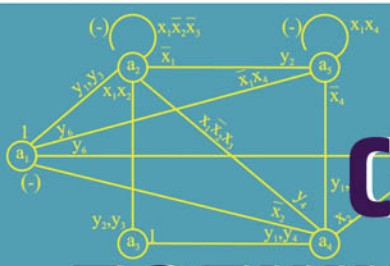
СЕРИЯ СИСТЕМЫ ПРОЕКТИРОВАНИЯ

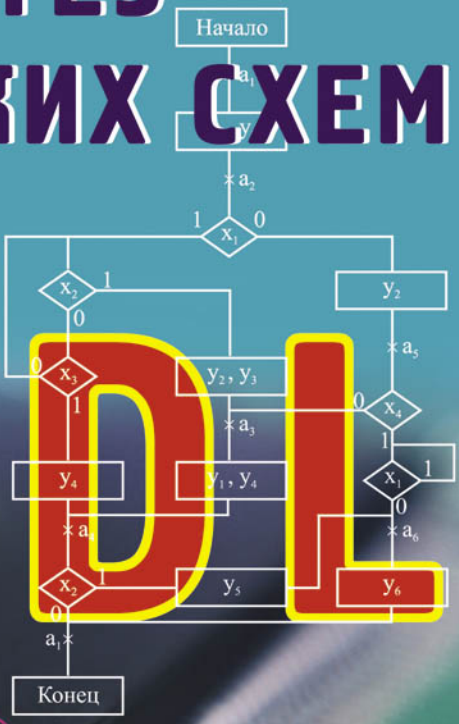


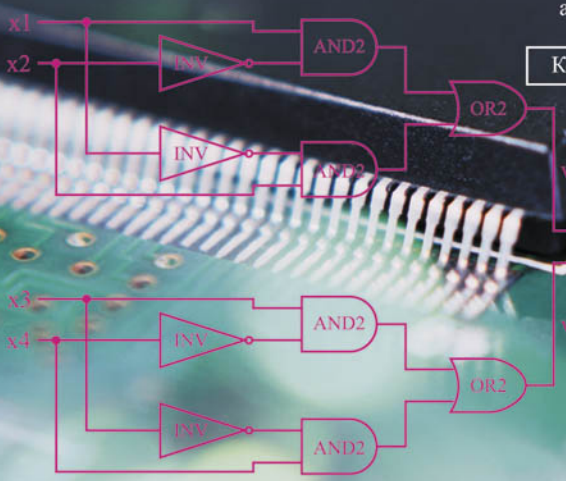
П.Н. БИБИЛО

СИНТЕЗ ЛОГИЧЕСКИХ СХЕМ

С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА

VHDL





«СОЛОН-Р»

П.Н. Бибило

Синтез логических схем с использованием языка VHDL —М.: СОЛОН-Р, 2009. —384 с.: ил.

ISBN 5-93455-152-3

Рассматриваются теоретические основы синтеза логических схем по спецификациям на высокоуровневом языке VHDL, предназначенном для проектирования цифровых систем, описывается "синтезируемое" подмножество языка VHDL в системе автоматического синтеза Leonardo, приводятся примеры синтезируемых (схемно реализуемых) конструкций языка и примеры синтезируемых описаний типовых схем, изучается проблема эффективного управления процессом синтеза. Язык VHDL является международным стандартом в системах автоматизации проектирования и предназначен для спецификации, моделирования и синтеза цифровых систем на основе заказных и программируемых пользователями сверхбольших интегральных схем.

Книга предназначена для первоначального ознакомления как с языком VHDL, так и с проблемой синтеза схем по спецификациям на языке VHDL. Может быть полезна студентам, аспирантам и специалистам, занимающимся разработкой электронной аппаратуры с помощью средств САПР.

103001, г. Москва, а/я 82
Телефоны:
(095) 254-44-10, 252-36-96, 252-25-21
E-mail: Solon-R@coba.ru

ISBN 5-93455-152-3

О г л а в л е н и е

Глава 3. Организация описания цифровой системы

Глава 1

Теоретические основы синтеза логических схем

1.1. VHDL – язык проектирования цифровых систем

Проектирование дискретных управляющих и вычислительных систем и устройств обычно разбивается на ряд крупных этапов. На начальном – *алгоритмическом* – этапе проектирования осуществляется моделирование исходного формального задания на проектирование системы, написанного на языке высокого уровня, верификация и переход от исходного описания к функциональному описанию. Формальное задание на проектирование часто называют *проектом*. На следующем этапе *логического* проектирования проект детализируется – решаются задачи синтеза логических схем в заданных технологических базисах, проводится моделирование схем на логическом (0,1) уровне, строятся тесты. Последующие этапы проектирования определяются элементным базисом: если устройство реализуется в виде отдельной БИС (большой интегральной схемы) или СБИС (сверхбольшой интегральной схемы), то решаются задачи *схемотехнического* и *топологического* проектирования кристаллов (рис.1.1), если же устройство реализуется на стандартных, уже выпускаемых промышленностью микросхемах, то дополнительно решаются задачи проектирования печатных плат и т. д. На каждом этапе проектирования проект (проектная информация) *детализируется*, например, результатом проектирования заказной СБИС является детальное описание топологии, пригодное для технологических установок производства кристаллов. Использование программируемых логических интегральных схем (ПЛИС), выполненных в виде СБИС, требует

решения специфических задач “трансляции” исходных алгоритмических описаний (описаний логических схем) в технологический формат файла программирования микросхемы либо формат файла описания конфигурации ПЛИС. Если цифровая система не может быть реализована на одной ПЛИС, то приходится решать задачу разбиения (декомпозиции) проекта на сеть взаимосвязанных ПЛИС.

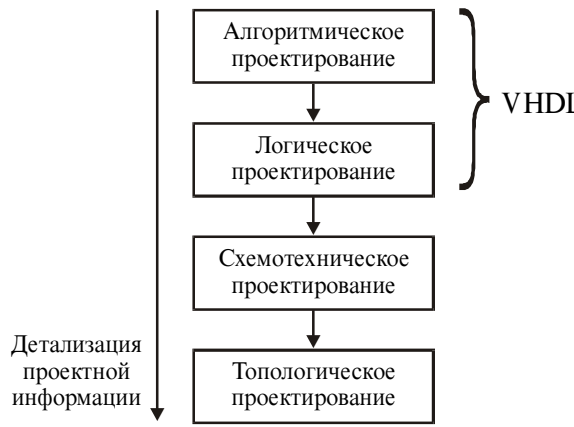


Рис. 1.1. Этапы проектирования СБИС

Начальные этапы алгоритмического и логического проектирования являются определяющими. Именно на данных этапах определяются основные характеристики систем и устройств: сложность, быстродействие, тестопригодность.

Язык VHDL является входным языком многих современных систем автоматизированного проектирования заказных, полузаказных (вентильных матриц, базовых матричных кристаллов), программируемых логических интегральных схем – Programmable Logic Devices (PLD) – и программируемых пользователями вентильных матриц – Field-Programmable Gate Arrays (FPGA) [14].

VHDL предназначен, в первую очередь, для моделирования проектируемой цифровой системы на начальных этапах

проектирования – алгоритмическом и логическом. VHDL позволяет описывать поведение, т.е. алгоритмы функционирования цифровых систем, а также проводить иерархическое функционально-структурное описание систем, имеет средства для описания параллельных асинхронных процессов, регулярных (систолических) структур и в то же время имеет все признаки языка программирования высокого уровня – позволяет создавать свои типы данных, имеет широкий набор арифметических и логических операций и т.д.

VHDL имеет свою историю развития, этот язык является результатом коллективной работы многих людей, приложивших немалые усилия для того, чтобы язык получил всемирное распространение и стал фактически мировым стандартом. История развития VHDL связана со стандартами языка.

Стандарты языка VHDL

VHDL'87. Язык VHDL был разработан в США по инициативе министерства обороны этой страны. В 1987 г. VHDL был принят в качестве стандарта ANSI/IEEE Std 1076-1987. Данный стандарт часто называют VHDL'87 [23].

VHDL'93 соответствует стандарту ANSI/IEEE Std 1076-1993, который появился в 1993 г. Данный **основной** стандарт [24] ориентирован на проектирование цифровых систем и поддерживается в настоящее время основными промышленными системами моделирования и синтеза. Книга [16] посвящена отличиям стандарта VHDL'93 от стандарта VHDL'87.

VHDL-AMS соответствует стандарту Std 1076.1-1999 и включает расширения, дающие возможность описания моделей аналоговых и смешанных (цифро-аналоговых) схем [25].

Синтезируемое подмножество языка VHDL описано в стандарте P1076.6 (IEEE P1076.6/D1.12 Draft Standard For VHDL Register Transfer Level Synthesis, 1998). В стандарте определяются те конструкции языка, которые могут быть реализованы определенными логическими схемами. Стандарт ориентирован на единообразность описания (и понимания) алгоритмических конструкций в различных САПР, что должно обеспечить

переносимость проектов цифровых систем из одной системы проектирования в другую.

VHDL изначально разрабатывался как язык моделирования и использовался именно для целей моделирования. В последнее время на передний план в области САПР вышла проблема синтеза логических схем – центральная проблема этапа логического проектирования. С ростом степени интеграции и усложнением микроэлектронного элементного базиса внимание к данной проблеме не только не ослабевает, но, наоборот, даже усиливается. Это объясняется тем, что в развитых системах сквозного проектирования этап логического синтеза приходится автоматизировать – проблема эффективного логического синтеза перешла из разряда теоретических в практическую область. Поэтому был разработан ряд систем автоматического синтеза схем по описаниям на языке VHDL. Одной из наиболее известных является система ***Leonardo*** – так сокращенно будем называть систему автоматического синтеза логических схем LeonardoSpectrum HDL Synthesis. Проекты цифровых систем и устройств (VHDL-программы, VHDL-коды) представленные в данной книге, моделировались с помощью системы моделирования ***ModelSim*** (ModelSim EE/plus, версия 5.2e), с помощью системы Leonardo (версия v1999.1) проводился автоматический синтез схем.

С использованием языка VHDL алгоритмическое и логическое проектирование цифровых систем, реализуемых на заказных и полузаказных СБИС, включает следующие основные этапы.

1. Составление и моделирование алгоритмического VHDL-описания, представляющего собой проект будущей цифровой системы.

Роль моделирования огромна – только выполняя моделирование, можно наблюдать поведение системы на заданных входных воздействиях, устанавливать параллелизм и асинхронность происходящих в системе процессов, строить временные диаграммы, исправлять ошибки, добиваясь нужного функционирования системы во времени.

Важнейшей особенностью данного этапа является то, что VHDL-программа должна быть написана на подмножестве языка VHDL, называемым ***синтезируемым*** подмножеством, т.е. таким

подмножеством, в рамках которого возможна схемная реализация конструкций языка (операторов, типов данных и т.д.).

! *Изучение синтезируемого подмножества языка VHDL, которое (подмножество) используется в системе Leonardo для стандарта VHDL'93, и является одной из основных целей данной книги.*

! *В дальнейшем важные положения, на которые следует обратить особое внимание, также будут сопровождаться восклицательным знаком.*

2. Автоматический синтез схемы в заданной целевой библиотеке синтеза и представление синтезированной логической схемы на языке VHDL.

С точки зрения представления проектной информации в процессе синтеза происходит детализация проектной информации: достаточно абстрактное алгоритмическое описание заменяется структурным описанием логической схемы, которая должна реализовать требуемое поведение.

3. Моделирование синтезированной логической схемы на языке VHDL с целью установления эквивалентности (совпадения) поведения алгоритмического описания и полученной результирующей логической схемы.

Повторное моделирование необходимо по следующей основной причине. Автоматический синтез проводится без учета временных соотношений между относительными задержками сигналов, определенных в алгоритмической модели. При автоматическом синтезе требуемые задержки сигналов не принимаются в расчет, основное внимание уделяется правильной реакции системы на заданные наборы входных сигналов, т.е. правильно реализуются функции системы, но **не гарантируются** правильные реакции системы в точно указанные в алгоритмической модели моменты времени. Например, в функциональной модели могут отсутствовать задержки сигналов (это допускается в языке VHDL), "срабатывание" такого VHDL-кода при моделировании осуществится за нулевое время, но любая аппаратная реализация при моделировании потребует не нулевого времени, другими

словами, моделируемая аппаратура будет генерировать сигналы с некоторой конечной задержкой, в то время как поведенческое описание даст нулевую задержку.

Практически все схемы, приведенные в данной книге и автоматически синтезированные, были повторно промоделированы, их поведение было сравнено с поведением исходных моделей.

Повторное моделирование может вестись не только на языке VHDL – система Leonardo позволяет сохранять построенную логическую схему в различных форматах, которые являются входными для последующих этапов проектирования заказных (специализированных) СБИС либо ПЛИС.

По существу на этапе 3 решается задача **верификации** (сравнения) исходного алгоритмического и результирующего структурного описания. Для получения достаточного соответствия между поведением двух моделей может потребоваться некоторое изменение алгоритмической модели, например, введение определенных временных задержек для некоторых сигналов, после чего нужно будет опять выполнять синтез, и так далее – этапы алгоритмического и логического проектирования будут выполняться многократно, пока задача верификации не будет решена. Некоторые аспекты методики верификации синтезируемых VHDL-описаний рассмотрены в работе [13], посвященной стандарту IEEE P1076.6 на синтезируемое подмножество языка VHDL. Повторное моделирование обычно требует построения тестов для схем по тестам для алгоритмических описаний. Здесь важно знать, как при синтезе преобразуются (кодируются) входные и выходные данные. Кодирование данных изучается в разд. 4.5, однако временная верификация схем, построенных по VHDL-описаниям, требует отдельного изучения и в данной книге не рассматривается.

Основные этапы автоматического синтеза логической схемы в системе Leonardo представлены на рис. 1.2. Высокоуровневый и логический синтез будут подробно рассмотрены в следующих разделах первой главы, особенности систем моделирования и синтеза будут рассмотрены в четвертой главе.

В качестве целевой библиотеки синтеза, которая использовалась при реализации практически всех (исключения будут особо оговорены) алгоритмических VHDL-описаний, представленных в данной книге, была “урезанная” библиотека

проектирования полузаказных СБИС на основе отечественных базовых матричных кристаллов (БМК) серии К1574 [11]. Таким образом, представленные синтезируемые VHDL-описания, особенно типовых устройств и блоков, могут представлять интерес для проектирования полностью заказных СБИС и полузаказных СБИС на основе БМК.

Рис. 1.2. Основные этапы автоматического синтеза в системе Leonardo

Если проектирование в Leonardo ведется для указанной проектировщиком ПЛИС, то полученное в результате работы синтезатора структурное описание схемы должно быть передано в соответствующую САПР, которая позволяет создать файл программирования ПЛИС, либо создать файл для загрузки конфигурации – для FPGA. В качестве такой САПР, например для ПЛИС фирмы Altera, может быть MAX+PLUSII.

1.2. Высокоуровневый синтез

Процесс получения функционально-структурного описания системы (схемы) по ее алгоритмическому описанию на языке высокого уровня называется *высокоуровневым синтезом* в отличие от *логического* синтеза, когда по функционально-структурному описанию цифровой системы надо получить логическую схему из заданных базисных логических элементов (рис.1.2). Логический синтез будет рассмотрен в следующем разделе.

Основными проблемами высокоуровневого синтеза являются

- преобразование (кодирование) данных (входных, выходных, промежуточных), используемых в алгоритмических описаниях в данные, используемые для описания результирующих логических схем;
- реализация операторов и конструкций языка высокого уровня соответствующими логическими подсхемами – *компилятами*;
- оптимизация (минимизация) аппаратурных затрат в рамках критериев “сложность – быстродействие”.

Так как электронные схемы строятся чаще всего на основе принципа двоичного представления информации, то данные любых типов кодируются двоичными (булевыми) кодами. Виды используемых в системе Leonardo кодов будут рассмотрены далее, а пока можно представлять, что различные кодируемые объекты представляются различными наборами (векторами) нулей и единиц.

Проблему реализации операторов компилятами рассмотрим более подробно. Термин “компилят” в какой то мере объясняется тем, что высокоуровневый синтез в системе Leonardo осуществляется на основе ***компилятивного подхода.***

! *Каждый синтезируемый оператор или конструкция языка VHDL заменяется при автоматическом синтезе своей подсхемой – компилятом.*

Рассмотрим простейший пример – схемную реализацию оператора сложения целых чисел, выбираемых из числового интервала [0, 3], т.е. устройства для сложения любой пары чисел из множества {0, 1, 2, 3}.

Приведем первое в данной книге VHDL-описание поведения такого устройства, называемого сумматором. Программу, написанную на языке VHDL, будем называть *VHDL-кодом*.

Синтезируемый VHDL-код.

```
entity adder_2 is
port( A, B : in integer range 0 to 3;
         C : out integer range 0 to 6);
end adder_2;
architecture functional of adder_2 is
begin
C <= A + B;
end functional;
```

Термин "*Синтезируемый VHDL-код*", которым предваряется программа на языке VHDL, свидетельствует о том, что по данному алгоритмическому описанию автоматически с помощью синтезатора Leonardo может быть построена логическая схема, реализующая требуемое поведение проектируемой системы – в данном случае сумматора. В тексте VHDL-кода жирным шрифтом отмечены зарезервированные слова языка VHDL, не жирным шрифтом показаны слова, выбранные *проектировщиком,* – так будем называть в дальнейшем разработчика проектов цифровых систем на языке VHDL, или программиста, пишущего программы на языке VHDL.

Далее, слово **entity** обозначает описание проектируемой системы "в целом", т.е. *интерфейса* схемы. Интерфейс схемы состоит из имени схемы (adder_2) и описания входов, выходов. Входы и выходы схемы называются на языке VHDL портами (**port**). В качестве входных портов, что отмечается словом **in**, являются два порта с именами A, B. Выходным (**out**) является порт с именем C. На входные порты поступают целые числа из интервала [0, 3]. Декларация (начальное определение) такого типа данных осуществляется указанием типа **integer** и диапазона **range** 0 **to** 3. Результирующее число C (C=A+B) может быть в диапазоне **range** 0 **to** 6, что и указывается после типа out выходного сигнала.

Заканчивается описание интерфейса схемы adder_2 словами **end** adder_2**;**

С ключевого (зарезервированного) слова **architecture** начинается алгоритмическая часть описания (архитектурное тело), собственно и определяющая требуемое поведение проектируемой системы. Архитектурное тело имеет свое уникальное имя functional, которое связывается (**is**) с интерфейсом схемы.

Алгоритмическое описание начинается с ключевого слова **begin** и состоит из двух операторов: + (оператор сложения) и <= (оператор назначения сигнала). Операторы языка VHDL будут изучены далее, а пока можно представлять, что оператор <= передает значение A+B из правой части выражения C **<=** A + B; в левую часть C, т.е. назначает (определяет) выходной сигнал схемы. Заканчивается описание архитектурного тела functional словами **end** functional**;**

Итак, имеем довольно простое алгоритмическое описание, которое требуется преобразовать в структурное описание, вернее говоря, построить логическую схему сумматора и описать полученную схему на формальном языке. Язык VHDL позволяет описывать не только алгоритмы, но и структуры синтезированных схем, поэтому в данной книге *результаты синтеза также будут представляться в виде VHDL-кодов*.

Первое, что нужно сделать при переходе к логической схеме, – это представить числа в двоичном коде. Закодируем число A двумя булевыми переменными aa2, aa1. Пусть A=(aa2, aa1), т.е. переменная aa2 задает старший, а переменная aa1 – младший разряд двоичного представления числа A. Аналогично можно представить B=(bb2, bb1), C=(cc3, cc2,cc1), где bb2, cc3 – старшие разряды, bb1, cc1 – младшие разряды чисел B, C соответственно. Интерфейс схемы, реализующей оператор + , будет иметь вид (рис. 1.3).

Затем возникает вторая проблема: как получить логическую схему сумматора, когда осуществлен переход к булевым переменным. Сумматоры – широко известные в практике проектирования устройства, известны самые различные схемы сумматоров – сумматоры с последовательным переносом, параллельным переносом и т.д. Можно создать *библиотеку* сумматоров и выбирать из нее требуемый по разрядности сумматор. Таким образом, для двухразрядного сумматора можно найти

подходящую библиотечную реализацию. Однако число разрядов складываемых чисел может варьироваться, кроме того, могут складываться числа, различные по разрядности, поэтому библиотека очень сильно разрастется – библиотечный подход к реализации стандартных операторов языка высокого уровня не приемлем на практике.

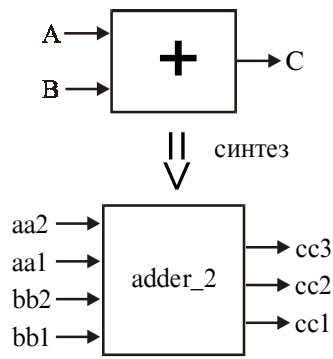


Рис.1.3. Реализация оператора "+" логической схемой adder_2

Схемы, реализующие стандартные операторы (например, арифметические операторы сложения, вычитания, умножения и деления чисел), назовем *генерируемыми модулями*. Генерируемый модуль – параметризуемый по разрядности компилят. В Leonardo автоматический синтез разделен условно на два этапа высокоуровневый и логический синтез (рис. 1.2). После высокоуровневого синтеза схема представляется в виде генерируемых модулей и комбинационных двухвходовых логических элементов, после логического синтеза получается результирующая схема из элементов целевой библиотеки синтеза.

Осуществить получение логических уравнений для описания функций генерируемого модуля можно различными способами.

Генерация уравнений из структурного описания. Можно составит формальное описание регулярной схемы устройства на случай произвольной (*N*) разрядности и получать структурное

описание для конкретного значения *N*. Такая возможность существует и в языке VHDL, примеры будут рассмотрены далее. Относительно сумматоров такая возможность достаточно очевидна – можно описать каскадную схему многоразрядного сумматора в виде последовательно соединенных одноразрядных сумматоров и "высекать" сумматор требуемой разрядности. Далее такие структуры сумматоров будут приведены и описаны на языке VHDL. Весьма компактное описание функционирования одноразрядного сумматора можно хранить библиотеке системы синтеза, получение логических функций для многоразрядного сумматора сведется к генерации рекуррентных логических выражений.

Генерация уравнений из таблиц истинности. Такой подход пригоден для схем комбинационной логики. Например, можно написать специальную программу, генерирующую таблицу истинности функций *n*-разрядного сумматора, затем получить алгебраические выражения логических функций и представить их в виде комбинационной схемы из двухвходовых логических элементов. Возможны, очевидно, и комбинированные подходы. Далее в книге будут даны различные способы описания сумматоров.

В виде генерируемых модулей представляются арифметические и другие операции языка VHDL. Логические функции генерируемых модулей получаются в системе Leonardo по требуемой разрядности операндов в операциях языка VHDL. Для проектировщика внутреннее промежуточное представление схемы после высокоуровневого синтеза возможно лишь на уровне представления схемы в графическом редакторе, детали внутренней работы Leonardo как на этапе высокоуровневого, так и логического синтеза недоступны для проектировщика.

Мы рассмотрели реализацию только одного оператора языка VHDL. Если в алгоритмическом описании имеется множество операторов, то на этапе высокоуровневого синтеза осуществляется замена конструкций языка VHDL логическими схемами – триггерами и двухвходовыми комбинационными логическими элементами. Напомним, что исходное VHDL-описание должно быть синтезируемым.

Рассмотрим пример VHDL-кода, в котором имеется несколько операторов. Пусть нам требуется описать цифровую систему, которая выполняет (реализует) следующее поведение. На вход

системы поступают два числа A и B, принимающие значения из множества {1,2,3,4} целых чисел. Система должна на первом шаге одновременно сложить и перемножить данные числа, на втором шаге – из произведения A×B вычесть сумму A+B. Оператор умножения в языке VHDL обозначается символом `*`.

Связи операторов алгоритмического описания изображены на рис. 1.4, а.

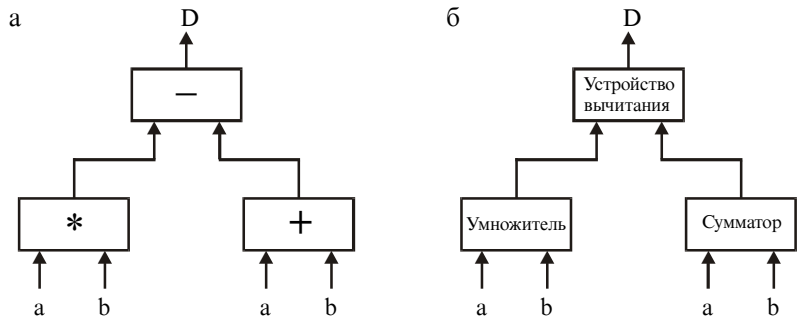


Рис. 1.4. Покрытия алгоритмического описания компилятами:
а - связи операторов ; б - структурная схема

На языке VHDL данное поведение можно выразить следующим образом.

Синтезируемый VHDL-код

```
entity vlsi is
port (a,b : in INTEGER range 1 to 4;
        D : out INTEGER range -1 to 8);
end vlsi;

architecture functional_1 of vlsi is
signal v : integer range 2 to 8;
signal w : integer range 1 to 16;
begin
addition : process(a,b)
```