := RESULT + ABIT;
       COUNTER := COUNTER + 1;
       EXIT BITS WHEN COUNTER = 32;
    END LOOP BITS;
    RETURN (RESULT);
END BV_TO_INT;
-----------------------------------------------------
-- INTEGER_TO_BITVECTOR
-- IN:     INTEGER, VALUE AND WIDTH
-- RETURN: BITVECTOR, WITH RIGHT BIT
-- THE MOST SIGNIFICANT
-----------------------------------------------------
FUNCTION INT_TO_BV(VAL, WIDTH : INTEGER)
         RETURN BIT_VECTOR IS
  VARIABLE RESULT : BIT_VECTOR(0 TO WIDTH-1) :=
                    (OTHERS => '0');
```

```
  VARIABLE BITS   : INTEGER := WIDTH;
BEGIN
    IF (BITS > 32) THEN
       BITS := 32;
    ELSE
       ASSERT 2**BITS > VAL
         REPORT "VALUE TOO BIG FOR BIT_VECTOR WIDTH"
         SEVERITY WARNING;
    END IF;
    FOR I IN 0 TO BITS - 1 LOOP
        IF ((VAL/(2**I) MOD 2 = 1) THEN
            RESULT(I) := '1';
        END IF;
    END LOOP;
    RETURN (RESULT);
END BV_TO_INT;
-----------------------------------------------------
-- INCREMENT_TO_BITVECTOR
-- IN:     BITVECTOR
-- RETURN: BITVECTOR
-----------------------------------------------------
FUNCTION INC_TO_BV(A : BIT_VECTOR) RETURN BIT_VECTOR IS
  VARIABLE S     : BIT_VECTOR(A'RANGE);
  VARIABLE CARRY : BIT;
BEGIN
  CARRY := '1';
  FOR I IN A'LOW TO A'HIGH LOOP
      S(I)  := A(I) XOR CARRY;
      CARRY := A(I) AND CARRY;
  END LOOP;
  RETURN (S);
END INC_TO_BV;
-----------------------------------------------------
-- MAJORITY FUNCTION FOR 3-INPUTS
-- IN:     BITS
-- RETURN: BIT
-----------------------------------------------------
FUNCTION MAJORITY(A,B,C : BIT) RETURN BIT IS
BEGIN
  RETURN ((A AND B) OR (A AND C) OR (B AND C));
END MAJORITY;
-----------------------------------------------------
```

Рис. 3.30. Примеры VHDL-функций

```
-----------------------------------------------------
-- USING FUNCTIONS
-----------------------------------------------------
-- DESIGN OF FULL ADDER(FA)
-----------------------------------------------------
```

```
ENTITY FA IS PORT(
   A,B,CARRY_IN  : IN BIT;
   SUM,CARRY_OUT : OUT BIT);
END FA;
ARCHITECTURE FA OF FA IS
   FUNCTION MAJORITY(A,B,C : BIT) RETURN BIT IS
   BEGIN
     RETURN ((A AND B) OR (A AND C) OR (B AND C));
   END MAJORITY;
BEGIN
   SUM       <= A XOR B XOR CARRY_IN;
   CARRY_OUT <= MAJORITY(A,B,CARRY_IN);
END FA;
-------------------------------------------------------
-- PACKAGE CONTAINING TWO FUNCTIONS
-------------------------------------------------------
PACKAGE MY_PACKAGE IS
   FUNCTION INC_TO_BV(A : BIT_VECTOR) RETURN BIT_VECTOR;
   FUNCTION MAJORITY(A,B,C : BIT) RETURN BIT;
END MY_PACKAGE;
PACKAGE BODY MY_PACKAGE IS
-------------------------------------------------------
-- MAJORITY FUNCTION FOR 3-INPUTS
-- IN:     BITS
-- RETURN: BIT
-------------------------------------------------------
FUNCTION MAJORITY(A,B,C : BIT) RETURN BIT IS
BEGIN
  RETURN ((A AND B) OR (A AND C) OR (B AND C));
END MAJORITY;
-------------------------------------------------------
-- INCREMENT_TO_BITVECTOR
-- IN:     BITVECTOR
-- RETURN: BITVECTOR
-------------------------------------------------------
FUNCTION INC_TO_BV(A : BIT_VECTOR) RETURN BIT_VECTOR IS
  VARIABLE S     : BIT_VECTOR(A'RANGE);
  VARIABLE CARRY : BIT;
BEGIN
  CARRY := '1';
  FOR I IN A'LOW TO A'HIGH LOOP
      S(I)  := A(I) XOR CARRY;
      CARRY := A(I) AND CARRY;
  END LOOP;
  RETURN (S);
END INC_TO_BV;
-------------------------------------------------------
END MY_PACKAGE;
```

Рис. 3.31. Примеры применения VHDL-функций

3.8. Пакеты и конфигурации

Пакет (Package) определяет группу логически связанных понятий, таких как типы, объекты данных этих типов и подпрограммы с параметрами этих типов.

Пакет состоит из:

- описания (декларации) пакета,
- тела пакета.

Описание пакета содержит описание всех понятий, которые могут быть явно использованы вне пакета. Тело пакета содержит порядок вычислений подпрограмм (процедур и функций), которые специфицированы в описании пакета, а также определения локальных имен. Описание пакета не требует обязательного присутствия тела пакета.

Любое описание, содержащееся в описании пакета, видимо из некоторого VHDL-блока, если пакет предварительно скомпилирован в одну из библиотек проекта, а VHDL-блок содержит спецификатор использования этого пакета (Use Clause), обеспечивающий прямую видимость описаний.

На рис. 3.32 приведен пример пакета mypack. Описание данного пакета содержит два подтипа данных:

- eightbit: 8-разрядный битовый вектор,
- fourbit: 4-разрядный битовый вектор;

а также описание функции shif_right.

В теле же пакета содержится тело данной функции, осуществляющей сдвиг входного битового вектора на один разряд вправо.

```
------------------------------------------------------------
-- PACKAGE EXAMPLE
------------------------------------------------------------
PACKAGE mypack IS
  SUBTYPE eightbit IS BIT_VECTOR(0 TO 7);
  SUBTYPE fourbit IS BIT_VECTOR(0 TO 3);
  FUNCTION shift_right(val : BIT_VECTOR)
  RETURN BIT_VECTOR;
END mypack;
PACKAGE BODY mypack IS
  FUNCTION shift_right(val : BIT_VECTOR) RETURN BIT_VECTOR IS
    VARIABLE result : BIT_VECTOR(0 TO (val'LENGTH - 1));
  BEGIN
    result := val;
    IF (val'LENGTH > 1) THEN
      FOR i IN 0 TO (val'LENGTH - 2) LOOP
        result(i) := result(i + 1);
      END LOOP;
      result(val'LENGTH - 1) := '0';
    ELSE
      result(0) := '0';
    END IF;
    RETURN result;
  END shift_right;
END mypack;
```

Рис. 3.32. VHDL-пакет

3.9. Стили проектирования на VHDL

В технической литературе упоминаются четыре стиля проектирования на VHDL:

- Pure Structural Architecture (Чисто структурное архитектурное тело);
- Pure Behavioral Architecture (Чисто поведенческое архитектурное тело);
- Data Flow Architecture (Архитектурное тело потокового стиля);
- Mixed Architecture (Архитектурное тело смешанного стиля).

На рис. 3.33 эти стили проектирования проиллюстрированы на примере проектирования полного сумматора.

```
LIBRARY ieee;
ENTITY XOR2 IS
  PORT(P1,P2:  IN  bit;
       PZ   :  OUT bit);
END XOR2;
ARCHITECTURE xor2 OF XOR2 IS
BEGIN
  PZ <= P1 XOR P2;
END xor2;
-----------------------------------------
ENTITY AND2 IS
  PORT(P1,P2:  IN  bit;
       PZ   :  OUT bit);
END AND2;
ARCHITECTURE and2 OF AND2 IS
BEGIN
  PZ <= P1 AND P2;
END and2;
-----------------------------------------
ENTITY OR3 IS
  PORT(P1,P2,P3:  IN  bit;
       PZ      :  OUT bit);
END OR3;
ARCHITECTURE or3 OF OR3 IS
BEGIN
  PZ <= P1 OR P2 OR P3;
END or3;
-----------------------------------------
ENTITY FA IS
  PORT(A, B, CIN: IN  bit;
       SUM, COUT: OUT bit);
END FA;
-----------------------------------------
ARCHITECTURE FA_MIXED OF FA IS
   COMPONENT XOR2
       PORT(P1,P2 : IN  bit;
            PZ    : OUT bit);
   END COMPONENT;
   SIGNAL S1 : bit;
BEGIN
```

```
   X1: XOR2 PORT MAP(A,B,S1);
   PROCESS(A,B,CIN)
       VARIABLE T1,T2,T3: bit;
   BEGIN
       T1 := A AND B;
       T2 := B AND CIN;
       T3 := A AND CIN;
       COUT <= T1 OR T2 OR T3;
   END PROCESS;
   SUM <= S1 XOR CIN;
END FA_MIXED;
-----------------------------------------
ARCHITECTURE FA_BEHAVE OF FA IS
BEGIN
P_COUT:PROCESS(A,B,CIN)
         VARIABLE T1,T2,T3: bit;
       BEGIN
          T1 := A AND B;
          T2 := B AND CIN;
          T3 := A AND CIN;
          COUT <= T1 OR T2 OR T3;
       END PROCESS;
P_SUM: PROCESS(A,B,CIN)
         VARIABLE S1: bit;
       BEGIN
         S1 := A XOR B;
         SUM <= S1 XOR CIN;
       END PROCESS;
END FA_BEHAVE;
-----------------------------------------
ARCHITECTURE FA_DATAFLOW OF FA IS
BEGIN
   COUT <= (A AND B) OR (A AND CIN) OR (B AND CIN);
   SUM  <= (A XOR B) XOR CIN;
END FA_DATAFLOW;
-----------------------------------------
ARCHITECTURE FA_STRUCTURE OF FA IS
   COMPONENT XOR2
       PORT(P1,P2 : IN  bit;
            PZ    : OUT bit);
   END COMPONENT;
   COMPONENT AND2
       PORT(P1,P2 : IN  bit;
            PZ    : OUT bit);
   END COMPONENT;
   COMPONENT OR3
       PORT(P1,P2,P3 : IN  bit;
            PZ       : OUT bit);
   END COMPONENT;
   SIGNAL S1   : bit;
   SIGNAL AB   : bit;
```

```
   SIGNAL ACIN : bit;
   SIGNAL BCIN : bit;
BEGIN
   X1: XOR2 PORT MAP(A,B,S1);
   X2: XOR2 PORT MAP(S1,CIN,SUM);
   A1: AND2 PORT MAP(A,B,AB);
   A2: AND2 PORT MAP(A,CIN,ACIN);
   A3: AND2 PORT MAP(B,CIN,BCIN);
   O1:  OR3 PORT MAP(AB,ACIN,BCIN,COUT);
END FA_STRUCTURE;
-----------------------------------------
ENTITY fa_tb IS
---------------------------
END fa_tb;
ARCHITECTURE fa_tb OF fa_tb IS
        CONSTANT off_period : time := 30 ns;
        CONSTANT on_period  : time := 20 ns;
        COMPONENT FA
           PORT(A, B, CIN: IN  bit;
                SUM, COUT: OUT bit);
        END COMPONENT;
        SIGNAL  sA     : bit ;
        SIGNAL  sB     : bit ;
        SIGNAL  sCIN   : bit ;
        SIGNAL  sSUM   : bit ;
        SIGNAL  sCOUT   : bit ;
BEGIN
  dut : FA
        PORT MAP(
             A  => sA,
             B  => sB,
          CIN  => sCIN,
          SUM  => sSUM,
          COUT  =>sCOUT);
     sCIN <= '1' AFTER off_period WHEN sCIN = '0'
           ELSE '0' AFTER on_period;
     sA <= sCIN'delayed(10 ns);
     sB <= sCIN'delayed(25 ns);
END fa_tb;
```

Рис. 3.33. Стили проектирования на VHDL (на примере полного сумматора)

Архитектура FA_MIXED является архитектурой смешанного стиля. Структурная часть здесь представлена реализацией вентиля XOR2 (с меткой X1). Поведенческой частью является процесс, внутри которого с помощью переменных T1, T2, T3 и сигнала COUT вычисляется значение переноса на выходе сумматора. Наконец, оператор присвоения для сигнала SUM вычисляет значение суммы на выходе сумматора.

Поведенческая архитектура FA_BEHAVE сумматора содержит два процесса:

- P_COUT (вычисляет значение переноса);
- P_SUM (вычисляет значение суммы).

Наконец, архитектура FA_DATAFLOW является архитектурой потокового стиля.

3.10. Структура программы моделирования на VHDL

В гл. 2 уже рассматривались вопросы, связанные с построением испытательной программы (Test Bench), необходимой для выполнения процесса моделирования. Здесь мы перечислим основные способы формирования внешних воздействий для моделируемого объекта, а также напишем программу моделирования в альтернативном стиле, как совокупность тестовых векторов, механизма их подачи на входы моделируемого объекта и контроля правильности выходных реакций моделируемого объекта.

На рис. 3.34 внешние воздействия для моделируемого объекта носят периодический характер (Waveform Generation) и получаются с помощью операторов присвоения для сигналов, выполняющих в общем случае несколько транзакций (актов присваивания новых значений).

```
library ieee;
use ieee.std_logic_1164.all;
entity waveform_generation is
  port( reset     : out bit;
        clk1,clk2 : out std_logic :='0');
end waveform_generation;
architecture arch of waveform_generation is
begin
      reset <= '0',
               '1' after 100 ns,
               '0' after 180 ns,
               '1' after 210 ns;
      two_phase: process
                 begin
                   clk1 <= 'U' after  50 ns,
                           '1' after 100 ns,
                           'X' after 150 ns,
                           'U' after 200 ns,
                           'Z' after 220 ns,
                           '0' after 250 ns;
                   clk2 <= 'U' after 100 ns,
                           '1' after 120 ns,
                           'X' after 150 ns,
                           'U' after 250 ns,
                           'Z' after 270 ns,
                           '0' after 300 ns;
                    wait for 350 ns;
                 end process;
end arch;
```

Рис. 3.34. Формирование непериодических внешних воздействий

На рис. 3.35 внешние воздействия для моделируемого объекта носят периодический характер (Repetitive Patterns). Эти воздействия получаются следующими способами:

- с помощью оператора присвоения, в правой части которого содержится тот же сигнал, что и в левой, но с отрицанием, так получается простейший генератор импульсов (причем длительность импульса и паузы одинакова);
- с помощью процесса, который генерирует серию импульсов (причем длительность импульса и паузы здесь в общем случае различна и задается с помощью двух констант);
- с помощью условного оператора присваивания, вновь использующего константы для задания длительности импульса и паузы;
- и, наконец, несколько серий импульсов со сдвигом друг относительно друга можно получить с помощью атрибута s'delayed (T).

```
library ieee;
entity repetitive_patterns is
  port( apr, clk                    : inout bit;
        d_clk, dly1_clk, dly2_clk   : out  bit);
end repetitive_patterns ;
architecture arch of repetitive_patterns is
      constant off_period : time := 30 ns;
      constant on_period  : time := 20 ns;
begin
      apr <= not apr after 20 ns;
                 process
                   begin
                   wait for off_period;
                   d_clk <= '1';
                   wait for on_period;
                   d_clk <= '0';
                 end process;
     clk <= '1' after off_period when clk = '0'
            else '0' after on_period;
     dly1_clk <= clk'delayed(10 ns);
     dly2_clk <= clk'delayed(25 ns);
end arch;
```

Рис. 3.35. Формирование периодических внешних воздействий

На рис. 3.36 приведено описание (модель) проверяемого объекта — трехразрядного счетчика, для которого напишем программу моделирования (Test Bench) (рис. 3.37).

```
-------------------------------------------------------
-- DESIGN UNDER TEST(DUT)- 3-BIT COUNTER
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
PACKAGE MY_COUNTER_PACKAGE IS
  COMPONENT COUNTER
     PORT(CLOCK,RESET : IN STD_LOGIC;
          COUNTER     : INOUT STD_LOGIC_VECTOR(2 DOWNTO 0));
  END COMPONENT;
END MY_COUNTER_PACKAGE;
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

```
ENTITY COUNTER IS
     PORT(CLOCK,RESET : IN STD_LOGIC;
          COUNTER     : INOUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END COUNTER;
USE WORK.STD_ARITH.ALL;
ARCHITECTURE ARCH_COUNTER OF COUNTER IS
BEGIN
COUNTER_FUNCTION: PROCESS(CLOCK,RESET)
                  BEGIN
                    IF RESET = '1' THEN
                         COUNTER <= (OTHERS => '0');
                    ELSIF (CLOCK'EVENT AND CLOCK = '1')THEN
                         COUNTER <= COUNTER + 1;
                    END IF;
                  END PROCESS;
END ARCH_COUNTER;
-------------------------------------------------------
```

Рис. 3.36. Описание проверяемого обьекта — 3-bit Counter

```
-------------------------------------------------------
-- TEST BENCH FOR DESIGN UNDER TEST(DUT)- 3-BIT COUNTER
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY TB_COUNTER IS
END TB_COUNTER;
USE WORK.MY_COUNTER_PACKAGE.ALL;
ARCHITECTURE MYTB OF TB_COUNTER IS
  SIGNAL CLOCK,RESET : STD_LOGIC;
  SIGNAL COUNTER     : STD_LOGIC_VECTOR(2 DOWNTO 0);
  TYPE TEST_VECTOR IS RECORD
        CLOCK   : STD_LOGIC;
        RESET   : STD_LOGIC;
        COUNTER : STD_LOGIC_VECTOR(2 DOWNTO 0);
  END RECORD;
  TYPE TEST_VECTOR_ARRAY IS ARRAY(NATURAL RANGE )
       OF TEST_VECTOR;
  CONSTANT TEST_VECTORS: TEST_VECTOR_ARRAY := (
    -- RESET THE COUNTER
        (CLOCK => '0',RESET => '1', COUNTER => "000"),
        (CLOCK => '1',RESET => '1', COUNTER => "000"),
        (CLOCK => '0',RESET => '0', COUNTER => "000"),
    -- CLOCK THE COUNTER SEVERAL TIMES
        (CLOCK => '1',RESET => '0', COUNTER => "001"),
        (CLOCK => '0',RESET => '0', COUNTER => "001"),
        (CLOCK => '1',RESET => '0', COUNTER => "010"),
        (CLOCK => '0',RESET => '0', COUNTER => "010"),
        (CLOCK => '1',RESET => '0', COUNTER => "011"),
        (CLOCK => '0',RESET => '0', COUNTER => "011"),
```

```
        (CLOCK => '1',RESET => '0', COUNTER => "100"),
        (CLOCK => '0',RESET => '0', COUNTER => "100"),
        (CLOCK => '1',RESET => '0', COUNTER => "101"),
        (CLOCK => '0',RESET => '0', COUNTER => "101"),
        (CLOCK => '1',RESET => '0', COUNTER => "110"),
        (CLOCK => '0',RESET => '0', COUNTER => "110"),
        (CLOCK => '1',RESET => '0', COUNTER => "111"),
        (CLOCK => '0',RESET => '0', COUNTER => "111"),
        (CLOCK => '1',RESET => '0', COUNTER => "010"),
    -- RESET THE COUNTER
        (CLOCK => '0',RESET => '1', COUNTER => "000"),
        (CLOCK => '1',RESET => '1', COUNTER => "000"),
        (CLOCK => '0',RESET => '0', COUNTER => "000"),
    -- CLOCK THE COUNTER SEVERAL TIMES
        (CLOCK => '1',RESET => '0', COUNTER => "001"),
        (CLOCK => '0',RESET => '0', COUNTER => "001"),
        (CLOCK => '1',RESET => '0', COUNTER => "010"),
        (CLOCK => '0',RESET => '0', COUNTER => "010"),
        (CLOCK => '1',RESET => '0', COUNTER => "011"),
        (CLOCK => '0',RESET => '0', COUNTER => "011"),
        (CLOCK => '1',RESET => '0', COUNTER => "100"),
        (CLOCK => '0',RESET => '0', COUNTER => "100"),
        (CLOCK => '1',RESET => '0', COUNTER => "101"),
        (CLOCK => '0',RESET => '0', COUNTER => "101"),
        (CLOCK => '1',RESET => '0', COUNTER => "110"),
        (CLOCK => '0',RESET => '0', COUNTER => "110"),
        (CLOCK => '1',RESET => '0', COUNTER => "111"),
        (CLOCK => '0',RESET => '0', COUNTER => "111"),
        (CLOCK => '1',RESET => '0', COUNTER => "010"),
  );
BEGIN
-- INSTANTIATE DESIGN UNDER TEST
DUT: COUNTER PORT MAP(CLOCK   => CLOCK,
                      RESET   => RESET,
                      COUNTER => COUNTER);
-- APPLY TEST VECTORS AND CHECK RESULTS
TESTING: PROCESS
           VARIABLE VECTOR: TEST_VECTOR;
           VARIABLE ERRORS: BOOLEAN := FALSE;
         BEGIN
           FOR I IN TEST_VECTORS'RANGE LOOP
             -- GET VECTOR I
             VECTOR := TEST_VECTORS(I);
             -- SHEDULE VECTOR I
             CLOCK <= VECTOR.CLOCK;
             RESET <= VECTOR.RESET;
             -- WAIT FOR CIRCUIT TO SETTLE
             WAIT FOR 10 NS;
             -- CHECK OUTPUT VECTORS
```

```
            IF COUNTER /= VECTOR.COUNTER THEN
               ASSERT FALSE
                      REPORT "COUNTER IS WRONG VALUE"
                      ERRORS := TRUE;
            END IF;
          END LOOP;
          -- ASSERT REPORTS ON FALSE
          ASSERT NOT ERRORS
                 REPORT "TEST VECTORS FAILED"
                 SEVERITY NOTE;
          ASSERT ERRORS
                 REPORT "TEST VECTORS PASSED"
                 SEVERITY NOTE;
          WAIT;
        END PROCESS;
END MYTB;
-------------------------------------------------------
```

Рис. 3.37. Описание испытательной программы для проверяемого обьекта — 3-bit Counter

В данной программе внешние воздействия и эталонные реакции находятся в массиве тестовых векторов (константа TEST_VECTORS, имеющая тип данных Array Type).

Процесс, помеченный меткой TESTING, осуществляет подачу тестовых векторов на входы моделируемого счетчика, одновременно сравнивая выходные реакции счетчика с его эталонными реакциями с помощью операторов проверки Assert Statements.

3.11. Стандарты языка VHDL и другие стандарты EDA

3.11.1. Стандарты языка VHDL разработанные и разрабатываемые в IEEE

Краткое обозначение	Источники информации	Краткое описание
IEEE Standard 1076 VHDL	www.eda.org/vasg [28a]	В настоящее время язык VHDL официально определен стандартом IEEE Standard 1076-2001, IEEE Standard VHDL Language Reference Manual. Этот документ разработан подкомитетом DASC (IEEE Design Automation Standards Committee), носящим имя VHDL Analysis and Standardization Group (VASG).
VHDL Programming Language Interface	www.eda.org/vhdlpli [73a]	Подкомитет VHDL PLI Task Force разрабатывает спецификацию интерфейса языка программирования VHDL для баз данных и симуляторов VHDL-проектирования
IEEE Standard 1076.1 VHDL-AMS	www.eda.org/vhdl-ams [29a]	Стандарт IEEE Standard 1076.1-1999, IEEE Standard VHDL Analog and Mixed-Signal Extensions определяет расширения VHDL для моделирования аналоговых и цифровых компонентов. Этот расширенный язык известен под неформальным именем VHDL-AMS

Краткое обозначение	Источники информации	Краткое описание
IEEE Standard 1076.2 Mathematical Packages	www.eda.org/matb [30a]	Стандарт IEEE Standard 1076.2-1996, IEEE Standard VHDL Mathematical Packages описывает два математических пакета **math_real** и **math_complex**
IEEE Standard 1076.3 Synthesis Packages	www.eda.org/vhdlsyntb [31a]	Стандарт IEEE Standard 1076.3-1997, IEEE Standard VHDL Synthesis Packages описывает два пакета **numeric_bit** и **numeric_std**. Этот стандарт также специфицирует интерпретацию инструментами синтеза стандартных логических типов, определенных в пакете **std_logic_1164**
IEEE Standard 1076.4 VITAL	www.eda.org/vital [32a]	Фирмами-разработчиками и производителями ASIC была создана организация, получившая название VITAL Committee (VITAL- VHDL Initiative Towards ASIC Libraries). Эта организация разработала стандартную практику включения детальной временной информации в VHDL-модели ячеек ASIC. Так возник стандарт IEEE Standard 1076.4-2001, IEEE Standard VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification. Этот стандарт специфицирует пакет **vital_timing** в библиотеке **ieee**, содержащий типы данных для спецификации задержек распространения сигнала между контактами (Pin-to-Pin Propagation Delay) и временные ограничения (Timing Constraints), включая максимальное, минимальное и номинальное значения. Этот пакет также обеспечивает процедуры для временных проверок (Timing Checks). Данный стандарт также специфицирует пакет **vital_primitives**, который обеспечивает подпрограммы, помогающие описывать поведение ячеек ASIC. Последняя версия стандарта 2001 года описывает еще один пакет, **vital_memory**, который определяет типы данных и подпрограммы для разработки моделей памяти
IEEE P1076.5 VHDL Utility Library	www.eda.org/libutil [22a]	Подкомитет P1076.5 Working Group разрабатывает библиотеку утилит в дополнение к стандартным пакетам IEEE
IEEE Standard 1076.6 VHDL Synthesis Interoperability	www.eda.org/siwg [33a]	Стандарт IEEE Standard 1076.6-1999, IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis описывает модели, принятые в существующих инструментах синтеза. В дальнейшем планируется снять как можно больше существующих ныне ограничений на VHDL-код, предназначенный для синтеза
IEEE P1577 Object-Oriented VHDL	www.eda.org/oovhdl [25a]	Подкомитет P1577 Object-Oriented VHDL (OOVHDL) Working Group разрабатывает расширения VHDL для поддержки обьектно-ориентированного моделирования (Object-Oriented Modeling)
IEEE P1551 System and Interface Description	www.eda.org/sid [24a]	Подкомитет P1551 Working Group разрабатывает Standard for VHDL Electronic Digital System and Interface Design, который позволит разработчикам описывать интерфейсы и протоколы связи между модулями, независимо от их реализации
IEEE Standard 1164 Multivalue Logic System	www.eda.org [34a]	Стандарт IEEE Standard 1164-1993, IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164) описывает широко используемый пакет **std_logic_1164**

Краткое обозначение	Источники информации	Краткое описание
IEEE Standard 1029.1 WAVES	www.eda.org/waves [26a]	Стандарт IEEE Standard 1029.1-1998, IEEE Standard for VHDL Waveform and Vector Exchange to Support Design and Test Verification (WAVES) Language Reference Manual описывает обмен стимулами и реакциями между средами моделирования и тестирования

3.11.2. Другие стандарты автоматизации проектирования в электронике (ELECTRONIC DESIGN AUTOMATION-EDA)

Краткое обозначение	Источники информации	Краткое описание
IEEE Standard 1364 Verilog	www.ovi.org/ieee-1364/Welcome.html [35a]	Стандарт IEEE Standard 1364-2001, IIEEE Standard Description Language Based on the Verilog HDL описывает другой язык описания аппаратуры высокого уровня – Verilog HDL, очень популярный в Северной Америке и Азии. Рабочая группа, поддерживающая данный стандарт, пользуется поддержкой организации, именуемей Open Verilog International (OVI)
IEEE P1364.1 Verilog Synthesis Interoperability	www.eda.org/vlog-synth	Подкомитет P1364.1 Working Group разрабатывает спецификацию синтезируемого подмножества языка Verilog HDL, аналогичное подмножеству языка VHDL, определенного в стандарте IEEE Standard 1076.6
OVI Verilog-AMS	www.eda.org/verilog-ams [53a]	Подкомитет Verilog-AMS Technical Subcommittee of OVI разрабатывает расширения языка Verilog HDL для моделирования аналоговых и аналого-цифровых систем
IEEE Standard 1499 OMF	www.eda.org/omf [37a]	Стандарт IEEE Standard 1499-1998, IIEEE Standard Interface for Hardware Description Models of Electronics Components описывает интерфейс между моделями и симуляторами для различных платформ, разработанный организацией Open Model Forum
SLDL Initiative	www.inmet.com/SLDL [67a]	Инициатива SLDL (System Level Design Language Initiative) была сформулирована организацией EDA Industry Council. Это инициатива разрабатывать языки для поддержки спецификаций проектирования микроэлектронных систем на высоком уровне. Для этой цели разработан язык описания Rosetta
OVI Design Constraints	www.eda.org/dcwg [52a]	Подкомитет OVI Design Constraints Working Group разрабатывает язык для описания ограничений проектирования DCDL (Design Constraints Descriptions Language)
Virtual Socket Interface Alliance	www.vsi.org [72a]	Организация VSIA (Virtual Socket Interface Alliance) ставит главной своей задачей разработку эффективных систем на одном кристалле (System on a Chip). Главное внимание уделяется использованию готовых блоков интеллектуальных функций (Predesigned Blocks of Intellectual Property) – т. н. виртуальных компонент (Virtual Components (VCs)) по терминологии VSIA. VSIA коллекционирует спецификации виртуальных разъемов (Virtual Sockets), которые описывают функционирование виртуальных компонент (VC Functionality), их физические и электрические характеристики

Краткое обозначение	Источники информации	Краткое описание
IEEE P1497 Standard Delay Format	www.eda.org/sdf [23a]	Подкомитет IEEE P1497 Working Group продолжает работу над файловым форматом SDF (Standard Delay Format) для описания детальной временной информации о проекте
OVI Advanced Library Format	www.eda.org/alf [51a]	Комитет Advanced Library Format (ALF) Technical Committee организации OVI разработал формат для описания библиотеки ячеек ASIC для инструментов синтеза, инструментов анализа временных соотношений и мощности, а также других инструментов физического проектирования (Physical Design Tools)
IEEE Standard 1481 Delay and Power Calculation	www.eda.org/dpc www.si2.org/ieee1481 [36a]	Стандарт IEEE Standard 1481-1999, I IEEE Standard for Delay and Power Calculation описывает систему DPCS (Delay and Power Calculation System) и язык DCL (Delay CalculationLanguage), которые были разработаны организациями SI2 (Silicon Integration Initiative) и OVI. Языком DCL пользуются поставщики ячеек ASIC для задания порядка вычислений задержек для ячеек данной технологической библиотеки (Technology Library). Система DPCS используется в инструментах анализа временных соотношений (Timing Analysis Tools), которые генерируют SDF-файлы с детальной временной информацией о проекте
SI^2 Open Library Architecture	www.si2.org/ola [64a]	Архитектура Open Library Architecture (OLA) является развитием архитектуры SI2 и базируется на интеграции ALF и DCL. Планируется в будущем оформить эту архитектуру в виде стандарта IEEE
CHDStd	www.si2.org/CHDStd [17a]	Программа CHDStd (Chip Hierarchical Design System Technical Data Standard) спонсируется организациями SI2 и SEMATECH. Эта программа определяет интегрированную модель данных и процедурный интерфейс для доступа к проектным данным. Такой подход базируется на технологии, разработанной в IBM
EIA-682 Electronic Design Interchange Format (EDIF)	www.edif.org [20a]	Формат EDIF является стандартом обмена проектной информацией между EDA-инструментами (EDA Tools), а также между производителями электронной техники. Стандарт был разработан организацией EIA (Electronic Industries Alliance), а затем, организацией IEC (International Electrotechnical Commision), адаптирован в качестве международного стандарта
EIA-567-A Component Modeling and Interface	[19a]	Организация EIA разработала стандарт VHDL Hardware Component Modeling and Interface Standard для единообразной подготовки документации моделей компонент
EIA/IS-103-A Library of Parameterized Modules	www.edif.org/lpmweb [18a]	Стандарт Library of Parameterized Modules (LPM) позволяет реализовать цифровые проекты в различных технологиях (PLDs, FPGAs, Standard Cells). Версии LPM – описания существуют в форматах EDIF, VHDL и Verilog

Часть 2. Системы проектирования на языке VHDL

Глава 4. Работа с VHDL в среде системы моделирования ModelSim

4.1. Введение в ModelSim

Рассмотрим работу с проектами на языке VHDL в системе моделирования ModelSim 5.5b фирмы Model Technology Inc. для операционной системы Microsoft Windows 95/98/ME/NT/2000 [44a—46a].

ModelSim VHDL поддерживает следующие стандарты этого языка моделирования аппаратуры:

- IEEE Standard 1076-1987. Standard VHDL Language Reference Manual;
- IEEE Standard 1076-1993. Standard VHDL Language Reference Manual (ANSI);
- IEEE Standard 1164-1993. Multivalue Logic System for VHDL Interoperability (Std_logic_1164);
- IEEE Standard 1076.2-1996. VHDL Mathematical Packages;
- IEEE Standard 1076.4-1995. VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification.

Интерфейс системы моделирования является единообразным для всех платформ персональных компьютеров.

Многие описанные ниже операции выполняются с помощью выбора соответствующих клавиш или пунктов меню. Когда это будет необходимо, эквиваленты командной строки панели VSIM (VSIM command line (PROMPT:)) или верхнего меню (MENU) для этого выбора будут показаны в скобках. Вот, например, три опции для ввода команды **run—all**:

- с помощью клавиши верхней линейки инструментов  ;
- с помощью ввода команды с командной строки: (PROMPT: run -all);
- с помощью выбора в верхнем меню: (MENU:).

Технология «drag and drop» позволяет в среде данной системы моделирования копировать и перемещать сигналы между окнами. В случае использования этой технологии будем использовать обозначение DRAG&DROP. В основном наша работа будет протекать в главном окне системы (Main window). Отметим, что можно «прокрутить» историю ввода команд в этом окне (command history) с помощью стрелок вверх и вниз или просмотреть эту историю с помощью нескольких комбинаций клавиш для быстрого вызова (shortcut).

Shortcut	Description
click on prompt	left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor
his or history	shows the last few commands (up to 50 are kept)

Текст, вводимый в главном окне системы (Main transcript), может быть сохранен в специальном (макро-) файле с расширением *.do, чтобы воспроизвести написанные ранее команды в случае необходимости (reusing commands).

Сохранение введенного ранее текста можно выполнить до или во время выполнения процесса моделирования. Для этого необходимо выбрать из главного меню (Main menu) **File > Save Transcript As**. Восстановить ранее введенный текст можно с помощью команды **do**

do < do file name >.

Например, если мы сохранили совокупность команд для выполнения компиляции в файле **mycompile**, можно восстановить их с помощью одной команды

do mycompile.do.

Если же мы не хотим сохранять ранее введенный текст, существует опция очистки окна **File > Clear Transcript**.

4.2. Создание проекта

В данном разделе мы изучим особенности диалогового окна «Welcome to ModelSim» и обсудим процесс создания проекта. Проект является совокупностью объектов, «сущностей» (entities), предназначенных для реализации процесса проектирования с помощью языка описания аппаратуры (HDL design) согласно соответствующей технической спецификации.

Проекты облегчают взаимодействие между системой моделирования и пользователем, способствуют эффективной организации файлов (organizing files) и установке опций моделирования (simulating settings). Как минимум, проекты содержат рабочую библиотеку (work library) и отчет о текущем состоянии сеанса работы с системой моделирования (session state), который находится в файле с расширением.mpf. Проект может также содержать:

- исходные файлы на одном из HDLs (HDL source files) или ссылки на такие файлы;
- другие файлы, такие как Readme, или разнообразную проектную документацию (project documentation);
- локальные библиотеки (local libraries);
- ссылки на глобальные библиотеки (global libraries).

Во время первоначальной загрузки ModelSim пользователь видит диалоговое окно «Welcome to ModelSim» (рис. 4.1):

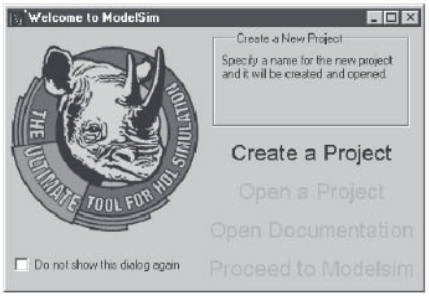


Рис. 4.1. Диалоговое окно «Welcome to ModelSim»

С помощью этого диалогового окна пользователь может:
- создать новый проект (**Create a Project**);
- открыть существующий проект (**Open a Project**);
- открыть справочную документацию по ModelSim (**Open Documentation**).

Для перехода к главному окну системы моделирования (ModelSim Main window) пользователь должен выбрать опцию **Proceed to ModelSim**.

Первоначальный запуск системы моделирования ModelSim можно выполнить одним из следующих способов:
- щелкнув соответствующую пиктограмму (Windows shortcut icon);
- выбрав соответствующий пункт в стартовом меню (Start menu);
- набрав в командной строке экрана MS DOS prompt **modelsim.exe**.

Для создания нового проекта надо выполнить следующие действия.

1. Выбрать опцию **Create Project** из диалогового окна Welcome to ModelSim (или пункт **File > New > Project** в главном окне системы). Это приведет к открытию соответствующего диалогового окна Create Project (рис. 4.2).

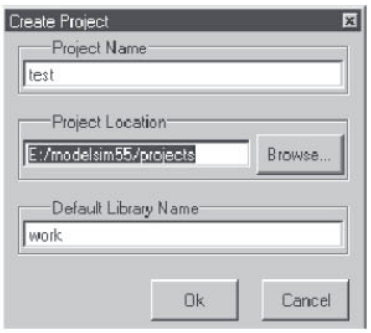


Рис. 4.2. Диалоговое окно «Create Project»

2. В этом диалоговом окне ввести имя проекта, например «test» в поле Project Name, и выбрать директорию для его хранения. Например, в поле Default Library Name сохраним установку «work».

3. Щелкнуть по клавише OK, что приводит к появлению главного окна системы (ModelSim Main window) с двумя подокнами (панелями): Workspace и VSIM. В свою очередь, панель Workspace содежит две секции: Project и Library.

Как видим, имя проекта индицируется в строке состояния (status bar).

4. Для присоединения к проекту новых файлов, содержащих информацию об объектах проектирования (design units), щелкнуть правой кнопкой мыши и выбрать Add file to Project (рис. 4.4).

5. Например, присоединим к проекту два Verilog-файла. Для этого надо щелкнуть по клавише Browse диалогового окна Add file to Project (рис. 4.5) и найти не-

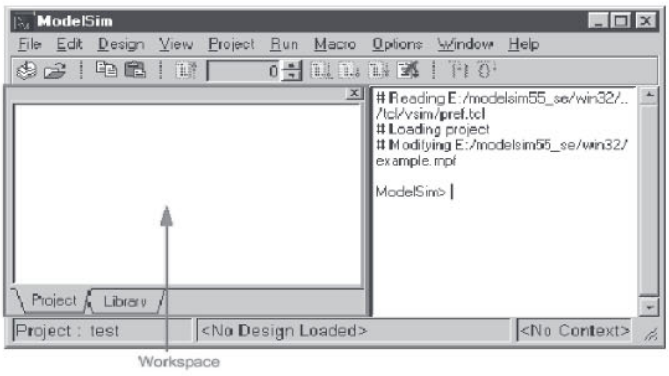


Рис. 4.3. Главное окно системы моделирования «ModelSim Main window»

обходимые файлы (в данном случае counter.v и tcounter.v). Затем установить опцию Reference from current location и нажать клавишу OK.

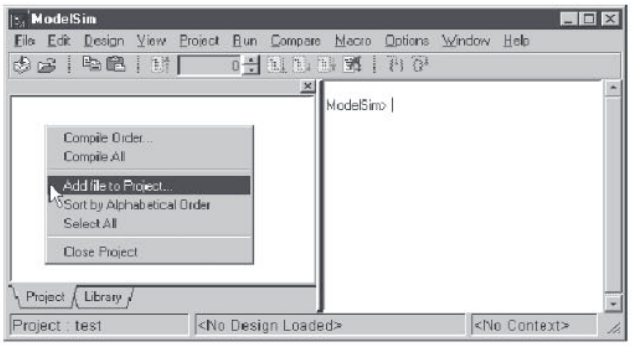


Рис. 4.4. Подготовка для присоединения к проекту нескольких файлов

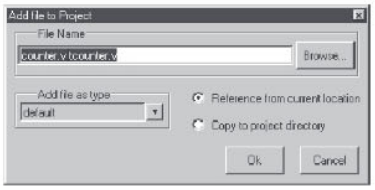


Рис. 4.5. Диалоговое окно «Add file to Project»

6. Откомпилировать проект, щелкнув правой кнопкой мыши на секции Project, и выбрать **Compile All** (рис. 4.6).

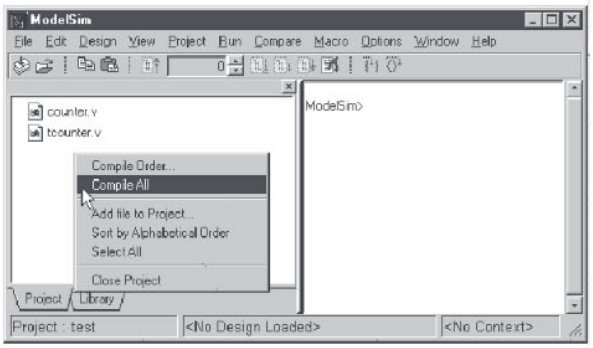


Рис. 4.6. Выполнение компиляции файлов проекта

7. Для просмотра результатов компиляции щелкнуть по ярлычку Library. Таким образом, можно увидеть скомпилированные объекты проектирования (design units) (рис. 4.7).

8. Загрузить объект проектирования двойным щелчком по соответствующему имени (например, counter) в секции Library (рис. 4.8). Это приведет к появлению новой секции в рабочем пространстве Workspace — sim.

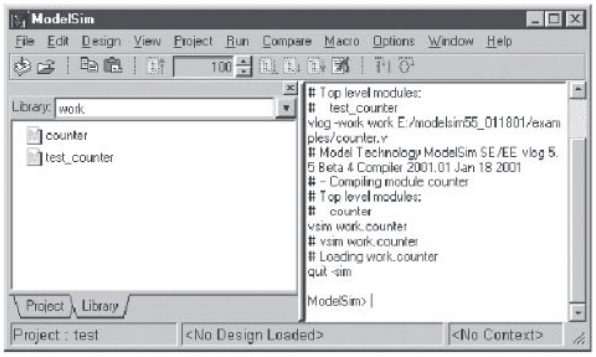


Рис. 4.7. Откомпилированные обьекты проектирования

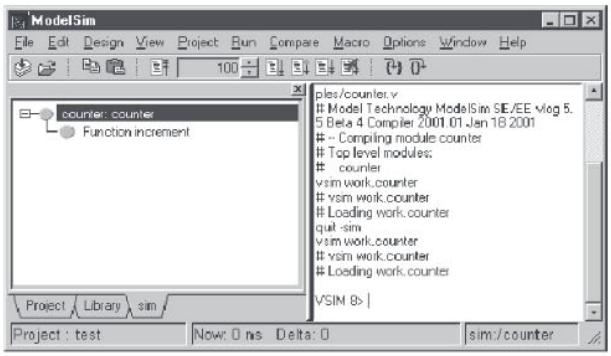


Рис. 4.8. Создание новой секции рабочего пространства «sim»

В данной секции отображается внутренняя структура загруженного объекта проектирования — counter. Сейчас уже возможно запустить процесс моделирования, проанализировать его результаты и отладить их. Для прекращения же процесса моделирования выбираем **Design > End Simulation**, и если вслед за этим хотим закрыть проект, то необходимо сделать следующее: **File > Close > Project**.

4.3. Моделирование на языке VHDL

Здесь мы обсудим следущие вопросы:
1. создание библиотеки;
2. компилирование VHDL-файла;
3. загрузка симулятора (simulator);
4. основные окна системы моделирования ModelSim;
5. работа с мышью и системой меню;
6. запуск ModelSim с помощью команды **run**;
7. работа со списком сигналов;
8. просмотр временных диаграмм (waveforms);
9. задание значений для сигналов;
10. запуск моделирования в пошаговом режиме;

11. установка точек останова (breakpoints).

Начнем с создания новой директории для нашего примера. Создадим директорию и затем скопируем в нее VHDL-файлы (*.vhd) из директории **\<install_dir>\modeltech\examples** в новую директорию.

Сделаем новую директорию текущей. Для этого можно запустить ModelSim из новой директории или выбрать в меню основного окна системы моделирования команду **File > Change Directory**.

Перед компиляцией VHDL-кода мы нуждаемся в библиотеке проектирования (design library) для хранения откомпилированных файлов. Для создания такой библиотеки в меню главного окна системы выберем

Design > Create a New Library.

Затем в появившемся диалоговом окне (рис. 4.9) выберем опцию (PROMPT: vlib work vmap work work):

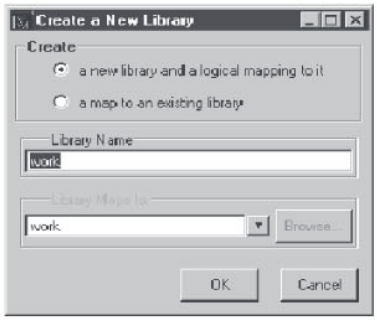


Рис. 4.9. Диалоговое окно создания новой библиотеки

a new library and a logical mapping to it. В поле **Library name** наберем **«work»** и нажмем клавишу **OK**. Тем самым создана поддиректория work (наша design library) в текущей директории. Система моделирования ModelSim будет сохранять в этой поддиректории специальный файл _info. Не следует пытаться создавать директорию для библиотеки, используя команды Windows, так как специальный файл _info в этом случае не будет создаваться. Для создания такой директории следует использовать меню **Design** или команду **vlib** в VSIM prompt или DOS prompt.

Скомпилируем файл counter.vhd в новой библиотеке — начнем с выбора клавиши **Compile** в линейке инструментов главного окна  . (PROMPT: vcom counter. vhd)

Это приведет к открытию диалогового окна **Compile HDL Source Files** (рис.4.10).

(Вместо открытия этого диалогового окна можно использовать команду vcom из командной строки (см. выше).)

В открывшемся диалоговом окне выберем counter .vhd в списке имен файлов и нажмем клавишу Compile. Можно скомпилировать несколько файлов в течение одного сеанса моделирования, последовательно выбрав нужные имена.

Сейчас загрузим скомпилированный объект проекта — counter. Для этого используется клавиша Load Design, нажатие которой открывает диалоговое окно Load Design  (рис. 4.11).

Альтернативным путем открытия диалогового окна Load Design является использование команды **vsim** с именем counter в командной строке (см. выше).

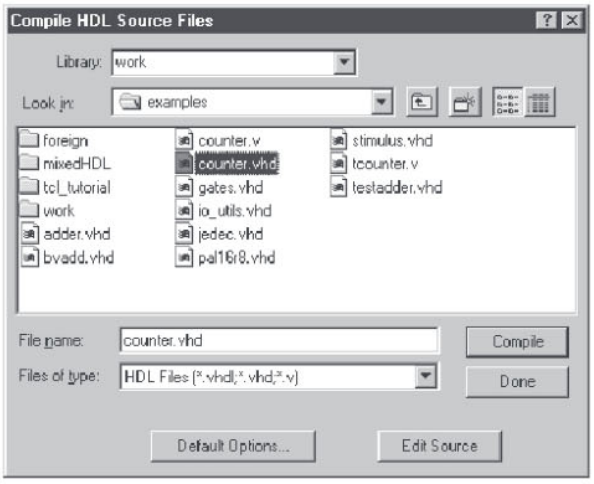


Рис. 4.10. Диалоговое окно Compile HDL Source Files

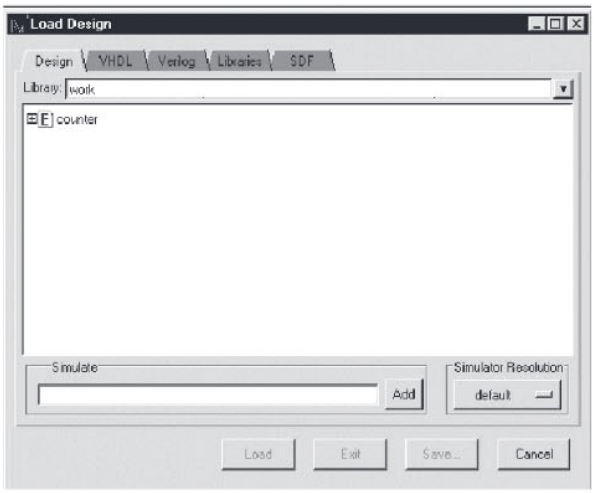


Рис. 4.11. Диалоговое окно Load Design

Диалоговое окно Load Design позволяет выбрать библиотеку и объект проекта верхнего уровня (top-level design unit) для выполнения процесса моделирования. Есть возможность также выбрать предел разрешения (resolution limit) для моделирования (по умолчанию — 1 ns).

Таким образом, процесс моделирования использует следующие установки:

- Simulator Resolution : default (по умолчанию — 1 ns);
- Library : work;
- Design Unit : counter.

Щелкнув по соответствующему ярлычку со знаком «+» (рис. 4.12), можно увидеть все связанные с entity архитектурные тела.

Рис. 4.12. Диалоговое окно Compile HDL Source Files

Для принятия установок процесса моделирования щелкнем по клавише Load.

Выберем **View > All** в меню главного окна системы для открытия всех окон ModelSim (альтернативный путь: PROMPT : view).

Из меню окна **signals** выбираем **View > List > Signals in Region**. Эта команда отображает сигналы верхнего уровня (top-level signals) в окне списка (рис. 4.13). То же самое можно проделать с помощью команды PROMPT: add list/counter/*.

Теперь добавим эти сигналы в окно Wave, выбрав **View > Wave > Signals in Region** в меню окна **signals** или использовав команду **add**:

PROMPT : add wave/counter/*.

Начинаем процесс моделирования с задания входных воздействий на входе синхроимпульсов clk. Для этого, щелкнув по панели VSIM, вводим следующую команду в VSIM prompt:

force clk 1 50, 0 100 — repeat 100.

То же самое можно осуществить с помощью главного меню

MENU: Signals > Edit > Clock,

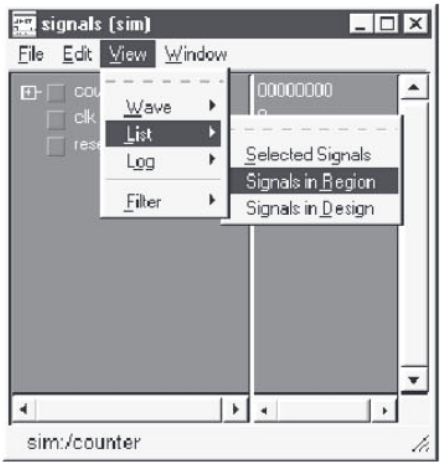


Рис. 4.13. Диалоговое окно signals

задав затем соответствующие параметры во всплывающем диалоговом окне. ModelSim интерпретирует эту команду **force** следующим образом:

- вход clk устанавливается в единицу спустя 50 ns после начала моделирования и переходит в состояние нуль спустя 100 ns после начала моделирования;
- вышеописанный фрагмент входных воздействий должен быть повторен с периодом 100 ns.

Существуют две различные функции Run, связанные с клавишами линейки инструментов в главном окне системы Main или в окне Wave

 **Run.** При нажатии клавиши Run процесс моделирования и затем, через 100 ns останавливается (PROMPT: run 100) (MENU: Run > Run 100 ns)

 **Run-All.** При нажатии клавиши Run-All моделирование выполняется без остановки, и чтобы прервать этот процесс, необходимо нажать клавишу **Break**.

(PROMPT: run −all) (MENU: Run > Run −All)

 **Break.** Для прерывания процесса моделирования необходимо использовать клавишу Break в главном окне системы или в окне Wave. При этом всплывает окно **source** (рис. 4.14). Стрелка в этом окне **source** указывает на следующий оператор языка VHDL (HDL statement), который будет выполняться.

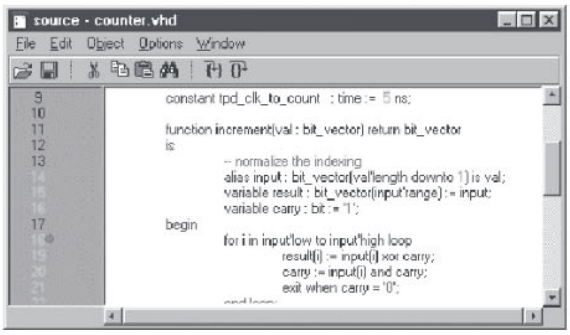


Рис. 4.14. Диалоговое окно source

Теперь установим точку останова на строке 18 окна **source**. Для этого надо переместить курсор на окно **source** и «прокрутить» это окно вертикально до тех пор, пока строка 18 не станет видна. Щелкнем на строке 18 для установки точки останова (breakpoint) — при этом увидим красную точку на строке, где установлена точка останова. Точка останова может становиться недоступной, если щелкнуть по ней: при этом вокруг нее появляется кружок. Новый щелчок делает ее вновь доступной. Чтобы удалить точку остановка, надо щелкнуть по номеру строки правой кнопкой мыши и выбрать **Remove Breakpoint 18**. (Заметим, что точки останова могут быть установлены лишь на строках.)

 **Continue Run.** Выбрав кнопку **Continue Run**, можно возобновить выполнение процесса моделирования. Если система ModelSim вновь попадет в точку останова согласно стрелке в окне **source**, сообщение об остановке процесса моделирования появится в главном окне системы.

(PROMPT: run −continue)(MENU: Run > Continue).

 **Step.** Щелкнув по клавише **Step**, можно осуществить пошаговое выполнение процесса моделирования. Заметим, что при этом изменяются значения переменных в окне Variables. (PROMPT: run−step) (PROMPT: step).

Когда процесс моделирования завершен, можно выйти из системы с помошью команды **quite-force**.

4.4. Отладка проекта на языке VHDL

В этом разделе мы обсудим следующие вопросы:

– создание теста на языке VHDL (VHDL testbench), который является архитектурным телом (VHDL architecture), позволяющим тестировать проект на языке VHDL, обеспечивая внешние воздействия для выполнения моделиро-

вания (simulation stimuli) и проверку выходных реакций объекта проекта на эти воздействия;
- размещение имени логической библиотеки (logical library name) в реальной библиотеке (actual library);
- изменение заданной по умолчанию длины интервала моделирования (default run length);
- распознавание сообщения оператора контроля (assertion messages) в командном окне (command window);
- изменение уровня прерывания для оператора контроля (assertion break level);
- перезапуск процесса моделирования при использовании команды **restart**;
- проверка составных типов данных (composite types), отображаемых в окне переменных проекта (**Variable window**);
- изменение значения переменной;
- использование строба (strobe) для инициирования строки в окне списка (List window);
- изменение разрядности сигналов, отображаемых в окне списка.

Для подготовки к моделированию необходимо создать новую директорию для нашего примера и скопировать следующие файлы из

\<install_dir>\modeltech\examples

- gates.vhd
- adder.vhd
- testadder.vhd

в новую директорию. Далее, необходимо убедиться, что новая директория является текущей директорией. Этого можно добиться, запустив ModelSim из новой директории или используя команду **File > Change Directory** в меню главного окна ModelSim.

Для создания новой библиотеки введем команду на панели VSIM prompt: **Vlib library_2** и скомпилируем исходные файлы в новой библиотеке вводом следующей команды:

vcom −work library_2 gates.vhd adder.vhd testadder.vhd.

Теперь разместим новую директорию в рабочей библиотеке. Это можно сделать, отредактировав Library section в файле modelsim.ini или создав имя логической библиотеки с помощью команды **vmap** (тогда система модифицирует сама этот файл).

Теперь запустим систему моделирования выбором **Design > Load Design** из меню главного окна или щелкнув по иконке Load Design. Вслед за этим появляется окно диалога Load Design (рис. 4.15).

В этом окне диалога мы выполним следующие действия:
- убедимся, что единица времени моделирования (simulator resolution) соответствует единице, устанавливаемой по умолчанию (default resolution), — наносекунде (ns);
- выберем конфигурацию test_adder_structural;
- щелкнем по кнопке Load для подтверждения сделанных установок (settings)

(PROMPT: vsim −t ns work.test_adder_structural).

Теперь откроем все окна системы ModelSim, введя следующую команду в VSIM prompt:

view *

(Main MENU: View > All).

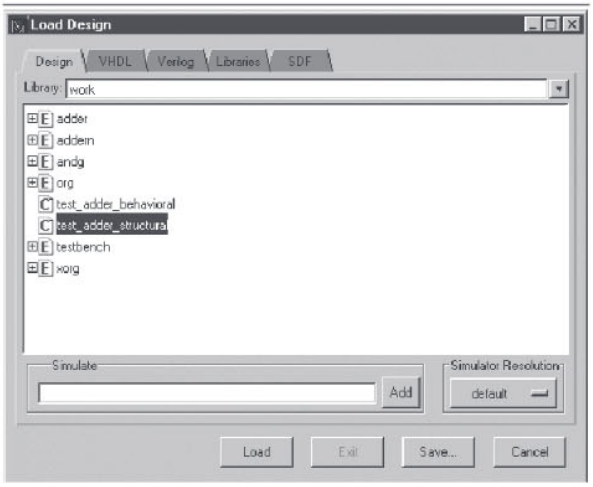


Рис. 4.15. Диалоговое окно Load Design

Перетащим сигналы верхнего уровня проекта (top-level signals) в окно списка (List window) следующим образом:

- убедимся, что иерархическая структура не развернута (нет квадратиков со знаком «–» (no minus boxes);
- выберем все сигналы в окне сигналов (Signal window):

(Signals MENU: **View > List > Signals in Region**) (PROMPT: **add list ***)

и тогда перетащим выбранные ранее сигналы в окно списка.

Запишем также сигналы верхнего уровня в окно временных диаграмм (Wave window) вводом команды

add wave *

(Signals MENU: **View > Wave > Signals in Region**) (DRAG&DROP).

Заменим интервал времени моделирования по умолчанию на интервал в 1000 ns с помощью селектора в линейке инструментов главного окна (run length selector) (заметим, что две стрелки (рис. 4.16) позволяют изменить интервал времени моделирования — run length):

(Main MENU: **Options > Simulations > Defaults**).

Запустим систему моделирования, выбрав клавишу Run в главном окне.

 (PROMPT: run)

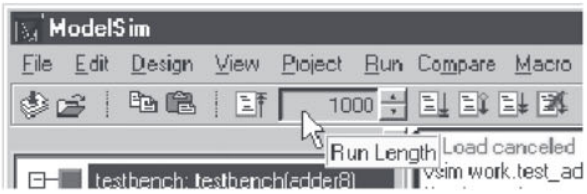


Рис. 4.16. Изменение интервала времени моделирования

Сообщение в главном окне, которое мы получили, свидетельствует о наличии ошибки (рис. 4.17). Наша задача — обнаружить причину этой ошибки. Для этого выполним следующие действия.

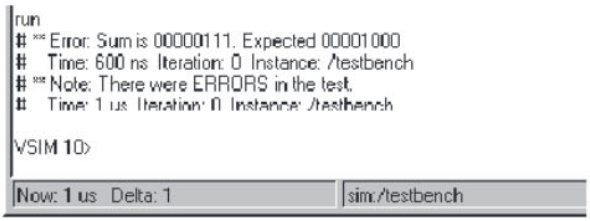


Рис. 4.17. Сообщение об ошибке

Вначале изменим опции оператора контроля процесса моделирования. Для этого выберем **Options > Simulation** из меню главного окна системы моделирования.

Это приводит к появлению диалогового окна **Simulations Options** (рис. 4.18).

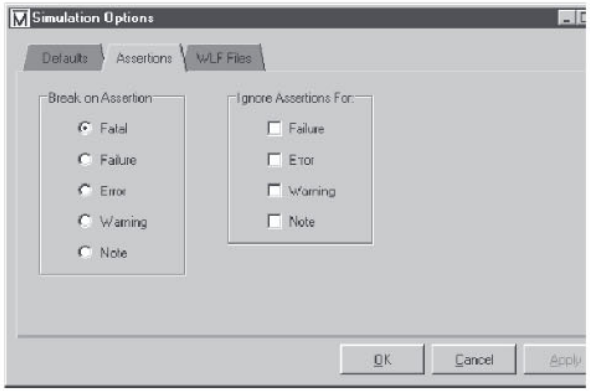


Рис. 4.18. Диалоговое окно Simulation Options

Выберем в данном окне закладку Assertions. Изменим выбор в группе Break on Assertion (Останов по оператору контроля)-Fatal (Фатальная ошибка) на Error (Ошибка) и нажмем клавишу OK.

Вновь запустим процесс моделирования, выбрав клавишу **Restart**  в главной линейке инструментов:

(Main MENU: **File** > **Restart**) (PROMPT: **restart**).

Убедимся, что все пункты в диалоговом окне **Restart** (рис. 4.19) отмечены и нажмем клавишу **Restart**.

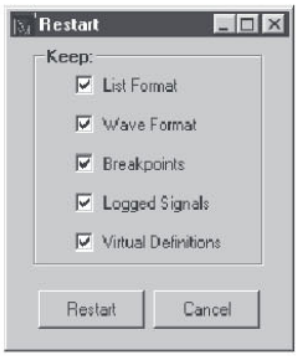


Рис. 4.19. Диалоговое окно Restart

Теперь из линейки инструментов главного окна системы выбираем клавишу **Run**  :

(Main MENU: Run > Run 1000 ns) (PROMPT: run).

Заметим, что на этот раз в момент фиксации ошибки всплывает диалоговое окно **source**, в котором стрелка указывает на оператор контроля, который является источником сообщения о данной ошибке (рис. 4.20).

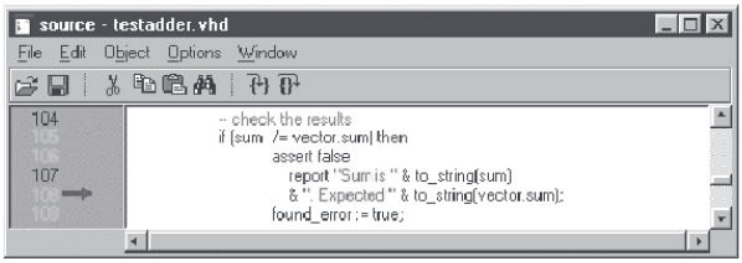


Рис. 4.20. Диалоговое окно source с указанием на источник сообщения об ошибке

Сейчас в окне переменных (Variables window) можно видеть, что i = 6. Это означает, что процесс моделирования остановлен на шестой итерации теста (test pattern's loop) (рис. 4.21).

Просмотрим переменную test_patterns, щелкнув по квадратику «+», а затем и шестую запись в этом массиве (test_patterns(6)), вновь щелкнув по соответствующему квадратику «+» (рис. 4.22).

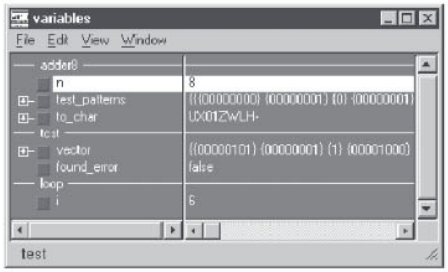


Рис. 4.21. Диалоговое окно variables

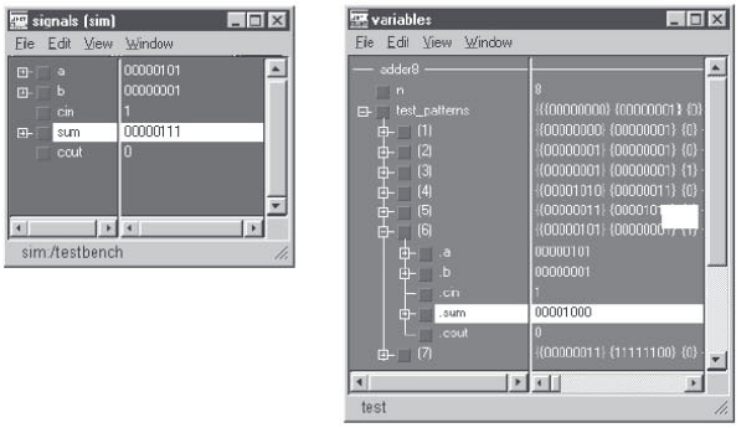


Рис. 4.22. Диалоговые окна signals и variables, используемые в процессе отладки

Оператор контроля ASSERT показывает (рис. 4.20), что значение сигнала **sum** не равно значению поля **sum** в поле переменных. Заметим, что сумма входов **a**, **b** и **cin** должна быть равной содержимому поля, т. е. налицо ошибка в тестовом векторе.

Отредактируем содержимое тестового вектора, выбирая поле **sum** записи щелчком по имени переменной и **Edit > Change** в меню окна. Это приведет к возникновению диалогового окна **change** (рис. 4.23)

Выберем четыре последних бита (1000) в этом поле и заменим это содержимое на 0111, щелкнув затем по кнопке **Change**.

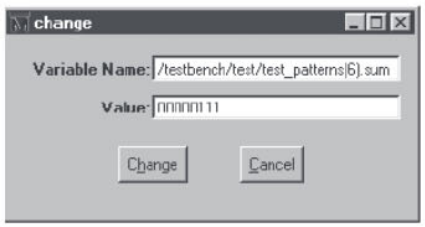


Рис. 4.23. Диалоговое окно change

Теперь выберем клавишу **Run**  в линейке инструментов главного окна и вновь запустим процесс моделирования. Главным итогом является то, что сейчас (Main MENU: **Run > Run 1 us**) (PROMPT: run) процесс моделирования выполняется без ошибок (рис. 4.24).

По умолчанию в окне List window выводится информация о каждой смене состояния сигналов. Следующие действия изменят такое поведение системы моделирования. Например, мы хотим, чтобы значения сигналов выводились на экран каждые 100 ns.

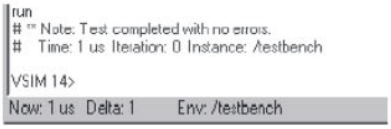


Рис. 4.24. Панель VSIM с сообщением о том, что тест выполняется без ошибок

Для этого в меню окна List window выберем **Prop > Display Props**. Это приведет к появлению диалогового окна **Modify Display Properties (list)** (рис. 4.25).

В этом диалоговом окне выбираем закладку **Triggers** и выполняем следующие шаги:

- отменить опцию **Trigger On: Signals**, чтобы не позволить отображать каждое изменение значения сигналов;
- выбрать опцию **Trigger On: Strobe**, чтобы позволить отображение значений сигналов в соответствии с регулярно повторяющимся сигналом строба;
- ввести 100 в поле **Strobe Period**;
- ввести 70 в поле **First Strobe at**;
- щелкнуть по клавише OK для принятия сделанных установок.

Последнее действие состоит в изменении системы счисления на десятичную для сигналов **a**, **b**, **sum**. Для этого выберем в меню **Prop > Signal Prop**. Это приведет к открытию диалогового окна **Modify Signal Properties (list)**. В окне списка (List window) выберем сигнал, свойства которого мы хотим изменить и затем перейдем к открывшемуся окну диалога (рис. 4.26).

Выполним следующие изменения свойств сигналов:

– выберем сигнал **a**, затем щелкнем по клавишам **Decimal** и **Apply**;
– выберем сигнал **b**, затем щелкнем по клавишам **Decimal** и **Apply**;
– выберем сигнал **sum**, затем щелкнем по клавишам **Decimal** и **OK**.

Завершая сеанс моделирования, выходим из системы, введя следующую команду:

quit-force.

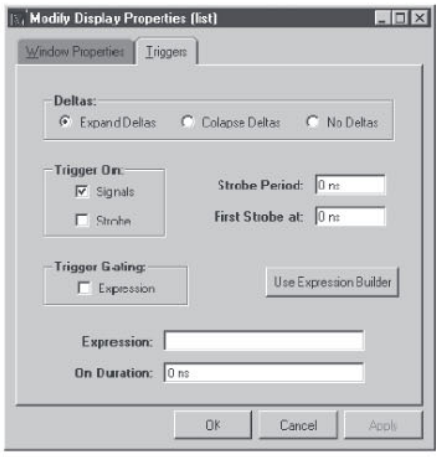


Рис. 4.25. Диалоговое окно Modify Display Properties (list)

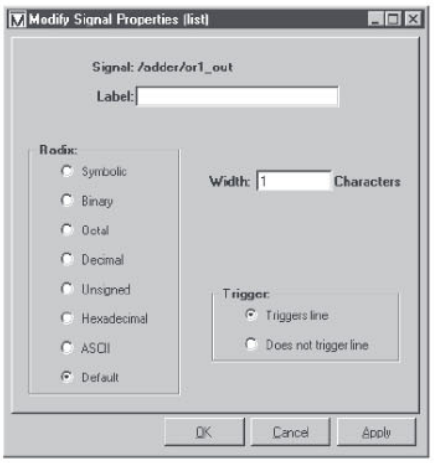


Рис. 4.26. Диалоговое окно Modify Signal Properties (list)

4.5. Запуск моделирования с помощью командных файлов

Здесь мы обсудим, как запустить процесс моделирования с помощью командных файлов (batch-mode VHDL simulation); как выполнить макро-файл (*.do) и как просмотреть сохраненную историю моделирования.

Режим моделирования с помощью командных файлов (batch-mode simulation) позволяет выполнить несколько команд моделирования, которые записаны в некотором текстовом файле. При этом мы создаем текстовый файл со списком команд, которые подлежат выполнению в процессе моделирования. Такой режим полезен, когда мы хотим выполнить процесс моделирования с одним и тем же фиксированным набором команд (возможно, в циклическом режиме).

Моделирование с помощью командных файлов выполняется в окне MS DOS prompt. Здесь нам предстоит сделать следующее.

1. Создадим новую директорию и сделаем ее текущей директорией. Затем скопируем наш файл с кодом VHDL в эту директорию:

   \<install_dir>\modeltech\examples\counter.vhd.

2. Создадим новую библиотеку проекта:

   vlib work.

3. Разместим эту библиотеку в рабочей директории:

   vmap work work.

4. Скомпилируем исходный файл:

   vcom counter.vhd.

5. Будем использовать макрофайл для хранения входных воздействий (stimulus). Такой файл для нашего примера находится по следующему адресу:

   <install_dir>\modeltech\examples\stim.do.

6. Создадим командный файл, используя текстовый редактор. Введем следующие три строки и сохраним их в файле с именем **yourfile** в текущей директории:

   add list -decimal *

   do stim.do

   write list counter.lst

7. Для запуска процесса моделирования в этом режиме введем

   vsim −wlf saved.wlf counter < yourfile.

Это означает:

- запуск системы для моделирования единицы проекта **counter**;
- система сохраняет результаты моделирования в лог-файле **saved.wlf**;
- для значений сигналов и переменных в процессе моделирования будет использоваться десятичная система счисления;
- для проведения моделирования будут использоваться тестовые воздействия, хранящиеся в файле **stim.do**;
- результаты моделирования будут сохраняться в файле **counter.lst**.

8. Сохраненные в файле **saved.wlf** результаты моделирования просмотрим с помощью переключателя **view**:

   vsim -view saved.wlf.

9. Для этого необходимо открыть соответствующие окна системы с помощью выбора пункта **View** в главном окне (Main window) или с помощью эквивалентной команды

   view signals list wave.

(Заметим, что если мы откроем на этом этапе окно процесса или окно переменных, то обнаружим, что они пусты. Это означает, что можно просматривать сохраненные результаты моделирования (например, временные диаграммы в окне **wave**), но нет возможности проверять их интерактивно.)

10. Для просмотра всех сигналов в окнах wave и list введем две следующие команды:

    add wave *;

    add list *.

11. Для выхода из системы моделирования необходимо ввести команду

    quit-f.

4.6. Выполнение команд при начальной загрузке системы

В данном разделе обсудим следующие детали:

- как специфицировать единицу проекта;
- как отредактировать файл modelsim.ini;
- как выполнить команды при начальной загрузке с помошью макрофайла (*.do).

1. Будем использовать макрофайл для обеспечения информации о действиях в момент начальной загрузки системы. Этот файл надо скопировать в текущую директорию из директории инсталляции системы:

   \<install_dir>\modeltech\examples\startup.do.

2. Отредактируем файл modelsim.ini для спецификации команд, которые будут выполняться после загрузки проекта. Для этого откроем этот файл в текстовом редакторе и допишем еще одну строку

   Startup = do startup.do,

   а затем сохраним файл modelsim.ini.

3. Просмотрим макрофайл, который должен использовать предопределенную переменную **Sentity** для выполнения различных действий при запуске различных проектов.

4. Запустим систему моделирования и специфицируем единицу проекта верхнего уровня для моделирования вводом следующей команды:

   vsim counter.

Заметим, что система моделирования загружает единицу проекта без использования диалогового окна **Load Design**. Это удобно, если приходится моделировать тот же объект неоднократно.

Отметим также, что для открытия всех окон системы для отображения процесса моделирования необходимо ввести команду **view *** в макрофайл.

5. Для выхода из системы моделирования используем команду

   quit-f.

6. В случае, если миновала необходимость в использовании макрофайла, необходимо в текстовом редакторе закомментировать строку «Startup» в файле modelsim.ini.

4.7. Просмотр временных диаграмм

В данном разделе мы обсудим следующие вопросы:
- использование временного курсора;
- увеличение или уменьшение изображения временных диаграмм;
- использование в окне временных диаграмм (Wave window) комбинаций клавиш для быстрого вызова (shortcuts);
- комбинирование элементов данных в виртуальные объекты;
- создание и просмотр наборов данных.

Когда впервые появляется окно временных диаграмм, временной курсор находится в начале оси времени. Любой щелчок в этом окне приводит к тому, что курсор занимает текущую позицию мыши (рис. 4.27).

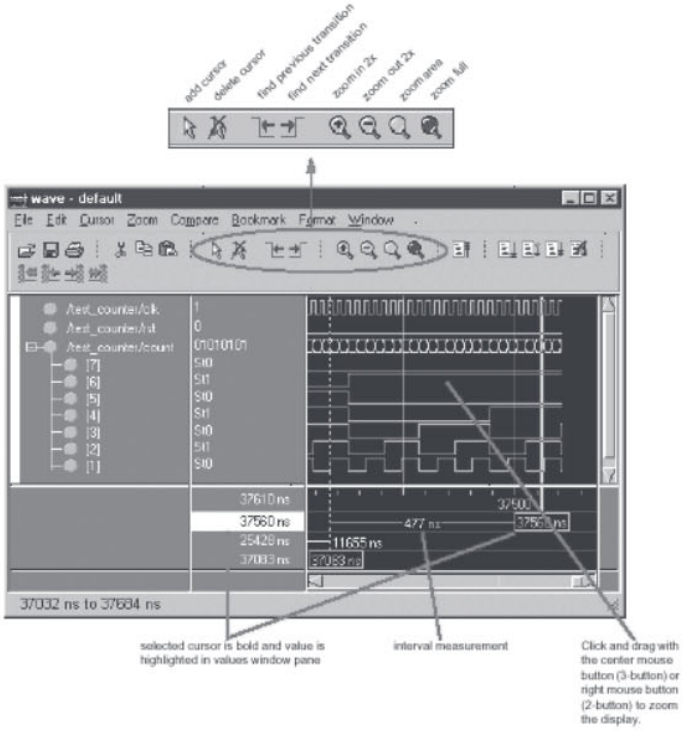


Рис. 4.27. Диалоговое окно wave

Есть возможность использовать до 20 курсоров в окне временных диаграмм, выбирая в верхнем меню **Cursor > Add Cursor** или нажимая соответствующую клавишу **Add Cursor** (рис. 4.27). Выбранный курсор рисуется жирной сплошной линией; все другие курсоры рисуются с помощью тонких штриховых линий. Можно

удалять курсоры, выбирая их и используя вехнее меню **Cursor > Delete Cursor** или с помощью клавиши **Delete Cursor** (см. выше).

Значения курсора (в списке **Goto**) соответствуют времени моделирования для этого курсора. Выбрать определенный курсор можно с помощью верхнего меню **Cursor > Goto**.

Каждый курсор отображается вместе с полем, содержащим соответствующее значение времени моделирования (time box). Это поле располагается в нижней части окна (рис. 4.27). Когда мы имеем более одного курсора, каждое такое поле появляется на отдельной дорожке внизу. Использование курсоров позволяет производить разнообразные временные измерения, например, ModelSim автоматически показывает разность между двумя смежными позициями курсора (рис. 4.27).

Если щелкнуть на изображении временных диаграмм, курсор немедленно переместится в выбранную позицию для мыши. Другой путь для позиционирования множества курсоров — использование мыши в дорожках полей временных значений внизу окна. Щелчок в любом месте дорожки заставляет соответствующий курсор переместиться в позицию мыши. Курсоры разработаны с привязкой к ближайшему краю импульса сигнала (налево), выбранному с помощью мыши. Чтобы модифицировать дистанцию привязки (snap distance), надо выбрать **Edit > Display Properties**. Можно также разместить курсор без привязки перетаскиванием курсора в область перед изменениями сигналов. Есть еще одна возможность работы с курсорами: перемещать курсор к следующему переходу сигнала в новое состояние (transition) с помощью двух следующих клавиш:

Теперь обсудим изменение масштаба изображения для временных диаграмм. Для этого можно использовать линейку инструментов окна или вызвать меню Zoom, щелкнув правой клавишей трехкнопочной мыши на панели временных диаграмм.

Опции меню Zoom включают:

- **Zoom Full**: перерисовывает изображение, чтобы показать всю картину моделирования от нуля до текущего момента моделирования;
- **Zoom In**: увеличивает вдвое резолюции и уменьшает видимый интервал по горизонтали;
- **Zoom Out**: уменьшение вдвое резолюции и увеличение видимого интер-вала по горизонтали;
- **Zoom Last**: восстанавливает изображение, которое было перед последней zoom-операцией;
- **Zoom Area**: используется для создания zoom-области;

with
Mouse Button 1

– **Zoom Range**: вызывает окно диалога, которое позволяет ввести начальное и конечное значения времени для интервала моделирования, подлежащего изображению.

Изменение интервала изображения можно выполнить с помощью клавиш линейки инструментов (рис. 4.28).

	**Zoom in 2x** zoom in by a factor of two from the current view		**Zoom area** use the cursor to outline a zoom area
	**Zoom out 2x** zoom out by a factor of two from the current view		**Zoom Full** zoom out to view the full 1 range of the simulation from time 0 to the current time

Рис. 4.28. Клавиши для выполнения операций zooming

Изменение интервала изображения можно выполнить с помощью мыши. Для этого необходимо позиционировать курсор мыши с левой стороны выбранного интервала изображения и нажать среднюю клавишу трехкнопочной мыши или правую клавишу двухкнопочной мыши. Пока клавиша нажата, «тащить» изображение вправо до достижения желаемого размера.

Существуют и комбинации клавиш быстрого вызова для выполнения операции zooming (рис. 4.29)

Key	Action
i I or +	zoom in
o 0 or –	zoom out
f or F	zoom full
l ur L	zoom last
r or R	zoom range
<arrow up>	scroll waveform display up
<arrow down>	scroll waveform display down
<arrow left>	scroll waveform display left
<arrow right>	scroll waveform display right
<page up>	scroll waveform display up by page
<page down>	scroll waveform display down by page
<tab>	searches forward (right) to the next transition on the selected signal
<shift-tab>	searches backward (left) to the previous transition on the selected signal
<Control-f>	opens the find dialog box; searches within the specified field in the pathname pane for text strings

Рис. 4.29. Клавиши и комбинации клавиш быстрого вызова (shortcuts) для выполнения операций zooming

Объединение (комбинирование) элементов, данных в окне **wave**, позволяет комбинировать сигналы в шины или группы. Используется выбор в меню **Edit** > **Combine** для вызова диалогового окна **Combine Selected Signals** (рис. 4.30). Шина является совокупностью сигналов, ассоциированных в определенном порядке для создания нового виртуального сигнала со специфическим значением.

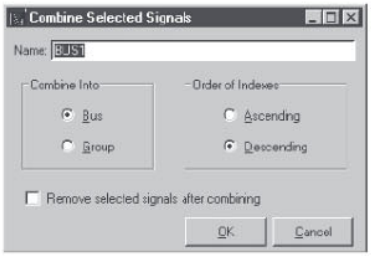


Рис. 4.30. Диалоговое окно Combine Selected Signals

На рис. 4.31 три сигнала комбинируются в виде новой шины, называемой BUS1. Заметим, что новая шина имеет значение, составленное из значений ее компонент (сигналов), упорядоченных в определенном порядке.

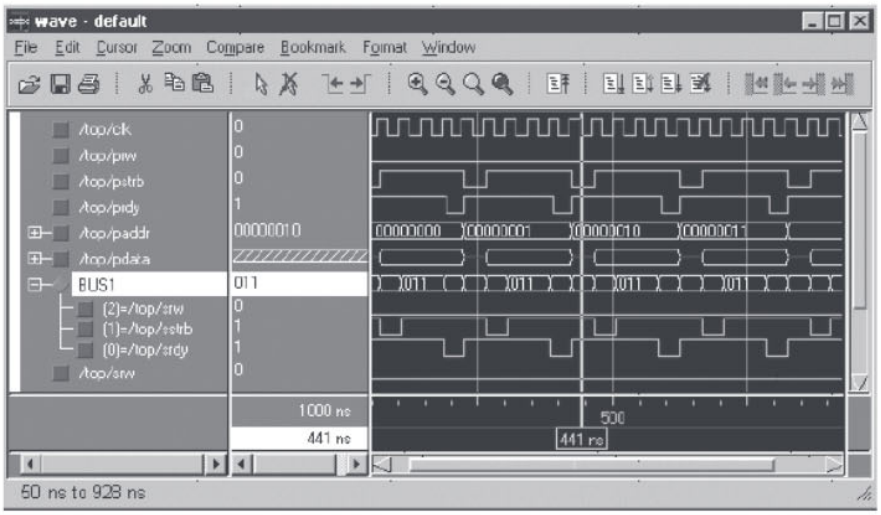


Рис. 4.31. Диалоговое окно wave с примером создания шины (Виртуальные объекты помечаются оранжевым ромбом.)

Глава 5. Работа с VHDL в среде системы моделирования Active-HDL

5.1. Введение в Active-HDL

Рассмотрим работу с VHDL-проектами в системе моделирования Active-HDL 3.5 фирмы ALDEC. Интегрированная среда Active-HDL базируется на стандарте (Microsoft Foundation Class) подобного GUI-интерфейса (рис. 5.1) (1а, 2а).

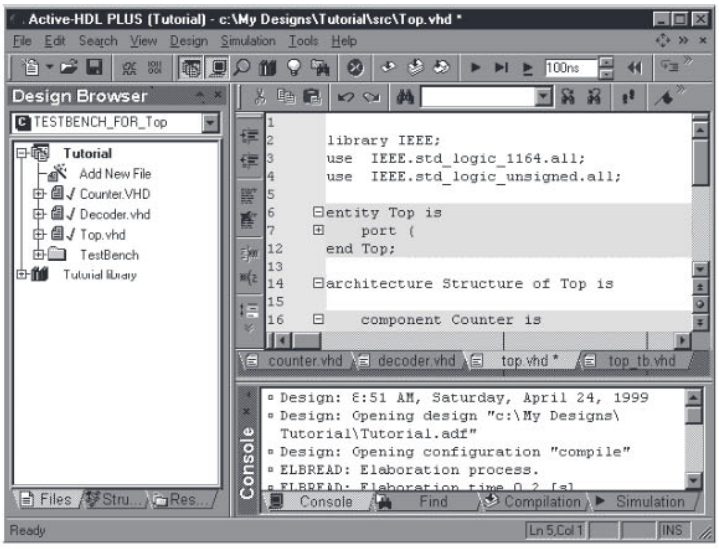


Рис. 5.1. GUI-интерфейс системы моделирования Active-HDL

Каждое окно является перемещаемым и фиксируемым диалоговым окном или окном приложения. Основные части интегрированной системы представлены следующими окнами:

- окно системы просмотра проекта (Design Browser);
- окно проводника по проекту (Design Explorer);
- окно текстового редактора (HDL Editor);
- окно редактора блок-диаграмм (Block Diagram Editor);
- окно редактора диаграмм состояний для цифровых автоматов (State Machine Editor);
- окно редактора временных диаграмм (Waveform Editor);
- окно консоли (Console);
- окно менеджера библиотек (Library Manager);
- окно потока данных (Data Flow);
- окно списка (List);
- окно стека вызовов (Call Stack);
- окно процессов (Processes);
- окно наблюдения (Watch).

Система моделирования Active-VHDL стартует с появления следующего логотипа (рис. 5.2):

Вслед за окончанием процесса загрузки системы появляется диалоговое окно **Getting Started**, приведенное на рис. 5.3. Выберем в этом окне опцию **Create new design** и щелкнем по кнопке **OK**. Затем в появившемся окне **New Design Wizard** (рис. 5.4) введем:

- имя проекта (design name);
- местонахождение его папки (design folder);
- имя рабочей библиотеки (working library name).

Для перехода к следующей странице нажмем кнопку **Next**.

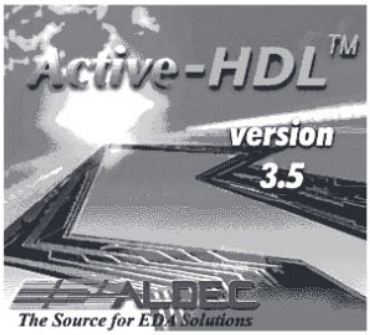


Рис. 5.2. Логотип Active-HDL PLUS Welcome Logo

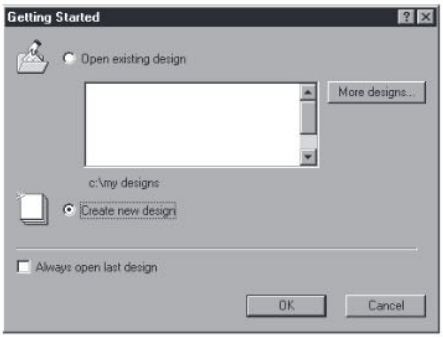


Рис. 5.3. Окно Getting Started

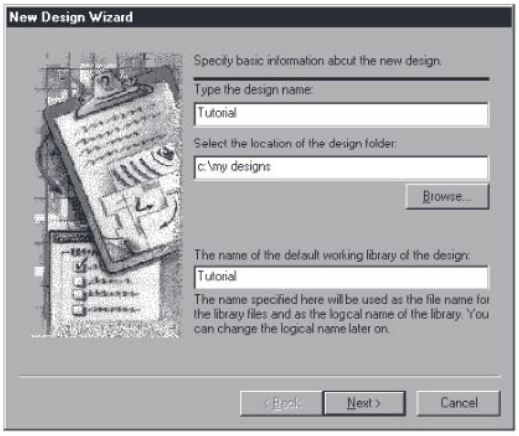


Рис. 5.4. Окно New Design Wizard

На следующей странице (рис. 5.5) разработчик имеет возможность выполнить следующие операции:

- создание новых файлов, содержащих VHDL-код специально для текущего проекта (Create new source files);
- добавление к текущему проекту уже существующих VHDL-файлов (Add existing resource files);

– добавление к текущему проекту списков соединений элементов (VHDL netlists), созданных системой проектирования Active-CAD фирмы ALDEC (Import a design from Active-CAD);
– создание пустого проекта (Create an empty design).

Щелкнув по кнопке **Next**, мы переходим к следующей странице нашего окна (Entity — рис. 5.6) и в ходе нового диалога добавим определенные компоненты к текущему проекту. Столбец **Source Type** позволяет нам выбрать описание в виде VHDL-кода или графическое описание, которое используется редактором моделей в виде цифровых автоматов **Finite State Machine Editor**.

Выбирая имя **Counter**, мы выбираем ресурс в виде VHDL-кода. Затем, нажав кнопку Ports, переходим к следующей странице окна **Design Wizard** (рис. 5.7), в которой получим возможность ввести описания портов интерфейса нашего проекта. Для ввода нового порта щелкнем по кнопке **New** и введем имя порта в поле **Name**. Можно также определить направление работы порта, используя управляющий элемент **Direction**. Для ввода шины мы имеем возможность определить ее размерность в поле **Range**.

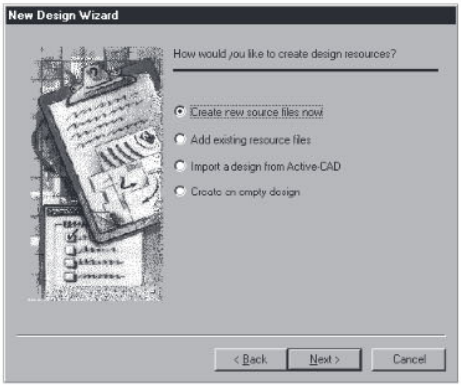


Рис. 5.5. Окно New Design Wizard — Resources

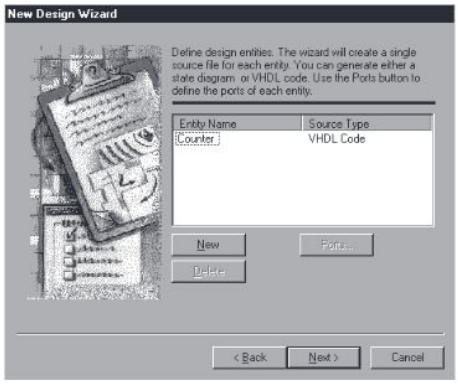


Рис. 5.6. Окно New Design Wizard — Entity

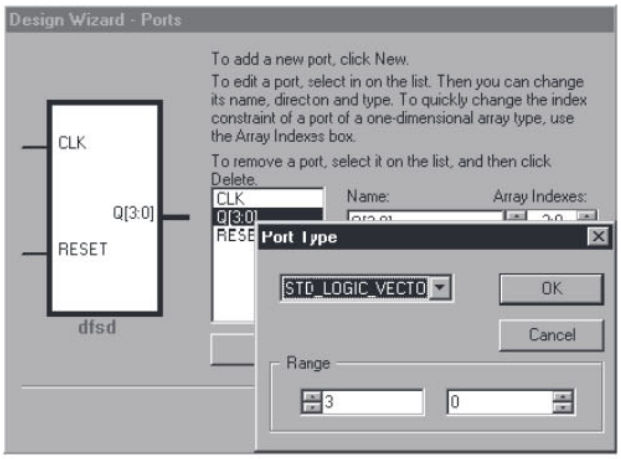


Рис. 5.7. Окно Design Wizard — Ports

Введем следующие три порта:

- CLK вход (in port);
- RESET вход(in port);
- Q[3:0] выходная шина, содержащая 4 линии (out bus port, range [3:0]).

Мы выбираем опцию **STD_LOGIC** для простого порта и опцию **STD_LOGIC_VECTOR** для шин. Эти действия присоединяют к нашему проекту IEEE Library и описания необходимых пакетов (packages declarations). Нажав на кнопку **Next**, переходим к последней странице текущего окна (рис. 5.8).

В случае, если все было сделано нами корректно и мы ни о чем не сожалеем, нажмем кнопку Finish.

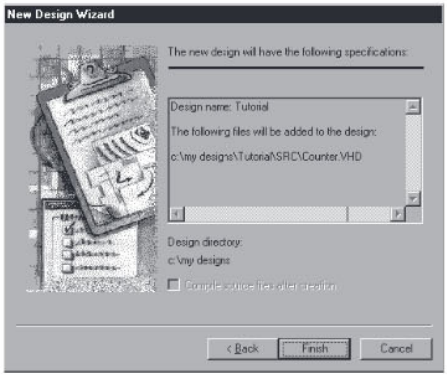


Рис. 5.8. Окно Design Wizard — Ports

5.2. Окно системы просмотра проекта (Design Browser)

С помощью окна **Design Browser** (рис. 5.9) можно просматривать содержимое нашего проекта (design contents). Как результат выполнения вышеописанных операций получим следующее содержимое проекта:

Из рис. 5.9, например, следует, что имя нашего проекта — **Tutorial**. На данной стадии проект содержит только один VHDL-файл: Counter.vhd и рабочую библиотеку **Tutorial library**. Компиляция VHDL-файла может быть выполнена командой **Compile** из меню **Design** или выбором одноименной команды из меню, всплывающего при щелчке над именем файла (рис. 5.10). После успешной компиляции появляется соответствующий значок в текущем окне (рис. 5.11). Щелкнув по маленькому крестику, мы видим, что в файле Counter.vhd находится описание интерфейса (сущность — entity) Counter и архитектурное тело Counter (рис. 5.12).

Рис. 5.9. Окно Design Browser

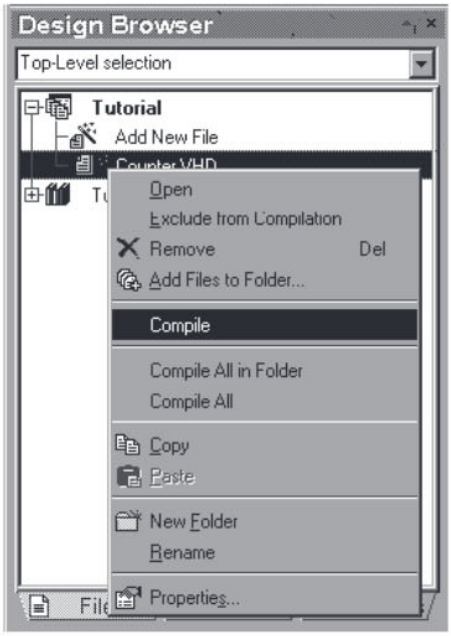


Рис. 5.10. Компиляция VHDL-файла

Таким образом, для моделирования у нас есть пара entity — architecture. Двойной щелчок по метке **Counter.vhd** открывает окно **HDL Editor**.

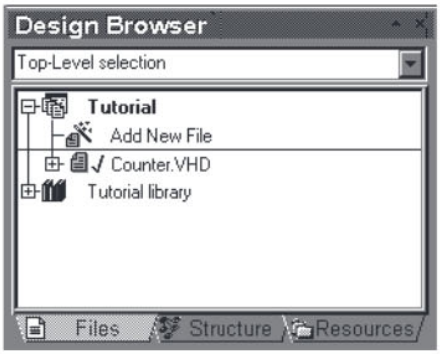


Рис. 5.11. Окно Design Browser после выполнения команды

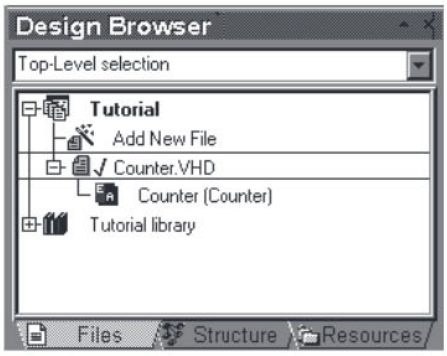


Рис. 5.12. Окно Design Browser с расширенной структурой проекта

5.3. Окно текстового редактора (HDL Editor)

Текстовый редактор выделяет специальным цветом ключевые слова VHDL и другие свойства кода для лучшего его понимания (рис. 5.13).

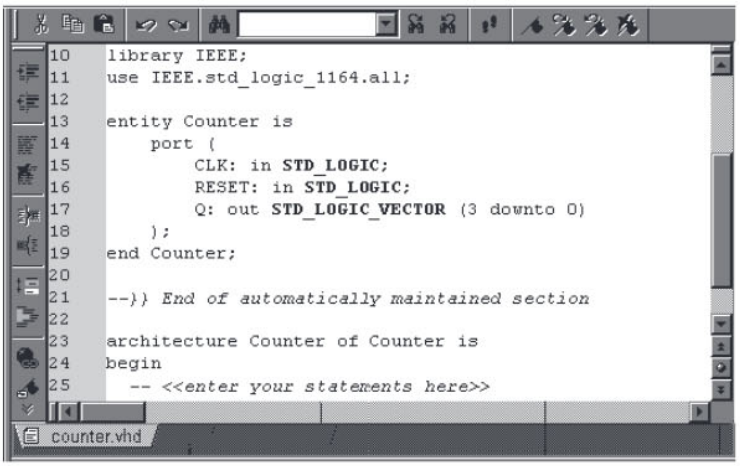


Рис. 5.13. Окно HDL Editor

Для определения структуры архитектурного тела Counter нажмем кнопку с изображением электрической лампочки в верхней части интерфейса системы. Тем самым мы вызываем окно **Language Assistant** (рис. 5.14).

В этом окне щелкнем вначале по метке **Tutorial** (рис. 5.15), а затем — по метке **Counter** (рис. 5.16).

В правой секции появившегося окна видно описание VHDL0-процесса, который является типовой моделью счетчика (BCD counter). Для вставки этого фрагмента в наш проект выполним следующие шаги.

1. Просмотреть окно **HDL Editor** и найти строку со следующим текстом:

<enter your statements here>.

2. Поместить курсор на следующую строку.
3. Выбрать метку **Counter** на дереве в окне **Language Assistant**.
4. Выбрать опцию **Use** из меню на рис. 5.16.
5. Фрагмент кода с моделью для BCD Counter поместить в выбранную на шаге 2 позицию.

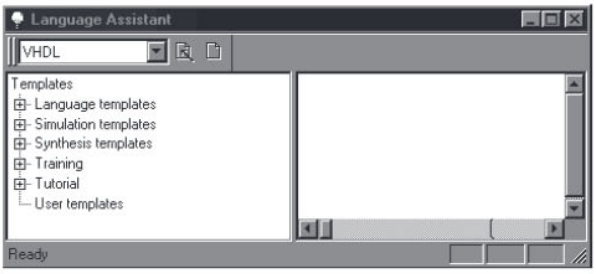


Рис. 5.14. Окно Language Assistant

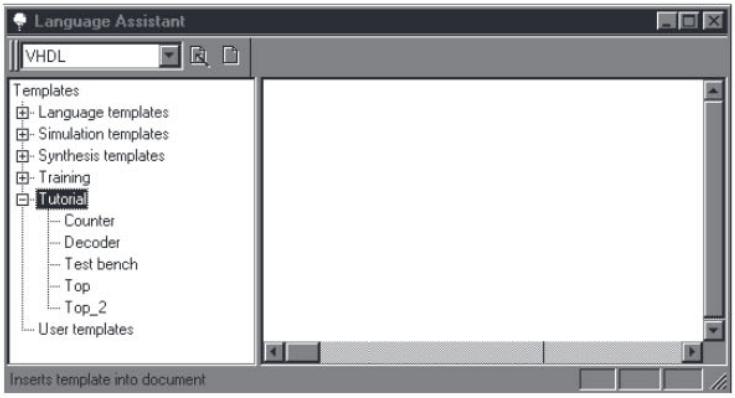


Рис. 5.15. Окно Language Assistant

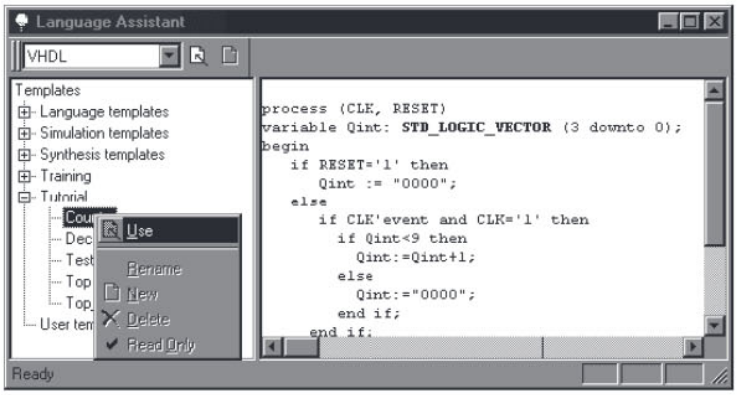


Рис. 5.16. Окно Language Assistant

Отредактированный код нуждается в присоединении соответствующих пакетов. Для примера добавим к созданному коду оператор *use IEEE.std_logic_unsigned.all.*

Полученный после выполнения описанных действий результат приведен на рис. 5.17:

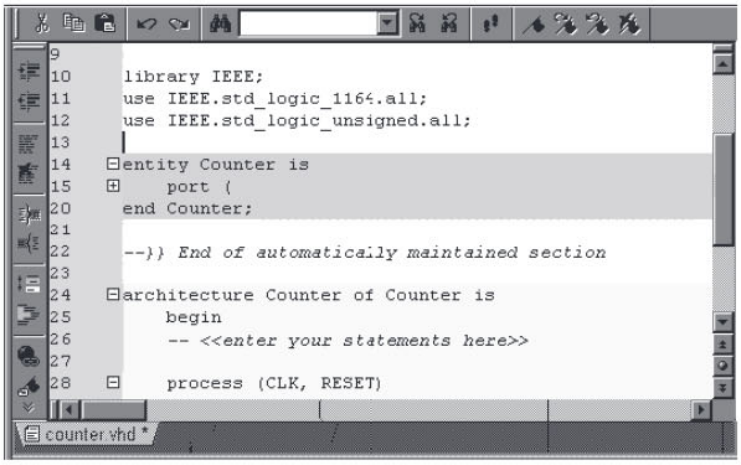


Рис. 5.17. Окно HDL Editor с кодом для Counter

Проверка синтаксиса выполняется выбором опции **Compile** в кратком меню в окне **Design Browser** (рис. 5.18). Если при компиляции обнаружено, что VHDL-файл содержит ошибки, специальный значок появляется у имени файла. В окне текстового редактора появляются отметки на месте ошибок и соответствующие разъяснения, а в окне **Console** мы находим список всех имеющихся в файле ошибок (рис. 5.19).

Видно, что в нашем случае надо исправить имя пакета и перекомпилировать VHDL-файл. Для добавления новых файлов к проекту необходимо выбрать пункт **New-VHDL Source** в меню **File** или щелкнуть метку **Add New File** на дереве окна **Design Browser** (рис. 5.20).

После выполнения этой команды появляется окно **New Source Wizard** (рис. 5.21):

В этом окне введем слово Decoder как имя следующего файла. Нажав на кнопку **Next**, перейдем к следующей странице этого окна (рис. 5.22).

Сейчас введем порты, как показано на рис. 5.23. Выберем тип данных **STD_LOGIC_VECTOR** для каждого порта.

Сейчас в окне **Design Browser** видны два различных проекта, находящихся в соответствующих VHDL-файлах: Counter.vhd и Decoder.vhd. Двойной щелчок по метке

Рис. 5.18. Краткое меню

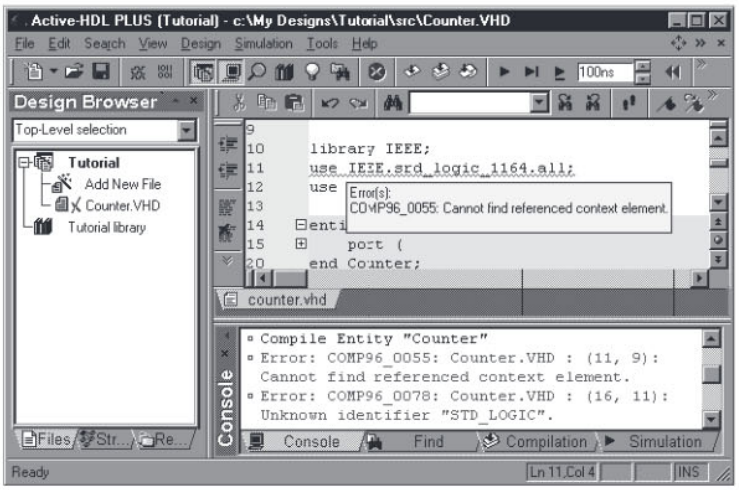


Рис. 5.19. Итоги работы компилятора после обнаружения ошибок

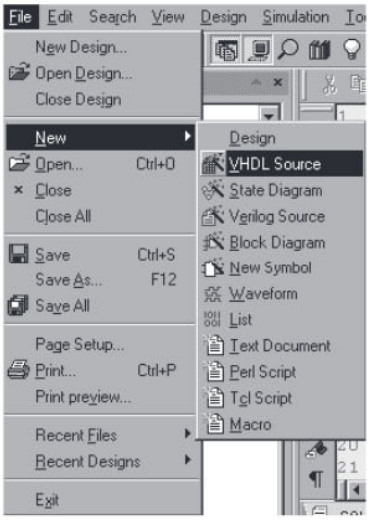


Рис. 5.20. Команда New-VHDL Source

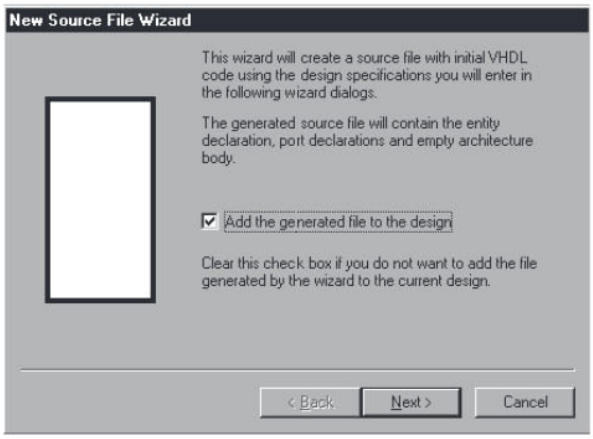


Рис. 5.21. Окно New Source Wizard

Decoder.vhd приводит к появлению в окне текстового редактора исходного VHDL-кода этого файла (рис. 5.24).

С помощью **Language Assistant** наполним соответствующим содержимым шаблон архитектуры для модели в файле Decoder.vhd, используя окно **Language Assistant** и выбирая в нем метку **Decoder** внутри папки **Tutorial** (рис. 5.25).

Затем поместим курсор внутрь упомянутого архитектурного тела, как показано на рис. 5.26, и применим опцию **Use** в кратком меню, возникающем при щелчке на метке **Decoder**.

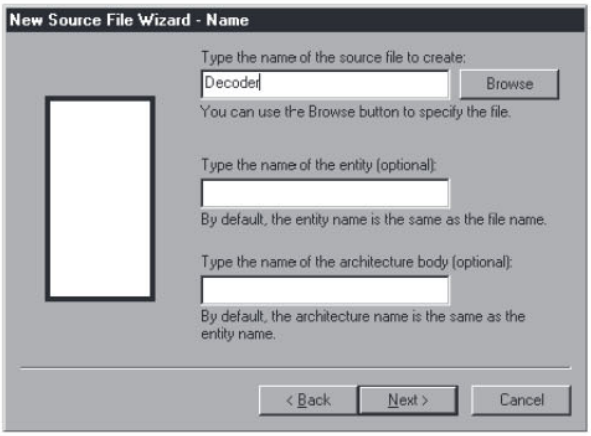


Рис. 5.22. Окно New Source File Wizard — Name

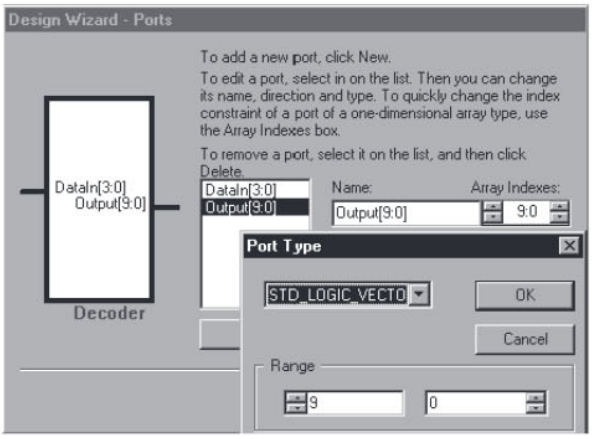


Рис. 5.23. Окно New Source Wizard — Ports

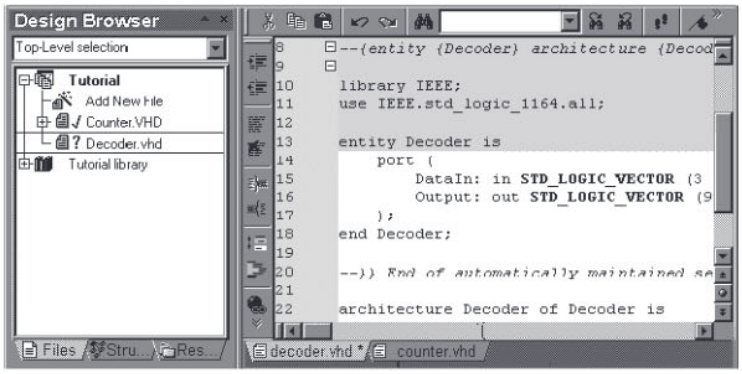


Рис. 5.24. Окна Design Browser и HDL Editor

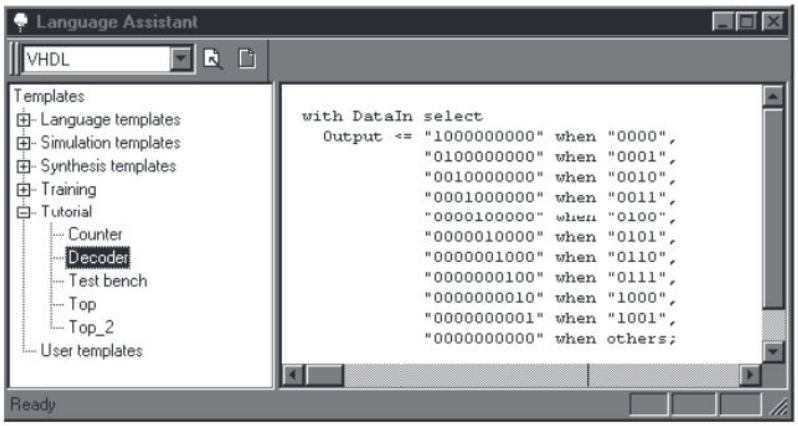


Рис. 5.25. Исходный код файла Decoder.vhd

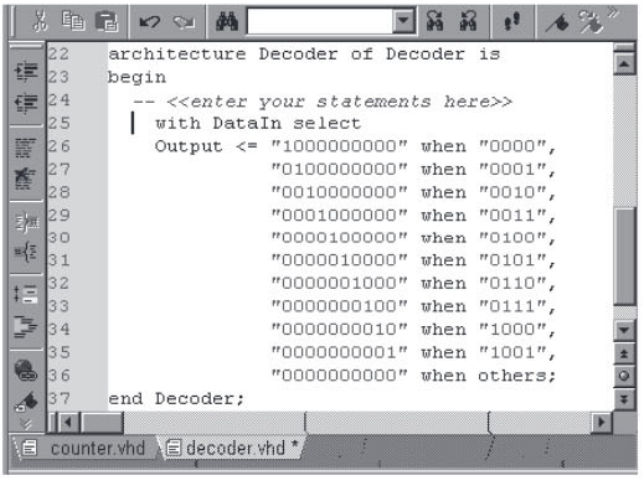


Рис. 5.26. Содержимое файла Decoder.vhd

5.4. Создание файла верхнего уровня проекта (Top Level File)

Выберем иконку **Add New File** в окне **Design Browser** и затем, во вновь открывшемся окне (рис. 5.27), введем имя нового файла **Top** и выберем иконку **VHDL Source Code**. Для получения архитектурного тела верхнего уровня проекта (Top Level architecture) выполним следующие действия.

1. Перейдем в окно **Language Assistant** и отыщем метку **Top** внутри папки **Tutorial**.
2. Выберем опцию **Use** из краткого меню для получения шаблона entity **Top** и соответствующей архитектуры **Structure**.

3. Перейдем в меню **File** и воспользуемся опцией **Save** для сохранения файла верхнего уровня проекта (рис. 5.28).

Содержание окна **Design Browser** после вышеописанных операций приведено на рис. 5.29, а содержание окна **HDL Editor** после генерации структуры и автоформатирования — на рис. 5.30. Теперь перекомпилируем наш проект с помощью опции **Compile All** во всплывающем меню при нажатии правой клавиши мыши (рис. 5.31).

После выполнения процесса компиляции в окне **Design Browser** можно наблюдать структуру нашего проекта с общим интерфейсом **Top** и архитектурным телом **Structure** (рис. 5.32). Видно, что наш проект содержит три VHDL-файла:

- Counter.vhd;
- Decoder.vhd;
- Top.vhd.

Выберем метку Top (Structure). Тем самым для выполнения в будущем процесса моделирования выбрана соответствующая пара: описание

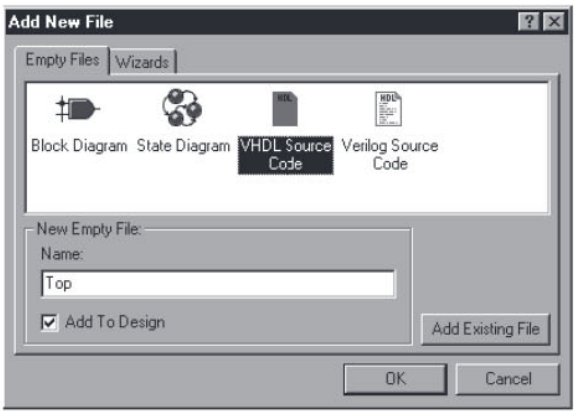


Рис. 5.27. Окно Add New File

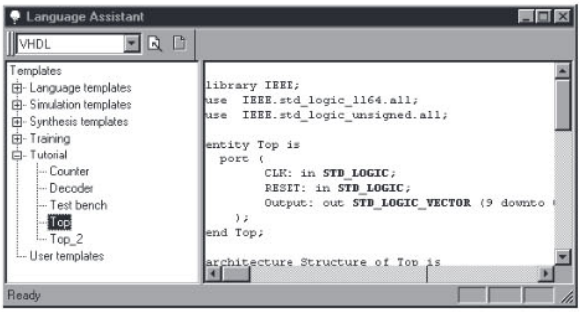


Рис. 5.28. Entity Top и архитектура Structure верхнего уровня

Рис. 5.29. Содержание окна Design Browser

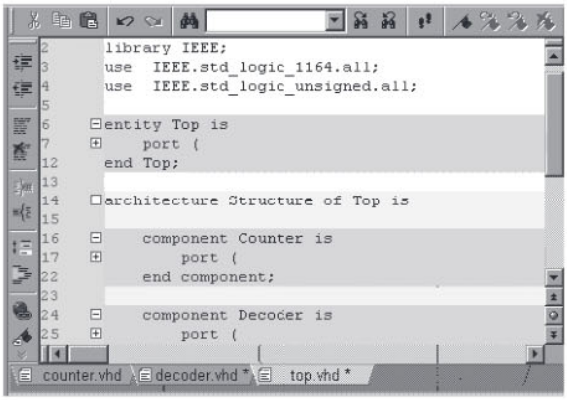


Рис. 5.30. Содержание окна HDL Editor

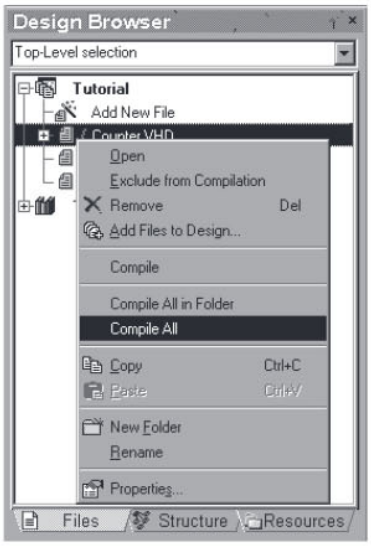


Рис. 5.31. Компиляция всех файлов проекта

Рис. 5.32. Выбор архитектуры верхнего уровня (Top Level Architecture)

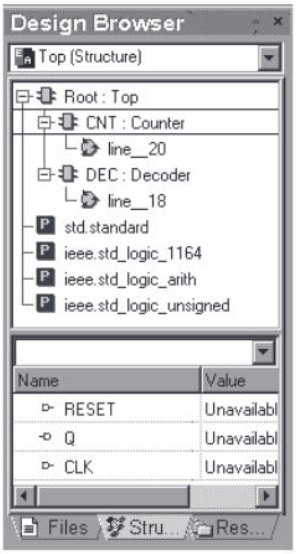


Рис. 5.33. Содержание закладки Structure

интерфейса на верхнем уровне **Top** и архитектурное тело верхнего уровня **Structure**. Выбрав закладку **Structure** в текущем окне, можно видеть подробную структуру проекта: проект **Top**, помеченный как Root, содержит два базовых компонента Counter(CNT) и Decoder(DEC). Код компонента Counter содержит 20 строк, а код компонента Decoder — 18 строк (рис. 5.33).

Отсюда же видно, что к проекту присоединены 4 стандартных пакета:

- standard из одноименной библиотеки;
- std_logic_1164;
- std_logic_arith;
- std_logic_unsigned из библиотеки IEEE.

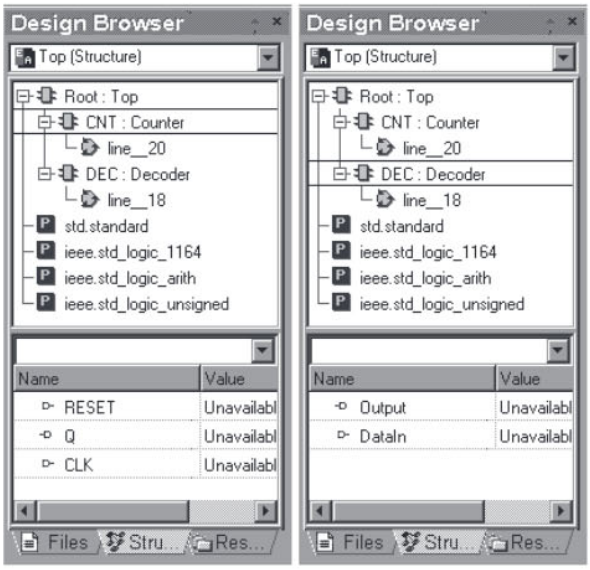


Рис. 5.34. Просмотр портов компонент

Каждая единица проекта (компонент) содержит несколько портов, внутренних сигналов и переменных. Система моделирования **Active-HDL** позволяет легко просматривать эти данные в нижней части окна **Design Browser** (рис. 5.34).

5.5. Выполнение процесса моделирования

Для начала моделирования необходимо прежде всего инициализировать симулятор, используя опцию **Initialize Simulation** из меню **Simulation**. После инициализации можно открыть новое окно **Waveform** (рис. 5.35) с помощью кнопки **New Waveform** в верхней части интерфейса системы.

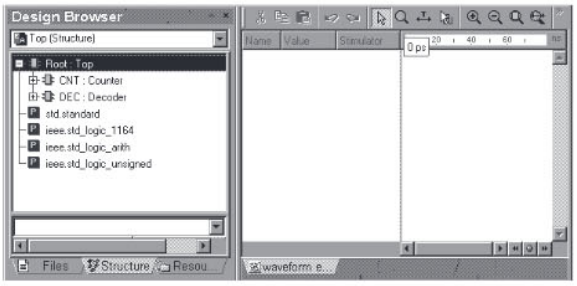


Рис. 5.35. Окно Waveform Viewer

Для просмотра сигналов в симуляторе необходимо открыть закладку **Structure** в окне **Design Browser**, выбрать проект, помеченный меткой **Root**, и с помощью стандартной drag-and-drop-операции перетащить в правое окно **Waveform** необходимые сигналы (рис. 5.36).

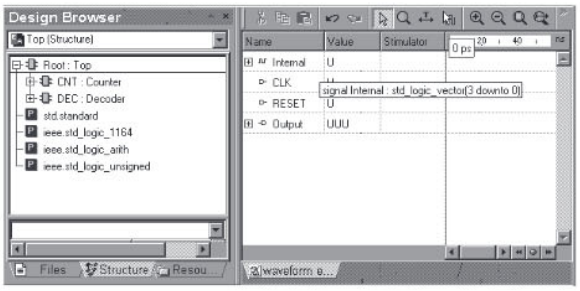


Рис. 5.36. Добавление сигналов в окно из окна Design Browser

Для удаления сигналов из окна необходимо выбрать соответствующий сигнал и нажать правую кнопку мыши, после чего в возникшем меню использовать опцию **Delete**.

Система моделирования **Active-HDL** позволяет просматривать результаты моделирования в табличном формате с точностью до delta-delay time. Окно **List Viewer** позволяет просмотреть величины сигналов (рис. 5.37). Для открытия этого окна необходимо щелкнуть по кнопке **New List**. Добавление сигналов в окне **List Viewer** выполняется аналогичным добавлению сигналов в окне **Waveform Viewer**, рассмотренному выше (рис. 5.38). Окно **List Viewer** является интерактивным дисплеем, отображающим все операции и результаты моделирования.

Для задания внешних воздействий (**Stimulators**) необходимо выбрать определенный сигнал в левой части окна **Waveform Viewer**, например выбрать сигнал CLK. Затем в возникшем после нажатия правой клавиши мыши меню выберем опцию **Stimulators** (рис. 5.39).

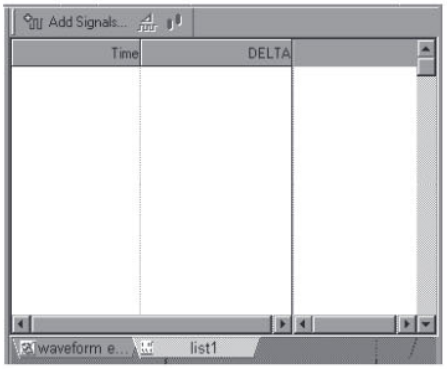


Рис. 5.37. Окно List Viewer

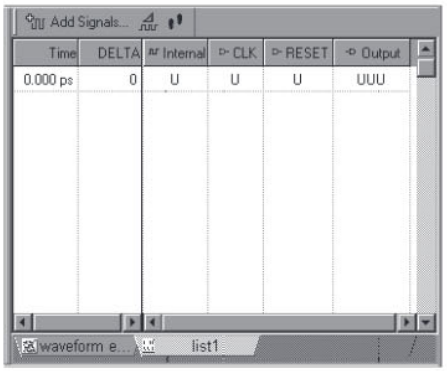


Рис. 5.38. Окно List Viewer в процессе добавления сигналов

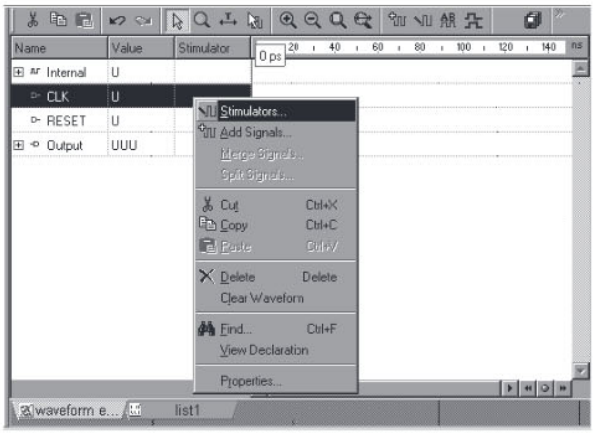


Рис. 5.39. Задание входных воздействий с помощью сигналов

Вслед за этим появляется окно специально для задания входных воздействий (рис. 5.40).

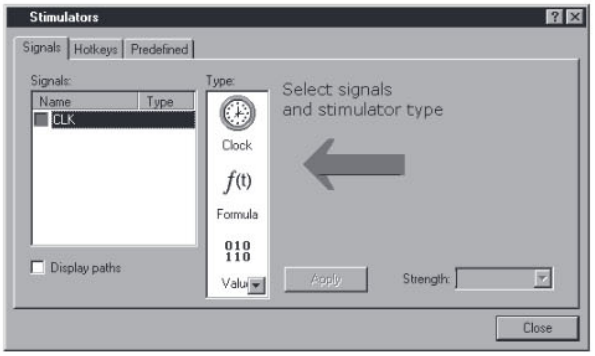


Рис. 5.40. Окно Stimulators

Выберем в качестве входного воздействия для порта CLK серию импульсов с определенными частотой и длительностью импульса (опция **Clock**), как показано на рис. 5.41.

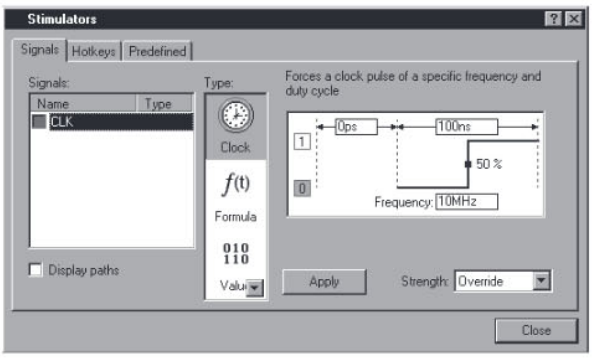


Рис. 5.41. Применение стимула в виде серии импульсов (Clock Stimulator) для сигнала CLK

Для задания параметров этой серии импульсов поместим указатель мыши на поле **Frequency** и введем значение 10 MHz. В качестве подтверждения выбора данного стимула щелкнем по клавише **Apply**. Затем выберем порт RESET в окне **Waveform Viewer**, опцию **Stimulators** во всплывшем меню и опцию **Formula** в окне **Stimulators**. В поле **Enter formula** введем следующее выражение: 1 0, 0 10000. Это означает, что начальное значение для RESET — 1, а затем в момент времени 0 оно меняется на 0 и остается таковым до момента времени 10000 ns (рис. 5.42). В подтверждение нажмем **Apply** и затем закроем текущее окно (**Close**). Выполнение шагов моделирования осуществляется с помощью клавиш **Trace Over** и **Run For**. На закладке **Waveform** получаем результаты моделирования в виде совокупности временных диаграмм (рис. 5.43).

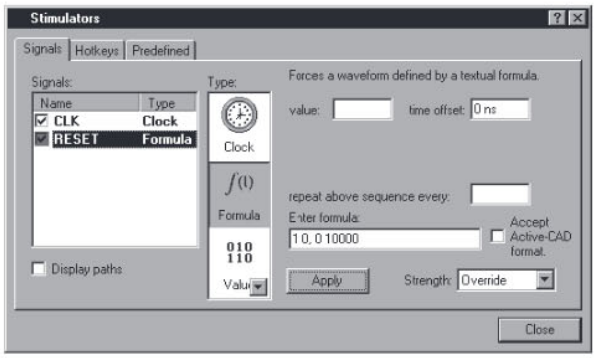


Рис. 5.42. Применение Formula Stimulator для порта RESET

Вместо результатов моделирования для шин могут быть получены результаты для их отдельных линий. Так, на рис. 5.44 результаты для отдельных линий шины Internal получаются щелчком по значку «+». Завершаем моделирование выбором

опции **End Simulation** в меню **Simulator**. Для отображения результатов в табличном формате предназначено окно **List Viewer** (рис. 5.45).

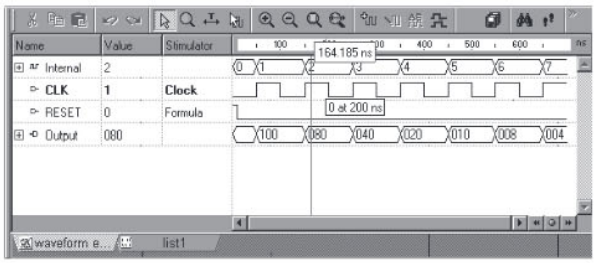


Рис. 5.43. Результаты моделирования

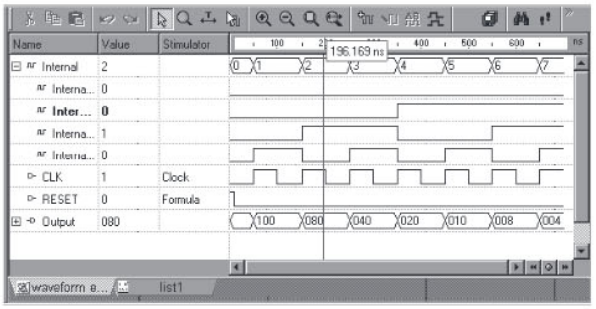


Рис. 5.44. Результаты моделирования с уточнением для шины Internal

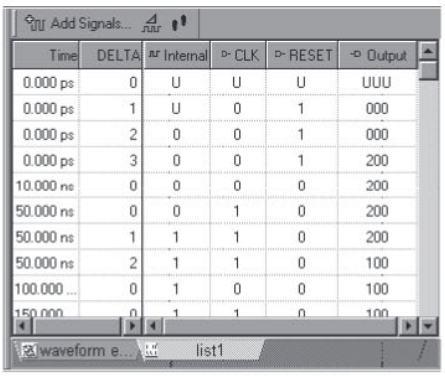


Рис. 5.45. Результаты моделирования в окне List Viewer

Для отображения результатов с точностью до delta delay надо щелкнуть по иконке с изображением  .

Система моделирования Active-HDL позволяет выполнить процесс моделирования и другим путем — с использованием автоматической генерации испытательных программ (Test Benches). Можно также воспользоваться в качестве внешних воздействий сигналами, отображаемыми в окне **Waveform Viewer**. Для этого надо экспортировать результаты моделирования в отдельные VHDL-процессы, ак-

тивизируемые обычными сигналами. При этом необходимо использовать опцию **Export Waveforms** из меню **Waveform**. В появившемся окне (рис. 5.46) вводится имя файла, который сохранит эти временные диаграммы.

После сохранения данных временных диаграмм с помощью кнопки **Save** мы готовы генерировать Test Bench. Для этого щелкнем правой клавишей мыши по имени архитектуры верхнего уровня в окне **Design Browser** и выберем команду **Generate Test Bench** (рис. 5.47).

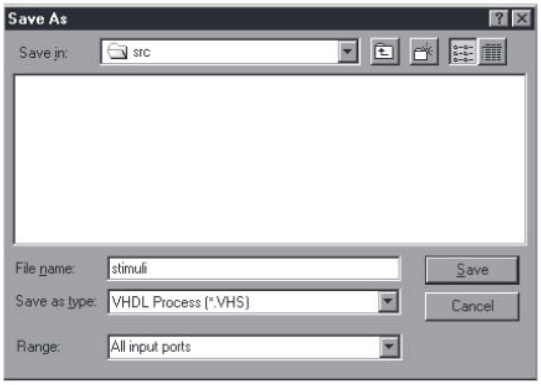


Рис. 5.46. Экспорт временных диаграмм в VHDL-процесс

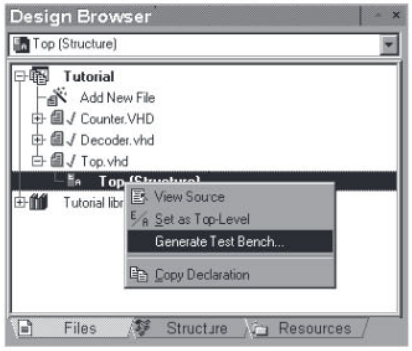


Рис. 5.47. Генерация Test Bench

В результате появляется окно мастера построения Test Benches (рис. 5.48).

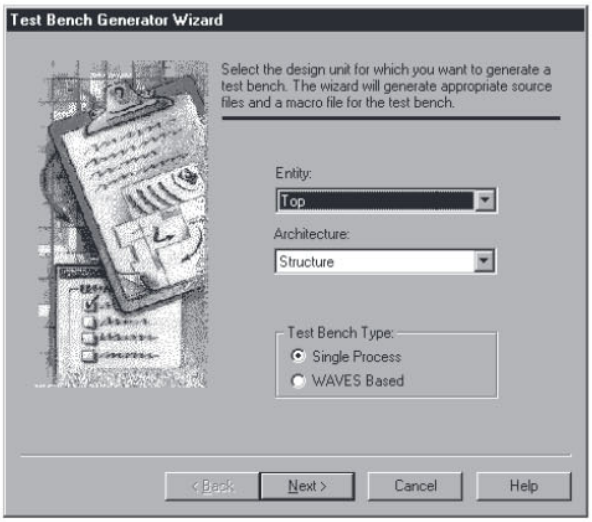


Рис. 5.48. Окно Test Bench Generator Wizard

На первой странице этого окна выбираем имена entity и architecture, для которых генерируется **Test Bench**, и нажимаем для подтверждения **Next**. Следующий шаг базируется на предварительно экспортированных и сохраненных временных диаграммах. Выберем на второй странице текущего окна опцию **Test vectors from**

file и введем в поле **Select a waveform file** имя нашего файла с временными диаграммами — stimuli.vhs (рис. 5.49), а затем перейдем к следующему окну (рис. 5.50). Здесь вводятся:

- имя entity для текущего Test Bench;
- имя файла, в котором будет сохранен текущий Test Bench;
- имя папки, в которой будет находиться этот файл.

На последнем этапе можно воспользоваться опцией для задания конфигурации моделирования временных соотношений (timing simulation) (рис. 5.51).

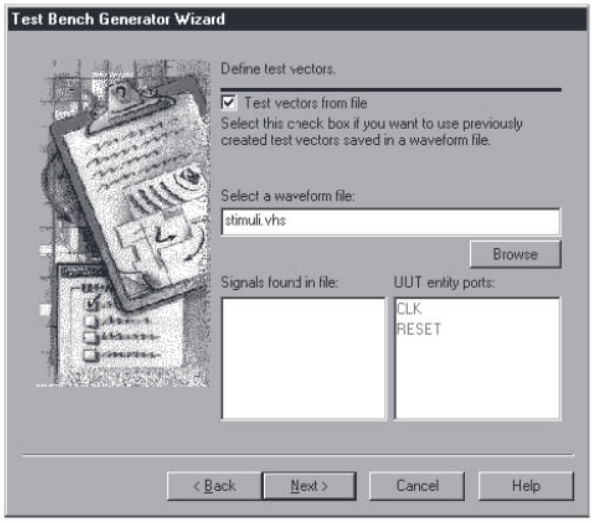


Рис. 5.49. Импорт тестовых векторов из файла

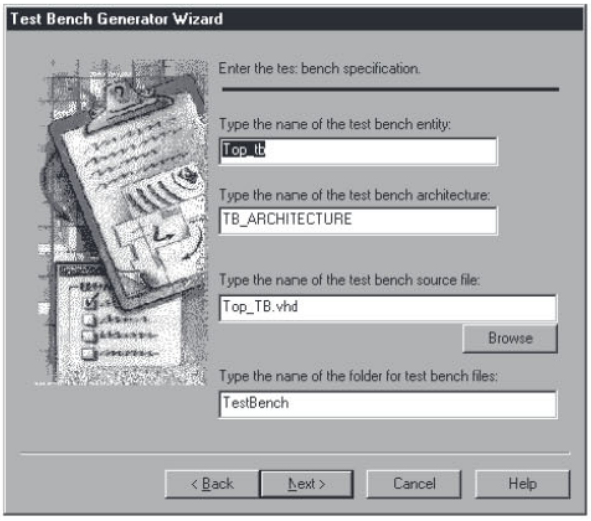


Рис. 5.50. Назначения для текущего Test Bench

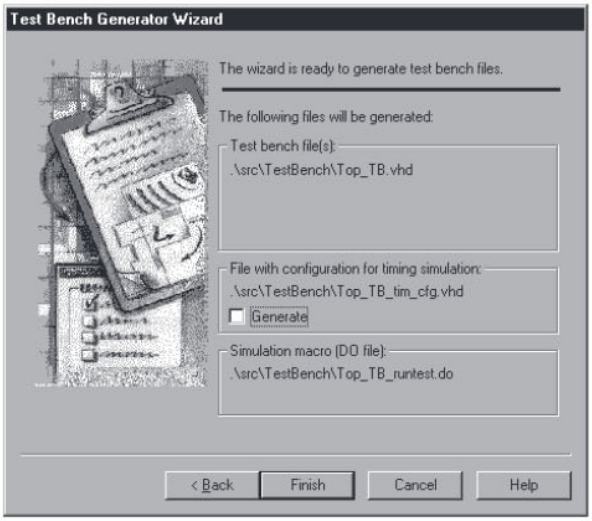


Рис. 5.51. Экспорт временных диаграмм в VHDL-процесс

В результате наших последних действий будет создана новая папка в окне **Design Browser**, в которой будет храниться файл с Test Bench и файл макро-команд (с расширением *.do) для автоматической компиляции и моделирования (рис. 5.52):

Для начала моделирования с помощью сгенерированного Test Bench щелкнем правой кнопкой мыши по метке файла макрокоманд и в появившемся меню выберем команду Execute (рис. 5.53).

Результаты выполнения файла макрокоманд после автоматической рекомпиляции и моделирования приведены на рис. 5.54. Система Active-HDL позволяет шаг за шагом проследить за исполнением кода в процессе моделирования. Окно с VHDL-кодом автоматически открывается после каждого нажатия клавиши пошагового выполнения моделирования. При этом оператор, который будет выполняться следующим, выделяется в окне **HDL Editor** желтым цветом. Вначале необходимо закончить предыдущий процесс моделирования выбором опции **End Simulation** из меню **Simulation** и инициализировать систему заново выбором опции Initialize.

Из меню **View** выберем опции **Watch**, **Processes** и **Call Stack** для вызова трех одноименных окон.

Окно **Processes** показывает статус процессов текущего проекта.

Окно **Watch** позволяет нам проверить значения сигналов и переменных.

Окно **Call Stack** является инструментом отладки и отображает список подпрограмм (процедур и функций), выполняющихся в текущем активном процессе. Здесь термин «процесс» используется для обозначения любого:

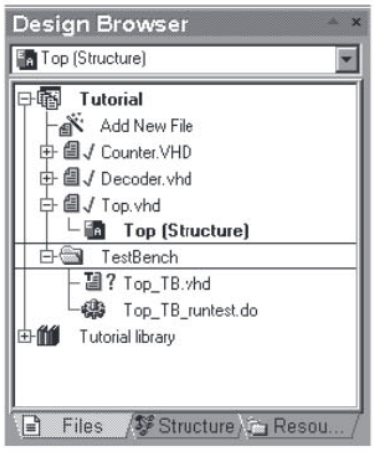


Рис. 5.52. Папка для нового Test Bench

– параллельного оператора процесса (concurrent process statement);
– параллельного оператора присвоения значений сигналам (concurrent signal assignment statement);
– параллельного оператора утверждения (concurrent assertion statement);
– параллельного оператора вызова процедуры (concurrent procedure call statement).

Для каждой подпрограммы это окно отображает следующую информацию:

– формальные параметры с их текущими значениями;
– локальные переменные и константы в телах подпрограмм с их текущими значениями.

В течение процесса отладки используются следующие команды:

– **Trace into** — выполняется одиночный VHDL-оператор, если этот оператор вызывает подпрограмму, то далее будут выполняться операторы, находящиеся в теле данной подпрограммы;
– **Trace over** — выполняется простая VHDL-команда, если эта команда вызывает подпрограмму, то все операторы, содержащиеся в теле подпрограммы, выполняются за один шаг;
– **Trace out** — выполняется столько операторов, сколько необходимо для завершения выполнения подпрограммы. Если налицо вложенные подпрограммы, эта команда завершает выполнение самой «внутренней» подпрограммы.

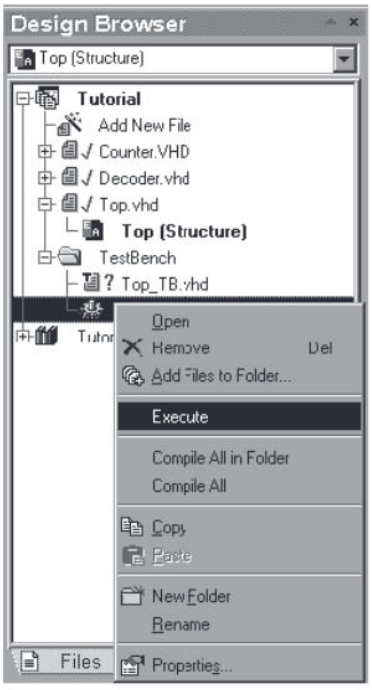


Рис. 5.53. Выполнение файла макрокоманд

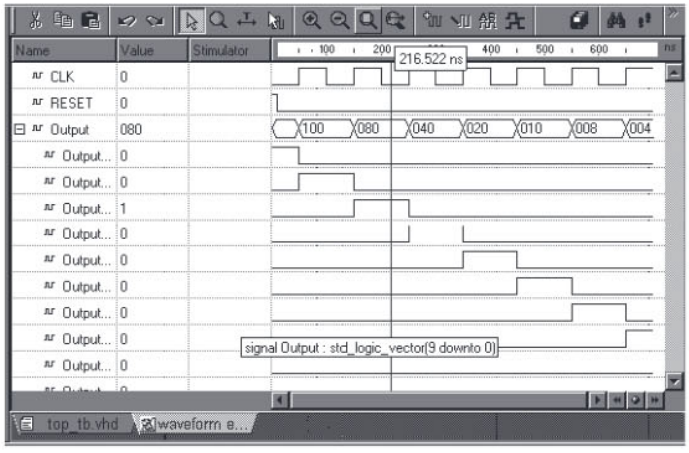


Рис. 5.54. Результаты моделирования

Используя эти команды, можно наблюдать:
- изменения значений сигналов в окне **Watch**;
- активность обычных процессов в окне **Processes**;
- локальные переменные и константы с их текущими значениями в телах дпрограмм в окне **Call Stack**.

Щелкнем по закладке **Structure Tab** в окне **Design Browser**. Выберем entity *TESTBENCH_FOR_Top* вместе с атрибутом *ROOT* и, используя технику drag&drop, перетащим сигналы этого интерфейса в окно **Watch**. Щелчок правой клавишей мыши в любом месте окна **Processes** и выбор опции **Show all** позволяют видеть активные и пассивные процессы в текущем цикле моделирования (рис. 5.55). Здесь вновь под процессом понимаем любой параллельный оператор. Пометка в виде восклицательного знака перед именем сигнала в окне **Watch** означает, что величина данного сигнала меняется в текущем цикле моделирования (таков, например, сигнал CLK на рис. 5.56).

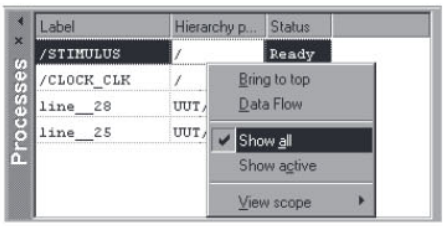


Рис. 5.55. Отображение всех процессов проекта в окне Processes

В окне **Processes** можно видеть список процессов и их текущий статус (ready или wait). Статус ready означает, что процесс в настоящий момент является активным (CLOCK_CLK на рис. 5.57). Такие процессы располагаются вверху списка. Остальные же процессы являются пассивными.

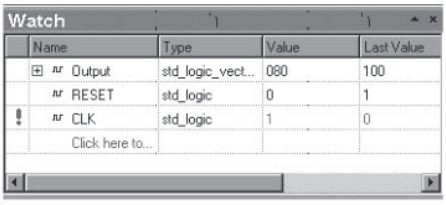


Рис. 5.56. Окно Watch

В окне **Call Stack** можно наблюдать только величины переменных, которые декларированы в выбранном процессе (процесс UUT/CNT, начинающийся со строки 20, переменная Qint — рис. 5.58). Окно **Data Flow** является инструментом, обеспечивающим графическое представление сигналов, входящих в процесс и выходящих из него. Это окно обеспечивает два различных представления:
- с процессом в центре окна;
- с сигналом в центре окна.

Выберем опцию **Data Flow** из меню **View**. В результате появляется одноименное окно (рис. 5.59).

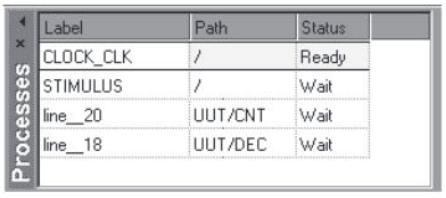


Рис. 5.57. Окно Processes

Щелкнем сейчас по имени сигнала *RESET* в этом окне. При этом появляется следующий фрагмент окна **Data Flow** (рис. 5.60). Отсюда видно, что данный сигнал получает значения в процессе *STIMULUS* и является входящим сигналом для процесса *UUT/CNT/line_28.*

Если же сейчас мы щелкнем по метке имени процесса *UUT/CNT/line_28*, откроется графическое представление этого процесса (рис. 5.61).

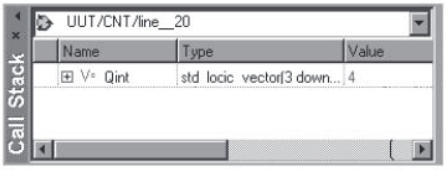


Рис. 5.58. Окно Call Stack

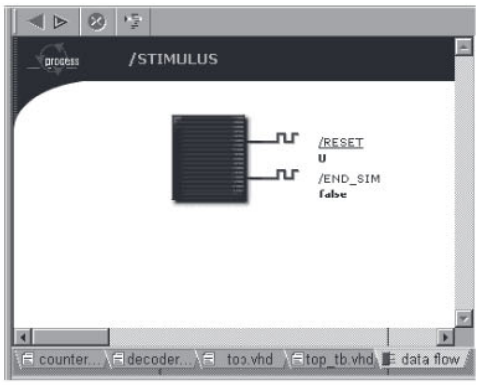


Рис. 5.59. Окно Data Flow

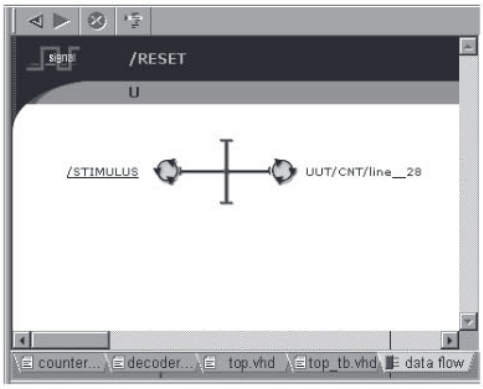


Рис. 5.60. Следующий фрагмент окна Data Flow

Используя клавишу **F5** на клавиатуре или кнопку **Run For** линейки инструментов графического интерфейса системы, можно увидеть изменение состояния выхода счетчика Q в данном окне (рис.5.62).

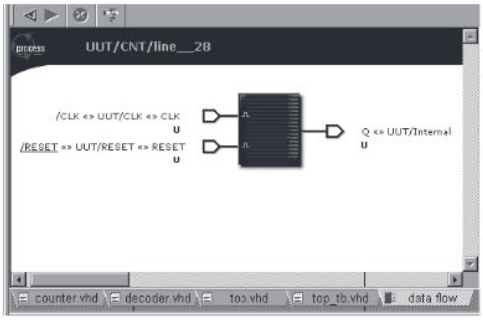


Рис. 5.61. Графическое представление процесса в окне Data Flow

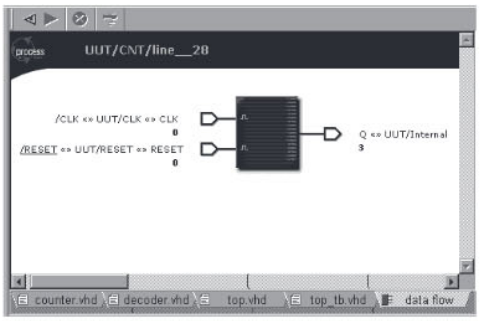


Рис. 5.62. Изменения сигналов в окне Data Flow

Глава 6. Работа с VHDL в среде системы OrCAD

Для работы с VHDL-проектами в среде системы OrCAD 9.2 (моделирования цифровых устройств и разработки ПЛИС) предназначен модуль OrCAD Express, включающий программу моделирования цифровых устройств Orcad Simulate.

Редактор проектов OrCAD Capture обеспечивает при этом для OrCAD Express все типы проектов, включающих принципиальные схемы или текстовые VHDL-описания.

Процесс проектирования ПЛИС (PLD Design Flow) содержит пять этапов [48a,49a]:

- создание проекта (Design Entry);
- функциональное моделирование (Functional Simulation);
- реализация проекта с помощью выбранной технологии (Design Implementation);
- размещение и трассировка (Place-and-Route);
- временное моделирование (Timing Simulation).

Здесь мы рассмотрим лишь первые два этапа.

6.1. Создание проекта

На этой стадии вырабатываются общие концепции проектирования. OrCAD Express обеспечивает инструменты, необходимые для определения функций проекта в виде схем (schematic form) или VHDL-моделей (VHDL-models). OrCAD Express позволяет создание смешанных проектов (Mixed Designs), содержащих папки схем (Schematic Folders) и VHDL-модели. Как правило, на данном этапе проектирования схемы создаются без рассмотрения их временных характеристик (Timing Characteristics).

Таким образом, даже если уже известна технология физической реализации проекта (Target Vendor Technology), логика проекта реализуется в общей форме, независимо от рассмотрения временных ограничений, присущих определенной технологии (Technology Specific Timing).

Для создания новой схемы проекта прежде всего необходимо выбрать папку Design Resources. Это делается выбором опции New/Design из меню File (рис. 6.1). При этом новая страница для создания схемы вносится в папку Design Resources в окне менеджера проектов (Project Manager) (рис. 6.2).

Присоединение VHDL-файлов (Mixing VHDL Files) к схемам текущего проекта (Schematic Design) осуществляется прикреплением VHDL-файла к соответствующему иерархическому блоку (Hierarchical Block). Для этого из меню Place выбирается опция Hierarchical Block (рис. 6.3). Иерархический блок представляет в схеме проекта VHDL-файл, прикрепленный к нему. Контакты иерархического блока (Hierarchical Pins) рассматриваются как связующие звенья между блоком (объектами прикрепленного VHDL-файла) и остальной схемой. Для размещения иерархического блока необходимо (рис. 6.4) в окне Place Hierarchical Block выполнить следующие действия:

- дать ссылку на этот блок в поле Reference;
- указать в поле Implementation Type — VHDL;

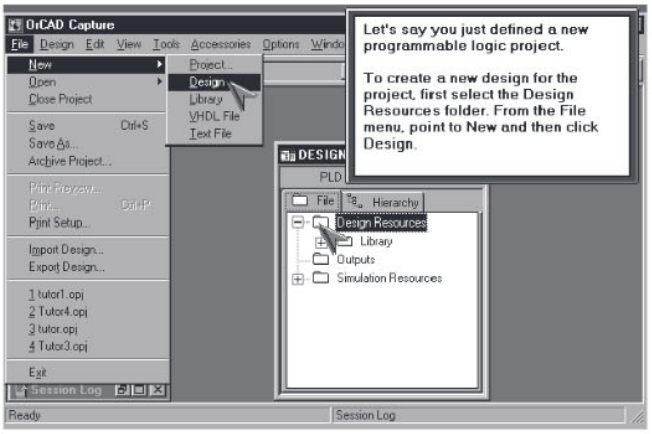


Рис. 6.1. Выбор опции New/Design из меню File

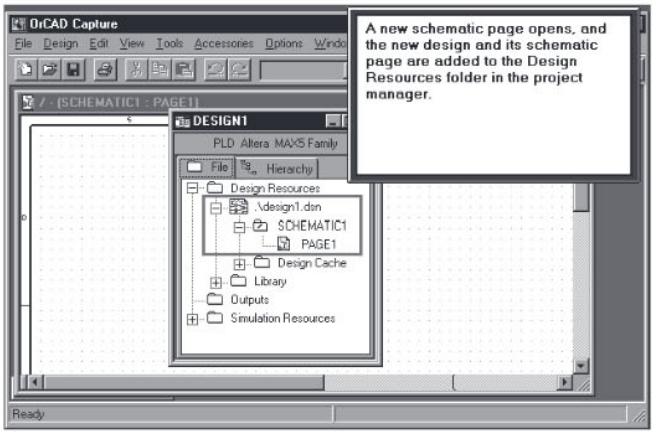


Рис. 6.2. Внесение новой страницы в папку Design Resources

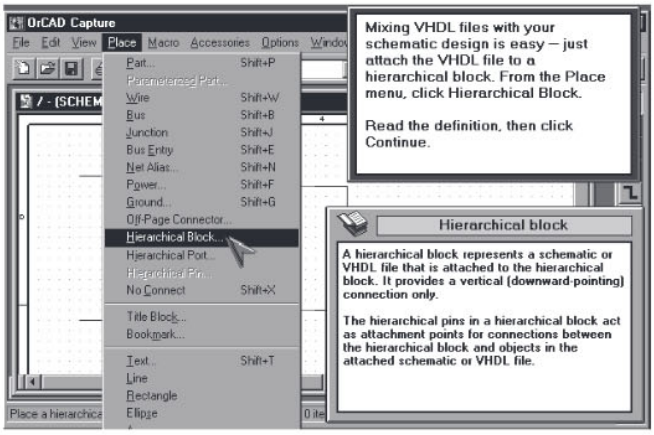


Рис. 6.3. Выбор опции Hierarchical Block из меню Place

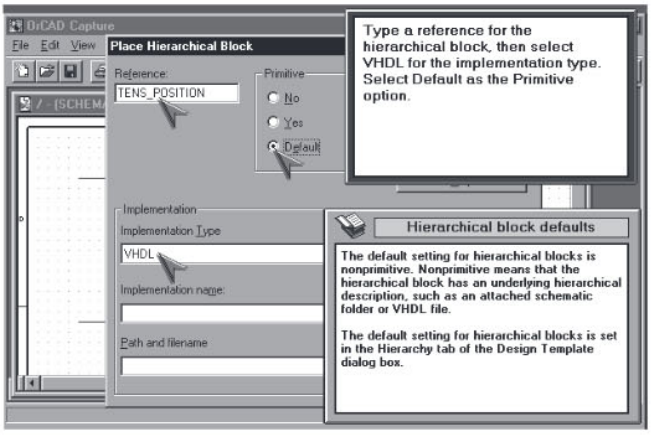


Рис. 6.4. Размещение иерархического блока в схеме проекта

- выбрать опцию Default в группе Primitive.

Опция Default для иерархического блока означает, что он не является примитивным и имеет описание в виде прикрепленного VHDL-файла.

Затем вводится имя блока в поле Implementation name, а также определяется путь к соответствующему VHDL-файлу и его имя (рис. 6.4).

Щелчок левой клавишей мыши инициирует начало рисования блока, а второй такой же щелчок приводит к завершению создания этого блока (рис. 6.5, 6.6).

Когда иерархический блок создается на основе существующей VHDL-модели, OrCAD Express автоматически добавляет контакты к блоку в соответствии со списком портов (Port List) этой модели (рис. 6.7, 6.8).

Для редактирования свойств иерархического блока необходимо сделать двойной щелчок мышью на его изображении. При этом открывается окно Property Editor (рис. 6.9). Для просмотра присоединенного VHDL-файла необходимо выбрать соответствующий иерархический блок и из меню View — опцию Descend Hierarchy (рис. 6.10). Таким образом мы получаем возможность просматривать

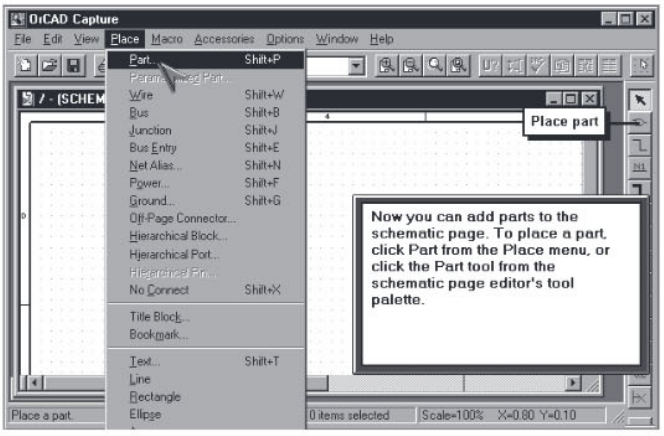


Рис. 6.5. Размещение иерархического блока в схеме проекта по команде Place/Part

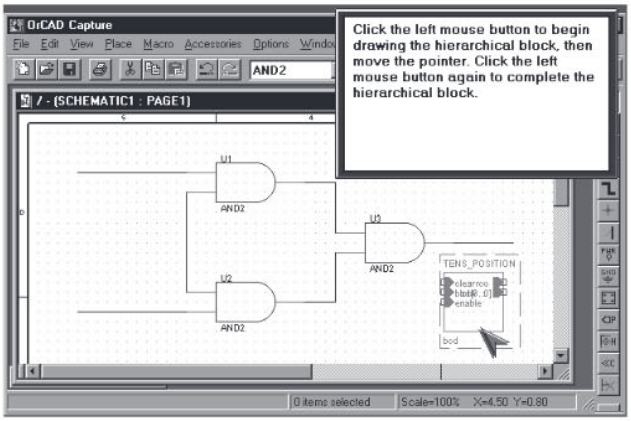


Рис. 6.6. Завершение создания иерархического блока

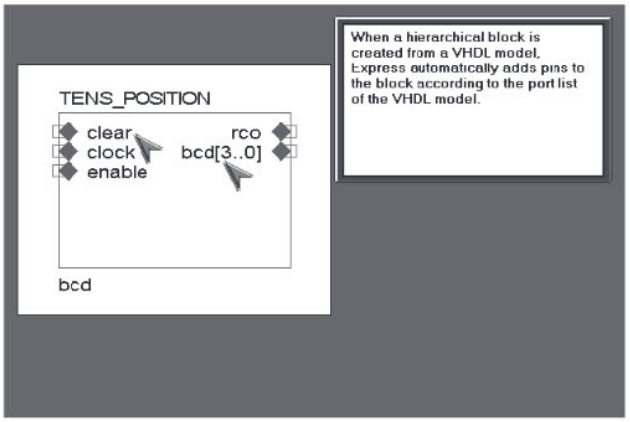


Рис. 6.7. Присоединение контактов к иерархическому блоку

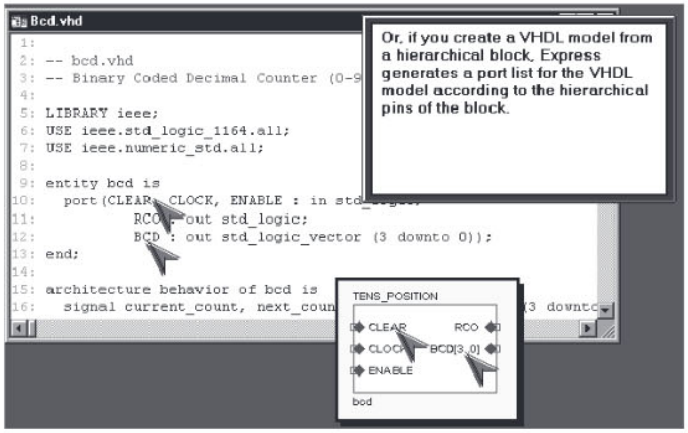


Рис. 6.8. Соответствие между контактами иерархического блока и списком портов VHDL-модели

VHDL-файл и редактировать его (рис. 6.11). Если VHDL-файл для данного иерархического блока еще не существует, то, в то время, как мы планируем иерархию нашего блока, текстовый редактор открывает шаблон-оболочку соответствующего VHDL-файла (VHDL Shell File) (рис. 6.12).

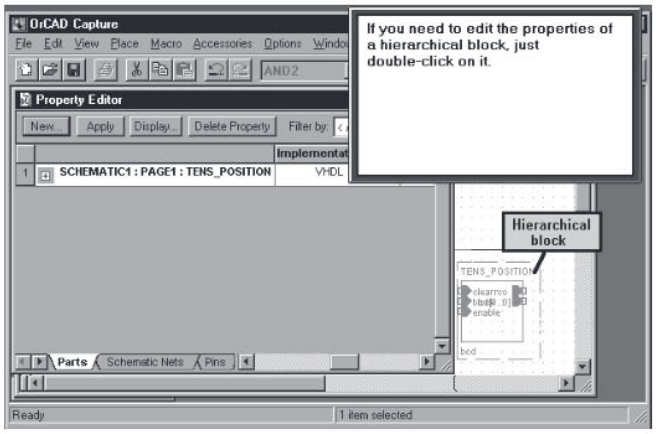


Рис. 6.9. Окно Property Editor для иерархического блока

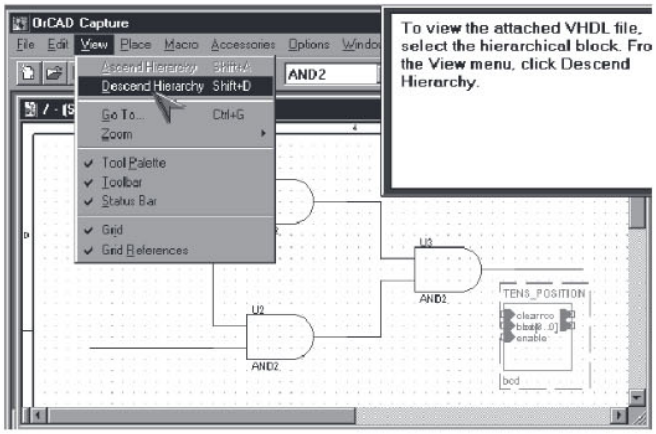


Рис. 6.10. Подготовка к просмотру присоединенного VHDL-файла

После добавления VHDL-кода в новый файл необходимо его сохранить, выбрав опцию Save из меню File (рис. 6.13). Если в дальнейшем возникает необходимость вновь просмотреть этот файл или отредактировать его, достаточно сделать двойной щелчок на его метке в окне Project Manager. Это приведет к открытию данного файла для редактирования (рис. 6.14). Кстати,текстовый редактор выделяет ключевые слова языка (VHDL Keywords) синим цветом (рис. 6.15).

В открытом VHDL-файле можно выполнить операцию поиска или проверить его на наличие синтаксических ошибок (Syntax Errors), выбрав опцию Check VHDL Syntax из меню Edit (рис. 6.16).

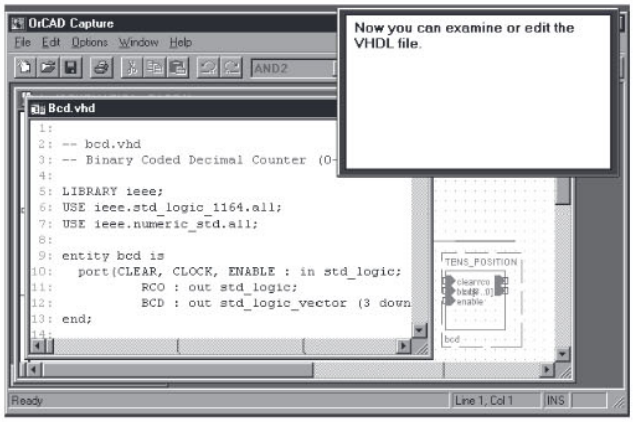


Рис. 6.11. Присоединенный к иерархическому блоку VHDL-файл доступен для просмотра и редактирования

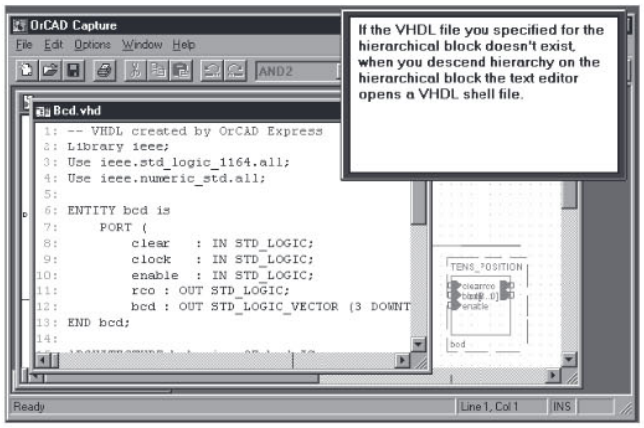


Рис. 6.12. Открытие шаблона-оболочки VHDL Shell File

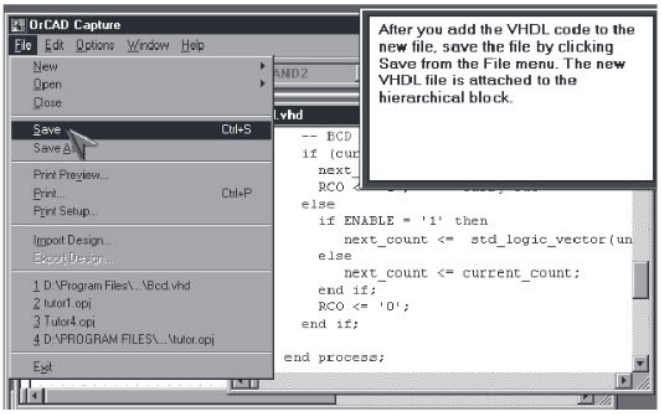


Рис. 6.13. Сохранение VHDL-файла для иерархического блока

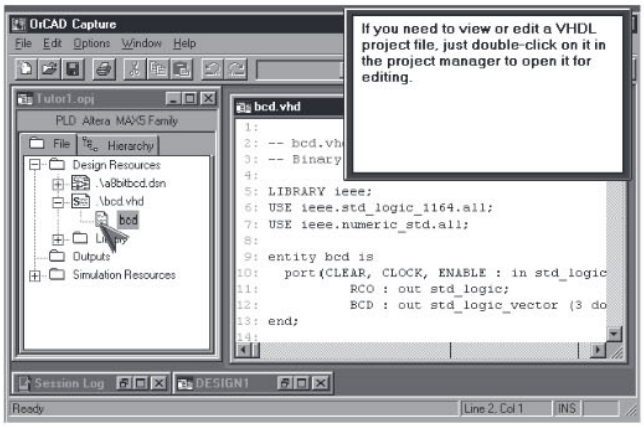


Рис. 6.14. Открытие VHDL-файла проекта для просмотра и редактирования

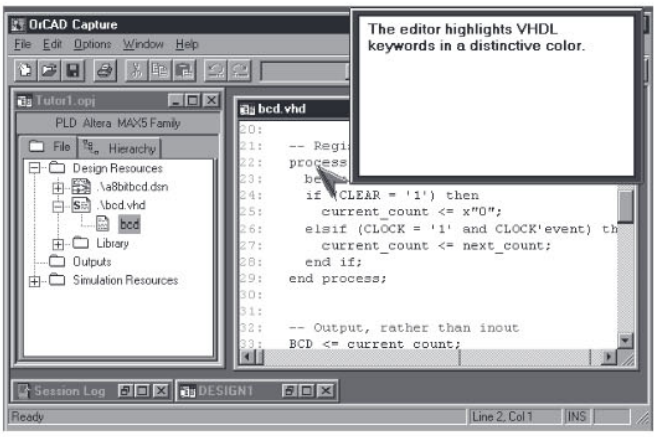


Рис. 6.15. Выделение ключевых слов языка VHDL синим цветом

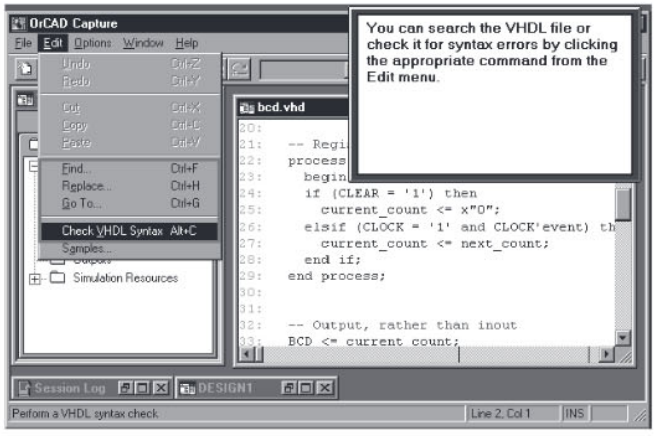


Рис. 6.16. Проверка синтаксических ошибок VHDL-файла

Когда начальное проектирование завершено, можно обновить ссылки на отдельные элементы проекта (Part References) или сделать новые ссылки выбором соответствующей папки и опции Annotate из меню Tools (рис. 6.17).

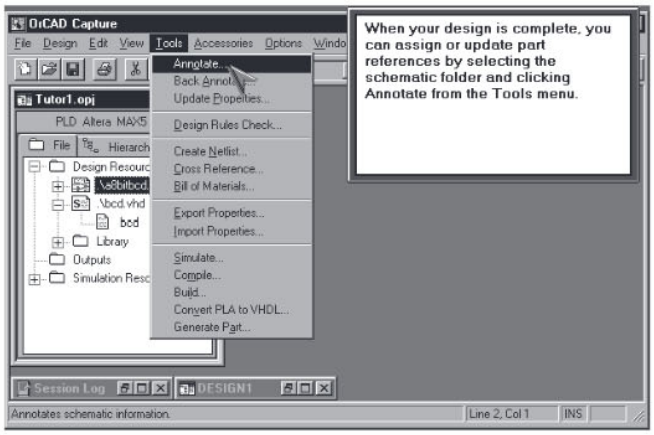


Рис. 6.17. Уточнение ссылок на элементы проекта и выполнение новых ссылок

Для верификации целостности проекта (Design Integrity) и обнаружения ошибок выбирается опция Design Rules Check из меню Tools (рис. 6.18).

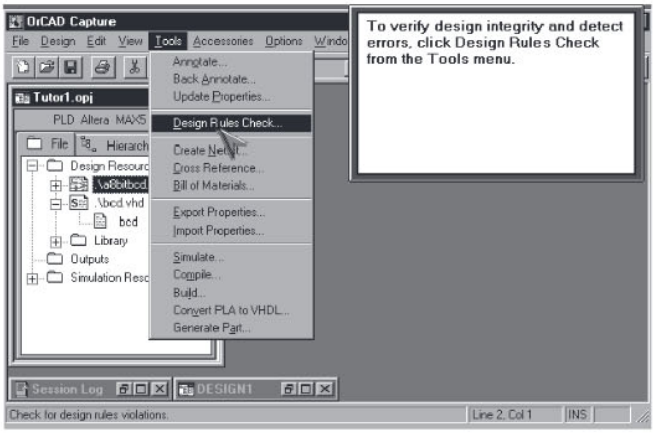


Рис. 6.18. Проверка файла на наличие синтаксических ошибок

Если обнаруживаются ошибки, разработчик получает сообщения о них в окне Session Log (рис. 6.19). После исправления этих ошибок возможен новый запуск программы проверки соблюдения правил проектирования.

6.2. Функциональное моделирование

На этом этапе проектирования осуществляется отладка проекта путем обнаружения ошибок в логике его работы (Design Logic), не принимая во внимание вре-

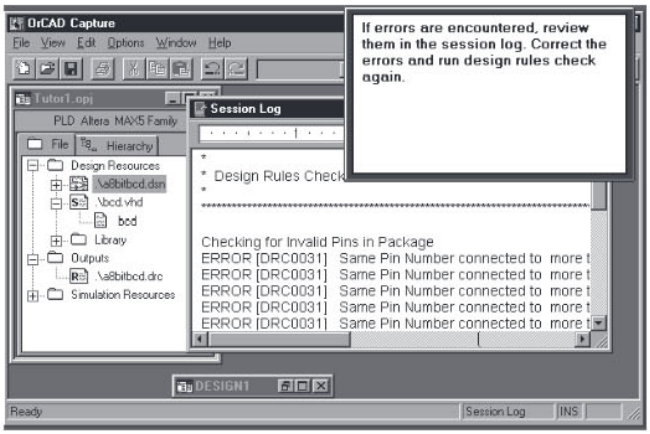


Рис. 6.19. Сообщения о синтаксических ошибках в окне Session Log

менные ограничения. Для выполнения функционального моделирования используются ресурсы, сосредоточенные в подпапке In Design (см. далее).

OrCAD Simulate может моделировать:

- списки соединений схем (Schematic Netlists);
- поведенческие VHDL-модели (VHDL Behavioral Models).

В процессе функционального моделирования необходимо использовать соответствующие входные воздействия (stimuli) для портов проекта и проанализировать выходы для выяснения степени совпадения полученных реакций с ожидаемыми эталонными реакциями (Expected Functionality).

В случае, если применение входных воздействий не приводит к ожидаемым результатам, необходимо проверить выполнение этапа создания проекта (Design Entry). У разработчика есть возможность интерактивной разработки входных воздействий с помощью специального редактора (Simulate Stimulus Editor) или путем разработки собственной испытательной программы (VHDL Test Bench) и внесения ее в текущий проект.

Таким образом, OrCAD Simulate обеспечивает инструмент для моделирования проекта и его функциональной верификации (Functional Simulation) с помощью входных воздействий (Stimulus Files), полученных интерактивным путем или оформленных в виде испытательной программы на языке VHDL (VHDL Test Bench).

Просмотр результатов моделирования (Simulation Results) осуществляется в окне квази-логического анализатора (Logic Analyzer — Like Window) или в форме таблицы истинности (окно List). С помощью VHDL — оператора ASSERT можно наблюдать за выполнением заданных пользователем условий (User-Defined Conditions) в процессе моделирования.

OrCAD Simulate использует различные ресурсы папки Simulation Resources окна Project Manager. На этапах функционального и временного моделирования в процессе проектирования соответствующие файлы (Simulation Files) вносятся в папку Simulation Resources (рис. 6.20).

Папка In Design содержит ресурсы для моделирования проекта на уровне кода (Source Level). Эти ресурсы включают:

- VHDL-код;
- списки соединений схем;

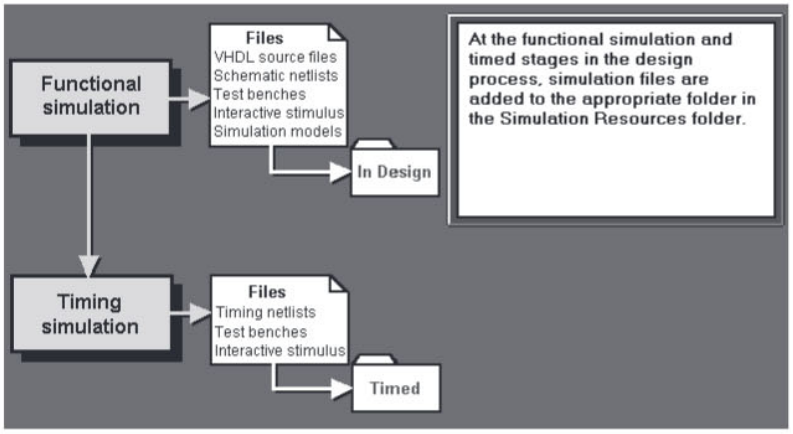


Рис. 6.20. Функциональное моделирования и моделирование временных соотношений (задержек)

- файлы внешних воздействий;
- испытательные программы.

Папка Timed содержит ресурсы для моделирования проекта после размещения и трассировки (Place & Route), поэтому эти ресурсы включают временную информацию для определенной технологии (Technology-Specific Timing Information):

- Standard Delay Files (SDFs);
- файлы внешних воздействий;
- испытательные программы.

Мощным свойством OrCAD Express является способность использовать симулятор со схемным редактором (Schematic Page Editor) в интерактивном режиме (рис. 6.21). Разработчик может видеть специфические состояния для узлов на схемной странице (рис. 6.22), выбирая узлы на схемной странице и параллельно анализируя соответствующие сигналы в OrCAD Simulate (рис. 6.23). Сконфигурировать OrCAD Express для интерактивного анализа (Interactive Probing) можно с помощью выбора опции Preferences из меню Options (рис. 6.24).

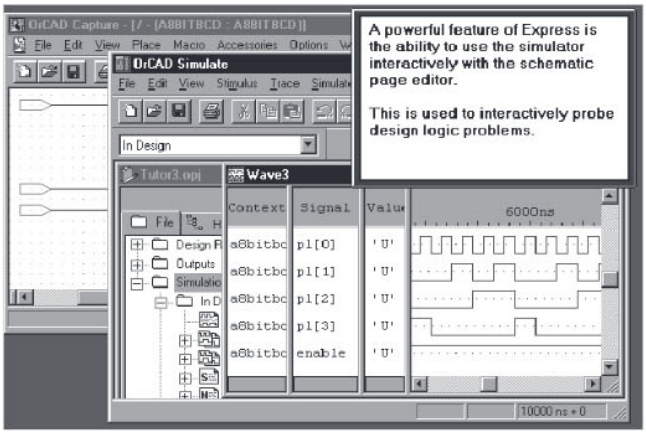


Рис. 6.21. Интерактивный режим работы симулятора со схемным редактором

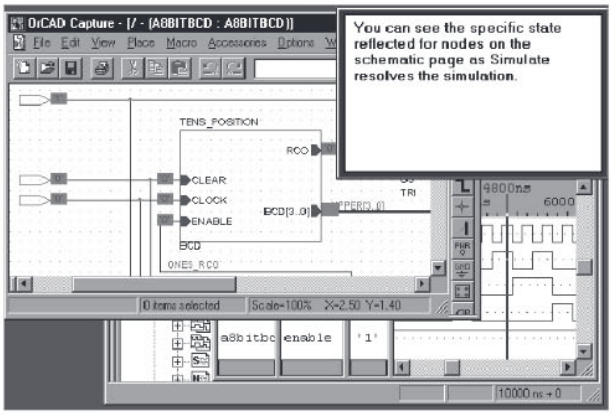


Рис. 6.22. Просмотр состояний узлов на схемной странице

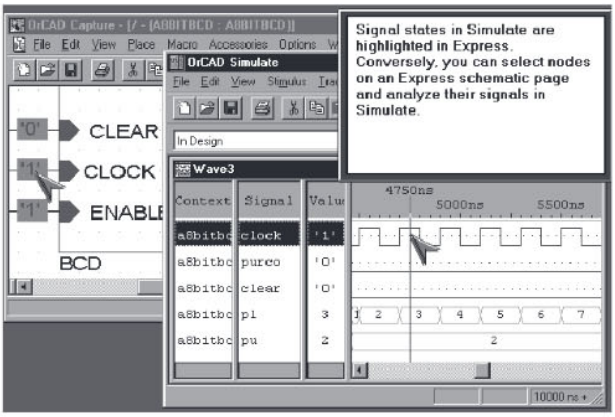


Рис. 6.23. Интерактивный режим работы: параллельный просмотр значений сигналов на схеме и соответствующих временных диаграмм

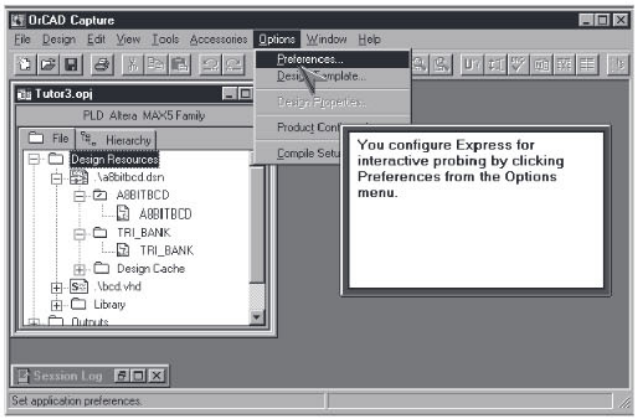


Рис. 6.24. Обращение к опции Options/Preferences для конфигурации OrCAD Express

Для запуска OrCAD Simulate прежде всего необходимо открыть окно Project Manager, а затем выбрать опцию Simulate из меню Tools (или нажать соответствующую кнопку в линейке инструментов — Simulate Toolbar Button) (рис. 6.25). В возникающем затем диалоговом окне Select Simulation Configuration выбирается опция In Design, так как на данном этапе в намерения разработчика не входит моделирование временных задержек (timing delays) (рис. 6.26).

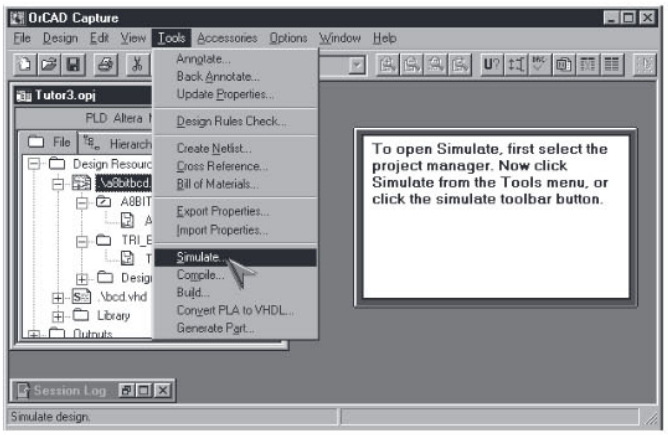


Рис. 6.25. Выбор опции Simulate из меню Tools

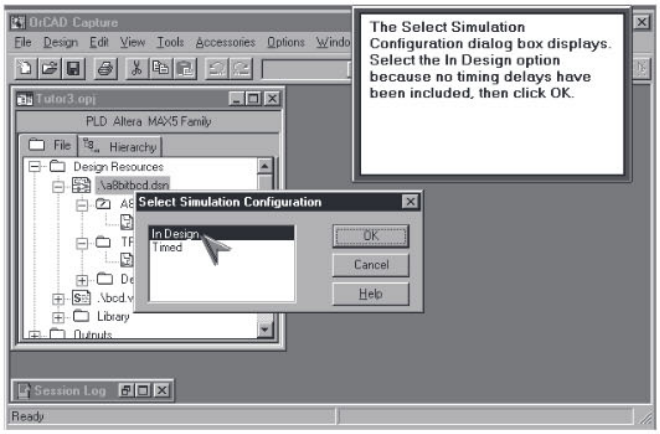


Рис. 6.26. Выбор опции In Design — функциональное моделирование

После сделанного выбора возникает окно OrCAD Simulate (рис. 6.27). Перед началом моделирования проекта необходимо обеспечить входные воздействия. Как уже упоминалось, входные воздействия обеспечиваются двумя путями:

- в ходе интерактивной процедуры определения сигналов,
- написанием испытательной программы (VHDL Test Bench).

Именно с помощью входных воздействий возможно тестирование проекта перед его физической реализацией (Physical Implementation). В дальнейшем тот же набор входных воздействий может быть использован после физической реализации проекта для верификации его временных соотношений (Design Timing).

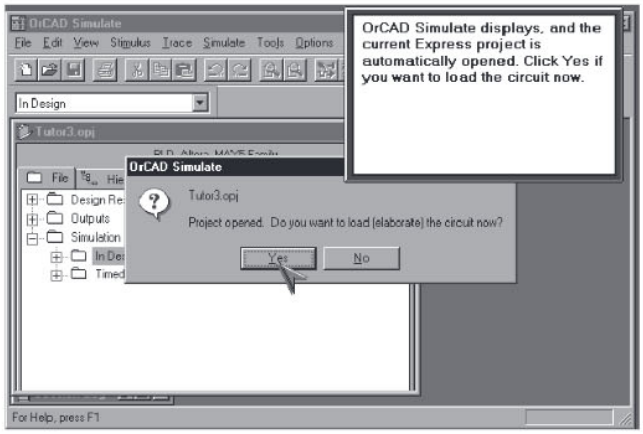


Рис. 6.27. Текущий проект открывается для моделирования в окне OrCAD Simulate

Если испытательная программа уже существует, то она вносится в проект выбором папки Simulation Resource и опции Project в окне Edit (рис. 6.28).

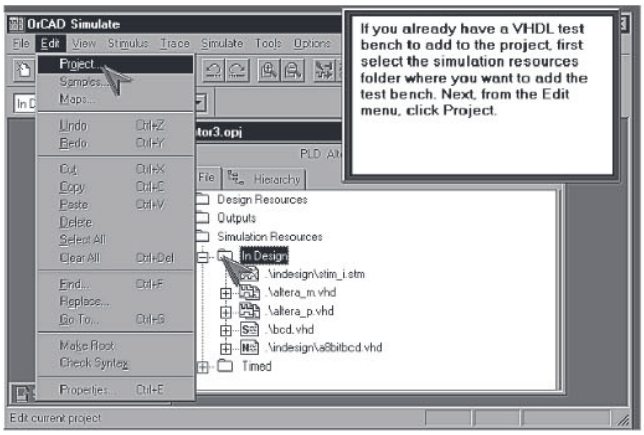


Рис. 6.28. Выбор опции Project/Edit для внесения испытательной программы в проект

В появляющемся затем окне Add File to Project Folder-In Design необходимо выбрать соответствующую испытательную программу и нажать клавишу Open (рис. 6.29). В случае, если OrCAD Simulate нуждается в уточнении содержимого VHDL-файла, выбираем опцию VHDL Testbench (рис. 6.30), и наша испытательная программа вносится в папку In Design (рис. 6.31).

Для создания же новой испытательной программы необходимо выбрать опцию Create Test Bench из меню Stimulus (рис. 6.32). Затем в одноименном диалоговом окне выбираем адекватное содержимое этой испытательной программы — контекст (Context), который может быть интерфейсом верхнего уровня (Top Level Entity) или любым другим интерфейсом (Entity) внутри проекта (рис. 6.33).

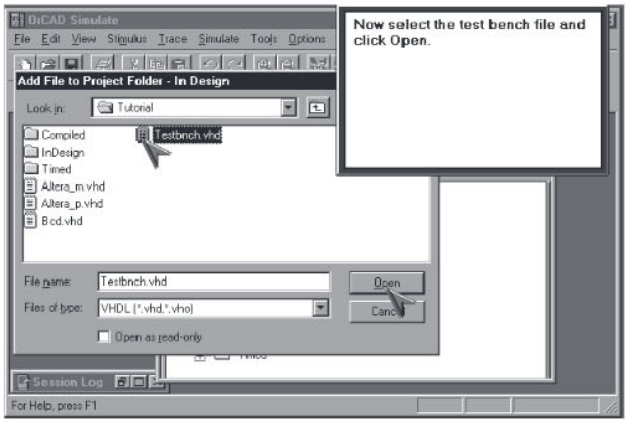


Рис. 6.29. Выбор испытательной программы для текущего проекта

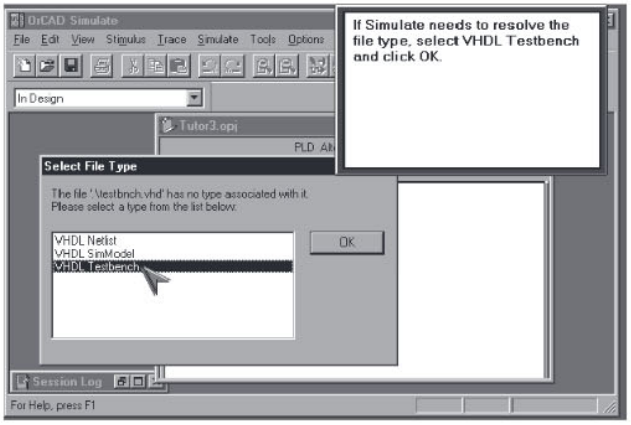


Рис. 6.30. Выбор опции VHDL Testbench для идентификации системой типа VHDL-файла

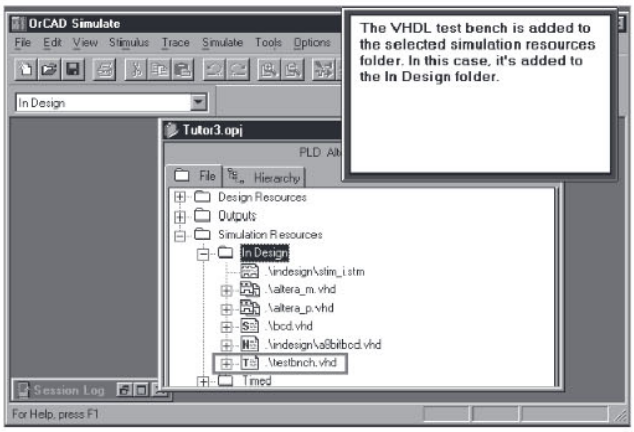


Рис. 6.31. Внесение испытательной программы в текущий проект

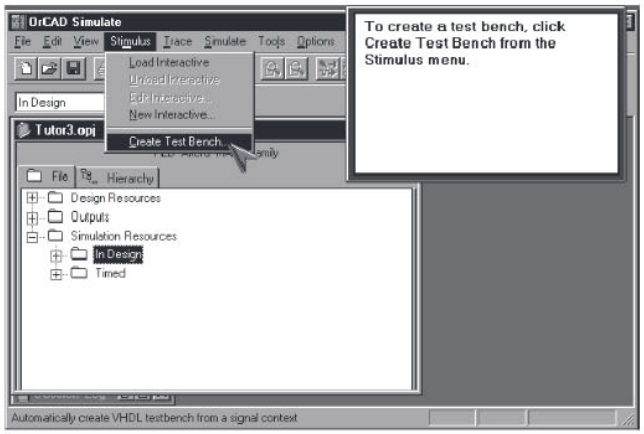


Рис. 6.32. Выбор опции Create Test Bench для создания новой испытательной программы

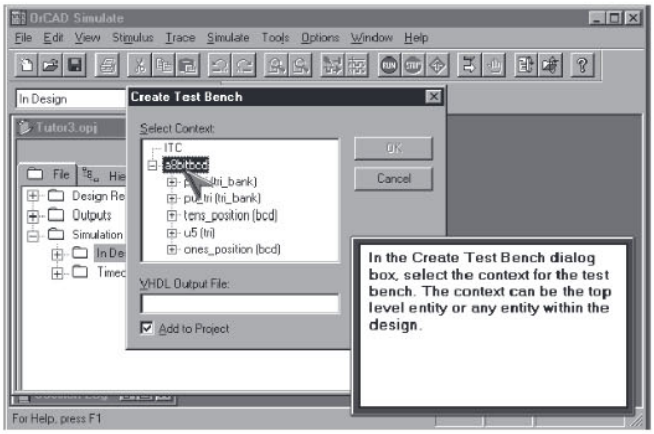


Рис. 6.33. Выбор содержимого будущей испытательной программы

Далее определим имя файла, который будет содержать данную испытательную программу, и выберем опцию Add To Project для вынесения этой программы в текущий проект (рис. 6.34).

Автоматически задается шаблон испытательной программы (Test Bench Template), который включает интерфейс ENTITY и шаблон соответствующего архитектурного тела ARCHITECTURE, имена для них и соответствующие определения сигналов (Signal Definitions) для этого интерфейса (рис. 6.35).

Теперь наступает момент ввода VHDL-кода, который описывает поведение входных сигналов, источником которых является создаваемая испытательная программа (Stimulus Behaviour). Рекомендуемые фрагменты VHDL-кода, готовые к употреблению, становятся доступными при выборе опции Samples из меню Edit (рис. 6.36). После завершения описания испытательной программы она сохраняется путем выбора опции Save из меню File.

Процесс моделирования запускается выбором опции Run из меню Simulate или нажатием кнопки Run в линейке инструментов (рис. 6.37). Дополнительно к это-

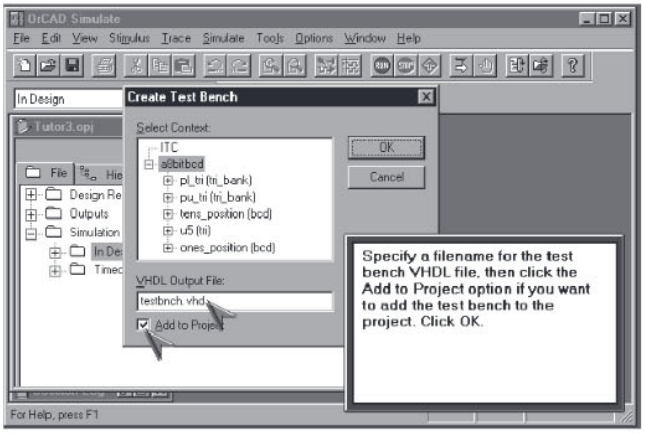


Рис. 6.34. Внесение испытательной программы в проект

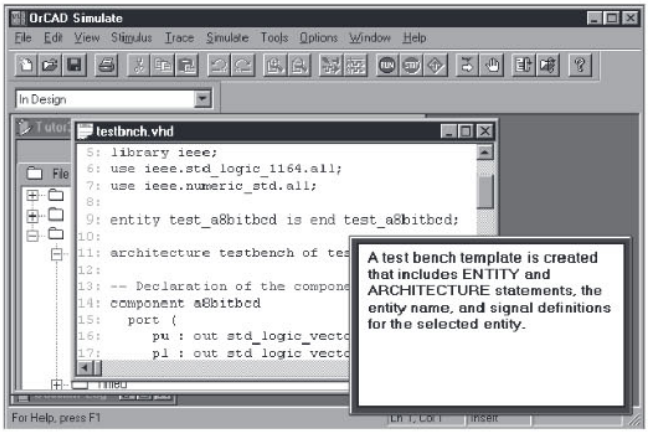


Рис. 6.35. Шаблон испытательной программы

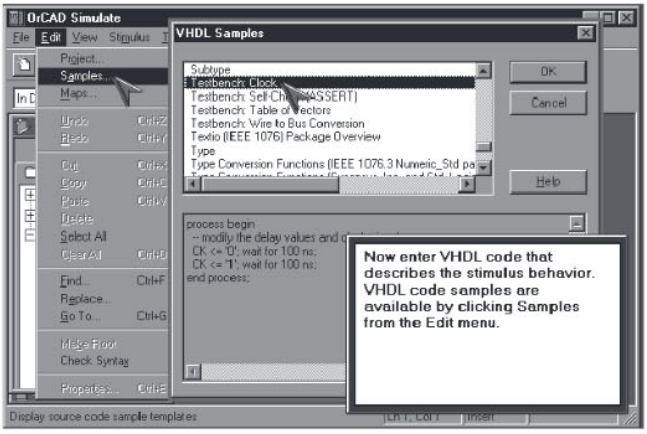


Рис. 6.36. Выбор рекомендуемых фрагментов VHDL-кода

му необходимо определить временной интервал моделирования в диалоговом окне Start Simulator (рис. 6.38). Результаты моделирования отображаются в виде временных диаграмм (Waveforms) в окне Wave и, параллельно, в табличной форме — в окне List (рис. 6.39).

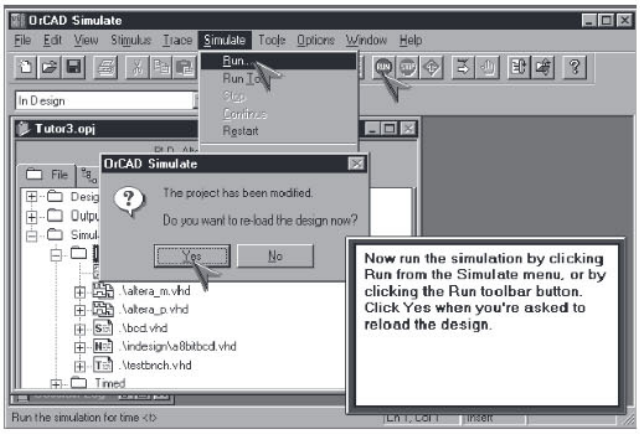


Рис. 6.37. Запуск процесса редактирования

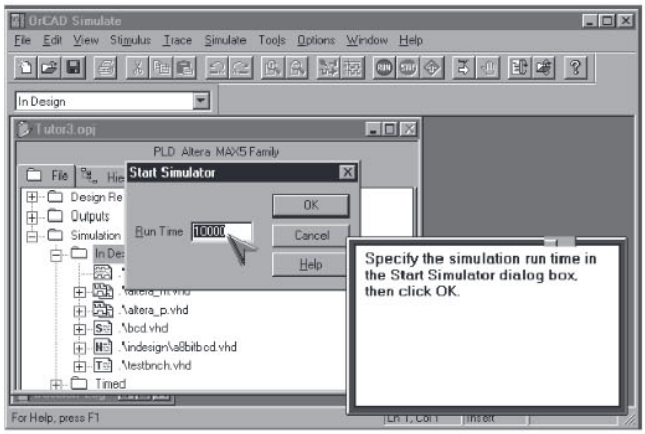


Рис. 6.38. Задание временного интервала моделирования

Для создания входных воздействий в интерактивном режиме работы существует опция New Interactive в меню Stimulus (рис. 6.40). Вслед за этим появляется диалоговое окно Interactive Stimulus. Здесь разработчик может создать:

- базовые выходные воздействия (basic) в виде переходов 0—1, 1—0;
- входные воздействия в виде серий и импульсов (clock);
- сложные входные воздействия (advanced) (рис. 6.41).

Для создания базовых интерактивных входных воздействий (basic interactive stimulus) выбирается закладка Basic (рис. 6.42).

Пакеты временных диаграмм для внешних воздействий (Waveform Patterns) создаются на закладке Advanced. При этом разработчик должен задать полное имя сигнала в поле Stimulate Signal Named или для этой же цели воспользоваться

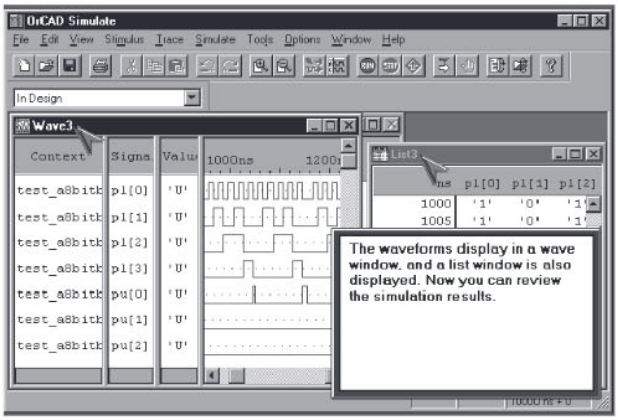


Рис. 6.39. Отображение результатов моделирования

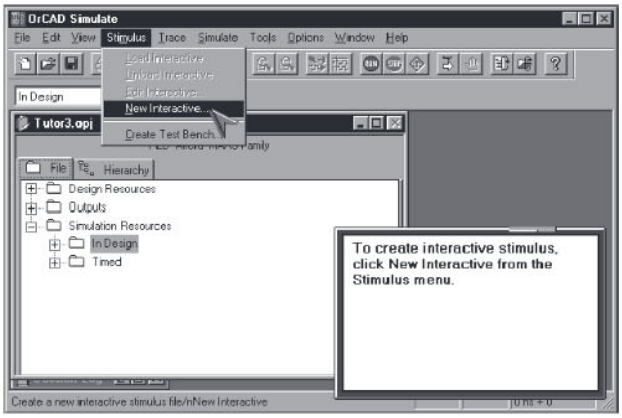


Рис. 6.40. Выбор опции New Interactive/Stimulus

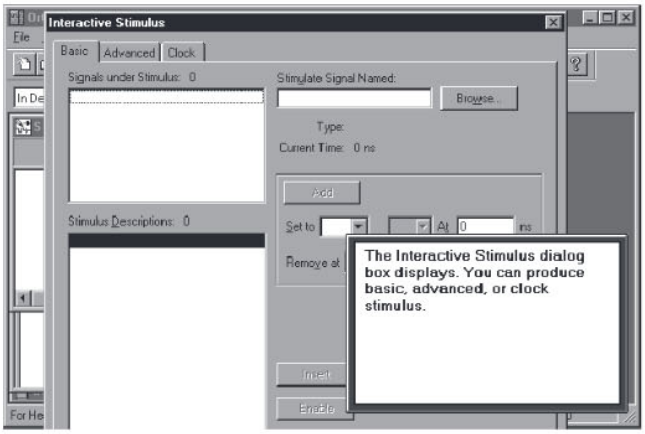


Рис. 6.41. Диалоговое окно для создания интерактивным путем входных воздействий

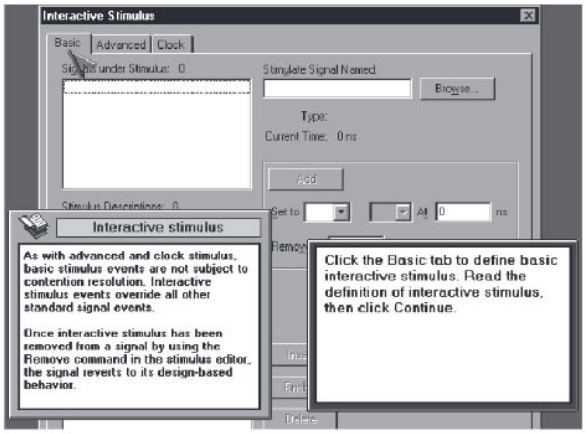


Рис. 6.42. Закладка Basic — создание интерактивных входных воздействий в виде стандартных событий для сигналов

кнопкой Browse (рис. 6.43). Нажав ее, получаем диалоговое окно Browse Signal (рис. 6.44). Выбрав в этом окне опцию All в группе List Signals of Type, можно увидеть список всех сигналов нашей модели (рис. 6.44). В этом списке выберем определенный сигнал (например, enable — рис. 6.45). Затем в поле Start зададим абсолютное время моделирования (Absolute Simulation Time) для точки отсчета времени для пакета временных диаграмм (Waveform Pattern) и установим начальное логическое значение сигнала (рис. 6.46). Щелчок по кнопке Set to приводит к вводу данного присваивания состояния (State Assignment) в список Stimulus Descriptions (рис. 6.47).

Допустим, что входная последовательность для сигнала enable уже создана, поэтому перейдем к определению входной последовательности, например, для сигнала clear. Для этого щелкнем кнопкой Browse и затем выберем сигнал clear в списке Signals in Context (рис. 6.48).

Вновь определим время старта (Start Time), а затем выберем для этого момента начальное состояние сигнала (Initial Signal State) и нажмем кнопку Set to — описа-

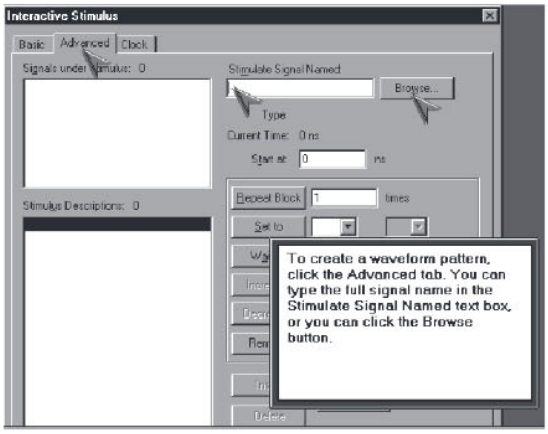


Рис. 6.43. Закладка Advanced для создания сложных входных воздействий

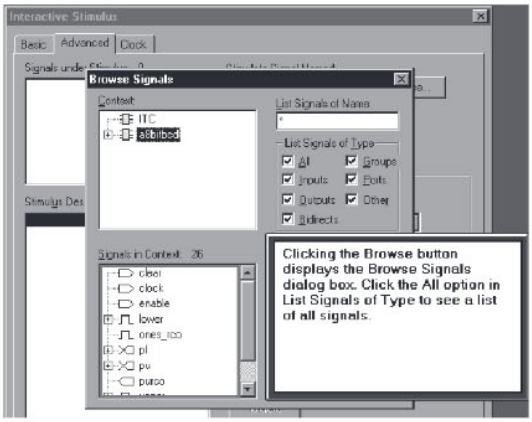


Рис. 6.44. Окно Browse Signal для просмотра сигналов нашей модели

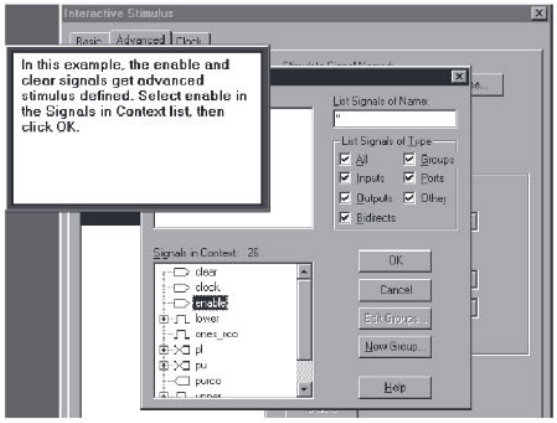


Рис. 6.45. Выбор сигнала enable для описания его поведения

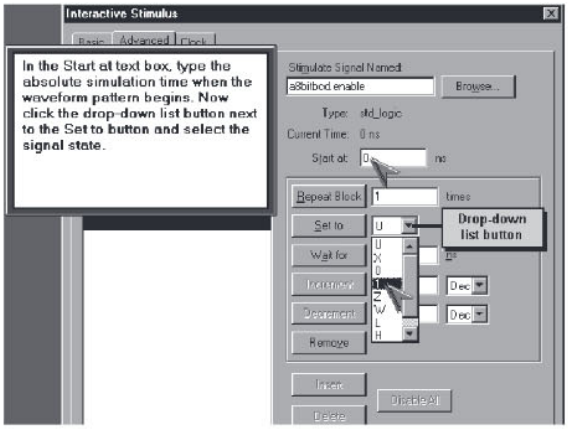


Рис. 6.46. Выбор сигнала enable для описания его поведения — назначение логического события (перехода из 0 в 1 — 0-> 1)

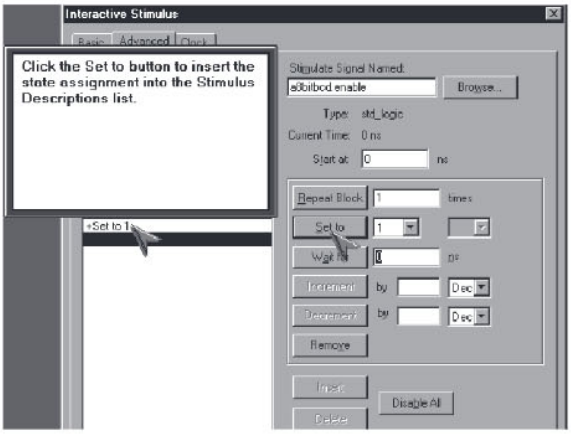


Рис. 6.47. Выбор сигнала enable для описания его поведения — фиксация логического события

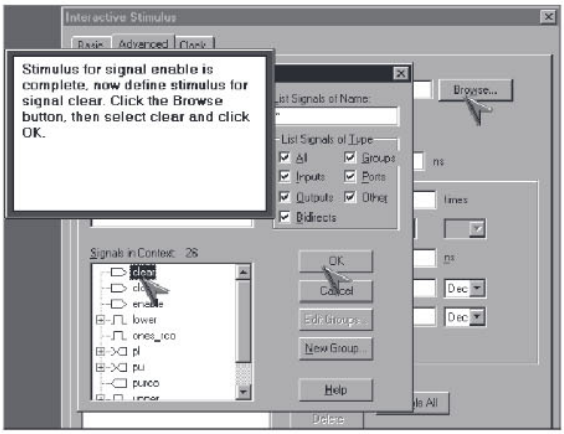


Рис. 6.48. Выбор следующего сигнала — clear

ние данного входного воздействия появится в списке Stimulus Descriptions (рис. 6.49).

Сейчас определим, как долго данное состояние приписывается сигналу clear. Для этого введем данную величину и нажмем кнопку Wait for (рис. 6.50). Для завершения задания входной последовательности для сигнала (входного порта) clear выберем следующее состояние сигнала (Next Sinal State) и щелкнем по кнопке Set to. Список Stimulus Descriptions List отображает все состояния (величины) данного входного воздействия (рис. 6.51).

Далее создадим входное воздействие для порта clock, щелкнув по закладке Clock, а затем по кнопке Browse.

В открывшемся окне в списке Signals in Context выберем сигнал clock (рис. 6.52). Специфицируем время старта, а затем уровень первоначальной паузы и ее длительность (Initial Pulse Signal State u State Duration Time (рис. 6.53). Эти же действия выполняются и для импульса (рис. 6.54).Таким образом, описан один период будущей серии импульсов. С помощью опции Repeat мы можем получить

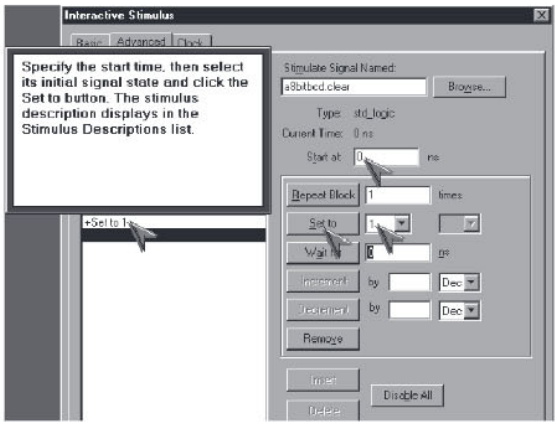


Рис. 6.49. Назначение логического события (0->1) для сигнала clear

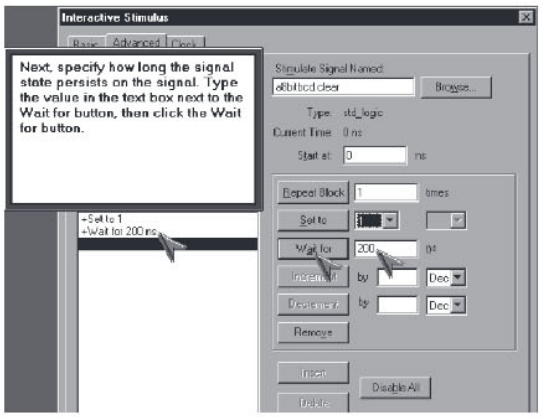


Рис. 6.50. Назначение паузы для сигнала clear

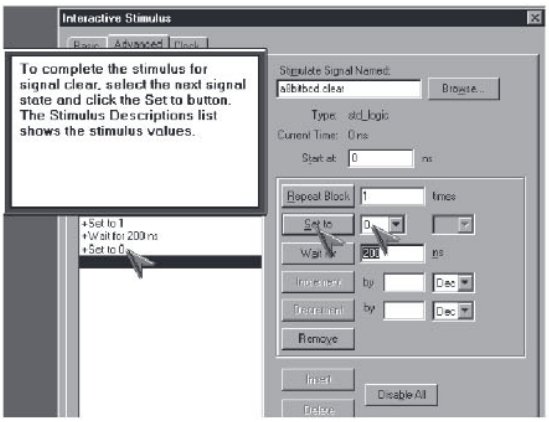


Рис. 6.51. Отображение состояний входного воздействия для порта clear в списке Stimulus Descriptions List

нужное количество таких периодов (Repeat times) или генерировать данную серию импульсов в течение всего интервала моделирования (Repeat Forever). Щелкнув затем по кнопке Add, мы вносим входное воздействие в виде серии импульсов в список Stimulus Descriptions List (рис. 6.55).

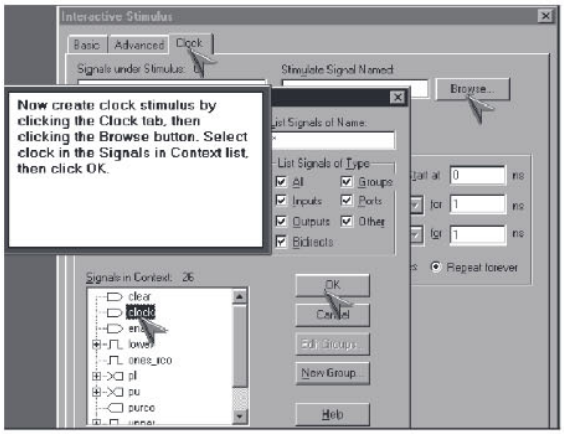


Рис. 6.52. Выбор сигнала (входного порта) clock для формирования соответствующего входного воздействия

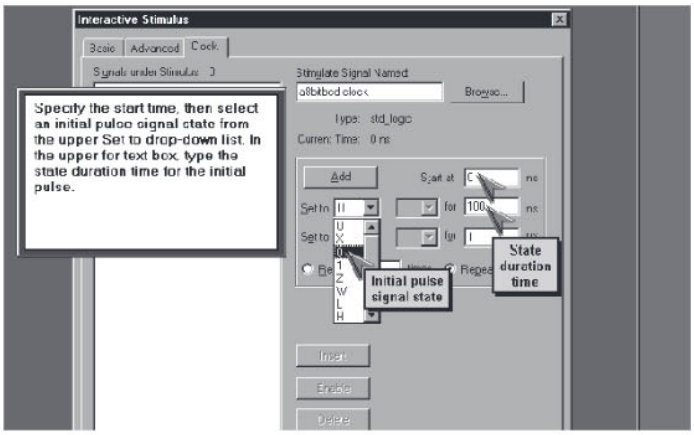


Рис. 6.53. Спецификация времени старта, уровня паузы и ее длительности

Для загрузки созданных в интерактивном режиме внешних воздействий (Interactive Stimulus File) щелкнем по клавише Yes, при этом в окне Stimulus можно наблюдать выбранные сигналы (рис. 6.56).

Для сохранения созданных внешних воздействий необходимо активизировать окно Stimulus, выбрать опцию Save из меню File, задать имя сохраняемого файла и щелкнуть по клавише Save (рис. 6.57). Затем файл внешних воздействий вносится в текущий проект (рис. 6.58).

Когда файл внешних воздействий уже является частью проекта, запустим процесс моделирования выбором опции Run из меню Stimulate и щелчком по клавише Yes (рис. 6.59). Затем подтвердим желание загрузить внешние (входные) воз-

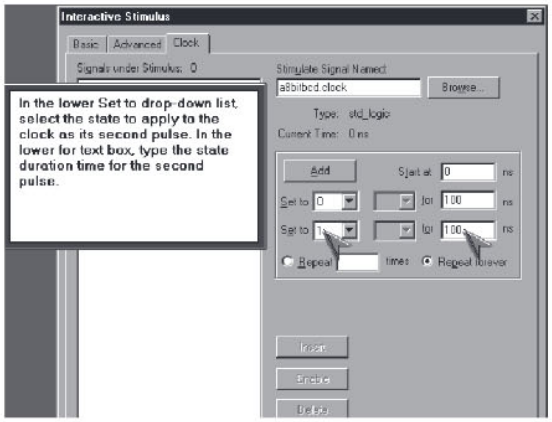


Рис. 6.54. Спецификация уровня импульса и его длительности

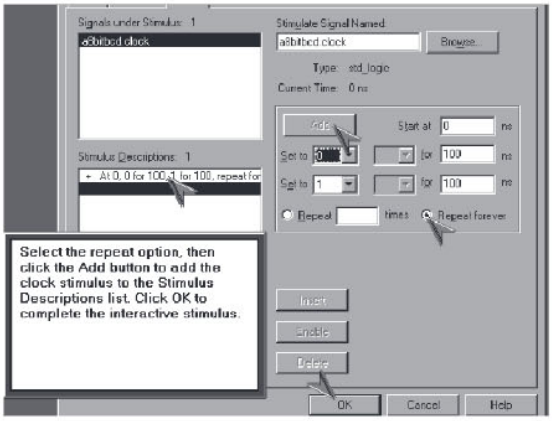


Рис. 6.55. Получение нужного количества периодов для серии импульсов с помощью опции Repeat

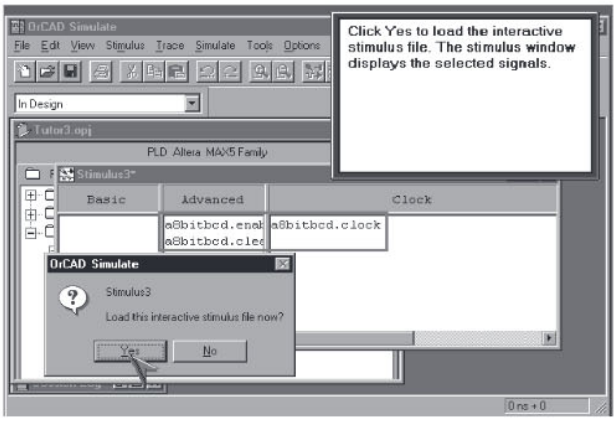


Рис. 6.56. Загрузка созданных в интерактивном режиме внешних воздействий

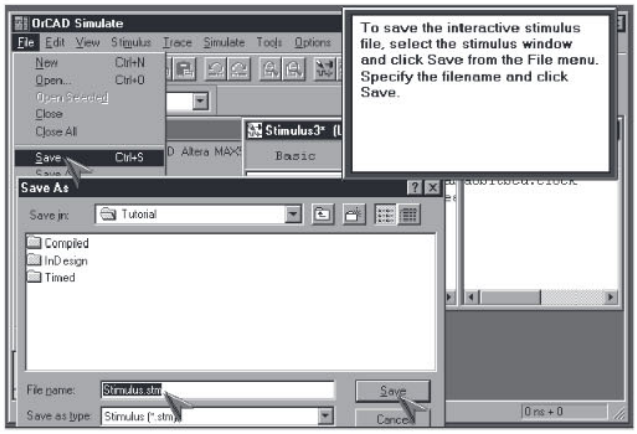


Рис. 6.57. Сохранение файла с интерактивными внешними воздействиями

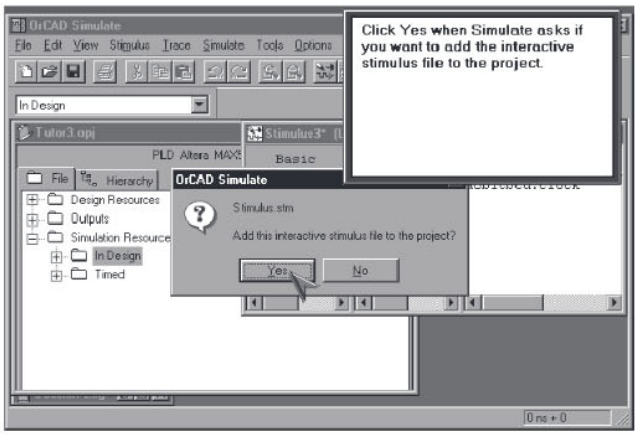


Рис. 6.58. Внесение файла внешних воздействий в текущий проект

действия для осуществления процесса моделирования (рис. 6.60) и назначим временной интервал для него (рис. 6.61).

Результаты моделирования появляются в виде временных диаграмм в окне Wave и в виде таблицы в окне List (рис. 6.62).

Для проведения интерактивного анализа результатов моделирования (Interactive Probing) выберем опцию Preference из меню Options и в открывшемся окне Preference Options отметим опцию Enable Capture/ Simulate Intertool Communication (рис. 6.63).

В процессе интерактивного анализа выбираем определенный сигнал в окне Wave, получаем возможность видеть местонахождение этого сигнала на странице схемы проекта (Schematic Page) (рис. 6.64). Двигая временный курсор (Time Cursor) вдоль выбранной временной диаграммы, мы наблюдаем соответствующие изменения величины сигнала на схемной странице (рис. 6.65).

OrCAD Simulate имеет несколько инструментов для отладки VHDL-файлов. Это особенно полезно при выполнении процесса моделирования в шаговом режиме. Вот эти отладочные инструменты (Stimulation Debugging Tools):

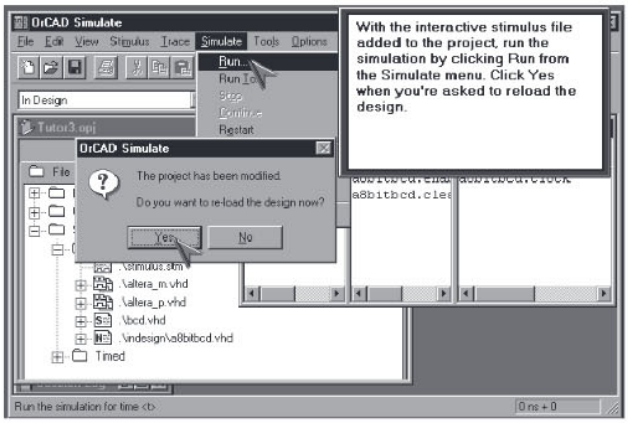


Рис. 6.59. Запуск процесса моделирования

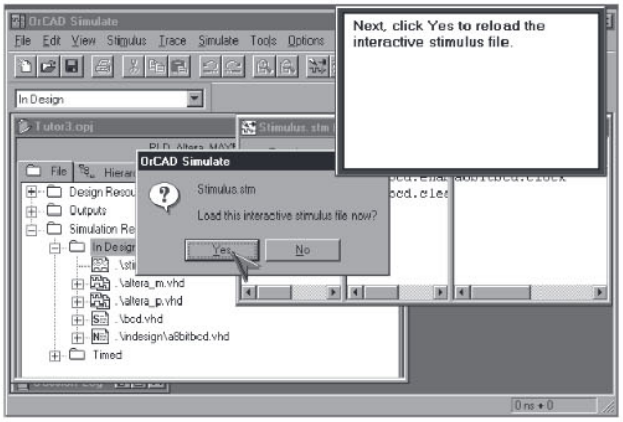


Рис. 6.60. Загрузка внешних (входных) воздействий для выполнения моделирования

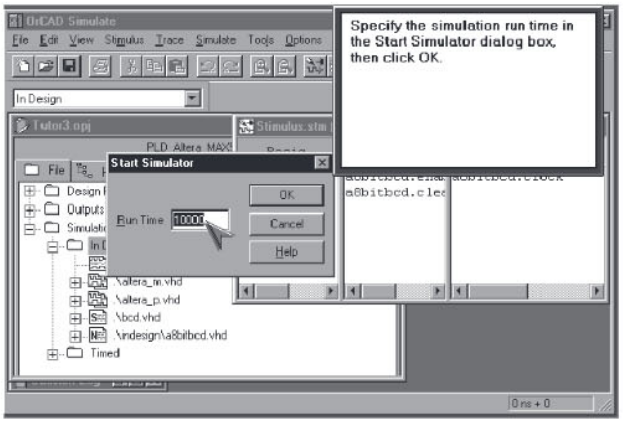


Рис. 6.61. Задание временного интервала для процесса моделирования

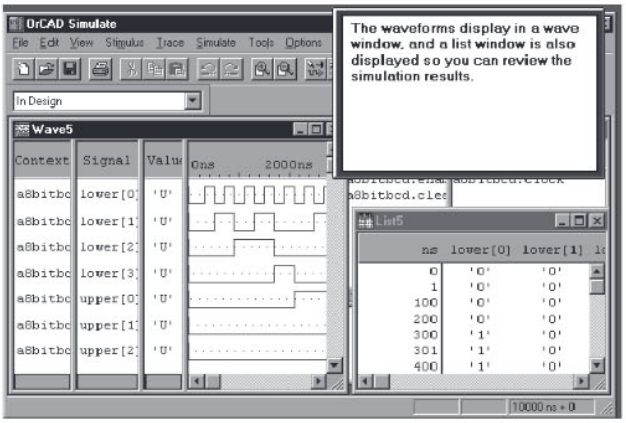


Рис. 6.62. Результаты моделирования

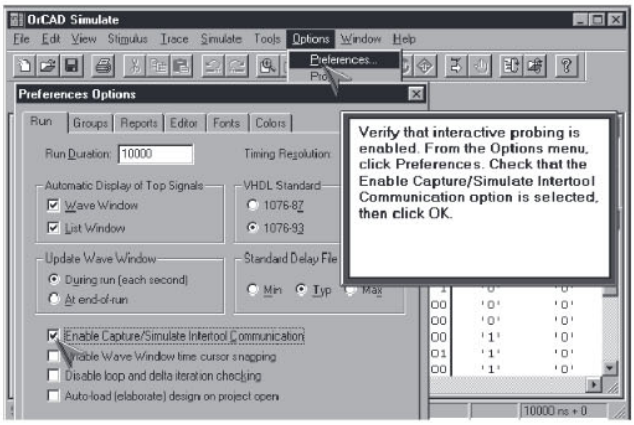


Рис. 6.63. Выбор опции Enable Capture/Simulate Intertool Communication для интерактивного анализа результатов

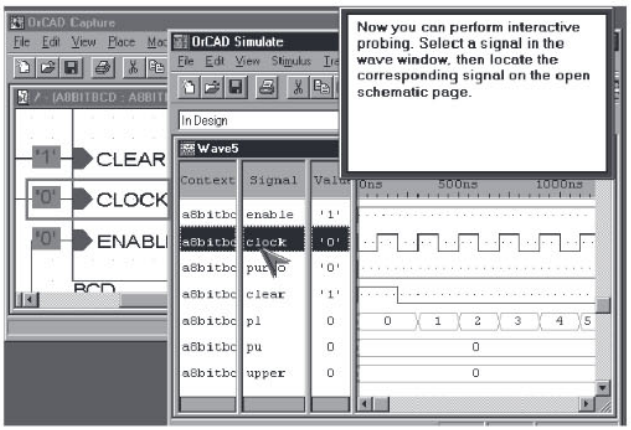


Рис. 6.64. Выполнение интерактивного анализа результатов моделирования (1)

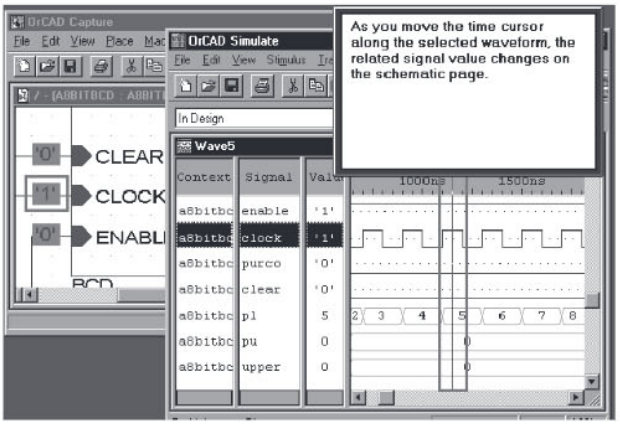


Рис. 6.65. Выполнение интерактивного анализа результатов моделирования (2)

- **Break on Expression**: устанавливает точку останова (breakpoint) в соответствии с некоторым логическим выражением (Logical Expression); если данное выражение получает значение «истинаW (true) в любой момент выполнения процеса моделирования, это приводит к немедленной остановке процесса;
- **Break on Line**: устанавливает точку останова на определенной строке VHDL-кода, и процесс моделирования останавливается перед ее исполнением;
- **Step**: осуществляет выполнение процесса моделирования в пошаговом режиме (на каждом шаге выполняется только одна строка VHDL-кода); при этом симулятор выделяет цветом строку. подлежащую исполнению;
- **Watch Window**: отображает выбранные сигналы и их значения в текущем времени моделирования (Current Simulation Time) — это полезно для режима пошагового моделирования.

Для выполнения процесса моделирования в пошаговом режиме необходимо выбрать опцию Step в меню Simulate или щелкнуть по соответствующей кнопке в линейке инструментов (рис. 6.66). При этом специальный маркер будет указывать на строку VHDL-кода, которая будет выполняться следующей (рис. 6.67).

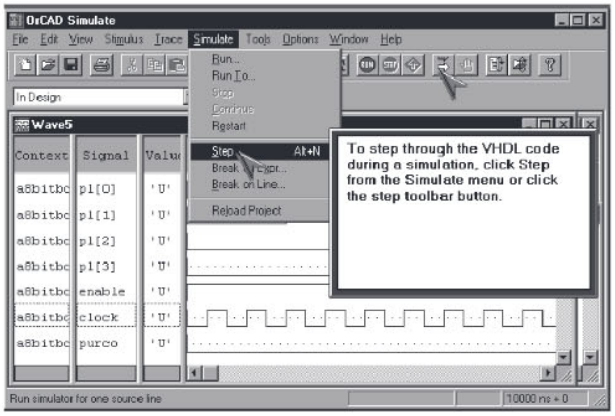


Рис. 6.66. Выбор пошагового режима моделирования

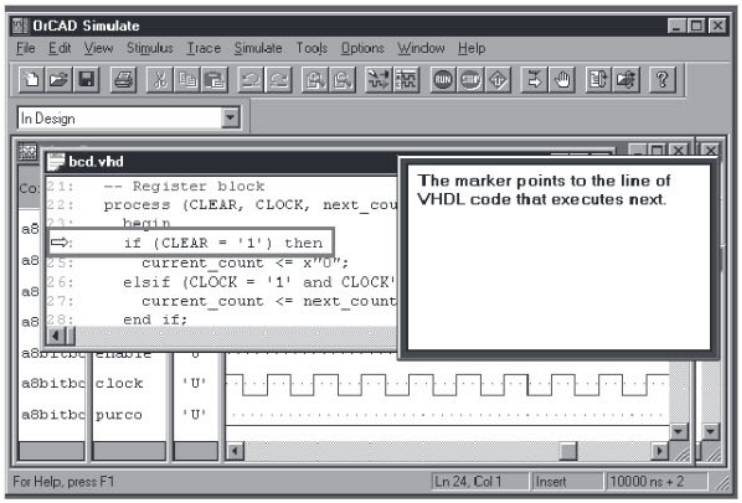


Рис. 6.67. Маркер, указывающий на строку VHDL-кода, которая будет выполняться следующей

Для выполнения каждого следующего шага необходимо щелкнуть по упомянутой выше кнопке в линейке инструментов или набрать на клавиатуре ALT+N . Такая техника позволяет решить точечные логические проблемы VHDL-кода (рис. 6.68).

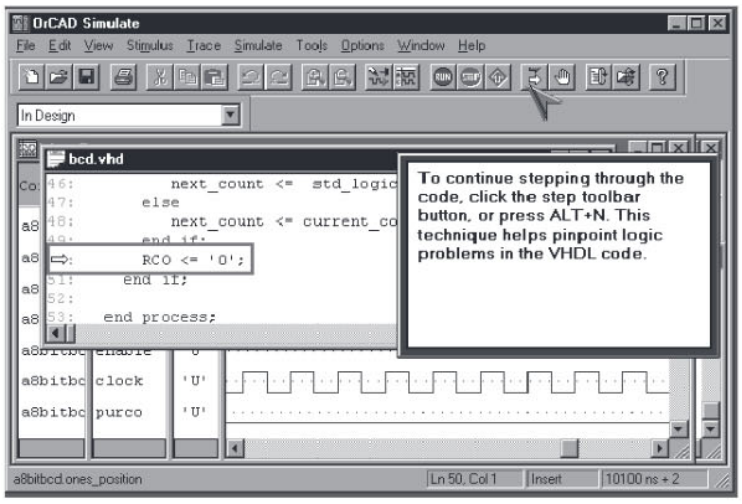


Рис. 6.68. Выполнение очередного шага моделирования

Окно Watch является другим отладочным инструментом пошагового режима моделирования. В этом окне отображаются выбранные сигналы и их значения для текущего времени моделирования (рис. 6.69). Можно также использовать это окно для просмотра состояний внутренних VHDL-переменных (Internal VHDL-Variables) в процессе моделирования. Текущее время моделирования индицируется над списком сигналов (рис. 6.70).

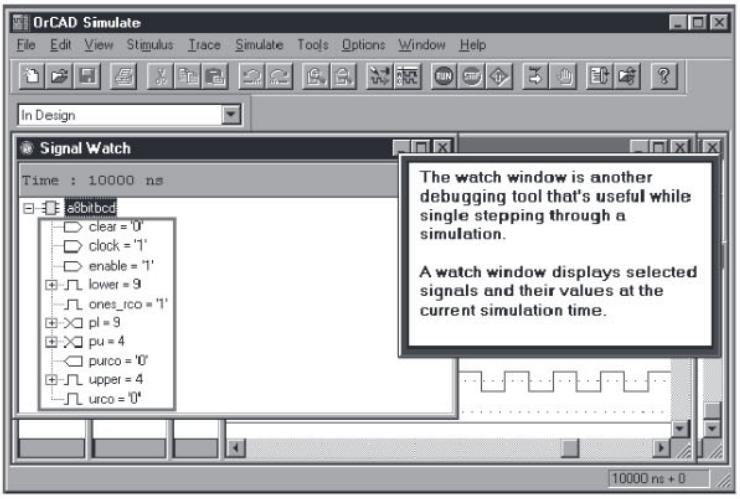


Рис. 6.69. Индикация значений сигналов и переменных в окне Signal Watch (1)

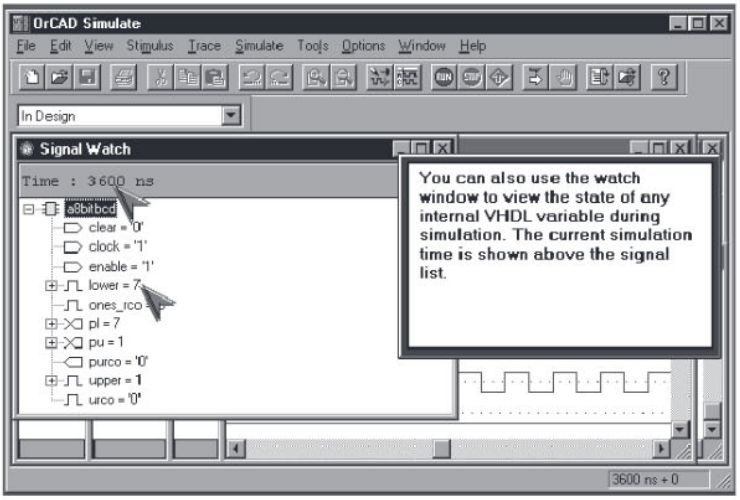


Рис. 6.70. Индикация значений сигналов и переменных в окне Signal Watch (2)

Глава 7. Работа с VHDL в среде системы Warp

В этой главе мы обсудим в общем случае последовательность операций в средах Warp2 и Warp3 по созданию, компиляции, синтезу и моделированию проектов, описанных на языке VHDL [71a, 72a].

7.1. Общее описание системы Warp

Warp2 позволяет пользователю описывать электронные цифровые проекты, используя язык VHDL, а затем компилировать и синтезировать эти описания в программируемые приборы (ПЛИС) фирмы Cypress Semiconductor: малые PLD's, Max340 EPLD's, Flash370 CPLD's и pASIC380 FPGAs.

Warp 2 включает три главных компонента:

1. Warp VHDL Compiler, который соответствует стандарту IEEE 1076/1164 и транслирует текстовые описания проекта на VHDL в форматы (файлы) JEDEC (Joint Electronic Device Engineering Council) и QDIF (Quick Design Interchange Format);
2. Nova JEDEC Functional Simulator, который позволяет верифицировать и затем корректировать проект с помощью моделирования его поведения;
3. SpDE™ Toolkit, содержащий набор инструментов для размещения (Fitting) проекта в ПЛИС pASIC380 FPGAs. Этот набор включает следующие основные инструменты:
   - планировщик размещения (Placer);
   - программу трассировки (Router);
   - логический анализатор (Logic Optimizer);
   - анализатор путей в схеме проекта (Path Analyzer).

Warp 3 позволяет описывать, компилировать и синтезировать VHDL- проекты, а также проекты, представленные набором схем (Schematic Descriptions), или комбинацией VHDL-описаний (VHDL Descriptions) со схемами проекта, в ПЛИС фирмы Cypress Semiconductor (Cypress PLDs).

Warp3 интегрирует в своей среде все компоненты Warp 2, а также две дополнительные программные среды:

1. Viewlogic Workview PLUS (для персональных компьютеров IBM PC и совместимых с ними);
2. Powerview CAE (для рабочих станций операционной системы UNIX).

Среда Viewlogic представлена иконкой Cockpit (рис. 7.1), которая является центральной точкой доступа ко всем инструментам Warp3. Среда Viewlogic представляет пользователю большой набор инструментов для реализации проектов. Вот некоторые из них:

- View Draw (иерархический схемный и символьный редактор);
- View Sim (мощный симулятор, максимально точно моделирующий временные задержки);
- View Trace (инструмент просмотра временных диаграмм, отражающих результаты моделирования).

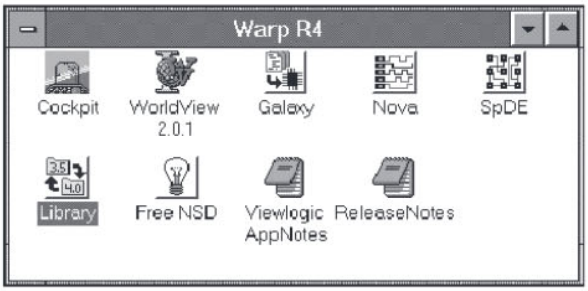


Рис. 7.1. Окно программной группы Warp R4

7.2. Пример разработки проекта

В этом примере будем использовать поведенческое VHDL-описание для объектов нижнего уровня проекта и VHDL-файл верхнего уровня проекта, использующий компоненты, описанные на нижнем уровне проекта.

В качестве примера спроектируем контроллер для автомата по продаже газированной воды. Этот автомат имеет два бункера для разлива обычной и диетической колы. Каждый бункер содержит три бидона напитка.

Наша схема должна разливать напиток при условии, что нажата соответствующая кнопка и один бидон (или более) доступен. Введем сигнал refill, который устанавливается в 1, когда нет доступных бидонов с напитком, и сигнал reset, установка которого в 1 означает, что все бункеры автомата вновь полны.

Здесь будет с помощью VHDL описана схема, которая управляет одним бункером (ему ставится в соответствие сигнал get_drink, чье значение является числом доступных бидонов в бункере; установка сигнала empty в 1 означает, что бункер уже пуст и нуждается в наполнении). Схема имеет имя binctr.

Мы получим описание верхнего уровня проекта (Top-Level Description), которое содержит два экземпляра binctr. Это описание получит имя refill. После этого описание будет скомпилировано и синтезировано в файл CY7C371 JEDEG, а затем будет выполнено моделирование поведения результирующего проекта с использованием симулятора Nova Functional Simulator.

Для запуска Warp 2 в ОС Windows необходим двойной щелчок по иконке Galaxy в окне программной группы Warp R4 (рис. 7.1). Для запуска же Warp3 необходим двойной щелчок по иконке Cockpit в том же окне.

7.3. Создание проекта

Двойной щелчок по иконке Galaxy приводит к появлению окна Galaxy (рис. 7.2), а двойной щелчок по иконке Cockpit приводит к появлению окна Workview PLUS Cockpit (рис. 7.3).

Пользователь Warp 2 в окне Galaxy для создания нового проекта должен щелкнуть по кнопке New в секции Edit правой клавишей мыши. Это приводит к открытию окна текстового редактора (VHDL Editor) (рис. 7.4). Пользователь Warp 3 для этой же цели щелкает по иконке Warp в окне Cockpit, а затем, после появления окна Galaxy щелкает по той же кнопке New.

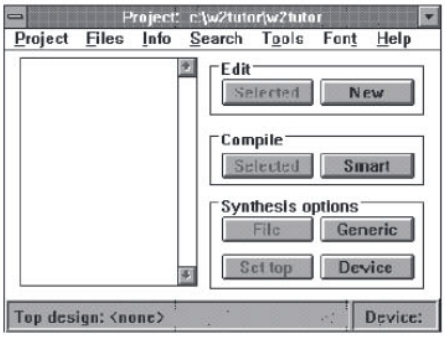


Рис. 7.2. Окно Galaxy

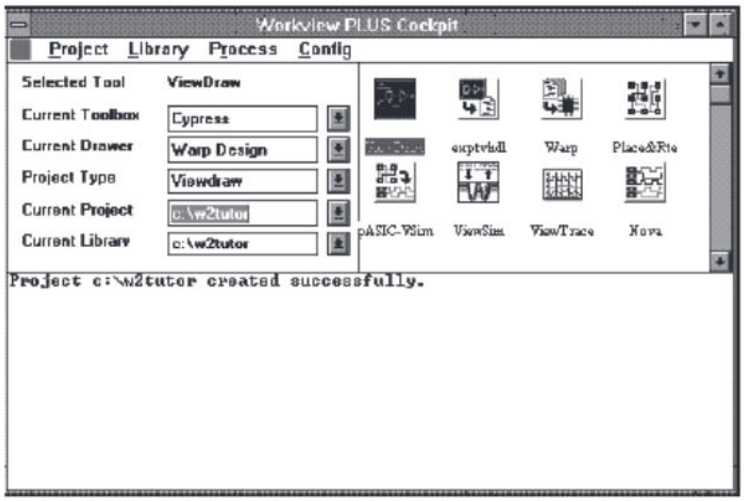


Рис. 7.3. Окно Workview PLUS Cockpit

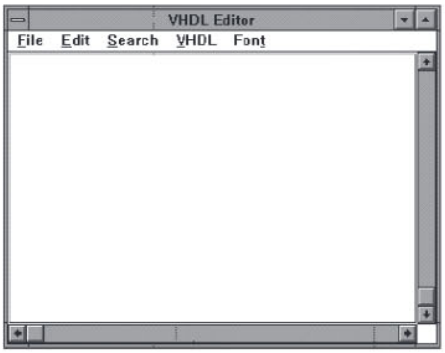


Рис. 7.4. Окно VHDL-редактора

VHDL-описание для binctr будет содержать три части:

- интерфейс (Entity Declaration), содержащий имя, направление и тип данных для каждого порта компонента binctr;
- архитектуру (Architecture), описывающую поведение этого компонента;
- программный пакет (Package Declaration), обеспечивающий соответствующую информацию о том, что компонент binctr будет использоваться на верхнем уровне проекта (High-Level Design).

Опишем интерфейс компонента binctr в окне VHDL-редактора (рис. 7.5) следующим образом:

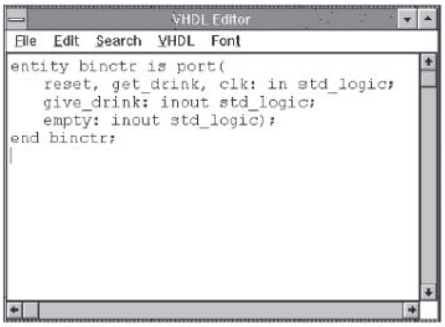


Рис. 7.5. Ввод описания интерфейса компонента binctr

```
entity binctr is port(
      reset, get_drink, clk        : in          std_logic;
      give_drink                   : inout       std_logic;
      empty                        : inout       std_logic);
end binctr;
```

Затем в том же окне введем описание нашего компонента:

```
architecture archbinctr of binctr is
constant full:         std_logic_vector(1 downto 0) := "11";
--max of 3 drinks/bin
signal remaining:    std_logic_vector(1 downto 0);
begin
    proc_label: process (clk,reset)
                begin
                if (reset='1') then
                        remaining  <=  full;
                        empty      <=  '0';
                        give_drink <= '0';
                elsif (clk'event and clk='1') then
                  if (remaining = "00") then
                        empty      <= '1';
                        give_drink <= '0';
                  elsif (get_drink ='1') then
                        remaining <= remaining - 1;
                        give_drink <= '1';
```

```
            elsif (get_drink = '0') then
                  give_drink <= '0';
            else
                  give_drink <= give drink;
                  remaining  <=  remaining ;
                  empty      <=  empty;
            end if;
          end if;
          end process;
    end archbinctr;
```

Теперь введем в окне VHDL-редактора перед описанием интерфейса текст программного пакета, содержащего определение для binctr как базового компонента:

```
    package binctr_pkg is
        component binctr
            port (reset,get_drink,clk      : in     std_logic;
                  give_drink               : inout  std_logic;
                  empty                    : inout  std_logic);
        end component;
    end binctr_pkg;
```

Этот пакет позволяет компилятору Warp Compiler использовать binctr как базовый компонент на верхнем уровне проекта.

В заключение, прибавим к проекту еще три строки перед заголовком пакета binctr_pkg, которые позволяют пользователю использовать в тексте проекта тип данных stg_logic, а также различные арифметические операции, описанные в пакете std_arith, находящемся в рабочей библиотеке.

Для проверки синтаксиса проекта, прежде всего, сохраним наш VHDL-файл binctr.vhd, выбрав опцию Save из меню File. Затем, в меню VHDL выберем опцию Compile. При этом запускается компилятор, который печатает сообщения о ходе компиляции (рис. 7.6). В данном случае налицо отсутствие сообщений об ошибках, что говорит об успешной компиляции. Поэтому закроем VHDL -файл, выбрав опцию Close из меню File.

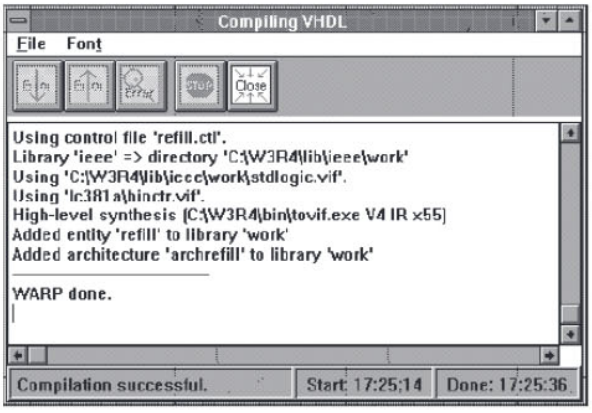


Рис. 7.6. Сообщения о ходе компиляции, выдаваемые компилятором

После того как мы определили поведение на нижнем уровне проекта, опишем верхний уровень, содержащий два экземпляра компонента binctr, и соединения между ними с помощью соответствующих внутренних сигналов.

Для этого вновь откроем VHDL-редактор в режиме создания нового VHDL-файла и введем следующие строки:

```
library ieee;
use ieee.std_logic_1164.all;
use work.binctr_pkg.all;

entity REFILL is
   port (GIVE_cola:     inout    std_logic;
         GIVE_diet:     inout    std_logic;
         REFILL_BINS:   out      std_logic;
         RESET:         in       std_logic;
         CLK:           in       std_logic;
         GET_diet:      in       std_logic;
         GET_cola:      in       std_logic);

attribute pin_numbers of refill: entity is
"GIVE_cola: 2 GIVE _ diet: 3 REFILL_ BINS: 4 RESET: 10" &
"CLK: 13_ diet: GET_diet: 11 GET _ cola: 35";

end REFILL;
```

Здесь атрибут присваивания (Assignment Attribute) pin_numbers определяет соответствие между сигналами, декларированными внутри entity REFILL, и специфическими номерами контактов пользователя (User Specified Pin Numbers). Эти номера присваиваются сигналами проекта и являются номерами контактов ПЛИС CY7C371 в корпусе PLCC. Если для физической реализации проекта будет использоваться другая ПЛИС или другой корпус, то эти номера контактов необходимо изменить соответственно.

Теперь введем текст соответствующего архитектурного тела для entry REFILL в окне VHDL-редактора (рис. 7.7), завершая формирование VHDL-файла refill. vhd:

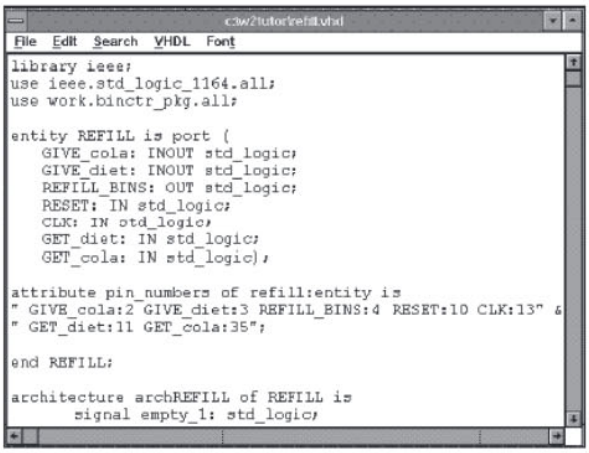


Рис. 7.7. Окончательный вид окна VHDL-редактора с файлом refill.vhd

```
architecture archREFILL of REFIIL is
        signal empty_ 1: std_logic:
        signal empty_2 : std_ logic;
begin
        bin_1: BINCTR
                port map (RESET => RESET,
                                GET_DRINK  => GET_cola,
                                CLK          => CLK,
                                GIVE_DRINK => GIVE_cola,
                                EMPTY        => empty_1);

        bin_2: BINCTR
                port map (RESET        => RESET,
                                GET_DRINK  => GET _diet,
                                CLK          => CLK,
                                GIVE_ DRINK => GIVE_diet,
                                EMPTY => empty_2);

         refill_bins < = `1` when ((empty_1=`1`) and (empty _2 =`1`))
                                else `0`;

end archREFILL;
```

Таким образом, верхний уровень проекта (рис. 7.8) уже написан и нуждается в сохранении выбором опций Save As из меню File. В качестве имени сохраняемого файла введем refill.vhd и закроем этот файл.

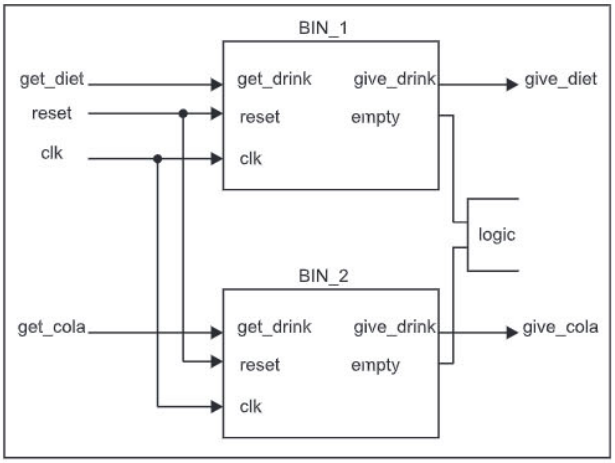


Рис. 7.8. Структурная схема проекта, описанного в файле refill.vhd

7.4. Компиляция и синтез файла верхнего уровня (TOP-LEVEL FILE)

Прежде всего добавим файл верхнего уровня к нашему проекту. Для этого, в меню Files окна Galaxy выберем опцию Add. В появляющемся диалоговом окне

(рис. 7.9) выберем refill.vhd для добавления его к проекту, уже содержащему файл binchr.vhd.

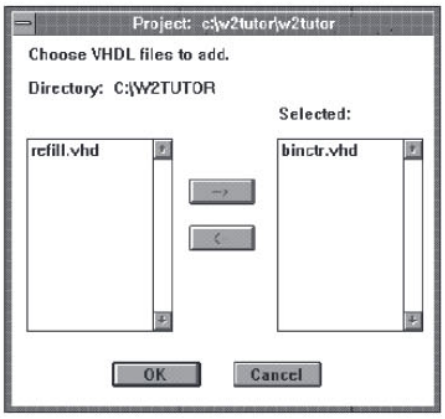


Рис. 7.9. Диалоговое окно добавления файлов к проекту

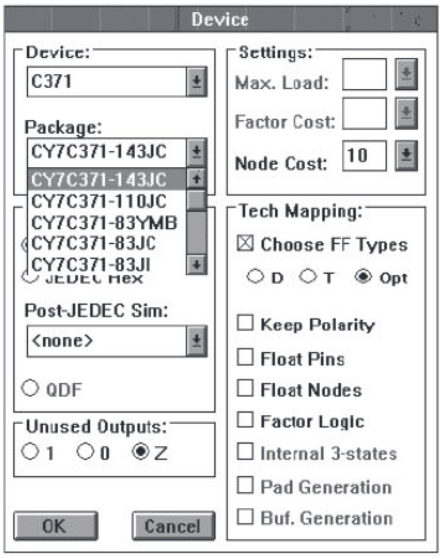


Рис. 7.10. Окно для выбора ПЛИС

Теперь щелкнем по клавише Device в группе Synthesis окна Galaxy для выбора ПЛИС, предназначенного для физической реализации проекта. В соответствующем окне (рис. 7.10) выберем тип ПЛИС-C371 и соответствующий тип корпуса (например, CY7C371-143JC).

Система Warp представляет пользователю возможность выбрать один из трех вариантов настройки неиспользуемых выходов ПЛИС: установить их в состояние логической единицы, логического нуля или в «третье» состояние Z (рис. 7.10). Мы выберем 'Z', и это будет означать, что по умолчанию все используемые выходы будут установлены в состояние Z.

В группе опций Tech Mapping этого же окна выбираем опцию Opt, которая позволяет программе размещения (Fitter) выбирать между триггерами DFF и TFF, чтобы получить реализацию логических формул с наименьшим числом элементов (Product Terms).

Настало время «официально» объявить refill. vhd файлом верхнего уровня нашего проекта. Для этого в окне Galaxy (рис. 7.11) щелкнем по кнопке Set Top в группе Synthesis options.

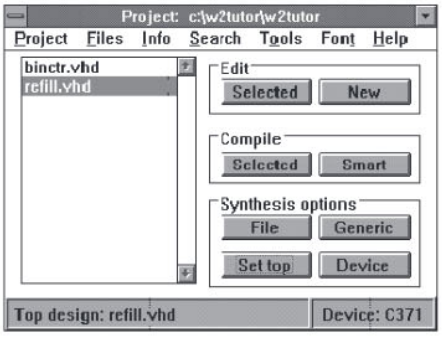


Рис. 7.11. Выбор файла верхнего уровня проекта (Top Design: refill.vhd) в окне Galaxy

Откомпилируем VHDL-файл refill.vhd, для чего щелкнем по кнопке Smart в группе Compile окна Galaxy. В результате генерируются два следующих файла:

- refill.jed: он может быть использован для программирования ПЛИС CY7C371, а также является входным файлом для функционального симулятора Nova;

- refill.rpt: содержит временную (и некоторую другую) информацию о синтезированном проекте. Можно прочесть его, выбрав в меню Info окна Galaxy опцию Report.

После компиляции и синтеза файла верхнего уровня проекта закроем окно Galaxy щелчком по кнопке Close вверху окна Galaxy.

7.5. Моделирование проекта с помощью симулятора Nova

Теперь мы должны выполнить моделироваие проекта, чтобы верифицировать его поведение (функции) с помощью симулятора Nova.

Для старта Nova JEDEC Functional Simulator в среде Warp 2 необходимо щелкнуть по кнопке Nova в программной группе WarpR4, а в среде Warp 3 по иконке Cockpit в той же группе.

В окне Nova откроем файл refill.jed, выбрав в меню File опцию Open (рис. 7.12). После щелчка по кнопке OK окно Nova приобретает вид, приведенный на рис. 7.13).

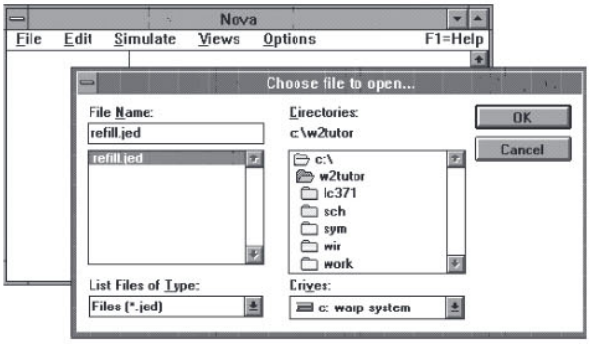


Рис. 7.12. Открытие файла refill.jed в окне Nova

Прежде всего оставим в окне Nova только те сигналы, которые представляют интерес. Для этого в меню Views выбираем опцию Edit Views и щелкаем по кнопке New View. При этом появляется диалоговое окно, приглашающее ввести новое имя для окна и временные диаграммы только для интересующих нас сигналов (рис. 7.14). Введем в качестве имени tutview и щелкнем по кнопке OK, закрывая это окно диалога.

Выберем следующие сигналы с помощью щелчка по имени сигналов, а затем щелчка по кнопке Add:

- clk;
- reset;
- get_cola;
- give_cola;
- give_diet;
- refill_bin.

После этого окно Nova приобретает вид, приведенный на рис. 7.15.

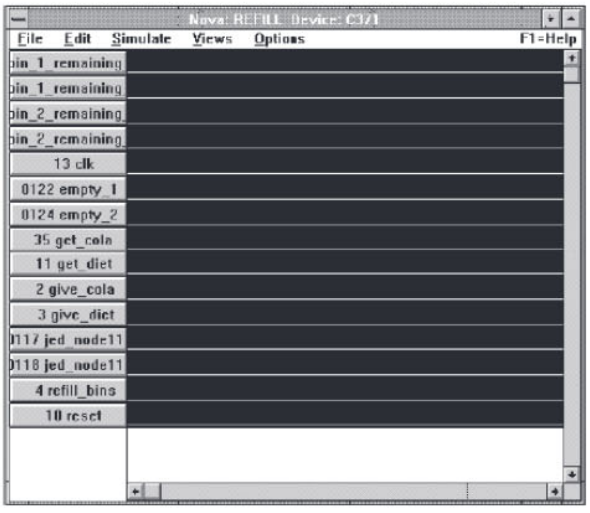


Рис. 7.13. Первоначальный вид окна Nova для файла refill.jed

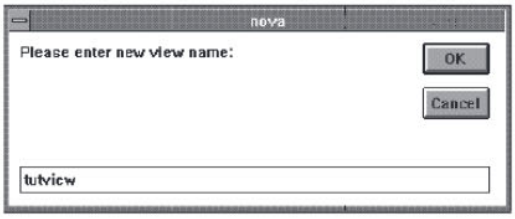


Рис. 7.14. Диалоговое окно, приглашающее ввести новое имя для окна временных диаграмм

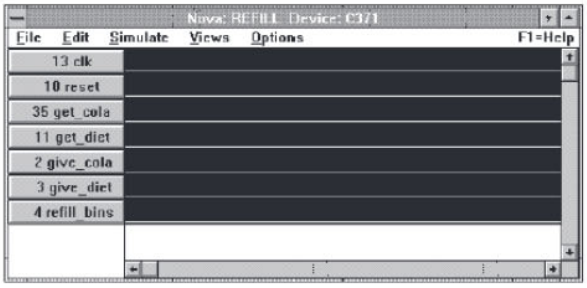


Рис. 7.15. Окно Nova tutview

Установим теперь величину интервала моделирования (Simulation Length), выбрав в главном окне Nova из меню Options опцию Simulation Length. В одноименном диалоговом окне установим интервал моделирования, равный 384 единицам времени моделирования, и, нажав на кнопку OK, закроем это окно.

Позиционируем курсор на клавише сигнала clk и щелкнем левой клавишей мыши, это приведет к тому, что траектория сигнала будет выделена голубым цветом. Затем выберем в меню Edit опцию Clock и в появляющемся диалоговом окне Edit/Clock щелкнем по клавише OK для принятия по умолчанию периода для сигнала Clock в 10 единиц времени моделирования (рис. 7.16).

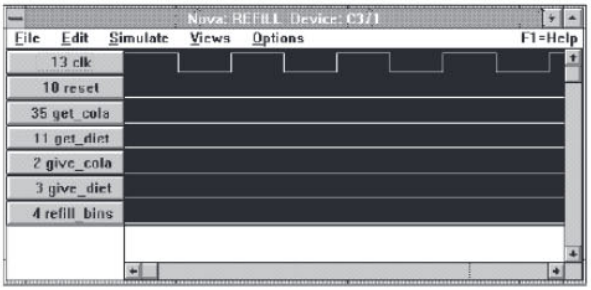


Рис. 7.16. Определение сигнала Clk

Вдобавок к тактовому сигналу Clock необходимо установить значения для следующих входных сигналов при выполнении процесса моделирования: reset, get_cola и get_diet. Конечный результат такой установки значений показан на рис. 7.17. Для примера расскажем об установке сигнала reset. Вначале установим

reset на уровень логической единицы для одного переднего фронта синхроимпульса Clock. Для этого:

- позиционируем курсор на сигнале reset слева от переднего фронта синхроимпульса;
- щелкнем, а затем прижмем левую клавишу мыши;
- «протащим» курсор вдоль линии сигнала вправо от переднего фронта clock и затем, в нужный момент, отпустим клавишу мыши — этот интервал сигнала reset приобретет голубой цвет;
- нажмем клавишу `1` на клавиатуре для установки сигнала reset в состояние логической единицы на выбранном интервале. В результате получаем импульс reset, окрашенный голубым цветом.

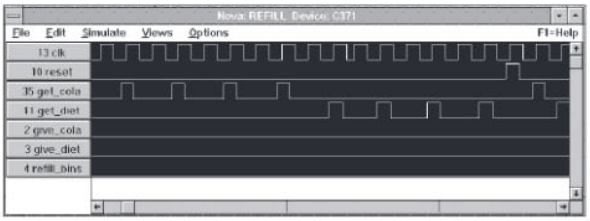


Рис. 7.17. Установка входных сигналов для выполнения моделирования

Для моделирования проекта выберем опцию Execute из меню Simulate. Результаты моделирования приведены на рис. 7.18.

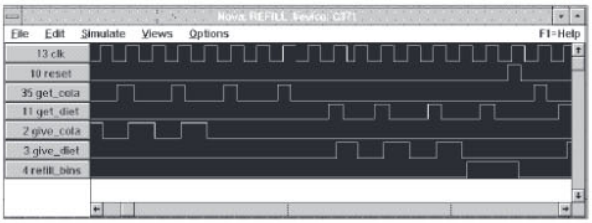


Рис. 7.18. Результаты моделирования проекта

Процесс моделирования начинается из состояния пустого автомата по разливу напитков (состояние сигнала refill_bin является стартовым для процесса моделирования).

Когда сигнал reset устанавливается в состояние логической единицы в начале процесса моделирования, сигнал refill_bin устанавливается в состояние логического нуля — это означает, что сейчас автомат готов к разливу напитков. В дальнейшем автомат разливает три порции колы на первые три требования (см. соотношения между импульсами сигналов get_cola и give_cola). После четвертого требования колы машина остается безучастной — бункер с колой пуст. Теперь автомат способен разливать только диетическую колу: он выдает три порции напитка в ответ на первые три соответствующих требования. После четвертого требования реакции не следует — бункер с диетической колой тоже пуст. В момент, когда оба бункера стали пустыми, устанавливается в единицу сигнал refill_bin. Он остается в этом состоянии до тех пор, пока не установится в единицу сигнал reset, сообщающий автомату, что оба бункера вновь наполнены.

7.6. Реализация проекта с помощью FPGA

Система Warp поддерживает применение ПЛИС семейства pASIC380 Family архитектуры Ultra Logic™ FPGAs. В этом разделе мы синтезируем наш проект в ПЛИС CY7C381A FPGA. Синтез осуществляется с помощью интерфейса Galaxy, а размещение и трассировка (Place & Route) проекта выполняются с помощью набора инструментов SpDE Toolkit. Этот набор инструментов генерирует все необходимые файлы для моделирования в среде View-Sim, которая является третьей частью в ряду средств моделирования. В предыдущих разделах файл refill.vhd компилировался для получения JEDEC-файла. В дальнейшем же, мы с помощью Warp получим QDIF-файл для ПЛИС CY7C381A-OJC 1k gate CMOS FPGA.

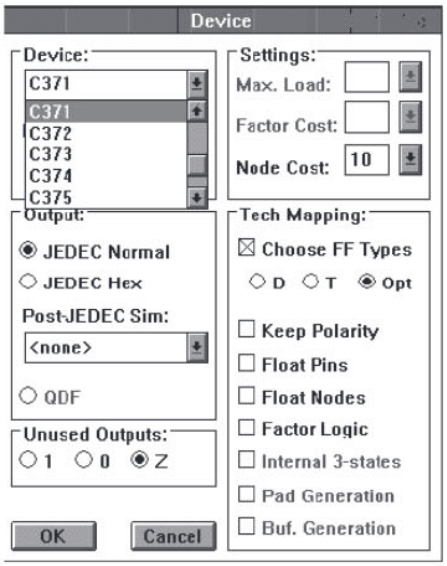


Рис. 7.19. Диалоговое окно Device

Вновь запустим окно Galaxy (об этом уже подробно сказано ранее), а затем выберем ПЛИС для физической реализации проекта щелчком по кнопке Device в группе Synthesis options окна Galaxy. При этом возникает диалоговое окно Device (рис. 7.18), в котором мы выберем тип ПЛИС и тип соответствующего корпуса, а также установим опции выходов проекта.

Прежде всего в списке Device выберем C381A. Вдобавок к этому необходимо выбрать тип корпуса в окне списка Package (рис. 7.9, рис. 7.19). Выберем тип корпуса CY7C381A-OJC.

Атрибуты pin_number, которые ранее использовались в refill.vhd, были специфическими для ПЛИС CY7C371143JC. Для нынешней же компиляции выберем опцию Float Pins в группе Tech Mapping. Остальные же параметры группы Tech Mapping выберем по умолчанию.

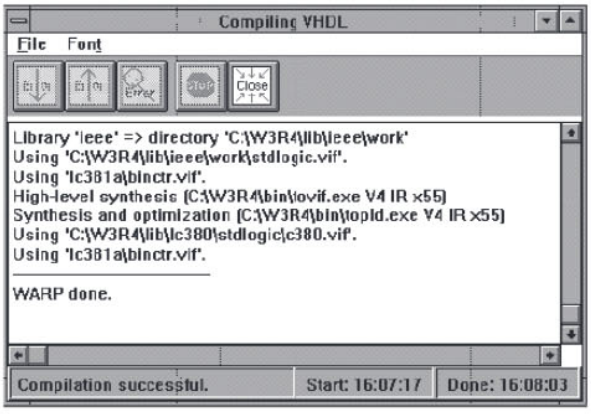


Рис. 7.20. Сообщения о ходе выполнения компиляции и синтеза

Запустим процесс компиляции (как уже было описано ранее). Система Warp , выполняя компиляцию и синтез проекта в CY7C381A, печатает сообщения, отражающие процесс выполнения компиляции и синтеза (рис. 7.20).

В результате выполнения этих операций генерируется два файла:

– refill.qdf: он используется как входной файл для

SpDE. SpDE выполняет при этом операции размещения и трассировки (Place & Route), генерируя файл с расширением *. lof, который используется для программирования ПЛИС;

– refill. rpt: он содержит информацию о синтезированном проекте и читается выбором опции report из меню Info окна Galaxy.

Сейчас сгенерируем QDIF-файл, соответствующий файлу refill.vhd, чтобы получить синтезируемое описание проекта для следующего размещения его в приборе FPGA. Такое размещение может быть выполнено с помощью набора инструментов SpDE, после чего у пользователя есть возможность получить необходимые файлы для моделирования синтезированного устройства в среде ViewSim.

Для вызова SpDE необходимо выбрать соответствующую опцию из меню Tools в окне Galaxy (это одинаково верно для Warp2 и Warp3).

Щелкнем по клавише folder, размещенной в верхней левой части окна SpDE (это эквивалентно выбору File-> Import -> QDIF). Затем в открывшемся окне выберем файл refill.qdf соответствующим щелчком, а затем щелкнем по клавише OK для импорта этого файла.

После импорта файла с помощью клавиши hammer (которая становится доступной в этот момент) выберем инструмент для размещения и трассировки.

Запуская процесс размещения и трассировки, щелкнем по клавише Run. Этот процесс займет порядка двух минут. После его успешного завершения мы получаем сообщение «All chosen SpDE tools ran successfully». Закрыв окно с этим сообщением, щелкнем по клавише OK и выберем опцию Full Fit из меню View. Это позволит нам увидеть свой проект размещенным внутри определенного прибора (рис. 7.21).

Сохраним наш проект выбором опции Save из меню File, а затем, заканчивая настоящий режим, выберем опцию Exit в том же меню.

Для будущего моделирования результатов работы SpDE необходимо сгенерировать модель для среды моделирования ViewSim (ViewSim Model), для чего:

– сделаем двойной щелчок по иконке pASIC -> VSIM в окне Cockpit,
– в появившемся окне в командной строке введем refill и щелкнем по кнопке OK.

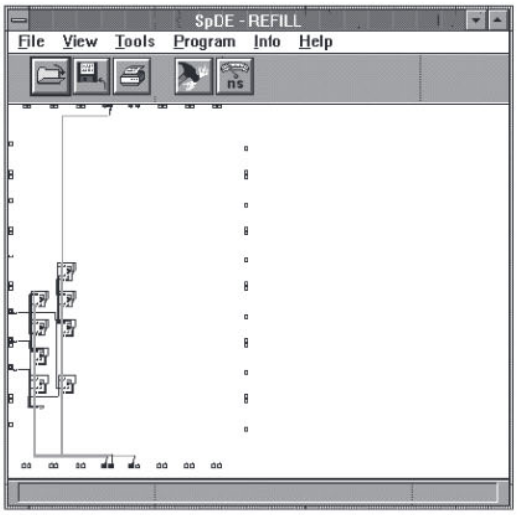


Рис. 7.21. Результаты работы SpDE для файла refill.qdf

7.7. Получение информации о присвоенных контактам номерах

Для получения информации о присвоенных контактам номерах (Pin Assignment Information) необходимо сделать следующее:

– двойной щелчок по иконке Galaxy в окне Cockpit;

– выбор файла refill.vhd соответствующим щелчком;
– выбор опции Annotate из меню File.

При этом появляется малое окно (рис. 7.22), позволяющее задать вид информации, которую мы хотим получить (о контактах, внутренних узлах или то и другое).

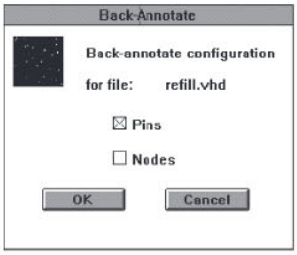


Рис. 7.22. Диалоговое окно для получения информации о номерах, присвоенных контактам

Требуемая информация будет сохранена в файле с расширением *.ctl.

7.8. Моделирование поведения проекта с помощью ViewSim

После синтеза проекта целесообразно промоделировать его поведение с целью оценить его временные характеристики (timing performance). Это гарантирует нам выполнение всех, определенных заранее, его функций. Для вызова симулятора ViewSim необходим двойной щелчок по иконке ViewSim в окне Cockpit. В появляющемся диалоговом окне введем имя проекта refill и щелкнем по клавише OK. При этом стартует симулятор ViewSim. В его главном окне (рис. 7.23) после появления промпта 'SIM>' введем имя refill в командной строке, что приведет к запуску командного файла refill.cmd, выполняющего следующую последовательность команд ViewSim:

```
wave REFILL.wfm clk reset get_cola give_cola get_diet
give_diet refill_bins
clock clk 0 1
h reset
1 get_cola
1 get_diet
cycle
1 reset
cycle
h_get_cola
cycle 4
1 get_cola
h get_diet
cycle 4
1 get_diet
h reset
```

cycle
1 reset

Эта последовательность команд осуществляет следующие действия:

- Определение временных диаграмм, подлежащих отображению (clk, reset, get_cola, give_cola, get_diet, give_diet, refill_bins);
- установка синхронизирующего сигнала (clock signal);
- инициализация входных воздействий для выполнения моделирования (reset);
- установка высокого уровня в течение 4 циклов синхросигнала для get_cola;
- установка низкого уровня в течение 4 циклов синхросигнала для get_cola и, вместе с тем, установка для get_diet высокого уровня для 4 циклов синхросигнала;
- установка для get_diet низкого уровня, после чего устанавливается высокий уровень для reset в течение одного цикла синхросигнала.

Результаты моделирования синтезированного проекта приведены на рис. 7.24.

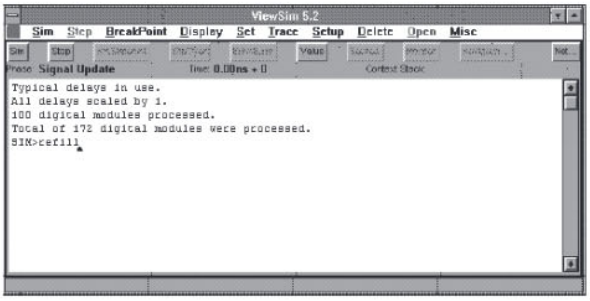


Рис. 7.23. Окно симулятора ViewSim

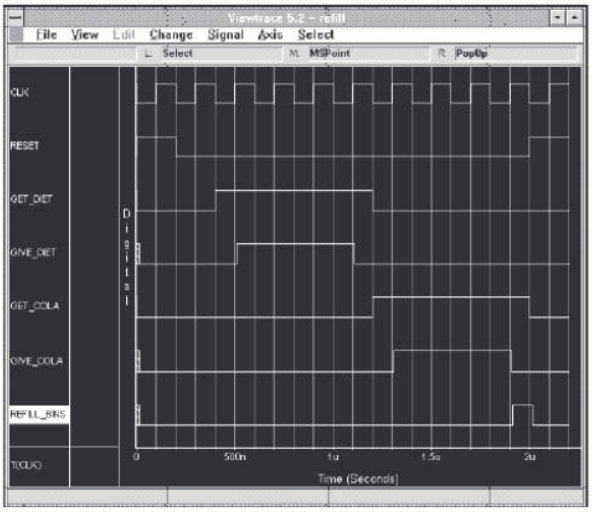


Рис. 7.24. Окно ViewTrace с результатами моделирования для файла refill.vhd

Глава 8. Работа с VHDL в среде системы проектирования Foundation Series 2.1i

8.1. Потоки проектирования в среде Foundation series

Среда проектирования Foundation Series фирмы Xilinx поддерживает два базовых потока проектирования при управлении проектами. Распорядитель проектов (Project Manager) готов работать с проектами на языках описания аппаратуры (HDLs, Text-Based Entry) или на схемотехническом уровне (Schematic, Schematic Entry) — рис. 8.1 [74а—76а].

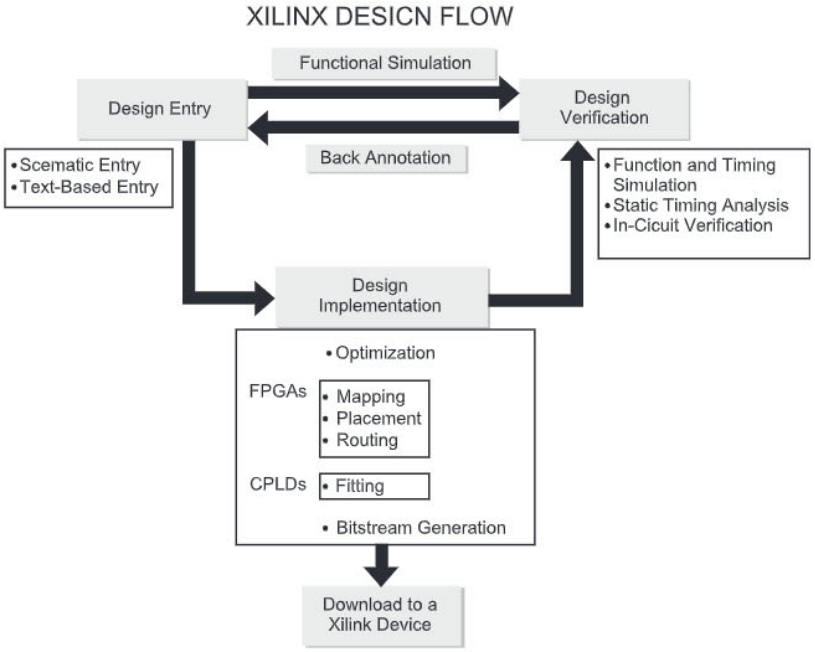


Рис. 8.1. Потоки проектирования в среде Foundation Series

В потоке HDL Flow проект может содержать описания на VHDL,Verilog или описания верхнего уровня проекта на схемотехническом уровне, которые включают наряду со схемными модулями модули, написанные на VHDL или Verilog.

Файлы на языках описания аппаратуры (HDL-files) создаются с помощью следующих встроенных редакторов:

- HDL Editor;
- Finite State Machine (FSM) Editor (работу с этим редактором мы опишем далее самым подробным образом).

Конечно же, HDL-файл может быть создан с помощью любого другого текстового редактора. Необходимо отметить, что в данном потоке проектирования также могут быть использованы модули на еще одном языке описания аппаратуры — XABEL (Xilinx ABEL). Эти модули, наряду с LogiBLOX и CORE Generator (известные

как XNF-файлы), используются для реализации проектов по методу черного ящика (Black Box Instantiation Method). По способу физической реализации проекта (с помощью FPGAs или CPLDs) данный поток проектирования имеет два варианта, приведенных на рис. 8.2 и 8.3. Отсюда видно, что языки описания аппаратуры в среде Foundation Series используются на следующих стадиях проектирования:

1) описание проектов (Design Entry);
2) моделирование для целей верификации проектов (Checkpoint Verification):
   – (Gate-Level Functional Simulation);
   – (Behavioral HDL Simulation);
   – (Post-Place&Route Gate-Level Timing Simulation);
   – (Post-Implementation Gate-Level Timing Simulation).

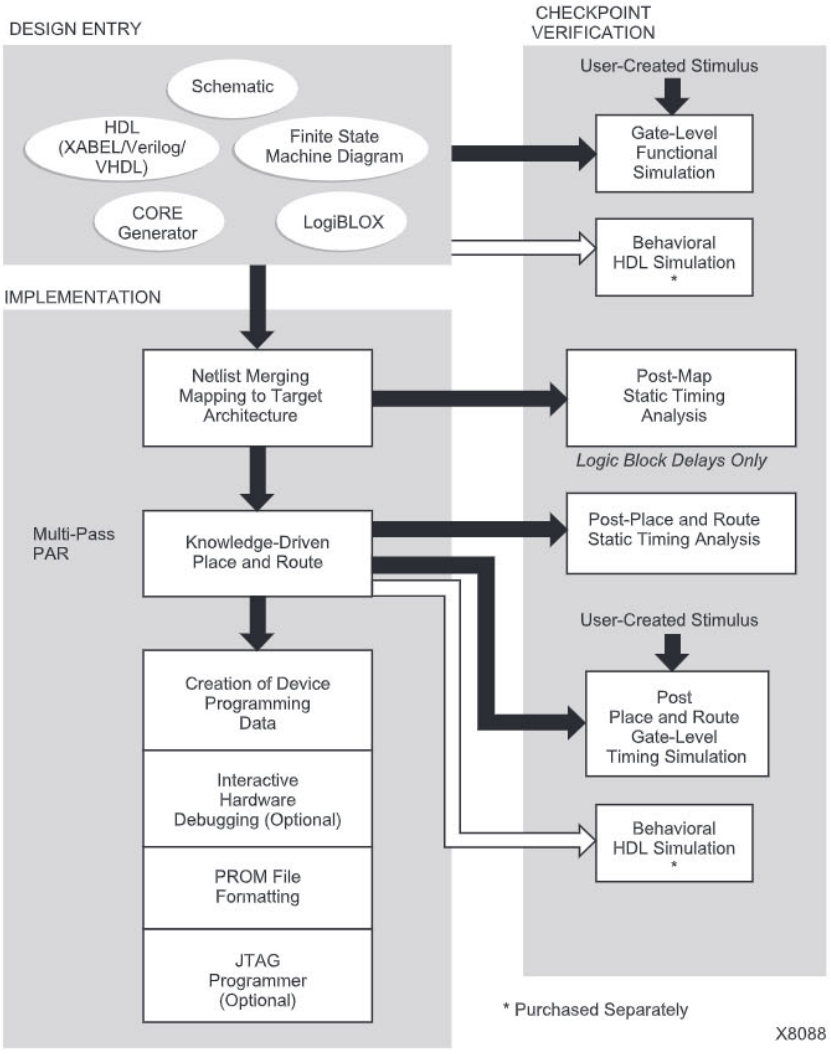


Рис. 8.2. Реализация проекта с помощью FPGAs

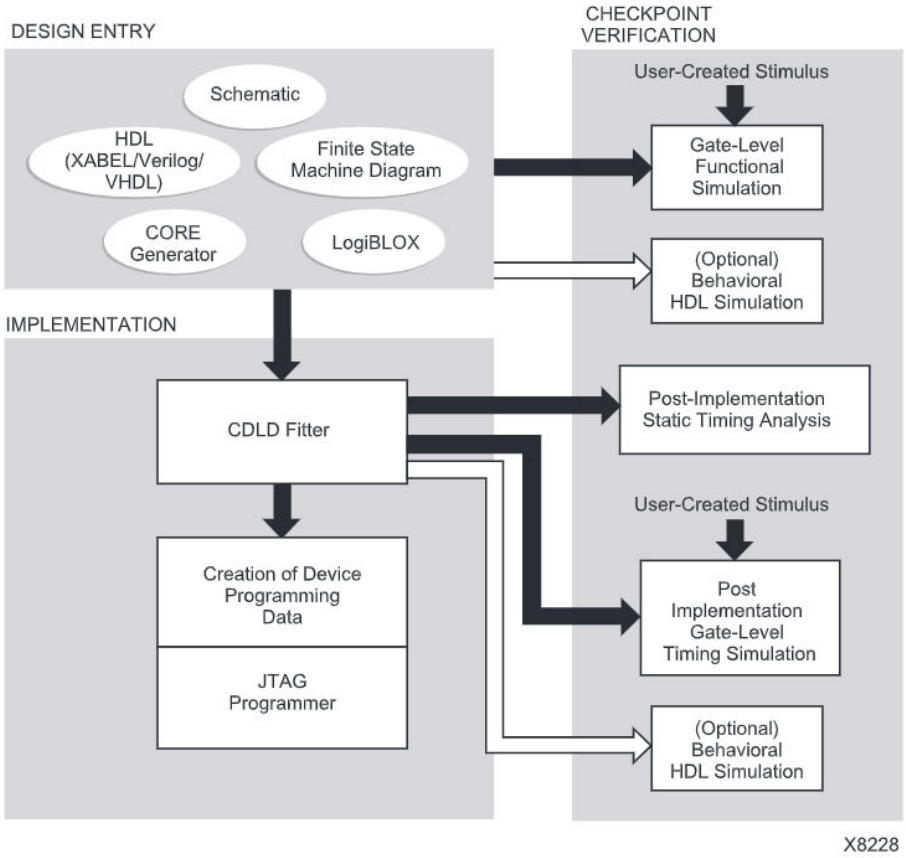


Рис. 8.3. Реализация проекта с помощью CPLDs

8.2. Открытие проекта

Для запуска распорядителя проектов (**Project Manager**) дважды щелкнем по его иконке (**Project Manager Icon**) в программной группе Foundation Series (рис. 8.4).

Рис. 8.4. Иконка распорядителя проектов

В возникающем диалоговом окне **Getting Started Dialog Box** (рис. 8.5) можно открыть уже существующий проект (рис. 8.6) или создать новый (рис. 8.7, *а*).

Для проектирования на схемном уровне (Schematic Flow) необходимо выбрать в последнем окне имя проекта, его папку семейство, тип прибора и его скорость. Для проектирования с помощью языков описания аппаратуры (HDL Flow) достаточно выбрать имя проекта и его папку (рис. 8.7, *б*).

Нажав клавишу **OK,** переходим в окно **Project Manager** с открытым новым проектом (рис. 8.8).

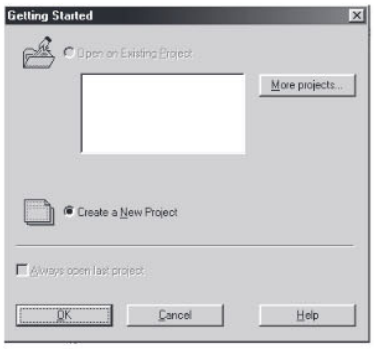


Рис. 8.5. Окно диалога Getting Started Dialog Box

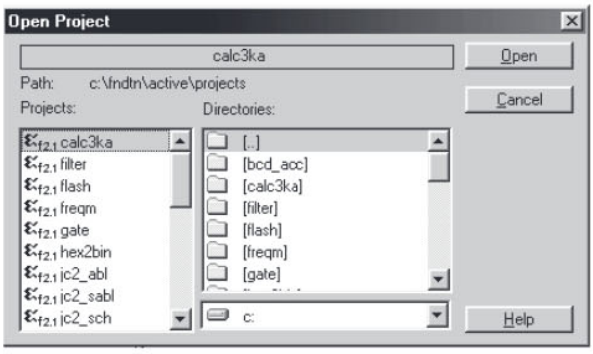


Рис. 8.6. Открытие уже существующего проекта

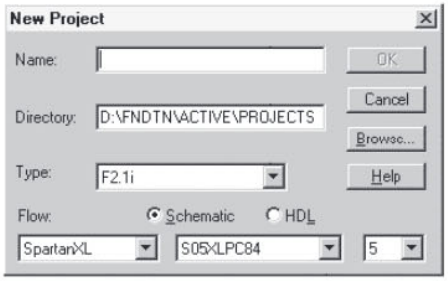


Рис. 8.7,а. Открытие нового проекта на схемном уровне

Рис. 8.7,б. Открытие нового проекта в HDL Flow

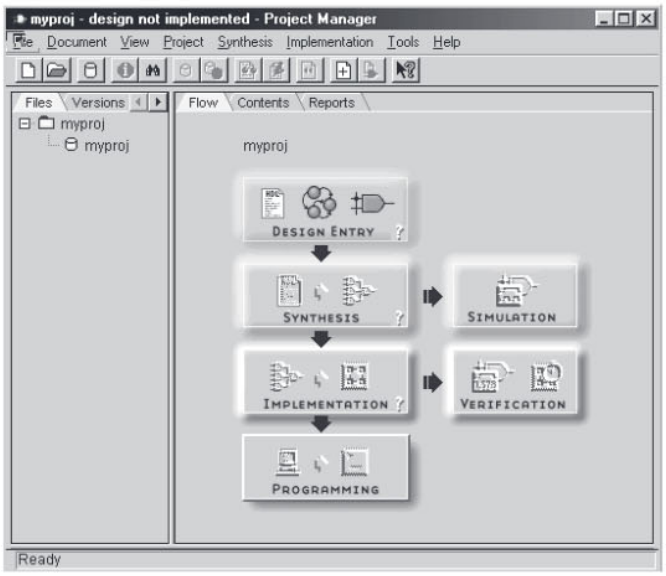


Рис. 8.8. Открытие проекта в окне Project Manager

8.3. Пример проектирования в среде Foundation Series

Предметом проектирования будет являться 4-разрядный счетчик Джонсона (4-bit Johnson Counter) (рис. 8.9) (JCOUNT). Управляющими входами счетчика являются:

- CLK — входной синхроимпульс;
- CE — сигнал разрешения синхроимпульса;
- CLR — сигнал сброса.

Выходы счетчика: Q0—Q3.

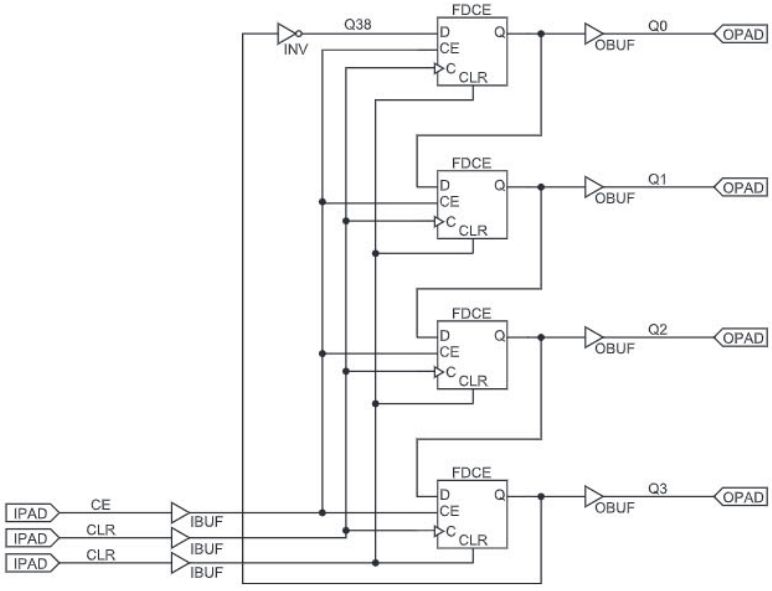


Рис. 8.9. Схема 4-разрядного счетчика Джонсона

Последовательность состояний счетчика Джонсона в нормальном режиме (т. е. без сигнала сброса) такова:

0000
0001
0011
0111
1111
1110
1100
1000
0000

Наш проект для счетчика Джонсона будет называться jct_vhdu. Для открытия проекта вызовем **Project Manager**, как было описано выше, и в диалоговом окне **Getting Started Dialog Box** (рис. 8.10) выберем опции:

– **More Projects**;
– **Open on Existing Project**.

В результате наш проект открывается в окне **Project Manager** (рис. 8.11).

Видно, что к нашему проекту присоединены автоматически три библиотеки:

- jct_vhdu — содержит объекты проекта;
- spartanx — содержит компоненты Xilinx Unified Library для данного семейства приборов;
- simprims — содержит модели приборов фирмы Xilinx для выполнения моделирования.

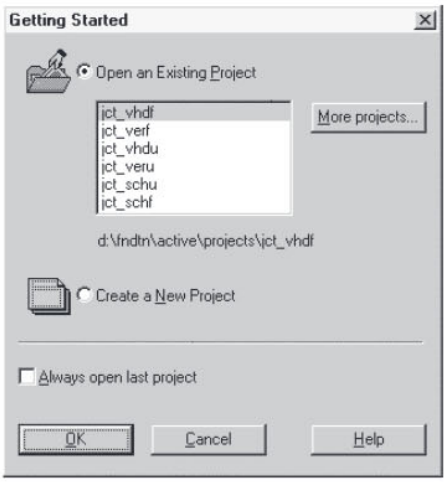


Рис. 8.10. Открытие проекта jct_vhdu

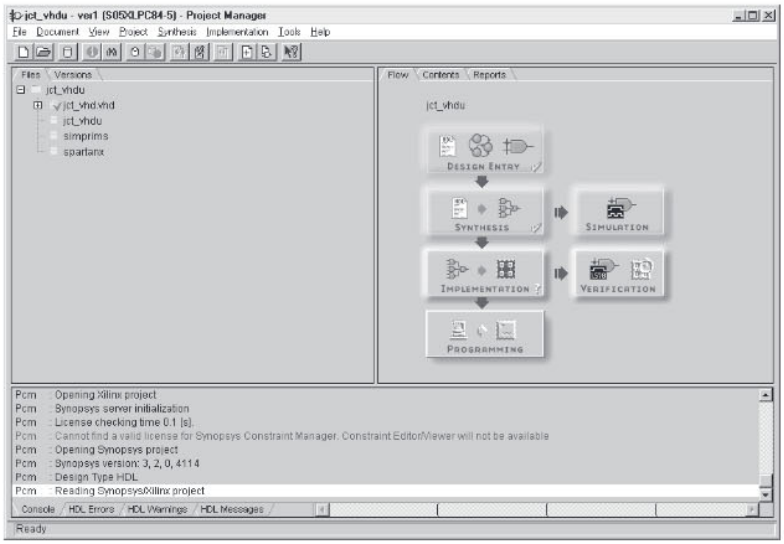


Рис. 8.11. Открытие проекта jct_vhdu в окне Project Manager

8.4. Ввод проекта на языке VHDL (HDL-Based Design Entry)

8.4.1. Добавление файла к проекту

Для добавления к проекту файла (например, jct_vhd.vhd) необходимо выбрать **Document->Add** в окне **Project Manager**. Затем в открывающемся диалоговом окне **Add document** надо ввести имя файла jct_vhd.vhd в поле списка **File name** и щелкнуть по клавише **Open**. Для открытия данного файла необходимо на закладке **Files** окна **Project Manager** дважды щелкнуть по имени файла.

8.4.2. Корректировка синтаксических ошибок

Для проверки синтаксиса VHDL-файла необходимо выбрать **Synthesis -> Check Syntax** в окне **HDL Editor**. В качестве учебного примера, файл jct_vhd.vhd имеет определенную синтаксическую ошибку: не продекларирован порт CLK (поэтому рядом с именем файла в окне **Project Manager** находится красный вопросительный знак — символ неверного синтаксиса). Исправим эту ошибку, введя правильное описание интерфейса нашего счетчика:

```
port (CLK : std_logic;
      CE : std_logic;
      CLR: std_logic;
      Q : std_logic_vector (3 downto 0)),
```

после чего вновь проверим правильность синтаксиса и убедимся, что ошибок больше нет.

8.4.3. Использование ассистента — знатока языка (Language Assistant)

В окне **HDL Editor** нам всегда доступен ассистент—знаток языка **Language Assistant**. Он готов обеспечить нас шаблонами (templates) — фрагментами VHDL-кода часто используемых логических компонент, таких как счетчики, триггеры, мультиплексоры, буфера и т. д. Мы можем создавать собственные фрагменты, которыми собираемся часто пользоваться. Для запуска **Language Assistant** необходимо выбрать **Tools -> Language Assistant** в окне **HDL Editor**.

Как видно из рис. 8.12, *а*, **Language Assistant** содержит три секции:

- **Language Templates** (конструкции языка);
- **Synthesis Templates** (описания устройств, предназначенные для синтеза);
- **User Templates** (конструкции языка, созданные пользователем).

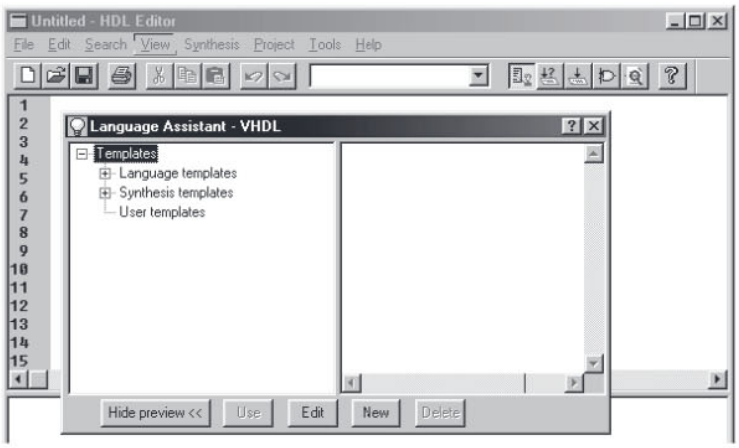


Рис. 8.12,а. Использование HDL Language Assistant

Выберем нужную нам секцию щелчком по соответствующему крестику, а затем в открывшемся списке фрагментов выберем нужный и увидим соответствующий код на правой стороне окна (рис. 8.12, *б*).

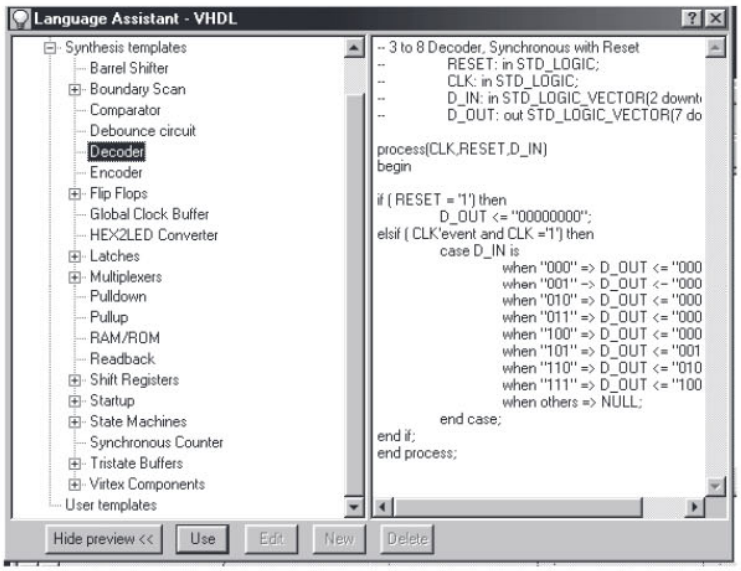


Рис. 8.12,б. Использование HDL Language Assistant (просмотр предлагаемого шаблона кода)

8.4.4. Синтез проекта

Когда все описание проекта введено и проверено на наличие синтаксических ошибок, мы можем перейти к следующей стадии проектирования — логическому синтезу (Logic Synthesis). На этой стадии наш VHDL-код компилируется в список соединений вентилей (netlist of gates) в формате XNF или EDIF. Этот список соединений является входной информацией для инструмента синтеза — Xilinx Implementation Tools. Вкратце расскажем о необходимых действиях в процессе синтеза.

1. Для выбора глобальных опций синтеза выберем **Synthesis -> Options**. Здесь можно установить частоту работы синтезируемого устройства (Default Frequency), а также проверить временные ограничения в окне **Export Timing Constraints**. Для их принятия щелкнем по клавише **OK**.
2. Выберем файл jct_vhd.vhd и щелкнем по клавише **Synthesis** на закладке **Flow** в окне **Project Manager**.

3. В появляющемся окне **Synthesis/Implementation** (рис. 8.13), введем информацию о приборе, которым хотим воспользоваться для физической реализации проекта (**Target Device**) (рис. 8.13) и нажмем кнопку **Run**.

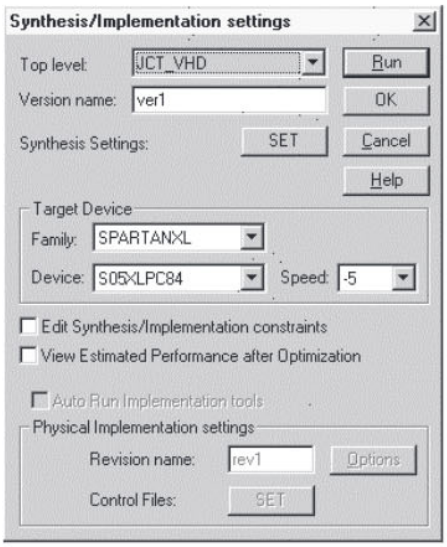


Рис. 8.13. Окно Synthesis/Implementation

8.4.5. Функциональное моделирование

Перед физической реализацией проекта (Design Implementation) у нас есть возможность верифицировать правильность созданной логики с помощью функционального моделирования. В среде проектирования Foundation Series нам доступен логический симулятор (Logic Simulator), который функционирует на вентильном уровне (Gate-Level Simulator). В дальнейшем мы поговорим и о моделировании временных задержек (Timing Simulation), которое выполняется после того, как проект будет реализован, т. е. после того, как будет выполнена операция размещения/трассировки (Place&Route) с помощью инструмента Xilinx Implementation Tools. Для запуска симулятора необходимо нажать клавишу **Simulation** на закладке **Flow** в окне **Project Manager**.

Процесс функционального моделирования включает три следующих этапа.

1. Выбор входных и выходных сигналов для просмотра в симуляторе (Adding Signals).
2. Назначение входных воздействий (Adding Stimulus).
3. Собственно запуск моделирования и просмотр временных диаграмм (Running Simulation&View Output Waveforms).

Для выбора сигналов, подлежащих просмотру, необходимо воспользоваться окном **Component Selection**. Это окно можно вызвать, находясь в окне логического симулятора (рис. 8.14, *а*), двумя путями:

- выбором **Signal -> Add Signals** в главном меню окна симулятора;
- щелчком по иконке  в линейке инструментов окна симулятора.

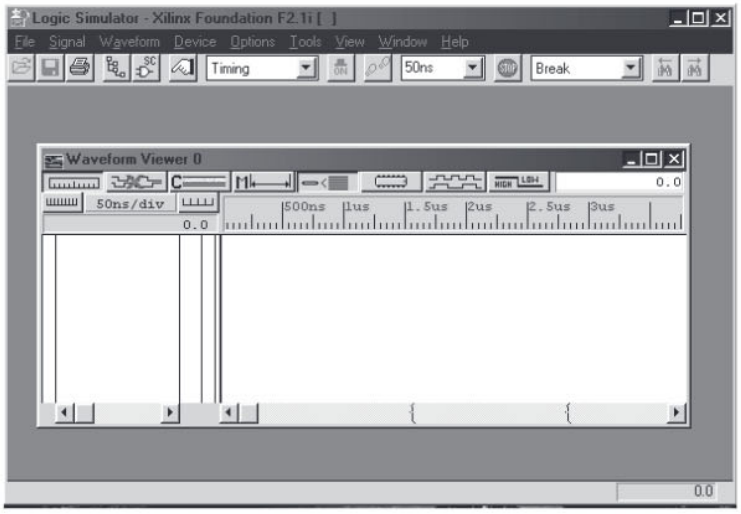


Рис. 8.14,а. Окно логического симулятора

Окно **Component Selection** (рис. 8.14, *б*) имеет три панели:

- **Signals Selection**;
- **Chip Selection**;
- **Scan Hierarchy**.

Панель **Signals Selection** отображает список всех доступных для просмотра сигналов на данном уровне иерархии. Средняя панель **Chip Selection** отображает список

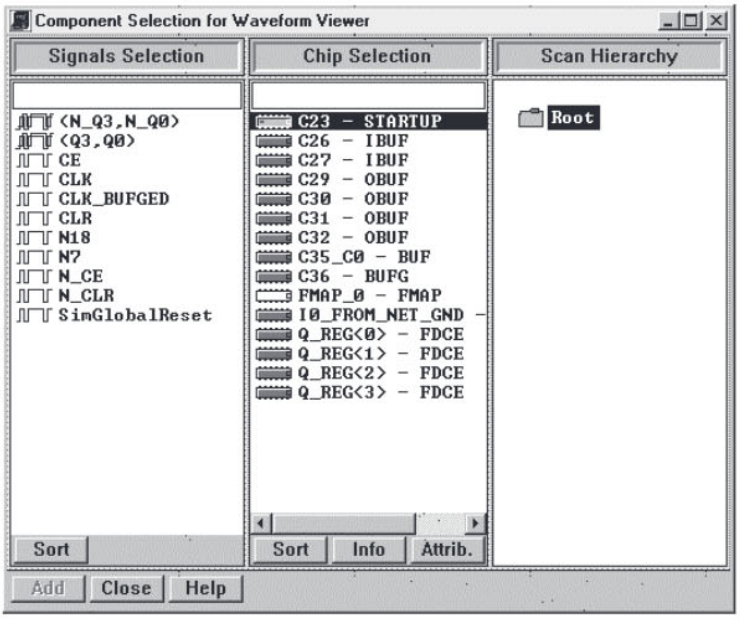


Рис. 8.14,б. Окно выбора сигналов для просмотра временных диаграмм

компонентов на данном уровне иерархии. На панели **Signals Selection** двойным щелчком по имени сигнала можно добавить его к окну **Waveform Viewer** (рис. 8.14, *а*). Для этой же цели достаточно одного щелчка по имени сигнала и щелчка по клавише **Add**. Любым из этих двух методов добавим к окну **Waveform Viewer** следующие сигналы и шину:

- CLK;
- CE;
- CLR;
- <Q3,Q0>.

Для удаления уже существующего сигнала в окне **Waveform Viewer** необходимо сделать щелчок правой клавишей мыши и выбрать **Delete Signals -> Selected** в возникающем меню. Для назначения внешних воздействий необходимо воспользоваться окном **Stimulator Selection** (рис. 8.15), которое вызывается щелчком по иконке  в линейке инструментов или выбором **Signal -> Add Stimulators...** В качестве источника входного синхросигнала для проекта JCOUNT воспользуемся встроенным в среду проектирования Foundation Series 16-разрядным бинарным счетчиком (**Bc** на рис. 8.15): подсоединим младший разряд этого счетчика (**B0**) ко входу CLK нашего проекта.

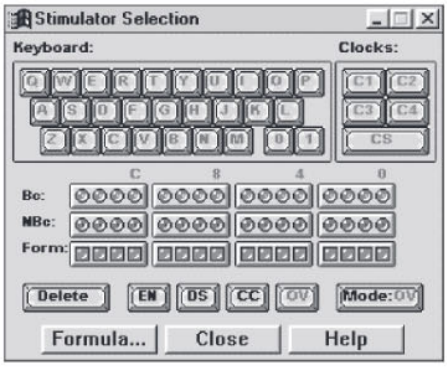


Рис. 8.15. Назначение внешних воздействий с помощью окна Stimulator Selection

Для этого выполним следующие действия.

1. В окне **Waveform Viewer** (рис. 8.14, *а*, 8.17) выберем щелчком сигнал CLK.
2. В окне **Stimulator Selection** щелкнем по **B0** (крайне правый желтый кружок).

Тотчас же можно видеть (рис. 8.17), что источник внешнего воздействия **B0** уже соединен со входом CLK.

3. Выберем **Options -> Preferences** в окне **Simulator**. При этом возникает окно **Preferences** (рис. 8.16). На закладке **Simulation** установим частоту 10 MHz для **B0**.
4. Щелчком по клавише **OK** закроем окно **Preferences**.

У нас есть возможность использовать клавиши клавиатуры в качестве внешних воздействий для входов нашего проекта (рис. 8.15). После назначения клавиши источником внешнего воздействия (стимулом) величина сигнала изменяется между 0 и 1, когда бы мы ни нажали соответствующую клавишу на PC-клавиатуре. Создадим, например, источник внешнего воздействия, связанный с клавишей **R** на клавиатуре, для входа CLR проекта JCOUNT.

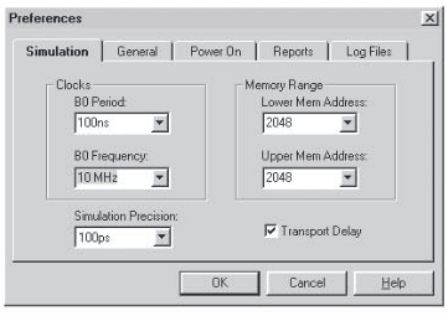


Рис. 8.16. Окно предпочтений (предустановок) симулятора

1. Щелкнем и перетянем клавишу **R** на клавиатуре в окне **Stimulator Selecti-**

on к сигналу CLR в окне **Waveform Viewer** (рис. 8.17) и увидим в этом окне букву **r**, свидетельствующую о том, что ко входу CLR подсоединен источник внешнего воздействия, связанный с клавиатурой.

2. Можно нажать клавишу **R** на PC-клавиатуре несколько раз, чтобы убедиться, что состояние данного стимула меняется в окне **Waveform Viewer**.

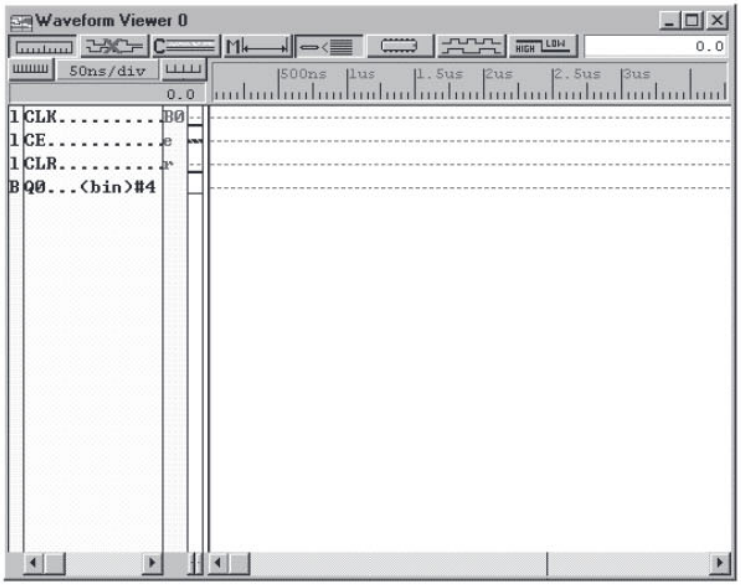


Рис. 8.17. Окно Waveform Viewer для просмотра временных диаграмм

3. Подобным же образом создадим еще один источник внешнего воздействия, связанный с клавиатурой (клавиша **E**), для входа CE проекта JCOUNT.
4. Закроем окно **Stimulator Selection** нажатием клавиши **Close**.

Сейчас мы видим в окне **Waveform Viewer** три стимула, присоединенных ко **всем** входам проекта JCOUNT, а также выходы проекта (шина Q). Теперь можно начать процесс моделирования. Для этого выполним следующие действия.

1. Установим в окне **Waveform Viewer** временной шаг моделирования в 100 ns (рис. 8.18).
2. Нажмем клавишу **R** на PC-клавиатуре до получения низкого уровня на входе CLR.
3. Нажмем клавишу **E** на PC-клавиатуре до получения высокого уровня на входе CE.
4. Щелкнем по кнопке **Step**  три раза для продвижения процесса моделирования в соответствии со синхросигналом **B0** установленной ранее частоты.
5. Нажмем клавишу **E** на PC-клавиатуре для моделирования сигнала, разрешающего поступление синхросигнала (clock enable signal).
6. Щелкнем по кнопке **Step** дважды для продвижения процесса моделирования.

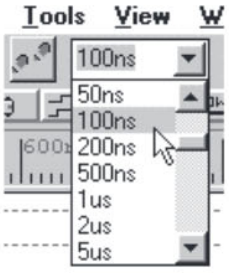


Рис. 8.18. Установка шага симулятора

7. Нажмем клавишу **E** на PC-клавиатуре.
8. Щелкнем по кнопке **Step** трижды для продвижения процесса моделирования.
9. Нажмем клавишу **R** на PC-клавиатуре до получения высокого уровня на входе CLR.
10. Вновь щелкнем по кнопке **Step** один раз.
11. Нажмем клавишу **R** на PC-клавиатуре до получения низкого уровня на входе CLR.
12. И в заключение щелкнем еще два раза по кнопке **Step**.

После выполнения всех вышеописанных действий получим картину, приведенную на рис. 8.19.

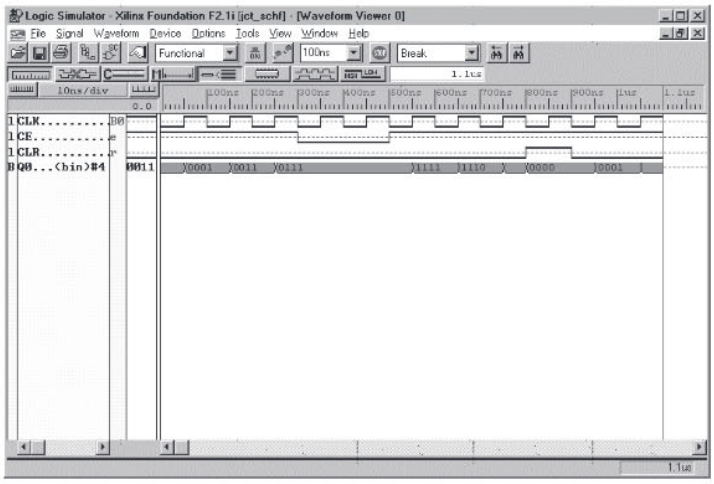


Рис. 8.19. Результаты функционального моделирования

Для того чтобы очистить окно временных диаграмм и вновь запустить процесс моделирования, необходимо выбрать **Waveform -> Delete -> All Waveforms with Power on** в окне симулятора. После чего у нас появится возможность выполнить моделирование и вновь поэкспериментировать с функционированием входов CLR и CE. Закрыть окно **Logic Simulator** можно с помощью выбора **File -> Exit**.

8.4.6. Физическая реализация проекта

Для запуска процесса физической реализации проекта нажмем клавишу **Implementation** на закладке **Flow** окна **Project Manager** (рис. 8.20).

Рис. 8.20. Клавиша Implementation

При этом открывается диалоговое окно **Synthesis/Implementation** (рис. 8.21). В данном окне нажмем клавишу **Options**, для того чтобы специфицировать проект в смысле оптимизации, размещения, трассировки и т. д.

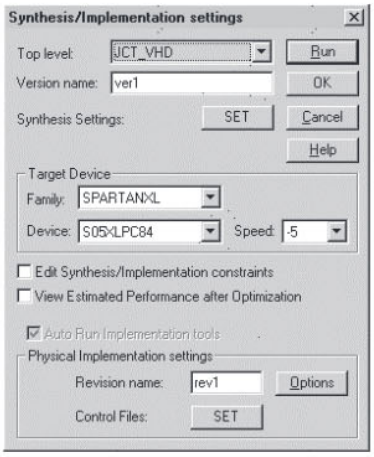


Рис. 8.21. Диалоговое окно Synthesis/Implementation

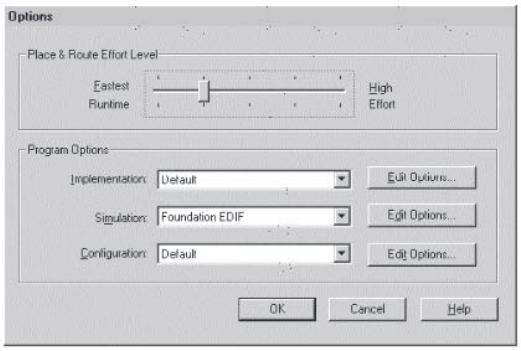


Рис. 8.22. Диалоговое окно Options

Нажатие клавиши **Options** приводит к появлению диалогового окна **Options** (рис. 8.22). После редактирования этих опций диалог заканчивается нажатием кнопки **Close**.

Сейчас нажмем клавишу **Run** в окне **Synthesis/Implementation** (рис. 8.21). При этом появляется окно **Flow Engine** (рис. 8.23), которое отображает процесс прохождения нашего проекта через стадии физической реализации. Когда процесс физической реализации завершается, окно **Flow Engine** автоматически закрывается и вновь становится видимым окно **Project Manager**. Необходимо лишь закрыть диалоговое окно **Flow Engine Completed Successfully** нажатием клавиши **OK**. Заметим, что статус физической реализации отображается на консоли в нижней части окна **Project Manager**. При щелчке по закладке **Versions** на левой панели окна **Project Manager** можно просмотреть структуру (иерархию) реализации нашего проекта (рис. 8.24).

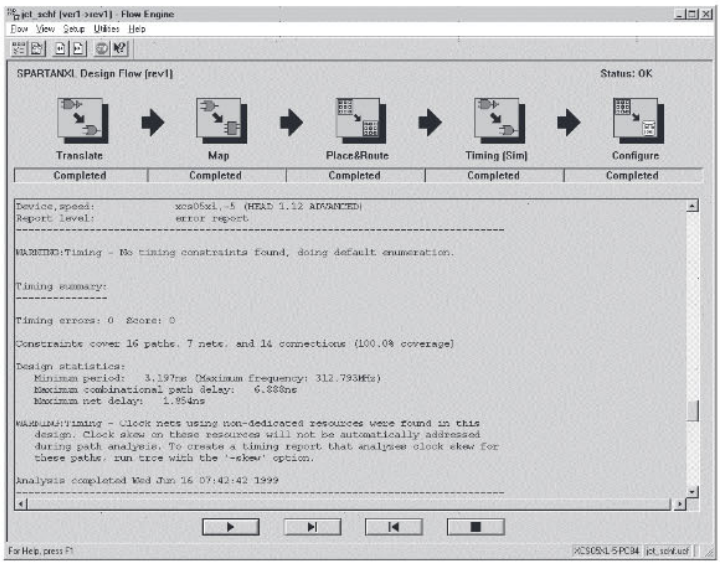


Рис. 8.23. Окно механизма потока проектирования (Flow Engine)

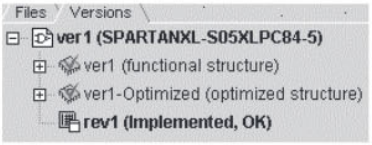


Рис. 8.24. Закладка Versions

8.4.7. Моделирование задержек (Timing Simulation)

Моделирование задержек использует информацию о задержках элементов и соединений, полученную при выполнении процесса трассировки проекта. Только с помощью моделирования задержек можно более точно воспроизвести поведение проекта в наихудшем случае (**worst-case conditions**). Для запуска процесса моделирования задержек щелкнем по иконке **Timing Simulation** на клавише **Verification** на закладке **Flow** окна **Project Manager**.

Для выполнения процесса моделирования задержек мы будем использовать уже существующий скрипт-файл, который служит источником внешних воздействий для выполнения моделирования. Для запуска редактора скриптов **Script Editor** выберем **Tools -> Script Editor** в окне симулятора (рис. 8.25). Возникающее при этом диалоговое окно позволяет выбрать нужный скрипт (script file): **Open: Existing Script File** (мы выбираем предварительно записанный файл **jcount.cmd**), а затем оно закрывается по нажатии клавиши **OK**. Запустить моделирование задержек можно непосредственно из **Script Editor** выбором **Execute -> Go**. Результаты моделирования можно просмотреть в окне **Waveform View** (рис. 8.25).

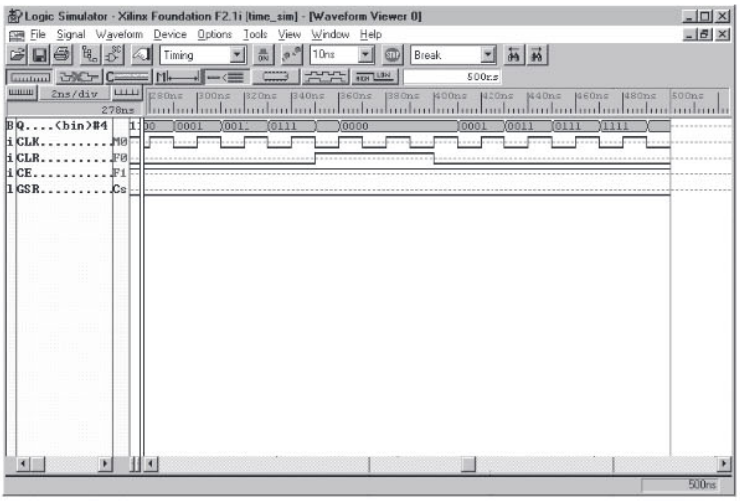


Рис. 8.25. Результаты моделирования задержек (Timing Simulation)

Детальное изучение временных диаграмм облегчается использованием иконки **Expand** 

.

8.5. Графический редактор для создания моделей в виде цифровых автоматов (Finite State Machine Editor) среды проектирования Foundation Series

Редакторы цифровых автоматов позволяют легко и просто осуществить графический ввод проектной информации. Так как модели в виде цифровых автоматов (State Machines) в общем виде технологически независимы, графические редакторы этого типа очень популярны у разработчиков аппаратуры. В частности, такие редакторы могут осуществить графический ввод моделей на языках описания аппаратуры (VHDL, Verilog HDL, ABEL...). В данном разделе мы рассмотрим применение редактора цифровых автоматов для ввода диаграмм состояний (State Machine Diagrams) и логического синтеза проекта.

8.5.1. Сравнение методов описания цифровых автоматов (State Machine Description Methods)

Остановимся на простом цифровом автомате, который используется для управления работой светофора (Traffic Lights Controller).

Вербальное (словесное) описание (Verbal Description)_выглядит следующим образом.

Когда горит красный свет (состояние RED, в котором соответствующие сигналы имеют следующие значения:

- ***LIGHT_GREEN <= '0',***
- ***LIGHT_YELLOW <= '0',***
- ***LIGHT_RED <= '1')***

и поступает сигнал (GO_GREEN <= '1'), что должен зажечься зеленый свет, то необходимо зажечь зеленый свет (перейти в состояние GREEN и задать соответствующим сигналам следующие значения:

- ***LIGHT_GREEN <= '1',***
- ***LIGHT_YELLOW <= '0',***
- ***LIGHT_RED <= '0').***

При поступлении сигнала о необходимости зажечь желтый свет (GO_YELLOW <= '1') он зажигается (переход в состояние YELLOW и установка следующих значений для соответствующих сигналов:

- ***LIGHT_GREEN <= '0',***
- ***LIGHT_YELLOW <= '1',***
- ***LIGHT_RED <= '0').***

И, наконец, при поступлении сигнала LIGHT_RED <= '1' вновь зажигается красный свет, после чего весь вышеописанный цикл может повториться.

Графическим методом описания цифровых автоматов является диаграмма состояний (State Machine Diagram). Диаграмма состояний для нашего контроллера приведена на рис. 8.26.

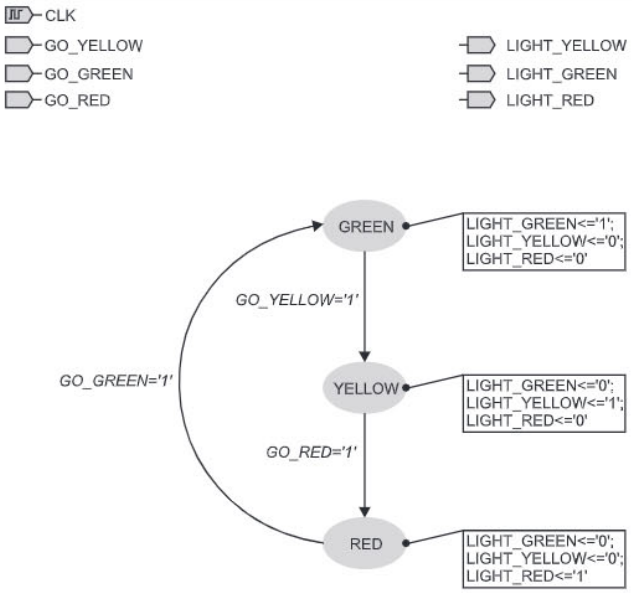


Рис. 8.26. Диаграмма состояний для контроллера светофора

Модель в виде цифрового автомата может быть реализована в виде программы на одном из языков описания аппаратуры (HDL Program, HDL Description) — VHDL, Verilog HDL, ABEL, CUPL и т. д. В свою очередь, такая программа может быть физически реализована в виде списка соединений (Netlist) с помощью программного инструмента логического синтеза (Logic Synthesis Software). Вот, например, описание на VHDL (VHDL Description) для контроллера управления светофором (рис. 8.27):

```
entity lights is
  port (CLK: in STD_LOGIC;
        GO GREEN: in STD  LOGIC;
        GO_RED: in STD_LOGIC;
        GO_YELLOW: in STD_LOGIC;
        LIGHT GREEN: out  STD LOGIC;
        LIGHT_RED:' out STD_LOGIC;
        LIGHT_YELLOW: out STD_LOGIC);
end;
architecture lights_arch of lights is
type LIGHTS type is  (GREEN, RED, YELLOW);
signal LIGHTS: LIGHTS_type;
begin
process (CLK)
begin
if CLK'event and CLK - '1' then
    case LIGHTS is
       when GREEN =>
           if GO_YELLOW-'1' then
               LIGHTS <=  YELLOW;
```

```
            end if;
        when RED =>
            if GO_GREEN='1' then
                LIGHTS <=  GREEN;
            end if;
        when YELLOW =>
            if GO RED-'1'  then
                LIGHTS <=  RED;
            end if;
        when others =>
            null;
    end case;
end if;
end process;
LIGHT GREEN <= '1' when  (LIGHTS = GREEN) else
               '0' when  (LIGHTS = RED) else
               '0' when  (LIGHTS = YELLOW)  else
               '0';
LIGHT_YELLOW <=  '0' when  (LIGHTS = GREEN)  else
                 '0' when  (LIGHTS = RED) else
                 '1' when  (LIGHTS = YELLOW)  else
                 '1';
LIGHT RED <=  '0' when  (LIGHTS = GREEN) else
              '1' when  (LIGHTS = RED) else
              '0' when  (LIGHTS = YELLOW)  else
              '0';
end lights arch;
```

Рис. 8.27. VHDL-описание контроллера

Главные достоинства и недостатки рассмотренных выше трех описаний можно свести в следующую таблицу на рис. 8.28.

Verbal Description	State Diagram	HDL Description
компактное	очень компактное	длинное
легко читаемо для людей	очень легко читаемо для людей	трудно читаемо для людей
трудно реализуемо	легко реализуемо	легко реализуемо
не требует дополнительной документации	не требует дополнительной документации	требует дополнительной документации

Рис. 8.28. Достоинства и недостатки различных методов описания

8.5.2. Создание диаграммы состояний

Рассмотрим проектирование цифрового автомата, способного играть в игру, называемую Blackjack. Наш проект будет синтезирован в виде прибора серии XC9500 фирмы Xilinx.

Вербальное описание игры Blackjack выглядит следующим образом. Целью игры является выбор нескольких карт, так чтобы суммарное число очков было бы

максимально близким к 21. Каждая карта имеет десятичное значение в интервале между 2 и 11, где 11 называется главным козырем (или козырным тузом) (ACE) и может при желании считаться за 1 очко. Игрок, набравший максимальное число очков, которое, однако, меньше 21, становится победителем.

Просьба получить следующую карту индицируется присвоением выходному сигналу SAY_CARD значения логической единицы. Этот сигнал генерируется всякий раз, когда общая сумма очков всех карт меньше 17.

Появление новой карты индицируется установкой входного сигнала NEW_CARD в состояние логической единицы.

Цифровой автомат устанавливает выходной сигнал SAY_BUST в состояние логической единицы, когда общая сумма очков превысит 21. В случае же, если общая сумма очков находится в интервале от 17 до 21, он устанавливает в состояние логической единицы выходной сигнал SAY_HOLD.

Общее число очков должно индицироваться с помощью выходного сигнала TOTAL. После достижения цифровым автоматом состояний SAY_HOLD или SAY_BUST он должен оставаться в одном из этих последних состояний до поступления сигнала сброса NEW_GAME на входе.

Сигнал сброса NEW_GAME переводит цифровой автомат в начальное состояние.

Реализация описанного выше цифрового автомата программной системой Foundation Series включает следующие этапы:

- начертание диаграммы состояний цифрового автомата (State Machine Diagram) с помощью графического редактора;
- генерация VHDL-кода на основе этой диаграммы;
- синтез проекта;
- физическая реализация проекта (Physical Implementation).

Создание нового проекта осуществляется следующим образом:

- запустить менеджер проектов (**Project Manager**);
- выбрать опцию **Create New Project** в диалоговом окне **Getting Started**;
- ввести имя проекта BLKJACK в поле **Name** открывшегося диалогового окна **New Project** (рис. 8.29);
- выбрать тип проекта **F2.1i** (Foundation Series) и опцию **HDL Flow** (это означает, что нет необходимости выбора специфицировать определенный прибор в данный момент);
- щелкнуть по кнопке **OK** завершая создание нового проекта (рис.).

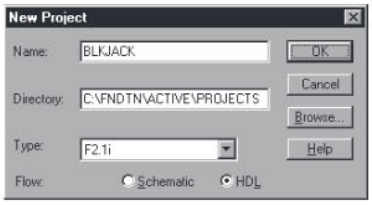


Рис. 8.29. Диалоговое окно Project Manager

При этом текущее диалоговое окно исчезает, и перед нами окно **Project Manager** (рис. 8.30).

Для запуска графического редактора необходимо щелкнуть по иконке  на панели **Flow** окна **Project Manager** (рис. 8.30). В появившемся окне **State Editor**

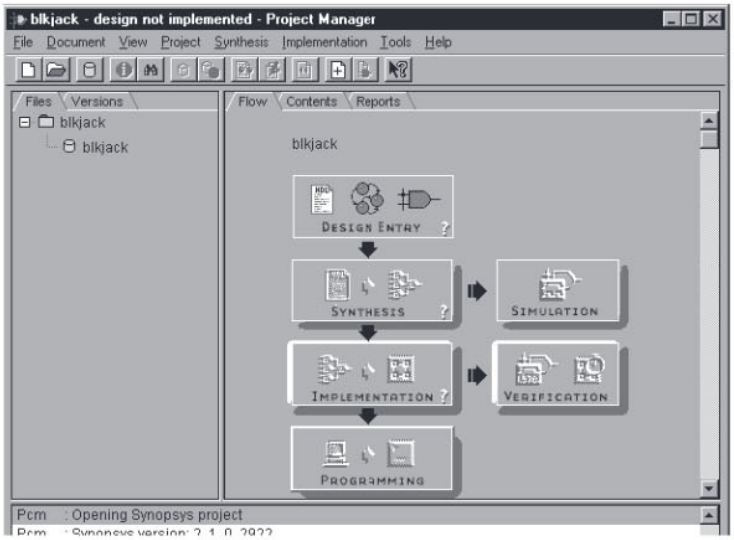


Рис. 8.30. Окно Project Manager

(рис. 8.31) выбирается опция **Use HDL Design Wizard** и нажимается кнопка **OK**, что приводит к запуску мастера проектов на языках описания аппаратуры **HDL Design Wizard** (рис. 8.32).

Мастер проектов HDL Design Wizard позволяет вводить имена портов и файлов, а также выбрать язык описания аппаратуры для текущего проекта в окне **Design Wizard — Language**, появляющемся при нажатии клавиши **Next** в текущем окне (рис. 8.33).

Выберем в качестве инструмента описания проекта язык VHDL и нажмем на клавишу Next, тем самым открывая следующее окно — **Design Wizard — Name** (рис. 8.34). В качестве имени файла проекта выберем BLKJACK и щелкнем по клавише Next. При этом появляется следующее окно — **Design Wizard — Ports** для ввода портов (входных и выходных сигналов) (рис. 8.35).

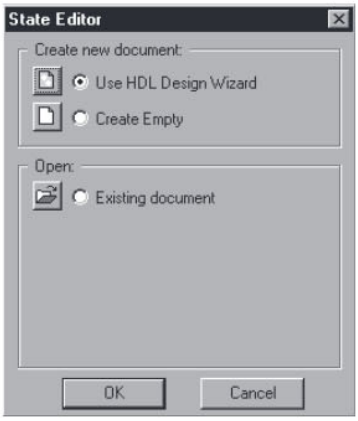


Рис. 8.31. Окно графического редактора цифровых автоматов

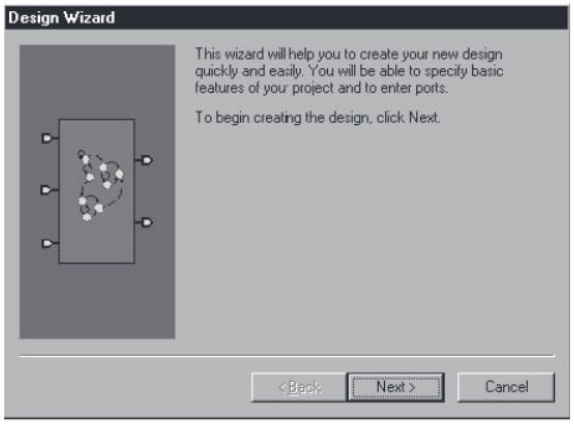


Рис. 8.32. Запуск мастера проектов HDL Design Wizard

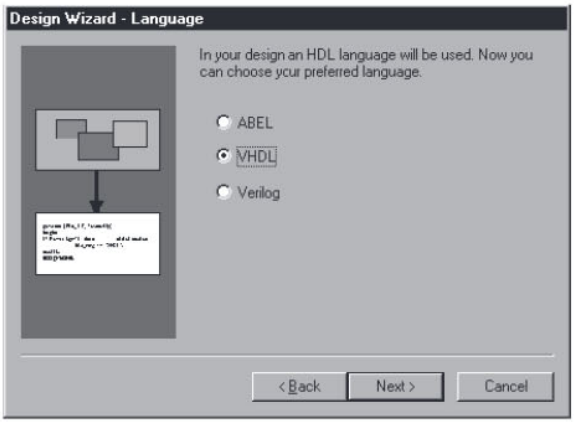


Рис. 8.33. Окно Design Wizard — Language

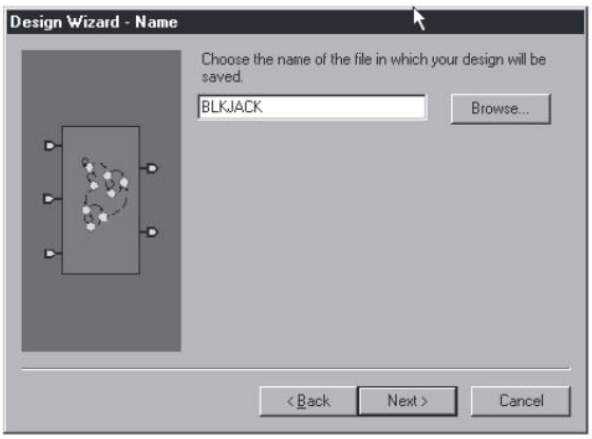


Рис. 8.34. Окно Design Wizard — Name

По умолчанию расширение файла, содержащего описание диаграммы состояний, — ASF. Таким образом, наш новый файл имеет полное имя BLAKJACK.ASF.

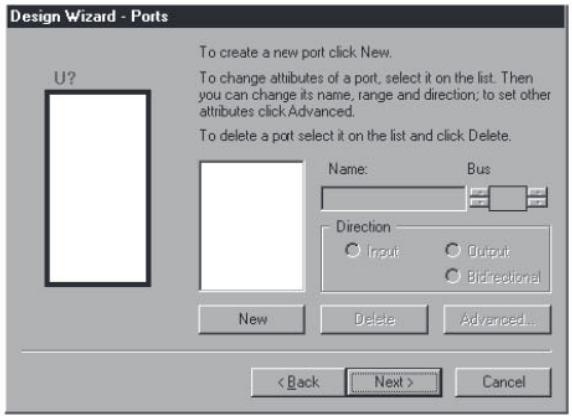


Рис. 8.35. Окно Design Wizard — Ports

Для добавления нового порта к интерфейсу проекта необходимо выполнить следующие операции в окне **Design Wizard — Ports** (рис. 8.36):

- щелкнуть по клавише **New**;
- ввести имя порта в поле **Name**;
- выбрать направление функционирования порта (Input, Output, Bi-directional) с помощью соответствующей опции в поле **Direction**.

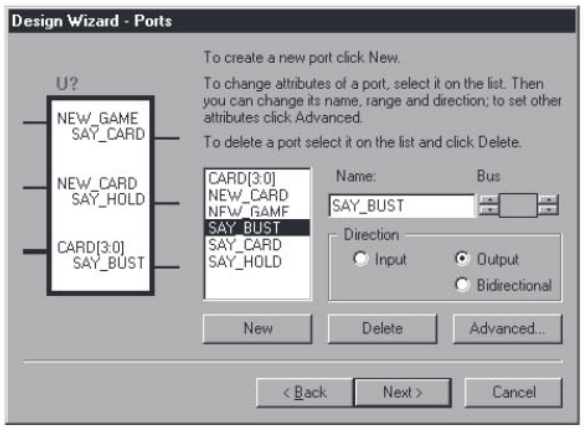


Рис. 8.36. Окно Design Wizard — Ports

Все уже присвоенные портам имена отображаются в окне над кнопкой **New**. Слева от этого окна располагается символическое изображение интерфейса проекта.

Используя вышеописанную процедуру, введем следующие порты:

- входы: NEW_GAME, NEW_CARD, CARD[3:0] (для ввода размерности шины (bus range [3:0]) необходимо щелкнуть по малым кнопкам в поле Bus справа от поля Name);
- выходы: SAY_CARD, SAY_HOLD, SAY_BUST.

Щелчок по клавише **Advanced** приводит к появлению окна **Advanced Port Settings**, которое позволяет задать тип порта (рис. 8.37). Это окно позволяет определить входные сигналы как последовательности импульсов (Clock), а выходные сигналы, как Combinatorial или Registered. Порты типа Registered сохраняют полученное значение в дополнительном регистре. Порты же типа Combinatorial, свое состояние в любое время в соответствии с изменениями входных сигналов.

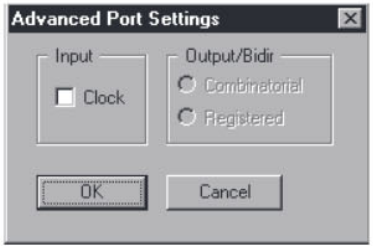


Рис. 8.37. Окно Advanced Port Settings

Графический редактор поддерживает создание лишь синхронных цифровых автоматов (Synchronous State Machines): для таких автоматов все переходы из одного состояния в другое выполняются только по поступлению синхроимпульсов, и проектируемая система не может функционировать без них.

После введения всех портов интерфейса проекта нажимается клавиша **Next**. В случае, если ни один из входных портов не определен как Clock, возникает диалоговое окно с напоминанием о необходимости определить хотя бы один входной сигнал как Clock (рис. 8.38).

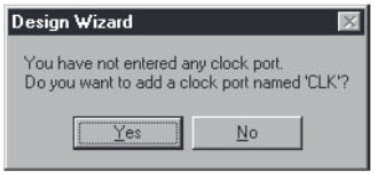


Рис. 8.38. Предупреждение об отсутствии источников синхроимпульсов

В ответ на данное предупреждение необходимо нажать клавишу **Yes**, что приводит к автоматическому добавлению входного сигнала типа Clock к интерфейсу проекта.

После завершения описанной процедуры ввода портов появляется окно **Design Wizard — Machines** (рис. 8.39). В этом окне выбираем опцию **One**, так как проектируем лишь один цифровой автомат. Если же мы выбираем опцию нескольких цифровых автоматов, то они будут преобразованы системой в несколько параллельных VHDL-процессов.

Новая диаграмма, созданная нами с помощью **Design Wizard**, приведена на рис. 8.40.

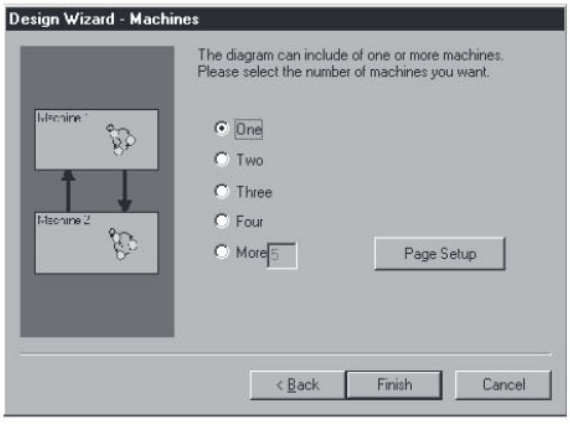


Рис. 8.39. Окно Design Wizard — Machines

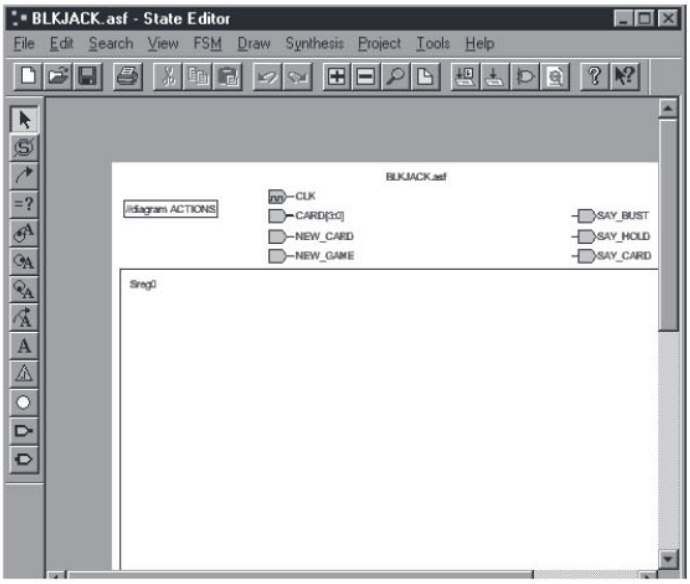


Рис. 8.40. Новая диаграмма состояний в окне графического редактора

Дополнительные порты ввода-вывода (I/O Extra Ports) могут быть присоединены к диаграмме состояний в любой момент времени щелчком по клавише  , находящейся в самом низу вертикальной линейки инструментов (рис. 8.40). Вот, например, сейчас необходимо создать выходной порт для индикации общей суммы очков, полученных в ходе игры. Для этого выполним следующие действия.

1. Щелкнем по клавише  (Output Port), а затем — повыше символа порта SAY_BUST (рис. 8.40). Это приведет к размещению нового порта Port1 над портом SAY_BUST.
2. Щелкнем по символу порта, уже размещенного на диаграмме состояний, правой клавишей мыши и выберем опцию **Properties** из появившегося локального меню. При этом возникает окно **Port Properties** (рис. 8.41).

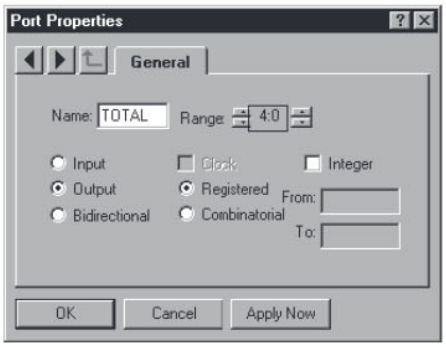


Рис. 8.41. Окно Port Properties

3. Введем имя нового порта TOTAL в поле **Name** и выберем 4:0 в поле **Range**. Выберем также для нашего порта две следующие опции: **Output** и **Registered**.
4. Завершая создание нового порта, щелкнем по клавише **OK**.

Рис. 8.42. Список выходных портов диаграммы состояний

Новые выходные порты перечисляются в верхнем правом углу главного окна графического редактора.

Цифровой автомат должен знать, есть ли среди полученных им карт козырной туз (ACE), так как, если общая сумма очков превышает 21 (при этом козырной туз считается за 11), существует возможность понизить ее, приняв козырной туз за 1.

Новая переменная Ace определяется в ходе следующей процедуры.

1. Щелкнем по клавише  на левой линейке инструментов.
2. Поместим курсор ниже метки Sreg0 и щелкнем клавишей мыши, при этом на диаграмме состояний появляется новый обьект — Signal1 (рис. 8.43).

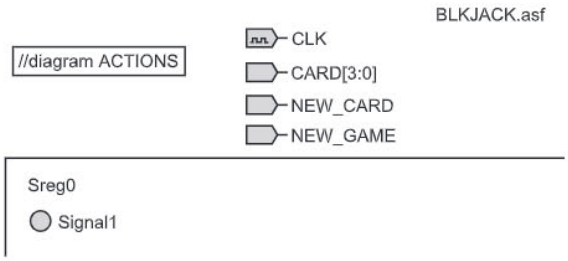


Рис. 8.43. Процедура определения дополнительной переменной

3. Щелкнем по иконке сигнала правой клавишей мыши и выберем опцию **Properties** в возникающем меню, что приводит к возникновению окна **Signal Properties** (рис. 8.44).
4. Введем имя Ace в поле **Name**.
5. Выберем в текущем окне опции **Combinatorial** и **Boolean**.
6. В заключение щелкнем вначале по клавише **OK**, а затем на свободном месте диаграммы состояний.

Каждый цифровой автомат должен быть инициализирован. В случае игры Blackjack инициализация должна выполняться при начале новой игры. Поэтому добавим к диаграмме состояний состояние сброса для выходных портов.

1. Щелкнем по клавише  , расположенной в вертикальной линейке инструментов.
2. Разместим символ состояния посредине диаграммы состояний (левая клавиша мыши используется для позиционирования графического символа состояния, а правая — для его фиксации

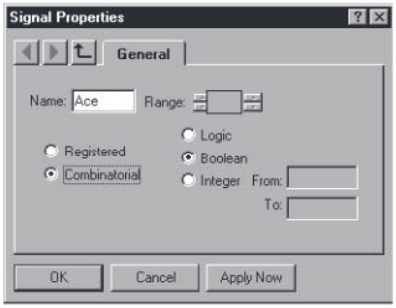


Рис. 8.44. Окно Signal Properties

в нужной позиции) (рис. 8.45). Новое состояние автоматически получает имя S1.

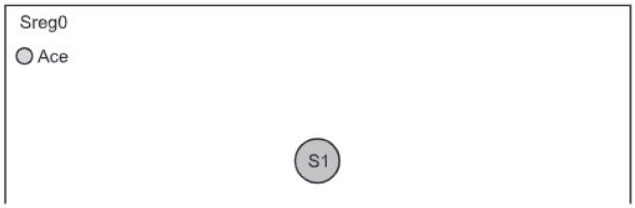


Рис. 8.45. Размещение графического символа состояния

3. Для редактирования имени состояния выполним двойной щелчок на имени S1. При этом открывается окно для редактирования (рис. 8.46).

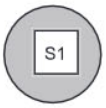


Рис. 8.46. Редактирование имени состояния

4. Заменим имя в окне для редактирования. Для окончания редактирования щелкнем по любому месту вне этого окна — появляется графический символ состояния с новым именем (рис. 8.47).

Рис. 8.47. Результат редактирования имени состояния

5. Выберем опцию **Sreg0** из меню **FSM\Machines**. При этом возникает окно **Machine Properties** (рис. 8.48). Это окно используется для определения глобальных параметров выбранного цифрового автомата. Некоторые из глобальных параметров, такие как **Sreg0**, именуются автоматически.
6. Введем вместо в поле. Таким образом, назначается имя для сигнала цифрового автомата (State Machine Signal), используемого для хранения текущего состояния в VHDL-коде.
7. Окно **Machine Properties** имеет несколько закладок для группировки различных опций.

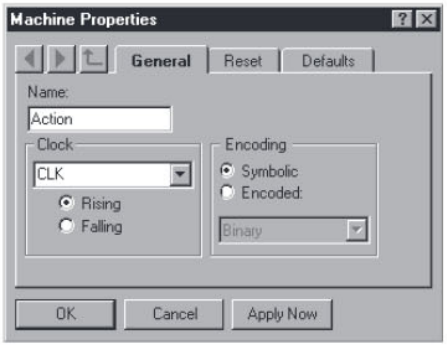


Рис. 8.48. Окно Machine Properties

Рис. 8.49. Установка сигнала сброса и состояния сброса

Щелкнем по закладке **Reset** (рис. 8.49) и произведем следующий выбор:
- сигнал сброса: NEW_GAME;
- состояние сброса (начальное состояние): START;

– тип сброса (синхронный): Synchronous;
– уровень активного сигнала сброса (высокий): High.

Это будет означать, что сигнал NEW_GAME будет использоваться как сигнал сброса для нашего цифрового автомата, и когда для этого сигнала будет установлен высокий уровень, цифровой автомат автоматически перейдет в состояние Start.

8. Щелкнем в заключение по клавише **OK** и увидим в окне графического редактора следующую диаграмму состояний (рис. 8.50).

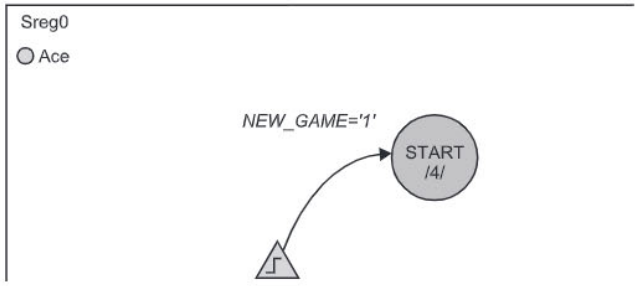


Рис. 8.50. Диаграмма состояний Action, включающая состояние сброса Start, сигнал сброса NEW_GAME, а также переменную Ace

После выполнения сброса автомат переходит в состояние **Start**. Теперь необходимо определить действия, связанные с этим состоянием (операции инициализации).

Для этого выполним следующие действия.

1. Щелкнем по кнопке **Entry Action**  .
2. Позиционируем указатель мыши на графическом символе состояния **State** и щелкнем левой клавишей мыши.
3. В открывшемся диалоговом поле введем код следующих операций инициализации:

```
Ace := false;
TOTAL <= "00000";
SAY_CARD <= '0';
SAY_HOLD <= '0';
SAY_BUST <= '0'.
```

4. В заключение щелкнем левой клавишей мыши вне диалогового поля, тем самым завершая ввод инициализирующих операций.

Заметим, что при выполнении описанных выше операций ввода мы использовали синтаксис языка VHDL:

– ввод строк завершался точкой с запятой;
– был использован оператор присвоения значений внутренним сигналам и портам интерфейса (которые также являются сигналами по определению);
– был использован оператор присвоения значений переменным :=;
– кавычки использовались для ввода строкового литерала, а апострофы — для ввода символьных литералов.

После введения кода операций инициализации наша диаграмма состояний примет следующий вид (рис. 8.51).

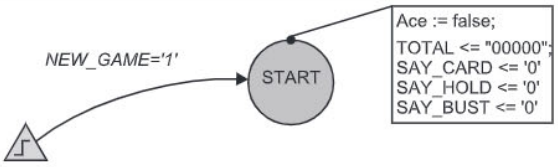


Рис. 8.51. Диаграмма состояний Action после введения операций инициализации

Следующим после состояния инициализации **Start** является состояние запроса карты, которому присвоим имя **Hit_me**. Для его создания предпримем следующие действия.

1. Добавим графический символ нового состояния к диаграмме состояний снизу от состояния **State,** используя описанную выше процедуру создания нового состояния;
2. Прочертим стрелку перехода из состояния **Start** в состояние **Hit_me**, щелкнув по кнопке **Transition**  и затем еще раз в пространстве между этими двумя состояниями.
3. Для завершения режима ввода щелкнем левой клавишей мыши, и диаграмма состояний Action примет следующий вид (рис. 8.52).

Для запроса новой карты необходимо установить сигнал SAY_CARD в единицу. Этот сигнал должен оставаться активным все время, пока автомат находится в данном состоянии. Для установки сигнала SAY_CARD в единицу выполним следующие действия.

1. Щелкнем по клавише **State Action**  .
2. Позиционируем указатель мыши на состоянии **Hit_me** и щелкнем левой клавишей мыши.
3. В поле редактирования введем SAY_CARD <= '1'.
4. После окончания редактирования щелкнем левой клавишей мыши и получим диаграмму состояний Action в следующем виде (рис. 8.53).

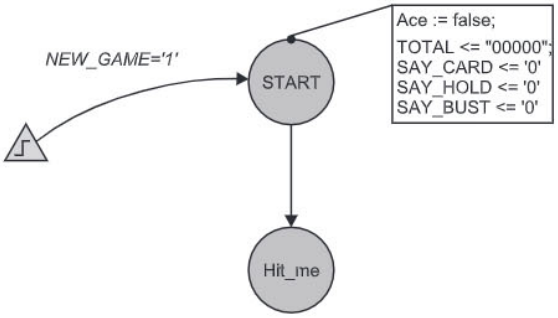


Рис. 8.52. Добавление к диаграмме состояний Action нового состояния Hit_me

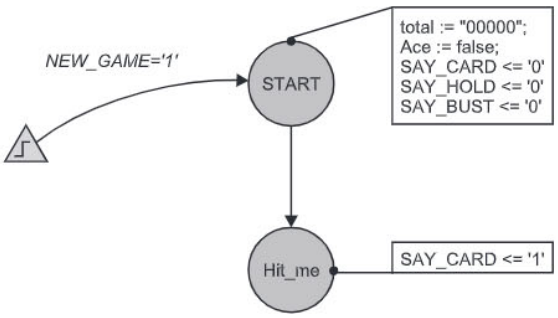


Рис. 8.53. Установка сигнала SAY_CARD в единицу для состояния Hit_me

К настоящему времени уже создан определенный фрагмент автомата. Перед дальнейшим наращиванием диаграммы состояний просмотрим VHDL-код, который уже сгенерирован, чтобы показать, как элементы диаграммы состояний соответствуют этому коду. Для этого сделаем следующее.

1. Выберем опцию **HDL Code Generation** из меню **Synthesis**.
2. В появившемся окне диалога **State Editor** щелкнем по клавише **Yes** для просмотра сгенерированного кода.

Сгенерированный с помощью State Editor VHDL-код можно разделить на следующие фрагменты

Library Section: всегда добавляется в начале проекта и обеспечивает доступ к библиотеке IEEE.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use lEEE.std_logic_unsigned.all;
```

Entity Declaration Section: содержит все порты, определенные в диаграмме состояний.

```
entity blkjack is
  port (CARD: in STD_LOGIC_VECTOR (3 downto 0);
        CLK: in STD_LOGIC;
        NEU_CARD: in STD_LOGIC;
        NEU_GAME: in STD_LOGIC;
        TOTAL: out STD_LOGIC_VECTOR(4 downto 0) ;
        SAY_BUST: out STD_LOGIC;
        SAY_CARD: out STD_LOGIC;
        SAY_HOLD: out STD_LOGIC);
end;
```

Global Declaration Section: определяет перечисляемый тип данных (enumerated type) и внутренний сигнал состояния выбранного перечисляемого типа для цифрового автомата, представленного на диаграмме, а также объекты и операции, являющиеся общими для всех цифровых автоматов (если таковых несколько).

```
architecture blkjack_arch of talkjack is
--auxiliary diagram declarations
--diagram DECLARATIONS;
-- SYMBOLIC ENCODED state machine: Action
type Action_type is (Hit_me, Start);
signal Action: Action_type;
begin
--concurrent signal assignment
--diagram ACTIONS;
```

Machine Declaration Section: определяет локальные объекты в процессах, содержащихся в модели цифрового автомата.

```
process  (CLK)
--auxiliary machine declarations
--machine DICLARATIONS
variable Ace: boolean;
```

Reset Definition Section: описывает операцию сброса для цифрового автомата.

```
begin
if CLK'event and CLK = '1' then
    if NEU_GAME = '1'  then
```

```
        Action  <= Start;
        TOTAL <=  "00000";
        Ace  :=  false;
```

Machine Action Description Section: описывает переходы цифрового автомата между определенными ранее его состояниями.

```
    else
    case Action is
        when  Hit_me  =>
        when  Start =>
           Action <= Hit_me;
        when  others  =>
           null;
    end case;
    end if;
end if;
end process;
```

Output Port Assignment Section: описывает процесс получения выходных сигналов цифрового автомата.

```
-- signal assignment statements for combinatorial
outputs
SAY_CARD <= '1'  when (Action = Hit_me)  else
            '0' ;
SAY_HOLD <= '0';
SAY_BUST <= '0';
end blkjack arch;
```

После того как новая карта затребована, ее необходимо дождаться: при появлении новой карты устанавливается в единицу входной сигнал NEW_CARD. После этого необходимо добавить к проекту фрагмент кода, обрабатывающий этот сигнал. Обработка включает следующие действия.

1. Добавим новое состояние к диаграмме состояний снизу от состояния **Hit_me**.
2. Назовем новое состояние **Got_it**, используя все ту же процедуру переименования **S1** в **Start**, описанную выше.
3. Щелкнем правой клавишей мыши внутри графического символа нового состояния (не «касаясь», однако, его имени).
4. В открывшемся кратком меню выберем опцию **Properties**.
5. Введем имя **Got_it** в поле **Name** открывшегося диалогового окна (рис. 8.54) и в качестве подтверждения щелкнем по клавише **OK**.
6. Добавим к диаграмме состояний переход (transition) из состояния **Hit_me** в состояние **Got_it**.
7. Добавим к только что обозначенному переходу условие его осуществления.

Для этого:

- щелкнем по клавише  **Condition**;
- щелкнем по линии перехода;
- введем в открывшемся диалоговом поле NEW_CARD = '1';
- щелкнем клавишей мыши вне этого диалогового поля, тем самым завершая процедуру ввода условия перехода.

Сейчас зададим действия автомата, выполняемые при входе в состояние (Entry Actions) **Got_it**, выбрав закладку **Actions** в окне **State Properties** (рис. 8.54):

- щелкнем правой клавишей мыши по графическому символу этого состояния;
- выберем опцию **Properties** в открывшемся кратком меню;
- щелкнем по закладке **Actions** (рис. 8.55);
- в поле **Entry** введем

```
TOTAL <= TOTAL + CARD;
Ace := (CARD = 11) or Ace;
```

- щелкнем по клавише **OK**, завершая тем самым ввод Entry Action.

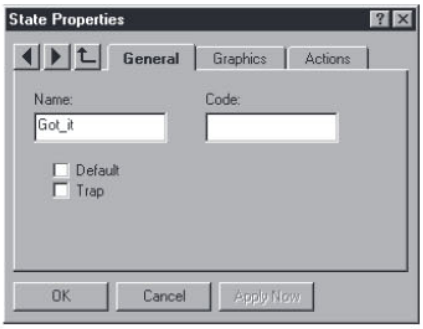


Рис. 8.54. Диалоговое окно State Properties

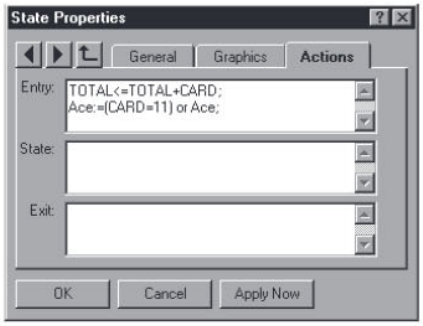


Рис. 8.55. Задание Entry Action для состояния Got_it

Теперь наша диаграмма состояний выглядит следующим образом (рис. 8.56).

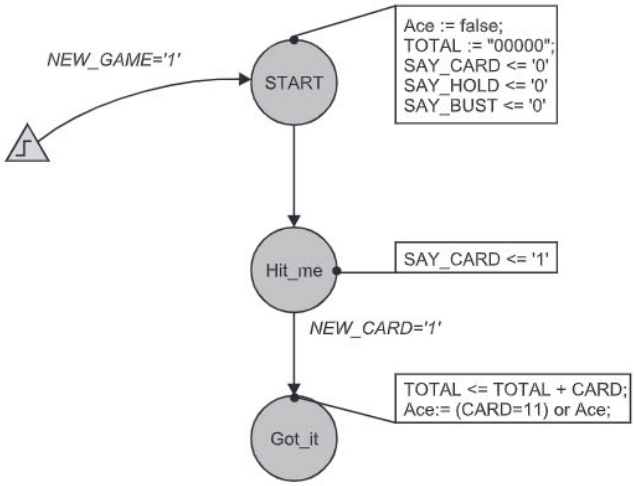


Рис. 8.56. Диаграмма состояний после добавления нового состояния Got_it

С получением новой карты общая сумма очков должна быть уточнена и проанализирована. Если она меньше 17, то автомат затребует новую карту и перейдет в состояние **Got_it**. Для реализации этого поведения:

– добавим состояние **Test16** к текущей диаграмме состояний снизу от состояния **Got_it**;
– добавим переход из состояния **Got_it** в состояние **Test16**;
– добавим условие этого перехода в виде NEW_CARD = '0'.

Теперь наша диаграмма состояний выглядит, как показано на рис. 8.57.

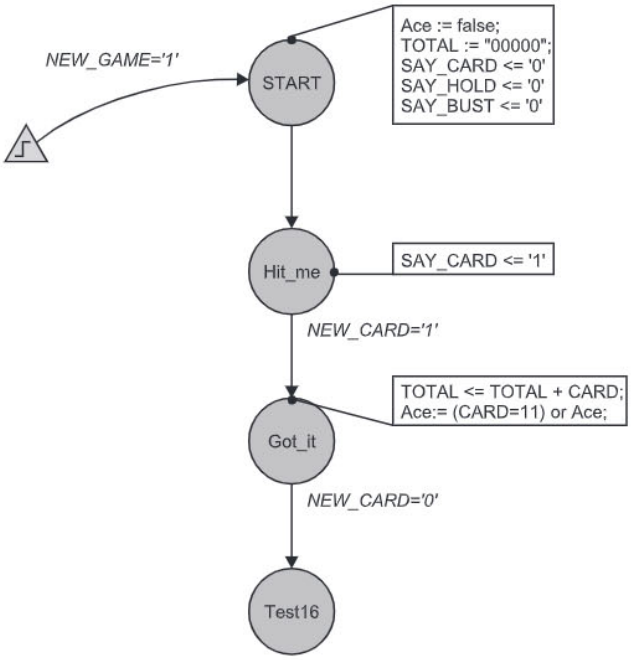


Рис. 8.57. Диаграмма состояний после добавления нового состояния Test16

Добавим теперь еще одно состояние **Test21** снизу от состояния **Test16**, а также переход из **Test16** в **Test21** с условием TOTAL > 16 и переход из **Test16** к **Hit_me** с условием @ELSE (условие @ELSE выполняется в случае, когда не выполняется ни одно из других условий перехода для данного состояния).

К настоящему моменту диаграмма состояний цифрового автомата уже содержит 5 состояний и выглядит следующим образом (рис. 8.58).

В момент, когда автомат переходит в состояние **Test21**, общая сумма очков превышает 16. Сейчас необходимо проверить, достигает ли она 21. Если нет, то устанавливается в единицу сигнал SAY_HOLD. Если же общая сумма очков превышает 21, то устанавливается в единицу сигнал SAY_BUST, тем самым сигнализируя о проигрыше цифрового автомата. Перед тем, как «вывесить флаг» SAY_BUST <= '1', автомат проверяет, находится ли в его распоряжении козырной туз (ACE). В случае если ACE присутствует, то общая сумма очков уменьшается на 10 для исправления ситуации «перебора».

Для описания такого поведения проделаем следующее.

1. Добавим к нашей диаграмме состояний еще два новых состояния: **Bust** и **Hold** (рис. 8.59).
2. Добавим теперь два соответствующих перехода в новые состояния: из **Test21** в **Bust** и из **Test21** в **Hold**.

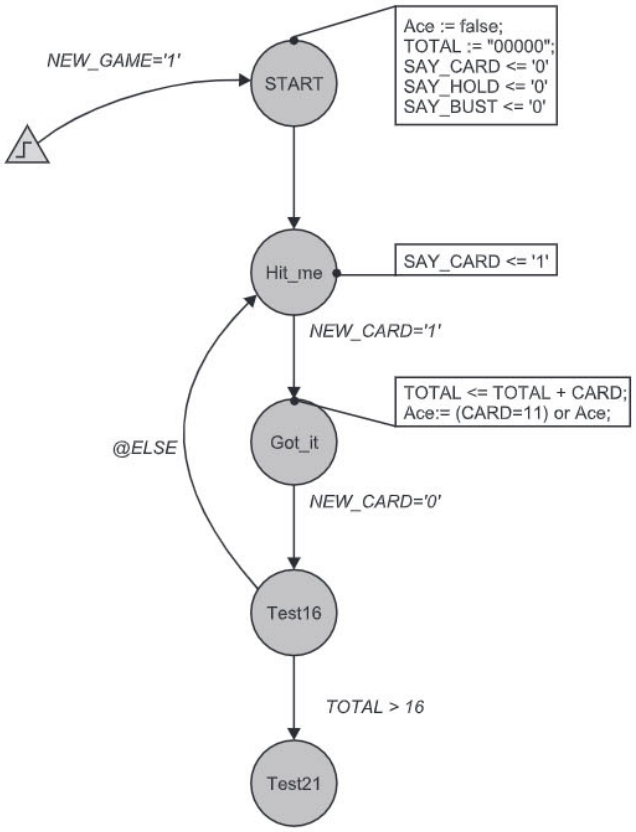


Рис. 8.58. Текущее состояние диаграммы состояний цифрового автомата

3. Переходу **Test21=>Hold** приписывается условие TOTAL<22.
4. К состоянию **Hold** «прикрепим» операцию SAY_HOLD <= '1'.
5. Добавим переход от состояния **Test21** к состоянию **Test16** с условием Ace (Ace является булевой переменной, поэтому не требуется никаких дополнительных операторов отношения в тексте этого условия).
6. Операцией, связанной с этим переходом, будет TOTAL <= TOTAL — 10 (при определении этой операции используется клавиша  , затем производится щелчок по линии перехода и ввод кода операции).
7. К переходу **Test21=> Bust** добавим условие @ELSE.

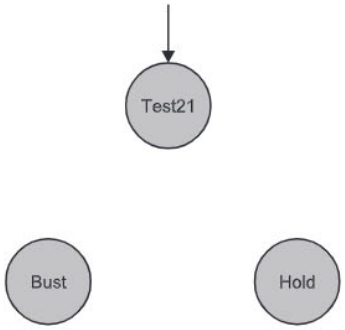


Рис. 8.59. Добавление двух новых состояний Bust и Hold

8. И последнее, с состоянием **Bust** связывается операция SAY_BUST <= '1'.

Окончательный вид диаграммы состояний приведен на рис. 8.60.

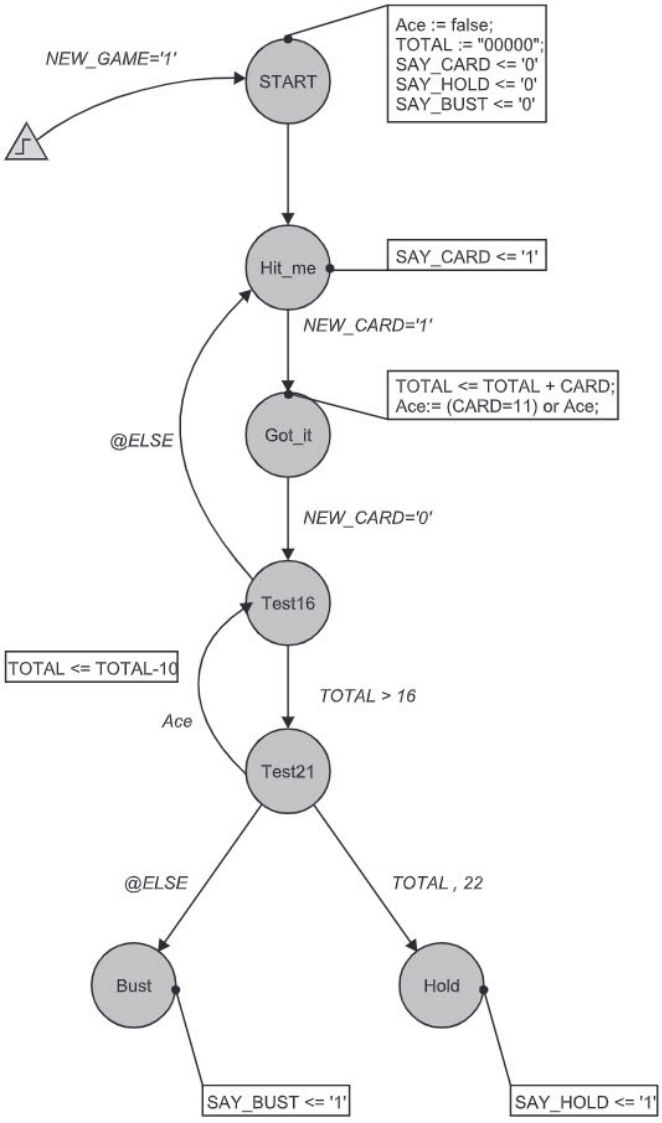


Рис. 8.60. Окончательный вид диаграммы состояний проектируемого цифрового автомата

Заметим, что в состоянии **Test21** оба условия TOTAL < 22 и Ace могут удовлетворяться в одно и то же время, поэтому необходимо назначить соответствующие приоритеты переходам, как показано на рис. 8.61.

Для этого необходимо выполнить следующие действия.

1. Щелкнем по соответствующей линии перехода правой клавишей мыши.
2. Выберем опцию **Priority** в возникающем кратком меню.
3. Выберем требуемый уровень приоритета и вновь щелкнем клавишей мыши. (Переходы без присвоенных приоритетов выполняются позже помеченных приоритетами переходов.)

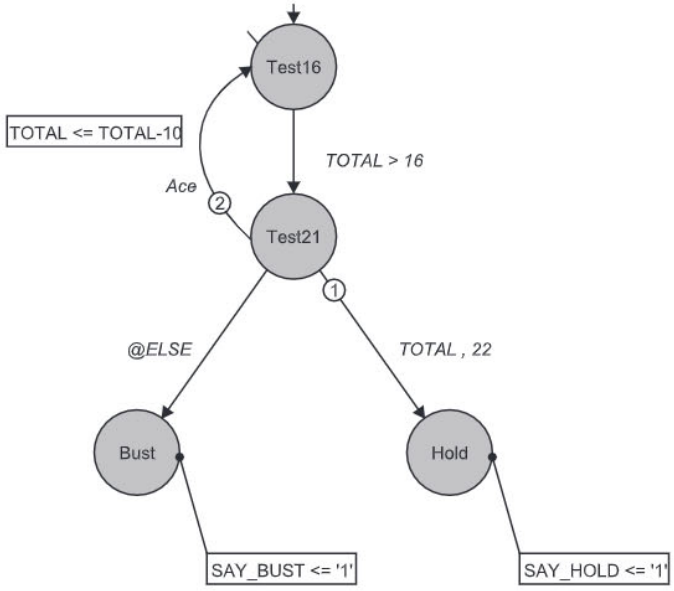


Рис. 8.61. Назначение приоритетов для переходов

Когда цифровой автомат компилируется в цифровую логическую схему, текущие состояния сохраняются в регистре, состоящем из ряда триггеров. Однако если существуют определенные параллельные переходы между состояниями, то могут возникнуть гонки (Races), порождаемые паразитными импульсами (Glitches) в комбинационной логике, и цифровой автомат окажется в неправильном состоянии (Wrong State). Рекомендуемым методом кодирования (Encoding Type) для всех FPGAs является так называемое **прямое кодирование** (One-Hot Assignment), при котором для каждого состояния только один бит регистра устанавливается в единицу. Однако такой метод кодирования требует избыточного количества триггеров и поэтому не рекомендуется для CPLDs с ограниченными ресурсами триггеров. Графический редактор поддерживает прямое кодирование и бинарное кодирование (binary coding). Другие методы кодирования могут быть созданы ручным присваиванием кодов каждому состоянию. Для примера используем бинарное кодирование.

1. Выберем опцию **Action** из меню **FSM\Machines**.

2. В появляющемся окне **Machine Properties** (рис. 8.62) выберем **Binary Encoded**.

(Опция **Symbolic**, в свою очередь, позволяет инструменту синтеза выбрать автоматически предпочтительный ему метод кодирования. Например, система синтеза Metamor XVHDL по умолчанию

Рис. 8.62. Выбор метода бинарного кодирования

выбирает бинарное кодирование, если у состояний нет кодов, созданных вручную.)

3. В заключение выберем опцию **Preferences** из меню **Tools** и отметим в одноименном окне (рис. 8.63) опцию **Display State Codes** для того, чтобы индицировать связанные с состояниями коды, а затем, завершая операцию кодирования, щелкнем по клавише **OK**.

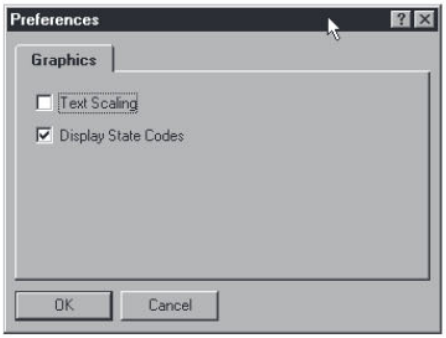
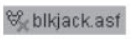


Рис. 8.63. Выбор опции Display State Codes для визуализации кодов состояний

8.5.3. Локализация и корректировка ошибок, синтез проекта

При добавлении файла диаграммы состояний к определенному проекту он анализируется с помощью системы синтеза (Synthesis Tool):

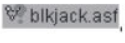
- сохраняем диаграмму состояний;
- выберем **Add to project** из меню **State Editor/ Project** и перейдем к окну **Project Manager**.

В этом окне теперь можно видеть иконку файла *blkjack.asf* 

(в закладке **Files**). Красный крестик на иконке говорит о наличии ошибок, выявленных в процессе анализа автоматически сгенерированного кода.

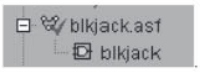
В сгенерированном коде есть одна ошибка, связанная с определением типа порта TOTAL. Сигнал TOTAL определен как OUT, в связи с чем он не может читаться внутри архитектурного тела. Поэтому всякий раз, когда сигнал TOTAL встречается в правой части операторов присваивания для сигналов (Signal Assignments), мы будем получать сообщение об ошибке. Для корректировки этой ошибки выполним следующие действия:

- перейдем в окно **State Editor**;
- щелкнем по символу порта TOTAL правой клавишей мыши и в возникшем кратком меню выберем опцию **Properties** — это приводит к открытию диалогового окна **Port Properties**;
- изменим тип порта на **Bi-directional**;
- сохраним измененный файл диаграммы состояний.

После описанных выше действий в окне **Project Manager** наблюдаем измененную иконку нашего файла 

, свидетельствующую о том, что вновь требуется выполнить процесс анализа. Поэтому щелкнем по иконке правой клавишей мыши и выберем в открывшемся кратком меню опцию **Analysis**. Результатом

нового анализа является новое изменение иконки файла 

, свидетельствующее об отсутствии ошибок в файле диаграммы состояний. Слева от этой иконки находится маленький крестик, щелчок по нему приводит к появлению диаграммы иерархии проекта (Design Hierarchy) 

В данном случае, например, видно, что проект содержит только один объект (Design Entity) по имени blkjack.

Для получения физической реализации сгенерированного к настоящему моменту VHDL-кода диаграммы состояний щелкнем по иконке **Synthesis** 

на закладке **Flow**. Это приводит к появлению диалогового окна (рис. 8.64) **Synthesis/Implementation settings**.

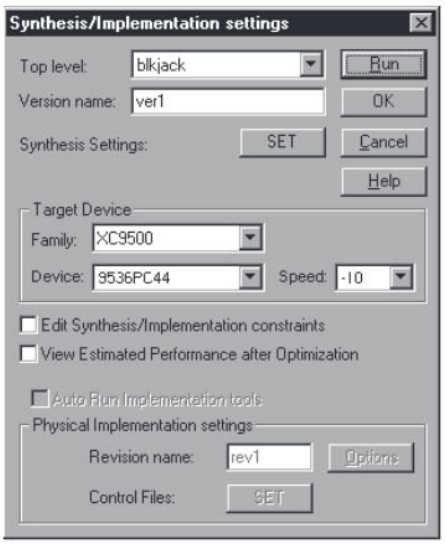
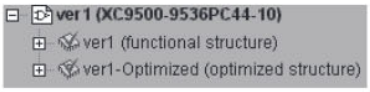


Рис. 8.64. Окно для задания опций физической реализации проекта

В случае принятия предлагаемых в окне опций необходимо щелкнуть по клавише OK и подождать завершения процесса синтеза. После чего перейдем в закладку Version окна Project Manager и увидим здесь иерархию синтезированного проекта:



(У нас есть возможность сейчас щелкнуть по иконке версии правой клавишей мыши и выбрать опцию View Report из краткого меню, что приведет к просмотру отчета о выполненном синтезе (Synthesis Report), который очень важен тем, что показывает, как много ресурсов затрачено на проект. Проделав определенные изменения в диаграмме состояний, можно получить ощутимое увеличение или уменьшение числа используемых триггеров (CLBs — Configurable Logic Blocks).)

8.5.4. Функциональное моделирование проекта

Следующим шагом в физической реализации проекта является функциональное моделирование. Это позволяет проверить, удовлетворяет ли реализация проекта в ее теперешнем виде всем требованиям к его функционированию. Для проведения функционального моделирования выполним следующие действия.

1. Щелкнем по кнопке  на закладке **Flow** для запуска логического симулятора **Logic Simulator**.

2. Выберем опцию **Add Signals** в меню **Signal**.

3. С помощью двойного щелчка выберем следующие сигналы для просмотра в окне **Waveform Viewer** (рис. 8.65). Здесь переменные состояния представлены тремя следующими сигналами:

   – ACTION_0;
   – ACTION_1;
   – ACTION_2.

С помощью этих трех сигналов отображается текущее состояние автомата (Current Machine State). Заметим, что окно **Waveform Viewer** может отображать шины в сжатом или развернутом виде. Для задания режима отображения существует кнопка  .

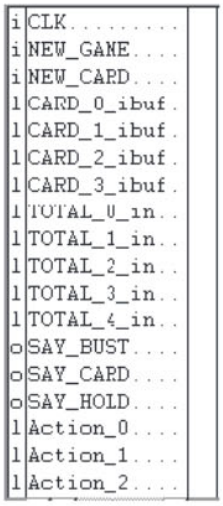


Рис. 8.65. Список сигналов для просмотра в окне Waveform Viewer

4. Закроем текущее диалоговое окно и выберем опцию **Add Stimulators** из меню **Signal** (для задания входных воздействий для выполнения процесса моделирования) и поставим в соответствие входу CLK источник входного воздействия (stimulator) B0.

5. Щелкнем по клавише **Formula** в окне **Stimulator Selection** и определим следующие три источника входных воздействий F0, F1 и F2 (рис. 8.66):

– присвоим формулу (H20L580)10 для источника F0;

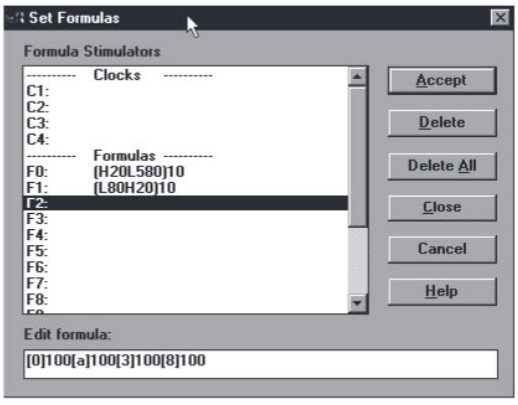


Рис. 8.66. Задание трех источников входных воздействий

– присвоим формулу (L80H20)10 для источника F1;
– присвоим формулу [0]100[a]100[3]100[8]100 для источника F2.

6. Закроем текущее окно, щелкнув по клавише **Close**.
7. Соединим:
– источник F0 с входом NEW_GAME;
– источник F1 с входом NEW_CARD;
– источник F2 с входом-шиной CARD (эта шина задает число очков на карте: от 2 до 11).
8. Удерживая клавишу **Ctr**, выберем шины CARD, ACTION_0 и TOTAL.
9. Для этих шин выберем опцию **Display Decimal** в меню **Signal\Bus**, чтобы получать их значения в десятичном виде.

После описанных действий имена сигналов выглядят в окне **Waveform Viewer** следующим образом (рис. 8.67).

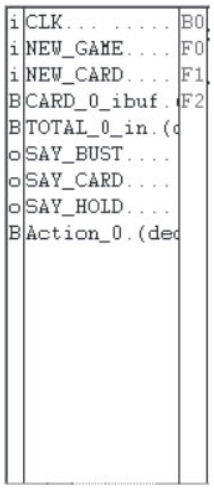


Рис. 8.67. Список сигналов в окне Waveform Viewer указанием источников внешних воздействий

Теперь с помощью кнопок **Step** и **Long** будем запускать процесс моделирования о тех пор, пока не будет установлен в единицу сигнал SAY_HOLD или сигнал SAY_BUST.

Примерные результаты моделирования приведены на рис. 8.68.

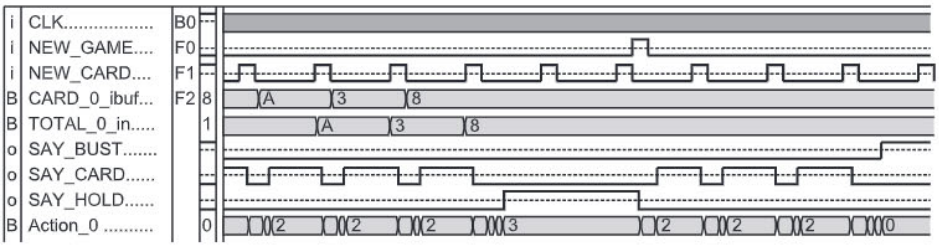
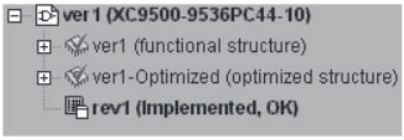


Рис. 8.68. Результаты моделирования в окне Waveform Viewer

8.5.5. Физическая реализация проекта

Для завершения разработки проекта — получения его физической реализации щелкнем по кнопке **Implementation** в закладке **Flow** окна **Project Manager**.

Затем подождем, когда процесс получения физической реализации проекта завершится, и тогда перейдем в закладку **Versions**. Здесь можно просмотреть иерархию физической реализации проекта:



Теперь проект готов для выполнения временной верификации (Timing Verification), а также для программирования микросхемы. Для пользователя доступны два инструмента временной верификации в закладке **Flow**:

- **Timing Simulation**;
- **Timing Analysis**.

С их помощью можно проверить соответствие проекта определенным временным ограничениям (Timing Requirements) перед программированием соответствующей микросхемы.

Таким образом, графический редактор цифровых автоматов State Editor является мощным инструментом для создания управляющих устройств с помощью языков описания аппаратуры (HDLs). Графические диаграммы являются превосходным средством документирования проекта и позволяют получать эффективные и надежные результаты для процесса логического синтеза проекта.

Глава 9. Работа с VHDL в среде системы моделирования VHDL Simili

9.1. Введение в VHDL Simili

Система моделирования **VHDL Simili** фирмы **Symphony EDA** содержит очень быстрые компилятор (**VHDL Compiler**), симулятор (**VHDL Simulator**) и интегрированную среду проектирования, пригодную для работы с Windows и с Linux. Начиная с версии 2.1, фирма **Symphony EDA** предлагает графический интерфейс пользователя (рис. 9.1).

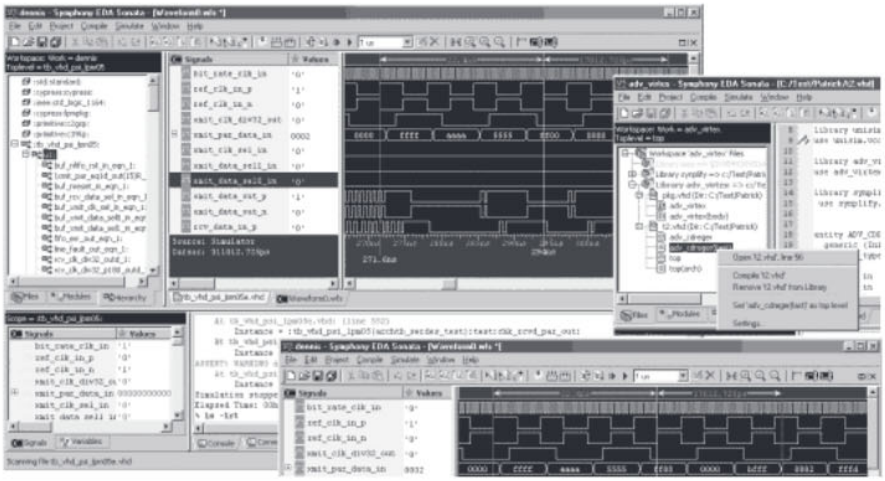


Рис. 9.1. Система моделирования VHDL Simili с графическим интерфейсом Sonata

После инсталляции системы создается программная группа с меню, приведенным на рис. 9.2.

Пункт «Browse VHDL Simili» позволяет познакомиться с системой моделирования VHDL Simili Environment. Пункт «Documentation» открывает страницу документации верхнего уровня в HTML. Пункт Sonata означает вход в интегрированную среду проектирования Symphony EDA's Integrated Development Environment, которая является графическим интерфейсом пользователя для системы моделиро-

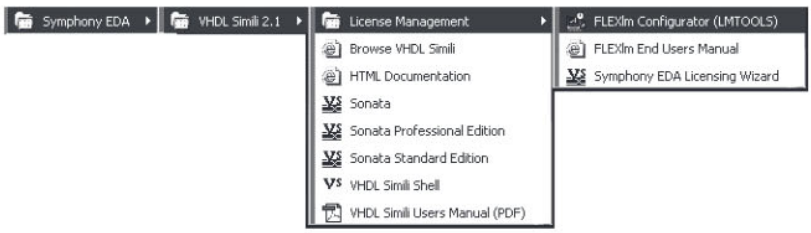


Рис. 9.2. Меню программной группы VHDL Simili Environment

вания VHDL Simili. Пункт «VHDL Simili Shell» создает среду для командной строки в окне MS DOS.

Sonata является мощной средой для управления большими VHDL-проектами и служит интерфейсом для компилятора и симулятора. Сами по себе компилятор и симулятор являются инструментами, вызываемыми из командной строки следующими двумя командами соответственно: vhdlp и vhdle. Sonata делится на четыре главные панели с изменяемыми габаритами (рис. 9.3).

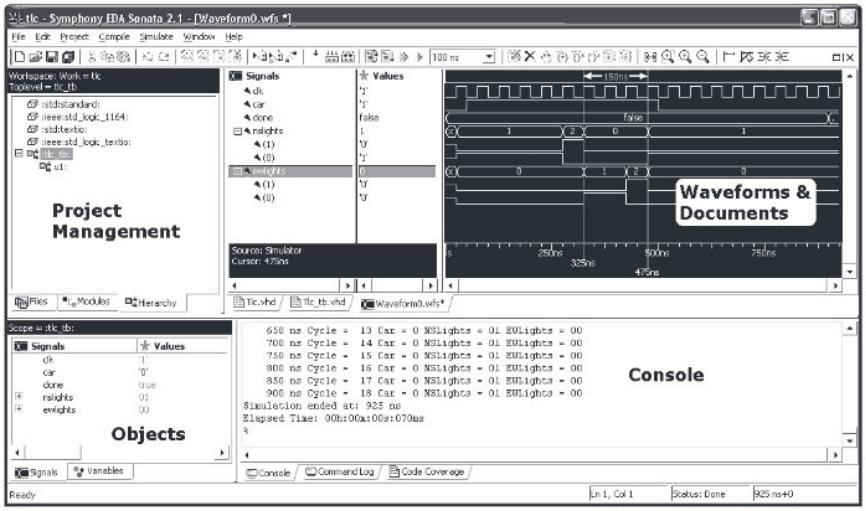


Рис. 9.3. Четыре главные панели интерфейса Sonata

Панель **Project Management** отображает статус текущего проекта. Эта панель содержит следующие закладки:

- **Files**: эта закладка отображает проект в режиме просмотра «File». Так, можно просмотреть различные библиотеки проекта и находящиеся в них файлы исходного кода (source files);
- **Modules**: закладка отображает проект в скомпилированной форме, т. е. проектная информация включает лишь действительно откомпилированные файлы в каждой библиотеке;
- **Hierarchy**: эта закладка используется только в течение активного сеанса моделирования (currently active simulation session). Она отображает иерархическую структуру проекта, подлежащего моделированию.

Панель **Document** предназначена для управления всеми открытыми документами. Документом является текстовый файл или файл временных диаграмм (waveform file).

Панель **Console** отображает все сообщения от различных программных инструментов. Она включает следующие закладки:

- **Console**: является вариантом известного окна TCL (Tool Control Language) Command Window. Консоль воспринимает команды из командной строки как TCL-команды;
- **Command Log**: это окно отображает все вводимые на консоль команды.

Панель **Objects** содержит актуальные данные (Valid Data) лишь в течение активного сеанса моделирования. Она содержит следующие закладки:

- **Signals**: здесь отображаются все сигналы и их текущие значения. Отсюда можно извлекать определенные сигналы и переносить их в окно временных диаграмм (Waveform Window), отображающее текущий сеанс моделирования;
- **Variables**: отображает значения констант, параметров и переменных.

9.2. Управление проектами в среде Sonata (Sonata-Project Management)

Среда **Sonata** является интегрированной средой разработки VHDL-проектов (Integrated Development Environment-IDE). Каждый проект представляется рабочим пространством (Workspace), также называемым проектом. Функции рабочего пространства, такие как открытие (opening), сохранение (saving), закрытие (closing) и создание рабочих пространств, доступны из меню File. Рабочее пространство создается выбором File->New Workspace... Этот выбор приводит к появлению окна диалога **Create New Workspace** (рис. 9.4).

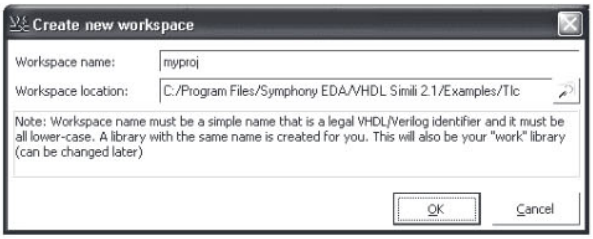


Рис. 9.4. Окно диалога Create New Workspace

Используя это диалоговое окно, можно создать рабочее пространство в директории по своему выбору. Заметим, что все файлы и библиотеки данного рабочего пространства будут относиться именно к этой директории независимо от их реального расположения на диске.

При создании рабочего пространства создается также библиотека с тем же самым именем. Эта библиотека становится текущей рабочей библиотекой. Когда рабочее пространство создается, оно сразу же инициализируется с помощью файла **symphony.ini**. Если этот файл существует в текущей директории, то именно он и будет использован. В противном случае для вышеуказанной цели используется файл **symphony.ini**, находящийся в директории **Bin**, созданной при инсталляции системы.

Как только рабочее пространство будет создано, становится возможным выполнение всех других задач. Следует помнить, что все компиляции и все процессы моделирования выполняются относительно все той же директории, специфицированной для нашего рабочего пространства.

Функция библиотек может быть описана следующими ключевыми положениями.

1. Каждый проект в среде **Sonata** является рабочим пространством, которое управляет одной или несколькими библиотеками.
2. Каждая библиотека является VHDL-библиотекой.
3. Библиотека содержит набор объектов проекта (Design Units). Объектами проекта являются интерфейсы сущности, архитектуры, конфигурации, заголовки и тела пакетов.

4. Существует концепция текущей рабочей библиотеки (current working library). Она отображается в левой верхней части панели **Project Management**. Кроме того, текущая рабочая библиотека выделена красным цветом в закладках **Files** и **Modules** этой панели.
5. Библиотека может быть предназначена только для чтения (read-only). Такая библиотека не позволяет компилировать VHDL-файлы для создания/модификации объектов проекта.
6. Все операции, относящиеся к управлению проектами, применяются к текущей рабочей библиотеке (current working library).
7. Каждая библиотека может содержать набор VHDL-файлов. Файлы могут быть добавлены к библиотеке с помощью контекстного меню, вызываемого щелчком правой клавишей мыши или с помощью меню **Project**.
8. Можно отсоединить файл от библиотеки выбором имени данного файла на закладке **Files** и нажатием клавиши **Delete**. Заметим, что удаление файла из библиотеки не означает физического удаления данного файла из файловой системы.

Меню **Project** содержит следующие функции управления библиотеками:

1) добавление или удаление файлов;
2) переупорядочение файлов;
3) изменение текущей рабочей библиотеки;
4) удаление скомпилированных объектов (не файлов исходного кода) из текущей рабочей библиотеки;
5) изменение статуса «только для чтения» (read-only status) для текущей библиотеки;
6) присоединение существующей библиотеки к текущему рабочему пространству или отсоединение ее.
7) создание новой библиотеки и присоединение ее к текущему рабочему пространству.

Меню **Compile** предназначено для выполнения процессов компиляции. Используя это меню, можно выполнить следующие задачи.

1. Compile the current file being edited: скомпилировать отредактированный файл, находящийся в текущей рабочей библиотеке.
2. Compile selected files: скомпилировать несколько выбранных (в закладке **Files** окна **Project Management**) файлов.
3. Build: скомпилировать все файлы, нуждающиеся в компиляции (производится «интеллектуальная» компиляция — smart compilation — в зависимости от временных меток (timestamps) создания и модификации файлов).
4. Compile All: компилировать все файлы независимо от их временных меток.

В среде **VHDL Simili** VHDL-файлы компилируются с помощью компилятора **vhdlp**, а моделирование выполняется с помощью симулятора **vhdle**.

9.3. Установка опций Sonata-Workspace

Установка (модификация) опций для рабочего пространства может быть выполнена с помощью выбора пунктов меню **Project -> Settings** или **Compile -> Settings**. Далее мы обсудим три вида установки опций.

1. Установка опций для компилятора.

Закладка установки опций для компилятора **Compiler Settings** (рис. 9.5) отображает содержимое рабочего пространства в древовидной форме (слева) и опции данного объекта рабочего пространства (справа).

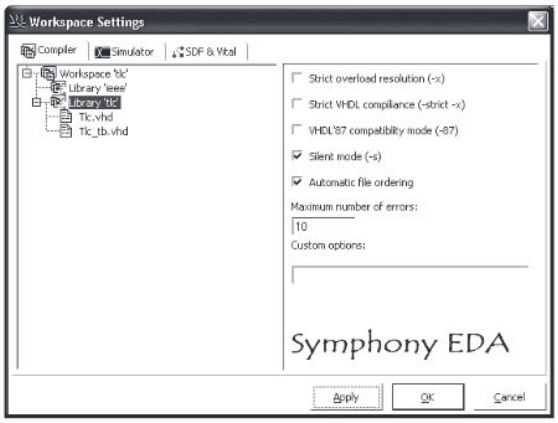


Рис. 9.5. Закладка Compiler Settings

2. Установка опций для симулятора.

Закладка установки опций для симулятора **Simulator Settings** (рис. 9.6) отображает опции управления для среды моделирования (simulation environment).

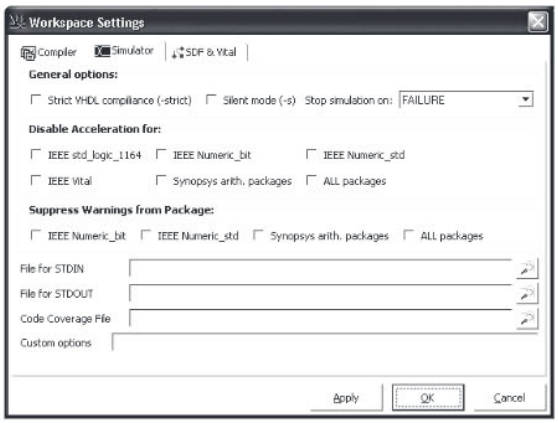


Рис. 9.6. Закладка Simulator Settings

Здесь секция **General options** содержит общие опции, контролирующие поведение симулятора. Секция **Disable Acceleration for** позволяет отменить ускорение для процесса компиляции за счет использования встроенных программных пакетов (built-in packages). Таким образом, выбор этой опции ведет к замедлению процесса моделирования. Секция **Suppress Warnings from Package** позволяет подавить сообщения вида **Warning**, встроенные в программные пакеты симулятора. У нас есть возможность специфицировать файлы для входного (STDIN) и выходного (STDOUT) стандартных потоков. В поле **Custom options** можно специфицировать произ-

вольные опции. При этом именно пользователь является ответственным за корректность вводимых опций.

3. Установка опций для **SDF&Vital**.

Используя закладку **SDF/Vital Settings** (рис. 9.7), можно специфицировать SDF-параметры.

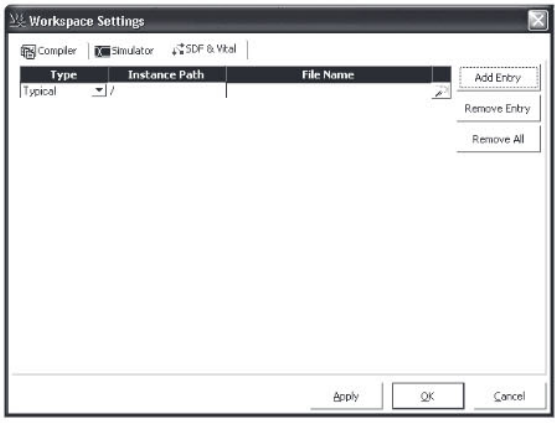


Рис. 9.7. Закладка SDF/Vital Settings

9.4. Предпочтения (Preferences) Sonata-User

Пользователь может управлять многими визуальными и поведенческими свойствами графического интерфейса **Sonata**. Стартуя, **Sonata** отображает местоположение файла предпочтений (preference file). Хотя этот файл является текстовым, не рекомендуется редактировать его вручную. Вместо этого необходимо использовать меню **Edit -> Preferences** для редактирования предпочтений. На рис. 9.8—9.10 приведены соответственно диалоги редактирования предпочтений для временных диаграмм, окна консоли и текстового редактора.

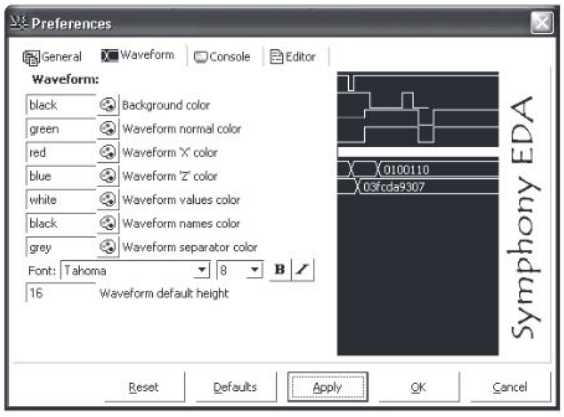


Рис. 9.8. Окно диалога Waveform Preferences

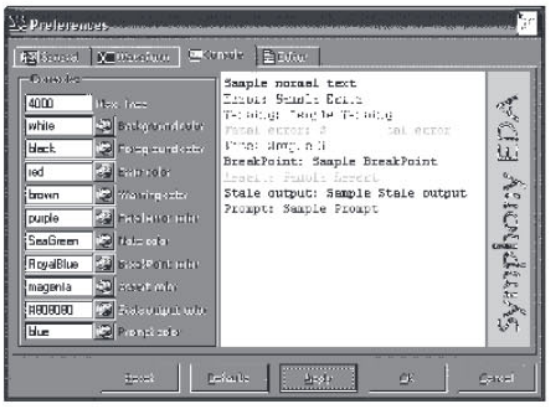


Рис. 9.9. Окно диалога Console Window Preferences

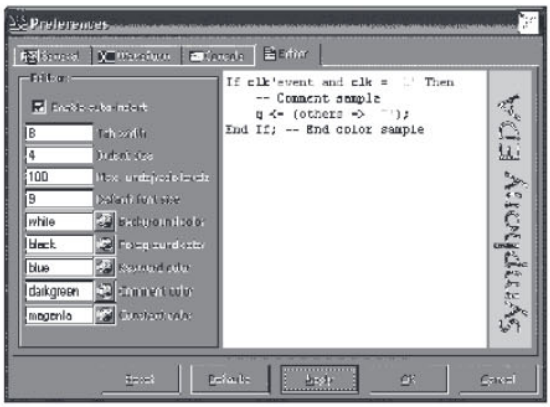


Рис. 9.10. Окно диалога Waveform Preferences

9.5. Аргументы командной строки

В Windows редко возникает необходимость использовать аргументы командной строки. Однако это полезная возможность управления интерфейсом **Sonata** при запуске. Общий формат командной строки таков:

sonata [options] [--] [file1 ... fileN] [workspacename].

Sonata поддерживает следующие аргументы командной строки (на всех аппаратных платформах):

- uselicense license-feature-name — таким образом, можно проверить наличие лицензии на определенную версию **Sonata** (sonatapro, sonata или sonatafree);
- L — эта опция активизирует мастера лицензирования (licensing wizard);
- guisource filename — может быть использована для исходного Tcl/ Tk-файла в контексте интерфейса **Sonata** автоматически при его запуске;
- source filename — может быть использована для исходного Tcl-файла в нормальном контексте при запуске;

- tclcmd command — эта опция может быть использована для запуска любой Tcl-команды во время запуска самого интерфейса;
- маркирует конец списка аргуметов;
- workspacename — специфицирует имя файла (с расширением .sws) опционального рабочего пространства, которое заменит открытое последним рабочее пространство;
- file1 ... fileN — список текстовых файлов, открываемых в текстовом редакторе (**Sonata Text Editor**) при запуске интерфейса.

9.6. Моделирование с графическим интерфейсом Sonata

В этом разделе мы обсудим процесс моделирования и просмотр его результатов с помощью окна временных диаграмм (Waveform Viewer).

9.6.1. Процесс моделирования

Предположим, что все необходимые файлы скомпилированы успешно. Тогда для выполнения моделирования необходимо выполнить следующие шаги.

1. Прежде всего необходимо идентифицировать верхний уровень иерархии проекта (Top-Level of the Design Hierarchy). Верхним уровнем проекта является имя интерфейса, или архитектуры, или конфигурации, который(ая) служит корнем (root) нашего проекта. В большинстве случаев таким корнем является интерфейс испытательной программы (Test Bench). Если верхний уровень проекта уже установлен, мы увидим его имя в верхнем левом углу панели **Workspace**. Для изменения или установки верхнего уровня проекта можно использовать меню **Simulate -> Select toplevel...**
2. Запуская компилятор **vhdle**, **Sonata** базируется на установках для рабочего пространства (для их изменения необходимо использовать меню **Project -> Settings**). Одна из наиболее важных опций — **SDF**-установки для задержек. **Sonata** работает с симулятором **vhdle** в режиме клиент-сервер, где симулятор является сервером в фоновом режиме, а **Sonata** становится клиентом. Все установки рабочего пространства передаются симулятору.
3. Самой важной составной частью работы симулятора является уточнение модели (Elaboration — см. главу 2). При этом симулятор считывает скомпилированную форму проекта с диска, идентифицирует верхний уровень проекта, а затем устанавливает все единицы проекта (component instances), начиная с

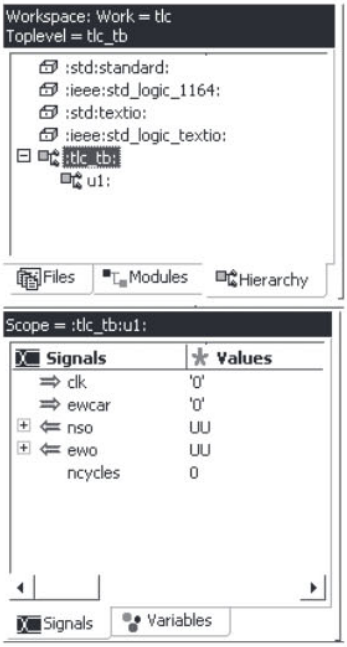


Рис. 9.11. Панели Hierarchy и Objects после стадии уточнения модели

верхнего уровня проекта. Результатом этого процесса является инициализация всего проекта и установление его иерархической структуры (рис. 9.11). После этой стадии симулятор ждет инструкций от клиента (пользователя или **Sonata**) для выполнения моделирования.

4. После того как уточнение модели завершилось, **Sonata** уточняет свое состояние в соответствии с информацией об иерархии проекта, полученной от фонового симулятора. Кроме того, **Sonata** создает окно **Waveform** (на панели **Document**). В нижнем правом углу линейки состояния (**Sonata Status Bar**) отображается текущее состояние симулятора (**Simulation Run-Status**, **Current Simulation Time**):

Когда симулятор остановлен (находится в состоянии stopped state), можно работать с панелями **Hierarchy** и **Objects** (при запуске симулятора это невозможно по соображениям поддержания должной производительности симулятора). В любое время можно использовать кнопку  для того, чтобы остановить симулятор и взаимодействовать с вышеупомянутыми панелями. Затем можно вновь запустить симулятор с помощью кнопок  или  .

5. В конце стадии уточнения модели время моделирования полагается равным нулю. Продвижение времени моделирования осуществляется двумя путями:
   - фиксированным шагом с помощью органа управления для спецификации его величины  ;
   - по мере поступления событий. При этом нажатие кнопки приводит к выполнению моделирования на всем запланированном интервале (до тех пор, пока все события не будут обработаны, или не будет достигнута верхняя граница интервала моделирования TIME'HIGH).

9.6.2. Панель Objects

Объекты VHDL-проекта имеют определенные значения. Примерами объектов являются:

- сигналы (signals);
- переменные (variables);
- параметры (generics);
- константы (constants) и т. д.

Не являются объектами:

- типы данных (types);
- подтипы данных (subtypes);
- подпрограммы (subprograms) и др.

Панель **Objects** отображает текущее состояние моделируемых объектов. В отличие от окна временных диаграмм, в котором отображаются текущие значения объектов и вся их предыстория, панель **Objects** обеспечивает лишь моментальный снимок состояния объектов (отображение истории всех объектов на панели **Ob-**

jects требует памяти в несколько гигабайт даже для моделирования проектов умеренных размеров).

Текущее содержимое для этой панели может быть определено на закладке **Hierarchy** панели **Workspace** следующим образом:

– сделать видимым интересующую часть иерархии проекта;
– дважды щелкнуть по интересующему нас пункту иерархии проекта или воспользоваться контекстным меню (context-sensitive menu), вызываемым щелчком правой клавиши мыши.

Панель **Objects** уточняется всякий раз, когда в работе симулятора есть пауза.

9.6.3. Временные диаграммы (Waveforms)

Временная диаграмма является графической формой представления истории изменений величин сигналов в данном проекте. Окно временных диаграмм, связанное с симулятором, является динамическим окном временных диаграмм (dynamic waveform window — изменяется по мере продвижения времени моделирования), а другие окна временных диаграмм, представляющие файлы на диске, являются статическими (никогда не изменяющимися).

Большинство операций с временными диаграммами выполняется с помощью контекстного меню или с помощью следующей линейки кнопок:

Контекстное меню (рис. 9.12) вызывается из окна временных диаграмм нажатием правой клавиши мыши.

Управление просмотром диаграмм осуществляется с помощью следующих клавиш:

 — увеличение изображения (Zoom);

 — уменьшение изображения (Zoom out);

 — увеличение определенной области окна временных диаграмм;

 — просмотр временной диаграммы на всю ее длину.

Sonata поддерживает два вида электронных закладок для временных диаграмм (**Waveform Bookmarks**):

– маркеры (Markers, Waveform Cursors): создаются нажатием левой клавиши мыши;
– виды (Views): создаются нажатием правой клавиши мыши.

Электронные закладки могут быть созданы с помощью окна диалога (рис. 9.13), вызываемого с помощью кнопки  или с помощью опции **Bookmarks...** в рассмотренном выше контекстном меню.

У нас есть возможность сохранить командный файл, который позволит восстановить диаграммы в будущем сеансе работы (**Sonata Session**). Для создания такого командного файла надо использовать пункт контекстного меню **Save 'add wave'**

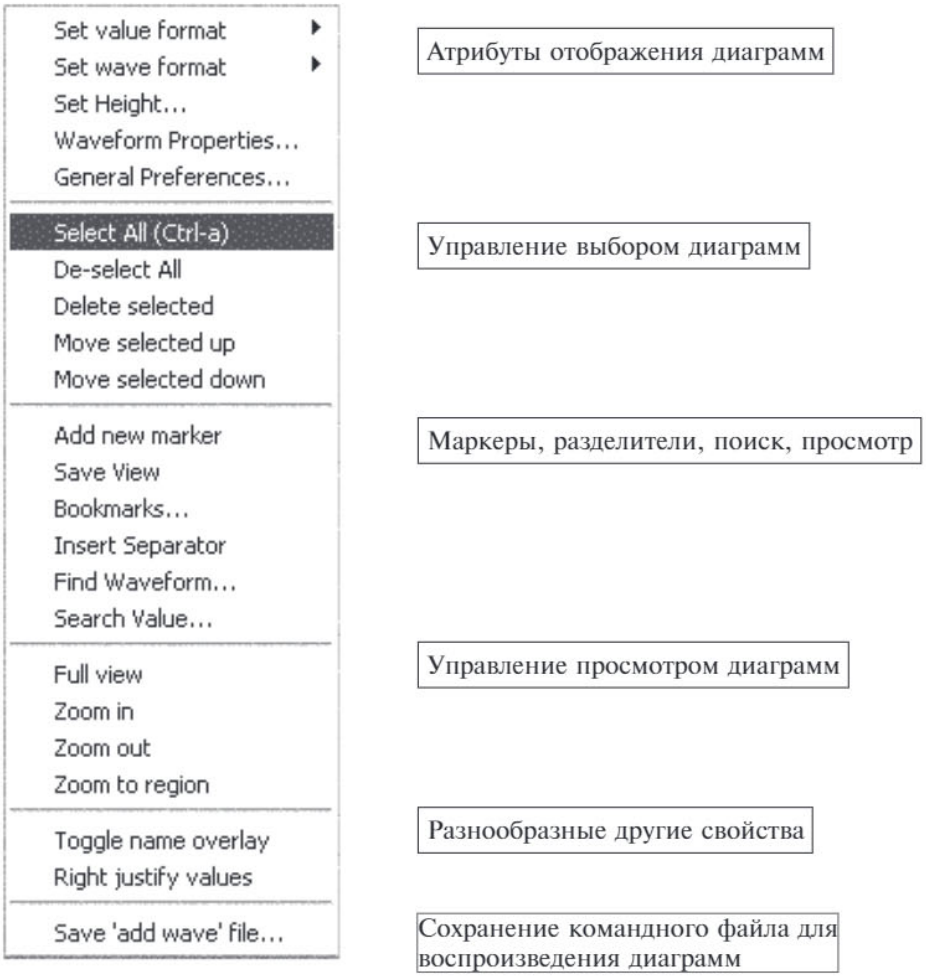


Рис. 9.12. Контекстное меню в окне Waveform

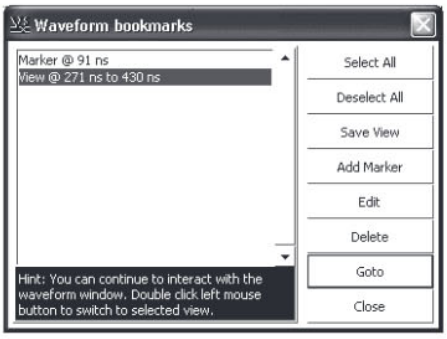


Рис. 9.13. Окно диалога Waveform Bookmark Management

file... После того как командный файл сохранен, можно восстановить эти временные диаграммы с помощью Tcl-команды source filename.tcl в окне консоли.

Для управления свойствами отображения диаграмм необходимо воспользоваться пунктом **Waveform Properties** контекстного меню. При этом возникает следующее окно диалога (рис. 9.14).

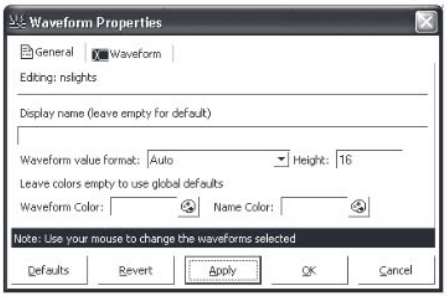


Рис. 9.14. Окно диалога Controlling Waveform Display Properties

Используя это окно, можно управлять отображением имен, цветов, высотой временных диаграмм, форматом их печати. Для управления форматом временных диаграмм (цифровой, аналоговый) используется окно диалога на рис. 9.15:

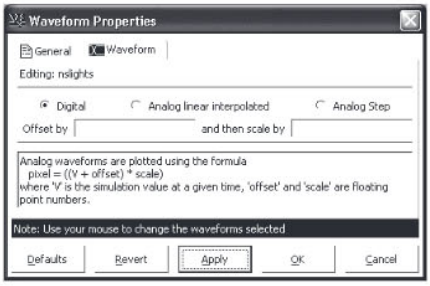


Рис. 9.15. Окно диалога Waveform format-digital or analog

И наконец, для выполнения процессов поиска различных величин и моментов определенных событий используется следующее диалоговое окно (рис. 9.16).

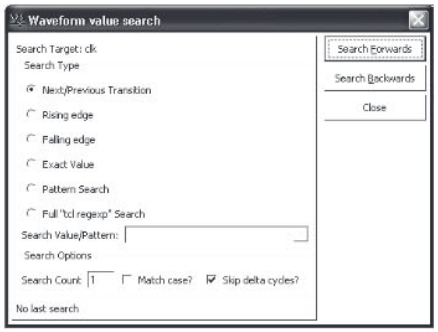


Рис. 9.16. Окно диалога Search for a Waveform Value

Используя кнопки и, можно перейти к моментам следующей или предыдущей установки новых значений для сигналов проекта без использования окна диалога на рис. 9.16.

9.7. Просмотр кода в интерфейсе sonata (Sonata Code Coverage)

Профессиональная версия интерфейса **Sonata** обладает мощным средством просмотра VHDL-кода. Главным образом это средство служит для оценки эффективности разработанной испытательной программы (Test Bench): в какой степени она «покрывает» (проверяет) VHDL-код на уровне регистровых передач (RTL).

Инструмент просмотра (**Coverage Tool**) был разработан для двух наиболее используемых моделей.

1. Использование одиночной испытательной программы (**Single Test Bench Usage**). В этом случае разработчик испытательной программы, предназначенной для определенного проверяемого объекта (**Device Under Test**), может проверить, насколько эффективна эта испытательная программа. При использовании этой модели созданная ранее база данных результатов просмотра предыдущего кода (**Code-Coverage Database**) стирается.
2. Использование нескольких испытательных программ (**Multiple Test Bench Usage**). В этом случае разработчик множества испытательных программ (даже сотен или тысяч), предназначенных для оценки различных функциональных аспектов определенного проверяемого объекта (**Device Under Test**), может оценить суммарное покрытие кода этим множеством испытательных программ. В этом случае все выполняющиеся процессы моделирования уточняют состояние одной и той же базы данных **Code-Coverage Database**.

Просмотр кода выполняется путем выбора опции **-coverage** для симулятора **vhdle**.

Существует три пути анализа данных, собранных в процессе моделирования:

1) использование интерфейса **Sonata** для этой цели;

2) конвертация базы данных в текстовый формат для дальнейшей обработки;

3) просмотр результатов в HTML-файле (в основном для презентаций и печати).

Для загрузки базы данных **Code-Coverage Database** в интерфейс **Sonata** необходимо воспользоваться меню **File -> Open Coverage Database...** При этом мы увидим карту просмотра (**Coverage Map**) и общую статистику на закладке **Code Coverage** консоли (рис. 9.17).

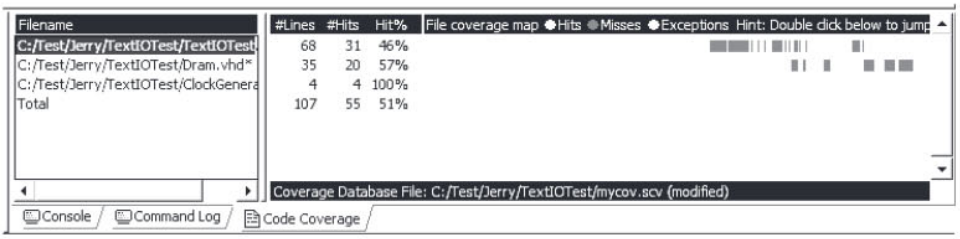


Рис. 9.17. Малая база данных просмотра кода на закладке Code Coverage

На рис. 9.17 справа показаны результаты просмотра для каждого файла. Каждая строка таблицы содержит:

- число строк кода в файле;
- число строк кода, которые были активизированы в ходе процесса моделирования, и их процентное выражение.

Заметим, что числа в колонке **#Lines** не включают строки комментариев, пустые строки или строки деклараций. Они являются числами операторов, которые были выполнены в процессе моделирования. В колонке **#Hits** данные о покрытии кода получают графическое выражение (зеленым цветом выражается количество активизированных операторов-**hits**, красным — количество неактивизированных операторов-**misses**; желтым — количество строк кода исключительных ситуаций (**exceptions**).

Двойной щелчок на правой стороне карты просмотра кода **Coverage Map** приводит к появлению окна текстового редактора (рис. 9.18), отображающего информацию о покрытии кода из базы данных **Coverage Database**.

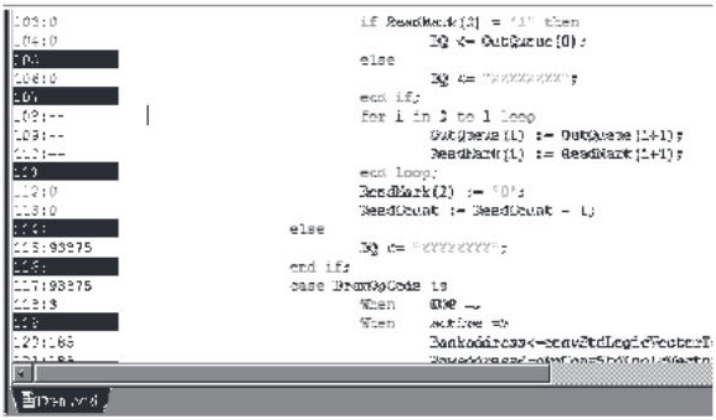


Рис. 9.18. Окно текстового редактора с информацией о покрытии кода (Coverage Data)

Для получения инфомации о покрытии кода в текстовой форме (human readable format(text)) необходимо воспользоваться пунктом контекстного меню **Generate text report file...** В окне **Code Coverage**. После генерации такого отчета (**report file**) он автоматически откроется в окне текстового редактора.

Для получения этой же информации в формате HTML необходимо выбрать опцию **Generate Html report file...** В том же окне.

Часть 3. Проектирование цифровых систем на VHDL

Глава 10. Проектирование комбинационных схем

10.1. Мультиплексоры и другие базовые комбинационные схемы на VHDL

Наиболее распространенными способами задания логических функций, которые являются математическими моделями комбинационных схем, считаются [7р, 2р]:

- алгебраическое представление;
- табличное представление;
- представление через бинарную декомпозицию (в пределе- в виде двоичного дерева решений);
- декомпозиция в априорно заданном базисе функций меньшего числа аргументов.

Здесь мы рассмотрим лишь первые два способа представления.

Реализация комбинационных схем на основе алгебраического представления осуществляется [7р. 2р]:

- с использованием условных и безусловных параллельных операторов присваивания;
- с использованием последовательной формы описания функционирования с помощью операторов процессов (при этом все входные сигналы комбинационной схемы включаются в списки чувствительности соответствующих процессов (или в эквивалентные им операторы ожидания wait) для того, чтобы любое изменение на входе комбинационной схемы инициировало бы исполнение соответствующего оператора присваивания.

На рис. 10.1 приведены шесть примеров описания мультиплексора на VHDL. Этот мультиплексор имеет 4 входа, один выход и управляющий сигнал. Соответствующим образом описаны интерфейсы данного мультиплексора (MUX4_1, MUX4_2, MUX 4_3):

XO, X1, X2, X3 — входы мультиплексора;
Y — выход мультиплексора;
SEL — управляющий сигнал.

```
-------------------------------------------------------
-- 4-INPUT MULTIPLEXER EXAMPLE1:BEHAVIORAL DESCRIPTION
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX4_1 IS
  PORT(SEL          : IN  INTEGER;
       X0,X1,X2,X3 : IN  STD_ULOGIC;
       Y           : OUT STD_ULOGIC);
END MUX4_1;
ARCHITECTURE ARCH1_MUX4_1 OF MUX4_1 IS
```

```
BEGIN
  MULTIPLEXING : PROCESS(SEL,X0,X1,X2,X3)
                 BEGIN
                   CASE SEL IS
                     WHEN 0 =>
                        Y <= X0;
                     WHEN 1 =>
                        Y <= X1;
                     WHEN 2 =>
                        Y <= X2;
                     WHEN 3 =>
                        Y <= X3;
                     WHEN OTHERS =>
                        Y <= 'X';
                   END CASE;
                 END PROCESS;
END ARCH1_MUX4_1;
-------------------------------------------------------
-- 4-INPUT MULTIPLEXER EXAMPLE2:BEHAVIORAL DESCRIPTION
-------------------------------------------------------
ARCHITECTURE ARCH2_MUX4_1 OF MUX4_1 IS
  CONSTANT PROPAGATION_DELAY : TIME:=5 NS;
BEGIN
  MULTIPLEXING : PROCESS
                 BEGIN
                   CASE SEL IS
                     WHEN 0 =>
                        Y <= X0 AFTER PROPAGATION_DELAY;
                        WAIT ON SEL,X0;
                     WHEN 1 =>
                        Y <= X1 AFTER PROPAGATION_DELAY;
                        WAIT ON SEL,X1;
                     WHEN 2 =>
                        Y <= X2 AFTER PROPAGATION_DELAY;
                        WAIT ON SEL,X2;
                     WHEN 3 =>
                        Y <= X3 AFTER PROPAGATION_DELAY;
                        WAIT ON SEL,X3;
                     WHEN OTHERS =>
                        Y <= 'X';
                   END CASE;
                 END PROCESS;
END ARCH2_MUX4_1;
-------------------------------------------------------
-- 4-INPUT MULTIPLEXER EXAMPLE3:BEHAVIORAL DESCRIPTION
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX4_2 IS
  PORT(SEL         : IN  BIT_VECTOR(1 DOWNTO 0);
```

```
       X0,X1,X2,X3 : IN  STD_ULOGIC;
       Y           : OUT STD_ULOGIC);
END MUX4_2;
ARCHITECTURE ARCH1_MUX4_2 OF MUX4_2 IS
BEGIN
  MULTIPLEXING : PROCESS
                 BEGIN
                   IF    SEL="00" THEN
                     Y <= X0;
                   ELSIF SEL="01" THEN
                     Y <= X1;
                   ELSIF SEL="10" THEN
                     Y <= X2;
                   ELSIF SEL="11" THEN
                     Y <= X3;
                   ELSE
                     Y <= 'X';
                   END IF;
                   WAIT ON X0,X1,X2,X3,SEL;
                 END PROCESS;
END ARCH1_MUX4_2;
-------------------------------------------------------
-- 4-INPUT MULTIPLEXER EXAMPLE4:DATAFLOW DESCRIPTION
-------------------------------------------------------
ARCHITECTURE ARCH2_MUX4_2 OF MUX4_2 IS
BEGIN
  MULTIPLEXING : Y <= X0 WHEN SEL= "00" ELSE
                      X1 WHEN SEL= "01" ELSE
                      X2 WHEN SEL= "10" ELSE
                      X3 WHEN SEL= "11" ELSE
                      'X';
END ARCH2_MUX4_2;
-------------------------------------------------------
-- 4-BIT WIDE MULTIPLEXER EXAMPLE5:DATAFLOW DESCRIPTION
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX4_3 IS
  PORT(SEL         : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
       X0,X1,X2,X3 : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
       Y           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END MUX4_3;

ARCHITECTURE ARCH1_MUX4_3 OF MUX4_3 IS
BEGIN
   Y(3) <=    (X0(3)AND NOT(SEL(1))AND NOT(SEL(0)))
           OR (X1(3)AND NOT(SEL(1))AND SEL(0))
           OR (X2(3)AND SEL(1)AND NOT(SEL(0)))
           OR (X3(3)AND SEL(1)AND SEL(0));
```

```
   Y(2) <=     (X0(2)AND NOT(SEL(1))AND NOT(SEL(0)))
           OR (X1(2)AND NOT(SEL(1))AND SEL(0))
           OR (X2(2)AND SEL(1)AND NOT(SEL(0)))
           OR (X3(2)AND SEL(1)AND SEL(0));

   Y(1) <=     (X0(1)AND NOT(SEL(1))AND NOT(SEL(0)))
           OR (X1(1)AND NOT(SEL(1))AND SEL(0))
           OR (X2(1)AND SEL(1)AND NOT(SEL(0)))
           OR (X3(1)AND SEL(1)AND SEL(0));

   Y(0) <=     (X0(0)AND NOT(SEL(1))AND NOT(SEL(0)))
           OR (X1(0)AND NOT(SEL(1))AND SEL(0))
           OR (X2(0)AND SEL(1)AND NOT(SEL(0)))
           OR (X3(0)AND SEL(1)AND SEL(0));
END ARCH1_MUX4_3;
-------------------------------------------------------
-- 4-INPUT MULTIPLEXER EXAMPLE6:DATAFLOW DESCRIPTION
-------------------------------------------------------
ARCHITECTURE ARCH3_MUX4_2 OF MUX4_2 IS
BEGIN
   WITH SEL SELECT
     Y <= X0 WHEN "00",
          X1 WHEN "01",
          X2 WHEN "10",
          X3 WHEN "11";
END ARCH3_MUX4_2;
```

Рис. 10.1. Модели мультиплексора на VHDL

Архитектурное тело ARCH1_MUX4_1 содержит параллельный оператор процесса с меткой MULTIPLEXING, внутри которого находится последовательный оператор CASE, описывающий логику функционирования мультиплексора. Список чувствительности данного процесса содержит все входные сигналы мультиплексора (включая управляющий сигнал): SEL, X0, X1, X2, X3.

Архитектурное тело ARCH2_MUX4_1 представляет собой практически ту же модель, однако процесс, находящийся в данном архитектурном теле, не содержит списка чувствительности: его заменяют эквивалентные операторы ожидания wait on. Кроме того, данная модель использует задержку, описанную константой PROPAGATION_DELAY.

Архитектурное тело ARCH1_MUX4_2 также содержит параллельный оператор процесса, который, в свою очередь, содержит последовательный оператор IF, описывающий логику функционирования мультиплексора.

Архитектурное тело ARCH2_MUX4_2 содержит параллельный оператор присваивания when...else, а архитектурное тело ARCH3_MUX4_2 — параллельный оператор присваивания по выбору with...select.

И, наконец, архитектурное тело ARCH1_MUX4_3 имеет чисто потоковый стиль (Data Flow).

Табличное представление логической функции реализуется следующими способами [7р, 2р]:

– представлением таблицы истинности логической функции в виде константного битового вектора или массива;

– представлением таблицы истинности функции с помощью последовательных операторов присваивания по выбору или условного присваивания.

Например, можно описать таблицу истинности логической функции трех аргументов следующим образом [2p]:

```
process (X1,X2,X3)
type FUNC_VALUES is array (0 to7) of bit;
variable MEM : FUNC_VALUES := "10110001";
begin
f <= MEM (INTVAL (X1 & X2 & X3)) after total_delay;
end process;
```

Здесь таблица истинности описывается переменной MEM, имеющей тип бинарного массива. Эта переменная получает начальное константное значение, содержащее, в нашем случае функции трех переменных, 8 бит. Ключ к нужному элементу массива значений функции создается с помощью операции конкатенации, а функция INTVAL является функцией преобразования трехбитовой комбинации аргументов функции в целое число.

Из соображений экономии места мы не будем рассматривать другие базовые комбинационные схемы. Соответствующие примеры легко найти в технической литературе (рис. 10.2).

Комбинационная схема	Номер книги	Страницы
Comparator	66a	p.p. 106,112,114,115,116,526
Comparator	4a	p.p.120,124,131,134,144,146,155,157,164,173, 175,178, 183,187,223,224,320,380,381
Comparator 74LS85	4a	p.p. 376, 535
Decoder	66a	p.p.181, 184
Decoder	4a	p.p. 275, 524
Encoder (Generic Priority)	9a	p. 292
Encoder (Priority)	66a	p.169
Encoder (Priority)	4a	p. 527
Multiplexer 74LS157	4a	p. 536
Multiplexer	66a	p.163, 180
Multiplexer	81a	p. 40
Multiplexer	40a	p.p.184, 190
Multiplexer	4a	p. 43,272,291,296,299,302,303
Дешифратор полный	13p	с. 438
Дешифратор полный 74×138	18p	с.с. 431–436
Компаратор	18p	с.с. 498, 500, 603–606
Мультиплексор	13p	с.с. 434–436
Мультиплексор	18p	с.с. 478–479
Схема проверки на четность	18p	с. 487
Схема проверки на четность 74×280	18p	с. 488

Комбинационная схема	Номер книги	Страницы
Шифратор	18р	с.с. 597–598
Шифратор приоритетный	18р	с.с. 448, 601–602
Шифратор приоритетный	13р	с. 432

Рис. 10.2. VHDL-модели базовых комбинационных схем в технической литературе

Все сказанное выше применимо и к арифметическим цепям, которые зачастую реализуются как чисто комбинационные схемы (см. п. 10.3 и 10.4).

10.2. Арифметико-логическое устройство на VHDL

В качестве примера разработки комбинационной схемы возьмем проектирование простого арифметического устройства на один бит (рис. 10.3) [Tanenbaum A. Structured Computer Organization 4th ed. — Prentice-Hall, 1999].

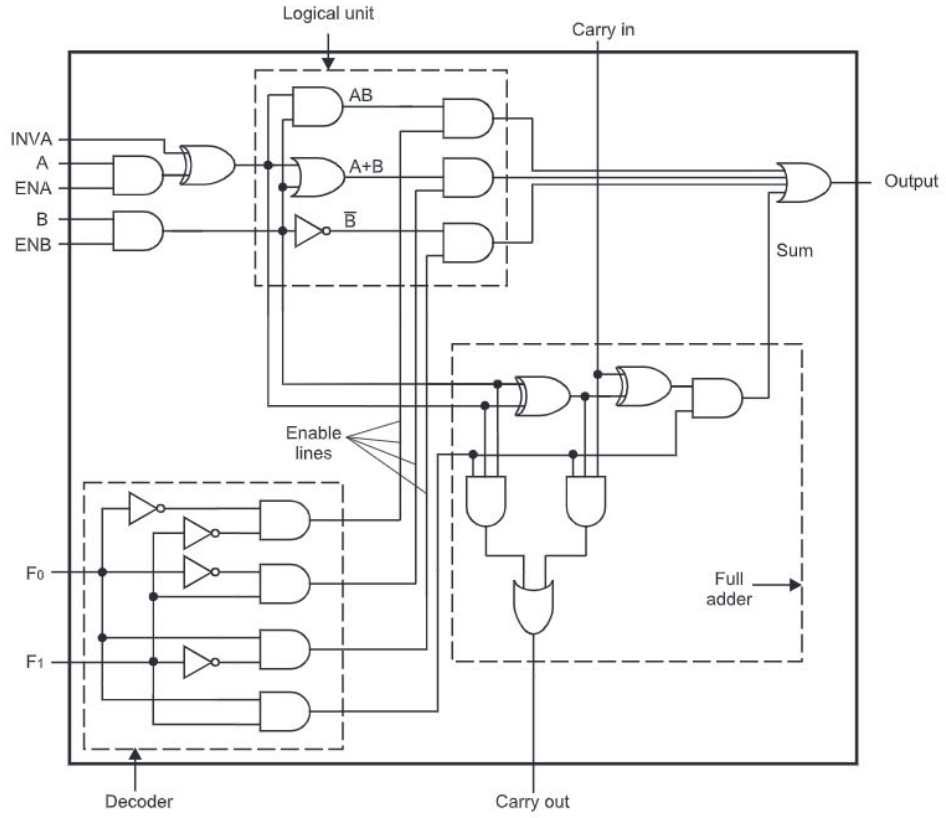


Рис. 10.3. Арифметико-логическое устройство на один бит

Наш проект размещается в четырех файлах (рис. 10.4):
– alu1bit.vhd;

– dc. vhd;
– fa.vhd;
– lu.vhd.

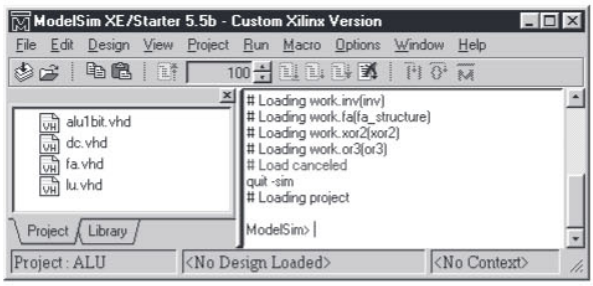


Рис. 10.4. Проект ALU в окне ModelSim

Файл dc. vhd. (рис. 10.5) содержит модель декодера, являющегося составной частью нашего АЛУ. Файл lu.vhd (рис. 10.6) содержит модель элементарного логического устройства — второй составной части АЛУ. Файл fa.vhd уже был рассмотрен нами ранее при обсуждении стилей описания цифровых схем (гл. 3). Он содержит модель полного сумматора — третьей составной части АЛУ.

```
entity dec is
  port( x :in  bit_vector(0 to 1);
        y :out bit_vector(0 to 3));
end dec;
architecture Procedural of dec is
begin
    process(x)
    begin
       case x is
            when "00"=> y <= "1000";
            when "01"=> y <= "0100";
            when "10"=> y <= "0010";
            when "11"=> y <= "0001";
            when others => y <= "0000";
       end case;
    end process;
end Procedural;
-----------------------------------------
ENTITY dec_tb IS
--------------------------
END dec_tb;
ARCHITECTURE dec_tb OF dec_tb IS
        constant off_period : time := 30 ns;
        constant on_period  : time := 20 ns;
        COMPONENT DEC
           PORT(x: in  bit_vector(0 to 1);
                y: out bit_vector(0 to 3));
        END COMPONENT;
        SIGNAL  sx     : bit_vector(0 to 1) ;
        SIGNAL  sy     : bit_vector(0 to 3) ;
BEGIN
  dut : DEC
        PORT MAP(
             x  => sx,
             y  => sy);
```

```
     sx(0) <= '1' after off_period when sx(0) = '0'
              else '0' after on_period;
     sx(1) <= sx(0)'delayed(10 ns);
END dec_tb;
```

Рис. 10.5. Модель декодера (Decoder) — составной части АЛУ

```
library ieee;
-----------------------------------------
entity OR2 is
  port(P1,P2:  in  bit;
       PZ   :  out bit);
end OR2;
architecture or2 of OR2 is
begin
  PZ <= P1 or P2;
end or2;
-----------------------------------------
entity INV is
  port(I :  in  bit;
       O :  out bit);
end INV;
architecture INV of INV is
begin
  O <= not I;
end INV;
-----------------------------------------
entity LU is
  port(A,B,C1,C2,C3     :  in  bit;
       AB,APLUSB,NOTB   :  out bit);
end LU;
-----------------------------------------
architecture LU_STRUCTURE of LU is
   component AND2
       port(P1,P2 : in  bit;
            PZ    : out bit);
   end component;
   component OR2
       port(P1,P2 : in  bit;
            PZ    : out bit);
   end component;
   component INV
       port(I :  in  bit;
            O :  out bit);
   end component;
   signal SAB     : bit;
   signal SAPLUSB : bit;
   signal SNOTB   : bit;
begin
   A1: AND2 port map(A,B,SAB);
   O1:  OR2 port map(A,B,SAPLUSB);
```

```
   I1:  INV port map(B,SNOTB);
   A2: AND2 port map(SAB,C1,AB);
   A3: AND2 port map(SAPLUSB,C2,APLUSB);
   A4: AND2 port map(SNOTB,C3,NOTB);
end LU_STRUCTURE;
-----------------------------------------
```

Рис. 10.6. Модель логического устройства (Logical unit) — составной части АЛУ

Файл alu1bit.vhd (рис. 10.7) содержит структурную модель всего АЛУ. Внутри архитектурного тела ALU1BIT_STRUCTURE описаны как компоненты все составные части АЛУ: DEC, FA, LU, AND2, OR4. Связи между компонентами описываются с помощью внутренних сигналов SAB...SUM и операторов создания экземпляров компонент port map.

```
-----------------------------------------
ENTITY OR4 IS
  PORT(P1,P2,P3,P4 :  IN  bit;
       PZ          :  OUT bit);
END OR4;
ARCHITECTURE or4 OF OR4 IS
BEGIN
  PZ <= P1 OR P2 OR P3 OR P4;
END or4;
-----------------------------------------
ENTITY alu1bit IS
  PORT(A,B,ENA,ENB,CARRY_IN  :  IN  bit;
       F                     :  IN  bit_vector(0 TO 1);
       OUTPUT,CARRY_OUT      :  OUT bit);
END alu1bit;
-----------------------------------------
ARCHITECTURE ALU1BIT_STRUCTURE OF alu1bit IS
   COMPONENT DEC
       PORT(x : IN  bit_vector(0 TO 1);
            y : OUT bit_vector(0 TO 3));
   END COMPONENT;
   COMPONENT FA
       PORT(A,B,CIN         :  IN  bit;
            SUM,COUT        :  OUT bit);
   END COMPONENT;
   COMPONENT LU
       PORT(A,B,C1,C2,C3    :  IN  bit;
            AB,APLUSB,NOTB  :  OUT bit);
   END COMPONENT;
   COMPONENT AND2
       PORT(P1,P2           :  IN  bit;
            PZ              :  OUT bit);
   END COMPONENT;
   COMPONENT OR4
       PORT(P1,P2,P3,P4     :  IN  bit;
            PZ              :  OUT bit);
```

```
   END COMPONENT;
   SIGNAL SAB               : bit;
   SIGNAL SAPLUSB           : bit;
   SIGNAL SNOTB             : bit;
   SIGNAL SA                : bit;
   SIGNAL SB                : bit;
   SIGNAL SC1,SC2,SC3,SC4   : bit;
   SIGNAL SSUM,SUM          : bit;
BEGIN
   A1: AND2 PORT MAP(A,ENA,SA);
   A2: AND2 PORT MAP(B,ENB,SB);
   A3: AND2 PORT MAP(SSUM,SC4,SUM);
   O1:  OR4 PORT MAP(SAB,SAPLUSB,SNOTB,SUM,OUTPUT);
 DEC1:  DEC PORT MAP(F,y(0)=>SC1,y(1)=>SC2,y(2)=>SC3,y(3)=>SC4);
  LU1:   LU PORT MAP(SA,SB,SC1,SC2,SC3,SAB,SAPLUSB,SNOTB);
  FA1:   FA PORT MAP(SA,SB,CARRY_IN,SSUM,CARRY_OUT);
END ALU1BIT_STRUCTURE;
-----------------------------------------
ENTITY alu1bit_tb IS
---------------------------
END alu1bit_tb;
ARCHITECTURE alu1bit_tb OF alu1bit_tb IS
        CONSTANT off_period : time := 30 ns;
        CONSTANT on_period  : time := 20 ns;
        CONSTANT off_period1 : time := 60 ns;
        CONSTANT on_period1  : time := 40 ns;
        COMPONENT alu1bit
             PORT(A,B,ENA,ENB,CARRY_IN    :  IN  bit;
                  F                       :  IN  bit_vector(0 TO
1);
                  OUTPUT,CARRY_OUT        :  OUT bit);
        END COMPONENT;
        SIGNAL  NA,NB,NENA,NENB,NCARRY_IN,NOUTPUT,NCARRY_OUT : bit
;
        SIGNAL  NF                      : bit_vector(0 TO 1) ;
BEGIN
  dut : alu1bit
        PORT MAP(
             A         => NA,
             B         => NB,
             ENA       => NENA,
             ENB       => NENB,
             F         => NF,
             CARRY_IN  => NCARRY_IN,
             OUTPUT    => NOUTPUT,
             CARRY_OUT => NCARRY_OUT);
     NA <= '1' AFTER off_period WHEN NA = '0'
           ELSE '0' AFTER on_period;
     NB <= NA'delayed(10 ns);
```

```
    NENA<= '1' AFTER 10 ns;
    NENB<= '1' AFTER 10 ns;
    NF(0) <= '1' AFTER off_period1 WHEN NF(0) = '0'
             ELSE '0' AFTER on_period1;
    NF(1) <= NF(0)'delayed(10 ns);
    NCARRY_IN <= '0' AFTER off_period1 WHEN NCARRY_IN = '0'
                 ELSE '1' AFTER on_period1;
END alu1bit_tb;
```

Рис. 10.7. Структурная модель АЛУ

Для моделирования АЛУ написана испытательная программа alu1bit_tb, находящаяся в этом же файле. Входные последовательности для объекта моделирования (dut: alu1bit) задаются с помощью условных параллельных операторов NA... NCARRY_IN. При этом временные задержки описываются с помощью констакта on_period и off_period, а также с помощью атрибута delayed.

После загрузки этой испытательной программы (рис. 10.8) мы получаем в окне Wave системы моделирования ModelSim следующие результаты (рис. 10.9).

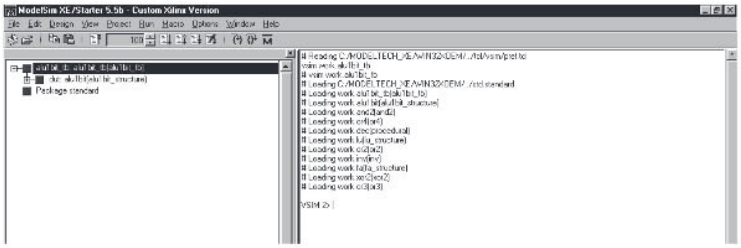


Рис. 10.8. Загрузка испытательной программы для модели АЛУ

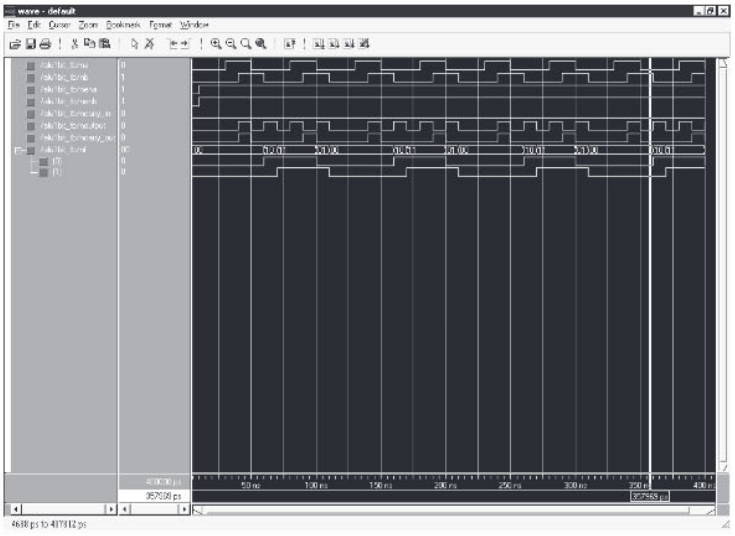


Рис. 10.9. Результаты моделирования АЛУ

О других моделях АЛУ можно получить представление из работ, приведенных на рис. 10.10.

Арифметическая комбинационная схема	Номер книги	Страницы
Adder (Full)	66a	p.p.511,521,522
Adder (Full)	81a	p.p.33,66,98,120,177
Adder (Full)	40a	p.196
Adder (Full) (74LS283)	4a	p.537
Adder (Full)	4a	p.p.52,63
Adder (Half)	81a	p.p.50,62,106,110,111
ALU	40a	p.p.316-317
ALU	14a	p.p.232-248
ALU	81a	p.107
ALU	66a	p.508
ALU	4a	p.526
Сумматор-вычитатель	18p	с.с.517-518
Сумматор	18p	с.609
Умножитель	18p	с.с.518-527

Рис. 10.10. VHDL-модели арифметических цепей в технической литературе

10.3. Сумматоры и вычитатели на VHDL

На рис. 10.11 приведена схема параллельного «волнового» сумматора с последовательным формированием переноса из разряда в разряд (Parallel Ripple-Carry Adder). Его модель приведена на рис. 10.12.

```
ENTITY full_adder IS
  PORT (cin  : IN  bit;
        a    : IN  bit;
        b    : IN  bit;
        sum  : OUT bit;
        cout : OUT bit);
END full_adder;
ARCHITECTURE FA of full_adder IS
BEGIN
     sum  <= a XOR b XOR cin;
     cout <= ((a OR b) AND cin) OR (a AND b);
END FA;
ENTITY ripple_adder_8 IS
  PORT (cin                    : IN  bit;
        a0,a1,a2,a3,a4,a5,a6,a7 : IN  bit;
```

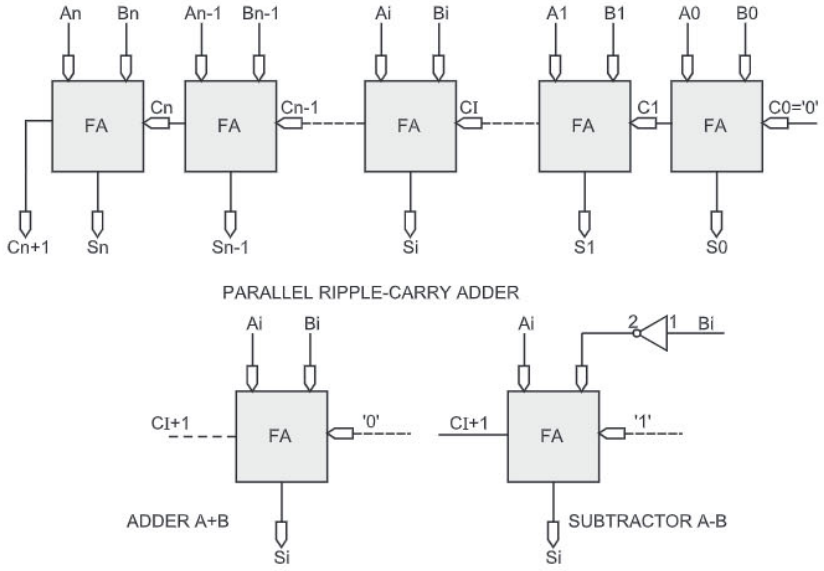


Рис. 10.11. Схема параллельного «волнового» сумматора

```
        b0,b1,b2,b3,b4,b5,b6,b7 : IN  bit;
        s0,s1,s2,s3,s4,s5,s6,s7 : OUT bit;
        cout                    : OUT bit);
END ripple_adder_8;
ARCHITECTURE SA of ripple_adder_8 IS
  COMPONENT full_adder
    PORT (cin  : IN  bit;
          a    : IN  bit;
          b    : IN  bit;
          sum  : OUT bit;
          cout : OUT bit);
    END COMPONENT;
  SIGNAL c1,c2,c3,c4,c5,c6,c7: bit;
BEGIN
  fa1: full_adder
       PORT MAP(cin,a0, b0, s0, c1  );
  fa2: full_adder
       PORT MAP(c1 ,a1, b1, s1, c2  );
  fa3: full_adder
       PORT MAP(c2 ,a2, b2, s2, c3  );
  fa4: full_adder
       PORT MAP(c3, a3, b3, s3, c4  );
  fa5: full_adder
       PORT MAP(c4, a4, b4, s4, c5  );
  fa6: full_adder
       PORT MAP(c5, a5, b5, s5, c6  );
  fa7: full_adder
       PORT MAP(c6, a6, b6, s6, c7  );
```

```
  fa8: full_adder
       PORT MAP(c7, a7, b7, s7, cout);
END SA;
```

Рис. 10.12. Структурная модель параллельного волнового сумматора с последовательным формированием переноса

Структурное архитектурное тело SA данного сумматора на 8 разрядов ripple_adder_8 содержит компонент full_adder, являющийся полным одноразрядным сумматором. С помощью операторов port map (Component Instatiation Statement) реализуются 8 экземпляров базового компонента, должным образом соединенных между собой.

Из рис. 10.11 видно, что одноразрядный сумматор может работать и в режиме вычитания. Следуя этому, можно построить арифметическое устройство, работающее в режиме сложения и вычитания (в соответствии со значением управляющего сигнала F) (Parallel Adder/Subtractor) (рис. 10.13).

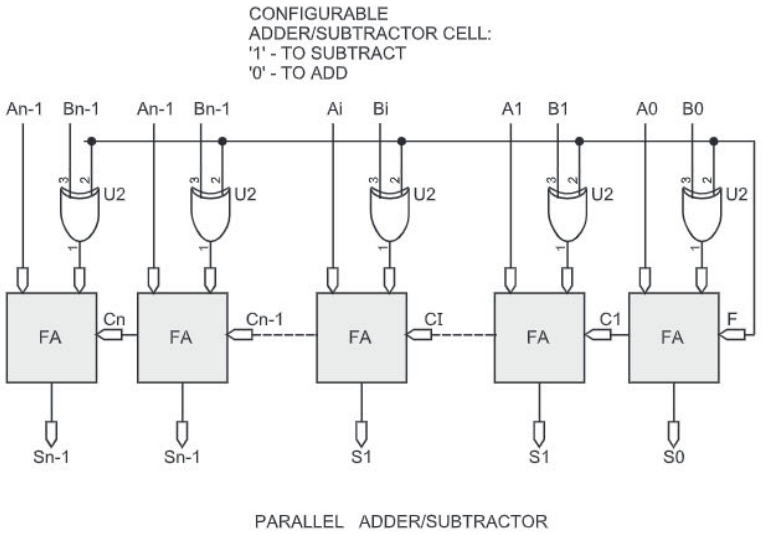


Рис. 10.13. Параллельный сумматор/вычитатель с последовательным формированием переноса

Структурная модель этого сумматора — вычитателя — приведена на рис. 10.14. Вновь базовым компонентом является одноразрядный полный сумматор. Моделями же вентилей XOR являются 8 параллельных операторов присваивания для сигналов sb0…sb7.

```
ENTITY full_adder IS
  PORT (cin  : IN  bit;
        a    : IN  bit;
        b    : IN  bit;
        sum  : OUT bit;
        cout : OUT bit);
END full_adder;
ARCHITECTURE FA of full_adder IS
BEGIN
```

```
     sum  <= a XOR b XOR cin;
     cout <= ((a OR b) AND cin) OR (a AND b);
END FA;
ENTITY ripple_adder_8 IS
  PORT (control                  : IN  bit;
        a0,a1,a2,a3,a4,a5,a6,a7 : IN  bit;
        b0,b1,b2,b3,b4,b5,b6,b7 : IN  bit;
        s0,s1,s2,s3,s4,s5,s6,s7 : OUT bit;
        cout                     : OUT bit);
END ripple_adder_8;
ARCHITECTURE SA of ripple_adder_8 IS
  COMPONENT full_adder
    PORT (cin  : IN  bit;
          a    : IN  bit;
          b    : IN  bit;
          sum  : OUT bit;
          cout : OUT bit);
    END COMPONENT;
  SIGNAL c1,c2,c3,c4,c5,c6,c7            : bit;
  SIGNAL sb0,sb1,sb2,sb3,sb4,sb5,sb6,sb7 : bit;
BEGIN
  sb0 <= b0 XOR control;
  sb1 <= b1 XOR control;
  sb2 <= b2 XOR control;
  sb3 <= b3 XOR control;
  sb4 <= b4 XOR control;
  sb5 <= b5 XOR control;
  sb6 <= b6 XOR control;
  sb7 <= b7 XOR control;
  fa1: full_adder
       PORT MAP(control, a0, sb0, s0, c1  );
  fa2: full_adder
       PORT MAP(c1     , a1, sb1, s1, c2  );
  fa3: full_adder
       PORT MAP(c2     , a2, sb2, s2, c3  );
  fa4: full_adder
       PORT MAP(c3     , a3, sb3, s3, c4  );
  fa5: full_adder
       PORT MAP(c4     , a4, sb4, s4, c5  );
  fa6: full_adder
       PORT MAP(c5     , a5, sb5, s5, c6  );
  fa7: full_adder
       PORT MAP(c6     , a6, sb6, s6, c7  );
  fa8: full_adder
       PORT MAP(c7     , a7, sb7, s7, cout);
      END SA;
```

Рис. 10.14. Структурная модель сумматора-вычитателя на 8 разрядов

Схема более быстродействующего сумматора (с параллельным формированием переноса — Carry Look Ahead Adder) на четыре разряда приведена на рис. 10.15.

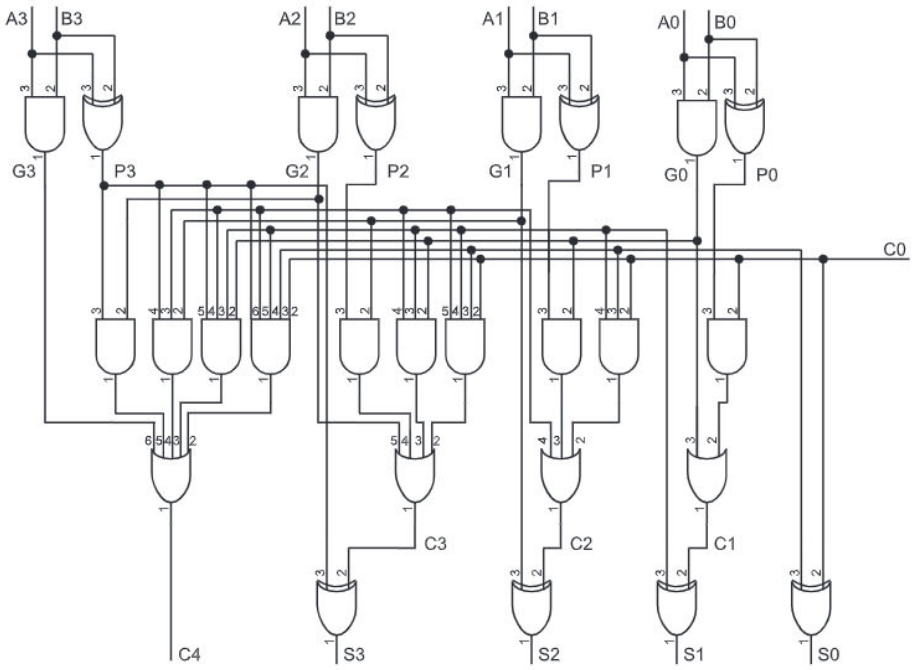


Рис. 10.15. Сумматор с параллельным формированием переноса (Carry Look Ahead Adder) на 4 разряда

Здесь для ускоренного формирования переноса используются две специальные функции:

1. Generate Function;
2. Propagate Function.

Обозначим эти две функции буквами G и P соответственно. Функция генерации (G) принимает значение 1, когда определенный разряд генерирует сигнал переноса (carry_out = 1) независимо от значения сигнала переноса (carry_in) на входе данного разряда. Поэтому для *i*-го разряда такого сумматора можно записать:

$$G_i = a_i \cdot b_i.$$

Функция распространения (P) принимает значение 1 в случае, когда сигнал переноса на выходе данного разряда (carry_out) устанавливается в 1 при carry_in = 1. Отсюда следует, что

$$P_i = a_i \vee b_i$$

Теперь мы можем записать выражение для переноса на входе *i*-го разряда

$$C_i = G_i \vee P_i \cdot C_{i-1}.$$

Отсюда видно, что сигнал переноса на входе *i*-го разряда устанавливается в одном из следующих случаев:

– $G_i = 1$ (т. е. сигнал переноса генерируется в самом *i*-м разряде) или

– $P_i = 1$ (на входах a_i и b_i лишь одна единица) и $C_i = 1$(сигнал переноса на входе *i*-го разряда устанавливается в 1).

С помощью последнего выражения можно получить формулы для вычисления сигналов переноса C1, C2, C3, C4:

C1 = G1 v CO · P1,

C2 = G1 v P1 · (GO v PO · CO) = G1 v GO · P1 v CO · P1 · P0;

C3 = G2 v P2 (G1 v GO · P1 v CO · P1 · P0) = G2 v G1 · P2 v GO · P1 · P2 v CO · P2 · P1 · P0;

C4 = G3 v P3 · (G2 v G1 · P2 v GO · P1 · P2 v CO ·P2 · P1· P0)

= G3 v G2 · P3 v G1 · P3 · P2 v G0 · P3 · P2 · P1 v

v CO · P3 · P2 · P1 · P0.

Согласно этим формулам и строится схема на рис. 10.15.

Две структурные модели такого сумматора приведены на рис. 10.16 и 10.17.

```
ENTITY cla_adder_2 IS
  PORT (cin  : IN  bit;
        a0,a1      : IN  bit;
        b0,b1      : IN  bit;
        sum0,sum1  : OUT bit;
        g,p        : OUT bit);
END cla_adder_2;
ARCHITECTURE arch_cla_adder_2 OF cla_adder_2 IS
BEGIN
   sum0 <= a0 XOR b0 XOR cin;
   sum1 <= a1 XOR b1 XOR ((a0 AND b0)OR
          (a0 AND cin) OR (b0 AND cin));
   g <= (a1 AND b1) OR ((a1 OR b1) AND (a0 AND b0));
   p <= (a1 OR b1) AND (a0 OR b0);
END arch_cla_adder_2;
ENTITY cla_adder_8_2 IS
  PORT (cin                    : IN  bit;
        a0,a1,a2,a3,a4,a5,a6,a7 : IN  bit;
        b0,b1,b2,b3,b4,b5,b6,b7 : IN  bit;
        s0,s1,s2,s3,s4,s5,s6,s7 : OUT bit;
        cout                   : OUT bit);
END cla_adder_8_2;
ARCHITECTURE arch_cla_adder_8_2 OF cla_adder_8_2 IS
        COMPONENT cla_adder_2
           PORT (cin        : IN  bit;
                 a0,a1      : IN  bit;
                 b0,b1      : IN  bit;
                 sum0,sum1  : OUT bit;
                 G,P        : OUT bit);
        END COMPONENT;
        SIGNAL c2, c4, c6     : bit;
        SIGNAL g0, g1, g2, g3 : bit;
        SIGNAL p0, p1, p2, p3 : bit;
BEGIN
```

```
    cla_add1:  cla_adder_2
               PORT MAP (cin, ao, a1, b0, b1, sum0, sum1, g0, p0);
    cla_add2:  cla_adder_2
               PORT MAP (c2 , a2, a3, b2, b3, sum2, sum3, g1, p1);
    cla_add3:  cla_adder_2
               PORT MAP (c4 , a4, a5, b4, b5, sum4, sum5, g2, p2);
    cla_add4:  cla_adder_2
               PORT MAP (c6 , a6, a7, b6, b7, sum6, sum7, g3, p3);
    c2   <= g0 OR (p0 AND cin);
    c4   <= g1 OR (p1 AND g0) OR (p1 AND p0 AND cin);
    c6   <= g2 OR (p2 AND g1) OR (p2 AND p1 AND g0) OR
            (p2 AND p1 AND p0 AND cin);
    cout <= g3 OR (p3 AND g2) OR (p3 AND p2 AND g1) OR
            (p3 AND p2 AND p1 AND g0) OR
            (p3 AND p2 AND p1 AND p0 AND cin);
END arch_cla_adder_8_2;
```

Рис. 10.16. Структурная модель сумматора с параллельным формированием переноса cla_adder_8_2 на восемь разрядов

```
ENTITY cla_adder_3 IS
  PORT (cin  : IN  bit;
        a0,a1,a2        : IN  bit;
        b0,b1,b2        : IN  bit;
        sum0,sum1,sum2  : OUT bit;
        g,p             : OUT bit);
END cla_adder_3;
ARCHITECTURE arch_cla_adder_3 OF cla_adder_3 IS
BEGIN
   sum0 <= a0 XOR b0 XOR cin;
   sum1 <= a1 XOR b1 XOR c1;
   sum2 <= a2 XOR b2 XOR c2;
   g <= (a2 AND b2) OR ((a2 OR b2) AND (a1 AND b1)) OR
        ((a2 OR b2) AND (a1 OR b1) AND (a0 AND b0));
   p <= (a1 OR b1) AND (a0 OR b0);
END arch_cla_adder_3;
ENTITY cla_adder_8_3 IS
  PORT (cin                     : IN  bit;
        a0,a1,a2,a3,a4,a5,a6,a7 : IN  bit;
        b0,b1,b2,b3,b4,b5,b6,b7 : IN  bit;
        s0,s1,s2,s3,s4,s5,s6,s7 : OUT bit;
        cout                    : OUT bit);
END cla_adder_8_3;
ARCHITECTURE arch_cla_adder_8_3 OF cla_adder_8_3 IS
        COMPONENT cla_adder_3
           PORT (cin             : IN  bit;
                 a0,a1,a3        : IN  bit;
                 b0,b1,b3        : IN  bit;
                 sum0,sum1,sum2  : OUT bit;
                 G,P             : OUT bit);
```

```
        END COMPONENT;
        COMPONENT cla_adder_2
           PORT (cin        : IN  bit;
                 a0,a1      : IN  bit;
                 b0,b1      : IN  bit;
                 sum0,sum1  : OUT bit;
                 G,P        : OUT bit);
        END COMPONENT;
        SIGNAL   c3, c6     : bit;
        SIGNAL   g0, g1, g2 : bit;
        SIGNAL   p0, p1, p2 : bit;
BEGIN
    cla_add1:  cla_adder_3
               PORT MAP
(cin,ao,a1,a2,b0,b1,b2,sum0,sum1,sum2,g0,p0);
    cla_add2:  cla_adder_3
               PORT MAP
(c3,a3,a4,a5,b3,b4,b5,sum3,sum4,sum5,g1,p1);
    cla_add3:  cla_adder_2
               PORT MAP (c6,a6,a7,b6,b7,sum6,sum7,g2,p2);
    c3   <= g0 OR (p0 AND cin);
    c6   <= g1 OR (p1 AND g0) OR (p1 AND p0 AND cin);
    cout <= g2 OR (p2 AND g1) OR (p2 AND p1 AND g0) OR
            (p2 AND p1 AND p0 AND cin);
END arch_cla_adder_8_3;
```

Рис. 10.17. Структурная модель сумматора с параллельным формированием переноса cla_adder_8_3 на восемь разрядов

Структурное архитектурное тело arch_cla_adder_8_2 использует для построения сумматора с параллельным формированием переноса на 8 разрядов компонент, являющийся подобным сумматором на два разряда cla_adder_2 (моделью компонента является архитектурное тело arch_cla_adder_2, выполненное в стиле Data Flow).

В свою очередь, архитектурное тело arch_cla_adder8_3 использует для построения структурной модели сумматора с параллельным формированием переноса на 8 разрядов два базовых компонента:

1. cla_adder_3: трехзарядный подобный сумматор;
2. cla_adder_2: двухзарядный подобный сумматор.

Другие модели разнообразных симуляторов можно почерпнуть из источников, приведенных на рис. 10.10.

10.4. Умножители на VHDL

Принцип выполнения операции бинарного умножения (Binary Multiplication) изложен на рис. 10.18. Рассмотрим две технические реализации этой схемы вычислений.

На рис. 10.19 приведена блок-схема умножителя, называющегося Shift And Add Multiplier. Этот умножитель с помощью нескольких регистров (в том числе реги-

стров сдвига), параллельного сумматора и линейки вентилей выполняет ряд последовательных операций сдвига и сложений для достижения конечного результата. Рисунок 10.20 поясняет этот процесс.

VHDL-модель данного умножителя приведена на рис. 10.21. Архитектурное тело arch_shift_add_multiplier является поведенческой моделью умножителя, опе-

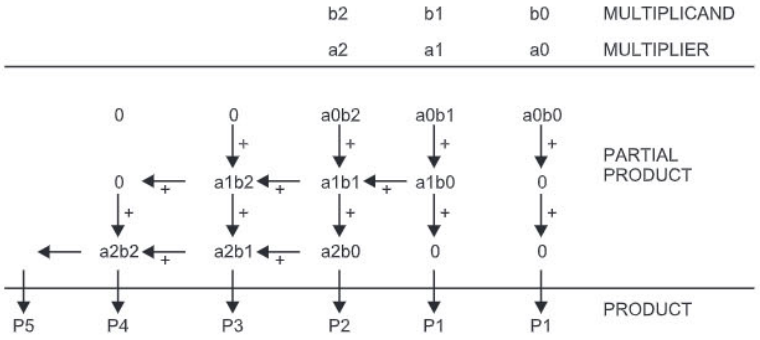


Рис. 10.18. Схема вычислений при выполнении операции бинарного умножения

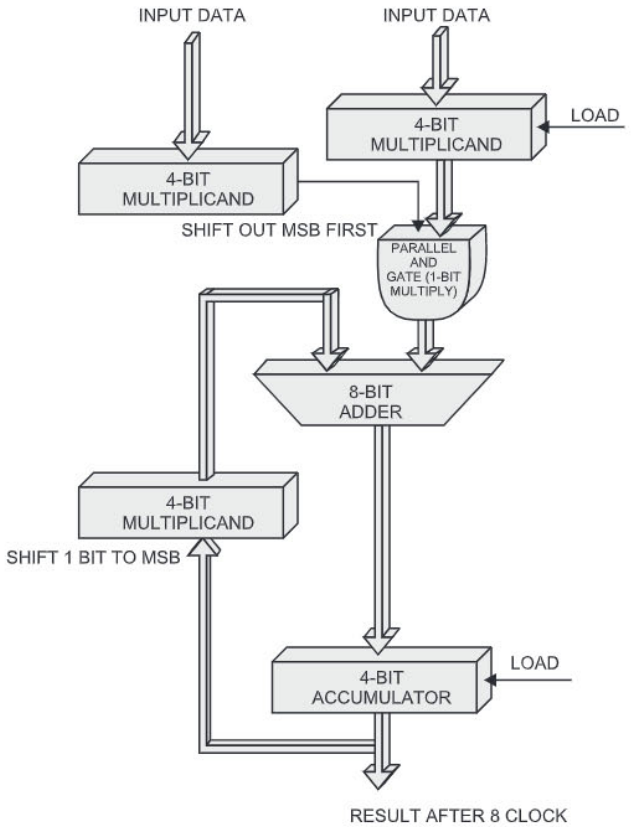


Рис. 10.19. Блок-схема умножителя Shift and Add Multiplier

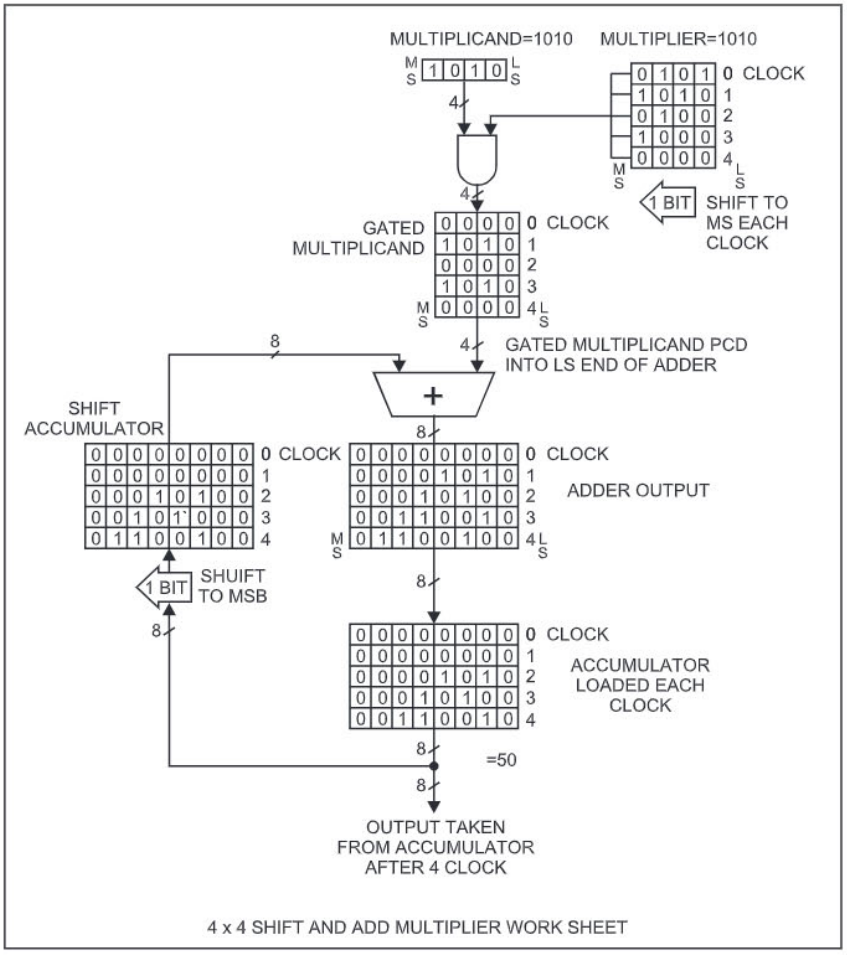


Рис. 10.20. Функционирование умножителя 4x4 Shift and Add Multiplier

рандами которого являются два 4-разрядных бинарных числа (4-bit Multiplier и 4-bit Multiplicand). Результатом работы этого умножителя через 4 такта является 8-разрядное бинарное число в аккумуляторе.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.std_arith.ALL;
ENTITY shift_add_multiplier IS
   PORT ( rd, wr, cs, clk: IN    std_logic;
          databus        : INOUT std_logic_vector(7 DOWNTO 0));
END shift_add_multiplier;
ARCHITECTURE arch_shift_add_multiplier OF shift_add_multiplier IS

   TYPE   statename IS (idle, shad1, shad2, shad3, shad4);
   SIGNAL present_state, next_state   : statename;
```

```
   SIGNAL multiplicand, multiplier    : std_logic_vector(3 DOWNTO
0);
   SIGNAL sum                         : std_logic_vector(8 DOWNTO
0);
SIGNAL accumulator, gated_multiplicand: std_logic_vector(7 DOWNTO
0);
BEGIN
-- COMBINATIONAL DATA PATH
gated_multiplicand(3 downto 0) <= multiplicand WHEN
                                  (multiplier(3)='1') ELSE "0000";
sum <= gated_multiplicand + (accumulator & '0');
-- SHIFT ACCUMULATOR AND ADD PARTIAL PRODUCT
statemachine: PROCESS(present_state, wr, cs)
              BEGIN
                CASE present_state IS
                     WHEN idle =>
                          IF (wr='0' AND cs='0') THEN
                             next_state <= shad1;
                          ELSE next_state <= idle;
                          END IF;
                     WHEN shad1 =>
                          next_state <= shad2;
                     WHEN shad2 =>
                          next_state <= shad3;
                     WHEN shad3 =>
                          next_state <= shad4;
                     WHEN shad4 =>
                          next_state <= idle;
                END CASE;
              END PROCESS;
synchronous: PROCESS(clk)
             BEGIN
               IF rising_edge(clk) THEN
                  IF (wr='0' cs='0') THEN            -- LOAD COM-
MAND
                     accumulator  <= (OTHERS => '0');
                     multiplicand <= databus(7 DOWNTO 0);
                     multiplier   <= databus(3 DOWNTO 0);
                  ELSIF (present_state /= idle) THEN -- SHIFT AND
ADD
                     accumulator  <= sum(7 DOWNTO 0);
                     multiplier   <= multiplier(2 DOWNTO 0)
                                     & multiplier(3);
                  END IF;
END PROCESS;
tristates:   PROCESS(cs, rd, accumulator)
             BEGIN
               IF (cs='0' AND rd='0') THEN           -- OUTPUT DATA
                   databus <= accumulator;
               ELSE
```

```
                databus <= (OTHERS => 'Z');

            END IF;
        END PROCESS;

END arch_shift_add_multiplier;
```

Рис. 10.21. Поведенческая модель умножителя 4x4 Shift and Add Multiplier

В данном архитектурном теле находятся три процесса. Процесс с меткой synchronous описывает четыре последовательные операции сдвига- сложения, выполняемые сумматором (8-bit Adder), сдвиговым регистром (8-bit Shifter) и аккумулятором (8-bit Accumulator). Каждая такая операция соответствует одному из четырех состояний (shad1, shad2, shad3, shad4), вырабатываемых процессом statemachine. Процесс tristate устанавливает результат вычисления на выходе умножителя при определенном сочетании управляющих сигналов.

Два параллельных оператора присваивания для сигналов gated_multiplicand и sum описывают прохождение оператора 4-bit Multiplicand через линейку вентилей И и сложения сдвинутого содержимого аккумулятора (Shift accumulator) с очередным частичным произведением (Partial Product).

Другая схема реализации операции бинарного умножения приведена на рис. 10.22. Отсюда вытекает техническая реализация умножителя в виде однородного двумерного массива ячеек (рис. 10.23).

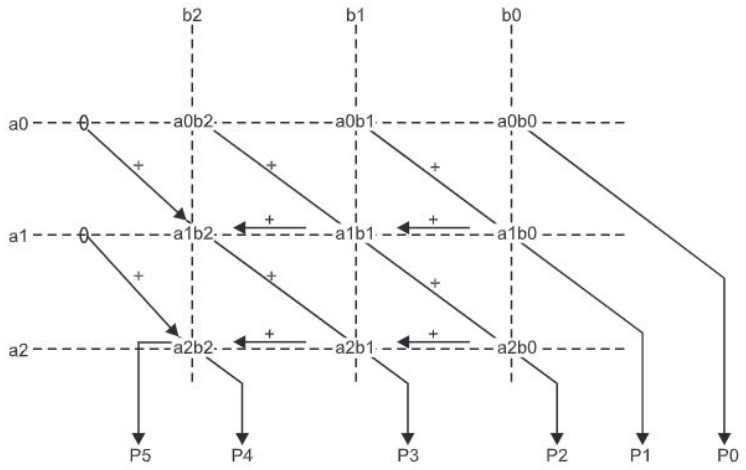


Рис. 10.22. Альтернативная реализация операции бинарного умножения

Структура типовой ячейки привидена на рис. 10.24. Отсюда видно, что типовая ячейка содержит полный одноразрядный сумматор и логический вентиль И.

Модель такого умножителя приведена на рис. 10.25. Архитектурное тело cell_struct является структурной моделью типовой ячейки, использующей в качестве базового компонента полный однозарядный сумматор (full adder). Вентиль И описывается оператором присваивания для сигнала x.

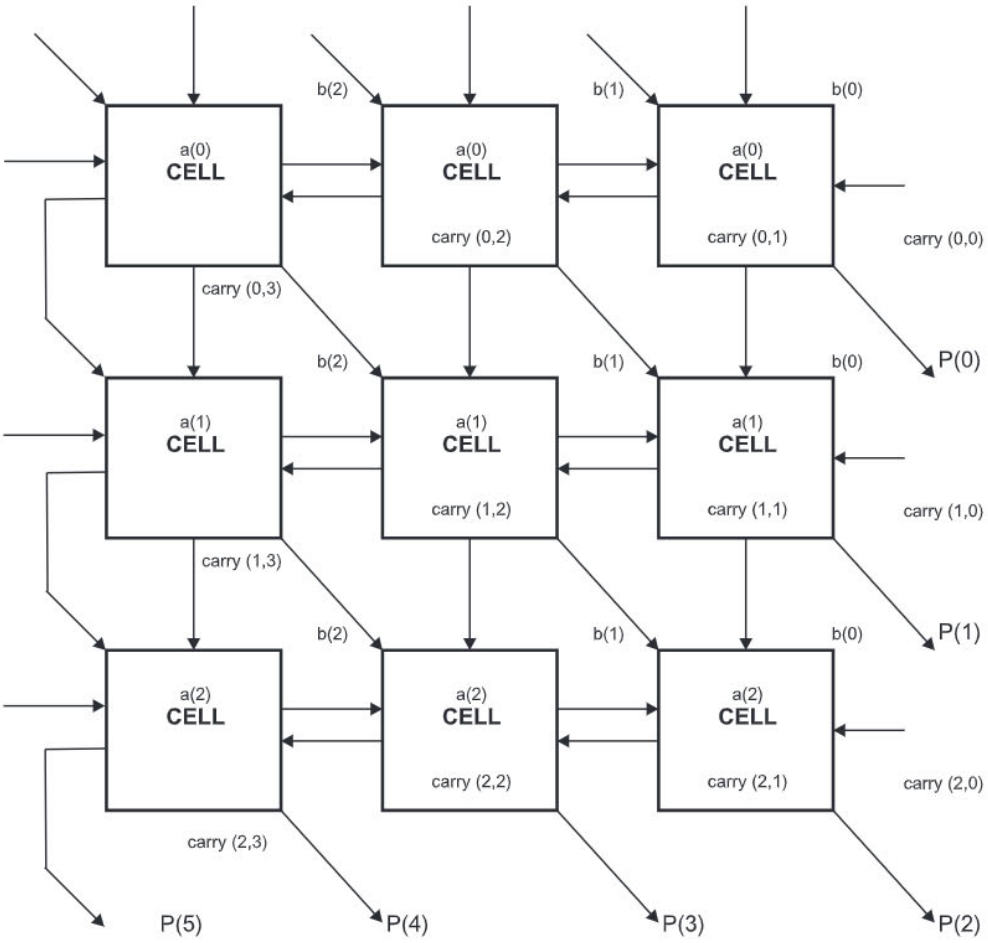


Рис. 10.23. Реализация умножителя в виде двумерного массива ячеек (3×3 Cellular Multiplier)

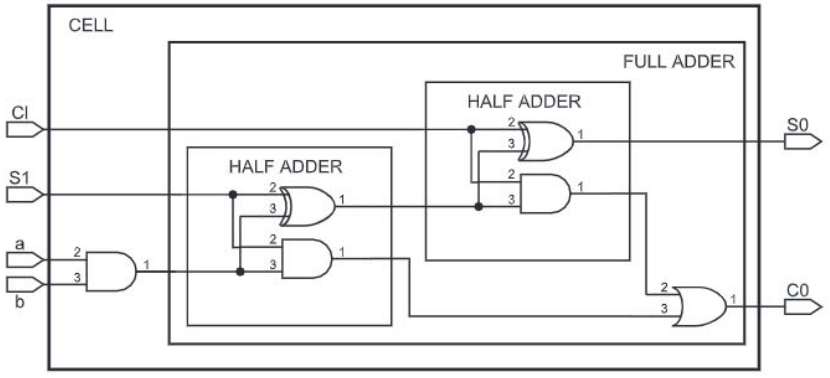


Рис. 10.24. Структура типовой ячейки распределенного умножителя (Multiplier Cell Construction)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY half_adder IS
  PORT(a,b    : IN std_logic;
       s,cout : OUT std_logic);
END half_adder;
ARCHITECTURE HA OF half_adder IS
BEGIN
      cout <= a AND b;
      s    <= a XOR b;
END HA;
----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY full_adder IS
  PORT(a,b,cin  : IN std_logic;
       s,cout   : OUT std_logic);
END full_adder;
ARCHITECTURE FA OF full_adder IS
  SIGNAL ps,c1,c2 : std_logic;

  COMPONENT half_adder IS
    PORT(a,b  : IN std_logic;
       s,cout : OUT std_logic);
  END COMPONENT;
BEGIN
      HA1: half_adder
           PORT MAP(a,b,ps,c1);
      HA2: half_adder
           PORT MAP(ps,cin,s,c2);
      cout <= c1 OR c2;
END FA;
----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY cell IS
  PORT(a,b,cin,sin  : IN  std_logic;
       sout,cout    : OUT std_logic);
END cell;
ARCHITECTURE cell_struct OF cell IS
    SIGNAL x std_logic;

    COMPONENT full_adder IS
      PORT(a,b,sin    : IN std_logic;
           s,cout     : OUT std_logic);
    END COMPONENT;

BEGIN
    FA1: full_adder
         PORT MAP (sin,x,cin,sout,cout);
```

```
        x <= a and b;

END cell_struct;
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY parameterised_cellular_multiplier IS
  GENERIC(N : natural RANGE 32 DOWNTO 2);
  PORT(A,B  : IN  std_logic_vector(N-1 DOWNTO 0);
       P    : OUT std_logic_vector(2*N-1 DOWNTO 0));
END parameterised_cellular_multiplier;
ARCHITECTURE multiplier_struct OF parameterised_cellular_multiplier
IS

  TYPE   sum_array   IS ARRAY(N   DOWNTO 0, N DOWNTO 0) OF std_lo-
gic;
  TYPE   carry_array IS ARRAY(N-1 DOWNTO 0, N DOWNTO 0) OF std_lo-
gic;
  SIGNAL sum  : sum_array;
  SIGNAL carry: carry_array;

  COMPONENT cell IS
    PORT(a,b,cin,sin  : IN  std_logic;
         sout,cout    : OUT std_logic);
  END COMPONENT;

BEGIN
  ROW: FOR I IN N-1 DOWNTO 0 GENERATE
    COL:   FOR J IN N-1 DOWNTO 0 GENERATE
           EX_CELL   : cell
                       PORT MAP(sum(I,J+1),carry(I,J),a(),b(),
                                sum(I+1,J),carry(I,J+1));

           TOP_ROW   : IF I=0 GENERATE
                          sum(I,J+1) <= '0';
           END TOP_ROW;

           BOTTOM_ROW: IF I=N-1 GENERATE
                          p(N+J) <= sum(I+1,J+1);
           END BOTTOM_ROW;

           LEFT_COL  : IF J=N-1 GENERATE
                          sum(I+1,J+1) <= carry(I,J+1);
           END LEFT_COL;

           RIGHT_COL : IF J=0 GENERATE
                          carry(I,J) <= '0';
                          p(I)     <= sum(I+1,J);
           END RIGHT_COL;
    END RIGHT_COL;
```

```
  END RIGHT_COL;

END multiplier_struct;
```

Рис. 10.25. Структурная модель умножителя NxN Cellular Multiplier

В описании интерфейса нашего умножителя (entity parameterised_cellular_multiplier) используется параметр N, являющийся натуральным числом в интервале от 0 до 32. Таким образом, число ячеек умножителя - NxN.

Сам массив ячеек получается с помощью оператора генерации (Generate Statement).

В заключение следует добавить, что если уж мы во всем полагаемся на инструмент синтеза, достаточно лишь написать для создания умножителя одну строчку кода (рис. 10.26).

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.std_arith.ALL;
ENTITY multiplier IS
   PORT ( a, b     : IN    std_logic_vector(3 DOWNTO 0);
          c        : INOUT std_logic_vector(7 DOWNTO 0));
END multiplier;
ARCHITECTURE multiplier_synth OF multiplier IS
BEGIN
      c <= a * b;

END multiplier_synth;
```

Рис. 10.26. VHDL-код для синтеза схемы умножителя

Глава 11. Проектирование синхронных схем с памятью на VHDL

11.1. Триггеры, регистры и счетчики на VHDL

Характерным свойством триггеров является сохранение текущего состояния при отсутствии определенных сочетаний на управляющих входах вне зависимости от изменения информации на входах данных.

Это свойство описывается на VHDL с помощью параллельных операторов блока или процесса. В первом случае управляющие сигналы включаются в охранное выражение блока, а во втором — в список чувствительности процесса или эквивалентный ему оператор wait.

После соответствующего события на управляющих входах триггер переходит в новое состояние, которое определяется предыдущим состоянием и состоянием входа данных.

Как известно, на практике используются следующие способы управления триггерами [7р, 2р]:

- асинхронное управление;
- синхронное управление, которое, в свою очередь, подразделяется на статическое управление (управление уровнем) и динамическое управление (управление фронтом).

На рис. 11.1 приведены примеры описаний синхронной логики. В примере 1 описывается функционирование D-триггера (или регистра — в зависимости от описаний сигналов Q и DATA_IN в интерфейсе устройства) без управляющего входа сброса с помощью процесса, имеющего список чувствительности. В примере 2, в отличие от предыдущего примера, вместо списка чувствительности используется эквивалентный оператор wait. В примере 3 описывается D-триггер с синхронным сбросом, а в примере 4 — с асинхронным сбросом. Пример 5 показывает, как можно описать с помощью процесса и оператора IF D-триггер с асинхронным сбросом и установкой. В примере 6 вместо входа установки имеется вход разрешения (ENABLE).

```
-------------------------------------------------
-- DESCRIBING SYNCHRONOUS LOGIC USING PROCESSES
-------------------------------------------------
-- NO RESET
-------------------------------------------------
-- EXAMPLE 1
REG16_NO_RESET:
PROCESS(CLOCK)
BEGIN
  IF CLOCK'EVENT AND CLOCK = '1' THEN
    Q <= DATA_IN;
  END IF;
END PROCESS;
-------------------------------------------------
-- EXAMPLE 2
REG16_NO_RESET:
```

```
PROCESS
BEGIN
  WAIT UNTIL CLOCK = '1';
  Q <= DATA_IN;
END PROCESS;
-------------------------------------------------
-- SYNCHRONOUS RESET
-------------------------------------------------
-- EXAMPLE 3
REG16_SYNCH_RESET:
PROCESS(CLOCK)
BEGIN
  IF CLOCK'EVENT AND CLOCK = '1' THEN
    IF SYNCH_RESET = '1' THEN
      Q <= "0000_0000_0000_0000";
    ELSE
      Q <= DATA_IN;
    END IF;
  END IF;
END PROCESS;
-------------------------------------------------
-- ASYNCHRONOUS RESET OR PRESET
-------------------------------------------------
-- EXAMPLE 4
REG16_ASYNCH_RESET:
PROCESS(ASYNCH_RESET,CLOCK)
BEGIN
  IF ASYNCH_RESET = '1' THEN
    Q <= (OTHERS => '0');
  ELSIF CLOCK'EVENT AND CLOCK = '1' THEN
    Q <= DATA_IN;
  END IF;
END PROCESS;
-------------------------------------------------
-- ASYNCHRONOUS RESET AND PRESET
-------------------------------------------------
-- EXAMPLE 5
REG16_ASYNCH_RESET&PRESET:
PROCESS(ASYNCH_RESET,ASYNCH_PRESET,CLOCK)
BEGIN
  IF ASYNCH_RESET = '1' THEN
    Q <= (OTHERS => '0');
  ELSIF ASYNCH_PRESET = '1' THEN
    Q <= (OTHERS => '1');
  ELSIF CLOCK'EVENT AND CLOCK = '1' THEN
    Q <= DATA_IN;
  END IF;
END PROCESS;
```

```
-------------------------------------------------
-- CONDITIONAL SYNCHRONOUS ASSIGNMENT
-------------------------------------------------
-- EXAMPLE 6
REG16_SYNCH_ASSIGN:
PROCESS(RESET,CLOCK)
BEGIN
  IF RESET = '1' THEN
    Q <= (OTHERS => '0');
  ELSIF CLOCK'EVENT AND CLOCK = '1' THEN
    IF ENABLE = '1' THEN
      Q <= DATA_IN;
    ELSE
      Q <= Q;
    END IF;
  END IF;
END PROCESS;
-------------------------------------------------
-- TRANSLATING A STATE FLOW DIAGRAM
-- TO A TW0-PROCESS FSM DESCRIPTION
-------------------------------------------------
-- EXAMPLE 7
ARCHITECTURE STATE_MACHINE OF STATE_FLOW_DIAGRAM IS
  TYPE STATE_TYPE IS (STATE_IDLE,STATE_DECISION,
                      STATE_READ,STATE_WRITE);
  SIGNAL PRESENT_STATE,NEXT_STATE: STATE_TYPE;
BEGIN
  STATE_COMB:
  PROCESS(PRESENT_STATE,READ_WRITE, READY)
  BEGIN
    CASE PRESENT_STATE IS
      WHEN IDLE =>
        IF READY = '1' THEN
          NEXT_STATE <= DECISION;
        ELSE
          NEXT_STATE <= IDLE;
        END IF;
      WHEN DECISION =>
        READ_ENABLE  <= '0';
        WRITE_ENABLE <= '0';
        IF (READ_WRITE = '1') THEN
          NEXT_STATE <= READ;
        ELSE
          NEXT_STATE <= WRITE;
        END IF;
      WHEN READ =>
        READ_ENABLE  <= '1';
        WRITE_ENABLE <= '0';
        IF (READY = '1') THEN
          NEXT_STATE <= IDLE;
```

```
        ELSE
          NEXT_STATE <= READ;
        END IF;
      WHEN WRITE =>
        READ_ENABLE  <= '0';
        WRITE_ENABLE <= '1';
        IF (READY = '1') THEN
          NEXT_STATE <= IDLE;
        ELSE
          NEXT_STATE <= WRITE;
        END IF;
    END CASE;
END PROCESS STATE_COMB;
STATE_CLOCKED:
PROCESS(CLOCK)
BEGIN
  IF (CLOCK'EVENT AND CLOCK = '1') THEN
    PRESENT_STATE <= NEXT_STATE;
  END IF;
END PROCESS;
END STATE_MACHINE;
-------------------------------------------------
```

Рис. 11.1. Примеры описаний синхронной логики

Если же вместо оператора процесса мы хотим использовать оператор блока, то для D-триггера-защелки напишем:

```
D_latch: block (clk= `1` or reset =`1`)
        begin
                Q <= guarded `0` when reset = `1`
                        else
                                Data_IN when clk = `1`
                        else
                                 Q;
                end block ;
```

Для D-триггера, срабатывающего по фронту, описание примет следующий вид:

```
dff: block ((clk =`1` and clk'event) or reset =`1`)
        begin
                Q <= guarded `0` when reset =`1`;
                         else
                                DATA _IN when (clk =`1` and clk`event`);
                                else
                                Q;
        end block
```

Примеры описаний интерфейса регистра на VHDL приведены на рис. 11.2.

В первом примере описывается интерфейс 16-разрядного регистра, а в примере 2 с помощью GENERIC вводится параметр N, описывающий разрядность регистра.

```
-------------------------------------------------
-- ENTITY DECLARATION
-------------------------------------------------
```

```
-- EXAMPLE 1
ENTITY REGISTER_16 IS
   PORT(
     CLK,RST,EN: IN  STD_LOGIC;
     DATA      : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
     Q         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
END REGISTER_16;
-----------------------------------------------------
-- ENTITY DECLARATION WITH GENERIC
-----------------------------------------------------
-- EXAMPLE 2
ENTITY REGISTER_N IS
   GENERIC(WIDTH: INTEGER := 16);
   PORT(
     CLK,RST,EN: IN  STD_LOGIC;
     DATA      : IN  STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
     Q         : OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
END REGISTER_N;
-----------------------------------------------------
```

Рис. 11.2. Примеры описаний интерфейса регистра

Пример 1 на рис. 11.3 является примером описания функционирования этого 16-разрядного регистра, имеющего входы сброса (RST), синхросигнала (CLK), разрешения (EN) и данных (DATA).

```
-------------------------------------------------------------
-- ARCHITECTURAL BODY
-------------------------------------------------------------
-- EXAMPLE 1
ARCHITECTURE ARCH_REGISTER_16 OF REGISTER_16 IS
BEGIN
  PROCESS(RST,CLK)
  BEGIN
    IF (RST = '1') THEN
      Q <= (OTHERS => '0');
    ELSIF (CLK'EVENT AND CLK = '1') THEN
      IF (EN = '1') THEN
        Q <= DATA;
      ELSE
        Q <=Q;
      END IF;
    END IF;
  END PROCESS;
END ARCH_REGISTER_16;
-------------------------------------------------------------
-- EXAMPLE 2
ARCHITECTURE ARCH_FSM OF FSM IS
  TYPE STATE_TYPE IS (STATE0,STATE1,STATE2)
  SIGNAL STATE    : STATE_TYPE;
  SIGNAL OUT1,OUT2: STD_LOGIC
```

```
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL CLK = '1';
      CASE STATE IS
        WHEN STATE0 =>
          STATE <= STATE1;
          OUT1  <= '1';
        WHEN STATE1 =>
          STATE <= STATE2;
          OUT2  <= '1';
        WHEN OTHERS =>
          STATE <= STATE3;
          OUT1  <= '0';
          OUT2  <= '0';
      END CASE;
  END PROCESS;
END ARCH_FSM;
-------------------------------------------------------------
```

Рис. 11.3. Примеры описаний регистра

На рис. 11.4 приведены описания регистра в виде компонента, который затем может быть использован для построения более длинных регистров (рис. 11.5).

```
-----------------------------------------------------
-- COMPONENT DECLARATION
-----------------------------------------------------
-- EXAMPLE 1
COMPONENT REGISTER_16
   PORT(
     CLK,RST,EN: IN  STD_LOGIC;
     DATA      : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
     Q         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
END COMPONENT;
-----------------------------------------------------
-- EXAMPLE 2
COMPONENT REGISTER_N IS
   GENERIC(WIDTH: INTEGER := 16);
   PORT(
     CLK,RST,EN: IN  STD_LOGIC;
     DATA      : IN  STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
     Q         : OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
END COMPONENT;
-----------------------------------------------------
```

Рис. 11.4. Описание регистра в виде компонента

Другие базовые последовательностные схемы (счетчики и т. д.) описываются моделью Хаффмана [2p], которая, в свою очередь, переводится на язык VHDL.

```
-------------------------------------------------------
-- COMPONENT INSTANTIATION
-------------------------------------------------------
-- EXAMPLE 1
ARCHITECTURE ARCH_REGISTER_16 OF REGISTER_16 IS
  COMPONENT REG_16
    PORT(
      CLK,RST,EN: IN  STD_LOGIC;
      DATA      : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
      Q         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
  END COMPONENT;
  SIGNAL CLOCK,RESET,ENABLE : STD_LOGIC;
  SIGNAL DATA_IN,DATA_OUT   : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  EXEMPLAR1_REGISTER_16:
                        REG_16
                        PORT MAP(
                          CLK =>  CLOCK,
                          RST =>  RESET,
                          EN  =>  ENABLE,
                          DATA => DATA_IN,
                          Q    => DATA_OUT);
END ARCH_REGISTER_16;
-------------------------------------------------------
-- EXAMPLE 2
ARCHITECTURE ARCH_REGISTER_32 OF REGISTER_32 IS
  COMPONENT REG_N IS
    GENERIC(WIDTH: INTEGER := 16);
    PORT(
      CLK,RST,EN: IN  STD_LOGIC;
      DATA      : IN  STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
      Q         : OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
    END COMPONENT;
  SIGNAL CLOCK,RESET,ENABLE : STD_LOGIC;
  SIGNAL DATA_IN,DATA_OUT   : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  EXEMPLAR1_REGISTER_32:
                        REG_N
                        GENERIC MAP(WIDTH => 32)
                        PORT MAP(
                          CLK =>  CLOCK,
                          RST =>  RESET,
                          EN  =>  ENABLE,
                          DATA => DATA_IN,
                          Q    => DATA_OUT);
END ARCH_REGISTER_32;
-------------------------------------------------------
-- EXAMPLE 3
ARCHITECTURE ARCH_REGISTER_16 OF REGISTER_16 IS
  COMPONENT REG_16
```

```
    PORT(
      CLK,RST,EN: IN  STD_LOGIC;
      DATA      : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
      Q         : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
  END COMPONENT;
  SIGNAL CLOCK,RESET,ENABLE : STD_LOGIC;
  SIGNAL DATA_IN,DATA_OUT   : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  EXEMPLAR1_REGISTER_16:
                        REG_16
                        PORT MAP(CLOCK,RESET,ENAB-
LE,DATA_IN,DATA_OUT);
END ARCH_REGISTER_16;
-------------------------------------------------
-- EXAMPLE 4
ARCHITECTURE ARCH_REGISTER_32 OF REGISTER_32 IS
  COMPONENT REG_N IS
    GENERIC(WIDTH: INTEGER := 16);
    PORT(
      CLK,RST,EN: IN  STD_LOGIC;
      DATA      : IN  STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
      Q         : OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
    END COMPONENT;
  SIGNAL CLOCK,RESET,ENABLE : STD_LOGIC;
  SIGNAL DATA_IN,DATA_OUT   : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  EXEMPLAR1_REGISTER_32:
                        REG_N
                        GENERIC MAP(32)
                        PORT MAP(CLOCK,RESET,ENAB-
LE,DATA_IN,DATA_OUT);

END ARCH_REGISTER_32;
```

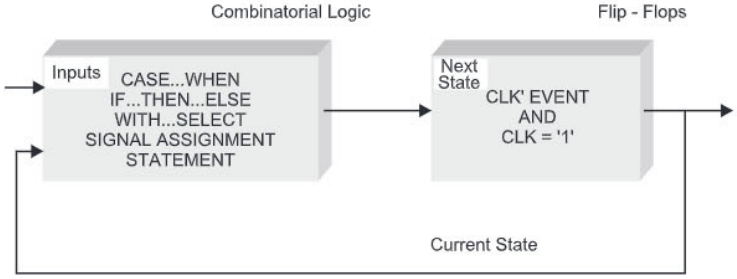
Рис. 11.5. Описания регистра, использующие описания других регистров в виде компонент

11.2. Проектирование цифровых автоматов

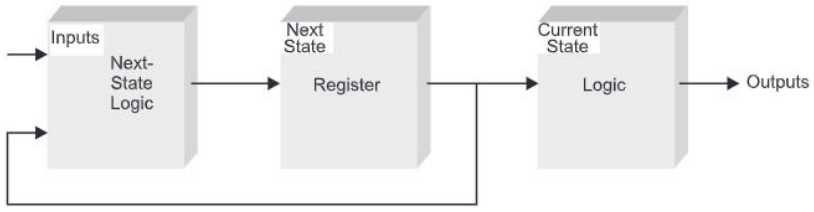
Модель Хаффмана, как известно, является моделью конечного цифрового автомата (Finite State Machine-FSM), включающего комбинационный блок, описываемый последовательными операторами VHDL (Combinatorial Logic), и блок синхронной памяти (Flip-Flops) (рис. 11.6, *а*). Известны два вида конечного цифрового автомата:

- автомат Мура (Moore FSM — рис. 11.6, *б*);
- автомат Мили (Mealy FSM) — рис. 11.6, *в*).

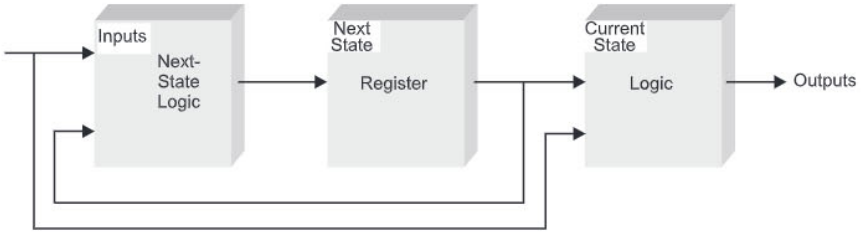
На рис. 11.7 представлены диаграммы состояний таблица состояний Symbolic FSM Table для автомата Мура с тремя состояниями (A, B, C), одним входом (W) и



а) Цифровой автомат как FSM



б) Автомат Мура (Moore FSM)



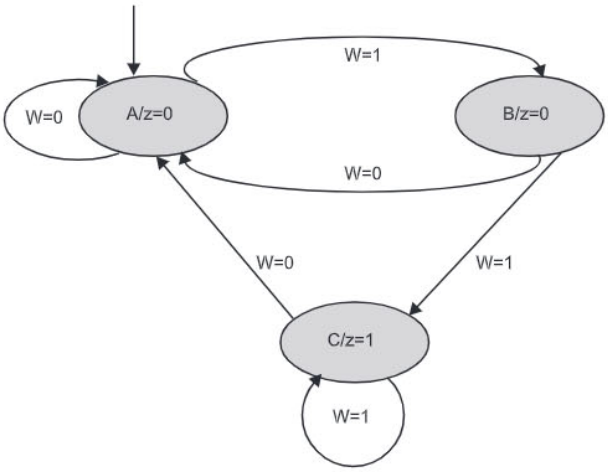
в) Автомат Мили (Mealy FSM)

Рис. 11.6. Цифровые автоматы Мура (Moore FSM) и Мили(Mealy FSM)

одним выходом (Z), а на рис. 11.8 — диаграмма и таблица состояний для автомата Мили с двумя состояниями (A и B), одним входом (W) и одним выходом (Z).

Соответствующие VHDL-модели для этих автоматов приведены на рис. 11.9.

```
-------------------------------------------------------
-- MOORE FSM EXAMPLE 1
-------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MOORE_FSM IS
  PORT(CLOCK, RESETN, W : IN  STD_LOGIC;
       Z                : OUT STD_LOGIC);
END MOORE_FSM;
```

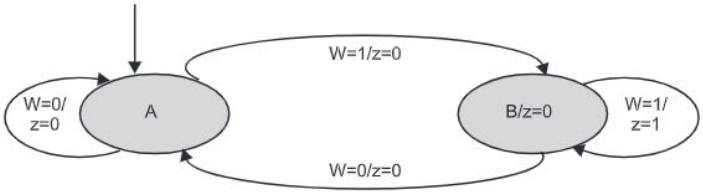


а) Диаграмма состояний для Moore FSM

Present State	Next State		Outputz
	W=0	W=1	
A	A	B	0
B	A	C	0
C	A	C	1

б). Таблица Symbolic FSM Table для Moore FSM

Рис. 11.7. Диаграмма и таблица состояний для автомата Мура



а) Диаграмма состояний для Mealy FSM

Present State	Next State		Outputz	
	W=0	W=1	W=0	W=1
A	A	B	0	0
B	A	B	0	1

б). Таблица Symbolic FSM Table для Mealy FSM

Рис. 11.8. Диаграмма и таблица состояний для автомата Мили

```
ARCHITECTURE BEHAVIOR1 OF MOORE_FSM IS
  TYPE STATE_TYPE IS (A, B, C);
  SIGNAL Y : STATE_TYPE;
BEGIN
  PROCESS(CLOCK, RESETN)
  BEGIN
     IF RESETN = '0' THEN
          Y <= A;
```

```
     ELSIF (CLOCK'EVENT AND CLOCK = '1') THEN
        CASE Y IS
           WHEN A =>
              IF W = '0' THEN
                   Y <= A;
              ELSE
                   Y <= B;
              END IF;
           WHEN B =>
              IF W = '0' THEN
                   Y <= A;
              ELSE
                   Y <= C;
              END IF;
           WHEN C =>
              IF W = '0' THEN
                   Y <= A;
              ELSE
                   Y <= C;
              END IF;
        END CASE;
     END IF;
  END PROCESS;
  Z <= '1' WHEN Y = C ELSE '0';
END BEHAVIOR1;
-------------------------------------------------------------
-- MOORE FSM EXAMPLE 2
-------------------------------------------------------------
ARCHITECTURE BEHAVIOR2 OF MOORE_FSM IS
  TYPE STATE_TYPE IS (A, B, C);
  SIGNAL Y_PRESENT, Y_NEXT : STATE_TYPE;
BEGIN
  PROCESS(W, Y_PRESENT)
  BEGIN
          CASE Y_PRESENT IS
           WHEN A =>
              IF W = '0' THEN
                   Y_NEXT <= A;
              ELSE
                   Y_NEXT <= B;
              END IF;
           WHEN B =>
              IF W = '0' THEN
                   Y_NEXT <= A;
              ELSE
                   Y_NEXT <= C;
              END IF;
           WHEN C =>
              IF W = '0' THEN
                   Y_NEXT <= A;
```

```
              ELSE
                   Y_NEXT <= C;
              END IF;
        END CASE;
  END PROCESS;
  PROCESS(CLOCK, RESETN)
  BEGIN
     IF RESETN = '0' THEN
          Y_PRESENT <= A;
     ELSIF (CLOCK'EVENT AND CLOCK = '1') THEN
          Y_PRESENT <= Y_NEXT;
     END IF;
  END PROCESS;
  Z <= '1' WHEN Y_PRESENT = C ELSE '0';
END BEHAVIOR2;
-------------------------------------------------------------
-- MOORE FSM EXAMPLE 3
-------------------------------------------------------------
ARCHITECTURE BEHAVIOR3 OF MOORE_FSM IS
    SIGNAL Y_PRESENT, Y_NEXT : STD_LOGIC_VECTOR(1 DOWNTO 0);
    CONSTANT A : STD_LOGIC_VECTOR(1 DOWNTO 0):= "00" ;
    CONSTANT B : STD_LOGIC_VECTOR(1 DOWNTO 0):= "01" ;
    CONSTANT C : STD_LOGIC_VECTOR(1 DOWNTO 0):= "11" ;
BEGIN
  PROCESS(W, Y_PRESENT)
  BEGIN
          CASE Y_PRESENT IS
           WHEN A =>
              IF W = '0' THEN
                   Y_NEXT <= A;
              ELSE
                   Y_NEXT <= B;
              END IF;
           WHEN B =>
              IF W = '0' THEN
                   Y_NEXT <= A;
              ELSE
                   Y_NEXT <= C;
              END IF;
           WHEN C =>
              IF W = '0' THEN
                   Y_NEXT <= A;
              ELSE
                   Y_NEXT <= C;
              END IF;
           WHEN OTHERS =>
              Y_NEXT <= A;
        END CASE;
  END PROCESS;
  PROCESS(CLOCK, RESETN)
```

```
  BEGIN
     IF RESETN = '0' THEN
          Y_PRESENT <= A;
     ELSIF (CLOCK'EVENT AND CLOCK = '1') THEN
          Y_PRESENT <= Y_NEXT;
     END IF;
  END PROCESS;
  Z <= '1' WHEN Y_PRESENT = C ELSE '0';
END BEHAVIOR3;
-------------------------------------------------------------
-- MEALY FSM EXAMPLE
-------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MEALY_FSM IS
  PORT(CLOCK, RESETN, W : IN  STD_LOGIC;
       Z                : OUT STD_LOGIC);
END MEALY_FSM;
ARCHITECTURE BEHAVIOR OF MEALY_FSM IS
  TYPE STATE_TYPE IS (A, B);
  SIGNAL Y : STATE_TYPE;
BEGIN
  PROCESS(CLOCK, RESETN)
  BEGIN
     IF RESETN = '0' THEN
          Y <= A;
     ELSIF (CLOCK'EVENT AND CLOCK = '1') THEN
        CASE Y IS
           WHEN A =>
              IF W = '0' THEN
                   Y <= A;
              ELSE
                   Y <= B;
              END IF;
           WHEN B =>
              IF W = '0' THEN
                   Y <= A;
              ELSE
                   Y <= B;
              END IF;
        END CASE;
     END IF;
  END PROCESS;
  PROCESS(Y, W)
  BEGIN
     CASE Y IS
        WHEN A =>
           Z <= '0';
        WHEN B =>
           Z <= 'W';
```

```
    END CASE;
  END PROCESS;
END BEHAVIOR;
```

Рис. 11.9. VHDL-модели для автоматов Мура и Мили (Moore FSM, Mealy FSM)

Поведенческая модель BEHAVIOR1 для автомата Мура содержит процесс, внутри которого используются последовательные операторы CASE и IF. Этот процесс описывает переходы между состояниями, а также процедуры сброса и синхронизации. Оператор присваивания для сигнала Z описывает состояние выхода автомата.

Поведенческая модель BEHAVIOR2 для автомата Мура содержит два процессора. Первый процессор описывает лишь переходы между состояниями автомата. Этот процесс активизируется при наличии изменения текущего состояния автомата. Поток таких изменений генерирует второй процесс в зависимости от состояний входов сброса и синхросигнала.

В поведенческой модели BEHAVIOR3 вместо перечисленного типа данных STATE_TYPE для состояний автомата используются три константы.

Архитектура BEHAVIOR для автомата Мили также включает два процесса. Первый процесс описывает переходы между состояниями в зависимости от синхросигнала и сигнала сброса. Второй же процесс описывает состояние выхода автомата.

Существует альтернативное представление конечного цифрового автомата в виде ASM (Algorithmic State Machine). Представление в виде FSM удобно для реализации автомата с помощью аппаратуры, представление же в виде ASM более удобны для реализации автомата в виде интегрированной системы, так как исходными данными здесь является блок-схема алгоритма функционирования автомата (ASM Chart). Блок-схема ASM Chart [10a] строится с помощью следующих типовых блоков (рис. 11.10):

- состояние системы (System State),
- блок принятия решения (Decision Box),
- блок условного выхода (Conditional Output Box).

Совокупность состояний автомата реализуется определенным числом блоков System State. Выходные сигналы автомата подразделяются на два вида:

- выходы Мура (они связаны с блоками System State, например H.ADD на рис. 11.10, *а*; при этом H означает, что данный сигнал активен в состоянии

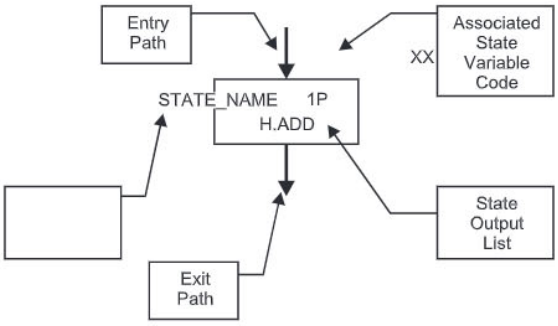


Рис. 11.10,а. Цифровой автомат как ASM. System state

логической единицы; если же сигнал записан в виде L.ADD, это означает, что он активен в состоянии логического нуля);

– выходы Мили (они связаны с переходами из состояния в состояние, например IL.CLOCK, IH.RESET на рис. 11.10, *в* ; буква I и означает, что данный выходной сигнал является выходом Мили).

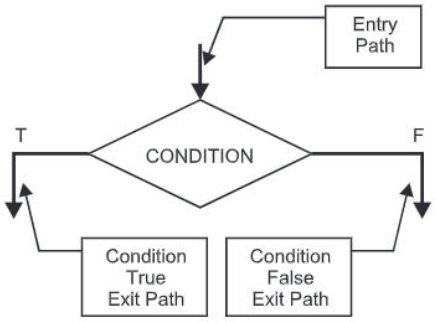


Рис. 11.10,б. Цифровой автомат как ASM. Decision box

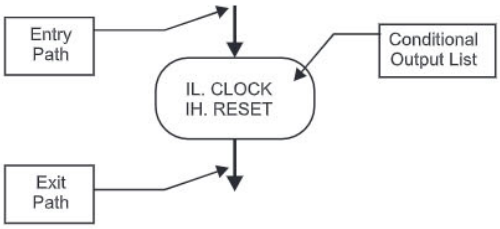


Рис. 11.10,в. Цифровой автомат как ASM. Conditional output box

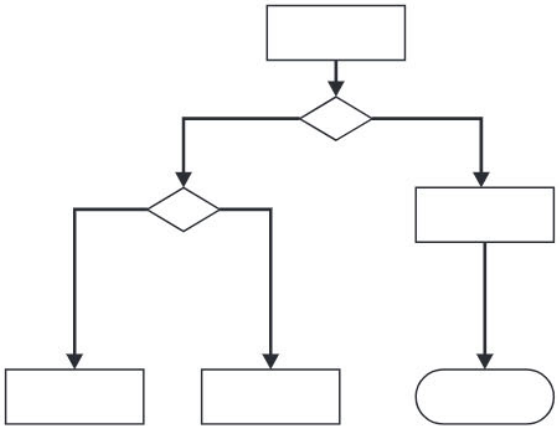


Рис. 11.10,г. Цифровой автомат как ASM. ASM block (ASM chart)

Исходное состояние автомата обозначается одним из трех возможных способов, приведенных на рис. 11.10, *д*.

С помощью блоков принятия решений осуществляется анализ выходных сигналов автомата.

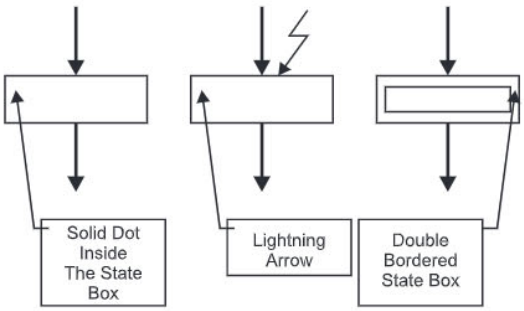


Рис. 11.10,д. Цифровой автомат как ASM. Reset or starting states

Блок-схема ASM CHART на рис. 11.10, *е*, например, содержит четыре состояния A, B, C и D и восемь возможных маршрутов перехода между состояниями:

- маршрут L1 (переход из состояния A в состояние B), реализуется при следующей комбинации входных сигналов: X1 = '0', X3 = '0';
- маршрут L2 (переход из состояния A в состояние B), при условии, что X1 = '1', X2 = '0';
- маршрут L3 (переход из состояния A в состояние C), при условии, что X1 = '0', X2 = '0'; X3 = '1';
- маршрут L4 (переход из состояния A в состояние C), при условии, что X1 = '0', X3 = '1';
- маршрут L5 (переход из состояния A в состояние D), при условии, что X1 = '1', X2 = '1';
- маршрут L6 (безусловный переход из состояния B в состояние A);
- маршрут L7 (безусловный переход из состояния C в состояние A);
- маршрут L8 (безусловный переход из состояния D в состояние A).

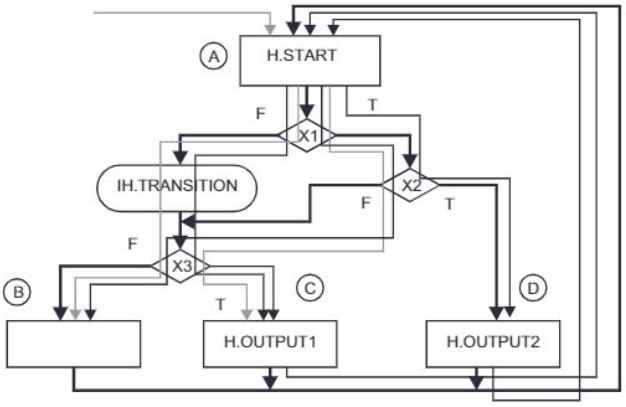


Рис. 11.10,е. Цифровой автомат как ASM. ASM block example

Состояние A автомата является его исходным состоянием (помечено с помощью способа 1 на рис. 11.10, *д*).

Данный автомат имеет три входных сигнала (X1, X2, X3) и четыре выходных. Три из них — выходы Мура, которые связаны с состояниями автомата:

- H.START (связан с состоянием A);
- L.OUTPUT1 (связан с состоянием C);
- H.OUTPUT2 (связан с состоянием D);

а четвертый — выход Мили, который связан с переходом из состояния A в состояние B или C (IH.TRANSITION).

Заметим, что выход Мили может быть записан и внутри одного из прямоугольников System State. Это будет означать, что данный выход Мили связан со всеми маршрутами, выходящими из этого состояния.

На рис. 11.11 приведена блок-схема ASM Chart, содержащая:

- 6 состояний (A, B, C, D, E, F),
- 3 входных сигнала (X1, X2, X3),
- 11 маршрутов перехода между состояниями L1 L11 (см. рис. 11.12),
- 5 выходных сигналов, два из которых — выходы Мура (H.OUT1, H.OUT2), а оставшиеся три — выходы Мили (IH.OUT3, IL.OUT4, IH.OUT5).

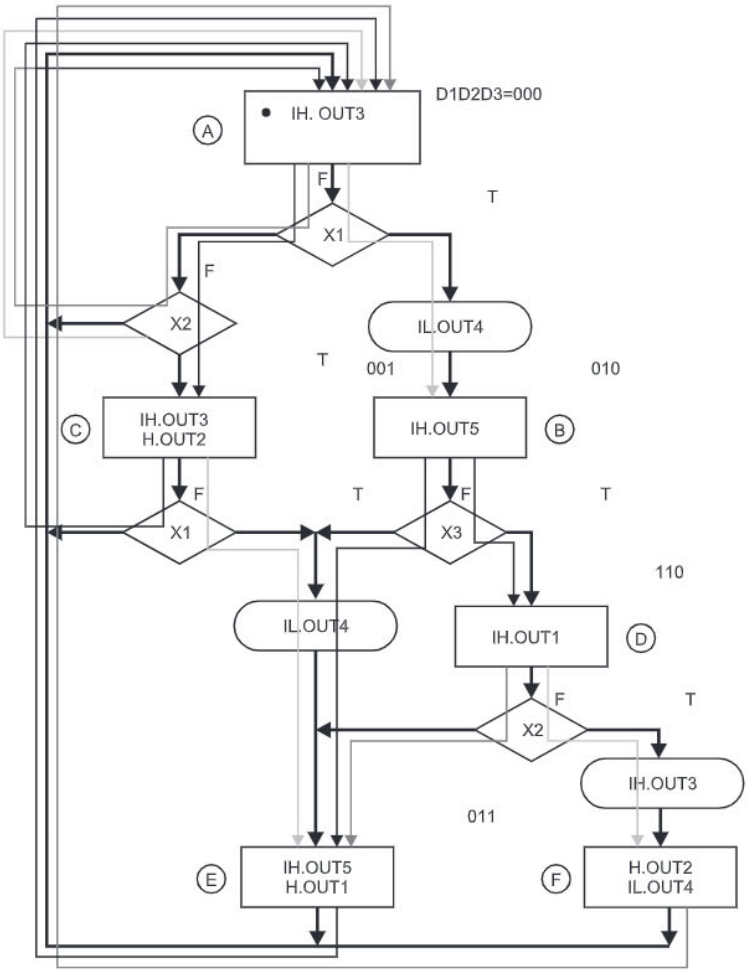


Рис. 11.11. Диаграмма состояний, переходов, входных и выходных сигналов ASM (ASM Chart)

По данной блок-схеме можно составить таблицу, называющуюся Symbolic ASM Table (рис. 11.12). Каждая строка таблицы соответствует одному из маршрутов (Linkpathes). Воспользовавшись кодами состояний и сигналов, можно составить еще одну таблицу — Assigned ASM Table (рис. 11.13).

Таблица Symbolic ASM Table служит базой для написания VHDL-модели ASM-автомата, приведенной на рис. 11.14. Результаты моделирования данного автомата приведены на рис. 11.15.

SYMBOLIC ASM TABLE

LINK-PATH	INPUTS			PRESENT STATE	NEXT STATE	MOORE OUTPUTS		MEALY OUTPUTS		
	X1	X2	X3			H.OUT1	H.OUT2	IH.OUT3	IL.OUT4	IH.OUT5
L1	F	F	–	A	A	–	–	Active	–	–
L2	T	–	–	A	B	–	–	Active	Active	–
L3	F	T	–	A	C	–	–	Active	–	–
L4	–	–	F	B	D	–	–	–	–	–
L5	–	–	T	B	E	–	–	–	–	Active
L6	F	–	–	C	A	–	Active	Active	–	–
L7	T	–	–	C	E	–	Active	Active	–	Active
L8	–	F	–	D	E	Active	–	–	–	–
L9	–	T	–	D	F	Active	–	Active	–	–
L10	–	–	–	E	A	Active	–	–	–	Active
L11	–	–	–	F	A	–	Active	–	Active	–

F – FALSE

T – TRUE

Рис. 11.12. Таблица состояний ASM

ASSIGNED ASM TABLE

LINK-PATH	INPUTS			PRESENT STATE	NEXT STATE	MOORE OUTPUTS		MEALY OUTPUTS		
	X1	X2	X3	D1 D2 D3	D1+D2+D3+	H.OUT1	H.OUT2	IH.OUT3	IL.OUT4	IH.OUT5
L1	0	0	–	0 0 0	0 0 0	0	0	1	1	0
L2	1	–	–	0 0 0	0 1 0	0	0	1	0	0
L3	0	1	–	0 0 0	1 0 0	0	0	1	1	0
L4	–	–	0	0 1 0	1 1 0	0	0	0	1	0
L5	–	–	1	0 1 0	0 1 1	0	0	0	1	1
L6	0	–	–	1 0 0	0 0 0	0	1	1	1	0
L7	1	–	–	1 0 0	0 1 1	0	1	1	1	1
L8	–	0	–	1 1 0	0 1 1	1	0	0	1	0
L9	–	1	–	1 1 0	1 0 0	1	0	1	1	0
L10	–	–	–	0 1 1	0 0 0	1	0	0	1	1
L11	–	–	–	1 0 0	0 0 0	0	1	0	0	0

Рис. 11.13. Закодированная таблица состояний ASM

```
-----------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-----------------------------------------
ENTITY SAMPLE1_ASM IS
  PORT(CLK,RESET               : IN  std_logic;  -- CONTROL SIGNALS
       X1,X2,X3                : IN  std_logic;  -- INPUT SIGNALS
       H_OUT1,H_OUT2           : OUT std_logic;  -- MOORE OUTPUTS
       IH_OUT3,IL_OUT4,IH_OUT5 : OUT std_logic); -- MEALY OUTPUTS
END SAMPLE1_ASM;
-----------------------------------------
ARCHITECTURE STATE_MACHINE OF SAMPLE1_ASM IS
   TYPE STATE_TYPE IS (STATE_A, -- ENUMERATED TYPE FOR STATES
                       STATE_B,
                       STATE_C,
                       STATE_D,
                       STATE_E,
                       STATE_F);
   SIGNAL PRESENT_STATE,NEXT_STATE : STATE_TYPE;
BEGIN
   STATE_LOGIC : PROCESS(PRESENT_STATE,X1,X2,X3)
                 BEGIN
                    H_OUT1  <= '0';
                    H_OUT2  <= '0';
                    IH_OUT3 <= '0';
                    IL_OUT4 <= '1';
                    IH_OUT5 <= '0';
   CASE PRESENT_STATE IS
      WHEN STATE_A =>
         IH_OUT3 <= '1' ;
         IF (X1 = '1')
           THEN
              NEXT_STATE <= STATE_B;
              IL_OUT4    <= '0';
         ELSIF (X2 = '1')
           THEN
              NEXT_STATE <= STATE_C;
           ELSE
              NEXT_STATE <= STATE_A;
         END IF;
      WHEN STATE_B =>
         IF (X3 = '1')
           THEN
              NEXT_STATE <= STATE_E;
              IH_OUT5 <= '1';
           ELSE
              NEXT_STATE <= STATE_D;
         END IF;
      WHEN STATE_C =>
         IH_OUT5 <= '1';
```

```
        H_OUT2  <= '1';
        IF (X1 = '1')
          THEN
             NEXT_STATE <= STATE_E;
             IH_OUT5 <= '1';
          ELSE
             NEXT_STATE <= STATE_A;
        END IF;
      WHEN STATE_D =>
        H_OUT1 <= '1';
        IF (X2 = '1')
          THEN
             NEXT_STATE <= STATE_F;
             IH_OUT3 <= '1';
        ELSE
             NEXT_STATE <= STATE_E;
        END IF;
      WHEN STATE_E =>
        IH_OUT5 <= '1';
        H_OUT1  <= '1';
        NEXT_STATE <= STATE_A;
      WHEN STATE_F =>
        H_OUT2  <= '1';
        IL_OUT4 <= '1';
        NEXT_STATE <= STATE_A;
  END CASE;
  END PROCESS STATE_LOGIC;
  SEQUENTIAL: PROCESS(CLK,RESET)
              BEGIN
       IF RESET = '1' -- GO TO STATE_A ON RESET
         THEN
            PRESENT_STATE <= STATE_A;
       ELSIF RISING_EDGE(CLK) -- LOAD NEXT STATE ON CLOCK EDGE
         THEN
            PRESENT_STATE <= NEXT_STATE;
       END IF;
  END PROCESS SEQUENTIAL;
END STATE_MACHINE;
-----------------------------------------
```

Рис. 11.14. VHDL — модель для ASM

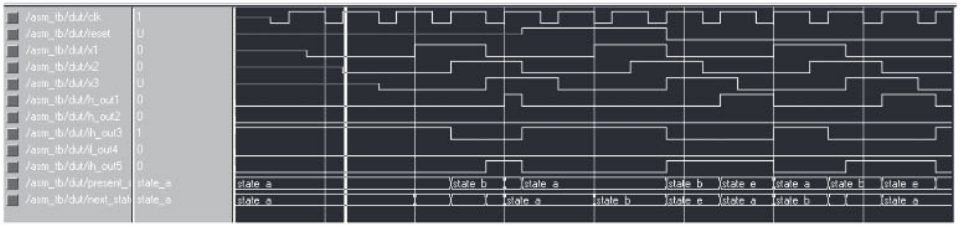


Рис. 11.15. Результаты моделирования ASM

11.3. Оперативная память на VHDL

Напишем теперь VHDL-модель оперативной памяти (RAM), схема которой приведена на рис. 11.16 (Tanenbaum A. Structured Computer Organization. — Prentice Hall, 1999).

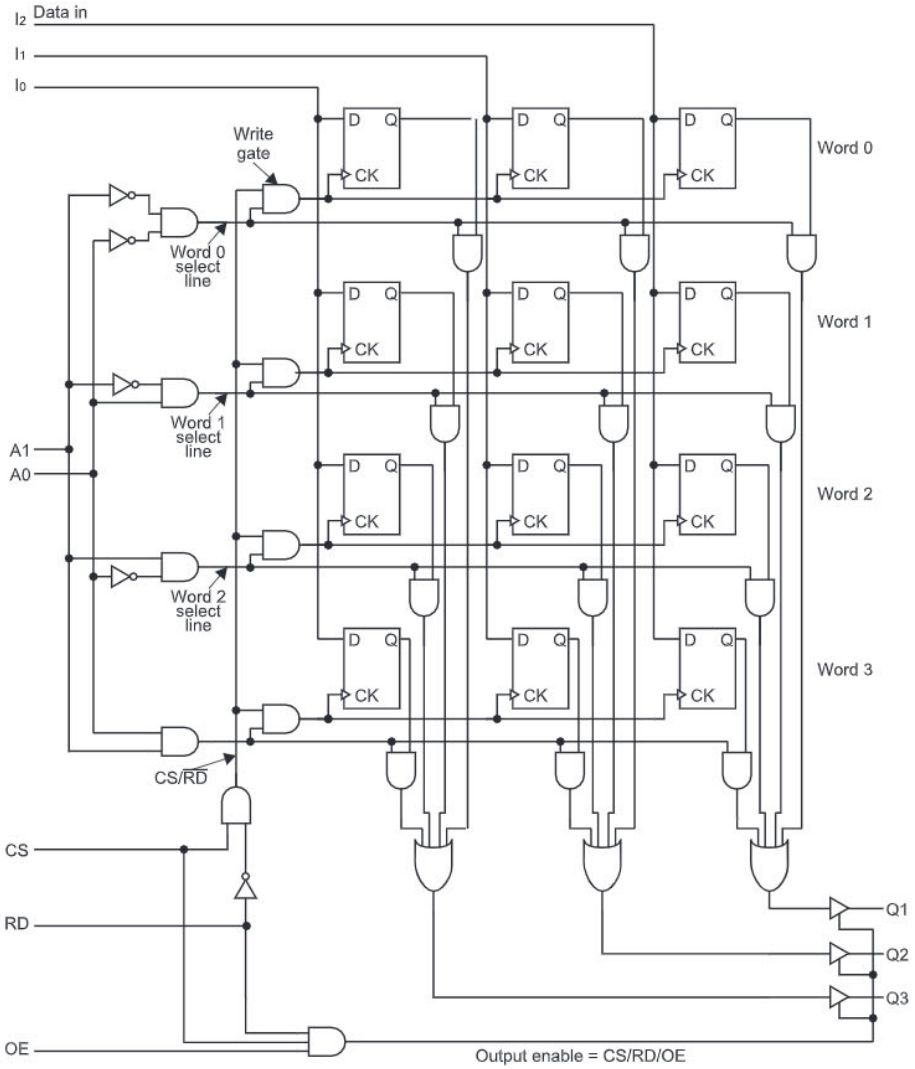


Рис. 11.16. Схема памяти 4x3 RAM

На рис. 11.17 приведена структурная модель такой памяти, использующая VHDL-компоненты DFF (D-триггер), BUFF (буфер на выходе), AND3 и AND2 (вентили И), OR4 (вентиль ИЛИ), INV (инвертор) и DEC (декодер).

```
-----------------------------------------
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.ALL;
ENTITY AND3 IS
  PORT(P1,P2,P3 :  IN  bit;
       PZ       :  OUT bit);
END AND3;
ARCHITECTURE and3 OF AND3 IS
BEGIN
  PZ <= P1 AND P2 AND P3;
END and3;
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY buff IS
  PORT(data_in  :  IN  bit;
       control  :  IN  bit;
       data_out :  OUT std_logic);
END buff;
ARCHITECTURE buff OF buff IS
BEGIN
  PROCESS(data_in,control)
  BEGIN
     IF(control='1') THEN
       IF(data_in='1') THEN
          data_out <= '1';
       ELSE
          data_out <= '0';
       END IF;
     ELSE
          data_out <= 'Z';
     END IF;
  END PROCESS;
END buff;
-----------------------------------------
ENTITY dff IS
  PORT(D  :  IN  bit;
       CK :  IN  bit;
       Q  :  OUT bit);
END dff;
ARCHITECTURE dff OF dff IS
BEGIN
  PROCESS(D,CK)
  BEGIN
     IF((CK='1')AND(NOT CK'stable)) THEN
        Q <= D;
     END IF;
  END PROCESS;
END dff;
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
ENTITY ram4x3bit IS
  PORT(I  : IN  bit_vector(0 TO 2);        -- Data IN
       A  : IN  bit_vector(0 TO 1);        -- Address
       CS : IN  bit;                       -- Chip Select
       RD : IN  bit;                       -- Read
       OE : IN  bit;                       -- Output
       O  : OUT std_logic_vector(0 TO 2)); -- Data OUT
END ram4x3bit;
-----------------------------------------
ARCHITECTURE ram4x3bit_structure OF ram4x3bit IS
   COMPONENT DFF
       PORT(D           :  IN  bit;
            CK          :  IN  bit;
            Q           :  OUT bit);
   END COMPONENT;
   COMPONENT BUFF
       PORT(data_in     :  IN  bit;
            control     :  IN  bit;
            data_out    :  OUT std_logic);
   END COMPONENT;
   COMPONENT AND3
       PORT(P1,P2,P3    :  IN  bit;
            PZ          :  OUT bit);
   END COMPONENT;
   COMPONENT AND2
       PORT(P1,P2       :  IN  bit;
            PZ          :  OUT bit);
   END COMPONENT;
   COMPONENT OR4
       PORT(P1,P2,P3,P4 :  IN  bit;
            PZ          :  OUT bit);
   END COMPONENT;
   COMPONENT INV
       PORT(I           :  IN  bit;
            O           :  OUT bit);
   END COMPONENT;
   COMPONENT DEC
       PORT(x           : IN  bit_vector(0 TO 1);
            y           : OUT bit_vector(0 TO 3));
   END COMPONENT;
   TYPE connections IS ARRAY(0 TO 3, 0 TO 2) OF bit;
   SIGNAL WORD0     : bit;
   SIGNAL WORD1     : bit;
   SIGNAL WORD2     : bit;
   SIGNAL WORD3     : bit;
   SIGNAL CS_NOTRD  : bit;
   SIGNAL CS_RD_OE  : bit;
   SIGNAL CONNECT   : connections;
   SIGNAL SUMLOG    : bit_vector(0 TO 11);
   SIGNAL SUMLOG1   : bit_vector(0 TO 2);
```

```
   SIGNAL CLK0      : bit;
   SIGNAL CLK1      : bit;
   SIGNAL CLK2      : bit;
   SIGNAL CLK3      : bit;
   SIGNAL RD1       : bit;
BEGIN
   A1 : AND2 PORT MAP(CS_NOTRD,WORD0,CLK0);
   A2 : AND2 PORT MAP(CS_NOTRD,WORD1,CLK1);
   A3 : AND2 PORT MAP(CS_NOTRD,WORD2,CLK2);
   A4 : AND2 PORT MAP(CS_NOTRD,WORD3,CLK3);
   A5 : AND2 PORT MAP(WORD0,CONNECT(0,0),SUMLOG(0));
   A6 : AND2 PORT MAP(WORD1,CONNECT(1,0),SUMLOG(1));
   A7 : AND2 PORT MAP(WORD2,CONNECT(2,0),SUMLOG(2));
   A8 : AND2 PORT MAP(WORD3,CONNECT(3,0),SUMLOG(3));
   A9 : AND2 PORT MAP(WORD0,CONNECT(0,1),SUMLOG(4));
   A10: AND2 PORT MAP(WORD1,CONNECT(1,1),SUMLOG(5));
   A11: AND2 PORT MAP(WORD2,CONNECT(2,1),SUMLOG(6));
   A12: AND2 PORT MAP(WORD3,CONNECT(3,1),SUMLOG(7));
   A13: AND2 PORT MAP(WORD0,CONNECT(0,2),SUMLOG(8));
   A14: AND2 PORT MAP(WORD1,CONNECT(1,2),SUMLOG(9));
   A15: AND2 PORT MAP(WORD2,CONNECT(2,2),SUMLOG(10));
   A16: AND2 PORT MAP(WORD3,CONNECT(3,2),SUMLOG(11));
   A17: AND2 PORT MAP(CS,RD1,CS_NOTRD);
   A18: AND3 PORT MAP(RD,CS,OE,CS_RD_OE);
   O1 :  OR4 PORT MAP(SUMLOG(0),SUMLOG(1),SUMLOG(2),SUMLOG(3),SUM-
LOG1(0));
   O2 :  OR4 PORT MAP(SUMLOG(4),SUMLOG(5),SUMLOG(6),SUMLOG(7),SUM-
LOG1(1));
   O3 :  OR4 PORT MAP(SUMLOG(8),SUMLOG(9),SUMLOG(10),SUM-
LOG(11),SUMLOG1(2));
  BUF1: BUFF PORT MAP(SUMLOG1(0),CS_RD_OE,O(0));
  BUF2: BUFF PORT MAP(SUMLOG1(1),CS_RD_OE,O(1));
  BUF3: BUFF PORT MAP(SUMLOG1(2),CS_RD_OE,O(2));
  INV1:  INV PORT MAP(RD,RD1);
  DEC1:  DEC PORT
MAP(A,y(0)=>WORD0,y(1)=>WORD1,y(2)=>WORD2,y(3)=>WORD3);
  DFF1:  DFF PORT MAP(I(0),CLK0,CONNECT(0,0));
  DFF2:  DFF PORT MAP(I(0),CLK1,CONNECT(1,0));
  DFF3:  DFF PORT MAP(I(0),CLK2,CONNECT(2,0));
  DFF4:  DFF PORT MAP(I(0),CLK3,CONNECT(3,0));
  DFF5:  DFF PORT MAP(I(1),CLK0,CONNECT(0,1));
  DFF6:  DFF PORT MAP(I(1),CLK1,CONNECT(1,1));
  DFF7:  DFF PORT MAP(I(1),CLK2,CONNECT(2,1));
  DFF8:  DFF PORT MAP(I(1),CLK3,CONNECT(3,1));
  DFF9:  DFF PORT MAP(I(2),CLK0,CONNECT(0,2));
  DFF10: DFF PORT MAP(I(2),CLK1,CONNECT(1,2));
  DFF11: DFF PORT MAP(I(2),CLK2,CONNECT(2,2));
  DFF12: DFF PORT MAP(I(2),CLK3,CONNECT(3,2));
END ram4x3bit_structure;
-----------------------------------------
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ram4x3bit_tb IS
---------------------------
END ram4x3bit_tb;
ARCHITECTURE ram4x3bit_tb OF ram4x3bit_tb IS
        COMPONENT ram4x3bit
          PORT(I  : IN  bit_vector(0 TO 2);        -- Data IN
               A  : IN  bit_vector(0 TO 1);        -- Address
               CS : IN  bit;                       -- Chip Select
               RD : IN  bit;                       -- Read
               OE : IN  bit;                       -- Output
               O  : OUT std_logic_vector(0 TO 2)); -- Data OUT
        END COMPONENT;
        SIGNAL  NCS,NRD,NOE  : bit;
        SIGNAL  NI           : bit_vector(0 TO 2) ;
        SIGNAL  NO           : std_logic_vector(0 TO 2) ;
        SIGNAL  NA           : bit_vector(0 TO 1) ;
BEGIN
  dut : ram4x3bit
        PORT MAP(
             I         => NI,
             A         => NA,
             CS        => NCS,
             RD        => NRD,
             OE        => NOE,
             O         => NO);
  NCS  <= '1';
  NRD  <= '1' AFTER 80 ns;
  NOE <= NRD;
  stimulus : PROCESS
  BEGIN
    ASSERT (false)
           REPORT "WRITE OPERATION"
           SEVERITY NOTE;
    NA <= "00";
    NI <= "101";
    WAIT FOR 20 ns;
    NA <= "01";
    NI <= "010";
    WAIT FOR 20 ns;
    NA <= "10";
    NI <= "101";
    WAIT FOR 20 ns;
    NA <= "11";
    NI <= "010";
    ASSERT (false)
           REPORT "READ OPERATION"
           SEVERITY NOTE;
    WAIT FOR 20 ns;
```

```
    NA <= "00";
    WAIT FOR 2 ns;
    ASSERT (NO="101")
           REPORT "RAM REACTION IS NOT GOOD!"
           SEVERITY ERROR;
    WAIT FOR 20 ns;
    NA <= "01";
    WAIT FOR 2 ns;
    ASSERT (NO="010")
           REPORT "RAM REACTION IS NOT GOOD!"
           SEVERITY ERROR;
    WAIT FOR 20 ns;
    NA <= "10";
    WAIT FOR 2 ns;
    ASSERT (NO="101")
           REPORT "RAM REACTION IS NOT GOOD!"
           SEVERITY ERROR;
    WAIT FOR 20 ns;
    NA <= "11";
    WAIT FOR 2 ns;
    ASSERT (NO="010")
           REPORT "RAM REACTION IS NOT GOOD!"
           SEVERITY ERROR;
    WAIT;
  END PROCESS ;
END ram4x3bit_tb;
```

Рис. 11.17. VHDL-модель памяти 4x3 RAM

Результаты моделирования с помощью испытательной программы ram4x3bit_tb приведены на рис. 11.19.

На рис. 11.18 приведен еще один вариант структурной модели памяти с помощью оператора Generate Statement. Здесь еще раз можно убедиться, что применение этого оператора к описанию схемы, имеющей регулярную структуру, существенно сокращает число строк VHDL-модели.

```
-----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ram4x3bit IS
  PORT(I  : IN  bit_vector(0 TO 2);        -- Data IN
       A  : IN  bit_vector(0 TO 1);        -- Address
       CS : IN  bit;                       -- Chip Select
       RD : IN  bit;                       -- Read
       OE : IN  bit;                       -- Output
       O  : OUT std_logic_vector(0 TO 2)); -- Data OUT
END ram4x3bit;
-----------------------------------------
ARCHITECTURE ram4x3bit_gen_structure OF ram4x3bit IS
   COMPONENT DFF
       PORT(D  :  IN  bit;
```

```
            CK :  IN  bit;
            Q  :  OUT bit);
    END COMPONENT;
    COMPONENT BUFF
        PORT(data_in  :  IN  bit;
             control  :  IN  bit;
             data_out :  OUT std_logic);
    END COMPONENT;
    COMPONENT AND3
        PORT(P1,P2,P3 :  IN  bit;
             PZ       :  OUT bit);
    END COMPONENT;
    COMPONENT AND2
        PORT(P1,P2 :  IN  bit;
             PZ    :  OUT bit);
    END COMPONENT;
    COMPONENT OR4
        PORT(P1,P2,P3,P4 :  IN  bit;
             PZ          :  OUT bit);
    END COMPONENT;
    COMPONENT inV
        PORT(I :  IN  bit;
             O :  OUT bit);
    END COMPONENT;
    COMPONENT DEC
        PORT(x : IN  bit_vector(0 TO 1);
             y : OUT bit_vector(0 TO 3));
    END COMPONENT;
    TYPE connections IS ARRAY(0 TO 3, 0 TO 2) OF bit;
    SIGNAL WORD      : bit_vector(0 TO 3);
    SIGNAL CS_NOTRD  : bit;
    SIGNAL CS_RD_OE  : bit;
    SIGNAL CONNECT   : connections;
    SIGNAL SUMLOG    : bit_vector(0 TO 11);
    SIGNAL SUMLOG1   : bit_vector(0 TO 2);
    SIGNAL CLK       : bit_vector(0 TO 3);
    SIGNAL RD1       : bit;
BEGIN
    A1 : AND2 PORT MAP(CS_NOTRD,WORD(0),CLK(0));
    A2 : AND2 PORT MAP(CS_NOTRD,WORD(1),CLK(1));
    A3 : AND2 PORT MAP(CS_NOTRD,WORD(2),CLK(2));
    A4 : AND2 PORT MAP(CS_NOTRD,WORD(3),CLK(3));
    g0 : FOR j IN 0 TO 2 GENERATE
        g1 : FOR i IN 0 TO 3 GENERATE
             ALL_AND : AND2
             PORT MAP(P1=>WORD(i),P2=>CONNECT(i,j),PZ=>SUM-
LOG(4*j+i));
        END GENERATE;
    END GENERATE;
    A17: AND2 PORT MAP(CS,RD1,CS_NOTRD);
```

```
   A18: AND3 PORT MAP(RD,CS,OE,CS_RD_OE);
   O1 :  OR4 PORT MAP(SUMLOG(0),SUMLOG(1),SUMLOG(2),SUMLOG(3),SUM-
LOG1(0));
   O2 :  OR4 PORT MAP(SUMLOG(4),SUMLOG(5),SUMLOG(6),SUMLOG(7),SUM-
LOG1(1));
   O3 :  OR4 PORT MAP(SUMLOG(8),SUMLOG(9),SUMLOG(10),SUM-
LOG(11),SUMLOG1(2));
  BUF1: BUFF PORT MAP(SUMLOG1(0),CS_RD_OE,O(0));
  BUF2: BUFF PORT MAP(SUMLOG1(1),CS_RD_OE,O(1));
  BUF3: BUFF PORT MAP(SUMLOG1(2),CS_RD_OE,O(2));
  inV1:  inV PORT MAP(RD,RD1);
  DEC1:  DEC PORT
MAP(A,y(0)=>WORD(0),y(1)=>WORD(1),y(2)=>WORD(2),y(3)=>WORD(3));
   g2 : FOR k IN 0 TO 2 GENERATE
        g3 : FOR l IN 0 TO 3 GENERATE
             ALL_DFF : DFF
             PORT MAP(D=>I(k),CK=>CLK(l),Q=>CONNECT(l,k));
        END GENERATE;
   END GENERATE;
END ram4x3bit_gen_structure;
-----------------------------------------
```

Рис. 11.18. VHDL-модель памяти 4x3 RAM, использующая оператор генерации экземпляров компонент (Generate Statement)

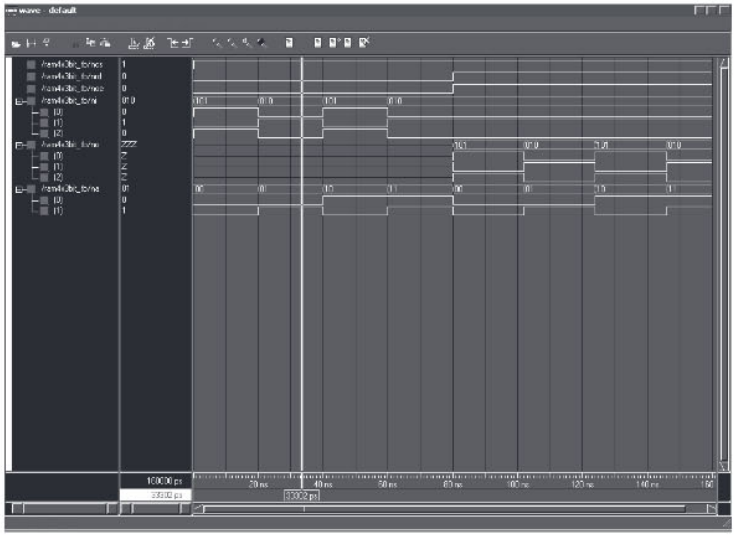


Рис. 11.19. Результаты моделирования памяти 4x3 RAM

11.4. Банки памяти на VHDL

Последним примером данной главы является VHDL-модель схемы управления банками памяти, приведенной на рис. 11.20. Сама VHDL-модель размещается на рис. 11.21.

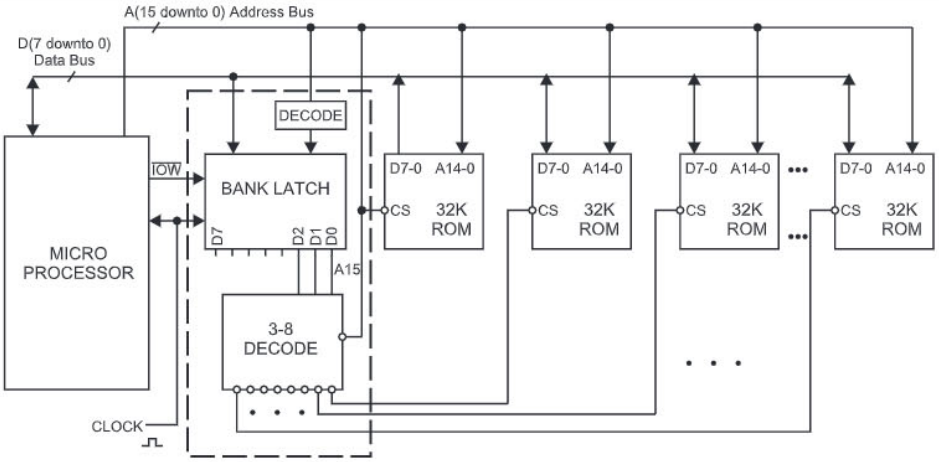


Рис. 11.20. Схема управления банками памяти

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY bank_switching_logic IS
  PORT(a        : IN  std_logic_vector(15 DOWNTO 0);
       d        : IN  std_logic_vector( 7 DOWNTO 0);
       nIOW,clk : IN  std_logic;
       set      : OUT std_logic_vector( 7 DOWNTO 0);
END bank_switching_logic;
ARCHITECTURE BSL_behave OF bank_switching_logic IS
  SIGNAL latch   : std_logic_vector( 7 DOWNTO 0);
  SIGNAL preset  : std_logic_vector( 7 DOWNTO 0);
BEGIN
  bsl : PROCESS(clk)
        BEGIN
          IF RISING_EDGE(clk) THEN
            IF a = x"0036" AND nIOW = '0' THEN
               latch <= d;
          END IF;
        END PROCESS;
        preset <= "1111 1110" WHEN latch(2 DOWNTO 0) = "000" ELSE
                  "1111 1101" WHEN latch(2 DOWNTO 0) = "001" ELSE
                  "1111 1011" WHEN latch(2 DOWNTO 0) = "010" ELSE
                  "1111 0111" WHEN latch(2 DOWNTO 0) = "011" ELSE
                  "1110 1111" WHEN latch(2 DOWNTO 0) = "100" ELSE
                  "1101 1111" WHEN latch(2 DOWNTO 0) = "101" ELSE
                  "1011 1111" WHEN latch(2 DOWNTO 0) = "110" ELSE
                  "0111 1111" WHEN OTHERS;

        set    <= preset WHEN a(15) = '1' ELSE "1111 1111";
END BSL_behave;
```

Рис. 11.21. VHDL-модель схемы управления банками памяти

Модель схемы управления банками памяти (всего их восемь) функционирует следующим образом.

По переднему фронту синхросигнала (соответствующий анализ выполняется стандартной функцией VHDL rising_edge(clk)) анализируется состояние шины адреса A (15: 0) и сигнала IOW микропроцессора.

В случае, если состояние шины адреса описывается шестнадцатиричным числом 0036, сигнал IOW находится в состоянии логического нуля, блок BANK-LATCH принимает восьмибитовое слово из шины данных

D (7 : 0). Три первых бита этого слова декодируются декодером DECODE. Выход этого декодера управляет выбором одного из восьми банков памяти (каждый емкостью 32 килобайта). В случае, если старший разряд шины адреса установлен в нуль, выбор банка данных не производится.

Глава 12. Модель микропроцессора на VHDL

12.1. Структура микропроцессора (МП) и система команд

В качестве примера проектирования цифровой системы рассмотрим написание модели «учебного» МП и его моделирование [58a, 40a]. Аналогичным образом пишется и модель микрокомпьютера [40a].

Подобные модели микропроцессоров можно найти в следующих работах:

- CPU AM2901 [66a];
- CPU SPIM [81a];
- CPU Parwan [47a];
- CPU DLX [6a].

Структура нашего 16-разрядного МП приведена на рис. 12.1.

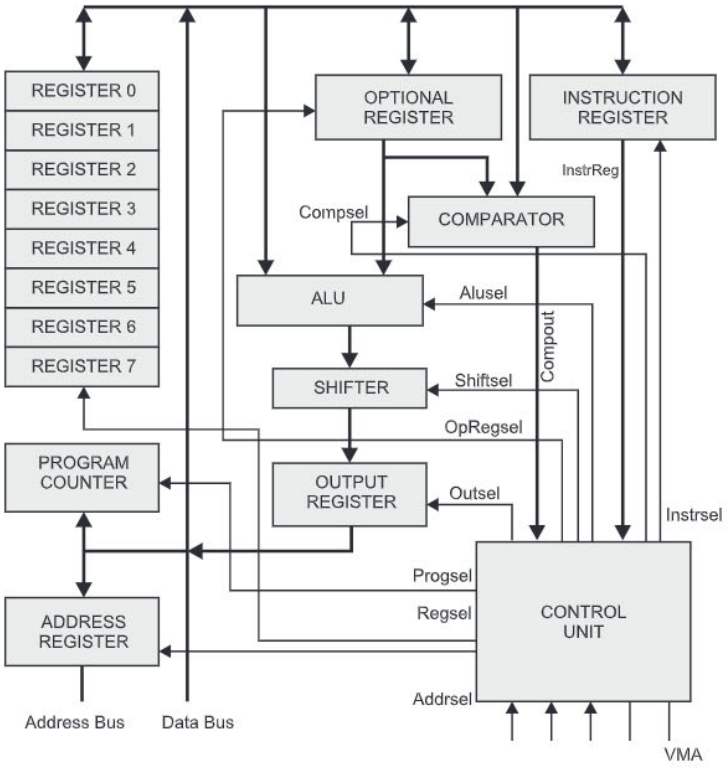


Рис. 12.1. Структура микропроцессора

Его составными частями являются:

- арифметико-логическое устройство (АЛУ) **ALU**, выполняющее ряд логических и арифметических операций;
- регистровая память **REGISTER0... REGISTER7**, содержащая 8 16-разрядных регистров, предназначенных для хранения операндов АЛУ и промежуточных результатов его работы;

- программный счетчик, указывающий на команду, подлежащую выполнению **PROGRAM COUNTER**;
- регистр команды **INSTRUCTION REGISTER**, хранящий очередную команду на стадии декодирования;
- компаратор **COMPARATOR**, проверяющий условия выполнения команд условного перехода;
- регистр операнда АЛУ **OPTIONAL REGISTER**, хранящий второй операнд для двуместных операций АЛУ;
- сдвиговый регистр **SHIFTER**, выполняющий сдвиговые операции для результатов работы АЛУ;
- регистр результата **OUTPUT REGISTER**, хранящий результат работы МП по обработке текущей команды;
- адресный регистр **ADDRESS REGISTER**, хранящий адрес ячейки при обращении к памяти;
- устройство управления **CONTROL UNIT**, являющееся источником следующих управляющих сигналов для остальных составных частей МП:
  - Addrsel: управление адресным регистром,
  - Regsel: управление регистровой памятью,
  - Progsel: управление программным счетчиком,
  - Outsel: управление регистром результата,
  - OpRegsel: управление регистром операнда АЛУ,
  - Shiftsel: управление сдвиговым регистром,
  - Alusel: управление АЛУ,
  - Compsel: управление компаратором,
  - Instrsel: управление регистром команды.

В свою очередь, устройство управления получает от компаратора информацию о проверке условий ветвления (Compout), а также код команды, подлежащей выполнению, из регистра команды (InstrReg).

В файлах проекта составные части МП носят следующие имена:

ALU	Alu
REGISTER0… REGISTER7	Reg7...Reg0
PROGRAM COUNTER	ProgCnt
INSTRUCTION REGISTER	InstrReg
COMPARATOR	Comp
OPTIONAL REGISTER	OpReg
SHIFTER	Shift
OUTPUT REGISTER	OutReg
ADDRESS REGISTER	AddrReg
CONTROL UNIT	Control

МП подключен к адресной шине Addr[15:0] и двунаправленной шине данных Data[15:0]. Кроме того, входными сигналами для МП являются:

- Ready (сигнал готовности памяти предоставить МП требуемые данные);
- Clock (синхросигнал);
- Reset (сигнал сброса),

а его выходными сигналами:

– R/W (Read/Write) (задание вида операции с памятью: чтение или запись);
– VMA (Valid Memory Address) (разрешение МП на считывание или запись данных).

Структура команд МП (содержащих одно или два слова) приведена на рис. 12.2, а на рис. 12.3 можно видеть систему команд нашего МП, составляющих его «учебный» ассемблер.

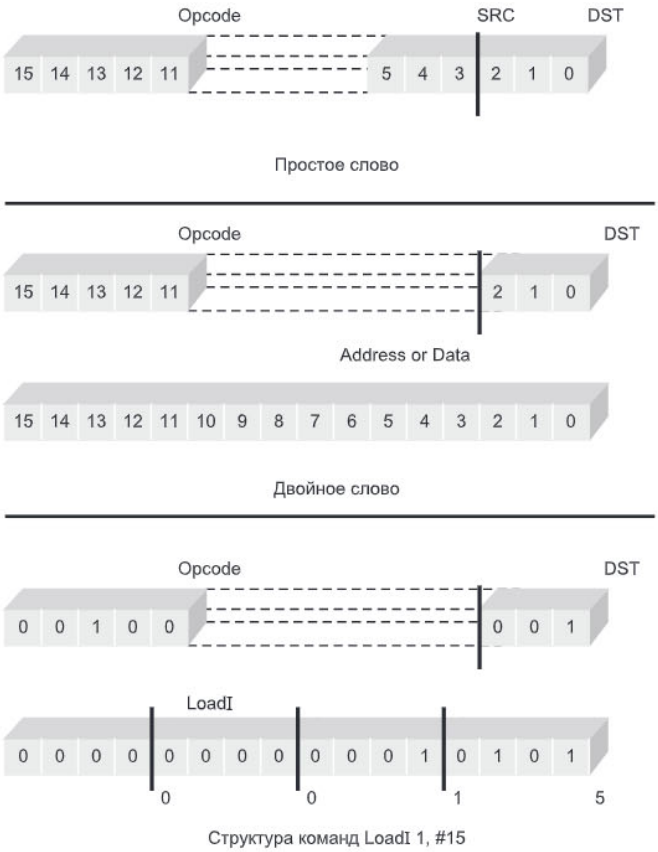


Рис. 12.2. Структура команд микропроцессора

Код операции	Команда	Комментарий
00000	NOP	Нет операции
00001	LOAD	Загрузка регистра
00010	STORE	Сохранение содержимого регистра в памяти
00011	MOVE	Пересылка значения между регистрами
00100	LOADI	Загрузка регистра непосредственным значением

Код операции	Команда	Комментарий
00101	BRANCHI	Переход к непосредственному адресу
00110	BRANCHGTI	Переход по условию «больше» к непосред. адресу
00111	INC	Инкремент
01000	DEC	Декремент
01001	AND	Логическое И для содержимого двух регистров
01010	OR	Логическое ИЛИ для содержимого двух регистров
01011	XOR	Логическое XOR для содержимого двух регистров
01100	NOT	Логическое НЕ для содержимого регистра
01101	ADD	Сложение для содержимого двух регистров
01110	SUB	Вычитание для содержимого двух регистров
01111	ZERO	Обнуление регистра
10000	BRANCHLTI	Переход по условию «меньше» к непосред. адресу
10001	BRANCHLT	Переход по условию «меньше»
10010	BRANCHNEQ	Переход по условию «не равно»
10011	BRANCHNEQI	Переход по условию «не равно» к непосред. адресу
10100	BRANCHGT	Переход по условию «больше» к непосред. адресу
10101	BRANCH	Безусловный переход
10110	BRANCHEQ	Переход по условию «равно»
10111	BRANCHEQI	Переход по условию «равно» к непосред. адресу
11000	BRANCHLTEI	Переход по условию «меньше или равно» к непосред. адресу
11001	BRANCHLTE	Переход по условию «меньше или равно»
11010	SHL	Сдвиг влево
11011	SHR	Сдвиг вправо
11100	ROTR	Циклический сдвиг вправо
11101	ROTL	Циклический сдвиг влево

Рис. 12.3. Система команд микропроцессора

12.2. Структура проекта

Наш проект с именем CPU содержит 11 следующих файлов VHDL-кода:

- Top.vhd: содержит модель верхнего уровня проекта (см. п.12.3);
- Mem.vhd: содержит модель внешней памяти для МП (см. п.12.4);
- Cpulib.vhd: содержит глобальные определения для проекта (см. п.12.5);
- Cpu.vhd: содержит структурную модель МП на уровне его компонент (см. п.12.6);
- Alu.vhd: содержит модель АЛУ МП (см. п.12.7);

- Comp.vhd: содержит модель компаратора МП (см. п.12.8);
- Control.vhd: содержит модель управляющего устройства МП (см. п.12.9);
- Reg.vhd: содержит модель регистра команды и регистра адреса МП (см. п.12.10);
- Trireg.vhd: содержит модель программного счетчика, регистра операнда АЛУ и регистра результата МП (см. п.12.11);
- Regarray.vhd: содержит модель массива регистров МП (см. п.12.12);
- Shift.vhd: содержит модель сдвигового регистра МП (см. п.12.13).

12.3. Модель верхнего уровня проекта

Для моделирования проекта CPU (CPU Design) мы не предусматриваем написание испытательной программы (Test Bench), которая явилась бы источником внешних воздействий (Outside Stimulus). Вместо этого в состав проекта CPU входит файл верхнего уровня проекта (Top Level of CPU Design) (рис. 12.4, *а* и *б*) — Top.vhd, который содержит два компонента:

- CPU (структурная модель собственно МП);
- MEM (модель внешней памяти).

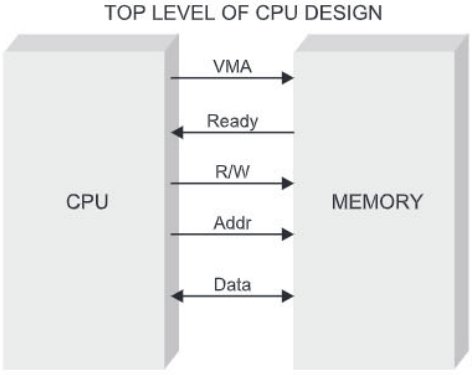


Рис. 12.4. Верхний уровень проекта (Top Level of CPU Design)

```
-------------------------------------------------------
-------
-- TOP
-------------------------------------------------------
-------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpu_lib.ALL;
ENTITY top IS
END top;

--  TOP LEVEL OF CPU DESIGN - ENTITY top
```

```
--                 _________                    _________
--                |CPU      |                  |MEMORY   |
--                |         |_____VMA_________|         |
--                |         |                  |         |
-- reset _____    |         |_____Ready_______|         |
--                |         |                  |         |
--                |         |_____R/W_________|         |
-- clock _____    |         |                  |         |
--                |         |_____Addr________|         |
--                |         |                  |         |
--                |         |_____Data________|         |
--                |_________|                  |_________|

ARCHITECTURE behave OF top IS
  COMPONENT mem                                        -- MEMORY COMPO-
NENT
    PORT (addr               : IN    bit16;
          sel, rw            : IN    std_logic;
          ready              : OUT   std_logic;
          data               : INOUT bit16);
  END COMPONENT;

  COMPONENT cpu                                        -- CPU COMPONENT
    PORT(clock, reset, ready : IN    std_logic;
         addr                : OUT   bit16;
         rw, vma             : OUT   std_logic;
         data                : INOUT bit16);
  END COMPONENT;

  SIGNAL addr, data          : bit16;
  SIGNAL vma, rw, ready      : std_logic;
  SIGNAL clock, reset        : std_logic := '0';

BEGIN
  clock <= NOT clock AFTER 50 ns;
  reset <= '1', '0'  AFTER 100 ns;
  m1 : mem PORT MAP (addr, vma, rw, ready, data);
  u1 : cpu PORT MAP (clock, reset, ready, addr, rw, vma, data);

END behave;
```

Рис. 12.4,а. Содержимое файла top.vhd (модель верхнего уровня проекта)

Именно модель внешней памяти обеспечит внешние воздействия (входную информацию — поток команд) для модели CPU при выполнении процесса моделирования.

12.4. Модель внешней памяти для МП

В модели внешней памяти (файл Mem.vhd — рис. 12.6, *б*) располагается программа, которая выполняет процедуру копирования блока данных из массива-источника (Source Array) в массив-приемник (Destination Array) (рис. 12.5).

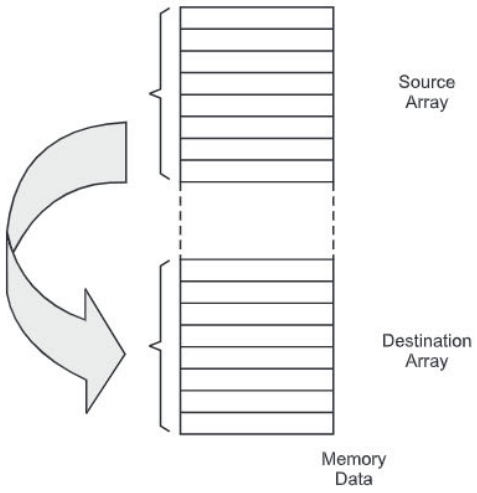


Рис. 12.5. Операция копирования блока данных (Block Copy Operation)

Эта программа написана на ассемблере нашего МП и выглядит следующим образом (рис. 12.6, *а* и *б*) с нашими комментариями:

Адрес	Содержимое ячейки	Комментарий
00	4001	loadl 1,#
01	0010	10
02	2002	loadl 2,#
03	0030	30
04	2006	loadl 6,#
05	002F	2F
06	080B	load 1,3
07	101A	store 3,2
08	300E	bgtl 1,6,#
09	0000	00
0A	3801	inc 1
0B	3802	inc 2
0C	280F	bral #
0D	0006	06
0E	0000	

Программа копирования блока данных

Адрес	Содержимое ячейки	Комментарий
0F	0000	
10	0001	Массив-источник (Source Array)
11	0002	
12	0003	
13	0004	
14	0005	
15	0006	
16	0007	
17	0008	
18	0009	
19	000A	
20	000B	
21	000C	
22	000D	
23	000E	
24	000F	
25	0010	
26	0000	
27	0000	
28	0000	
29	0000	
2A	0000	
2B	0000	
2C	0000	
2D	0000	
2E	0000	
2F	0000	
30	0000	Массив-приемник (Destination Array)
31	0000	
32	0000	
33	0000	
34	0000	
35	0000	

Адрес	Содержимое ячейки	Комментарий
36	0000	
37	0000	
38	0000	
39	0000	
3A	0000	
3B	0000	
3C	0000	
3D	0000	
3E	0000	
3F	0000	

Рис. 12.6,а. Структура модели памяти на верхнем уровне проекта

```
------------------------------------------------------------------
------
-- MEM
------------------------------------------------------------------
------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.cpu_lib.all;
------------------------------------------------------------------
------
                                  --  MEMORY INTERFACE DESCRIPTION - ENTITY
mem
                                  --              _________
ENTITY mem IS                     --             |MEMORY   |
  PORT( addr     : in  bit16;     -- addr_______|         |
        sel, rw  : in std_logic;  --             |         |
        ready    : out std_logic; -- data_______|         |_______ready
        data     : inout bit16);  --             |         |
END mem;                          --    rw_______|         |
                                  --             |_________|
                                  --                  |
                                  --                  |
                                  --                 sel
                                  --
------------------------------------------------------------------
------

architecture behave of mem is
begin
  memproc: process(addr, sel, rw)
    type t_mem is array(0 to 63) of bit16;
    variable mem_data : t_mem :=
```

```
------------------------------------------------------------------
------
-- DATA COPY PROGRAM
------------------------------------------------------------------
------
     ("0010000000000001",  --- 0  loadI 1, #    -- load source address
      "0000000000010000",  --- 1  10
      "0010000000000010",  --- 2  loadI 2, #   -- load destination address
      "0000000000110000",  --- 3  30
      "0010000000000110",  --- 4  loadI 6, #   -- load data end address
      "0000000000101111",  --- 5  2F
      "0000100000001011",  --- 6  load 1, 3    -- load reg3 with source
element
      "0001000000011010",  --- 7  store 3, 2   -- store reg3 at destination
      "0011000000001110",  --- 8  bgtI 1, 6, # -- compare to see if at end
of data
      "0000000000000000",  --- 9  00           -- if so just start over
      "0011100000000001",  --- A  inc 1        -- move source address to
next
      "0011100000000010",  --- B  inc 2        -- move destination address
to next
      "0010100000001111",  --- C  braI #       -- go to the next element to
copy
      "0000000000000110",  --- D  06
      "0000000000000000",  --- E
      "0000000000000000",  --- F
------------------------------------------------------------------
------
-- SOURCE ARRAY
------------------------------------------------------------------
------
      "0000000000000001",  --- 10  --- Start of source array
      "0000000000000010",  --- 11
      "0000000000000011",  --- 12
      "0000000000000100",  --- 13
      "0000000000000101",  --- 14
      "0000000000000110",  --- 15
      "0000000000000111",  --- 16
      "0000000000001000",  --- 17
      "0000000000001001",  --- 18
      "0000000000001010",  --- 19
      "0000000000001011",  --- 1A
      "0000000000001100",  --- 1B
      "0000000000001101",  --- 1C
      "0000000000001110",  --- 1D
      "0000000000001111",  --- 1E
      "0000000000010000",  --- 1F
      "0000000000000000",  --- 20
      "0000000000000000",  --- 21
      "0000000000000000",  --- 22
      "0000000000000000",  --- 23
      "0000000000000000",  --- 24
      "0000000000000000",  --- 25
      "0000000000000000",  --- 26
      "0000000000000000",  --- 27
```

```
      "0000000000000000",  --- 28
      "0000000000000000",  --- 29
      "0000000000000000",  --- 2A
      "0000000000000000",  --- 2B
      "0000000000000000",  --- 2C
      "0000000000000000",  --- 2D
      "0000000000000000",  --- 2E
      "0000000000000000",  --- 2F
-----------------------------------------------------------------
------
-- DESTINATION ARRAY
-----------------------------------------------------------------
------
      "0000000000000000",  --- 30  --- Start of destination array
      "0000000000000000",  --- 31
      "0000000000000000",  --- 32
      "0000000000000000",  --- 33
      "0000000000000000",  --- 34
      "0000000000000000",  --- 35
      "0000000000000000",  --- 36
      "0000000000000000",  --- 37
      "0000000000000000",  --- 38
      "0000000000000000",  --- 39
      "0000000000000000",  --- 3A
      "0000000000000000",  --- 3B
      "0000000000000000",  --- 3C
      "0000000000000000",  --- 3D
      "0000000000000000",  --- 3E
      "0000000000000000"); --- 3F
  begin
    data <= "ZZZZZZZZZZZZZZZZ";
    ready <= '0';
    if sel = '1' then
      if rw = '0' then
        data <= mem_data(CONV_INTEGER(addr(15 downto 0))) after 1 ns;
        ready <= '1';
      elsif rw = '1' then
        mem_data(CONV_INTEGER(addr(15 downto 0))) := data;
      end if;
    else
      data <= "ZZZZZZZZZZZZZZZZ" after 1 ns;
    end if;
  end process;

end behave;
```

Рис. 12.6,б. Содержимое файла mem.vhd (модель внешней памяти для МП)

Адрес	Команда	Комментарий
00	LoadI 1, #10	Загрузка адреса массива-источника в регистр 1
02	LoadI 2, #30	Загрузка адреса массива-приемника в регистр 2
04	LoadI 6, #2F	Загрузка адреса последней ячейки данных в регистр 6
06	LoadI 1, 3	Загрузка очередного элемента данных в регистр 3

Адрес	Команда	Комментарий
07	Store 3, 2	Сохранение очередного элемента данных в массиве-приемнике
08	Bgt 1, 6, #00	В случае, если все элементы данных массива-источника уже скопированы, осуществляется переход на начало программы; в противном случае, управление передается команде с адресом 0A
0A	Inc 1	Переход к следующему адресу в массиве-источника
0B	Inc 2	Переход к следующему адресу в массиве-приемника
0C	Bral #06	Переход к процедуре копирования следующего элемента

Интерфейс памяти (entity mem) содержит следующие сигналы:

addr: 16-разрядное слово адреса, позволяющее обратиться к нужной ячейке памяти;

sel: если данный сигнал разрешения равен единице, то возможно чтение данных из памяти; в противном случае на выходах памяти устанавливается третье состояние;

rw: задает вид операции с памятью: чтение (rw=0) или запись (rw=1);

ready: устанавливается в единицу, когда память готова предоставить выбранные из определенной ячейки данные МП;

data: 16-разрядное слово данных, предназначенное для записи в определенную ячейку или полученное в результате считывания из определенной ячейки.

Состояние памяти отображается с помощью переменной mem_data, существующей внутри процесса memproc. Эта переменная является массивом, содержащим 64 16-разрядных слова (ячейки памяти). Номер ячейки при обращении к памяти определяется с помощью стандартной функции CONV_INTEGER, преобразующей 16-разрядный бинарный вектор addr в целое число в диапазоне от 0 до 63. Эта функция находится в стандартном пакете std_logic_unsigned, присоединенном к файлу Mem.vhd (см. приложение В).

12.5. Пакет глобальных определений проекта

Файл Cpulib.vhd содержит типы и подтипы данных, а также константы, определенные специально для нашего проекта (рис. 12.7).

```
------------------------------------------------------------------
----------------------------------
-- CPULIB
------------------------------------------------------------------
----------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
PACKAGE cpu_lib IS
------------------------------------------------------------------
----------------------------------
-- DATA TYPE FOR SHIFTREG
```

```
-------------------------------------------------------------------
-----------------------------------
  TYPE     t_shift IS (shftpass, shl, shr, rotl, rotr);
-------------------------------------------------------------------
-----------------------------------
-- DATA TYPE AND CONSTANTS FOR ALU
-------------------------------------------------------------------
-----------------------------------
  SUBTYPE  t_alu   IS unsigned(3 DOWNTO 0);

  CONSTANT alupass  : unsigned(3 DOWNTO 0) := "0000";
  CONSTANT andOp    : unsigned(3 DOWNTO 0) := "0001";
  CONSTANT orOp     : unsigned(3 DOWNTO 0) := "0010";
  CONSTANT notOp    : unsigned(3 DOWNTO 0) := "0011";
  CONSTANT xorOp    : unsigned(3 DOWNTO 0) := "0100";
  CONSTANT plus     : unsigned(3 DOWNTO 0) := "0101";
  CONSTANT alusub   : unsigned(3 DOWNTO 0) := "0110";
  CONSTANT inc      : unsigned(3 DOWNTO 0) := "0111";
  CONSTANT dec      : unsigned(3 DOWNTO 0) := "1000";
  CONSTANT zero     : unsigned(3 DOWNTO 0) := "1001";
-------------------------------------------------------------------
-----------------------------------
-- DATA TYPE FOR COMP
-------------------------------------------------------------------
-----------------------------------
  TYPE     t_comp  IS (eq, neq, gt, gte, lt, lte);
-------------------------------------------------------------------
-----------------------------------
-- DATA TYPE FOR REG
-------------------------------------------------------------------
-----------------------------------
  SUBTYPE  t_reg   IS std_logic_vector(2 DOWNTO 0);
-------------------------------------------------------------------
-----------------------------------
-- DATA TYPE FOR CONTROL
-------------------------------------------------------------------
  TYPE     state   IS (reset1, reset2, reset3, reset4, reset5, reset6,
                       execute, nop, load, store, move, load2, load3,
                       load4, store2, store3, store4, move2, move3, mo-
ve4,
                       incPc, incPc2, incPc3, incPc4, incPc5, incPc6,
                       loadPc, loadPc2, loadPc3, loadPc4, bgtI2, bgtI3,
                       bgtI4, bgtI5, bgtI6, bgtI7, bgtI8, bgtI9,
bgtI10,
                       braI2, braI3, braI4, braI5, braI6, loadI2,
                       loadI3, loadI4, loadI5, loadI6, inc2, inc3,
inc4);
  SUBTYPE  bit16   IS std_logic_vector(15 DOWNTO 0);
end cpu_lib;
```

Рис. 12.7. Содержимое файла cpulib.vhd (пакет глобальных определений проекта)

Перечисленный тип данных t_shift позволяет задать вид операции для сдвигового регистра Shifter (см. также п.12.13):

- shftpass: нет сдвиговой операции;
- shl: сдвиг влево;
- shr: сдвиг вправо;
- rotl: циклический сдвиг влево;
- rotr: циклический сдвиг вправо.

С помощью типа данных t_alu и констант:

- alupass (нет операции);
- andOp (логическая операция И);
- orOp (логическая операция ИЛИ);
- notOp (логическая операция НЕ);
- xorOp (логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ);
- plus (арифметическая операция сложения);
- alusub (арифметическая операция вычитания);
- inc (инкремент);
- dec (декремент);
- zero (сброс выхода АЛУ в ноль) задается тип операции АЛУ МП (см. также п.12.7).

Перечисленный тип данных t_comp позволяет управлять компаратором (см. также п.12.8):

- eq (выход компаратора compout устанавливается в 1, если два операнда a и b равны);
- neq (выход компаратора compout устанавливается в 1, если два операнда a и b не равны);
- gt (выход компаратора compout устанавливается в 1, если операнд a больше операнда b);
- gte (выход компаратора compout устанавливается в 1, если операнд a больше или равен операнду b);
- lt (выход компаратора compout устанавливается в 1, если операнд a меньше операнда b);
- lte (выход компаратора compout устанавливается в 1, если операнд a меньше или равен операнду b).

С помощью подтипа данных t_reg осуществляется выбор нужного регистра в массиве из 8 регистров Regarray (см. также п.12.12). Перечисленный тип данных state задает все возможные состояния управляющего устройства МП (см. также п.12.9).

И наконец, подтип данных bit16 определяет 16-разрядный вектор с 9 возможными состояниями (std_logic), повсеместно используемый в данном проекте.

12.6. Модель МП

Файл Cpu.vhd содержит структурную модель МП, состоящую из следующих компонент (рис. 12.8):

- regarray (модель регистровой памяти (массива регистров Reg0...Reg7));
- reg (модель регистра адреса и регистра команды МП);

- trireg (модель регистра операнда АЛУ и программного счетчика МП);
- control (модель управляющего устройства МП);
- alu (модель АЛУ МП);
- shift (модель сдвигающего регистра МП);
- comp (модель компаратора МП),

соединенных между собой с помощью внутренних сигналов и операторов реализации этих компонент port map (Component Instantiation Statements) согласно структурной схеме МП, приведенной на рис. 12.1.

```
-------------------------------------------------------------------
--------------------------------
-- CPULIB
-------------------------------------------------------------------
--------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
PACKAGE cpu_lib IS
-------------------------------------------------------------------
--------------------------------
-- DATA TYPE FOR SHIFTREG
-------------------------------------------------------------------
--------------------------------
  TYPE     t_shift IS (shftpass, shl, shr, rotl, rotr);
-------------------------------------------------------------------
--------------------------------
-- DATA TYPE AND CONSTANTS FOR ALU
-------------------------------------------------------------------
--------------------------------
  SUBTYPE  t_alu   IS unsigned(3 DOWNTO 0);
  CONSTANT alupass  : unsigned(3 DOWNTO 0) := "0000";
  CONSTANT andOp    : unsigned(3 DOWNTO 0) := "0001";
  CONSTANT orOp     : unsigned(3 DOWNTO 0) := "0010";
  CONSTANT notOp    : unsigned(3 DOWNTO 0) := "0011";
  CONSTANT xorOp    : unsigned(3 DOWNTO 0) := "0100";
  CONSTANT plus     : unsigned(3 DOWNTO 0) := "0101";
  CONSTANT alusub   : unsigned(3 DOWNTO 0) := "0110";
  CONSTANT inc      : unsigned(3 DOWNTO 0) := "0111";
  CONSTANT dec      : unsigned(3 DOWNTO 0) := "1000";
  CONSTANT zero     : unsigned(3 DOWNTO 0) := "1001";
-------------------------------------------------------------------
--------------------------------
-- DATA TYPE FOR COMP
-------------------------------------------------------------------
--------------------------------
  TYPE     t_comp  IS (eq, neq, gt, gte, lt, lte);
-------------------------------------------------------------------
--------------------------------
-- DATA TYPE FOR REG
-------------------------------------------------------------------
```

```
------------------------------
  SUBTYPE  t_reg   IS std_logic_vector(2 DOWNTO 0);
-------------------------------------------------------------
------------------------------
-- DATA TYPE FOR CONTROL
-------------------------------------------------------------
------------------------------
  TYPE     state   IS (reset1, reset2, reset3, reset4, reset5, reset6,
                       execute, nop, load, store, move, load2, load3,
                       load4, store2, store3, store4, move2, move3,
                       move4,
                       incPc, incPc2, incPc3, incPc4, incPc5, incPc6,
                       loadPc, loadPc2, loadPc3, loadPc4, bgtI2, bgtI3,
                       bgtI4, bgtI5, bgtI6, bgtI7, bgtI8, bgtI9,
                       bgtI10,
                       braI2, braI3, braI4, braI5, braI6, loadI2,
                       loadI3, loadI4, loadI5, loadI6, inc2, inc3,
                       inc4);
  SUBTYPE  bit16   IS std_logic_vector(15 DOWNTO 0);
end cpu_lib;
```

Рис. 12.8. Содержимое файла cpu.vhd (модель МП)

12.7. Модель АЛУ МП

Модель АЛУ МП содержится в файле Alu.vhd (рис. 12.9).

```
-------------------------------------------------------------------
---------------------------------
-- ALU
-------------------------------------------------------------
---------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE work.cpu_lib.ALL;
-------------------------------------------------------------
---------------------------------
                                --  ALU INTERFACE DESCRIPTION -
                                --  ENTITY alu
                                --
                                --            _________
ENTITY alu IS                   --           | ALU     |
  PORT( a, b : IN  bit16;       --    a ______|         |
        sel  : IN  t_alu;       --            |         |______c
        c    : OUT bit16);      --            |         |
END alu;                        --    b ______|         |
                                --            |_________|
                                --                 |
                                --                 |
                                --                sel
```

```
-------------------------------------------------------------------
-----------------------------------
--   ALU FUNCTION TABLE
-------------|---------------------------------
-- SEL INPUT |    OPERATION
-------------|---------------------------------
--   0000    |    C=A          -- NOP
--   0001    |    C=A and B    -- AND
--   0010    |    C=A or  B    -- OR
--   0011    |    C=not A      -- NOT
--   0100    |    C=A xor B    -- XOR
--   0101    |    C=A + B      -- ADDITION
--   0110    |    C=A - B      -- SUBTRACTION
--   0111    |    C=A + 1      -- INCREMENT
--   1000    |    C=A - 1      -- DECREMENT
--   1001    |    C=0          -- C OUTPUT SET IN 0
-------------------------------------------------------------------
-----------------------------------
ARCHITECTURE rtl OF alu IS
BEGIN
  aluproc: PROCESS(a, b, sel)
  BEGIN
  CASE sel IS
-------------------------------------------------------------------
-----------------------------------
    WHEN alupass =>               -- NOP IN ALU
      c <= a                      AFTER 1 ns;
-------------------------------------------------------------------
-----------------------------------
    WHEN andOp =>                 -- EXECUTION OF LOGICAL OPERATION
                                  -- "AND"
      c <= a AND b                AFTER 1 ns;
-------------------------------------------------------------------
-----------------------------------
    WHEN orOp =>                  -- EXECUTION OF LOGICAL OPERATION
                                  --"OR"
      c <= a OR b                 AFTER 1 ns;
-------------------------------------------------------------------
-----------------------------------
    WHEN xorOp =>                 -- EXECUTION OF LOGICAL OPERATION
                                  -- "XOR"
      c <= a XOR b                AFTER 1 ns;
-------------------------------------------------------------------
-----------------------------------
    WHEN notOp =>                 -- EXECUTION OF LOGICAL OPERATION
                                  -- "AND"
      c <= NOT a                  AFTER 1 ns;
-------------------------------------------------------------------
-----------------------------------
```

```
    WHEN plus =>                    -- EXECUTION OF ARITHMETIC OPERATION
                                    -- "ADDITION"
      c <= a + b                    AFTER 1 ns;
-------------------------------------------------------------------
---------------------------------
    WHEN alusub =>                  -- EXECUTION OF ARITHMETIC OPERATION
                                    -- "SUBTRACTION"
      c <= a - b                    AFTER 1 ns;
-------------------------------------------------------------------
---------------------------------
    WHEN inc =>                     -- EXECUTION OF INCREMENT
      c <= a + "0000000000000001" AFTER 1 ns;
-------------------------------------------------------------------
---------------------------------
    WHEN dec =>                     -- EXECUTION OF DECREMENT
      c <= a - "0000000000000001" AFTER 1 ns;
-------------------------------------------------------------------
---------------------------------
    WHEN zero =>                    -- TO SET C IN 0
      c <=     "0000000000000000" AFTER 1 ns;
-------------------------------------------------------------------
---------------------------------
    WHEN OTHERS =>
      c <=     "0000000000000000" AFTER 1 ns;
-------------------------------------------------------------------
---------------------------------
  END CASE;
  END PROCESS;
END rtl;
```

Рис. 12.9. Содержимое файла alu.vhd (модель АЛУ МП)

Интерфейс АЛУ (entity alu) содержит следующие сигналы:

- a,b: два операнда АЛУ, представляющих собой два 16-разрядных бинарных вектора;
- sel: управляющий сигнал, определяющий операцию АЛУ (рис. 12.9);
- c: результат операции АЛУ (также 16-разрядный бинарный вектор).

Архитектурное тело модели содержит процесс aluproc, в котором последовательный оператор CASE анализирует сигнал sel и в соответствии с его значением выполняет необходимую операцию АЛУ.

12.8. Модель компаратора МП

Модель компаратора МП содержится в файле Comp.vhd (рис. 12.10).

```
-------------------------------------------------------------------
----------
-- COMP
-------------------------------------------------------------------
----------
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE work.cpu_lib.ALL;
-------------------------------------------------------------------
----------
                                  --  COMPARATOR INTERFACE
                                  --  DESCRIPTION – ENTITY comp
                                  --                 _________
ENTITY comp IS                    --               | COMP    |
  PORT( a, b    : IN  bit16;      --    a _______|           |
        sel     : IN  t_comp;     --              |           |_____ compout
       compout : OUT std_logic); --              |           |
END comp;                         --    b _______|           |
                                  --              |_________|
                                  --                   |
                                  --                   |
                                  --                  sel
-------------------------------------------------------------------
----------
--   COMPARATOR OPERATION TABLE
-------------|-----------------------------------
-- SEL INPUT |    COMPARISON
-------------|-----------------------------------
--   EQ      |    compout = 1 when a equals b
--   NEQ     |    compout = 1 when a is not equal b
--   GT      |    compout = 1 when a is greater than b
--   GTE     |    compout = 1 when a is greater than or equal to b
--   LT      |    compout = 1 when a is less than b
--   LTE     |    compout = 1 when a is less than or equal to b
-------------------------------------------------------------------
----------
ARCHITECTURE rtl OF comp IS
BEGIN
  compproc: PROCESS(a, b, sel)
  BEGIN
    CASE sel IS
-------------------------------------------------------------------
----------
      WHEN eq =>                      -- SEL = EQ
        IF a = b THEN
          compout <= '1' AFTER 1 ns;
        ELSE
          compout <= '0' AFTER 1 ns;
        END IF;
-------------------------------------------------------------------
----------
      WHEN neq =>                     -- SEL = NEQ
        IF a /= b THEN
          compout <= '1' AFTER 1 ns;
```

```
        ELSE
          compout <= '0' AFTER 1 ns;
        END IF;
-------------------------------------------------------------------
----------
      WHEN gt =>                        -- SEL = GT
        IF a > b THEN
          compout <= '1' AFTER 1 ns;
        ELSE
          compout <= '0' AFTER 1 ns;
        END IF;
-------------------------------------------------------------------
----------
      WHEN gte =>                       -- SEL = GTE
        IF a >= b THEN
          compout <= '1' AFTER 1 ns;
        ELSE
          compout <= '0' AFTER 1 ns;
        END IF;
-------------------------------------------------------------------
----------
      WHEN lt =>                        -- SEL = LT
        IF a < b THEN
          compout <= '1' AFTER 1 ns;
        ELSE
          compout <= '0' AFTER 1 ns;
        END IF;
-------------------------------------------------------------------
----------
      WHEN lte =>                       -- SEL = LTE
        IF a <= b THEN
          compout <= '1' AFTER 1 ns;
        ELSE
          compout <= '0' AFTER 1 ns;
        END IF;
-------------------------------------------------------------------
----------
    END CASE;
  END PROCESS;
END rtl;
```

Рис. 12.10. Содержимое файла comp.vhd (модель компаратора МП)

Интерфейс компаратора (entity comp) содержит следующие сигналы:

- a, b: два операнда компаратора, представляющих собой два 16-разрядных бинарных вектора;
- sel: управляющий сигнал, определяющий операцию сравнения компаратора, имеющий тип данных t_comp (см. п.12.5);
- compout: результат сравнения двух операндов (см. п.12.5 и рис. 12.10).

Архитектурное тело компаратора содержит процесс compproc, внутри которого оператор CASE анализирует сигнал sel и в соответствии с результатом анализа устанавливает выход компаратора compout в единицу или в ноль.

12.9. Модель управляющего устройства МП

Модель управляющего устройства МП содержится в файле Control.vhd (рис. 12.11). Здесь, прежде всего, описан его интерфейс (entity control).

```
-------------------------------------------------------------------
-- CONTROL
-------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpu_lib.ALL;
-------------------------------------------------------------------
-- CONTROL UNIT INTERFACE DESCRIPTION -  ENTITY control
-------------------------------------------------------------------
ENTITY control IS                    --                 ___________
 PORT( clock      : IN std_logic;    --  clock     ___|           |
      reset       : IN std_logic;    --  reset     ___| CONTROL   |
      instrReg    : IN bit16;        --  instrReg  ___|           |
       compout    : IN std_logic;    --  compout   ___|           |
       ready      : IN std_logic;    --  ready     ___|           |
       progCntrWr : OUT std_logic;   --                |           |____progCntrWr
       progCntrRd : OUT std_logic;   --                |           |____progCntrRd
       addrRegWr  : OUT std_logic;   --                |           |____addrRegWr
       addrRegRd  : OUT std_logic;   --                |           |____addrRegRd
       outRegWr   : OUT std_logic;   --                |           |____outRegWr
       outRegRd   : OUT std_logic;   --                |           |____outRegRd
       shiftSel   : OUT t_shift;     --                |           |____shiftSel
       aluSel     : OUT t_alu;       --                |           |____aluSel
       compSel    : OUT t_comp;      --                |           |____compSel
       opRegRd    : OUT std_logic;   --                |           |____opRegRd
       opRegWr    : OUT std_logic;   --                |           |____opRegWr
       instrWr    : OUT std_logic;   --                |           |____instrWr
       regSel     : OUT t_reg;       --                |           |____regSel
       regRd      : OUT std_logic;   --                |           |____regRd
       regWr      : OUT std_logic;   --                |           |____regWr
       rw         : OUT std_logic;   --                |           |____rw
       vma        : OUT std_logic    --                |           |____vma
       );                            --                |___________|
END control;
-------------------------------------------------------------------
ARCHITECTURE rtl OF control IS
  SIGNAL current_state, next_state : state;
BEGIN
  nxtstateproc: PROCESS( current_state, instrReg, compout, ready)
  BEGIN
-------------------------------------------------------------------
    progCntrWr <= '0';                                    -- CONTROL UNIT
INITIALIZATION
    progCntrRd <= '0';
    addrRegWr  <= '0';
    outRegWr   <= '0';
    outRegRd   <= '0';
```

```
    shiftSel   <= shftpass;
    aluSel     <= alupass;
    compSel    <= eq;
    opRegRd    <= '0';
    opRegWr    <= '0';
    instrWr    <= '0';
    regSel     <= "000";
    regRd      <= '0';
    regWr      <= '0';
    rw         <= '0';
    vma        <= '0';
-----------------------------------------------------------------
    CASE current_state IS                                  -- RESET PROCESS
      WHEN reset1 =>
         aluSel <= zero AFTER 1 ns;
         shiftSel <= shftpass;
         next_state <= reset2;
      WHEN reset2 =>
        aluSel <= zero;
        shiftSel <= shftpass;
        outRegWr <= '1';
        next_state <= reset3;
      WHEN reset3 =>
        outRegRd <= '1';
        next_state <= reset4;
      WHEN reset4 =>
         outRegRd <= '1';
         progCntrWr <= '1';
         addrRegWr <= '1';
         next_state <= reset5;
      WHEN reset5 =>
         vma <= '1';
         rw <= '0';
         next_state <= reset6;
      WHEN reset6 =>
         vma <= '1';
         rw <= '0';
         IF ready = '1' THEN
           instrWr <= '1';
           next_state <= execute;
         ELSE
           next_state <= reset6;
         END IF;
-----------------------------------------------------------------
      WHEN execute =>                                      -- COMMAND
                                                           -- EXECUTION
        CASE instrReg(15 DOWNTO 11) IS
          WHEN "00000" =>                                  -- NOP
            next_state <= incPc;
          WHEN "00001" =>                                  -- LOAD
            regSel <= instrReg(5 DOWNTO 3);
            regRd <= '1';
            next_state <= load2;

          WHEN "00010" =>                                  -- STORE
            regSel <= instrReg(2 DOWNTO 0);
            regRd <= '1';
            next_state <= store2;
          WHEN "00011" =>                                  -- MOVE
```

```
            regSel <= instrReg(5 DOWNTO 3);
            regRd <= '1';
            aluSel <= alupass;
            shiftSel <= shftpass;
            next_state <= move2;
          WHEN "00100" =>                                    -- LOADI
            progcntrRd <= '1';
            alusel <= inc;
            shiftsel <= shftpass;
            next_state <= loadI2;
          WHEN "00101" =>                                    -- BRANCHIMM
            progcntrRd <= '1';
            alusel <= inc;
            shiftsel <= shftpass;
            next_state <= braI2;
          WHEN "00110" =>                                    -- BRANCHGTIMM
            regSel <= instrReg(5 DOWNTO 3);
            regRd <= '1';
            next_state <= bgtI2;
          WHEN "00111" =>                                    -- INC
            regSel <= instrReg(2 DOWNTO 0);
            regRd <= '1';
            alusel <= inc;
            shiftsel <= shftpass;
            next_state <= inc2;

          WHEN OTHERS =>
            next_state <= incPc;
        END CASE;
-----------------------------------------------------------------------
-- LOAD REGISTER
-----------------------------------------------------------------------
      WHEN load2 =>
         regSel <= instrReg(5 DOWNTO 3);
         regRd <= '1';
         addrregWr <= '1';
         next_state <= load3;
      WHEN load3 =>
        vma <= '1';
        rw <= '0';
        next_state <= load4;
      WHEN load4 =>
        vma <= '1';
        rw <= '0';
        regSel <= instrReg(2 DOWNTO 0);
        regWr <= '1';
        next_state <= incPc;
-----------------------------------------------------------------------
-- STORE REGISTER
-----------------------------------------------------------------------
      WHEN store2 =>
        regSel <= instrReg(2 DOWNTO 0);
        regRd <= '1';
        addrregWr <= '1';
        next_state <= store3;
      WHEN store3 =>
        regSel <= instrReg(5 DOWNTO 3);
        regRd <= '1';
        next_state <= store4;
```

```
      WHEN store4 =>
        regSel <= instrReg(5 DOWNTO 3);
        regRd <= '1';
        vma <= '1';
        rw <= '1';
        next_state <= incPc;
-----------------------------------------------------------------
-- MOVE VALUE TO REGISTER
-----------------------------------------------------------------
      WHEN move2 =>
        regSel <= instrReg(5 DOWNTO 3);
        regRd <= '1';
        aluSel <= alupass;
        shiftsel <= shftpass;
        outRegWr <= '1';
        next_state <= move3;
      WHEN move3 =>
        outRegRd <= '1';
        next_state <= move4;
     WHEN move4 =>
       outRegRd <= '1';
       regSel <= instrReg(2 DOWNTO 0);
       regWr <= '1';
       next_state <= incPc;
-----------------------------------------------------------------
-- LOAD REGISTER WITH IMMEDIATE VALUE
-----------------------------------------------------------------
      WHEN loadI2 =>
        progcntrRd <= '1';
        alusel <= inc;
        shiftsel <= shftpass;
        outregWr <= '1';
        next_state <= loadI3;
      WHEN loadI3 =>
        outregRd <= '1';
        next_state <= loadI4;
      WHEN loadI4 =>
        outregRd <= '1';
        progcntrWr <= '1';
        addrregWr <= '1';
        next_state <= loadI5;
      WHEN loadI5 =>
        vma <= '1';
        rw <= '0';
        next_state <= loadI6;
      WHEN loadI6 =>
        vma <= '1';
        rw <= '0';
        IF ready = '1' THEN
          regSel <= instrReg(2 DOWNTO 0);
          regWr <= '1';
          next_state <= incPc;
        ELSE
          next_state <= loadI6;
        END IF;
-----------------------------------------------------------------
-- BRANCH TO IMMEDIATE ADDRESS
-----------------------------------------------------------------
      WHEN braI2 =>
```

```
        progcntrRd <= '1';
        alusel <= inc;
        shiftsel <= shftpass;
        outregWr <= '1';
        next_state <= braI3;
      WHEN braI3 =>
        outregRd <= '1';
        next_state <= braI4;
      WHEN braI4 =>
        outregRd <= '1';
        progcntrWr <= '1';
        addrregWr <= '1';
        next_state <= braI5;
      WHEN braI5 =>
        vma <= '1';
        rw <= '0';
        next_state <= braI6;
      WHEN braI6 =>
        vma <= '1';
        rw <= '0';
        IF ready = '1' THEN
          progcntrWr <= '1';
          next_state <= loadPc;
        ELSE
          next_state <= braI6;
        END IF;
-----------------------------------------------------------------
-- BRANCH GREATER THAN
-----------------------------------------------------------------
      WHEN bgtI2 =>
        regSel <= instrReg(5 DOWNTO 3);
        regRd <= '1';
        opRegWr <= '1';
        next_state <= bgtI3;
      WHEN bgtI3 =>
        opRegRd <= '1';
        regSel <= instrReg(2 DOWNTO 0);
        regRd <= '1';
        compsel <= gt;
        next_state <= bgtI4;
      WHEN bgtI4 =>
        opRegRd <= '1' AFTER 1 ns;
        regSel <= instrReg(2 DOWNTO 0);
        regRd <= '1';
        compsel <= gt;
        IF compout = '1' THEN
          next_state <= bgtI5;
        ELSE
          next_state <= incPc;
        END IF;
      WHEN bgtI5 =>
        progcntrRd <= '1';
        alusel <= inc;
        shiftSel <= shftpass;
        next_state <= bgtI6;
      WHEN bgtI6 =>
        progcntrRd <= '1';
        alusel <= inc;
        shiftsel <= shftpass;
```

```
        outregWr <= '1';
        next_state <= bgtI7;
      WHEN bgtI7 =>
        outregRd <= '1';
        next_state <= bgtI8;
      WHEN bgtI8 =>
        outregRd <= '1';
        progcntrWr <= '1';
        addrregWr <= '1';
        next_state <= bgtI9;
      WHEN bgtI9 =>
        vma <= '1';
        rw <= '0';
        next_state <= bgtI10;
      WHEN bgtI10 =>
        vma <= '1';
        rw <= '0';
        IF ready = '1' THEN
          progcntrWr <= '1';
          next_state <= loadPc;
        ELSE
          next_state <= bgtI10;
        END IF;
-----------------------------------------------------------------
-- INCREMENT
-----------------------------------------------------------------
     WHEN inc2 =>
        regSel <= instrReg(2 DOWNTO 0);
        regRd <= '1';
        alusel <= inc;
        shiftsel <= shftpass;
        outregWr <= '1';
        next_state <= inc3;
     WHEN inc3 =>
        outregRd <= '1';
        next_state <= inc4;
     WHEN inc4 =>
        outregRd <= '1';
        regsel <= instrReg(2 DOWNTO 0);
        regWr < '1';
        next_state <= incPc;
-----------------------------------------------------------------
-- LOAD PROCESS FOR PROGRAM COUNTER
-----------------------------------------------------------------
      WHEN loadPc =>
        progcntrRd <= '1';
        next_state <= loadPc2;
      WHEN loadPc2 =>
        progcntrRd <= '1';
        addrRegWr <= '1';
        next_state <= loadPc3;
      WHEN loadPc3 =>
        vma <= '1';
        rw <= '0';
        next_state <= loadPc4;

      WHEN loadPc4 =>
        vma <= '1';
        rw <= '0';
```

```
        IF ready = '1' THEN
          instrWr <= '1';
          next_state <= execute;
        ELSE
          next_state <= loadPc4;
        END IF;
-------------------------------------------------------------------
-- INCREMENT FOR PROGRAM COUNTER
-------------------------------------------------------------------
      WHEN incPc =>
        progcntrRd <= '1';
        alusel <= inc;
        shiftsel <= shftpass;
        next_state <= incPc2;
      WHEN incPc2 =>
        progcntrRd <= '1';
        alusel <= inc;
        shiftsel <= shftpass;
        outregWr <= '1';
        next_state <= incPc3;
      WHEN incPc3 =>
        outregRd <= '1';
        next_state <= incPc4;
      WHEN incPc4 =>
        outregRd <= '1';
        progcntrWr <= '1';
        addrregWr <= '1';
        next_state <= incPc5;
      WHEN incPc5 =>
        vma <= '1';
        rw <= '0';
        next_state <= incPc6;
      WHEN incPc6 =>
        vma <= '1';
        rw <= '0';
        IF ready = '1' THEN
          instrWr <= '1';
          next_state <= execute;
        ELSE
          next_state <= incPc6;
        END IF;
      WHEN OTHERS =>
        next_state <= incPc;
   END CASE;
END PROCESS;
-------------------------------------------------------------------
-- CLOCK AND RESET INPUTS ANALYSIS. SYNCHRONOUS TRANSITION TO CURRENT_STATE
-------------------------------------------------------------------
  controlffProc: PROCESS(clock, reset)
  BEGIN
    IF reset = '1' THEN
      current_state <= reset1 AFTER 1 ns;
    ELSIF clock'event AND clock = '1' THEN
      current_state <= next_state AFTER 1 ns;
    END IF;
  END PROCESS;
END rtl;
```

Рис. 12.11. Содержимое файла control.vhd (модель управляющего устройства МП)

Архитектурное тело модели содержит два процесса: nxtstateproc и controlffProc. Список чувствительности первого процесса nxtstateproc показывает, что этот процесс активизируется в следующих случаях:

- произошла смена состояния управляющего устройства (сигнал current_state);
- поступила для выполнения новая команда (сигнал instrReg);
- произошло изменение выходного сигнала компаратора (сигнал compout);
- внешняя память сигнализирует о готовности данных для МП (сигнал ready).

Активизируясь, данный процесс прежде всего выполняет инициализацию выходных управляющих сигналов, а затем с помощью оператора CASE анализируется текущее состояние, и в зависимости от результата анализа выполняется одна из следующих цепочек переходов между состояниями устройства (вместе с операциями, связанными с каждым из этих состояний):

- последовательность состояний сброса Reset (состояния reset1...reset6);
- последовательность состояний загрузки Load (состояния load1...load4);
- последовательность состояний сохранения Store (состояния store1...store4);
- последовательность состояний пересылки Movee (состояния move1...move4);
- последовательность состояний загрузки непосредственным операндом LoadI (состояния loadI1...loadI6);
- последовательность состояний условного перехода BraI (состояния braI1...braI6);
- последовательность состояний условного перехода BgtI (состояния bgtI1...bgtI10);
- последовательность состояний инкремента (состояния inc1...inc4);
- последовательность состояний загрузки программного счетчика LoadPc (состояния loadPc1...loadPc4);
- последовательность состояний инкремента для программного счетчика (состояния incPc1...incPc6).

В текущем состоянии execute еще один оператор CASE анализирует поступающую команду (NOP, LOAD, STORE, MOVE, LOADI, BRANCHI, BRANCHGTI).

Второй процесс controlffProc анализирует вход reset, и в случае поступления сигнала сброса устройство возвращается в состояние reset1. Если же сигнала сброса нет, то процесс анализирует наличие положительного фронта на входе синхросигнала clk. В случае, если положительный фронт налицо, осуществляется переход в определенное ранее следующее состояние нашего управляющего автомата.

12.10. Модель регистра МП

Модель регистра МП содержится в файле Reg.vhd (рис. 12.12).

```
--------------------------------------------------------------------
----------
-- REG
--------------------------------------------------------------------
----------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpu_lib.ALL;
--------------------------------------------------------------------
```

```
--------
                                --  REGISTER INTERFACE DESCRIPTION -
                                --  ENTITY reg
                                --              _________
ENTITY reg IS                   --             | REG     |
  PORT( a    : IN  bit16;       --   a _______|         |
        clk  : IN  std_logic;   --             |         |_______q
        q    : OUT bit16);      --             |         |
END reg;                        -- clk _______|         |
                                --             |_________|
                                --
                                --
                                --
-------------------------------------------------------------
-------
ARCHITECTURE rtl OF reg IS
BEGIN
  regproc: PROCESS
  BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    q <= a AFTER 1 ns;
  END PROCESS;
END rtl;
```

Рис. 12.12. Содержимое файла reg.vhd

Интерфейс регистра (entity reg) содержит следующие сигналы:

- a: входная информация для регистра (16-разрядный бинарный вектор);
- clk: синхросигнал для регистра;
- q: выход регистра (16-разрядный бинарный вектор).

Процесс regproc в архитектурном теле модели регистра анализирует наличие положительного фронта синхроимпульса на входе clk и в случае наличия такового осуществляет «регистрацию» входной информации.

Данная модель регистра используется в качестве модели регистра адреса и регистра команды МП (см. п.12.6 и рис. 12.8).

12.11. Модель регистра МП с тремя состояниями

Файл Trireg.vhd (рис. 12.13) содержит модель регистра с тремя состояниями.

```
-------------------------------------------------------
---------------
-- REG
-------------------------------------------------------
--------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpu_lib.ALL;
-------------------------------------------------------
---------------
```

```
                                --  REGISTER INTERFACE
                                --  DESCRIPTION – ENTITY reg
                                --                _________
ENTITY reg IS                   --               | REG     |
  PORT( a    : IN  bit16;       --    a _______|         |
        clk  : IN  std_logic;   --               |         |_______q
        q    : OUT bit16);      --               |         |
END reg;                        -- clk _______|         |
                                --               |_________|
                                --
                                --
                                --
-------------------------------------------------------------
---------------
ARCHITECTURE rtl OF reg IS
BEGIN
  regproc: PROCESS
  BEGIN
    WAIT UNTIL clk'event AND clk = '1';
    q <= a AFTER 1 ns;
  END PROCESS;
END rtl;
```

Рис. 12.13. Содержимое файла trireg.vhd

Она используется в качестве модели регистра операнда АЛУ и программного счетчика МП (см. п.12.6 и рис. 12.8).

Интерфейс регистра с тремя состояниями (entity trireg) содержит следующие сигналы:

- a: входная информация для регистра с тремя состояниями (16-разрядный бинарный вектор);
- clk: синхросигнал для регистра с тремя состояниями;
- q: выход регистра с тремя состояниями (16-разрядный бинарный вектор);
- en: сигнал разрешения для регистра с тремя состояниями.

Архитектурное тело модели содержит два процесса.

Процесс triregdata анализирует наличие положительного фронта синхро-импульса на входе clk и в случае наличия такового осуществляет промежуточное хранение поступившей входной информации.

Процесс trireg3st анализирует состояние входа en. В случае, если en = 1, поступившая входная информация фиксируется на выходе регистра; если en = 0, то выход регистра переходит в третье состояние. В любом другом случае (у сигнала en тип данных std_logic, т. е. этот сигнал имеет 9 возможных значений) выход регистра не определен.

12.12. Модель регистровой памяти (массива регистров) МП

Модель регистровой памяти (массива из восьми регистров Reg0...Reg7) находится в файле Regarray.vhd (рис. 12.14).

```
-------------------------------------------------------------------
-----------------
-- REGARRAY
-------------------------------------------------------------------
-----------------
library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE work.cpu_lib.ALL;
-------------------------------------------------------------------
-----------------
                                --  REGARRAY INTERFACE
                                --  DESCRIPTION – ENTITY regarray
                                --                  __________
ENTITY regarray IS              --            | REGARRAY |
  PORT( data : IN  bit16;       --data ______|          |
        sel  : IN  t_reg;       --           |          |
        en   : IN  std_logic;   -- sel ______|          |______q
        clk  : IN  std_logic;   --           |          |
        q    : OUT bit16);      --  en ______|          |
END regarray;                   --           |          |
                                -- clk ______|          |
                                --           |          |
                                --           |__________|
                                --
-------------------------------------------------------------------
-----------------
ARCHITECTURE rtl OF regarray IS
  TYPE   t_ram IS ARRAY (0 TO 7) OF bit16;
  SIGNAL temp_data : bit16;
BEGIN
-------------------------------------------------------------------
-----------------
-- REGISTER SELECTION AND DATA STORAGE
-------------------------------------------------------------------
-----------------
  PROCESS(clk,sel)
    VARIABLE ramdata : t_ram;
  BEGIN
    IF clk'event AND clk = '1' THEN
      ramdata(conv_integer(sel)) := data;
    END IF;
    temp_data <= ramdata(conv_integer(sel)) AFTER 1 ns;
  END PROCESS;
-------------------------------------------------------------------
-----------------
-- ENABLE INPUT ANALYSIS
-------------------------------------------------------------------
-----------------
  PROCESS(en, temp_data)
```

```
  BEGIN
    IF en = '1' THEN
      q <= temp_data AFTER 1 ns;
    ELSE
      q <= "ZZZZZZZZZZZZZZZZ" AFTER 1 ns;
    END IF;
  END PROCESS;
END rtl;
```

Рис. 12.14. Содержимое файла regarray.vhd (модель регистровой памяти МП)

Интерфейс регистровой памяти (entity regarray) содержит следующие сигналы:

- data: входная информация для регистровой памяти (16-разрядный бинарный вектор);
- sel: выбор одного из восьми регистров массива;
- en: сигнал разрешения для регистровой памяти;
- clk: синхросигнал для регистровой памяти;
- q: выход регистровой памяти (16-разрядный бинарный вектор).

Состояние регистровой памяти описывается переменной ramdata, представляющей собой массив из восьми 16-разрядных слов.

Архитектурное тело модели регистровой памяти содержит два процесса. Первый процесс анализирует наличие положительного фронта синхро-импульса на входе clk и в случае наличия такового осуществляет промежуточное хранение поступившей входной информации. При этом, как и ранее (см. п.12.4), стандартная функция CONV_INTEGER используется для преобразования (в данном случае — 3-разрядного) вектора в целое число для выбора одного из восьми 16-разрядных слов (моделей регистров).

Второй процесс анализирует состояние входа en. В случае, если en = 1, поступившая входная информация фиксируется на выходе регистровой памяти; если en = 0, то выход регистра устанавливается в третье состояние.

12.13. Модель сдвигового регистра МП

Файл Shift.vhd (рис. 12.15) содержит модель сдвигового регистра МП.

```
-------------------------------------------------------------
-----------------
-- SHIFT
-------------------------------------------------------------
-----------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpu_lib.ALL;
-------------------------------------------------------------
-----------------
                                --  SHIFT INTERFACE DESCRIPTION -
                                --  ENTITY shift
                                --                 _________
ENTITY shift IS                 --                | SHIFT   |
  PORT( a    : IN  bit16;       --   a ______|         |
```

```
        sel  : IN  t_shift;       --            |          |______y
        y    : OUT bit16);        --            |          |
END shift;                        -- sel _______|          |
                                  --            |__________|
                                  --
                                  --
                                  --
-------------------------------------------------------------------
-----------------
ARCHITECTURE rtl OF shift IS
BEGIN
  shftproc: PROCESS(a, sel)
  BEGIN
-------------------------------------------------------------------
-----------------
-- SHIFT OPERATION SELECTION
-------------------------------------------------------------------
-----------------
    CASE sel IS
      WHEN shftpass =>                                -- NO SHIFT
        y <= a AFTER 1 ns;
      WHEN shl =>                                     -- SHIFT LEFT
EXECUTION
        y <= a(14 DOWNTO 0) & '0' AFTER 1 ns;
      WHEN shr =>                                     -- SHIFT RIGHT
EXECUTION
        y <= '0' & a(15 DOWNTO 1) AFTER 1 ns;
      WHEN rotl =>                                    -- ROTATE LEFT
EXECUTION
        y <= a(14 DOWNTO 0) & a(15) AFTER 1 ns;
      WHEN rotr =>                                    -- ROTATE RIGHT
EXECUTION
        y <= a(0) & a(15 DOWNTO 1) AFTER 1 ns;
    END CASE;
  END PROCESS;
END rtl;
```

Рис. 12.15. Содержимое файла shift.vhd (модель сдвигового регистра МП)

Следующие сигналы составляют интерфейс сдвигового регистра (entity shift):

- a: входная информация для сдвигового регистра (16-разрядный бинарный вектор);
- sel: выбор операции сдвига (см. п.12.5);
- y: выходная информация сдвигового регистра (16-разрядный бинарный вектор).

Процесс shftproc в архитектурном теле модели сдвигового регистра с помощью последовательного оператора CASE анализирует задаваемый вид сдвиговой операции и выполняет ее с помощью операции конкатенации (Concatenation).

12.14. Моделирование МП в среде ModelSim

При создании проекта CPU в среде ModelSim (см. главу 4) на закладке Project мы видим все 11 файлов нашего проекта (рис. 12.16). После их компиляции на закладке sim отображается структура проекта CPU (рис. 12.17), содержащая все его компоненты с именами соответствующих интерфейсов и их архитектурных тел, а также используемые проектом стандартные пакеты.

Рис. 12.16. Файлы проекта CPU

Загрузка модели CPU для выполнения ее моделирования осуществляется на закладке Library выбором интерфейса top по команде Load Design (см. главу 4).

На рис. 12.18 приведены результаты моделирования на верхнем уровне проекта для трех циклов копирования элементов данных. При этом в окне Process (переменная mem_data — см. п.12.4) можно наблюдать состояние внешней памяти после трех циклов копирования элементов данных.

На рис. 12.19, *а* приведены результаты моделирования проекта во временном интервале от 0 до 800 ns (последовательность состояний сброса и начало выполнения команды loadI 1,#10).

На рис. 12.19, *б*. приведены результаты моделирования проекта во временном интервале от 630 до 1485 ns (завершение процесса сброса и выполнение команды loadI 1,#10).

На рис. 12.19, *в*. приведены результаты моделирования проекта во временном интервале от 5160 ns до 6 us (выполнение команды store 3,2).

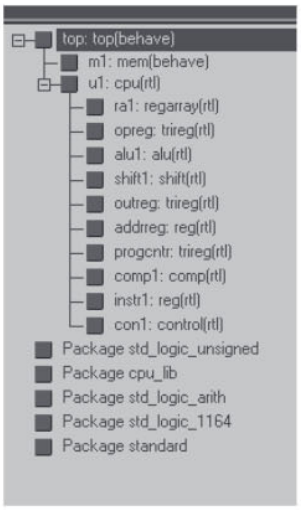


Рис. 12.17. Структура проекта CPU

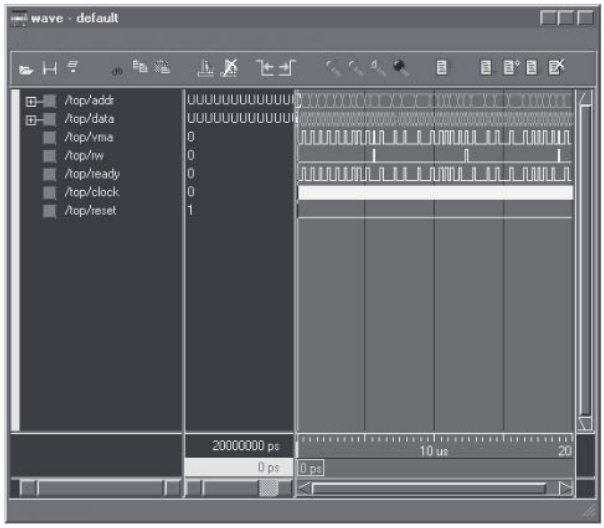


Рис. 12.18. Результаты моделирования CPU

На рис. 12.19, *г.* приведены результаты моделирования проекта во временном интервале от 6160 ns до 7 us (выполнение команды bgt 1,6,#00).

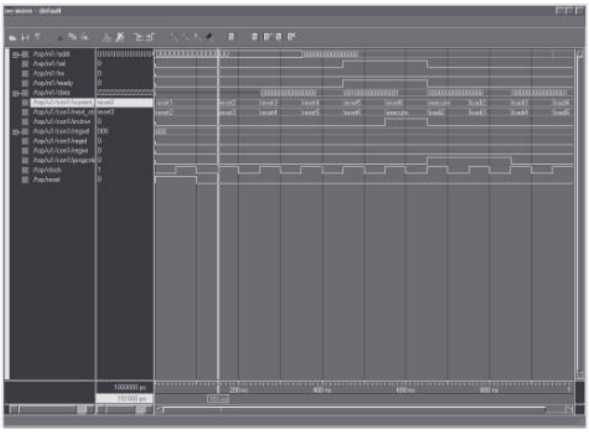


Рис. 12.19,а. Результаты моделирования (последовательность состояний Reset — процесс сброса, начало выполнения команды loadl 1, #10)

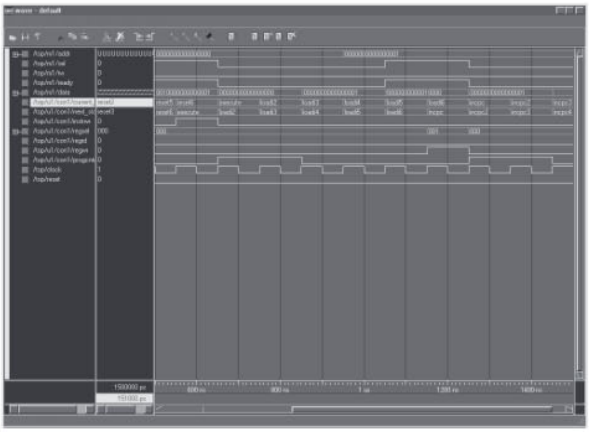


Рис. 12.19,б. Результаты моделирования (выполнение команды loadl 1, #10)

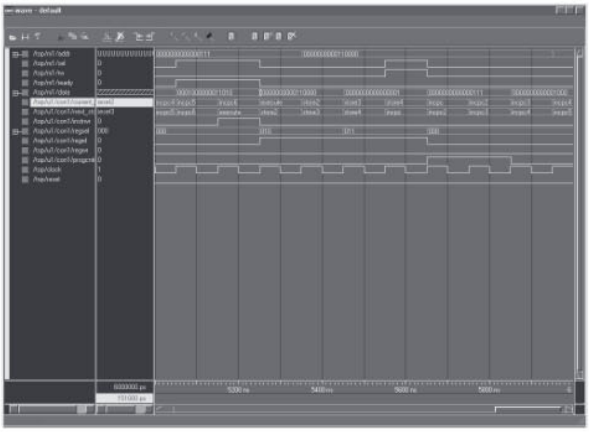


Рис. 12.19,в. Результаты моделирования (выполнение команды store 3,2)

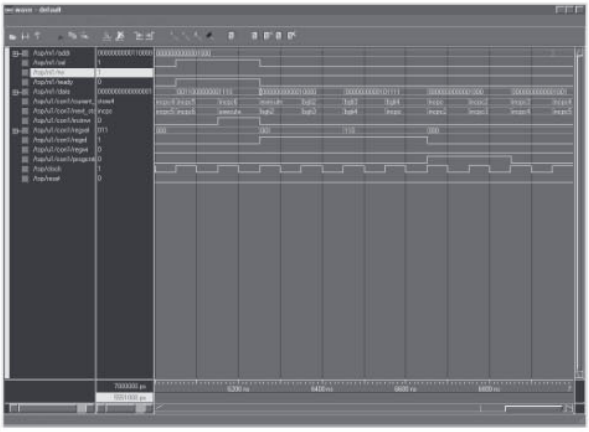


Рис. 12.19,г. Результаты моделирования (выполнение команды bgt 1, 6, #00)

Приложение А

Синтаксис языка VHDL

Design File

```
design_file <= design_unit {...}
design_unit <=
      {library_clause | use_ clause}
      library_unit
library_unit <=
      entity_declaration          | architecture_body
      | package_declaration       | package_body
      | configuration_declaration
library_clause <= library identifier {,  } ;
```

Library Unit Declarations

```
entity_declaration <=
      entity identifier is
         [ generic (generic_interface_list) ;]
         [ port (port_interface_list) ;]
         { entity_declarative_item}
      [ begin
         { concurrent_assertion_statement
            | passive_concurrent_procedure_call_statement
            | passive_process_statement} ]
      end [ entity ] [ identifier ] ;
entity_declarative_item <=
    subprogram_declaration          | subprogram_body
    | type_declaration              | subtype_declaration
    | constant_declaration          | signal_declaration
    | shared_variable_declaration   | file_declaration
    | alias_declaration
    | attribute_declaration         | attribute_specification
    | disconnection_specification   | use_clause
    | group_template_declaration    | group_declaration
architecture_body <=
    architecture identifier of entity_name is
        { block_declarative_item}
    begin
        { concurrent_statement }
    end [ architecture ] [ identifier ] ;
configuration_ declaration <=
    configuration identifier of entity_name is
        { use_clause | attribute_specification | group_declaration }
        block_configuration
    end [ configuration ] [ identifier ] ;
block_configuration <=
```

```
    for ( architecture_name
        | block_statement_label
        | generate_statement_label [ ( ( discrete_range | static_expression ) ) ] )
        { use_clause }
        { block_configuration
        | for component_specification
              [ binding_indication ; ]
              [ block_configuration ]
          end for ; }
    end for ;
package_declaration <=
    package_identifier is
        { package_declarative_item }
end [ package ] [identifier ] ;
package_declarative_item <=
        subprogram_declaration
        | type_declaration              | subtype_declaration
        | constant_declaration          | signal_declaration
        | shared_variable_declaration   | file_declaration
        | alias_declaration             | component_declaration
        | attribute_declaration         | attribute_specification
        | disconnection_specification   | use clause
        | group_template_declaration    | group_declaration
package_body <=
        package body identifier is
            { package_body_declarative_item }
        end [package body] [ identifier ] ;
package_body_declarative_item <=
        subprogram_declaration          | subprogram_body
        | type_declaration              | subtype_declaration
        | constant_declaration          | shared_variable_declaration
        | file_declaration              | alias_declaration
        | use_clause
        | group_template_declaration    | group_declaration
```

Declarations and Specifications

```
subprogram_specification <=
        procedure ( identifier | operator_symbol ) [ parameter_interface_list ) ]
        | [ pure | impure ]
          function ( identifier | operator_symbol )
              [ (parameter_interface_list ) ] return type_mark
    subprogram_declaration <= subprogram_specification ;
subprogram_body <=
    subprogram_specification is
        { subprogram_declarative_part }
    begin
        { sequential_statement }
    end [ procedure | function ] [ identifier | operator_symbol ] ;
subprogram_declarative_part <=
```

```
    subprogram_declaration                  | subprogram_body
    | type_declaration                      | subtype_declaration
    | constant_declaration                  | variable_declaration
    | file_declaration                      | alias_declaration
    | attribute_declaration                 | attribute_specification
    | use_clause
    | group_template_declaration            | group_declaration

type_declaration <=
    type identifier is type_definition ;
    | type identifier ;
type_definition <=
    enumeration_type_definition             | integer_type_definition
    | floating_type_definition              | physical_type_definition
    | array_type_definition                 | record_type_definition
    | access_type_definition                | file_type_definition
constant_declaration <=
     constant identifier { ,   } : subtype_indication [:= expression] ;
signal_declaration <=
      signal identifier { ,   } : subtype_indication [register | bus ]
                                  [ := expression ] ;
variable_declaration <=
    [ shared ] variable identifier { ,   } : subtype_indication [ := expression ] ;
file declaration <=
      file identifier { ,   }: subtype_indication
          [ [ open file_open_kind_expression] is string_expression ] ;
alias_declaration <=
    alias ( identifier | character_literal | operator_symbol_) [ : subtype_indication ]
      is name [ signature ] ;
component_declaration <=
    component identifier [is]
      [ generic (generic_interface_list ) ; ]
      [ port (port_interface_list ) ; ]
    end component [ identifier ] ;
attribute_declaration <= attribute identifier : type_mark ;
attribute_specification <=
    attribute identifier of entity_name_list : entity_class is expression ;
entity_name_list <=
    ( ( identifier | character_literal | operator_symbol ) [ signature ] ) { ,  }
    | others
    | all
entity_class <=
   entity      | architecture  | configuration  | procedure    | function
   | package  | type          | subtype | constant     | signal
   | variable | component     | label
   | literal  | units         | group  | file
configuration_specifIcation <=
```

```
    for component_specification binding_indication ;
component_specifIcation <=
   ( instantiation_label { ,   } |others | all) : component_name
binding_indication <=
    [ use ( entity entity_name [ (architecture_identifier) ]
          | configuration configuration_name
          | open ) ]
    [ generic map ( generic_association_list ) ]
    [ port map ( port_association_list) ]
disconnection_specification <=
    disconnect ( signal_name { ,   } |others | all ) : type_mark
       after time_expression ;
group_template_declaration <=
     group identifier is ( ( entity_class []){,   });
group_declaration <=
     group identifier : group_template_name (( name | character_literal ){,   });
use_clause <= use selected_name {,   };
```

Type Definitions

```
enumeration_type_definition <= ( ( identifier | character_literal) { ,   } )
integer_type_definition <=
     range ( range_attribute_name
          | simple_expression ( to | downto ) simple_expression )
floating_type_definition <=
     range ( range_attribute_name
          | simple_expression ( to | downto } simple_expression )
physical_type_definition <=
     range ( range_attribute_name
          | simple_expression ( to | downto } simple_expression)
       units
          identifier ;
          { identifier = physical_literal ; }
       end units [ identifier ]
array_type_definition <=
    array ( ( type_mark range  ) { ,   } )of element_subtype_indication
   | array ( discrete_range { ,   } )of element_subtype_indication
record_type_definition <=
    record
      ( identifier {,   }: subtype_indication ;)
      {...}
    end record [ identifier_]
access_type_defmition <= access subtype_indication
file_type_defnition <= file of type_mark
subtype_declaration <= subtype identifier is subtype_indication ;
subtype_indication <=
    [ resolution_function_name ]
    type_mark
```

```
        [ range ( range_attribute_name
                | simple_expression ( to | downto } simple_expression )
          | ( discrete_range { , } ) ]
discrete_range <=
    discrete_subtype_indication
    | range_attribute_name
    | simple_expression ( to | downto) simple_expression
type_mark <= type_name | subtype_name
```

Concurrent Statements

```
concurrent_statement <=
    block_statement
    | process_statement
    | concurrent_procedure_call_statement
    | concurrent_assertion_statement
    | concurrent_signal_assignment_statement
    | component_instantiation_statement
    | generate_statement
block_statement <=
    block_label :
    block [ ( guard_expression ) ] [ is ]
       [ generic ( generic_interface_list ) ;
       [ generic map ( generic_association_list ) ; ] ]
       [ port ( port_interface_list) ;
       [ port map ( port_association_list) ; ] ]
        { block_declarative_item }
    begin
        { concurrent_statement}
    end block [ block_label ] ;
block_declarative_item <=
    subprogram_declaration        | subprogram_body
    | type_declaration            | subtype_declaration
    | constant_declaration        | signal_declaration
    | shared_vanable_declaration  | file_declaration
    | alias_declaration           | component_declaration
    | attribute_declaration       | attribute_specification
    | configuration_specification | disconnection_specification
    | use_clause
    | group_template_declaration| group_declaration
process_statement <=
    [ process_label : ]
    [ postponed ] process [ (signal_name { , }) ] [is]
        { process_declarative_item }
    begin
        { sequential_statement}
    end [ postponed ] process [ process_label ] ;
process_declarative_item <=
    subprogram_declaration        | subprogram_body
    | type_declaration            | subtype_declaration
```

```
    | constant_declaration          | variable_declaration
    | file_declaration              | alias_declaration
    | attribute_declaration         | attribute_specification
    | use_clause
    | group_template_declaration   | group_declaration
concurrent_procedure_call_statement <=
    [ label : ]
    [ postponed ] procedure_name [ (parameter_association_list) ] ;
concurrent_assertion_statement <=
     [label:]
     [ postponed ] assert boolean_expression
                    [ report expression ] [ severity expression ] ;
concurrent_signal_assignment_statement <=
     [ label : ] [ postponed ] conditional_signal_assignment
     | [ label : ] [ postponed ] selected_signal_assignment
conditional_signal_assignment <=
     ( name [ aggregate ) <=
        [ guarded ] [ delay_mechanism ]
            { waveform when boolean_expression else }
            waveform [ when boolean_expression ] ;
selected_signal_assignment <=
      with expression select
       ( name | aggregate) <=
            [ guarded ] [ delay_mechanism ]
            { waveform when choices , }
            waveform when choices ;
component_instantiation_statement <=
      instantiation_label :
       ( [ component ] component_name
        | entity entity_name [ ( architecture_identifier ) ]
        | configuration configuration_name )
       [ generic map (generic_association_list) ]
       [ port map (port_association_list) ] ;
generate_statement <=
    generate_label :
    ( for identifier in discrete_range | if boolean_expression ) generate
       [ { block_declarative_item }
    begin ]
     {concurrent_statement}
    end generate [ generate_label ] ;
```

Sequential Statements

```
sequential_statement <=
    wait_statement                | assertion_statement
    | report_statement            | signal_assignment_statement
    | variable_assignment_statement | procedure_call statement
    | if_statement                | case_statemenl
    | loop_statement              | next_statement
```

```
    | exit_statement                    | return_statement
    | null_statement
wait_statement <=
    [ label : ] wait [ on signal_name { ,   } ]
                     [ until boolean_expression ]
                     [ for time_expression ] ;
assertion_statement <=
    [ label : ] assert boolean_expression
                [ report expression ] [ severity expression ] ;
report_statement <= [ label : ] report expression [ severity expression ] ;
signal_assignment_statement <=
    | label: | ( name | aggregate ) <= [ delay_mechanism ] waveform
delay_mechanism <=
    transport | [ reject time_expression ] inertial
waveform <=
    ( value_expression [ after time_expression ]
     | null [ after time_expression ] ) { ,   }
     | unaffected
variable_assignment_statement <=
    [ label : | ( name | aggregate ) := expression ;
procedure_call_statement <=
    [ label : ] procedure_name [ (parameter_association_list ) ] ;
if_statement <=
    [ if_label : ]
    if boolean_expression then
        { sequential_statement}
     { elsif booleane_expression then
        { sequential_statement} }
      [ else
        { sequential_statement} ]
      end if [ if_label ] ;
case_statement <=
     [ case_label : ]
     case expression is
        ( when choices => { sequential_statement} )
        {...}
     end case [ case_label ] ;
loop_statement <=
     [ loop_label : ]
     [ while condition | for identifier in discrete_range ] loop
         { sequential_statement}
     end loop [ loop_label ] ;
next_statement <= [ label : ] next [ loop_label ] [ when boolecm_expression ] ;
exit_statement <= [ label: ] exit [ loop_label ] [ when boolean_expression ] ;
return_statement <= [ label : ] return [ expression ] ;
null_statement <= [ label : ] null ;
```

Interfaces and Associations

```
interface_list <=
    ([constant ] identifier { ,   } : [in ] subtype_indication
                                      [ := static_expression ]
    | [ signal ] identifier {, }:[ mode ] subtype_indication [bus ]
                                      [:= static_expression]
    | [ variable ] identifier { ,   } : [ mode ] subtype_indication
                                      [ := static_expression ]
    | file identifier { ,   } : subtype_indication ) { ;   }
mode <= in | out | inout | buffer | linkage
association_list <= ( [ formal_part => ] actual_part ) { ,   }
formal_part <=
   generic _name
   | port_name
   | parameter_name
   | function_name ( ( generic_name [ port_name | parameter_name ) )
   | type_mark ( ( generic_name | port_name | parameter_name ) )
actual_part <=
   expression
   | signal_name
   | variable_name
   | open
   | function_name ( ( signal_name | variable_name ) )
   | type_mark ( ( signal_name | variable_name ) )
```

Expressions

```
expression <=
   relation { and relation }        | relation [ nand relation ]
   | relation { or relation }       | relation [ nor relation ]
   | relation { xor relation }      | relation { xnor relation }
relation <= shift_expression [ ( = | /= | < | <= | > | >= ) shift_expression ]
shift_expression <=
   simple_expression [ ( sll | srl | sla | sra | rol | ror) simple_expression ]
simple_expression <=[ + | − ] term { ( + | − | & ) term }
term <= factor { ( * | / | mod | rem ) factor }
factor <= primary [ ** primary ] | abs primary | not primary
primary <=
   name                            | literal
   | aggregate                     | function_call
   | qualified_expression          | type_mark ( expression )
   | new subtype_indication        | new qualified_expression
   | ( expression)
function_call <=function_name [ ( parameter_association_list) ]
qualified_expression <= type_mark ' ( expression ) | type_mark ' aggregate
name <=
   identifier
   | operator_symbol
   | selected_name
   | ( name | function_call ) ( expression {, })
```

```
    | ( name | function_call ) ( discrete_range )
    | attribute_name
selected_name <=
     ( name | function_call) . ( identifier | character_literal | operator_symbol | all )
operator_symbol <= " { graphic_character } "
attribute_name <= ( name | function_call) [ signature ] ' identifier [ ( expression ) ]
signature <=[ [ type_mark { ,   } ] [return type_mark ] ]
literal <=
    decima_literal
    | based_literal
    | [ decimal_literal | based_literal ] unit_name
    | identifier
    | character_literal
    | string_literal
    | bit_string_literal
    | null
decimal_literal <= integer [ . integer ][E[+] integer | E − integer ]
based_literal <=
    integer # based_integer [ . based_integer ]#[E[+] integer ( E − integer ]
integer <= digit { [ _ ] ... }
based_integer <= ( digit | letter ){[_]...}
character_literal <= ' graphic_character '
string_literal <= " { graphic_character } "
bit_string_literal <=(B|O|X)"[( digit | letter){[ _ ]...}]"
aggregate <= ( ( [ choices => ] expression ) { ,   } )
choices <= ( simple_expression | discrete_range | identifier | others ) { | ... }
label <= identifier
identifier <= letter { [ _ ] ( letter | digit) } | \ graphic_character { ... } \
```

Приложение Б

Различные виды операторов и краткие, формальные примеры их использования

Statement or Clause	Example(s)
Access Type	TYPE access_type IS ACCESS type_to_be_accessed;
Aggregate	record_type := (first, second, third);
Alias	ALIAS opcode : BIT_VECTOR (0 TO 3) IS INSTRUCTION (10 TO 13);
Architecture	ARCHITECTURE architecture_name OF entity_name IS --declare some signals here BEGIN --put some concurrent statements here END architecture_name
Array Type	TYPE array_type IS ARRAY (0 TO 7) OF BIT;
Assert	ASSERT x>10 REPORT «x is too small» SEVERITY ERROR;
Attribute Declaration	ATTRIBUTE attribute_name: attribute_type;
Attribute Specification	ATTRIBUTE attribute_name OF entity_name:entity_class IS value;
Block Statement	Block_name:BLOCK --declare some stuff here BEGIN --put some concurrent statements here END BLOCK block_name;

Statement or Clause	Example(s)
Case Statement	CASE some_expression IS WHEN some_value=> --do_some_stuff WHEN some_other_value=> --do_some_other_stuff WHEN OTHERS=> --do_some_default_stuff END CASE;
Component Declaration	COMPONENT component_name PORT(port1_name:port1_type; port2_name:port2_type; port3_name:port3_type; END COMPONENT;
Component Instantiation	instance_name:component_name PORT MAP (first_port,second_port,third_port) instance_name:component_name PORT MAP (formal1=>actual1, formal2=>actual2);
Conditional Signal Assignment	target<=first_value WHEN(x=y) ELSE second_value WHEN a=>b ELSE third_value;
Configuration Declaration	CONFIGURATION configuration_name OF entity_name IS FOR architecture_name FOR instance_name: entity_name USE ENTITY library_name.entity_name (architecture_name); END FOR; FOR instance_name: entity_name USE CONFIGURATION library_name.configuration_name; END FOR; END FOR; END configuration_name;
Constant Declaration	CONSTANT constant_name: constant_type := value;

Statement or Clause	Example(s)
Entity Declaration	ENTITY entity_name IS PORT(port1:port1_type; port2:port2_type); END entity_name;
Exit Statement	EXIT; EXIT WHEN a<=b; EXIT loop_label WHEN x=z;
File Type Declaration	TYPE file_type_name IS FILE OF data_type;
File Object Declaration	FILE file_object_name: file_type_name IS IN «/absolute/path/name»;
For Loop	FOR loop_variable IN start TO end LOOP --do_some_stuff END LOOP;
Function Declaration	FUNCTION function_name(parameter1:parameter1_type; parameter2:parameter2_type) RETURN return_type;
Function Body	FUNCTION function_name(parameter1:parameter1_type; parameter2:parameter2_type) RETURN return_type IS BEGIN --do some stuff END function_name;
Generate Statement	generate_label:FOR gen_var IN Start TO end GENERATE label:component_name PORT MAP(…); END GENERATE;
Generic Declaration	GENERIC(generic1_name:generic1_type; generic2_name:generic2_type);

Statement or Clause	Example(s)
Generic Map	GENERIC MAP(generic1_name=>value1,value2);

Guarded Signal Assignment	g1:BLOCK(clk='1' AND clk'EVENT)
	BEGIN
	q<=GUARDED d AFTER 5 NS;
	END BLOCK;

IF Statement	IF x<=y THEN
	--some statements
	END IF;
	IF z>w THEN
	--some statements
	ELSIF q<r THEN
	--some more statements
	END IF;
	IF a=b THEN
	--some statements
	ELSIF c=d THEN
	--some more statements
	ELSE
	--even more statements
	END IF;

Incomplete Type	TYPE type_name;
Library Declaration	LIBRARY library_name;
Loop Statement	FOR loop_variable IN start TO end LOOP --do lots of stuff END LOOP;

Next Statement	IF i<0 THEN
	NEXT;
	END IF;

Others Clause	WHEN OTHERS=> --do some stuff
Package Declaration	PACKAGE package_name IS --declare some stuff END PACKAGE;

Statement or Clause	Example(s)
Package Body	PACKAGE BODY package_name IS --put subprogram bodies here END package_name;

Physical Type	TYPE physical_type_name
	IS RANGE start TO end
	UNITS
	unit1;
	unit2=10unit1;
	unit3=10unit2;
	END UNITS;

Port Clause	PORT(port1_name:port1_type; port2_name:port2_type);
Port Map Clause	PORT MAP(port1_name=> signal1,signal2);
Procedure Declaration	PROCEDURE procedure_name(parm1:IN parm1_type; parm2:OUT parm2_type; parm3:INOUT parm3_type);
Procedure Body	PROCEDURE procedure_name(parm1:IN parm1_type; parm2:OUT parm2_type; parm3:INOUT parm3_type) IS BEGIN --do some stuff END procedure_name;
Process Statement	PROCESS(signal1,signal2,signal3) --declare some stuff BEGIN --do some stuff END PROCESS;
Record Type	TYPE record_type IS RECORD field1:field1_type; field2:field2_type; END RECORD;

Statement or Clause	Example(s)
Report Clause	ASSERT x=10 REPORT «some string»;
Return Statement	RETURN;
Selected Signal Assignment	WITH z SELECT x<=1 AFTER 5 NS WHEN 0, 2 AFTER 5 NS WHEN 1, 3 AFTER 5 NS WHEN OTHERS;
Severity Clause	ASSERT x>5 REPORT «some string» SEVERITY ERROR;
Signal Assignment	a<=b AFTER 20 NS;
Signal Declaration	SIGNAL x : xtype;
Subtype Declaration	SUBTYPE bit8 IS INTEGER RANGE 0 TO 255;
Transport Signal Assignment	x<=TRANSPORT y AFTER 50 NS;
Type Declaration	TYPE color is(red,yellow,blue,green,orange);
Use Clause	USE WORK.my_package.all;
Variable Declaration	VARIABLE variable_name: variable_type;
Variable Assignment	a:= 5; a:= b + c;
Wait Statement	WAIT ON a,b,c; WAIT UNTIL clock'EVENT AND clock='1'; WAIT FOR 100 NS; WAIT ON a,b UNTIL b>10 FOR 50 NS;
While Loop	WHILE x>15 LOOP --do some stuff END LOOP;

Приложение B

Стандартные пакеты языка VHDL

Standard

The predefined types, subtypes and functions of VHDL are defined in a package called standard, stored in the library std. Each design unit in a design is automatically preceded by the following context clause:

library std, work; **use** std.standard.**all**;

so the predefined items are directly visible in the design. The package standard is listed here. The comments indicate which operators are implicitly defined for each explicitly defined type. These operators are also automatically made visible in design units. The types universal_integer and universal_real are anonymous types. They cannot be referred to explicitly.

```
package standard is
   type boolean is (false, true);
      -- implicitly declared for boolean operands:
      -- "and", "or", "nand", "nor", "xor", "xnor", "not" return boolean
      -- "=", "/=", "<", "<=", ">", ">=" return boolean
   type bit is ('0', '1');
      -- implicitly declared for bit operands:
      -- "and", "or", "nand", "nor", "xor", "xnor", "not" return bit
      -- "=", "/=*, "<", "<=", ">", ">=" return boolean
type character is (
   nul,    soh,    stx,    etx,    eot,    enq,    ack,    bel,
   bs,     ht,     If,     vt,     ff,     cr,     so,     si,
   dle,    dcl,    dc2,    dc3,    dc4,    nak,    syn,    etb,
   can,    em,     sub,    esc,    fsp,    gsp,    rsp,    usp,
   '',     '!',    '"',    '#',    '$',    '%',    '&',    '',
   '(',    ')',    '*',    '+',    ',',    '-',    '.',    '/',
   '0',    '1',    '2',    '3',    '4',    '5',    '6',    '7',
   '8',    '9',    ':',    ';',    '<',    '=',    '>',    '?',
   '@',    'A',    'B',    'C',    'D',    'E',    'F',    'G',
   'H',    'I',    'J',    'K',    'L',    'M',    'N',    'O',
   'P',    'Q',    'R',    'S',    'T',    'U',    'V',    'W',
   'X',    'Y',    'Z',    '[',    '\',    ']',    '^',    '_',
   '`',    'a',    'b',    'c',    'd',    'e',    'f',    'g',
   'h',    'i',    'j',    'k',    'l',    'm',    'n',    'o',
   'p',    'q',    'r',    's',    't',    'u',    'v',    'w',
   'x',    'y',    'z',    '{',    '|',    '}',    '~',    del
   C128,   C129,   C130,   c131,   c132,   C133,   c134,   c135,
   C136,   c137,   c138,   c139,   c140,   c141,   c142,   c143,
   c144,   c145,   c146,   c147,   c148,   c149,   c150,   c151,
   c152,   c153,   c154,   c155,   c156,   c157,   c158,   c159,
```

```
    ' ',    'i',    '¢',    '£',    '¤',    '¥',    ' ',    '§',
    '¨',    '©',    'ª',    '«',    '¬',    '-',    '®',    '¯',
    'º',    '±',    '²',    '³',    '´',    'µ',    '¶',    '·',
    ' ',    '¹',    'º',    '»',    '¼',    '½',    '¾',    '¿',
    'À',    'Á',    'Â',    'Ã',    'Ä',    'Å',    'Æ',    'Ç',
    'È',    'É',    'Ê',    'Ë',    'Ì',    'Í',    'Î',    'Ï',
    'Ð',    'Ñ',    'Ò',    'Ó',    'Ô',    'Õ',    'Ö',    '×',
    'Ø',    'Ù',    'Ú',    'Û',    'Ü',    'Ý',    'Þ',    'ß',
    'à',    'á',    'â',    'ã',    'ä',    'å',    'æ',    'ç',
    'è',    'é',    'ê',    'ë',    'ì',    'í',    'î',    'ï',
    'ð',    'ñ',    'ò',    'ó',    'ô',    'õ',    'ö',    '÷',
    'ø',    'ù',    'ú',    'û',    'ü',    'ý',    'þ',    'ÿ');
      -- implicitly declared for character operands:
      -- "=", "/=", "<", "<=", ">", ">=" return boolean
   type severityjevel is (note, warning, error, failure);
      -- implicitly declared for severityjevel operands:
      -- "=", "/=", "<", "<=", ">", ">=" return boolean
   type universal_integer is range implementation defined;
      -- implicitly declared for universal integer operands:
      -- "=", "/=", "<", "<=", ">", ">=" return boolean
      -- "**",_ "*" _"/","+","-", "abs", "rem", "mod"  return universal integer
   type universal_real is range implementation defined;
      -- implicitly declared for universal real operands:
      -- "=", "/=", "<", "<=", ">", ">=" return boolean
      -- "**", "*", "/", "+", "-", "abs" return  universal real
   type integer is range implementation  defined;
— implicitly declared for integer operands:
      — "=", "/=", "<", "<=", ">", ">=" return boolean
      — "**", "*", "i","+","-", "abs", "rem", "mod" return integer
   subtype natural  is integer range 0 to integer'high;
   subtype positive is integer range 1 to integer'high;
   type real is range implementation defined,
      -- implicitly declared for real operands:
      -- "=", "/=", "<", "<=", ">", ">=" return boolean
      -- "**", "*","/","+","_", "abs"  return real
   type time is range implementation defined
      units
         fs;
         ps  =1000fs;
         ns  =1000ps;
         us  = 1000ns;
         ms  = 1000 us;
         sec = 1000 ms;
         min = 60 sec;
         hr  = 60 min;
      end units;
```

```
    -- implicitly declared for time operands:
    -- "=", "/=", "<", "<=", ">", ">=" return boolean
    -- "*", "+"i"_", "abs" return time
    --"/" return time or universal integer
  subtype delay_length is time range 0 fs to time'high;
  impure function now return delay_length;
  type string is array (positive range <>) of character;
    -- implicitly declared for string operands:
    -- "=", "/=", "<", "<=", ">", ">=" return boolean
    -- "&" return string
  type bit_vector is array  (natural  range <>) of bit;
    -- implicitly declared for bit_vector operands:
    -- "and", "or", "nand",  "nor", "xor",  "xnor", "not" return bit_vector
    -- "sll", "srl", "sla", "sra", "rol", "ror" return bit_vector
    -- "=", "/=", "<", "<=", ">", ">=" return boolean
    -- "&" return bit_vector
  type file_open_kind is (read_mode, write_mode, append_mode);
    -- implicitly declared for file_open_kind operands:
    --"=", "/=","<", "<=", ">", ">=", return boolean
  type file_open_status is (open_ok, status_error,
                            name_error, mode_error);
    -- implicitly declared for file_open_status operands:
    -- "=", "/=", "<", "<=", ">", ">=" return boolean
  attribute foreign: string;
end package standard;
```

VHDL-87

The following items are not included in standard in VHDL-87: the types file_open_kind and file_open_status; the subtype delayjength; the attribute foreign; the operators **xnor**, **sll**, **srl**, **sla**, **sra**, **rol** and **ror**. The result time of the function now is time. The type character includes only the first 128 values, corresponding to the ASCII character set.

Standard Logic 1164

```
------------------------------------------------------------
--    Title      :std_logic_1164 multi-value logic system
--    Library    :This package shall be compiled into a
--               :library symbolically named IEEE.
--               :
--    Developers :IEEE model standards group (par 1164)
--    Purpose    :This packages defines a standard for
--               :designers to use in describing the
--               :interconnection data types used in vhdl
--               :modeling.
--               :
--    Limitation :The logic system defined in this
```

```
--               :package may be insufficient for
--               :modeling switched transistors, since
--               :such a requirement is out of the scope
--               :of this effort. Furthermore,
--               :mathematics, primitives, timing
--               :standards, etc. are considered
--               :orthogonal issues as it relates to this
--               :package and are therefore beyond the
--               :scope of this effort.
--               :
--    Note       :No declarations or definitions shall be
--               :included in, or excluded from this
--               :package. The "package declaration"
--               :defines the types, subtypes and
--               :declarations of std_logic_1164. The
--               :std_logic_1164 package body shall be
--               :considered the formal definition of the
--               :semantics of this package. Tool
--               :developers may choose to implement the
--               :package body in the most efficient
--               :manner available to them.
--               :
-- -------------------------------------------------------------
--    modification history :
-- -------------------------------------------------------------
--   version | mod. date:|
--    V4.200 | 01/02/92  |
-- -------------------------------------------------------------
PACKAG  std_logic_1164 IS
     ------------------------------------------------------
     --  logic state system (unresolved)
     ------------------------------------------------------
     TYPE std_ulogic IS ( 'U',    -- Uninitialized
                          'X',    -- Forcing  Unknown
                          '0',    -- Forcing  0
                          '1',    -- Forcing  1
                          'Z',    -- High Impedance
                          'W',    -- Weak    Unknown
                          'L',    -- Weak     0
                          'H',    -- Weak     1
                          '-'     -- Don't care
                        );

-----------------------------------------------------------------
    -- unconstrained array of std_ulogic for use with the
    -- resolution function
    ---------------------------------------------------------------
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE  )
     OF std_ulogic;
```

```
------------------------------------------------------------------
    -- resolution function

------------------------------------------------------------------
    FUNCTION resolved ( s : std_ulogic_vector ) RETURN
     std_ulogic;
    ------------------------------------------------------------------
    -- *** industry standard logic type ***
    ------------------------------------------------------------------
    SUBTYPE std_logic IS resolved std_ulogic;
    ------------------------------------------------------------------
    -- unconstrained array of std_logic for use in
    -- declaring signal arrays
    ------------------------------------------------------------------
    TYPE std_logic_vector IS ARRAY ( NATURAL RANGE ) OF
     std_logic;
    ------------------------------------------------------------------
    -- common subtypes
    ------------------------------------------------------------------
    SUBTYPE X01    IS resolved std ulogic RANGE 'X' TO
                   '1'; -- {'X', '0', '1')
    SUBTYPE X01Z   IS resolved std_ulogic RANGE 'X' TO
                   'Z'; -- ('X','0','1','Z')
    SUBTYPE UX01   IS resolved std_ulogic RANGE 'U' TO
                   '1'; -- ('U', 'X', '0', '1')
    SUBTYPE UX01Z  IS resolved std_ulogic RANGE 'U' TO
                   'Z'; -- ('U','X','0','1','Z')
    ------------------------------------------------------------------
    -- overloaded logical operators
    ------------------------------------------------------------------
    FUNCTION "and"      ( l : std_ulogic; r : std_ulogic )
     RETURN UX01;
    FUNCTION "nand"     ( l : std ulogic; r : std ulogic )
     RETURN 0X01;
    FUNCTION "or"       ( l : std_ulogic; r : std_ulogic )
     RETURN UX01;
    FUNCTION "nor"      ( l : std_ulogic; r : std_ulogic )
     RETURN UX01;
    FUNCTION "xor"      ( l : std ulogic; r : std ulogic )
     RETURN UX01;
    -- function "xnor"  ( l : std_ulogic; r : std_ulogic )
    -- return uxOl;
    FUNCTION "not"      ( l : std_ulogic               )
     RETURN UX01;
    ------------------------------------------------------
    -- vectorized overloaded logical operators
    ------------------------------------------------------
    FUNCTION "and"  ( l, r : std_logic_vector  ) RETURN
     std_logic_vector;
```

```
    FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN
     std_ulogic_vector;
    FUNCTION "nand" ( l, r : std_logic_vector  ) RETURN
     std_logic_vector;
    FUNCTION "nand" ( l, r : std_ulogic_vector ) RETURN
     std_ulogic_vector;
    FUNCTION "or"   ( l, r : std_logic_vector  ) RETURN
     std_logic_vector;
    FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN
     std_ulogtc_vector;
    FUNCTION "nor"  ( l, r : std_logic_vector  ) RETURN
     std_logic_vector;
    FUNCTION "nor"  ( l, r : std_ulogic_vector ) RETURN
     std_ulogic_vector;
    FUNCTION "xor"  ( l, r : std_logic_vector  ) RETURN
     std logic vector;
    FUNCTION "xor"  ( l, r : std_ulogic_vector ) RETURN
     std_ulogic_vector;
--  ------------------------------------------------------
--  Note : The declaration and implementation of the "xnor"
--  function is specifically commented until at which time
--  the VHDL language has been officially adopted as
--  containing such a function. At such a point, the
--  following comments may be removed along with this
--  notice without further "official" ballotting of this
--  std_logic_1164 package. It is the intent of this effort
--  to provide such a function once it becomes available
--  in the VHDL standard.
--  ------------------------------------------------------
--  function "xnor" ( l, r : std_logic_vector  ) return
--  std_logic_vector;
--  function "xnor" ( l, r : std_ulogic_vector ) return
--  std_ulogic_vector;
    FUNCTION "not"  ( l : std_logic_vector  ) RETURN
     std logic vector;
    FUNCTION "not"  ( l : std_ulogic_vector ) RETURN
     std_ulogic_vector ;
    ------------------------------------------------------
    -- conversion functions
    ------------------------------------------------------
    FUNCTION To_bit       ( s : std_ulogic;       xmap :
                            BIT := '0') RETURN BIT;
    FUNCTION To_bitvector ( s : std_logic_vector ; xmap :
                            BIT := '0') RETURN BIT VECTOR;
    FUNCTION To bitvector ( s t std_ulogic vector; xmap :
                            BIT := '0') RETURN BIT_VECTOR;
    FUNCTION TO_StdULogic       ( b : bit             )
     RETURN std_ulogic;
```

```
FUNCTION To_StdLogicVector  ( b : bit_vector       )
 RETURN std_logic_vector;
FUNCTION To_StdLogieVector  ( s : std_ulogic_vector)
 RETURN std_logic_vector;
FUNCTION To_StdULogicVector ( b : bit_vector       )
 RETURN std_ulogic_vector;
FUNCTION To_StdULogicVector ( s : std_logic_vector )
 RETURN std_ulogic__vector;
-------------------------------------------------------
--  strength strippers and type converters
-------------------------------------------------------
FUNCTION To_X01 ( s : std_logic_vector              )
 RETURN std_logic_vector;
FUNCTION To_X01  ( s : std_ulogic_vector            )
 RETURN  std_ulogic_vector;
FUNCTION To X01  ( s : std_ulogic                   )
 RETURN  X01;
FUNCTION To_X01  ( b : bit_vector                   )
 RETURN std logic vector;
FUNCTION TO_X01  ( b : bit_vector                   )
 RETURN  std_ulogic_vector;
FUNCTION To X01  ( b : bit                          )
 RETURN  X01;
FUNCTION To_X01Z ( s : std_logic_vector             )
 RETURN std_logic vector;
FUNCTION To_X01Z ( s : std_ulogic_vector            )
 RETURN std_ulogic vector;
FUNCTION TO X01Z ( s : std ulogic                   )
 RETURN  X01Z;
FUNCTION To_X01Z ( b : bit_vector                   )
 RETURN  std_logic vector;
FUNCTION To_X01Z ( b : bit_vector                   )
 RETURN std_ulogic_vector;
FUNCTION TO X01Z ( b : BIT                          )
 RETURN  X01Z;
FUNCTION To_UX01            ( s : std_logic_vector  )
 RETURN  std_logic_vector;
FUNCTION TO_UX01            ( s : std_ulogic_vector )
 RETURN  std_ulogic_vector;
FUNCTION To_UX01            ( s : std_ulogic        )
 RETURN  0X01;
FUNCTION To_UX01            ( b : bit_vector        )
 RETURN std loglc_vector;
FUNCTION TO_UX01            ( b : bit_vector        )
 RETURN  std_ulogic_vector;
FUNCTION To_UX01            ( b : bit               )
 RETURN  UX01;
-------------------------------------------------------------
--  edge detection
```

```
    ------------------------------------------------------------
    FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN
     BOOLEAN;
    FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN
     BOOLEAN;
    ------------------------------------------------------------
    --  object contains an unknown
    ------------------------------------------------------------
    FUNCTION Is_X ( s : std_ulogic_vector ) RETURN
     BOOLEAN;
    FUNCTION Is_X ( s : std_logic_vector  ) RETURN
     BOOLEAN;
    FUNCTION Is_X ( 8 : std_ulogic        ) RETURN
     BOOLEAN;
    END Std_logic_1164;
------------------------------------------------------------
--    Title      :std_logic_1164 multi-value logic system
--    Library    :This package shall be compiled into a
--               :library symbolically named IEEE.
--               :
--    Developers :IEEE model standards group (par 1164)
--    Purpose    :This packages defines a standard for
--               :designers to use in describing the
--               :interconnection data types used in vhdl
--               :modeling.
--               :
--    Limitation :The logic system defined in this
--               :package may be insufficient for
--               :modeling switched transistors, since
--               :such a requirement is out of the scope
--               :of this effort. Furthermore,
--               :mathematics, primitives, timing
--               :standards, etc. are considered
--               :orthogonal issues as it relates to this
--               :package and are therefore beyond the
--               :scope of this effort.
--               :
--    Note       :No declarations or definitions shall be
--               :included in, or excluded from this
--               :package. The "package declaration"
--               :defines the types, subtypes and
--               :declarations of std_logic_1164. The
--               :std_logic_1164 package body shall be
--               :considered the formal definition of the
--               :semantics of this package. Tool
--               :developers may choose to implement the
--               :package body in the most efficient
--               :manner available to them.
--               :
```

```
-- ------------------------------------------------------
--    modification history :
-- ------------------------------------------------------
--   version | mod. date:|
--    V4.200 | 01/02/92  |
-- ------------------------------------------------------
PACKAGE BODY std_logic_1164 IS
    -- ------------------------------------------------------------
    --  local types
    -- ------------------------------------------------------------
    TYPE stdlogic_ld    IS ARRAY (std_ulogic)             OF
std_ulogic;
    TYPE stdlogic_table IS ARRAY (std_ulogic, std_ulogic) OF
std_ulogic;
    -- ------------------------------------------------------------
    --  resolution function
    -- ------------------------------------------------------------
    CONSTANT resolution_table : stdlogic_table :- (
-----------------------------------------------------------------
--|  U     X     0     1     Z     W     L     H     -        |   |
-----------------------------------------------------------------
  ( 'U',  'U',  'U',  'U',  'U',  'U',  'U',  'U',  'U' ), -- | U |
  ( 'U',  'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X' ), -- | X |
  ( 'U',  'X',  '0',  'X',  '0',  '0',  '0',  '0',  'X' ), -- | 0 |
  ( 'U',  'X',  'X',  '1',  '1',  '1',  '1',  '1',  'X' ), -- | 1 |
  ( 'U',  'X',  '0',  '1',  'Z',  'W',  'L',  'H',  'X' ), -- | Z |
  ( 'U',  'X',  '0',  '1',  'W',  'W',  'W',  'W',  'X' ), -- | W |
  ( 'U',  'X',  '0',  '1',  'L',  'W',  'L',  'W',  'X' ), -- | L |
  ( 'U',  'X',  '0',  '1',  'H',  'W',  'W',  'H',  'X' ), -- | H |
  ( 'U',  'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X' )  -- | - |
  );
FUNCTION resolved ( s : std_ulogic_vector ) RETURN
std_ulogic IS
VARIABLE result : std_ulogic := 'Z';  -- weakest state
default
    BEGIN
        -- the test for a single driver is essential
        -- otherwise the loop would return 'X' for a
        -- single driver of '-' and that would conflict
        -- with the value of a single driver unresolved
        -- signal.
        IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
        ELSE
             FOR i IN S'RANGE LOOP
                 result := resolution_table( result, s(i));
             END LOOP;
        END IF;
        RETURN result;
    END resolved;
    -------------------------------------------------------------
```

```
    --  tables for logical operations
    ---------------------------------------------------------------
    --  truth table for "and" function
    CONSTANT and_table : stdlogic_table := (
--  ---------------------------------------------------------------
--|  U     X     0     1     Z     W     L     H     -        |   |
--  ---------------------------------------------------------------
  ( 'U',  'U',  '0',  'U',  'U',  'U',  '0',  'U',  'U' ),  --| U |
  ( 'U',  'X',  '0',  'X',  'X',  'X',  '0',  'X',  'X' ),  --| X |
  ( '0',  '0',  '0',  '0',  '0',  '0',  '0',  '0',  '0' ),  --| 0 |
  ( 'U',  'X',  '0',  '1',  'X',  'X',  '0',  '1',  'X' ),  --| 1 |
  ( 'U',  'X',  '0',  'X',  'X',  'X',  '0',  'X',  'X' ),  --| Z |
  ( 'U',  'X',  '0',  'X',  'X',  'X',  '0',  'X',  'X' ),  --| W |
  ( '0',  '0',  '0',  '0',  '0',  '0',  '0',  '0',  '0' ),  --| L |
  ( 'U',  'X',  '0',  '1',  'X',  'X',  '0',  '1',  'X' ),  --| H |
  ( 'U',  'X',  '0',  'X',  'X',  'X',  '0',  'X',  'X' )   --| - |
  );
  --  truth table  for  "or"  function
  CONSTANT or_table   :   stdlogic_table   :=   (
  --  ---------------------------------------------------------------
  --|U     X     0     1     Z     W     L     H     -        |   |
  --
---------------------------------------------------------------------------
  ( 'U',  'U',  'U',  '1',  'U',  'U',  'U',  '1',  'U' ),  --| U |
  ( 'U',  'X',  'X',  '1',  'X',  'X',  'X',  '1',  'X' ),  --| X |
  ( 'U',  'X',  '0',  '1',  'X',  'X',  '0',  '1',  'X' ),  --| 0 |
  ( '1',  '1',  '1',  '1',  '1',  '1',  '1',  '1',  '1' ),  --| 1 |
  ( 'U',  'X',  'X',  '1',  'X',  'X',  'X',  '1',  'X' ),  --| Z |
  ( 'U',  'X',  'X',  '1',  'X',  'X',  'X',  '1',  'X' ),  --| W |
  ( 'U',  'X',  '0',  '1',  'X',  'X',  '0',  '1',  'X' ),  --| L |
  ( '1',  '1',  '1',  '1',  '1',  '1',  '1',  '1',  '1' ),  --| H |
  ( 'U',  'X',  'X',  '1',  'X',  'X',  'X',  '1',  'X' )   --| - |
  );
  --  truth table  for  "xor"  function
  CONSTANT xor_table   :   stdlogic_table   :=   (
  --  ---------------------------------------------------------------
  --|U     X     0     1     Z     W     L     H     -        |   |
  --  ---------------------------------------------------------------
  ( 'U',  'U',  'U',  'U',  'U',  'U',  'U',  'U',  'U' ),  --| U |
  ( 'U',  'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X' ),  --| X |
  ( 'U',  'X',  '0',  '1',  'X',  'X',  '0',  '1',  'X' ),  --| 0 |
  ( 'U',  'X',  '1',  '0',  'X',  'X',  '1',  '0',  'X' ),  --| 1 |
  ( 'U',  'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X' ),  --| Z |
  ( 'U',  'X',  'X',  'X',  'X',  'X',  'X',  'X',  'X' ),  --| W |
  ( 'U',  'X',  '0',  '1',  'X',  'X',  '0',  '1',  'X' ),  --| L |
  ( 'U',  'X',  '1',  '0',  'X',  'X',  '1',  '0',  'X' ),  --| H |
  ( 'U',  'X',  'X',  'X',  'X'.  'X',  'X',  'X',  'X' )   --| - |
  );
  --  truth table  for  "not"  function
  CONSTANT not_table:   stdlogic_ld   :=
```

```
  --  ------------------------------------------------------------
  --|U     X     0     1     Z     W     L     H     -       |
  --  ------------------------------------------------------------
  ( 'U',  'X',  '1',  '0',  'X',  'X',  '1',  '0',  'X' );

-------------------------------------------------------------------
  --  overloaded logical  operators   (  with optimizing hints   )

-------------------------------------------------------------------
     FUNCTION "and"     ( l   :   std_ulogic;   r  : std ulogic   )
      RETURN 0X01   IS
     BEGIN
          RETURN   (and tabled,   r) ) ;
     END  "and";
     F0NCTION "nand"  (  l   :   std_ulogic;   r  :  std ulogic   )
      RETURN 0X01   IS
     BEGIN
          RETURN     (not_table   (   and_table(l,   r) ) ) ;
     END  "nand";
    FUNCTION "or"   ( l : std_ulogic; r : std_ulogic )
     RETURN UX01 IS
    BEGIN
         RETURN (or_table(l, r) ) ;
    END "or";
    FUNCTION "nor"  ( l : std_ulogic; r : std ulogic )
      RETURN UX01 IS
    BEGIN
         RETURN  (not_table ( or_table( l, r )));
    END "nor";
    FUNCTION "xor"  ( l : std ulogic; r : std ulogic )
     RETURN UX01 IS
    BEGIN
         RETURN (xor_table(l, r));
    END "xor";
--  function "xnor"  ( l : std_ulogic; r : std_ulogic )
--  return uxOl is
--  begin
--       return not_table(xor_table(l, r) ) ;
--  end "xnor";
    FUNCTION "not"  ( l : Std_ulogic ) RETURN UX01 IS
    BEGIN
         RETURN (not_table(l));
    END "not";

-------------------------------------------------------------------
    -- and

-------------------------------------------------------------------
```

```
FUNCTION "and"  { l,r s std_logic_vector ) RETURN
  std_logic_vector IS
    ALIAS Iv : std logic vector ( 1 TO l' LENGTH ) IS :
    ALIAS rv : std_logic_vector ( 1 TO r' LENGTH ) IS :
    VARIABLE result : std_logic_vector ( 1 TO
                      l' LENGTH );
BEGIN
    IF ( l' LENGTH /= r' LENGTH ) THEN
        ASSERT FALSE
        REPORT "arguments of overloaded 'and' operator
          are not of the same length"
        SEVERITY FAILURE;
    ELSE
        FOR i IN result 'RANGE LOOP
            result(i) := and table (lv(i), rv(i));
        END LOOP;
    END IF;
    RETURN result;
END "and";
-------------------------------------------------------
FUNCTION "and"  ( l,r : std_ulogic_veetor ) RETURN
 std_ulogic_vector IS
    ALIAS Iv : std_ulogic_vector ( 1 TO l' LENGTH )
               IS l;
    ALIAS rv : std ulogic vector ( 1 TO r' LENGTH )
               IS r;
    VARIABLE result : std_ulogic_vector ( 1 TO
                      l' LENGTH );
BEGIN
     IF  (  l'LENGTH /= r'LENGTH ) THEN
         ASSERT FALSE
         REPORT "arguments of overloaded  'and'  operator
           are not of the  sane  length"
         SEVERITY FAILURE;
     ELSE
         FOR i IN result'RANGE LOOP
             result(i)  :-  and_table  (lv(iXI rv(i));
         END LOOP;
     END IF;
     RETURN result;
END  "and";
-------------------------------------------------------
--  nand
-------------------------------------------------------
FUNCTION "nand"   { l,r  :  std_logic_vector ) RETURN
  std_logic_vector IS
     ALIAS  lv :  std_logic_vector ( l TO l'LENGTH  )
                  IS l;
     ALIAS  rv :  std_logic_vector ( l TO r'LENGTH  )
                  IS r;
```

```
     VARIABLE result  :  std_logic_vector ( l TO
                         l'LENGTH );
BEGIN
     IF  (  l'LENGTH /= r'LENGTH ) THEN
         ASSERT FALSE
         REPORT "arguments of overloaded  'nand'
           operator are not of  the same length"
         SEVERITY FAILURE;
     ELSE
         FOR i IN result'RANGE LOOP
             result(i)  :=  not_table(and_table (lv(i),
               rv(i)));
         END LOOP;
     END  IF;
     RETURN  result;
END "nand";
-------------------------------------------------------------
FUNCTION "nand"   (  l,r  :  std_ulogic_vector )  RETURN
 std_ulogic_vector  IS
    ALIAS lv :  std_ulogic_vector ( l TO l'LENGTH  )
                IS l;
    ALIAS rv :  std_ulogic_vector ( l TO r'LENGTH  )
                IS r;
    VARIABLE result :  std_ulogic_vector ( l TO
                       l'LENGTH );
BEGIN
    IF  (  l'LENGTH /= r'LENGTH )  THEN
         ASSERT FALSE
         REPORT "arguments of overloaded  'nand'
           operator are not of the same length"
         SEVERITY FAILURE;
    ELSE
         FOR i IN result'RANGE LOOP
             result(i)  :=  not_table(and_table (lv(i),
               rv(i)));
         END LOOP;
    END  IF;
    RETURN  result;
END "nand";
-----------------------------------------------------------
--  or
-----------------------------------------------------------
FUNCTION "or"   ( l,r :  std_logic_vector ) RETURN
 std_logic_vector IS
    ALIAS  lv :  std_logic_vector ( l TO l'LENGTH  )
                 IS l;
    ALIAS  rv :  std_logic_vector ( l TO r'LENGTH  )
                 IS r;
    VARIABLE result : std_logic_vector ( 1 TO
                      l'LENGTH );
```

```
BEGIN
    IF  (  l'LENGTH /= r'LENGTH ) THEN
          ASSERT FALSE
          REPORT "arguments of overloaded  'or' operator
           are not  of the  same  length"
          SEVERITY FAILURE;
    ELSE
          FOR i IN result'RANGE LOOP
             result(i)  :=  or_table (lv(i),  rv(i));
          END LOOP;
    END IF;
    RETURN result;
END "or";
-------------------------------------------------------
FUNCTION "or"   (  l,r :  std_ulogic_vector )  RETURN
 std_ulogic_vector IS
    ALIAS  lv :  std_ulogic_vector  ( 1 TO  l'LENGTH )
                 IS l;
    ALIAS  rv :  std_ulogic_yector  ( 1 TO  r'LENGTH )
                 IS r;
    VARIABLE result : std_ulogic_vector  ( 1 TO
                      l'LENGTH );
BEGIN
    IF  (  l'LENGTH /= r'LENGTH ) THEN
         ASSERT FALSE
         REPORT "arguments of overloaded  'or' operator
           are not  of the  sane  length"
         SEVERITY FAILURE;
    ELSE
         FOR i IN result'RANGE LOOP
             result(i)  :=  or_table (lv(i),  rv(i))”
         END LOOP;
    END IF;
    RETURN result;
END "or";
-------------------------------------------------------
--  nor
-------------------------------------------------------
FUNCTION "nor"   ( l,r : std_logic_vector )  RETURN
 std_logic_vector IS
    ALIAS  lv :  std_logic_vector ( 1 TO l'LENGTH  )
                 IS l;
    ALIAS  rv :  std_logic_vector ( 1 TO r'LENGTH  )
                 IS r;
    VARIABLE result : std_logic_vector ( 1 TO
                      l'LENGTH );
BEGIN
IF (  l'LENGTH /= r'LENGTH ) THEN
         ASSERT FALSE
         REPORT "arguments of overloaded  'nor'  operator
```

```
            are not of  the  same  length"
          SEVERITY FAILURE;
      ELSE
          FOR i IN result'RANGE LOOP
              result(i)  :=  not_table(or_table (lv(i),
              rv(i)));
          END LOOP;
      END IF;
      RETURN result;
END  "nor";
-------------------------------------------------------
FUNCTION "nor"   (  l,r  : std_ulogic_vector ) RETURN
  std_ulogic_vector  IS
      ALIAS  lv :  std_ulogic_vector ( 1 TO  l'LENGTH  )
                   IS l;
      ALIAS  rv :  std_ulogic_vector ( 1 TO  r'LENGTH  )
                   IS r;
      VARIABLE result :  std_ulogic_vector  ( 1 TO
                         l'LENGTH );
BEGIN
      IF (  l'LENGTH /= r'LENGTH ) THEN
          ASSERT FALSE
          REPORT "arguments of overloaded  'nor'  operator
            are not of  the  same  length"
          SEVERITY FAILURE;
      ELSE
          FOR i IN result'RANGE LOOP
              result(i)  :=  not_table(or_table <lv(i),
              rv(i)));
          END LOOP;
      END IF;
      RETURN result;
END  "nor";
-------------------------------------------------------
--  xor
-------------------------------------------------------
FUNCTION "xor"   {  l,r  : std_logic_vector ) RETURN
  std_logic_vector  IS
      ALIAS  lv :  std_logic_vector ( 1 TO l'LENGTH )
                   IS l;
      ALIAS  rv :  std_logic_vector ( 1 TO r'LENGTH )
                   IS r;
      VARIABLE result  :  std_logic_vector ( 1 TO
                          l'LENGTH  );
BEGIN
      IF ( l'LENGTH /= r'LENGTH ) THEN
          ASSERT FALSE
          REPORT "arguments of overloaded  'xor'  operator
            are not of  the  same  length"
          SEVERITY FAILURE;
```

```
        ELSE
            FOR i IN result'RANGE LOOP
                result(i)  :=  xor.table (lv(i),  rv(i));
            END LOOP;
        END IF;
        RETURN result;
    END  "xor";
    ------------------------------------------------------
    FUNCTION  "xor"   (  l,r s std_ulogic_vector ) RBTURN
      std_ulogic_veetor IS
        ALIAS lv : std_ulogic_vector  ( 1  TO l'LENGTH )
                   IS l;
        ALIAS rv : std_ulogic_vector  ( 1  TO r'LENGTH )
                   IS r;
        VARIABLE result : std_ulogic_vector  ( 1 TO
                          l'LENGTH );
    BEGIN
        IF (  l'LENGTH / r'LENGTH  ) THEN
            ASSERT FALSE
            REPORT "arguments of overloaded 'xor'  operate]
               are not  of  the same length"
            SEVERITY FAILURE;
        ELSE
            FOR i IN result'RANGE  LOOP
                result(i)  := xor_table (lv(i),  rv(i));
            END LOOP;
        END IF;
        RETURN result;
    END  "xor";
--  ------------------------------------------------------
--  --  xnor
--  ------------------------------------------------------
--  Note : The declaration and implementation of the "xnor"
--  function is specifically commented until at which time
--  the VHDL language has  been officially adopted as
--  containing such a function.  At such a point, the
--  following comments may he  removed along with this
--  notice without further "official" ballotting of this
--  std_logic_1164 package. It is  the intent of this effort
--  to provide such a function once it becomes available
--  in the VHDL standard.
--  ------------------------------------------------------
--  function "xnor"   (  l,r :  std_logic_vector ) return
      std_logic_vector is
--      alias  lv  :  std_logic_vector  ( 1 to  l'length  )
                      is l;
--      alias  rv  :  std_logic_vector  ( 1 to  r'length  }
                      is r;
--      variable  result :  std_logic_vector  { 1 to
                            l'length );
```

```
--  begin
--      if  (  l'length /= r'length }  then
--           assert false
--           report "arguments of  overloaded 'xnor'
--           operator are not  of the  same  length"
--           severity failure;
--      else
--           for i in result'range loop
--               result(i)  :=  not_table(xor_table (lv(i),
--               rv(i)));
--           end loop;
--      end if;
--      return result;
--  end "xnor";
--  ------------------------------------------------------
--  function "xnor"   (  l,r :  std_ulogic_vector )  return
      std_ulogic_vector is
--      alias  lv :  std_ulogic_vector ( 1 to l'length  )
                   is l;
--      alias  rv :  std_ulogic_veetor ( 1 to r'length  )
                   is r;
--      variable result : std_ulogic_vector ( 1 to
                        l'length );
--  begin                                      i>
--      if  (  l'length /= r'length ) then
--           assert false
--           report "arguments of overloaded  'xnor'
             operator are not of the  same  length"
--           severity failure;
--      else
--           for i in result'range loop
--               result(i)  := not_table(xor_table  (lv(i),
                 rv{i)));
--           end loop;
--      end if;
--      return result;
--  end "xnor";
    ------------------------------------------------------
    --  not
    ------------------------------------------------------
    FUNCTION  "not"   (  l : std_logic_vector ) RETURN
      std_logic_vector IS
        ALIAS lv : std_logic_vector  ( 1 TO  l'LENGTH  )
                   IS l;
        VARIABLE result : std_logic_vector  ( 1 TO
                          l'LENGTH )  :=  (OTHERS  =>  'X');
    BEGIN
        FOR i IN result'RANGE LOOP
            result(i)  := not_table(  lv(i)  );
        END LOOP;
```

```
    RETURN result;
END;
-------------------------------------------------------
FUNCTION  "not"   (  1 : std_ulogic_vector ) RETURN
  std_ulogic_vector IS
    ALIAS lv : std_ulogic_vector  ( 1 TO l'LENGTH  )
               IS l;
    VARIABLE result : std_ulogic_vector ( 1  TO
                      l'LENGTH }  :=  (OTHERS  =>  'X');
BEGIN
    FOR i IN result'RANGE LOOP
        result(i)  := not_table(  lv(i)  );
    END LOOP;
    RETURN result;
END;
-------------------------------------------------------
-- conversion tables
-------------------------------------------------------
TYPE logic__x01_table IS ARRAY (std_ulogic'LOW TO
  std_ulogic'HIGH)  OF X01;
TYPE logic_x01z_table IS ARRAY (std_ulogic'LOW TO
std_ulogic'HIGH)  OF X01Z;
TYPE logic_ux01_table IS ARRAY (Std_ulogic'LON TO
  std_ulogic'HIGH)  OF UX01;
-------------------------------------------------------------
--  table name  :  cvt_to_x01
--
--  parameters  :
--          in  :  std_ulogic   -- some logic value
--  returns     :  xOl          -- state value of  logic
                                   value
--  purpose     :  to convert  state-strength to state
                   only
--
--  example     :  if (cvt_to_x01 (input_signal) =  '1'  )
                   then ...
--
-------------------------------------------------------------
CONSTANT cvt_to_x01  : Iogic_x01_table  :=  (

                      'X',   --  'U'
                      'X',   --  'X'
                      '0',   --  '0'
                      '1',   --  '1'
                      'X',   --  'Z'
                      'X',   --  'W'
                      '0',   --  'L'
                      '1',   --  'H'
                      'X',   --  '-'
                       );

-------------------------------------------------------------
```

```
--  table name  :  cvt_to_x01z
--
--  parameters  :
--          in  :  std_ulogic  -- some logic value
--  returns     :  x01z        -- state value of logic value
--  purpose     :  to convert state-strength to state only
--
--  example     :  if (cvt_to_x01z (input_signal) = '1' )
                   then ...
--
-------------------------------------------------------------
CONSTANT cvt_to_x01z  : logic_x01z_table  :=  (
                        'X',  --  'U'
                        'X',  --  'X'
                        '0',  --  '0'
                        '1',  --  '1'
                        'Z',  --  'Z'
                        'X',  --  'W'
                        '0',  --  'L'
                        '1',  --  'H'
                        'X'   --  '-'
                        );
-------------------------------------------------------------
--  table name  :  cvt_to_ux01
--
--  parameters  :
--          in  :  std_ulogic   -- some logic value
--  returns     :  ux01         -- state value of  logic value
--  purpose     :  to convert state-strength to state only
--
--  example     :  if (cvt_to_ux01 (input_signal) =  '1'  )
                   then ...
--
-----------------------------------------------------------
CONSTANT cvt_to_ux01 : logic_ux01_table := (
                       'U',   --  'U'
                       'X',   --  'X'
                       '0',   --  '0'
                       '1',   --  '1'
                       'X',   --  'Z'
                       'X',   --  'W'
                       '0',   --  'L'
                       '1',   --  'H'
                       'X',   --  '-'
                       );
-------------------------------------------------------------
--  conversion functions
-------------------------------------------------------------
FUNCTION To_bit        ( s  : std_ulogic;         xmap  :
 BIT := '0')  RETURN BIT IS
BEGIN
```

```
            CASE S IS
                WHEN  '0'  |  'L' => RETURN ('0');
                WHEN  '1'  |  'H' => RETURN ('1');
                WREN  OTHERS => RETURN xmap;
            END CASE;
    END;
    ------------------------------------------------------------
    FUNCTION To_bitvector  (  s  :  std_logic_vector ; xmap :
                              BIT := '0')  RETURN BIT_VECTOR
                              IS
         ALIAS  sv  :  std_logic_vector ( s'LENGTH-1 DOWNTO
                       0 ) IS s;
         VARIABLE  result :  BIT_VECTOR ( s'LENGTH-1 DOWNTO
                           0 );
    BEGIN
         FOR i  IN  result'RANGE LOOP
             CASE sv(i) IS
                 WHEN  '0'  |  'L' => result(i) := '0';
                 WHEN  '1'  I  'H' => result(i) := '1';
                 WHEN  OTHERS => result (i)  := xmap;
             END CASE;
         END LOOP;
         RETURN result;
    END;
    ------------------------------------------------------------
    FUNCTION To_bitvector  {  s  :  std_ulogic_vector; xmap :
                              BIT := '0')  RETURN BIT_VECTOR IS
         ALIAS  sv  :  std_ulogic_vector ( s'LENGTH-1 DOWNTO
                       0 ) IS s;
         VARIABLE  result :  BIT_VECTOR ( S'LENGTH-1 DOWNTO
                            0 );
    BEGIN
    FOR i IN result'RANGE LOOP
             CASE sv(i) IS
                 WHEN  '0'  |  'L'  => result(i) :=  '0';
                 WHEN  '1'  |  'H'  => result(i) :=  '1';
                 WHEN  OTHERS  =>  result(i) := xmap;
             END CASE;
         END LOOP;
         RETURN result;
    END;

------------------------------------------------------------------
    FUNCTION To_StdULogic        ( b  : BIT                )
                                   RETURN Std_ulogic IS
    BEGIN
         CASE b IS
             WHEN '0'  => RETURN  '0';
             WHEN '1'  => RETURN  '1';
         END CASE;
    END;
```

```
-------------------------------------------------------------------
    FUNCTION To_StdLogicVector   (  b : BIT_VECTOR )
                                   RETURN std_logic_vector IS
         ALIAS bv :  BIT_VECTOR ( b'LENGTH-1 DOWNTO 0 ) IS b;
         VARIABLE result :  std_logic_vector ( b'LENGTH-1
                           DOWNTO 0 );
    BEGIN
         FOR i IN result'RANGE LOOP
             CASE bv(i) IS
                 WHEN  '0'  =>  result(i)  := '0';
                 WHEN  '1'  =>  result(i)  := '1';
             END CASE;
         END LOOP;
         RETURN result;
    END;
    -------------------------------------------------------------------
    FUNCTION To_StdLogicVector   (  s :  std_ulogic_vector )
                                   RETURN std_logic_vector IS
        ALIAS  sV :  std_ulogic_vector ( s'LENGTH-1 DOWNTO
                    0 ) IS s;
        VARIABLE result :  std_logic_vector ( s'LENGTH-1
                           DOWNTO 0  );
    BEGIN
        FOR i  IN result'RANGE LOOP
             result(i)  := sv(i);
        END LOOP;
        RETURN result;
    END;
    -------------------------------------------------------------------
    FUNCTION To_StdULogicVector  (  b :  BIT_VECTOR        )
                                   RETURN std_ulogic_vector IS
        ALIAS  bv :  BIT_VECTOR ( b'LENGTH-1 DOWNTO 0  )
                     IS b;
        VARIABLE result :  std_ulogic_vector ( b'LENGTH-1
                           DOWNTO 0  );
    BEGIN
         FOR i  IN result'RANGE LOOP
              CASE bv(i) IS
                  WHEN  '0'  =>  result(i)  := '0';
                  WHEN  '1'  =>  result(i)  := '1';
              END CASE;
         END LOOP;
         RETURN result;
    END;
   -------------------------------------------------------------------
    FUNCTION To_StdULogicVector  (  s :  std_logic_vector )
     RETURN std_ulogic_vector IS
        ALIAS  sv  :  std_logic_vector (  s'LENGTH-1 DOWNTO 0
         )  IS s;
        VARIABLE  result :  std_ulogic_vector ( s'LENGTH-1
```

```
          DOWNTO 0 );
    BEGIN
        FOR i  IN result'RANGE LOOP
            result(i)  := sv(i);
        END LOOP;
        RETURN result;
    END;

-------------------------------------------------------------
    --  strength strippers and  type converters

-------------------------------------------------------------
    --  to xOl
-------------------------------------------------------------
    FUNCTION To_X01   (  s  :  std_logic_vector )  RETURN
       std_logic_vector  IS
        ALIAS sv  : std_logic_vector  (  1 TO s'LENGTH ) IS s;
        VARIABLE result : std_logic_vector (  1 TO s'LENGTH
       );
    BEGIN
        FOR i IN  result'RANGE  LOOP
            result(i) :  cvt_to_x01  (sv(i));
        END LOOP;
        RETURN result;
    END;

-------------------------------------------------------------
    FUNCTION To_X01   (  s  :  std_ulogic_vector ) RETURN
       std_ulogic_vector IS
        ALIAS sv  : std_ulogic_vector {  1 TO s'LENGTH ) IS
          s;
         VARIABLE result  :  std_ulogic_vector  ( 1 TO
           s'LENGTH );
    BEGIN
       FOR i IN  result'RANGE  LOOP
            result(i) :=  cvt_to_x01  (sv(i));
        END LOOP;
        RETURN result;
    END;

-------------------------------------------------------------
    FUNCTION To_X01   (  s  :  std_ulogic ) RETURN  XOl IS
    BEGIN
        RETURN (cvt_to_x01(s));
    END;

-------------------------------------------------------------
    FUNCTION To_X01   (  b  :  BIT_VECTOR ) RETURN
       std_logic_vector  IS
        ALIAS bv  : BIT_VECTOR  (  1 TO b'LENGTH ) IS b;
        VARIABLE result : std_logic_vector (  1 TO b'LENGTH
```

```
      );
BEGIN
    FOR i IN  result'RANGE LOOP
        CASE  bv(i)  IS
            WHEN 'O' => result(i)  :=  '0';
            WHEN '1' => result(i)  :=  '1';
        END CASE;
    END LOOP;
    RETURN result;
END;
-------------------------------------------------------------
FUNCTION To_X01   (  b :  BIT_VECTOR ) RETURN
   std_ulogic_vector IS
    ALIAS bv : BIT_VECTOR  (  1 TO b'LENGTH ) IS b;
    VARIABLE result  : std_ulogic_vector  (  1 TO b'LENGTH
   ) ;
BEGIN
    FOR i IN result'RANGE  LOOP
        CASE bv(i)  IS
             WHEN '0' => result(i)  :=  '0';
             WHEN '1' => result(i)  :=  '1';
        END CASE;
    END LOOP;
    RETURN result;
END;
-------------------------------------------------------------
FUNCTION TO_X01   (  b :  BIT ) RETURN  X01 IS
BEGIN
        CASE b IS
             WHEN '0' => RETURN ('0');
             WHEN '1' => RETURN ('1');
        END CASE;
END;
-------------------------------------------------------------
--  to_x01z
-------------------------------------------------------------
FUNCTION To_X01Z   (  S : std_logic_vector ) RETURN
   std_logic_vector IS
    ALIAS sv : std_logic_vector  (  1 TO s'LENGTH ) IS s;
    VARIABLE result  : std_logic_vector  (  1 TO s'LENGTH
   );
BEGIN
    FOR i IN result'RANGE  LOOP
        result (i)  : cvt_to_x01z (sv(i));
    END LOOP;
    RETURN result;
END;
-------------------------------------------------------------
FUNCTION To_X01Z   (  S : std_ulogic_vector ) RETURN
   std_ulogic_vector IS
    ALIAS sv : std_ulogic_vector (  1 TO s'LENGTH ) IS
```

```
          s;
        VARIABLE result  : std_ulogic_vector  (  1 TO s'LENGTH
         );
    BEGIN
        FOR i IN result'RANGE  LOOP
            result(i)  : cvt_to_x01z (sv(i));
        END LOOP;
        RETURN result;
    END;
    -------------------------------------------------------------
    FUNCTION To_X01Z   (  s :  Std_ulogic ) RETURN  X01Z  IS
    BEGIN
        RETURN  (cvt_tO_x01z(s));
    END;

-----------------------------------------------------------------
    FUNCTION TO_X01Z   (  b :  BIT_VECTOR ) RETURN
       std_logic_vector IS
        ALIAS bv  : BIT_VECTOR  (  1 TO b'LENGTH ) IS b;
        VARIABLE result  :  std_logic_vector  (1 TO b'LENGTH
       );                                      >s
    BEGIN
        FOR i IN  result'RANGE  LOOP
            CASE  bv(1)  IS
                WHEN  '0'  =>  result(i)  := '0';
                WHEN  '1'  =>  result(i)  := '1';
            END CASE;
        END LOOP;
        RETURN result;
    END;

-------------------------------------------------------------------
    FUNCTION To_X01Z   (  b :  BIT_VECTOR ) RETURN
       std_ulogic_vector IS
        ALIAS bv  : BIT_VECTOR  (  1 TO b'LENGTH ) IS b;
        VARIABLE result  :  std_ulogic_vector ( 1 TO b'LENGTH
       );
    BEGIN
        FOR i IN  result'RANGE  LOOP
            CASE  bv(i)  IS
                WHEN  '0'  =>  result(i)  := '0';
                WHEN  '1'  =>  result(i)  := '1';
            END CASE;
        END LOOP;
        RETURN result;
    END;

-----------------------------------------------------------------
    FUNCTION TO_X01Z   (  b :  BIT ) RETURN  X01Z IS
    BEGIN
            CASE  b IS
```

```
                WHEN  '0'  =>  RETURN('O');
                WHEN  '1'  =>  RETURN('l');
            END CASE;
    END;

-------------------------------------------------------------------
    --  tO_UX01                    ;

-------------------------------------------------------------------
    FUNCTION To_UX01   (  s :  std_logic_vector ) RETURN
       std_logic_vector  IS
        ALIAS sv  : std_logic_vector  (  1 TO  S'LENGTH ) IS s;
        VARIABLE result  :  std_logic_vector  (  1 TO s'LENGTH
          );
    BEGIN
        FOR i IN result'RANGE  LOOP
            result(i)  := cvt_to_ux01 (sv(i));
        END LOOP;
        RETURN result;
    END;
    -------------------------------------------------------------------
    FUNCTION To_UX01   (  s :  std_ulogic_vector ) RETURN
       std_ulogic_vector IS
        ALIAS sv : std_ulogic_vector  (  1 TO s'LENGTH ) IS
          s;
        VARIABLE result  : std_ulogic_vector (  1 TO s'LENGTH
      );
    BEGIN
        FOR i IN result'RANGE LOOP
            result(i)  := cvt_to_ux01  (sv(i));
        END LOOP;
        RETURN result;
    END;
    ---------------------------------------------------------------
    FUNCTION To_UX01   (  s :  std_ulogic ) RETURN  UX01  IS
    BEGIN
        RETURN (cvt_to_ux01(s));
    END;
    -------------------------------------------------------------
    FUNCTION TO_UX01   (  b :  BIT_VECTOR ) RETURN
       std_logic_vector IS
        ALIAS bv : BIT_VECTOR  (  1 TO  b'LENGTH ) IS b;
        VARIABLE result  : std_logic_vector  (  1 TO b'LENGTH
          );
    BEGIN
        FOR i IN result'RANGE LOOP
            CASE bv(i) IS
                WHEN  '0'  =>  result(i)  := '0';
                WHEN  '1'  =>  result(i)  := '1';
            END CASE;
        END LOOP;
```

```
        RETURN result;
    END;
-------------------------------------------------------------
    FUNCTION To_UX01   (  b :  BIT_VECTOR ) RETURN
       std_ulogic_vector  IS
        ALIAS bv : BIT_VECTOR  (  1 TO  b'LENGTH ) IS b;
        VARIABLE result  : std_ulogic_vector (  1 TO b'LENGTH
          );
    BEGIN
        FOR i IN result'RANGE LOOP
            CASE bv(i) IS
                WHEN  '0'  =>  result(i)  := '0';
                WHEN  '1'  =>  result(i)  := '1';
            END CASE;
        END LOOP;
        RETURN result;
    END;
    -------------------------------------------------------------
    FUNCTION To_UX01   (  b :  BIT ) RETURN   UX01  IS
    BEGIN
            CASE b IS
                WHEN  '0'  =>  RETURN('O');
                WHEN  '1'  =>  RETURN('l');
            END CASE;
    END;
    -------------------------------------------------------------
    --  edge detection
    -------------------------------------------------------------
    FUNCTION rising_edge  (SIGNAL s  : std_ulogic)  RETURN
      BOOLEAN IS
    BEGIN
        RETURN  (s'EVENT AND (To_X01(S) = '1') AND
                             (To_X01(s'LAST_VALUE) =  '0'));
    END;
    FUNCTION falling_edge (SIGNAL s  : std_ulogic)  RETURN
      BOOLEAN IS
    BEGIN
        RETURN  (s'EVENT AND (To_X01(s) = '0') AND
                             (To_X01(s'LAST_VALUE) =  '1'));
    END;
   -------------------------------------------------------------
   --  object contains an unknown
   -------------------------------------------------------------
   FUNCTION Is X ( s : std_ulogic_vector ) RETURN BOOLEAN
     IS
   BEGIN
       FOR i XN s' RANGE LOOP
           CASE S(i) IS
              WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN
                TRUE;
              WHEN OTHERS => NULL;
```

```
          END CASE;
       END LOOP;
       RETURN FALSE;
   END;
   ------------------------------------------------------------
   FUNCTION Is_X ( s : std_logic_vector ) RETURN BOOLEAN
      IS
   BEGIN
      FOR i IN s' RANGE LOOP
          CASE S(i) IS
              WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN
                TRUE;
                WHEN OTHERS => NULL;
          END CASE;
      END LOOP;
      RETURN FALSE;
   END;
------------------------------------------------------------
   FUNCTION Is_X (  s : std_ulogic         ) RETURN  BOOLEAN
      IS
   BEGIN
       CASE S  IS
            WHEN 'U' |  'X'  |  'Z'  |  'W'  |  '-' => RETURN
             TRUE;
            WHEN OTHERS => NULL;
       END  CASE;
       RETURN  FALSE;
   END;
END std_logic_1164;
```

Standard Mathematical

Real Mathematical Operations

package math_real **is**
constant copyrightnotice: string := "Copyright 1996 IEEE. All rights reserved.";

```
------------------------------------------------------------
    --  Constant Definitions

------------------------------------------------------------
  constant math_e :             real := 2.71828_18284_59045_23536;
  constant math_1_over_e :      real := 0.36787_94411_71442_32160;
  constant math_pi :            real := 3.14159_26535_89793_23846;
  constant math_2_pib:          real := 6.28318_53071_79586_47693;
  constant math_1_over_pi:      real := 0.31830_98861_83790_67?54;
  constant math_pi_over_2 :     real := 1.57079_63267_94896_61923;
  constant math_pi_over_3 :     real := 1.04719_75511_96597_74615;
  constant math_pi_over_4 :     real := 0.78539_81633_97448_30962;
  constant math_3_pi_over_2 :   real := 4.71238_89803_84689_85769;
  constant math_log_of_2 :      real := 0.69314_71805_59945_30942;
```

```
  constant math_log_of_10 :      real := 2.30258_50929_94045_68402;
  constant math_log2_of_e :      real := 1.44269_50408_88963_4074;
  constant math_log10_of_e :     real := 0.43429_44819_03251_82765;
  constant math_sqrt_2 :         real := 1.41421 _35623_73095_04880;
  constant math_1_over_sqrt_2 : real := 0.70710_67811_86547_52440;
  constant math_sqrt_pi :        real := 1.77245_38509_05516_02730;
  constant math_deg_to_rad :     real := 0.01745_32925_19943_29577;
  constant math_rad_to_deg :     real := 57.29577_95130_82320_87680;

--------------------------------------------------------------------
    -- Function Declarations

--------------------------------------------------------------------
  function sign  (x: in real) return real;
  function ceil  (x: in real) return real;
  function floor (x: in real) return real;
  function round (x: in real) return real;
  function trunc (x: in real) return real;
  function "mod"   (x, y: in real) return real;
  function realmax (x, y: in real) return real;
  function realmin (x, y: in real) return real;
  procedure uniform (variable seed1,seed2 : inout positive;
                     variable x: out real);
  function sqrt (x: in real) return real;
  function cbrt (x: in real) return real;
  function "**" (x: in integer; y: in real) return real;
  function "**" (x: in real; y : in real)   return real;
  function exp  (x: in real) return real;
  function log  (x: in real) return real;
  function log2 (x: in real) return real;
  function log10(x: in real) return real;
  function log  (x: in real; base: in real) return real;
  function sin  (x: in real) return real;
  function cos  (x: in real) return real;
  function tan  (x: in real) return real;
  function arcsin  (x: in real) return real;
  function arcos   (x: in real) return real;
  function arctan  (y: in real) return real;
  function arctan  (y: in real; x: in real) return real;
  function sinh    (x: in real) return real;
  function cosh    (x: in real) return real;
  function tanh    (x: in real) return real;
  function arcsinh (x: in real) return real;
  function arccosh (x: in real) return real;
  function arctanh (x: in real) return real;
end math_real;
```

Complex Mathematical Operations

```
use work.math_real.all;
package math_complex is
```

```
    constant copyrightnotice: string := "Copyright 1996 IEEE. All
rights reserved.";
    ------------------------------------------------------------
    -- Type Definitions
    ------------------------------------------------------------
    type complex is
        record
         re : real;  --real part
         im :  real;  --imaginary part
        end record;
    subtype positive_real   is real range 0.0 to real'high;
    subtype principal_value is real range -math_pi to math_pi;
    type complex_polar is
        record
        mag : positive_real; -- magnitude
        arg : principal_value;-- angle in radians; -math_pi is illegal
        end record;
    ------------------------------------------------------------
    -- Constant Definitions
    ------------------------------------------------------------
    constant math_cbase_1 : complex := complex'(1.0, 0.0);
    constant math_cbase_j : complex := complex'(0.0, 1.0);
    constant math_czero   : complex := complex'(0.0, 0.0);
    ------------------------------------------------------------
    -- Overloaded equality and inequality operators for complex_polar
    -- (equality and inequality operators for complex are predefined)
    ------------------------------------------------------------
function "="(I:in complex_polar; r : in complex_polar) return boolean;
function "/="(I:in complex_polar; r: in complex_polar) return boolean;
------------------------------------------------------------
-- Function Declarations
------------------------------------------------------------
function cmplx (x: in real;  y : in real:= 0.0) return complex;
function get_principal_value (x : in real) return principal_value;
function complex_to_polar (z : in complex) return complex_polar;
function polar_to_complex (z : in complex_polar) return complex;
function "abs" (z: in complex) return positive_real;
function "abs" (z: in complex_polar) return positive_real;
function arg (z: in complex) return principal_value;
function arg (z: in complex_polar) return principaLvalue;
function "-" (z: in complex) return complex;
function "-" (z: in complex_polar)  return complex_polar;
function conj (z: in complex) return complex;
function conj (z: in complex_polar) return complex_polar;
function sqrt (z: in complex) return complex;
function sqrt (z: in complex_polar) return complex_polar;
function exp (z: in complex) return complex;
function exp (z: in complex_polar) return complex_polar;
function log (z: in complex) return complex;
function log2 (z: in complex) return complex;
```

```
function log10 (z: in complex) return complex;
function log (z: in complex_polar)  return complex_polar;
function log2 (z: in complex_polar) return complex_polar;
function log10 (z: in complex_polar) return complex_polar;
function log (z: in complex;  base: in real) return complex;
function log (z: in complex_polar; base: in real) return complex_polar;
function sin (z : in complex) return complex;
function sin (z : in complex_polar) return complex_polar;
function cos (z : in complex) return complex;
function cos (z : in complex_polar) return complex_polar;
function sinh (z : in complex) return complex;
function sinh (z : in complex_polar) return complex_polar;
function cosh (z : in complex) return complex;
function cosh (z : in complex_polar) return complex_polar;
------------------------------------------------------------
-- Arithmetic Operators
------------------------------------------------------------
function "+" ( I in complex; r: in complex ) return complex;
function "+" ( I in real; r: in complex ) return complex;
function "+" ( I in complex; r: in real ) return complex;
function "+" (I in complex_polar; r : in complex_polar) return complex_polar;
function "+" ( I in real; r: in complex_polar ) return complex_polar;
function "+" ( I in complex_polar; r: in real ) return complex_polar;
function "-" ( I in complex; r: in complex ) return complex;
function "-" ( I in real; r: in complex ) return complex;
function "-" ( I in complex; r: in real ) return complex;
function "-" ( I in complex_polar; r : in complex_polar ) return complex_polar;
function "-" ( I in real; r: in complex_polar ) return complex_polar;
function "-" ( I in complex_polar; r: in real ) return complex_polar;
function "*" ( I in complex; r: in complex ) return complex;
function "*" ( I in real; r: in complex ) return complex;
function "*" ( I in complex; r: in real ) return complex;
function "*" ( I in complex_polar; r : in complex_polar ) return complex_polar;
function "*" ( I in real; r: in complex_polar ) return complex_polar;
function "*" ( I in complex_polar; r: in real ) return complex_polar;
function"/" (I in complex; r: in complex) return complex;
function"/" (I in real; r: in complex) return complex;
function"/" (I in complex; r: in real) return complex;
function"/" (I in complex_polar;  r: in complex_polar) return complex_polar;
function"/" (I in real; r: in complex_polar) return complex_polar;
function"/" (I in complex_polar;  r: in real) return complex_polar;
end math_complex;
```

Standard1076.3 VHDL Synthesis Packages

Bit-Vector Numeric Operations

```
package numeric_bit is
 constant copyrightnotice: string := "Copyright © 1997 IEEE. All rights reserved.";
------------------------------------------------------------------
```

```
---- Numeric Array Type Definitions
------------------------------------------------------------------
type unsigned is array (natural range <>) of bit;
type signed is array (natural range <>) of bit;
------------------------------------------------------------------
---- Arithmetic Operators:
------------------------------------------------------------------
function "abs" (arg : signed) return signed;
function "-" (arg : signed) return signed;
function "+" (I, r: unsigned) return unsigned;
function "+" (I, r: signed) return signed;
function "+" (I: unsigned; r: natural) return unsigned;
function "+" (I: natural; r: unsigned) return unsigned;
function "+" (I: integer; r: signed) return signed;
function "+" (I: signed; r: integer) return signed;
function "-" (I, r: unsigned) return unsigned;
function "-" (I, r: signed) return signed;
function "-" (I: unsigned; r: natural) return unsigned;
function "-" (I: natural; r: unsigned) return unsigned;
function "-" (I: signed; r: integer) return signed;
function "-" (I: integer; r: signed) return signed;
function "*" (I, r: unsigned ) return unsigned;
function "*" (I, r: signed) return signed;
function "*" (I: unsigned; r: natural) return unsigned;
function "*" (I: natural; r: unsigned) return unsigned;
function "*" (I: signed; r: integer) return signed;
function "*" (I: integer; r: signed) return signed;
function "/" (I, r: unsigned) return unsigned;
function "/" (I, r: signed) return signed;
function "/" (I: unsigned;  r: natural) return unsigned;
function "/" (I: natural; r: unsigned) return unsigned;
function "/" (I: signed; r: integer) return signed;
function "/" (I: integer; r: signed) return signed;
function "rem" (I, r: unsigned ) return unsigned;
function "rem" (I, r: signed) return signed;
function "rem" (I: unsigned;  r: natural) return unsigned;
function "rem" (I: natural; r: unsigned ) return unsigned;
function "rem" (I: signed; r: integer) return signed;
function "rem" (I: integer; r: signed) return signed;
function "mod" (I, r: unsigned) return unsigned;
function "mod" (I, r: signed ) return signed;
function "mod" (I: unsigned; r: natural) return unsigned;
function "mod" (I: natural; r: unsigned) return unsigned;
function "mod" (I: signed; r: integer) return signed;
function "mod" (I: integer; r: signed) return signed;
------------------------------------------------------------------
---- Comparison Operators
------------------------------------------------------------------
function ">" (I, r: unsigned) return boolean;
function ">" (I, r: signed) return boolean;
```

```
function ">" (I: natural; r: unsigned) return boolean;
function ">" (I: integer; r: signed) return boolean;
function ">" (I: unsigned; r: natural) return boolean;
function ">" (I: signed; r: integer) return boolean;
function "<" (I, r: unsigned) return boolean;
function "<" (I, r: signed) return boolean;
function "<" (I: natural; r: unsigned) return boolean;
function "<" (I: integer; r: signed) return boolean;
function "<" (I: unsigned; r: natural) return boolean;
function "<" (I: signed; r: integer) return boolean;
function "<=" (I, r: unsigned) return boolean;
function "<=" {I, r: signed) return boolean;
function "<=" (I: natural; r: unsigned) return boolean;
function "<=" (I: integer;  r: signed) return boolean;
function "<=" (I: unsigned;  r: natural) return boolean;
function "<=" (I: signed; r: integer) return boolean;
function ">=" (I, r: unsigned) return boolean;
function ">=" (I, r: signed) return boolean;
function ">=" (I: natural;  r: unsigned) return boolean;
function ">=" (I: integer;  r: signed) return boolean;
function ">=" (I: unsigned;  r: natural) return boolean;
function ">=" (I: signed; r: integer) return boolean;
function "=" (I, r: unsigned) return boolean;
function "=" (I, r: signed) return boolean;
function "=" (I: natural; r: unsigned) return boolean;
function "=" (I: integer; r: signed) return boolean;
function "=" (I: unsigned; r: natural) return boolean;
function "=" (I: signed; r: integer) return boolean;
function "/=" (I, r: unsigned) return boolean;
function "/=" (I, r: signed) return boolean;
function "/=" (I: natural;  r: unsigned) return boolean;
function "/=" (I: integer;  r: signed) return boolean;
function "/=" (I: unsigned; r: natural) return boolean;
function "/=" (I: signed;  r: integer) return boolean;
-------------------------------------------------------------------
---- Shift and Rotate Functions
-------------------------------------------------------------------
function shift_left (arg : unsigned; count: natural) return unsigned;
function shift_right (arg : unsigned;count: natural) return unsigned;
function shift_left (arg : signed; count: natural) return signed;
function shift_right (arg : signed; count: natural) return signed;
function rotate_left (arg : unsigned; count: natural) return unsigned;
function rotate_right (arg : unsigned;count: natural) return unsigned;
function rotate_left (arg : signed; count: natural) return signed;
function rotate_right (arg : signed; count: natural) return signed;
function "sll" (arg : unsigned;  count:  integer) return unsigned;
function "sll" (arg : signed; count: integer) return signed;
function "srl" ( arg : unsigned;  count: integer) return unsigned;
function "srl" (arg : signed; count: integer) return signed;
function "rol" (arg : unsigned;  count: integer) return unsigned;
```

```
function "rol" (arg : signed; count: integer) return signed;
function "ror" (arg : unsigned; count: integer) return unsigned;
function "ror" (arg : signed; count: integer) return signed;
-------------------------------------------------------------
---- Resize Functions
-------------------------------------------------------------
function resize (arg : signed;  new_size : natural) return signed;
function resize (arg : unsigned; new_size : natural) return unsigned;
-------------------------------------------------------------
---- Conversion Functions
-------------------------------------------------------------
function to_integer (arg : unsigned) return natural;
function to_integer (arg : signed) return integer;
function to_unsigned (arg, size : natural) return unsigned;
function to_signed (arg : integer; size : natural) return signed;
-------------------------------------------------------------
---- Logical Operators
-------------------------------------------------------------
function "not" (I: unsigned) return unsigned;
function "and" (I, r: unsigned) return unsigned;
function "or" (I, r: unsigned) return unsigned;
function "nand" (I, r: unsigned) return unsigned;
function "nor" (I, r: unsigned) return unsigned;
function "xor" (I, r: unsigned) return unsigned;
function "xnor" (I, r: unsigned) return unsigned;
function "not" (I: signed) return signed;
function "and" (I, r: signed) return signed;
function "or" (I, r: signed) return signed;
function "nand" (I, r: signed) return signed;
function "nor" (I, r: signed) return signed;
function "xor" (I, r: signed) return signed;
function "xnor" (I, r: signed) return signed;
-------------------------------------------------------------
---- Edge Detection Functions
-------------------------------------------------------------
function rising_edge (signal s : bit) return boolean;
function falling_edge (signal s : bit) return boolean;
end numeric_bit;
```

Standard-Logic-Vector Numeric Operations

```
library ieee;
use ieee.std_logic_1164.all;

package numeric_std is
constant copyrightnotice: string := "Copyright © 1997 IEEE. All
rights reserved.";
-------------------------------------------------------------
---- Numeric Array Type Definitions
-------------------------------------------------------------
type unsigned is array (natural range <>) of std_logic;
```

```
type signed is array (natural range <>) of std_logic;
------------------------------------------------------------------
---- Declaration of Arithmetic Operators, Comparison Operators,
---- Shift and Rotate Functions, Resize Functions, Conversion Functions,
---- and Logical Operators exactly as in package numeric_bit.
------------------------------------------------------------------
...
------------------------------------------------------------------
---- Match Functions
------------------------------------------------------------------
function std_match (I, r: std_ulogic) return boolean;
function std_match (I, r: unsigned) return boolean;
function std_match (I, r: signed) return boolean;
function std_match (I, r: std_logic_vector) return boolean;
function std_match (I, r: std_ulogic_vector) return boolean;
------------------------------------------------------------------
---- Translation Functions
------------------------------------------------------------------
function to_01 (s: unsigned; xmap : std_logic := '0') return unsigned;
function to_01 (s : signed;  xmap : std_logic := '0') return signed;
end numeric_std;
```

VHDL-87

The overloaded versions of the shift, rotate and xnor operators must be commented out of the numeric_bit and numeric_std packages if VHDL-87 compatibility is required.

Std_logic_unsigned

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

package STD_LOGIC_UNSIGNED is

  function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;

  function "+"(L: STD_LOGIC_VECTOR; R: INTEGER) return
    STD_LOGIC_VECTOR;

  function "+"(L: INTEGER; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;

  function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return
    STD_LOGIC_VECTOR;

  function "+"(L: STD_LOGIC; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;

  function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;

  function "-"(L: STD_LOGIC_VECTOR; R: INTEGER) return
    STD_LOGIC_VECTOR;
```

```
function "-"(L: INTEGER; R: STD_LOGIC_VECTOR) return
  STD_LOGIC_VECTOR;
function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return
  STD_LOGIC_VECTOR;
function "-"(L: STD_LOGIC; R: STD_LOGIC_VECTOR) return
  STD_LOGIC_VECTOR;
function "+"(L: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
function "*"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  STD_LOGIC_VECTOR;
function "<"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  BOOLEAN;
function "<"(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
function "<"(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
function "<="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  BOOLEAN;
function "<="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
function "<="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
function ">"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  BOOLEAN;
function ">"(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
function ">"(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
function ">="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  BOOLEAN;
function ">="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
function ">="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
function "="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  BOOLEAN;
function "="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
function "="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
function "="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
  BOOLEAN;
function "="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
function "="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
function SHL(ARG:STD_LOGIC_VECTOR;COUNT:
  STD_LOGIC_VECTOR)   return STD_LOGIC_VECTOR;
function SHR(ARG:STD_LOGIC_VECTOR;COUNT:
  STD_LOGIC_VECTOR)   return STD_LOGIC_VECTOR;
function CONV_INTEGER(ARG: STD_LOGIC_VECTOR) return
  INTEGER;
```

```
-- remove this since it is already in std_logic_arith
-- function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE:  INTEGER)
-- return STD_LOGIC_VECTOR;
end STD_LOGIC_UNSIGNED;
```

Std_logic_signed

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
package STD_LOGIC_SIGNED is
  function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC_VECTOR; R: INTEGER) return
    STD_LOGIC_VECTOR;
  function "+"(L: INTEGER; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return
    STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR)
    return  STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC_VECTOR; R: INTEGER) return
    STD_LOGIC_VECTOR;
  function "-"(L: INTEGER; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return
    STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "ABS"(L: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "*"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    STD_LOGIC_VECTOR;
  function "<"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    BOOLEAN;
  function "<"(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
  function "<"(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
  function "<="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
```

```
    BOOLEAN;
  function "<="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
  function "<="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
  function ">"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    BOOLEAN;
  function ">"(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
  function ">"(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
  function ">="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    BOOLEAN;
  function ">="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
  function ">="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
  function "="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    BOOLEAN;
  function "="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
  function "="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
  function "="(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return
    BOOLEAN;
  function "="(L: STD_LOGIC_VECTOR; R: INTEGER) return BOOLEAN;
  function "="(L: INTEGER; R: STD_LOGIC_VECTOR) return BOOLEAN;
  function SHL(ARG:STD_LOGIC_VECTOR;COUNT:
    STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function SHR(ARG:STD_LOGIC_VECTOR;COUNT:
    STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function CONV_INTEGER(ARG: STD_LOGIC_VECTOR) return INTEGER;
-- remove this since it is already in std_logic_arith
-- function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
-- return STD_LOGIC_VECTOR;
end STD_LOGIC_SIGNED;
```

Textio

```
------------------------------------------------------------------
-- ANSI/IEEE Std 1076-1993
-- IEEE Standard VHDL Language Reference Manual
------------------------------------------------------------------
package TEXTIO is
-- Type Definitions for Text I/O
--
  type LINE is access STRING;     -- A LINE is a pointer to a STRING value.
  type TEXT is file of STRING;    -- A file of variable-length ASCII records.
  type SIDE is (RIGHT, LEFT);     -- For justifying output data within fields.
  subtype WIDTH is NATURAL;       -- For specifying widths of output fields.
```

```
--
-- Standard text files:
-- Note these are different from VHDL'87
--
  file INPUT : TEXT open READ_MODE  is "STD_INPUT";
  file OUTPUT: TEXT open WRITE_MODE is "STD_OUTPUT";
--
-- Input routines for standard types:
  procedure READLINE (file F: TEXT; L: out LINE);
  procedure READ (L: inout LINE; VALUE: out BIT; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out BIT);
  procedure READ (L: inout LINE; VALUE: out BIT_VECTOR; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out BIT_VECTOR);
  procedure READ (L: inout LINE; VALUE: out CHARACTER; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out CHARACTER);
  procedure READ (L: inout LINE; VALUE: out INTEGER; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out INTEGER);
  procedure READ (L: inout LINE; VALUE: out REAL; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out REAL);
  procedure READ (L: inout LINE; VALUE: out STRING; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out STRING);
  procedure READ (L: inout LINE; VALUE: out TIME; GOOD: out BOOLEAN);
  procedure READ (L: inout LINE; VALUE: out TIME);
-- Output routines for standard types
  procedure WRITELINE (file F: TEXT; L: inout LINE);
  procedure WRITE (L: inout LINE; VALUE: in BIT;
     JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);
  procedure WRITE (L: inout LINE; VALUE: in BIT_VECTOR;
     JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);
  procedure WRITE (L: inout LINE; VALUE: in BOOLEAN;
     JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);
  procedure WRITE (L: inout LINE; VALUE: in CHARACTER;
     JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);
  procedure WRITE (L: inout LINE; VALUE: in INTEGER;
     JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);
  procedure WRITE (L: inout LINE; VALUE: in REAL;
 JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0; DIGITS: in NATURAL :=0);
  procedure WRITE (L: inout LINE; VALUE: in STRING;
     JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);
  procedure WRITE (L: inout LINE; VALUE: in TIME;
  JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0; UNIT : in TIME ;= ns);
end TEXTIO;
```

Список литературы на русском языке

1р. Антонов А. П. Язык описания цифровых устройств Altera HDL. М.: Радио-Софт, 2002.

2р. Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL Пер. с англ. М.: Мир, 1992.

3р. Бибило П. Н. Основы языка VHDL. М.: Солон-Р, 2000.

4р. Бибило П. Н. Системы проектирования. Основы языка VHDL. М.:Солон-Р, 2002.

5р. Бибило П. Н. Синтез логических схем с использованием языка VHDL. М.: Солон-Р, 2002.

6р. Глушков В. М., Капитонова Ю. В., Летичевский А. А. О языках описания данных в автоматизированных системах проектирования вычислительных машин (ПРОЕКТ) // Кибернетика. 1970. №6.

7р. Грушвицкий Р. И., Мурсаев А. Х., Угрюмов Е. П. Проектирование систем на микросхемах программируемой логики. СПб.: БХВ-Петербург, 2002.

8р. Закревский А. Д. Алгоритмический язык ЛЯПАС и автоматизация синтеза дискретных автоматов. Томск: Изд-во Томского университета, 1966.

9р. Майоров С. А., Новиков Г. И. Структуры электронных вычислительных машин. Л.: Машиностроение, 1979.

10р. Поляков А. К. Язык VHDL в САПР цифровых систем. М.: МЭИ, 1999.

11р. Разевиг В. Д. Система проектирования цифровых устройств OrCAD. М.: Солон-Р, 2000.

12р. Разевиг В. Д. Система проектирования OrCAD 9.2. М.: Солон-Р, 2001.

13р. Стешенко В. Б. ПЛИС фирмы «ALTERA»: элементная база, система проектирования и языки описания аппаратуры. М.: Додэка-XXI, 2002.

14р. Стешенко В. Б. EDA. Практика автоматизированного проектирования радиоэлектронных устройств. М.: Нолидж, 2002.

15р. Толковый словарь по вычислительным системам / Под ред. Иллингуорта; пер. с англ. М.: Машиностроение, 1990.

16р. Угрюмов Е. П. Цифровая схемотехника. СПб.: БХВ — Санкт-Петербург, 2000.

17р. Угрюмов Е. П., Грушвицкий Р. И., Альшевский А. Н. БИС/СБИС с репрограммируемой структурой: Учебное пособие. СПб.: ГЭТУ, 1997.

18р. Уэйкерли Дж. Проектирование цифровых устройств: 2 т. Пер. с англ. М.: Постмаркет, 2002.

19р. Чу Я. Организация ЭВМ и программирования. М.: Мир, 1975.

20р. VHDL для моделирования, синтеза и формальной верифика- ции аппаратуры. Пер. с англ. М.: Радио и связь, 1995.

21р. VHDL'92. Новые свойства языка описания аппаратуры VHDL. Пер. с англ. М.: Радио и связь, 1995.

22р. Энциклопедия кибернетики. Киев: Главная редакция УСЭ, 1975.

23р. Суворова Е. А., Шейнин Ю. Е. Проектирование цифровых систем на VHDL. СПб: БХВ — Петербург, 2003.

Список литературы на английском языке

1a. ALDEC's HDL Entry and Simulation Tutorial. — Aldec, Inc.: Henderson, NV 89014 (www.aldec.com).

2a. ALDEC's EVITA InterActive-HDL Tutorial. — Aldec, Inc.: Henderson, NV 89014 (www.aldec.com).

3a. Airiau R., Berge J. M. and Olive V. Circuit Synthesis with VHDL. — Kluwer Academic Publishers, Norwell, MA, 1994.

4a. Armstrong J. R.,Gray F. G. Structured Logic Design with VHDL. — Englewood Cliffs, NJ: Prentice Hall, 1993.

5a. Armstrong J. R. Chip-Level Modeling with VHDL. — Englewood Cliffs, NJ: Prentice Hall, 1988.

6a. Ashenden P. The Designer's Guide to VHDL(2nd ed.). — Morgan Kaufmann Publishers, 2002.

7a. Baker L. VHDL Programming with Advanced Topics. — John Wiley and Sons, Inc., 1993.

8a. Berge J. M. et al. VHDL Designer's Reference. — Kluwer Academic, 1993.

9a. Bhasker J. A VHDL Primer (3rd Edition). — Prentice Hall PTR Upper Saddle River, NJ 07458, 1999.

10a. Brown S., Vranesic Z. Fundamentals of Digital Logic with VHDL Design.-McGraw — Hill, 2000.

11a. Boundary-Scan Description Language (BSDL)-Brief Description www.asset-intertech.com.

12a.Carlson S. Introduction to HDL-Based Design Using VHDL. — Synopsis Inc., 1991.

13a. Carter J. W. Digital Designing with Programmable Logic Devices. — Prentice Hall, 1997.

14a. Chang K. C. Digital Design and Modeling with VHDL and Synthesis. — IEEE Computer Society Press, 1997.

15a. Coelho D. R. The VHDL Handbook. — Kluwer Academic, Norwell, MA, 1989.

16a. Cohen B. VHDL Coding Styles and Methodologies. — Kluwer Academic Publishers, Norwell, MA, 1994.

17a. CHDStd. — www.si2.org/CHDStd.

18a. EIA/IS-103-A Library of Parametrized Modules. — www.edif.org/lpmweb.

19a. EIA-567-A Component Modeling and Interface. — www.edif.org.

20a. EIA-682 Electronic Design Interchange Format(EDIF). — www.edif.org.

21a. Hill D.D. ADLIB: A modular, strongly-typed computer design language. — Proc. of the 4th International Symposium on Computer HDL, IEEE, 1979.

22a. IEEE P1076. 5 VHDL Utility Library. — www.eda.org/libutil.

23a. IEEE P1497 Standard Delay Format . — www.eda.org/sdf.

24a. IEEE P1551 System and Interface Description . — www.eda.org/sid.

25a. IEEE P1577 Object-Oriented VHDL. — www.eda.org/oovhdl.

26a. IEEE Standard 1029.1 — 1998. IEEE Standard for VHDL Waveform and Vector Exchange to Support Design and Test Verification (WAVES) Language.

27a. IEEE Standard 1076 — 1987. IEEE Standards Interpretations.

28a. IEEE Standard 1076 — 2001. IEEE Standard VHDL Language Reference Manual. — www.eda.org/vasg.

29a. IEEE Standard 1076.1 — 1999. IEEE Standard VHDL Analog and Mixed-Signal Extensions. — www.eda.org/vhdl-ams.

30a. IEEE Standard 1076.2 — 1996. IEEE Standard VHDL Mathematical Packages. — www.eda.org/vhdl-math.

31a. IEEE Standard 1076.3 — 1997. IEEE Standard VHDL Synthesis Packages. — www.eda.org/vhdlsynth.

32a. IEEE Standard 1076.4 — 1995. IEEE Standard VITAL Application — Specific Integrated Circuit (ASIC) Modeling Specification. — www.eda.org/vital.

33a. IEEE Standard 1076.6 — 1999. IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis. — www.eda.org/libutil.

34a. IEEE Standard 1164 — 1993. IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164). — www.eda.org.

35a. IEEE Standard 1364 — 1995. IEEE Standard Description Language Based on the Verilog Hardware Description Language. — www.ovi.org/ieee-1364/Welcome.html.

36a. IEEE Standard 1481 — 1999. IEEE Standard for Delay and Power Calculation.-www.eda.org/dpc, www.si2.org/ieee1481.

37a. IEEE Standard 1499 — 1998. IEEE Standard Interface for Hardware Description Models of Electronic Components.-www.eda.org/omf.

38a. IEEE Standard 1149.1 — 1990. IEEE Standard Tests Access Port and Boundary-Scan Architecture.

39a. Lee J. M. Verilog QuickStart. — Kluwer Academic Publishers, 1997.

40a. Lee S.Design of Computers and Other Complex Digital Devices. — Prentice Hall, 2000.

41a. Leung S. S., Shanblatt M. A. ASIC System Design with VHDL. — Boston: Kluwer Academic, 1989.

42a. Lipsett R., Schaefer C., Ussery C. VHDL: Hardware Description and Design. — Kluwer Academic, Norwell, MA, 1989.

43a. MAX + PLUS II. Programmable Logic Development System. Getting Started (ver. 10.0, 2000). — Altera Corporation: San Jose, CA 95134.

44a. ModelSim. Xilinx User's Manual (ver. 5.5b, 2001). — Model Technology Inc., Xilinx, Inc.: San Jose, CA 95124 ().

45a. ModelSim. Xilinx Tutorial (ver. 5.5b, 2001).-Model Technology Inc., Xilinx, Inc.: San Jose, CA 95124 (www.xilinx.com).

46a. ModelSim. Xilinx Command Reference (ver. 5.5b, 2001). — Model Technology Inc., Xilinx, Inc.: San Jose, CA 95124 ().

47a. Navabi Z. VHDL. Analysis and Modeling of Digital Systems(2nd ed.). — McGraw-Hill, 1998.

48a. ORCAD Capture for Windows. User's Guide. — Cadence Design Systems: San Jose, CA 95134 (www.cadence.com).

49a. ORCAD Express for Windows. User's Guide. — Cadence Design Systems, San Jose, CA 95134 (www.cadence.com).

50a. Ott D. E., Wilderotter T. J. A Designer's Guide to VHDL Synthesis. — Kluwer Academic Publishers, Norwell, MA, 1994.

51a. OVI Advanced Library Format. — www.eda.org/alf.

52a. OVI Design Constraints. — www.vhdl.org/dcwg.

53a. OVI Verilog-AMS. — www.eda.org/verilog-ams.

54a. Palnitkar S. Verilog HDL. A Guide to Digital Design and Synthesis. — SunSoft Press, 1996.

55a. Parker A., Wallace J. SLIDE, an I/O Hardware Description Language. — IEEE Transactions on Computers, vol. C-30, 1981.

56a. Pellerin D., Holley M. Digital Design Using ABEL.-Prentice Hall, 1994.

57a. Pellerin D.,Taylor D. VHDL Made Easy! — Prentice Hall, 1997.

58a. Perry D.VHDL(3rd ed.). — McGraw-Hill, 1999.

59a. Piloty R., Barbacci M., Borrione D. etc. CONLAN Report. — Springer-Verlag, 1983.

60a. Rushton A. VHDL for Logic Synthesis(2nd ed.). — John Wiley & Sons, 1998.

61a. Salcic Z. VHDL and FPLDs in Digital Systems Design. Prototyping and Customization. — Kluwer Academic Publishers, 1998.

62a. Salcic Z., Smailagic A. Digital Systems Design and Prototyping Using Field Programmable Logic and Hardware Description Languages (2nd ed.). — Kluwer Academic Publishers, 2000.

63a. Sheppard A., Giridhar V. Hierarchical Scan Description Language. Syntax Specification (Revision A — 1992). — ASSET InterTech,Inc; Texas Instruments Inc. — www.asset-intertech.com.

64a. SI2 Open Library Architecture. — www.si2.org/ola.

65a. Sjoholm S., Lindh L.VHDL for Designers. — Prentice Hall, 1997.

66a. Skahill K. VHDL for Programmable Logic. — Addison-Wesley, 1996.

67a. SLDL Initiative. — www.inmet.com/SLDL, www.ittc.ukans.edu/Projects /SLDG/rosetta.

68a. Smith D.J. A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs Using VHDL or Verilog. — Doone Publications, 1996.

69a. Smith M.J.S.Application-Specific Integrated Circuits. — Addison — Wesley, 1997.

70a. Thomas D. E., Moorby P.R. The Verilog Hardware Description Language (5th ed.). — Kluwer Academic Publishers, 2002.

71a. Van den Bout D. The Practical Xilinx Designer Lab Book, Ver. 1.5. — Prentice Hall, 1999.

72a. Virtual Socket Interface Alliance. — www.vsi.org

73a. VHDL Programming Language Interface — www.eda.org/vhdlpli.

74a. VHDL Simili User 's Manual (ver. 2.1 2002). — Symphony EDA: Beaverton, OR 97007.

75a. Wakerly J. F. Digital Design (Principles and Practice)(3rd ed.). — Prentice Hall, 2000.

76a. Warp VHDL Development System. User's Guide(ver. 4 1996). — Cypress Semiconductor: San Jose, CA 95124 (www.cypress.com).

77a. Warp VHDL Development System. Reference Manual(ver. 4 1996). — Cypress Semiconductor: San Jose, CA 95124 (www.cypress.com).

78a. Xilinx Foundation Series On-Line Help System (ver. F2.1i). — Aldec, Inc.: Henderson, NV 89014 (www.aldec.com), Xilinx, Inc.: San Jose, CA 95124 ().

79a. Xilinx Foundation Series. Synthesis and Simulation Design Guide. — Xilinx, Inc.: San Jose, CA 95124 (www.xilinx.com).

80a. Xilinx Foundation Series. Foundation Series 2.li User Guide. — Xilinx, Inc.: San Jose, CA 95124 (www.xilinx.com).

81a. Yalamanchili S. VHDL Starter's Guide. — Prentice Hall, 1998.

82a. Ashenden P., Peterson G., Teegarden D. The System Designer's Guide to VHDL-AMS. — Morgan Kaufmann Publishers, 2003.

Англо-русский словарь — указатель терминов

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Abstract State Machine	Абстрактный цифровой автомат
Abstract Structure	Абстрактная структура
Abstraction Level	Уровень абстракции
Access Time	Время доступа
Access Type	Тип данных Access языка VHDL
Accumulator Register	Регистр аккумулятора
Actual Time Behavior	Адекватное поведение во времени
Adder	Сумматор
Address Register	Регистр адреса
Algorithm	Алгоритм
Analysis	Анализ
Architectural Body	Архитектурное тело модели на VHDL
Architecture	Архитектура модели на VHDL
Architecture Declarative Part	Декларативная часть архитектуры
Architecture Statement Part	Исполняемая часть архитектуры
Area	Площадь
Arrangement of a System	Компоновка системы
Array Type	Тип данных Array языка VHDL
ASIC (Application Specific Integrated Circuit)	Интегральная схема, изготовленная для конкретного заказчика
ASM (Algorithmic State Machine)	Цифровой автомат в форме ASM
ASM Chart	Блок-схема цифрового автомата в форме ASM
Assembler	Язык Ассемблер
Assert Statement	Тип данных Assert языка VHDL
Assigned ASM Table	Кодированная таблица цифрового автомата в форме ASM
Assigning Values to Variables	Присвоение значений переменным
Assignment Statement	Оператор присваивания значений
Asynchronous State Machine	Асинхронный цифровой автомат
ATE (Automate Test Equipment)	Автоматическое оборудование для тестирования
ATPG (Automatic Test Pattern Generation)	Процесс автоматической генерации тестов
Attribute	Атрибут языка VHDL

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Automatic Synthesis	Автоматический синтез схем
Behavioral Architecture	Поведенческая архитектура
Behavioral Description	Поведенческое описание
Behavioral Level	Поведенческий уровень моделирования
Behavioral Model	Поведенческая модель
Behavioral Synthesis	Синтез на поведенческом уровне
Behavioral Synthesis Tool	Инструмент поведенческого синтеза
Behavioral VHDL Statements	Операторы «поведенческого» VHDL
Best Case	Наилучший случай
Bidirectional Signal	Двунаправленный сигнал
Binary Multiplication	Бинарное умножение
Bit String	Битовая строка
Bit Type	Тип данных Bit языка VHDL
Bit-Vector Type	Тип данных Bit-Vector языка VHDL
Black Box	«Черный ящик»
Black Box Principle	Принцип «черного ящика»
Block Diagram	Блочная диаграмма
Block Statement	Оператор блока языка VHDL
Boolean Equation	Булево уравнение
Boolean Type	Тип данных Boolean языка VHDL
Bottom Up Design	Восходящее проектирование
Building Hierarchies	Иерархии в построении моделей
CAD (Computer Aided Design) System	САПР
Carry Look Ahead Adder	Сумматор с ускоренным формированием переноса
Case-When Statement	Тип данных Case-When языка VHDL
Cellular Multiplier	Умножитель в виде массива ячеек
Character	Символ
Character Literal	Символьный литерал
Character Type	Тип данных Character языка VHDL
Clock Cycle	Период синхросигнала
Combinational Logic	Комбинационная логика
Combinational Logic Description	Описание комбинационной логики
Comment	Комментарий
Comparator	Компаратор
Component Choise	Выбор компонента

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Component Instance	Экземпляр компонента
Component Instantiation	Создание экземпляров компонентов
Component Library	Библиотека компонентов
Composite System	Составная (сложная) система
Composite Type	Составной тип данных языка VHDL
Computer Model	Компьютерная модель
Computer Model Netlist	Список соединений компьютерной модели
Computer Simulation	Компьютерное моделирование
Concatenation	Конкатенация
Concurrency	Параллельность функционирования
Concurrent (Parallel) Statement	Параллельный оператор
Concurrent Language Construction	Параллельная конструкция языка
Conditional Execution	Выполнение по условию
Conditional Output Box	Блок условного выхода ASM
Conditional Signal Assignment Statement	Условный оператор присвоения значений сигналам VHDL
Configuration	Конфигурация
Configuration Declaration	Декларация конфигурации
Constant	Константа
Constrained Array Type	Тип данных Array языка VHDL (с границами)
Constraint	Ограничение
Control Unit	Устройство управления
CPLD (Complex Programmable Logic Device)	Сложная ПЛИС
CPU (Central Processing Unit)	Центральное процессорное устройство
Current Focus of Design	Текущий этап («фокус») процесса проектирования
Custom Integrated Circuit	Интегральная схема, изготовленная по специальному заказу
Dash	Тире
Data Flow Style	Стиль потока данных
Data Path	Тракт передачи данных
Data Storage and Transformation	Хранение и преобразование данных
Data Storage Register	Регистр хранения данных
Data Transformation Unit	Устройство преобразования данных
Data Type	Тип данных

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Decision Box	Блок принятия решения ASM
Declarative Part	Декларативная часть
Delay	Задержка
Delay Representation	Представление задержки
Design	Процесс проектирования, проект
Design Complexity	Сложность проекта
Design Computer Tool	Компьютерный инструмент проектирования
Design Hierarchy	Иерархия проекта
Design Implementation	Реализация проекта
Design Method	Метод проектирования
Design Specification	Спецификация проекта
Design Tool	Инструмент проектирования
Design Unit	Структурная единица проекта
Design Verification	Верификация проекта
Development Method	Метод разработки
Development Phase	Этап разработки
Development Time	Время разработки
Device	Устройство
Different Abstraction Levels	Различные уровни абстракции
Differential Equation	Дифференциальное уравнение
Digital Development Environment	Среда цифрового проектирования
Discrete Event Simulation	Цифровое событийное моделирование
Discrete Type	Тип данных Discrete языка VHDL
DUT (Device Under Test)	Проверяемый объект (устройство)
Elaboration	Уточнение (предвыполнение) модели
Enclosing Model	Обьемлющая модель
Entity (Port) Declaration	Декларация интерфейса модели
Enumerated Type	Перечислимый тип
Error Message	Сообщение об ошибке
Evaluating Expression	Оценка выражения
Event	Событие
Execution	Исполнение
Executive Part	Исполняемая часть
Fan-Out	Разветвление по выходу
Fetching	Извлечение информации из памяти

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
File Type	Тип данных File языка VHDL
Finish Time	Время окончания
Fitter	Компоновщик
Fitting	Компоновка
Flip-Flop (FF)	Триггер
Floating-point Type	Тип данных с плавающей точкой языка VHDL
Floor Plan	План расположения на слое микросхемы
Flow Design	Поток процесса проектирования
FPGA (Field Programmable Gate Array)	Программируемый массив вентилей
FSM (Finite State Machine)	Конечный цифровой автомат
Full Adder	Полный сумматор
Function	Функция
Functional Domain	Область функциональных моделей
Functional Level	Функциональный уровень
Functional Module	Функциональный модуль
Functional Simulation	Функциональное моделирование
Functional Verification	Функциональная верификация
Functioning Prototype	Функционирующий прототип
Gate	Вентиль
Gate Delay	Задержка вентиля
Gate Description	Описание на вентильном уровне
Gate Level	Вентильный уровень
Generate Function	Функция генерации
Generate Statement	Оператор генерации языка VHDL
Generic Component	Типовой компонент
Generics	Конструкция VHDL, позволяющая использовать параметры при описании схем
Geometric Domain	Область геометрических моделей
Good Component	Исправный компонент
Graphical Design Entry	Графический ввод проекта
Graphics User Interface	Графический интерфейс пользователя
Guard Expression	Охранное выражение
Guarded Block	Охраняемый блок
Hardware Accelerator	Аппаратный ускоритель
Hardware Component	Компонент цифровой схемы

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Hardware Description	Описание аппаратуры (цифровой схемы)
Hardware Description Language (HDL)	Язык описания аппаратуры
Hardware Operating System	операционная система «Аппаратная»
Hardware Synthesis	Синтез аппаратуры
Hardware Units	Структурная единица проекта
HDL Text Editor	Текстовый редактор для языка VHDL
Hierarchically Composed System	Иерархическая составная система
Hierarchies	Иерархия
Hierarchy Design	Иерархическое проектирование
High Abstraction Level	Высокий уровень абстракции
High Impedance (Z)	Состояние высокого импеданса
Highest Behavioral Level	Наивысший поведенческий уровень
High-Level Hardware Design Language	Язык описания аппаратуры высокого уровня
Hold Test	Проверка удержания сигнала
Hybrid Model	Гибридная модель
I/O Device	Устройство ввода-вывода
I/O Signals	Сигналы входа-выхода
Identifier	Идентификатор
IEEE (Institute of Electronic Electrical Engineers)	Институт инженеров по электронике и электротехнике
If-Then-Else Statement	Оператор If-Then-Else языка VHDL
Inertial Delay	Инерциальная задержка
Inference	Логический вывод
Inheritance	Наследование
Initial Value	Первоначальное значение
Initialization Phase	Этап инициализации
Input Data	Входные данные
Input Port	Входной порт
Inputs	Входы
Instantiation Component Statement	Оператор получения экземпляров компонент
Integer Literal	Литерал-целое число языка VHDL
Integer Type	Тип данных Integer языка VHDL
Integrated Prototype Model	Интегрированная модель прототипа
Interconnect Delay	Задержка межсоединения
Interconnection	Межсоединение
Interface Signal	Сигнал интерфейса схемы

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Internal Node/Interconnection	Внутренние узел/межсоединение
Internal Signal	Внутренний сигнал архитектуры
Internal Wire Declaration	Декларация внутреннего соединения
IR (Instruction Register)	Регистр команд
Language Abstraction	Абстракция языка
Language Assistant	Мастер языковых конструкций
Layout Level	Уровень формирования топологии микросхемы
Leaf Component	Лист компонента в дереве проекта
Lexical Element	Лексический компонент
Life Cycle	Жизненный цикл изделия
Life Time	Время жизни изделия
Load Delay	Задержка загрузки
Logic Analyzer	Логический анализатор
Logic Level	Логический (вентильный) уровень
Logic Synthesis	Логический синтез
Logic Synthesis Tool	Инструмент логического синтеза
Loop Statement	Оператор цикла языка VHDL
Low-Level Hardware Design Language	Язык описания аппаратуры низкого уровня
Machine Code	Машинный код
MAR (Memory Address Register)	Регистр адреса при обращении к памяти
Mathematical Statement	Математический оператор
Maximum Design Reliability	Максимальная надежность проектирования
Mealy FSM	Конечный цифровой автомат Мили
Memory	Память
Microcode	Микрокод
Microprocessor	Микропроцессор
Minimum Cost	Минимальная цена
Minimum Design Time	Минимальное время проектирования
Mix Level Simulator	Симулятор смешанного уровня
Mixed Architectural Body	Смешанное архитектурное тело
Mixed Design	Смешанное проектирование
Model Accuracy	Точность модели
Modeling Domain	Область моделирования
Modeling Language	Язык моделирования

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Modifiability	Способность к модификации
Moore FSM	Конечный цифровой автомат Мура
Multiplexer	Мультиплексор
Multiplicand	Множитель
Multiplier	Умножитель, Множимое
Multiplier Cell	Ячейка умножителя
Netlist	Список соединений
Number	Число
Object-Based Language	Обьектно-базирующийся язык
Object-Oriented Language	Обьектно-ориентированный язык
Operation	Операция
Operator	Символ операции
Optional Register	Регистр второго операнда для АЛУ
Oscilloscope	Осциллограф
Out Signal File	Файл выходных сигналов
Output Data	Выходные данные
Output Port	Выходной порт
Output Register	Регистр результата (выхода)
Outputs	Выходы
Outputs from Simulation	Результаты моделирования
Outside Stimulus	Внешние воздействия (стимулы)
Package	Пакет
Parallel Adder/Subtractor	Параллельный сумматор-вычитатель
Parallel Process Representation	Представление
Parallel Statements for Signals	Параллельные операторы для сигналов
Partial Product	Частичное произведение
Partioning	Разбиение
PC (Program Counter)	Программный счетчик
PCB (Printed Circuit Board)	Печатная плата
PCB Design	Проектирование печатных плат
Physical Prototype	Физический прототип
Physical Type	Тип данных Physical языка VHDL
Place&Route	Размещение и трассировка
PLD (Programmable Logic Device)	ПЛИС
PLD Design Language	Язык проектирования ПЛИС

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
PMS (Processor-Memory-Switch) Level	Уровень «процессор-память-коммутатор»
Polygon	Полигон
Port	Порт схемы
Port Declaration	Декларация портов схемы
Post Layout Timing Simulation	Временное моделирование после формирования топологии микросхемы
Predefined Attribute	Предопределенный атрибут VHDL
Primary Input	Первичный вход схемы
Primary Output	Первичный выход схемы
Primitive Element	Примитивный элемент
Procedure	Процедура
Process Statement	Оператор процесса VHDL
Processor	Процессор
Programmable Circuit	Программируемая микросхема
Project Specification	Спецификация проекта
Project VHDL Description	VHDL-описание проекта
Propagate Function	Функция распространения
Prototype	Прототип
Prototyping	Создание прототипа
Pure Behavioral Style	Чисто поведенческий стиль проектирования
Purely Behavioral Model	Чисто поведенческая модель
Purely Structural Model	Чисто структурная модель
R/W (Read/Write)	Сигнал чтения-записи
RAM (Random Access Table)	Оперативная память
Rapid Prototyping	Быстрое макетирование
Real Literal	Литерал-вещественное число VHDL
Real Type	Тип данных Real языка VHDL
Record Type	Тип данных Record языка VHDL
Reducing Complexity	Сокращение сложности проекта
Redundant Hardware Units (Functions)	Избыточные единицы аппаратуры (функции)
Register	Регистр
Register-Transfer Language	Язык межрегистровых передач
Repeated Execution	Повторное исполнение
Repetitive Pattern	Повторяющаяся комбинация
Requirement Design Specification	Требование спецификации проекта

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Reserved Word	Зарезервированное слово языка
Reusable Component	Многократно используемый компонент
Ripple-Carry Adder	Волновой сумматор с последовательным формированием переноса
Root Design	Корень дерева проекта
Rotate Left Logical Operator	Операция циклического логического сдвига влево
Rotate Right Logical Operator	Операция циклического логического сдвига вправо
RTL (Register Transfer Level)	Уровень межрегистровых передач
RTL Synthesis	Синтез на уровне межрегистровых передач
Scalar Type	Скалярный тип данных языка VHDL
Scan Path Techniques	Техника сканирования путей в цифровых схемах
Scheduling a Transaction	Планирование транзакций
Schematic Component	Схемотехнический компонент
Schematics	Среда схемотехнического проектирования
Scheme	Схема
Selected Signal Assignment Statement	Оператор выборочного присвоения значений сигналам VHDL
Semantics	Семантика
Sensitivity List	Список чувствительности проекта
Sequential Language Construction	Последовательная конструкция языка
Sequential Logic	Последовательная логика
Sequential Logic Description	Описание последовательной логики
Sequential Statement	Последовательный оператор
Serial Adder	Последовательный сумматор
Setup Test	Проверка установки сигнала
Severity-Level Type	Тип данных Severity языка VHDL
Shared Component	Общий компонент
Shift and Add Multiplier	Умножитель, функционирующий посредством выполнения сдвигов и сложений
Shift Left Arithmetic Operator	Операция арифметического сдвига влево
Shift Left Logical Operator	Операция логического сдвига влево
Shift Register	Регистр сдвига
Shift Right Arithmetic Operator	Операция арифметического сдвига вправо
Shift Right Logical Operator	Операция логического сдвига вправо
Shifter	Сдвигатель, регистр сдвига
Signal	Сигнал

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Signal Assignment Statement	Оператор присваивания значений сигналам VHDL
Signal Declaration	Декларация сигнала
Signal Generator	Генератор сигналов
Silicon Die	Полупроводниковая пластина
Simulation	Моделирование
Simulation Cycle	Цикл моделирования
Simulation System	Система моделирования
Simulation Time	Время моделирования
Simulator	Симулятор, система моделирования
Special Symbol	Специальный символ
Special User Circuit	Микросхема, изготовленная по специальному заказу
Specification	Спецификация
Spike Test	Тест на броски питания
Standard Cell	Стандартная ячейка
Standard Library Cell	Стандартная библиотечная ячейка
Standartized Method	Стандартный метод
Start Time	Время старта
State Editor	Редактор для графического проектирования цифровых автоматов
State Machine	Цифровой автомат
String	Строка
String Literal	Литерал-строка
String Type	Тип данных String языка VHDL
Structural Architecture	Структурная архитектура
Structural Architecture Body	Структурное архитектурное тело
Structural Description	Структурное описание
Structural Domain	Область структурных моделей
Structural Model	Структурная модель
Structural VHDL	VHDL «Структурный»
Structural VHDL Code	Текст на структурном VHDL
Structural VHDL Statements	Операторы структурного VHDL
Subcomponent	Подкомпонент
Subprogram	Подпрограмма
Subprogram Call	Вызов подпрограммы
Subprogram Call Statement	Оператор вызова подпрограммы VHDL

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Subsystem	Подсистема
Subtractor	Вычитатель
Subtype	Подтип
Switch	Коммутатор
Symbolic ASM Table	Символьная таблица цифрового автомата в форме ASM
Symbolic FSM Table	Символьная таблица цифрового автомата в форме FSM
Synchronous State Machine	Синхронный цифровой автомат
Syntax	Синтаксис
Synthesis	Синтез
Synthesis Software	Программное обеспечение синтеза
Synthesis Tool	Инструмент синтеза
System State	Состояние системы
Target Technology	Выбранная технология
Technology Library	Технологическая библиотека
Technology Mapping	Наложение технологии
Technology-Dependent Netlist	Технологически-зависимый список соединений
Technology-Independent Design	Технологически-независимое проектирование
Technology-Specific Factor	Технологический фактор
Temporal Logic	Временная логика
Test Bench	Испытательная программа
Test Coverage	Покрытие теста
Test File	Тестовый файл
Test Input	Тестовый вход
Test Pattern	Тестовая комбинация
Test Vector	Тест-вектор
Testable Design	Тестопригодный проект
Testing	Тестирование
Time Base	База данных временного моделирования
Time Constraint	Временное ограничение
Time Granularity	Выбор шага моделирования по времени
Time Relation	Временное соотношение
Time Type	Тип данных Time языка VHDL
Timing	Расчет временных соотношений
Timing Analyser	Временной анализатор
Timing Description	Описание поведения во времени

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
Timing Simulation	Временное моделирование
Timing Verification	Временная верификация
Top Design	Верхний уровень проекта
Top Design Level	Верхний уровень проекта
Top Down Design	Нисходящее проектирование
Top Level of a Design	Верхний уровень проекта
Top Level of a Model	Верхний уровень модели
Traditional Schematic Design	Традиционное схемотехническое проектирование
Transaction	Транзакция
Transformation Unit	Устройство преобразования
Transistor Level	Транзисторный уровень
Transport Delay	Транспортная задержка
Truth Table	Таблица истинности
Type	Тип
Typical Case	Типичный случай
Unconstrained Array Type	Тип данных Array языка VHDL (без границ)
Undefined (X)	Неопределенное состояние
Use Clause	Указатель использования библиотек и пакетов
User-Defined Attribute	VHDL-атрибут, определенный пользователем
Valid Address	Действительный адрес
Validation	Проверка правильности проекта
Variable	Переменная
Variable Assignment Statement	Оператор присваивания значений переменным VHDL
Verification	Верификация
Verifying Functionality	Верифицирование функционирования
VHDL Analog Extension	Расширение VHDL для описания аналоговых схем
VHDL Capture	Ввод описания проекта на языке VHDL
VHDL Code	Текст на языке VHDL
VHDL Compiler	VHDL-компилятор
VHDL Component	VHDL-компонент
VHDL Description	Описание на VHDL
VHDL Design	Проектирование на VHDL
VHDL Modeling	Построение моделей на VHDL
VHDL Process	VHDL-процесс
VHDL Simulation	Моделирование на VHDL

АНГЛИЙСКИЙ ТЕРМИН	ПЕРЕВОД
VHDL Simulator	Симулятор VHDL
VHDL Synthesis	Синтез схем по описанию на VHDL
VHDL Synthesis Tool	Инструмент VHDL-синтеза
VHDL Types Classification	Классификация типов данных
VLSI (Very Large-Scale Integrated) Circuit	СБИС
VMA (Valid Memory Address)	Действительный адрес в памяти
Wait Statement	Оператор Wait языка VHDL
Waveform Generation	Генерация временных диаграмм
Wire	Соединение проводником
With-Select Statement	Оператор With-Select языка VHDL
Worst Case	Наихудший случай

Русско-английский словарь — указатель терминов

РУССКИЙ ТЕРМИН	ПЕРЕВОД
VHDL-атрибут, определенный пользователем	User-Defined Attribute
VHDL-компилятор	VHDL Compiler
VHDL-компонент	VHDL Component
VHDL-описание проекта	Project VHDL Description
VHDL-процесс	VHDL Process
Абстрактная структура	Abstract Structure
Абстракция языка	Language Abstraction
Автомат абстрактный цифровой	Abstract State Machine
Автомат асинхронный цифровой	Asynchronous State Machine
Автомат конечный цифровой	FSM (Finite State Machine)
Автомат конечный цифровой Мили	Mealy FSM
Автомат конечный цифровой Мура	Moore FSM
Автоматический синтез схем	Automatic Synthesis
Автоматическое оборудование для тестирования	(ATE (Automate Test Equipment
Адекватное поведение во времени	Actual Time Behavior
Алгоритм	Algorithm
Анализ	Analysis
Аппаратная операционная система	Hardware Operating System
Аппаратный ускоритель	Hardware Accelerator
Архитектура модели на VHDL	Architecture
Архитектурное тело модели на VHDL	Architectural Body
Атрибут языка VHDL	Attribute
База данных временного моделирования	Time Base
Библиотека компонентов	Component Library
Бинарное умножение	Binary Multiplication
Битовая строка	Bit String
Блок принятия решения ASM	Decision Box
Блок условного выхода ASM	Conditional Output Box
Блок-схема цифрового автомата в форме ASM	ASM Chart

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Блочная диаграмма	Block Diagram
Булево уравнение	Boolean Equation
Быстрое макетирование	Rapid Prototyping
Ввод описания проекта на языке VHDL	VHDL Capture
Вентиль	Gate
Вентильный уровень	Gate Level
Верификация	Verification
Верификация временная	Timing Verification
Верификация проекта	Design Verification
Верифицирование функционирования	Verifying Functionality
Верхний уровень модели	Top Level of a Model
Верхний уровень проекта	Top Design
Верхний уровень проекта	Top Design Level
Верхний уровень проекта	Top Level of a Design
Внешние воздействия (стимулы)	Outside Stimulus
Внутренние узел/межсоединение	Internal Node/Interconnection
Внутренний сигнал архитектуры	Internal Signal
Волновой сумматор с последовательным формированием переноса	Ripple-Carry Adder
Временная логика	Temporal Logic
Временное моделирование	Timing Simulation
Временное моделирование после формирования топологии микросхемы	Post Layout Timing Simulation
Временное ограничение	Time Constraint
Временное соотношение	Time Relation
Временной анализатор	Timing Analyser
Время доступа	Access Time
Время жизни изделия	Life Time
Время моделирования	Simulation Time
Время окончания	Finish Time
Время разработки	Development Time
Время старта	Start Time
Входной порт	Input Port
Входные данные	Input Data
Входы	Inputs

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Выбор компонента	Component Choise
Выбор шага моделирования по времени	Time Granularity
Выбранная технология	Target Technology
Вызов подпрограммы	Subprogram Call
Выполнение по условию	Conditional Execution
Высокий уровень абстракции	High Abstraction Level
Выходной порт	Output Port
Выходные данные	Output Data
Выходы	Outputs
Вычитатель	Subtractor
Генератор сигналов	Signal Generator
Генерация временных диаграмм	Waveform Generation
Гибридная модель	Hybrid Model
Графический ввод проекта	Graphical Design Entry
Графический интерфейс пользователя	Graphics User Interface
Двунаправленный сигнал	Bidirectional Signal
Действительный адрес	Valid Address
Действительный адрес в памяти	VMA (Valid Memory Address
Декларативная часть	Declarative Part
Декларативная часть архитектуры	Architecture Declarative Part
Декларация внутреннего соединения	Internal Wire Declaration
Декларация интерфейса модели	Entity (Port Declaration
Декларация конфигурации	Configuration Declaration
Декларация портов схемы	Port Declaration
Декларация сигнала	Signal Declaration
Дифференциальное уравнение	Differential Equation
Жизненный цикл изделия	Life Cycle
Задержка	Delay
Задержка вентиля	Gate Delay
Задержка загрузки	Load Delay
Задержка инерциальная	Inertial Delay
Задержка межсоединения	Interconnect Delay
Зарезервированное слово языка	Reserved Word
Идентификатор	Identifier
Иерархическая составная система	Hierarchically Composed System

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Иерархия	Hierarchies
Иерархия в построении моделей	Building Hierarchies
Иерархия проекта	Design Hierarchy
Избыточные единицы аппаратуры (функции)	Redundant Hardware Units (Functions)
Извлечение информации из памяти	Fetching
Институт инженеров по электронике и электротехнике	IEEE (Institute of Electronic and Electrical Engineers)
Инструмент VHDL-синтеза	VHDL Synthesis Tool
Инструмент логического синтеза	Logic Synthesis Tool
Инструмент поведенческого синтеза	Behavioral Synthesis Tool
Инструмент проектирования	Design Tool
Инструмент синтеза	Synthesis Tool
Интегральная схема, изготовленная для конкретного заказчика	(Application Specific Integrated ASIC Circuit)
Интегральная схема, изготовленная по специальному заказу	Custom Integrated Circuit
Интегрированная модель прототипа	Integrated Prototype Model
Исполнение	Execution
Исполняемая часть	Executive Part
Исполняемая часть архитектуры	Architecture Statement Part
Исправный компонент	Good Component
Испытательная программа	Test Bench
Классификация типов данных	VHDL Types Classification
Кодированная таблица цифрового автомата в форме ASM	Assigned ASM Table
Комбинационная логика	Combinational Logic
Комментарий	Comment
Коммутатор	Switch
Компаратор	Comparator
Компонент цифровой схемы	Hardware Component
Компоновка	Fitting
Компоновка системы	Arrangement of a System
Компоновщик	Fitter
Компьютерная модель	Computer Model
Компьютерное моделирование	Computer Simulation
Компьютерный инструмент проектирования	Design Computer Tool
Конкатенация	Concatenation

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Константа	Constant
Конструкция VHDL, позволяющая использовать параметры при описании схем	Generics
Конфигурация	Configuration
Корень дерева проекта	Root Design
Лексический компонент	Lexical Element
Лист компонента в дереве проекта	Leaf Component
Литерал-вещественное число VHDL	Real Literal
Литерал-строка	String Literal
Литерал-целое число языка VHDL	Integer Literal
Логический (вентильный) уровень	Logic Level
Логический анализатор	Logic Analyzer
Логический вывод	Inference
Логический синтез	Logic Synthesis
Максимальная надежность проектирования	Maximum Design Reliability
Мастер языковых конструкций	Language Assistant
Математический оператор	Mathematical Statement
Машинный код	Machine Code
Межсоединение	Interconnection
Метод проектирования	Design Method
Метод разработки	Development Method
Микрокод	Microcode
Микропроцессор	Microprocessor
Микросхема, изготовленная по специальному заказу	Special User Circuit
Минимальная цена	Minimum Cost
Минимальное время проектирования	Minimum Design Time
Многократно используемый компонент	Reusable Component
Множитель	Multiplicand
Моделирование	Simulation
Моделирование на VHDL	VHDL Simulation
Мультиплексор	Multiplexer
Наивысший поведенческий уровень	Highest Behavioral Level
Наилучший случай	Best Case
Наихудший случай	Worst Case
Наложение технологии	Technology Mapping

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Наследование	Inheritance
Неопределенное состояние	Undefined (X)
Нисходящее проектирование	Top Down Design
Область геометрических моделей	Geometric Domain
Область моделирования	Modeling Domain
Область структурных моделей	Structural Domain
Область функциональных моделей	Functional Domain
Общий компонент	Shared Component
Обьектно-базирующийся язык	Object-Based Language
Обьектно-ориентированный язык	Object-Oriented Language
Обьемлющая модель	Enclosing Model
Ограничение	Constraint
Оперативная память	RAM (Random Access Table)
Оператор If-Then-Else языка VHDL	If-Then-Else Statement
Оператор Wait языка VHDL	Wait Statement
Оператор With-Select языка VHDL	With-Select Statement
Оператор блока языка VHDL	Block Statement
Оператор выборочного присвоения значений сигналам VHDL	Selected Signal Assignment Statement
Оператор вызова подпрограммы VHDL	Subprogram Call Statement
Оператор генерации языка VHDL	Generate Statement
Оператор получения экземпляров компонент	Instantiation Component Statement
Оператор присваивания значений	Assignment Statement
Оператор присваивания значений переменным VHDL	Variable Assignment Statement
Оператор присваивания значений сигналам VHDL	Signal Assignment Statement
Оператор процесса VHDL	Process Statement
Оператор цикла языка VHDL	Loop Statement
Операторы «поведенческого» VHDL	Behavioral VHDL Statements
Операторы структурного VHDL	Structural VHDL Statements
Операция	Operation
Операция арифметического сдвига влево	Shift Left Arithmetic Operator
Операция арифметического сдвига вправо	Shift Right Arithmetic Operator
Операция логического сдвига влево	Shift Left Logical Operator
Операция логического сдвига вправо	Shift Right Logical Operator
Операция циклического логического сдвига влево	Rotate Left Logical Operator

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Операция циклического логического сдвига вправо	Rotate Right Logical Operator
Описание аппаратуры (цифровой схемы)	Hardware Description
Описание комбинационной логики	Combinational Logic Description
Описание на VHDL	VHDL Description
Описание на вентильном уровне	Gate Description
Описание поведения во времени	Timing Description
Описание последовательной логики	Sequential Logic Description
Осциллограф	Oscilloscope
Охранное выражение	Guard Expression
Охраняемый блок	Guarded Block
Оценка выражения	Evaluating Expression
Пакет	Package
Память	Memory
Параллельная конструкция языка	Concurrent Language Construction
Параллельность функционирования	Concurrency
Параллельные операторы для сигналов	Parallel Statements for Signals
Параллельный оператор	Concurrent (Parallel) Statement
Параллельный сумматор-вычитатель	Parallel Adder/Subtractor
Первичный вход схемы	Primary Input
Первичный выход схемы	Primary Output
Первоначальное значение	Initial Value
Переменная	Variable
Перечислимый тип	Enumerated Type
Период синхросигнала	Clock Cycle
Печатная плата	(PCB (Printed Circuit Board
План расположения на слое микросхемы	Floor Plan
Планирование транзакций	Scheduling a Transaction
ПЛИС	(PLD (Programmable Logic Device
Площадь	Area
Поведенческая архитектура	Behavioral Architecture
Поведенческая модель	Behavioral Model
Поведенческий уровень моделирования	Behavioral Level
Поведенческое описание	Behavioral Description
Повторное исполнение	Repeated Execution
Повторяющаяся комбинация	Repetitive Pattern

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Подкомпонент	Subcomponent
Подпрограмма	Subprogram
Подсистема	Subsystem
Подтип	Subtype
Покрытие теста	Test Coverage
Полигон	Polygon
Полный сумматор	Full Adder
Полупроводниковая пластина	Silicon Die
Порт схемы	Port
Последовательная конструкция языка	Sequential Language Construction
Последовательная логика	Sequential Logic
Последовательный оператор	Sequential Statement
Последовательный сумматор	Serial Adder
Построение моделей на VHDL	VHDL Modeling
Поток процесса проектирования	Flow Design
Предопределенный атрибут VHDL	Predefined Attribute
Представление	Parallel Process Representation
Представление задержки	Delay Representation
Примитивный элемент	Primitive Element
Принцип «черного ящика»	Black Box Principle
Присвоение значений переменным	Assigning Values to Variables
Проверка правильности проекта	Validation
Проверка удержания сигнала	Hold Test
Проверка установки сигнала	Setup Test
Проверяемый объект (устройство)	(DUT (Device Under Test
Программируемая микросхема	Programmable Circuit
Программируемый массив вентилей	(FPGA(Field Programmable Gate Array
Программное обеспечение синтеза	Synthesis Software
Программный счетчик	(PC (Program Counter
Проектирование восходящее	Bottom Up Design
Проектирование иерархическое	Hierarchy Design
Проектирование на VHDL	VHDL Design
Проектирование печатных плат	PCB Design
Прототип	Prototype
Процедура	Procedure

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Процесс автоматической генерации тестов	ATPG (Automatic Test Pattern Generation
Процесс проектирования, проект	Design
Процессор	Processor
Разбиение	Partioning
Разветвление по выходу	Fan-Out
Различные уровни абстракции	Different Abstraction Levels
Размещение и трассировка	Place&Route
Расчет временных соотношений	Timing
Расширение VHDL для описания аналоговых схем	VHDL Analog Extension
Реализация проекта	Design Implementation
Регистр	Register
Регистр адреса	Address Register
Регистр адреса при обращении к памяти	(MAR (Memory Address Register
Регистр аккумулятора	Accumulator Register
Регистр второго операнда для АЛУ	Optional Register
Регистр команд	(IR (Instruction Register
Регистр результата (выхода)	Output Register
Регистр сдвига	Shift Register
Регистр хранения данных	Data Storage Register
Редактор для графического проектирования цифровых автоматов	State Editor
Результаты моделирования	Outputs from Simulation
САПР	CAD (Computer Aided Design) System
САПР Active-HDL	CAD Active-HDL
САПР Foundation Series	CAD Foundation Series
САПР ModelSim	CAD ModelSim
САПР OrCAD	CAD OrCAD
САПР VHDL Simili (VHDL Sonata)	CAD VHDL Simili (VHDL Sonata)
САПР Warp	CAD Warp
СБИС	(VLSI (Very Large-Scale Integrated Circuit
Сдвигатель, регистр сдвига	Shifter
Семантика	Semantics
Сигнал	Signal
Сигнал интерфейса схемы	Interface Signal
Сигнал чтения-записи	R/W (Read/Write)

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Сигналы входа-выхода	I/O Signals
Символ	Character
Символ операции	Operator
Символьная таблица цифрового автомата в форме ASM	Symbolic ASM Table
Символьная таблица цифрового автомата в форме FSM	Symbolic FSM Table
Символьный литерал	Character Literal
Симулятор VHDL	VHDL Simulator
Симулятор смешанного уровня	Mix Level Simulator
Симулятор, система моделирования	Simulator
Синтаксис	Syntax
Синтез	Synthesis
Синтез аппаратуры	Hardware Synthesis
Синтез на поведенческом уровне	Behavioral Synthesis
Синтез на уровне межрегистровых передач	RTL Synthesis
Синтез схем по описанию на VHDL	VHDL Synthesis
Синхронный цифровой автомат	Synchronous State Machine
Система моделирования	Simulation System
Скалярный тип данных языка VHDL	Scalar Type
Сложная ПЛИС	CPLD (Complex Programmable Logic Device)
Сложность проекта	Design Complexity
Смешанное архитектурное тело	Mixed Architectural Body
Смешанное проектирование	Mixed Design
Событие	Event
Соединение проводником	Wire
Создание прототипа	Prototyping
Создание экземпляров компонентов	Component Instantiation
Сокращение сложности проекта	Reducing Complexity
Сообщение об ошибке	Error Message
Составная (сложная) система	Composite System
Составной тип данных языка VHDL	Composite Type
Состояние высокого импеданса	High Impedance (Z)
Состояние системы	System State
Специальный символ	Special Symbol
Спецификация	Specification

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Спецификация проекта	Design Specification
Спецификация проекта	Project Specification
Список соединений	Netlist
Список соединений компьютерной модели	Computer Model Netlist
Список чувствительности проекта	Sensitivity List
Способность к модификации	Modifiability
Среда схемотехнического проектирования	Schematics
Среда цифрового проектирования	Digital Development Environment
Стандартная библиотечная ячейка	Standard Library Cell
Стандартная ячейка	Standard Cell
Стандартный метод	Standartized Method
Стиль потока данных	Data Flow Style
Строка	String
Структурная архитектура	Structural Architecture
Структурная единица проекта	Design Unit
Структурная единица проекта	Hardware Units
Структурная модель	Structural Model
Структурное архитектурное тело	Structural Architecture Body
Структурное описание	Structural Description
Структурный VHDL	Structural VHDL
Сумматор	Adder
Сумматор с ускоренным формированием переноса	Carry Look Ahead Adder
Схема	Scheme
Схемотехнический компонент	Schematic Component
Таблица истинности	Truth Table
Текст на структурном VHDL	Structural VHDL Code
Текст на языке VHDL	VHDL Code
Текстовый редактор для языка VHDL	HDL Text Editor
Текущий этап («фокус») процесса проектирования	Current Focus of Design
Тест на броски питания	Spike Test
Тест-вектор	Test Vector
Тестирование	Testing
Тестовая комбинация	Test Pattern
Тестовый вход	Test Input
Тестовый файл	Test File

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Тестопригодный проект	Testable Design
Техника сканирования путей в цифровых схемах	Scan Path Techniques
Технологическая библиотека	Technology Library
Технологически-зависимый список соединений	Technology-Dependent Netlist
Технологический фактор	Technology-Specific Factor
Технологически-независимое проектирование	Technology-Independent Design
Тип	Type
Тип данных	Data Type
Тип данных Access языка VHDL	Access Type
Тип данных Array языка VHDL	Array Type
Тип данных Array языка VHDL (с границами)	Constrained Array Type
Тип данных Array языка VHDL (без границ)	Unconstrained Array Type
Тип данных Assert языка VHDL	Assert Statement
Тип данных Bit языка VHDL	Bit Type
Тип данных Bit-Vector языка VHDL	Bit-Vector Type
Тип данных Boolean языка VHDL	Boolean Type
Тип данных Case-When языка VHDL	Case-When Statement
Тип данных Character языка VHDL	Character Type
Тип данных Discrete языка VHDL	Discrete Type
Тип данных File языка VHDL	File Type
Тип данных Integer языка VHDL	Integer Type
Тип данных Physical языка VHDL	Physical Type
Тип данных Real языка VHDL	Real Type
Тип данных Record языка VHDL	Record Type
Тип данных Severity языка VHDL	Severity-Level Type
Тип данных String языка VHDL	String Type
Тип данных Time языка VHDL	Time Type
Тип данных с плавающей точкой языка VHDL	Floating-point Type
Типичный случай	Typical Case
Типовой компонент	Generic Component
Тире	Dash
Точность модели	Model Accuracy
Традиционное схемотехническое проектирование	Traditional Schematic Design
Тракт передачи данных	Data Path
Транзакция	Transaction

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Транзисторный уровень	Transistor Level
Транспортная задержка	Transport Delay
Требование спецификации проекта	Requirement Design Specification
Триггер	Flip-Flop (FF)
Указатель использования библиотек и пакетов	Use Clause
Умножитель в виде массива ячеек	Cellular Multiplier
Умножитель, Множимое	Multiplier
Умножитель, функционирующий посредством выполнения сдвигов и сложений	Shift and Add Multiplier
Уровень «процессор-память-коммутатор»	(PMS (Processor-Memory-Switch Level
Уровень абстракции	Abstraction Level
Уровень межрегистровых передач	(RTL (Register Transfer Level
Уровень формирования топологии 12микросхемы	Layout Level
Условный оператор присвоения значений сигналам VHDL	Conditional Signal Assignment Statement
Устройство	Device
Устройство ввода-вывода	I/O Device
Устройство преобразования	Transformation Unit
Устройство преобразования данных	Data Transformation Unit
Устройство управления	Control Unit
Уточнение (предвыполнение) модели	Elaboration
Файл выходных сигналов	Out Signal File
Физический прототип	Physical Prototype
Функциональная верификация	Functional Verification
Функциональное моделирование	Functional Simulation
Функциональный модуль	Functional Module
Функциональный уровень	Functional Level
Функционирующий прототип	Functioning Prototype
Функция	Function
Функция генерации	Generate Function
Функция распространения	Propagate Function
Хранение и преобразование данных	Data Storage and Transformation
Центральное процессорное устройство	(CPU (Central Processing Unit
Цикл моделирования	Simulation Cycle
Цифровое событийное моделирование	Discrete Event Simulation
Цифровой автомат	State Machine

РУССКИЙ ТЕРМИН	ПЕРЕВОД
Цифровой автомат в форме ASM	(Algorithmic State Machine) ASM
Частичное произведение	Partial Product
Черный ящик	Black Box
Число	Number
Чисто поведенческая модель	Purely Behavioral Model
Чисто поведенческий стиль проектирования	Pure Behavioral Style
Чисто структурная модель	Purely Structural Model
Экземпляр компонента	Component Instance
Этап инициализации	Initialization Phase
Этап разработки	Development Phase
Язык Ассемблер	Assembler
Язык межрегистровых передач	Register-Transfer Language
Язык моделирования	Modeling Language
Язык описания аппаратуры	Hardware Description Language (HDL)
Язык описания аппаратуры ABEL	ABEL HDL
Язык описания аппаратуры AHDL	AHDL (Altera HDL)
Язык описания аппаратуры CUPL	CUPL HDL
Язык описания аппаратуры PALASM	PALASM HDL
Язык описания аппаратуры Verilog HDL	Verilog HDL
Язык описания аппаратуры VHDL	VHDL (VHSIC HDL)
Язык описания аппаратуры высокого уровня	High-Level Hardware Design Language
Язык описания аппаратуры низкого уровня	Low-Level Hardware Design Language
Язык проектирования ПЛИС	PLD Design Language
Ячейка умножителя	Multiplier Cell

Содержание

Магазины Москвы и городов Московской области

Торговый Дом «Библио-Глобус»
Адрес: ул. Мясницкая, д. 6.
Тел.: 928-35-67.

Магазин «Московский Дом Книги»
Адрес: Новый Арбат, д. 8.
Тел.: 203-82-42, 291-78-32.

Магазин «Дом технической книги»
Адрес: Ленинский пр-т, д. 40.
Тел.: 137-60-38, 137-60-39.

Магазин «Молодая Гвардия»
Адрес: ул. Б. Полянка, д. 28.
Тел.: 238-26-86, 238-50-01.

Магазин «Дом книги на Соколе»
Адрес: Ленинградский пр-т, д. 78, к. 1.
Тел.: 152-82-82, 152-45-11.

Магазин «Дом книги на Войковской»
Адрес: Ленинградское шоссе, д. 13, стр. 1.
Тел.: 150-99-92, 150-69-17.

Торговый дом книги «Москва»
Адрес: ул. Тверская, д. 8, стр. 1.
Тел.: 797-87-16, 229-73-55.

Магазин «Дом книги на Новой»
Адрес: Шоссе Энтузиастов, д. 24/43.
Тел.: 361-68-34, 362-25-16.

Магазин «Дом книги в Медведково»
Адрес: Заревый проезд, д. 12.
Тел.: 478-48-97.

Магазин «Дом книги в Бибирево»
Адрес: ул. Мурановская, д. 12.
Тел.: 407-95-55, 406-47-77.

Магазин «Мир печати»
Адрес: ул. 2-я Тверская-Ямская, д. 54.
Тел.: 978-50-47.

Магазин «Дом книги на Преображенке»
Адрес: Преображенский вал, д. 16, стр. 1.
Тел.: 964-42-26.

Магазин «Дом книги в Сокольниках»
Адрес: ул. Русаковская, д. 27.
Тел.: 264-81-21.

Магазин «Мир школьника»
Адрес: ул. Шоссейная, 1.
Тел.: 179-57-17.

Магазин «Дом книги в Измайлово»
Адрес: Измайловская пл., д. 2, к. 1.
Тел.: 166-69-96.

Магазин «Дом Книги у Красных Ворот»
Адрес: ул. Садовая-Черногрязская, д. 5/9.
Тел.: 975-47-49.

Магазин «Фолиант»
Адрес: Шоссе Энтузиастов, д. 60А.
Тел.: 276-06-53.

Магазин «Радиотехника»
Адрес: ул. Новокузнецкая, д. 17/19.
Тел.: 953-69-53.

Internet-магазин «ДЕССИ»
Адрес: 107113, г. Москва, а/я 10.
Тел.: 304-72-31.

ООО «Дом деловой книги»
Адрес: ул. Марксистская, д. 9.
Тел.: 270-54-20, 270-54-21.

ООО «КУМ»
Адрес: ул. Ярцевская, д. 34, к. 1.
Тел.: 140-44-25.

Магазин «Алекс и Ко»
Адрес: г. Зеленоград, Панфиловский пр-т, к. 1106-Б. Тел.: 532-96-69.

ПБОЮЛ Тимохина И.Ю.
Адрес: г. Зеленоград, корп. 1420, кв. 60.
Тел.: 533-44-44, 537-60-59.

Магазин «Дом книги»
Адрес: г. Подольск, пр-т Ленина, д. 158.
Тел.: (276) 300-76.

Магазин «Книга»
Адрес: г. Балашиха, ш. Энтузиастов, д. 11.
Тел.: 529-56-25.

ООО «Элара»
Адрес: г. Сергиев-Посад.
Тел.: (254) 212-52, 455-05.

Магазины городов России и ближнего зарубежья

Магазин «Санкт-Петербургский Дом Книги»
Адрес: г. Санкт-Петербург, Невский проспект, д. 28.
Тел.: 318-49-15, 312-01-84.

Магазин «Техническая книга»
Адрес: г. Санкт-Петербург, Пушкинская пл., д. 2.
Тел.: 164-65-65, 164-62-77.

ООО «Elkom»
Адрес: г. Астрахань, ул. Красная, д. 12/2.
Тел.: 39-08-53.

ООО «Книжный меридиан»
Адрес: г. Красноярск, ул. Дубровинского, д. 52А.
Тел.: 27-14-29.

ЧП Ващенко С.В.
Адрес: г. Липецк, рынок 9-го микрорайона, контейнер 37; пр-т Победы, 29, «Дом быта», 2-й этаж, «Бизнес-книга».
Тел.: 77-04-25, 46-33-34.

ООО «Книжный мир»
Адрес: г. Нальчик, ул. Захарова, 103. Тел.: 5-52-01.

ООО «Топ книга»
Адрес: г. Новосибирск. Тел.: 36-10-26, 36-10-27.

Магазин «На Бульваре»
Адрес: г. Орел, ул. Московская, д. 36. Тел.: 43-54-69.

Магазин «Ампир»
Адрес: г. Орел, Бульвар Победы, д. 1. Тел.: 5-89-90.

Комаров Виктор Анатольевич
Адрес: Пермская обл., г. Лысьва, ул. Озерная, д. 11, кв. 89. Тел.: 44-56-41.

Магазин «Чакона»
Адрес: г. Самара, ул. Чкалова, 100.
Тел.: 42-96-28, 42-96-29.

Магазин радиодеталей «Элком»
Адрес: г. Самара, пр-т Кирова, 229; ул. Ивана Булкина, 81. Тел.: 59-85-14; 24-25-04.

Магазин «Электроника»
Адрес: г. Уфа, пр-т Октября, д. 108. Тел.: 33-10-29.

ЧП Вологин В.
Адрес: г. Запорожье.
Тел.: (10380612) 96-63-21.
e-mail: book@book.marka.net.ua.

ООО «АКМА»
Адрес: г. Пермь, ул. Г. Хасана, д. 16.
Тел.: 90-45-60, 90-93-02.

ООО «ЛИТЭКС»
Адрес: г. Красноярск, ул. Дудинская, д. 3А.
Тел.: 55-5035, 55-50-36.

ООО «Элко-М»
Адрес: г. Чебоксары. Тел.: 64-99-22, 64-99-33.

Свердловская областная организация Добровольного общества любителей книги РФ
Адрес: г. Екатеринбург, ул. Антона Валека, д. 8А.
Тел.: 71-72-72.

Магазин радиодеталей «ДЕЛЬТА»
Адрес: г. Новокузнецк, ул. Воровского, д. 13.
Тел.: 74-59-49.

ЧП Кудь А.Я.
Адрес: г. Харьков. Тел.: 776-43-29.

ГУП Красноярский магазин «Академкнига» торговой фирмы «Академкнига» РАН
Адрес: г. Красноярск, ул. Сурикова, д. 45.
Тел.: 27-03-90.

ЧП Гергерт Е.В.
Адрес: г. Тюмень, ул. Республики, д. 143.
Тел.: 22-81-95.

ЧП Антропов В.П.
Адрес: г. Омск, ул. 10 лет Октября, д. 48.
Тел.: (3812) 31-05-83.

Магазин ОАО «Курскккнига»
Адрес: г. Курск, ул. Ленина, д. 11.
Тел.: (0712) 22-38-42, 22-77-23.

ИП Ольхова О.Г.
Адрес: г. Н. Новгород, ул. Бекетова, д. 71, кв. 39.
Тел.: (8312) 33-02-91.

ГУП «Чувашский республиканский бибколлектор» Госкомпечати Чувашской Республики
Адрес: г. Чебоксары, ул. Петрова, д. 7.
Тел.: (8335) 62-15-67.

ООО «ТД Чародей»
Адрес: г. Томск, ул. Белинского, д. 55/НТБ ТПУ.
Тел.: (3822) 41-57-72.

МУП «Книжный магазин № 14»
Адрес: г. Екатеринбург, ул. Челюскинцев, д. 23.
Тел.: (3432) 53-24-89.

«Дом книги»
Адрес: г. Екатеринбург, ул. Валека, д. 12.
Тел.: (3432) 58-12-00.

ОАО «Уралкнига»
Адрес: г. Екатеринбург, ул. Аршинская, д. 23а.
Тел.: (3432) 71-25-24.

ГУП «Облбибколлектор»
Адрес: г. Екатеринбург, ул. Первомайская, д. 70.
Тел.: (3432) 74-27-47.

ООО «Амиталь»
Адрес: г. Воронеж, Ленинский пр-т, д. 157.
Тел.: (0732)23-00-02, 23-63-26.

ООО «Азбука»
Адрес: г. Петрозаводск, ул. Дзержинского, д. 4.
Тел.: (8142)78-55-03.

«Тезей»
Адрес: г. Мурманск. Тел.: (8152) 43-76-96

маг. Микроника
Адрес: Киев, ул. Марины Расковой, д. 13.
Тел.: (044) 517-73-77

ООО «Импульс»
Адрес: г. Тольятти, ул. Дзержинского, д.70.
Тел.: 32-74-85

ООО «ДОМО «ГЛОБУС»
Адрес: г. Казань, пр-д Коммуны, д. 10/72.
Тел.: 36-74-81; 99-50-74

ООО «ТАИС»
Адрес: г. Казань, ул. Гвардейская, д. 9, корп. А.
Тел.: (8432) 72-34-55

Магазин «ЭХО»
Адрес: г. Петрозаводск, ул. Андропова, д. 9.
Тел.: (8142) 78-36-01

Новые книги издательства «СОЛОН-Пресс»

В. П. Дьяконов
Энциклопедия Mathcad 2001i и Mathcad 11 (с CD-ROM).
Серия «Библиотека профессионала».

Первая энциклопедия по применению новейших и наиболее популярных систем компьютерной математики Mathcad 2001i и Mathcad 11. Описаны обширные применения этих систем не только в массовых математических, физических и научно-технических расчетах, но и в области математического моделирования, оптимизации, машинной графики, в обработке сигналов и изображений (в том числе с применением вейвлетов), в проектировании электронных устройств и др. Впервые дано описание ряда электронных книг для системы Mathcad и ее работа с компьютеризованной измерительной лабораторией PC-Lab 2000. Более 2000 примеров и иллюстраций!

Для научных работников, инженеров, преподавателей и студентов вузов и университетов, подготовленных радиолюбителей и всех пользователей системами компьютерной математики.

С. С. Байдачный
.NET Framework: разработка Windows приложений.
Серия «Библиотека профессионала».

.NET Framework — технология, разработанная компанией Microsoft для создания аппаратно и платформенно независимых программных продуктов. Кроме того, она не имеет привязки к какому-либо языку программирования высокого уровня. Так, вы можете использовать такие языки как C++, Visual Basic, C#, J# и несколько десятков других языков при разработке приложений. При этом вы можете использовать как один язык, так и одновременно несколько языков программирования. За два года своего существования .NET Framework заняла лидирующее место на рынке. Так, большинство компаний все свои новые программные продукты разрабатывают с использованием .NET Framework.

В любом книжном магазине можно увидеть большой спектр книг по .NET Framework и языкам программирования для разработки приложений на платформе .NET. Почему же следует купить именно эту книгу?

Во-первых, тут содержится все необходимое, чтобы подготовиться к разработке реальных Windows приложений. Нет необходимости покупать дополнительную литературу. Тут есть все.

Во-вторых, все примеры в книге написаны на языке программирования C#, который был разработан таким образом, что он наиболее подходит для программирования на платформе .NET. Кроме того, первая часть книги рассказывает именно о C#, параллельно погружая читателя в особенности .NET Framework. Тут описана не только первая версия этого языка, но и возможности, который будут доступны в C# 2.0. Спецификация этой версии уже была анонсирована Microsoft в составе .NET Framework 2.0.

Кроме этого, автор этой книги имеет большой опыт разработки .NET приложений, являясь при этом сертифицированным разработчиком и инструктором Microsoft (MCSD, MCT). Последнее позволяет автору читать сертифицированные курсы в учебном центре Microsoft по .NET технологиям и находиться в курсе всех возможностей платформы. Таким образом, если вы возьмете эту книгу за основу при подготовке к экзамену 70-316 — Разработка Windows приложений с использованием Visual Studio .NET, сдача которого необходима при сертификации на статус MCSD, то наверняка сдадите его без проблем.

Таким образом, эта книга будет полезна как начинающим программистам, которые смогут в совершенстве овладеть C# и возможностями .NET Framework, так и профессионалам, которые хотят систематизировать свои знания.

.NET Framework — технология будущего! И если вы не хотите остаться в прошлом, то эта книга для вас!

М.М.Соловьев

3DS MAX 6: Удивительный мир трехмерной графики (с CD-ROM).
Серия «Библиотека профессионала».

В последнее время заметно возрос интерес к созданию сложных графических построений при помощи персонального компьютера. Сегодня одним из наиболее мощных инструментов для таких работ является популярный пакет 3DS Max 6.

Вниманию читателя предлагается не просто очередная книга, посвященная работе с трехмерной графикой. Пособие является своеобразной всеобъемлющей энциклопедией, в которой помимо исчерпывающих текстовых описаний приведены 50 специально разработанных практических упражнений.

Дополнительная практическая ценность данного издания для пользователей определяется обширной подборкой материалов на прилагаемом компакт-диске.

Автор профессионально занимается данной тематикой, — он на протяжении ряда лет принимает участие в разработке роликов для телевизионной рекламы и создании моделей для трехмерных игр, преподает курсы по 3DS Max в учебном центре. Его перу принадлежат публикации «Трехмерный дизайн в программе 3D Studio Max версии 3 и 4» и «Трехмерный мир 3D Studio Max 5», выпущенные издательством «СОЛОН-Пресс» в 2002 и 2003 годах.

Книга рассчитана как на знатоков, профессионально занимающихся сложным моделированием, так и на новичков, желающих приобщиться к увлекательному миру трехмерной графики.