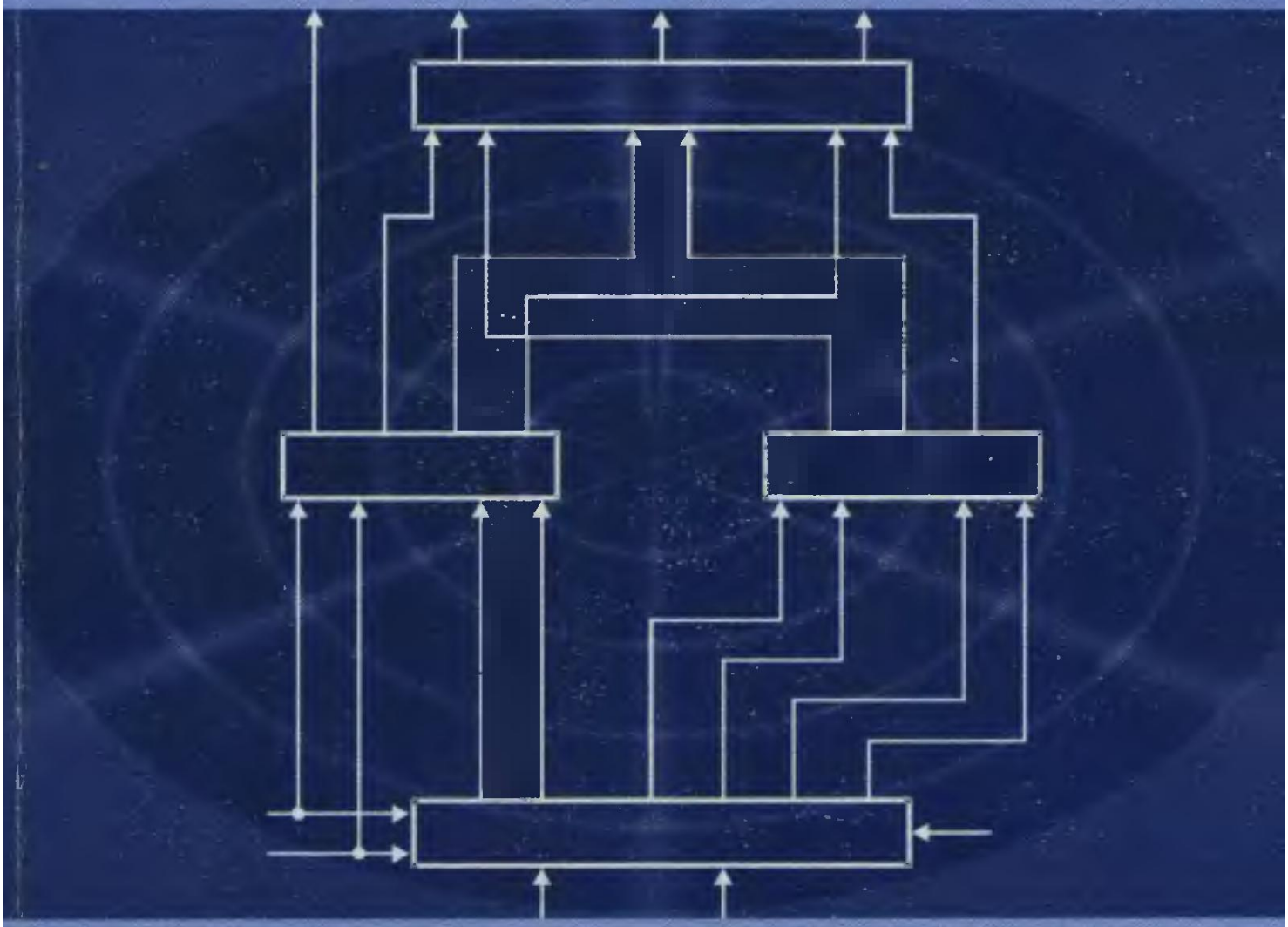
П. Н. Бибило

ОСНОВЫ ЯЗЫКА VHDL



URSS

П. Н. Бибило

ОСНОВЫ ЯЗЫКА VHDL

Допущено Министерством образования
Республики Беларусь в качестве учебного пособия
для студентов учреждений, обеспечивающих получение
высшего образования по специальностям
вычислительной техники,
радиоэлектроники и информатики

Издание третье, дополненное

URSS
МОСКВА

ББК 32.973–018

Бибило Петр Николаевич

Основы языка VHDL. Изд. 3-е, доп. — М.: Издательство ЛКИ, 2007. — 328 с.

Описывается применение языка VHDL на алгоритмическом и логическом уровнях проектирования цифровых систем. 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 с языком VHDL и может быть полезна студентам, аспирантам и специалистам, занимающимся разработкой электронной аппаратуры с помощью средств САПР.

Рецензенты:

д-р техн. наук В. Н. Ярмолик,
канд. техн. наук Л. А. Золоторевич

Издательство ЛКИ. 117312, г. Москва, пр-т Шестидесятилетия Октября, д. 9.
Формат 60×90/16. Печ. л. 20,5. Зак. № 1161.

Отпечатано в ООО «ЛЕНАНД».
117312, г. Москва, пр-т Шестидесятилетия Октября, д. 11А, стр. 11.

ISBN 978–5–382–00334–4

НАУЧНАЯ И УЧЕБНАЯ ЛИТЕРАТУРА

E-mail: URSS@URSS.ru
Каталог изданий в Интернете:
http://URSS.ru
Тел./факс: 7 (495) 135–42–16
URSS Тел./факс: 7 (495) 135–42–46

5251 ID 63471

9 785382 003344

Оглавление

Предисловие 5

Глава 1. Основные элементы языка VHDL 8

1.1. Структурное и поведенческое описание цифровой системы 8
1.2. Лексические элементы и типы данных 22
1.3. Декларации 37
1.4. Интерфейс и архитектура объекта 40
1.5. Атрибуты 43
1.6. Имена 49
1.7. Операторы 50
1.8. Понятие сигнала в языке VHDL 57
1.9. Дельта-задержка 62
Упражнения 66

Глава 2. Последовательные и параллельные операторы 71

2.1. Последовательные операторы 71
2.2. Параллельные операторы 88
Упражнения 110

Глава 3. Организация проекта 119

3.1. Подпрограммы 119
3.2. Функции 119
3.3. Процедуры 121
3.4. Разрешающие функции. Пакет std_logic_1164 123

3.5. Архитектура ... 129
3.6. Декларация интерфейса объекта ... 130
3.7. Карта портов и карта настройки ... 133
3.8. Конфигурация ... 134
3.9. Модули проекта и VHDL-библиотеки ... 138
Упражнения ... 140

Глава 4. Примеры проектирования на VHDL ... 143

4.1. Стили описания поведения ... 143
4.2. Формы описания сигналов ... 146
4.3. Описание автоматов ... 150
4.4. Отладка VHDL-описаний ... 168
4.5. Синтезируемое подмножество языка VHDL ... 170
Упражнения ... 182

Глава 5. Методические материалы ... 185

5.1. Лабораторные работы ... 185
5.2. Контрольная работа ... 263
5.3. Тестовые (да/нет) вопросы ... 271
5.4. Контрольные задачи ... 281
5.5. Литература по языку VHDL ... 290
5.6. Интернет-ресурсы ... 292

Литература ... 294

Приложения ... 295

1. Форма задания синтаксических конструкций языка VHDL ... 295
2. Синтаксис языка VHDL'93 ... 296
3. Пакет STANDARD ... 321
4. Пакет STD_LOGIC_1164 ... 323

Предисловие

Язык **VHDL** (Very high speed integrated circuits Hardware Description Language) является фактически международным стандартом в области автоматизации проектирования цифровых систем, это входной язык многих современных систем автоматизированного проектирования (САПР) как заказных, так и программируемых логических интегральных схем (ПЛИС) — Programmable Logic Devices (PLD) — и программируемых пользователями вентильных матриц — Field-Programmable Gate Arrays (FPGA). VHDL предназначен, в первую очередь, для спецификации — точного описания проектируемых систем и их моделирования на начальных этапах проектирования — алгоритмическом и логическом. С помощью VHDL можно моделировать электронные схемы с учетом реальных временных задержек.

В последнее время весьма успешно разрабатываются и системы синтеза схем по спецификациям на этом языке. Например, используя САПР WebPACK ISE, можно провести моделирование исходного описания схемы на языке VHDL, а затем синтезировать схему и получить файл настройки (конфигурации) микросхемы типа FPGA фирмы Xilinx. Использование САПР MAX+PLUSII позволяет решать аналогичные задачи для программируемых микросхем, выпускаемых фирмой Altera. Для заказных СБИС могут быть использованы САПР фирмы Mentor Graphics: система моделирования ModelSim позволяет провести моделирование описаний, представленных на языке VHDL, система синтеза LeonardoSpectrum позволяет автоматически получать по описаниям на языке VHDL схемы в заданных базисах логических элементов. Такие крупнейшие фирмы — производители программного обеспечения САПР в области микроэлектроники, как Cadence, Synop-

sys и многие другие используют язык VHDL в качестве языка исходного описания проектов.

VHDL — это мощный язык, он 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 обладает возможностями языка программирования высокого уровня — позволяет создавать свои типы данных, имеет широкий набор арифметических и логических операций и т.д.

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. Затем язык был усовершенствован, новый стандарт ANSI/IEEE Std 1076-1993 (стандарт VHDL'93) появился в 1993 г. Язык VHDL развивается, ему посвящаются международные конференции, выходят научные журналы, в которых изучаются проблемы использования VHDL. Он стал языком разработки международных проектов, в том числе осуществляемых с помощью всемирной компьютерной сети Internet. Знакомство с этим языком необходимо для эффективной работы по созданию самой разнообразной электронной аппаратуры на современной элементной базе сверхбольших интегральных схем.

Данная книга предназначена для первоначального ознакомления с языком VHDL. Приводимые в ней определения синтаксических конструкций языка ориентируются на стандарт VHDL'93. В приложении дается синтаксис стандарта VHDL'93. Основной упор в книге делается на применение языка VHDL на этапах алгоритмического и логического проектирования цифровых систем, она оказалась незаменимой при начальном изучении языка VHDL студентами Белгосуниверситета (специализация «Математическая электроника») и студентами факультета компьютерного проектирования Белорусского государственного университета информатики и радиоэлектроники. При написании книги за основу был взят материал лекционных курсов, читаемых автором в этих вузах на протяжении ряда лет. В конце каждой главы даются вопросы, задачи и упражнения для самостоятельной работы. Книга будет

полезна как для студентов и аспирантов соответствующих специальностей, так и для специалистов, занимающихся разработкой электронной аппаратуры с помощью средств САПР.

Данное третье издание книги пополнено пятой главой, содержащей методические материалы, взятые из практики преподавания языка VHDL в университетах.

Глава 1

Основные элементы языка VHDL

1.1. Структурное и поведенческое описание цифровой системы

Цифровая система может быть описана на уровне поведения посредством описания функций, реализуемых системой, и на структурном уровне.

Структурное описание — это описание системы в виде совокупности компонент (подсхем, элементов) и связей между компонентами.

Поведенческое описание — это описание системы при помощи некоторых процедур на уровне зависимостей выходов от входов. Иначе говоря, поведенческое описание задает алгоритм, реализуемый системой.

Цифровая система может быть сложной, как, например, микропроцессор или весьма простой, как логическая схема, состоящая из нескольких логических элементов (вентилей).

Естественно, некоторые компоненты системы в структурном описании могут состоять из нескольких частей — компонент более низкого уровня иерархии описания. Представляя отношение вхождения подсхем в схемы в виде графа, можно получить дерево (граф) иерархии описания всей системы.

Например, пусть цифровая система **S** (рис. 1.1) реализует следующий алгоритм. На входные полюсы системы **S** подаются два двухразрядных числа **a** = (a2, a1), **b** = (b2, b1), где a2, b2 — старшие разряды чисел **a**, **b** соответственно и x — управляющий сигнал [3].

Если x = 1, система **S** должна перемножить числа **a**, **b** и выдать четырехразрядное число **d** = (d4, d3, d2, d1), где **d** = **a** × **b**.

Если x = 0, то числа **a, b** должны быть сложены, при этом в разряде d4 всегда должен быть нуль, в разряде d3 — перенос c2, в разряде d2 — старший разряд суммы s2, в разряде d1 — младший разряд суммы s1.

Предполагается, что

$$(a2, a1) + (b2, b1) = (c2, s2, s1).$$

Мы описали алгоритм, который должна реализовать система S, на русском языке. Однако для автоматизированного (компьютерного) проектирования требуется формальная спецификация.

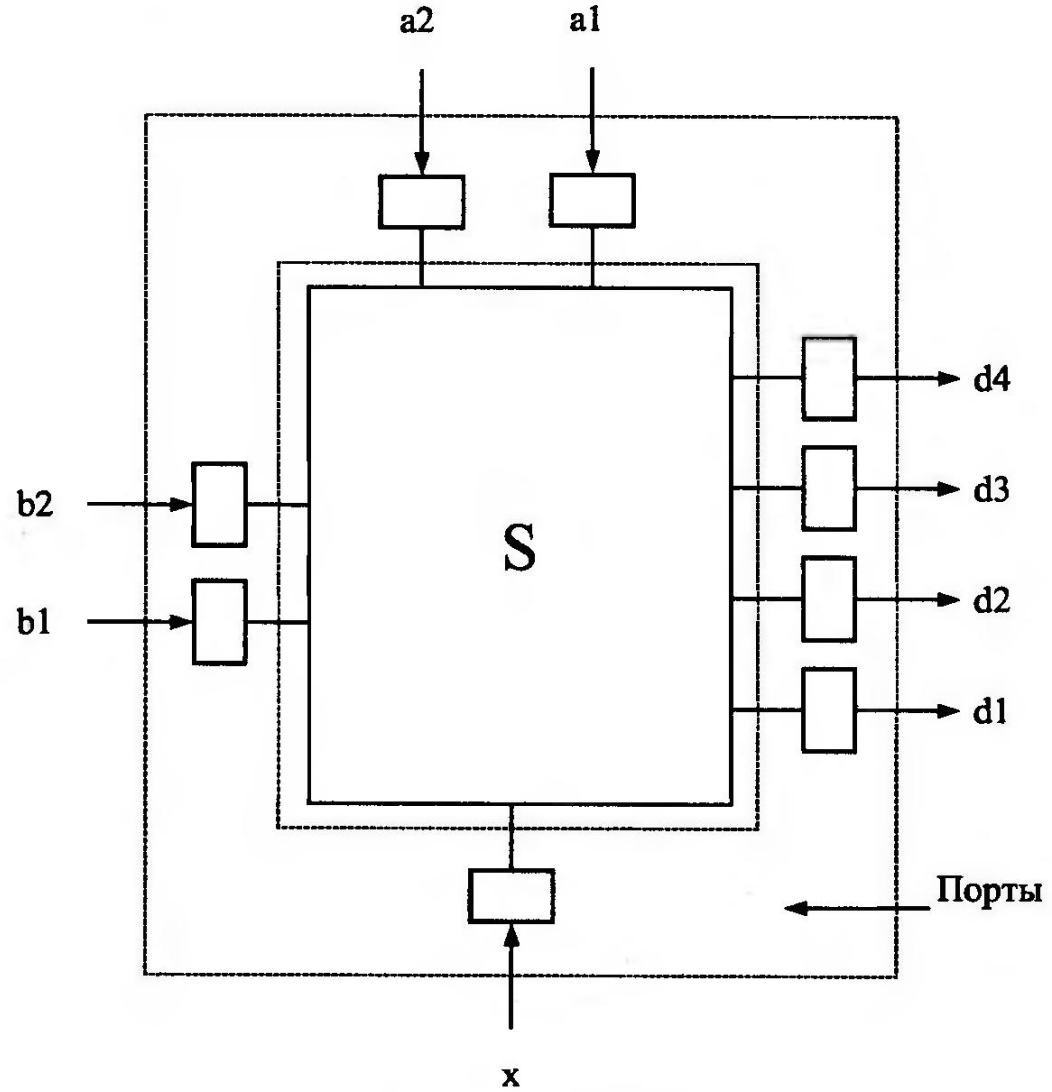


Рис. 1.1. Цифровая система S

Формальные спецификации должны быть записаны на соответствующем формальном языке. Позже будет показано, что алгоритм, реализуемый системой **S**, может быть очень коротко описан на языке VHDL. Если же рассматривать структурный уровень описания системы **S**, то можно легко увидеть, что в систему **S** входит *двухразрядный сумматор* и *двухразрядный умножитель*.

Двухразрядный сумматор — это устройство для сложения двухразрядных чисел, *двухразрядный умножитель* — устройство для перемножения двух чисел, каждое из которых имеет только два разряда. В систему **S** должно входить также простейшее устройство управления и схема дизъюнктивного объединения выходных сигналов. Структура системы **S** изображена на рис. 1.2.

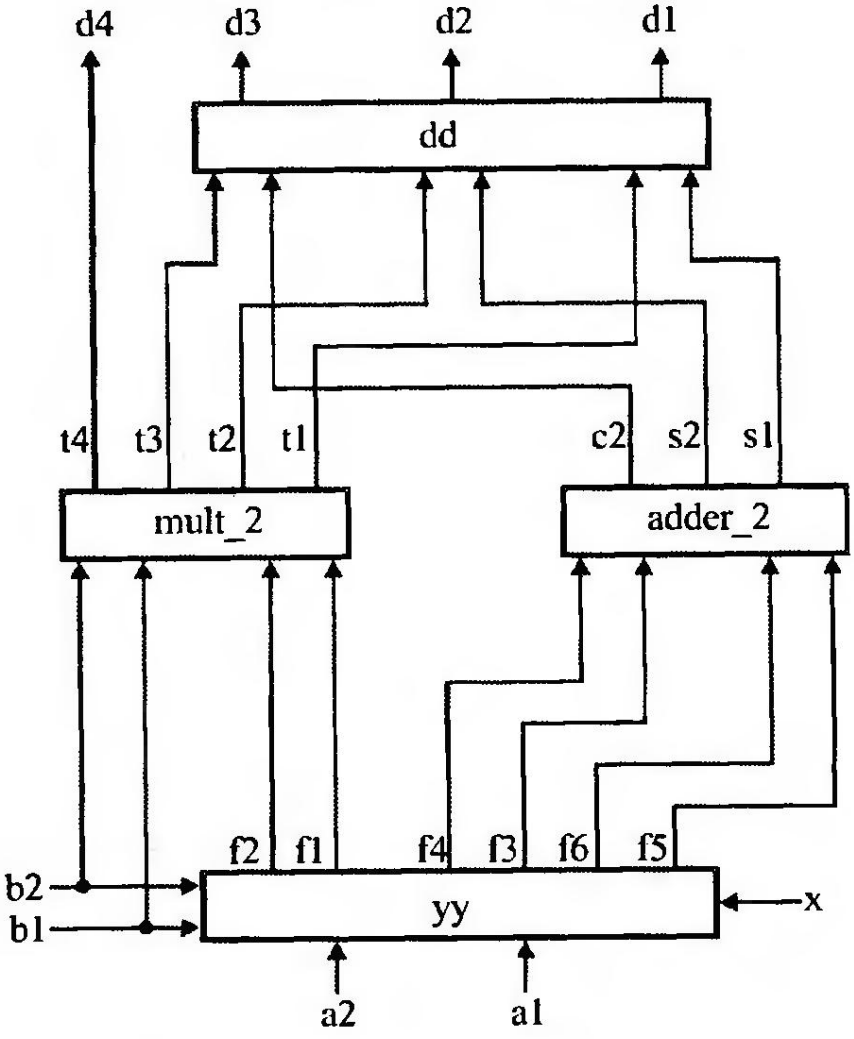


Рис. 1.2. Структура цифровой системы **S**

На рис. 1.2 **adder_2** — двухразрядный сумматор, **mult_2** — двухразрядный умножитель, **YY** — устройство управления, **dd** — схема дизъюнктивного формирования выходных сигналов. Схему **dd** образуют три дизъюнктора. Устройство управления **YY** является весьма простым и функционирует следующим образом:

если x = 0, то

(a2, a1) = (f4, f3),
(b2, b1) = (f6, f5),
(f2, f1) = (0, 0),

т. е. числа **a, b** подаются на вход сумматора **adder_2**.

Если же x = 1, то

(f4, f3) = (0, 0),
(f6, f5) = (0, 0),
(a2, a1) = (f2, f1),

т. е. числа **a, b** подаются на вход умножителя **mult_2**.

Пусть описание системы **S** имеет имя VLSI_1. Тогда иерархия структурного описания системы **S** будет иметь вид (рис.1.3)

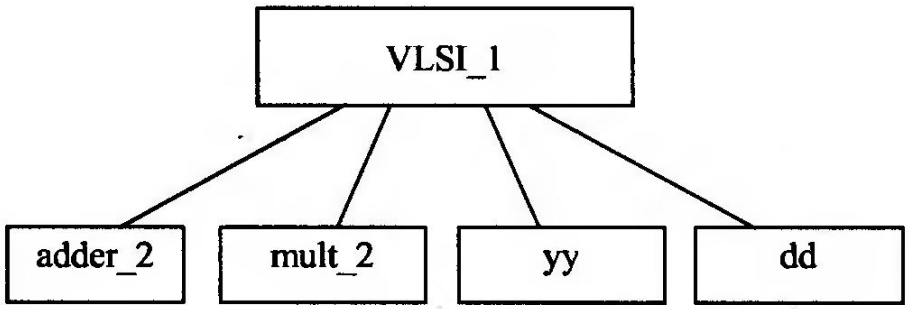


Рис. 1.3. Дерево иерархии структурного описания системы **S**

Рассмотрим двухразрядный умножитель **mult_2.** На вход данной схемы (рис. 1.4) поступают сигналы r1, r0, s1, s0. Сигналы r1, r0 интерпретируются как двухразрядное целое число **r** = (r1, r0), сигналы s1, s0 — как двухразрядное целое число **s** = (s1, s0).

Выходные сигналы в t3, t2, t1, t0 представляют собой разряды числа **t** = (t3, t2, t1, t0) — произведения чисел **s, r**:

(t3, t2, t1, t0) = (r1, r0)×(s1, s0).

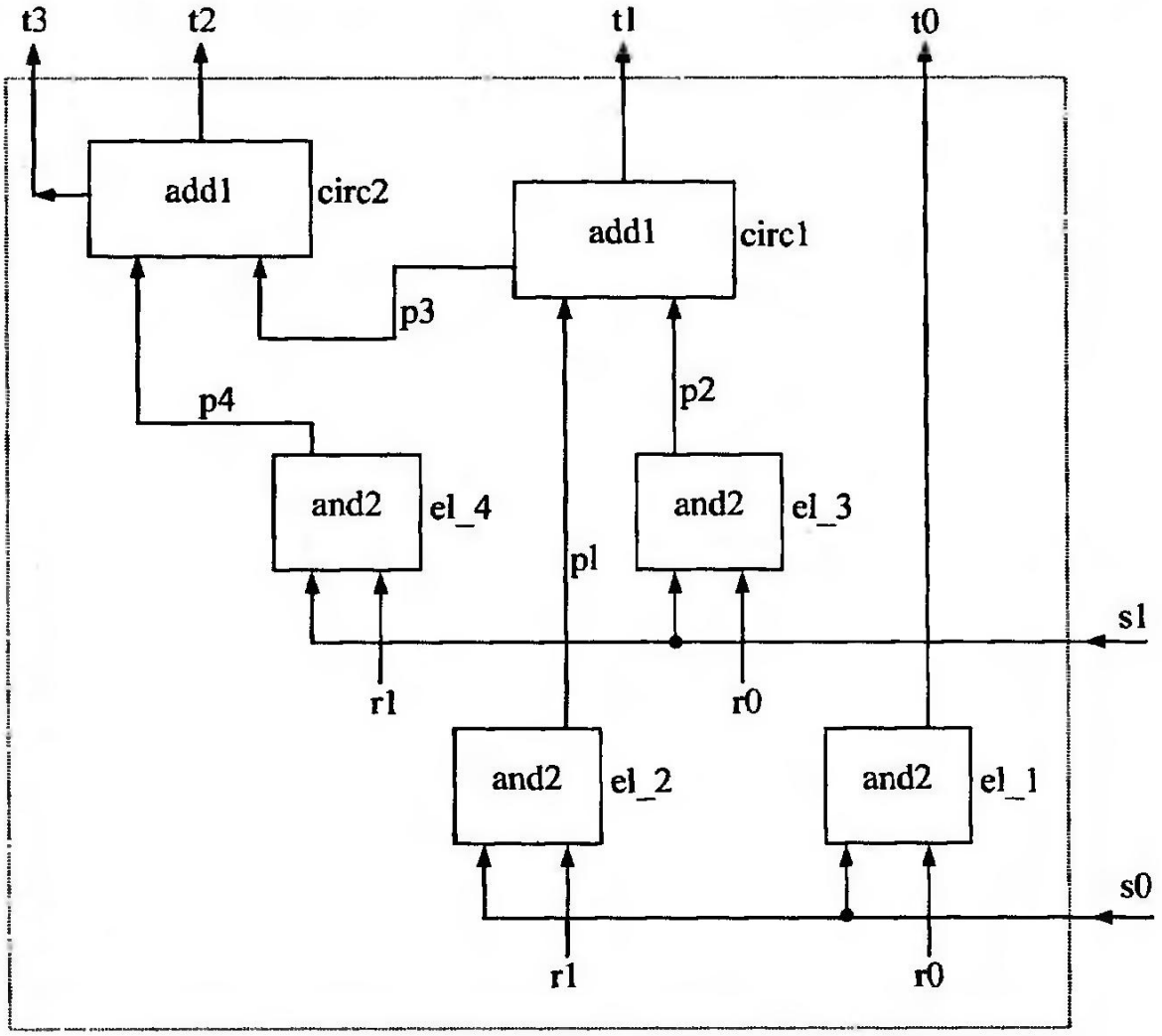


Рис. 1.4. Двухразрядный умножитель mult_2

В схему входят элементы двух типов — **add1** и **and2**.

Элемент **and2** представляет собой логический элемент И — двухвходовый конъюнктор.

Заметим, что на языке VHDL знак & употребляется не для обозначения логической операции «И» (конъюнкции), а для операции конкатенации векторов.

Оператором логической конъюнкции служит оператор **and**, поэтому описание функции элемента **and2** на языке VHDL выглядит следующим образом:

```
y <= x1 and x2;
```

где x1, x2 — имена входных сигналов, y — имя выходного сигнала, <= — оператор назначения сигнала (будет рассмотрен позже).

Элемент **add1** представляет собой одноразрядный полусумматор, функционирование которого описывается таблицей истинности (табл. 1.1).

Таблица 1.1

b1	b2	c1	s1
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

В данном случае b1, b2 интерпретируются как одноразрядные числа, s1 — сумма, c1 — перенос в следующий разряд.

В привычной математической записи булевы функции s1, c1 могут быть представлены следующим образом:

$$s1 = b1 \oplus b2 = (\ \bar{b}1 \wedge b2\ ) \vee (\ b1 \wedge \bar{b}2\ ),$$
$$c1 = b1 \wedge b2.$$

В языке VHDL функционирование элемента **add1** записывается следующим образом:

```
s1 <= ((b1 and (not b2)) or (not b1) and b2));
c1 <= b1 and b2;
```

где **or** — оператор логичсской дизъюнкции;

and — оператор логической конъюнкции;

not — оператор отрицания.

Итак, в дерево проекта схемы умножителя, которую назовем **mult_2**, входят элемент and2 и подсхема add1. Если использовать логические элементы **or2** (двухвходовый дизъюнктор) и **inv** (инвертор) и **and2** (двухвходовый конъюнктор) для реализации подсхемы **add1**, то дерево проекта будет трехуровневым.

Элементы **and2**, **or2**, **inv** являются *листьями* проекта. Листья проекта не имеют составных частей и называются *примитивами* проекта. Примитив описывается только на поведенческом уровне.

Объектами проекта для двухразрядного умножителя являются **mult_2, add1, and2.**

Обозначение корня дерева (**mult_2**) является именем проекта.

Каждый объект проекта имеет два различных типа описаний:

— описание объекта «в целом» (**entity**);

— описание архитектуры объекта (**architecture**).

Упрощенно можно сказать, что описание объекта «в целом» состоит из имени объекта и описания портов (входов, выходов) объекта. Описание объекта «в целом» в языке VHDL носит название «интерфейс» объекта. Чтобы отличать один объект проекта от другого, термин entity будет пониматься иногда и как объект проекта.

Для сигналов, подаваемых, снимаемых с портов указывается вид (режим, направление) сигнала: входной (**in**), выходной (**out**) и его тип.

Описание объекта проекта **and2** имеет вид.

```
entity and2 is                  -- декларация имени объекта проекта
port (x1, x2: in BIT;           -- декларация входных портов
          y: out BIT);          -- декларация выходного порта
end and2;

architecture functional of and2 is — декларация архитектуры
begin
y <= x1 and x2;                  -- описанис функции объекта
end functional;
```

В тексте данной программы имеются комментарии. Комментарий начинается двумя смежными дефисами и продолжается до конца строки.

В данном примере BIT — это тип сигнала. Архитектурное тело может определять поведение объекта проекта непосредственно (быть примитивом), либо представлять собой структурную деком-

позицию на более простые компоненты. Описание объекта проекта **add1**:

```
entity add1 is
port (b1, b2 : in BIT; c1, s1 : out BIT);
end add1;
architecture struct_1 of add1 is
begin
s1<= ((b1 and (not b2)) or ((not b1) and b2));
c1<= b1 and b2;
end struct_1;
```

Описание объекта проекта **mult_2**:

```
entity mult_2 is
port (s1, s0, r1, r0 : in BIT;
      t3, t2, t1, t0 : out BIT);
end mult_2;
architecture structure of mult_2 is
component
add1 port (b1, b2: in BIT;  c1, s1: out BIT);
end component;
signal p1, p2, p3, p4 : BIT;
begin
t0 <= r0 and s0;  p2 <= r0 and s1;  p1 <= r1 and s0;  p4 <= r1 and s1;
circ1: add1   port map (p1, p2, p3, t1);
circ2: add1   port map (p3, p4, t3, t2);
end structure;
```

В описании архитектуры объявляются (декларируются) две подсхемы (компоненты). После ключевого слова **begin** приводятся экземпляры описаний. Каждый экземпляр имеет уникальную метку (circ1, circ2 — метки). Каждый экземпляр имеет карту портов (**port map**). Карта портов отражает связь между входами, выходами описаний компонента и экземплярами компонента. Заметим, что в данном описании мы использовали понятие компонента (подсхемы) для add1, в то время как логические элементы И схемы мы описали на

функциональном уровне, не используя понятие компонента. Возможность проведения таких смешанных описаний является важной полезной особенностью языка VHDL. Данная гибкость весьма удобна при проектировании на начальных этапах, когда важно получить точное алгоритмическое описание, не вдаваясь в детали структурной организации некоторых частей проекта.

Аналогично можно рассмотреть двухразрядный сумматор **adder_2** (рис. 1.5), состоящий из двух подсхем **add1, add2**, где **add1** — это уже известный нам одноразрядный полусумматор, а **add2** — одноразрядный сумматор, функционирование которого описывается следующей таблицей истинности (табл. 1.2):

Таблица 1.2

c1	a1	a2	c2	s2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Описание объекта проекта **add2**:

```
entity add2 is
port (c1, a1, a2: in BIT; c2, s2: out BIT);
end add2;
architecture struct_1 of add2 is
begin
s2 <= ((not c1) and (not a1 ) and a2 ) or ((not c1) and a1 and (not a2))
or ( c1 and (not a1) and (not a2) ) or (a1 and a2 and c1);
c2 <= (a1 and c1) or (a2 and c1) or (a1 and a2 );
end struct_1;
```

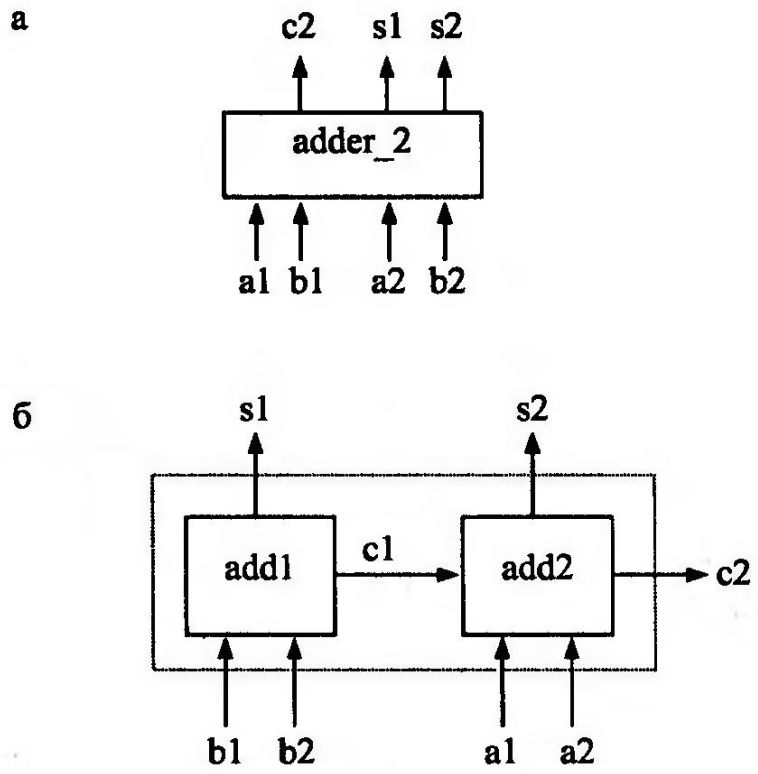


Рис. 1.5. Двухразрядный сумматор (a1, b1)+(a2, b2) = (c2, s2, s1):
а — условное обозначение; б — схема в виде каскадного соединения одноразрядного полусумматора add1 и одноразрядного сумматора add2

В дерево проекта для подсхемы **adder_2** входят подсхемы add1, add2, а VHDL-описание имеет вид

```
entity adder_2 is
port (a1, b1, a2, b2 : in BIT;
      c2, s2, s1 : out BIT);
end adder_2;

architecture structure of adder_2 is
component
add1
port (b1, b2: in BIT;
     c1, s1: out BIT);
end component;
```

```
component add2
port(c1, a1, a2: in BIT;
    c2, s2: out BIT);
end component;
signal c1: BIT;
begin
circ1: add1
port map (b1, b2, c1, s1);
circ2: add2
port map (c1, a1, a2, c2, s2);
end structure;
```

Можно заметить, что в различные подсхемы входит **add1**, при этом подсхема **add1**, входящая в сумматор **adder_2**, является листом проекта и поэтому описана на поведенческом уровне. Подсхема **add1**, входящая в умножитель **mult_2**, описана на структурном уровне. Предлагаем читателю самостоятельно описать подсхему **add1** в виде объекта проекта, используя примитивы **and2, or2, inv.**

Итак, VHDL-код для структурного описания системы **S** (см. рис. 1.2) выглядит следующим образом:

```
entity vlsi_1 is
port (a2, a1, b2, b1, x: in BIT;
     d4, d3, d2, d1: out BIT);
end vlsi_1;
architecture structure of vlsi_1 is
component adder_2              -- декларация компонента
port (a1, b1, a2, b2: in BIT;
     c2, s2, s1: out BIT);
end component;
component mult_2               -- декларация компонента
port(s1, s0, r1, r0: in BIT;
    t3, t2, t1, t0: out BIT);
end component;
component dd                   -- декларация компонента
port (x1, x2, x3, x4, x5, x6 : in BIT;
     y1, y2, y3 : out BIT);
```

```
end component;
component yy                    -- декларация компонента
port( a2, a1, b2, b1, x : in BIT;
    f6, f5, f4, f3, f2, f1 : out bit);
end component;

signal f1, f2, f3, f4, f5, f6, t4, t3, t2, t1, c2, s2, s1: BIT;
                -- декларация внутренних сигналов
begin
circ1: yy
port map ( a2, a1, b2, b1, x, f6, f5, f4, f3, f2, f1);
circ2: mult_2
port map ( f2, f1, b2, b1, d4, t3, t2, t1);
circ3: adder_2
port map ( f4, f3, f6, f5, c2, s2, s1);
circ4: dd
port map ( s1, t1, s2, t2, c2, t3, d1, d2, d3);
end structure;
```

Описания подсхем YY, dd на функциональном уровне имеют следующий вид:

```
entity YY is
port (a2, a1, b2, b1, x : in BIT;
     f6, f5, f4, f3, f2, f1 : out BIT);
end YY;
architecture struct_1 of YY is
begin
f1<= x and a1;
f2<= x and a2;
f3<= not x and a1;
f4 <= not x and a2;
f5 <= not x and b1;
f6 <= not x and b2;
end struct_1;

entity dd is
```

```
port (x1, x2, x3, x4, x5, x6 : in BIT;
      y1, y2, y3  : out BIT);
end dd;
architecture struct_1 of dd is
begin
y1<= x1 or x2;
y2<= x3 or x4;
y3<= x5 or x6;
end struct_1;
```

Дерево проекта для системы **S** изображено на рис. 1.6.

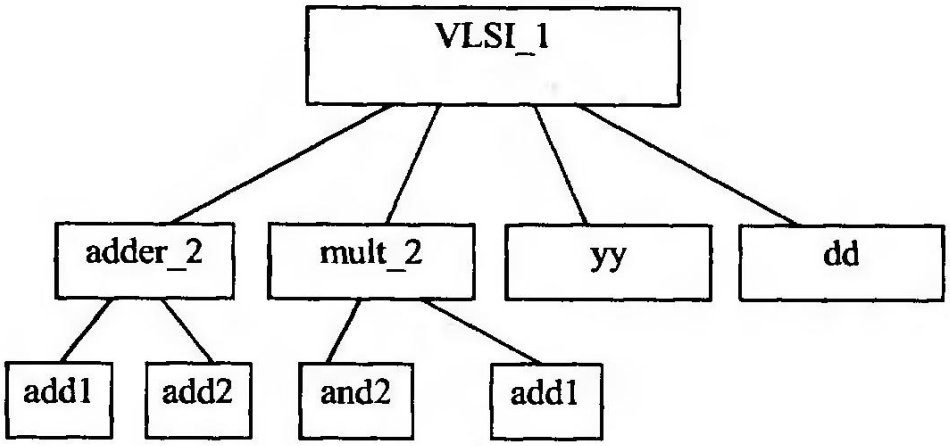


Рис.1.6. Дерево проекта цифровой системы **S**

Приведем один из вариантов алгоритмического описания системы **S** в целом. Следует обратить внимание на то, что входные и выходные сигналы интерпретируются как целые числа, что позволяет сделать алгоритмическое описание весьма компактным.

```
entity vlsi_1 is
port (a, b : in integer range 0 to 3;
    x : in BIT;
    D : out integer  range 0 to 15);
end vlsi_1;
```

```
architecture functional of vlsi_1 is
signal e: integer range 0 to 15;
begin
p0: process(a, b, x)
begin
  if (x = '0') then
e <= a + b;
  elsif ( x = '1') then
e <= a * b ;
  end if;
end process;
D <= e;
end functional;
```

Как видно, оно значительно компактнее структурного описания. Очевидно, что переход от алгоритмического описания к структурному представляет значительный практический интерес. К сожалению, автоматический переход возможен только для подмножества языка VHDL. Такое подмножество языка называется *синтезируемым*.

Получение функционально-структурной схемы по ее алгоритмическому описанию называется *высокоуровневым синтезом* в отличие от *логического* синтеза, когда по функционально-структурному описанию цифровой системы надо получить логическую схему из заданных базисных логических элементов. Программу, осуществляющую по VHDL-описанию синтез схемы, например схемы FPGA, часто называют *компилятором*. Однако в системах моделирования VHDL-кодов под компиляцией также понимается преобразование VHDL-кода в промежуточный язык, с которым оперируют непосредственно программы моделирования. Как и для языков программирования, сборку откомпилированных модулей осуществляет программа LINK. Программа, осуществляющая проверку синтаксической корректности, называется VHDL-*анализатором*.

Приведенное VHDL-описание системы **S** станет понятным позже, когда будут введены типы данных и основные операторы языка VHDL — операторы процессов, назначения сигналов и др.

1.2. Лексические элементы и типы данных

Лексические элементы, разделители, операторы.

Текст на языке VHDL есть последовательность раздельных лексических элементов, таких как

- идентификатор;
- разделитель;
- ключевое (зарезервированное) слово;
- литерал (десятичный, базовый, символьный, строковый, строка бит);
- комментарий.

Смежные лексические элементы разделяются

- разделителями;
- концами строк;
- знаками форматирования.

Разделитель есть один из следующих специальных символов:

& ( ) * + ? - . / : ; < = > |

Составной разделитель есть композиция двух смежных специальных символов

=> ** := /= >= <= <>

Пример. VHDL-предложение

```
A <= B and C;
```

имеет шесть лексических элементов «A», «<=», «B», «and», «C», «;» .

Два из шести лексических элементов являются разделителем:

« <= » (составной разделитель — оператор назначения сигнала);

« ; »

В качестве разделителей в данном примере используются пробелы, однако нет необходимости иметь разделитель между оператором « ; » и лексическим элементом «C».

Комментарий начинается с двух **смежных** дефисов и продолжается до конца строки. Комментарии не учитываются при моделировании VHDL-описаний и синтезе схем по VHDL-описаниям.

Идентификаторы

Определение.

basic_identifier ::= letter { [underline] letter_or_digit }

Внимание! Здесь и далее формальная запись синтаксических конструкций языка VHDL основывается на формах Бэкуса—Наура, употребление которых разъясняется в Приложении 1. Далее важная информация, на которую следует обратить особое внимание, будет сопровождаться только восклицательным знаком.

Для облегчения понимания конструкций языка VHDL в данной книге будем использовать также *упрощенную форму*: будем записывать общий вид конструкции, в записи конструкции будут ключевые слова и другие символы алфавита языка VHDL, фразы русского языка и символы — фигурные и квадратные скобки, вертикальные черточки. Используемые фигурные скобки, также как и при задании форм Бэкуса—Наура, будут служить для обозначения повторения выражения, заключенного в них; вертикальная черта — для обозначения альтернативных случаев; квадратные скобки — для обозначения необязательного выражения или необязательного слова (см. приложение 1).

! При изучении языка VHDL и написании программ настоятельно рекомендуем пользоваться точными определениями синтаксических конструкций, приводимыми в описаниях соответствующих стандартов [10, 11].

Идентификаторы употребляются как пользовательские имена и ключевые слова.

Идентификатор должен начинаться с буквы (не цифры). Может употребляться кроме букв и цифр, знак подчеркивания. Два подряд идущих подчеркивания не допускаются.

! В VHDL-коде нет различия между прописными и строчными буквами.

AbC7 эквивалентно aBC7,
A_3 не эквивалентно A3.

Идентификатор не должен оканчиваться подчеркиванием.

Пример.

Правильные Идентификаторы	Неправильные Идентификаторы
Carry_OUT	7AB (начинается с цифры)
Dim_Sum	A@B (специальный символ @)
Count7SUB_2goX	SUM_(кончается подчеркиванием)
AaBBb	PI_ _A (два подчеркивания подряд)

Зарезервированные (ключевые) слова

Как и многие другие языки программирования, VHDL имеет *ключевые* (*специальные*) слова.

Список ключевых слов (*стандарт VHDL'87*)

abs	access	after	alias	all
and	architecture	array	assert	attribute
begin	block	body	buffer	bus
case	component	configuration	constant	disconnect
downto	else	elsif	end	entity
exit	file	for	function	generate
generic	guarded	if	in	inout
is	label	library	linkage	loop
map	mod	nand	new	next

Окончание таблицы

nor	not	null	of	on
open	or	others	out	package
port	procedure	process	range	record
register	rem	report	return	select
severity	signal	subtype	then	to
transport	type	units	until	use
variable	wait	when	while	with
xor				

В стандарт VHDL'93 добавлены следующие слова:

group	impure	inertial	literal	postponed
pure	reject	rol	ror	shared
sla	sll	sra	srl	unaffected
xnor				

Литералы

Классификация литералов (**константных числовых** значений) языка VHDL приведена на рис. 1.7.

Десятичный литерал может быть целым, вещественным или вещественным с экспонентой.

Примеры.

Целые литералы: 21, 0, 1E2, 3e4, 123_000.

Вещественные литералы: 11.0, 0.0, 0.468, 3.141_592_6.

Вещественные литералы с экспонентой: 1.23E–11, 1.0E+4, 3.024E+23.

Знак экспоненты E может быть строчным либо прописным. Подчеркивание в десятичном литерале не является значащим. Экспонента для целого литерала не должна иметь знак минус.

Базовый литерал указывает на систему счисления: от двоичной до шестнадцатеричной:

Двоичная	2# 1111_1100#	Все эти литералы имеют значение 252
Шестнадцатеричная	16# fc#	
Шестнадцатеричная	016# 0FC#	
Десятичная	10#252#	
Семеричная	7# 510#	

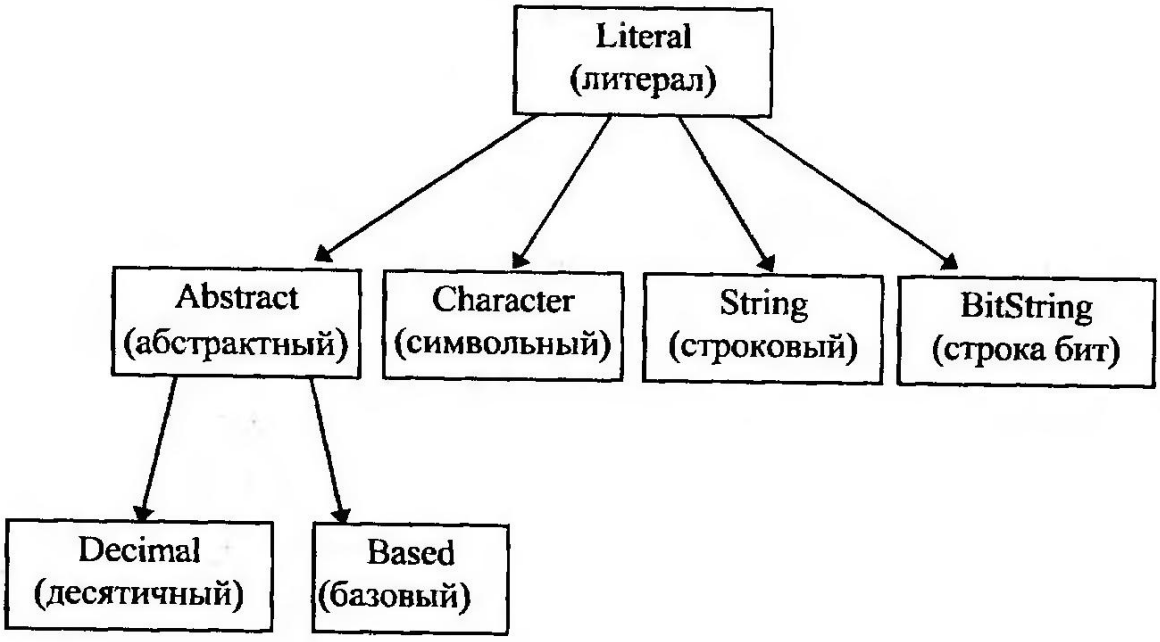


Рис. 1.7. Классификация литералов

Число 16#6#E1 означает 6 * 16 ** 1 — это число 96 в десятичной системе счисления: $96 = 6 \times 16^1$. Символ * — знак умножения, ** — возведение в степень. Число 2E–3 не является целым, 3e4 — это число 3×10^4.

Символьный литерал формируется одним из символов (букв) между апострофами:

'A','a','%', ''' (литерал — апостроф), ' ' (пробел);

'A' отличается от 'a' в случае символьного литерала.

Строковый литерал формируется как последовательность букв (возможно пустая) между двумя кавычками, употребляемыми как строковые скобки.

Строковый литерал должен располагаться в одной строке. Для формирования «длинных» строковых литералов может быть употреблена операция конкатенации & (напомним, что для оператора логического И употребляется оператор AND).

Литерал "строка бит"

Литерал "строка бит" формируется как последовательность цифр 0, ..., 9, A, ..., F (или a, ..., f) между двумя кавычками. Подчеркивание в таком литерале не является значащим.

Литерал "строка бит" может быть

B — бинарным;

O — восьмеричным;

X — шестнадцатеричным.

Очевидно, вместо прописных букв B, O, X могут употребляться строчные буквы b, o, x.

Длина строкового битового литерала есть число бит в последовательности, представляющей литерал. Для примера: все литералы X"F_FF", O"7777", B"1111_1111_1111" имеют длину 12.

Примеры.

```
B"1010110"      -- длина 7
O"126"          -- эквивалентно B"001_010_110", длина 9
X"56"           -- эквивалентно B"0101_0110", длина 8
```

Пакеты

Пакет (package) в VHDL — это VHDL-текст, который может включать множество деклараций:

- типов;
- подтипов;
- констант;
- процедур;
- функций;
- компонент;
- и др.

В теле пакета (package body) находятся тела функций, тела процедур и другие VHDL-описания (рис. 1.8). Однако в пакете не декларируются, а в тело пакета не записываются entity, архитектурные тела (architecture), конфигурации (configuration).

Функции и процедуры иногда называют *подпрограммами*. Пример функции языка VHDL: функция NOW возвращает текущее время системы моделирования. Приведем только пример декларации процедуры, более подробно подпрограммы будут рассмотрены далее.

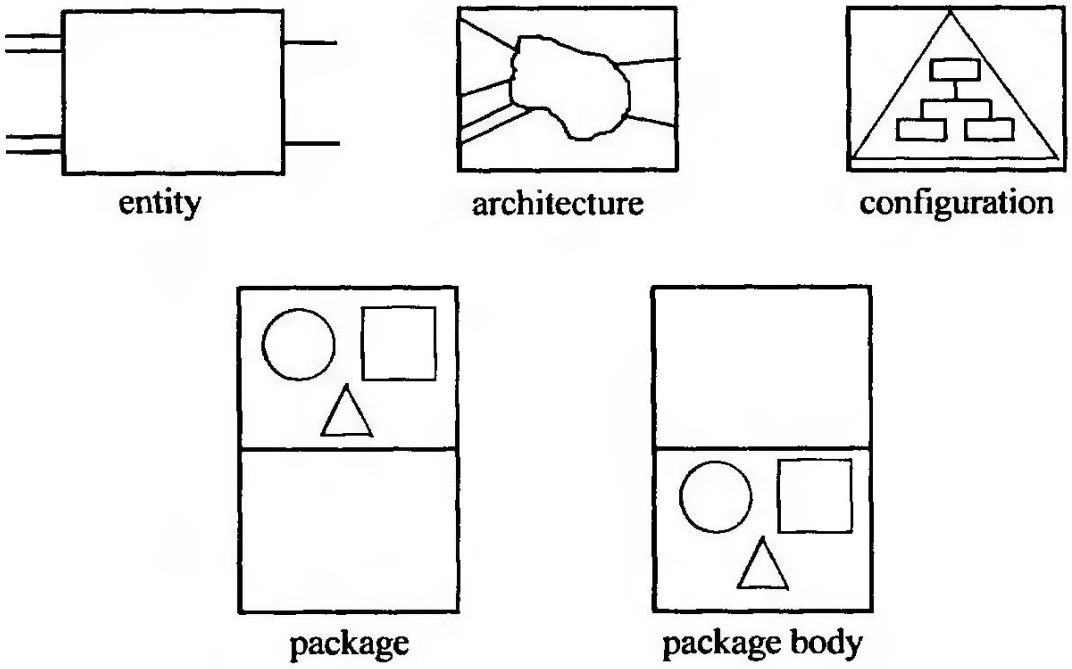


Рис. 1.8. В пакете могут размещаться различные VHDL-описания

Пакет multiplexer показываст, что процедура MX декларируется в пакете и тело процедуры определяется в теле пакета.

```
package multiplexer is
  procedure MX(
    signal SEL : in  bit;
```

```
    signal x0 : in bit;
    signal x1 : in bit;
    signal F : out bit);
end multiplexer;
package body multiplexer is
  procedure MX(
    signal SEL  : in bit;
    signal x0 : in bit;
    signal x1 : in bit;
    signal F : out bit) is
  begin
    case SEL is
      when '0'   => F <= x0;
      when others => F <= x1;
    end case;
  end MX;
end multiplexer;
```

Типы

Большинство языков программирования предусматривает различные типы данных, и язык VHDL не является в этом смысле исключением. Однако, поскольку этот язык используется для представления аппаратных проектов в самых разных вариантах, средства типизации данных приобретают здесь особенно важное значение. Например, они дают разработчику возможность представить группу линий (проводников) шины в виде

- массива битов;
- целого числа.

В языке VHDL реализована *строгая типизация*. Это означает, что смешение различных типов в одной операции является ошибкой. Средства строгого контроля типов играют ответственную роль, поскольку позволяют уточнять намерения разработчика [1].

Тип — поименованное множество значений с некоторыми общими характеристиками.

Подтип — подмножество значений данного типа.

Например, тип NATURAL есть подтип типа INTEGER.

Классификация типов языка VHDL дана на рис. 1.9.

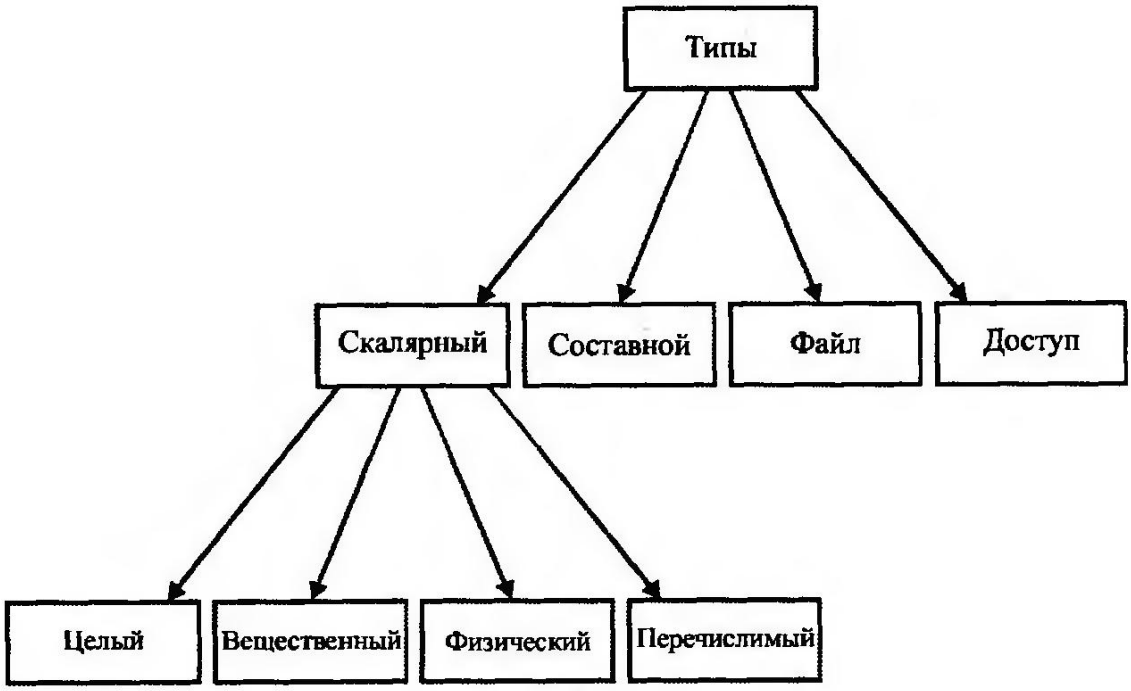


Рис. 1.9. Классификация типов

Типы определяются обычно в

- пакете;
- процессе;
- архитектурном теле.

Тип «целый»

Примеры декларации целых типов.

```
type byte_int is range 0 to 255;
type signed_word_int is range —32768 to 32767;
type bit_index is range 31 downto 0;
```

В табл. 1.3 приведены основные типы данных языка VHDL.

В пакете STANDARD (см. приложение 3), поставляемом с системами моделирования и синтеза, декларируются типы BOOLEAN, BIT, BIT_VECTOR, INTEGER, POSITIV, NATURAL, CHARACTER, STRING, а также тип TIME.

Таблица 1.3

Логические Типы	Арифметические типы	Символьные типы
BOOLEAN (булев)	INTEGER (целый)	CHARACTER (символьный)
BIT (битовый)	POSITIV (положительный)	STRING (строковый)
BIT_VECTOR (битовый вектор)	NATURAL (натуральный)	-
-	REAL (вещественный)	-

Тип **BOOLEAN** состоит из значений TRUE (истина), FALSE (ложь).

Все операторы IF (если) в языке должны проверять объекты или выражения этого типа.

Тип **BIT** состоит из значений 0,1. Значение TRUE типа BOOLEAN не эквивалентно значению 1 типа BIT, аналогично, значение FALSE не эквивалентно значению 0.

Тип **CHARACTER,** по сути дела, представляет набор символов кода ASCII.

Числовые типы имеют диапазон (range), или область значений.

Примеры.

```
type INDEX is range 0 to 9;   -- тип целый,
type VOLTAGE is range 0.0 to 10.0;   -- тип вещественный.
```

Физический тип

Поскольку VHDL — это язык описания аппаратуры, в нем предусмотрены физические типы. Например, тип TIME (время) может быть описан следующим образом:

```
type TIME is range 0 to 1E20
  units
  fs;
  ps        = 1000fs;
```

```
    ns      = 1000ps;
    us      = 1000ns;
    ms      = 1000us;
    s       = 1000ms;
    min     = 60s;
    nr      = 60min;
end units;
```

В качестве базовой выбрана фемтосекунда (10^{-15} с). Диапазон (1E20) обеспечивает максимальный период времени 100 000 с (27,7 ч). Естественно, диапазон ограничивается длиной слова инструментальной машины.

Перечислимые типы

Примеры перечислимых типов.

```
type logic_level is (unknown, low, undriven, high);
type alu_function is (disable, pass, add, subtract, multiply, divide);
type octal_digit is ('0', '1', '2', '3', '4', '5', '6', '7');
```

Определения некоторых перечислимых типов:

```
type severity_level is (note, warning, error, failure);
type boolean is (false, true);
type bit is ('0', '1');
```

Составные типы — это тип «массив» (индексируемый тип) и тип «запись» (структурный тип).

Массивы

Примеры декларации типа «массив».

```
type word is array (31 downto 0) of bit;
```

type memory **is array** (address) **of** word;

type transform **is array** (1 **to** 4, 1 **to** 4) **of** real;

type register_bank **is array** (byte **range** 0 **to** 132) **of** integer;

Тип **BIT_VECTOR** определяет массив битов, например BIT_VECTOR (0 to 7). Тип BIT_VECTOR есть тип массива, который неявно описывается следующим образом:

type BIT_VECTOR **is array** (NATURAL **range** <>) **of** BIT;

BIT — базовый тип элементов массива, выражение NATURAL range (диапазон натуральный) — это стандартный способ указать, что длина массива будет задаваться диапазоном натуральных чисел.

Пользователь должен задавать конкретный диапазон при применении такого типа, например,

BIT_VECTOR (0 to 3) — **возрастающий** диапазон,

BIT_VECTOR (7 downto 0) — **убывающий** диапазон.

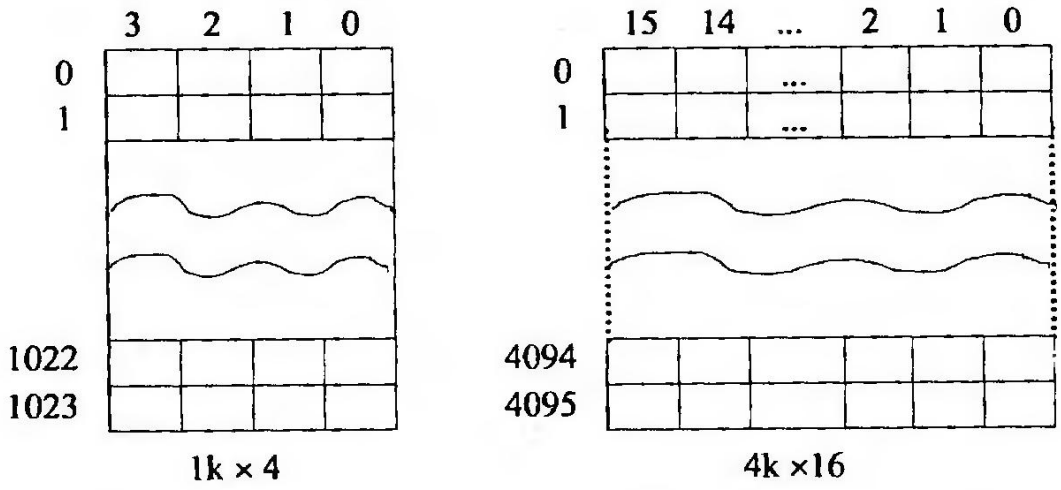
Пример.

signal DataBus: bit_vector (7 downto 0);

1	0	0	1	0	1	0	1	
7	6	5	4	3	2	1	0	номер разряда

DataBus = "10010101"
DataBus(7)='1'
DataBus(6)='0'
DataBus(5)='0'
DataBus(4)='1'
DataBus(3)='0'
DataBus(2)='1'
DataBus(1)='0'
DataBus(0)='1'

Примеры массивов, определяемых пользователем (рис. 1.10).



signal Mem1k4: array(0 to 1023) of bit_vector(3 downto 0);

signal Mem4k16: array(0 to 4095) of bit_vector(15 downto 0);

Рис. 1.10. Примеры массивов, определяемых пользователем

Пример двумерного массива.

Type TWO_DIMENSION **is array** (natural **range** <>, natural **range** <>) **of** bit;

Записи

Тип «запись» — это составной тип, состоящий из ряда полей.

Примеры декларации типа «запись».

```
type instruction is
  record
        op_code : processor_op;
        address_mode : mode;
        operand1, operand2: integer range 0 to 15;
  end record;
```

```
type Instr_T is record
  Mnemonic:      String (0 to 5);
  Code:          Bit_vector (3 downto 0);
  ExeCycles:     Integer;
  end record;
signal Instr1, Instr2, Instr3:        Instr_T;
```

Примеры сигналов типа Instr_T

```
Instr1.Mnemonic:"add reg1, reg2"
Instr1. Code:              "0010"
Instr1. ExeCycles:         2

Instr2.Mnemonic:"or reg1, reg2"
Instr2. Code:              "0001"
Instr2. ExeCycles:         1

Instr3.Mnemonic:"null reg1, reg2"
Instr3. Code:              "1110"
Instr3. ExeCycles:         8
```

Например, тип DATE (дата) можно описать как тип «запись» следующим образом:

```
type DATE is
  record
  DAY: INTEGER range 1 to 31;
  MONTH: MONTH_TYPE;
  YEAR:INTEGER range 0 to 3000;
end record;
```

Здесь MONTH_TYPE (имя месяца) — это перечислимый тип, содержащий имена месяцев. Определение данного перечислимого типа содержится в разд. 2.1 (см. описание оператора case).

VHDL предусматривает (см. рис. 1.9) определенные типы *файлов* (FILE) и типы *доступа* (ACCSESS). В данной книге они не рассматриваются.

Подтипы, конверсия типов

Подтип определяется как тип с ограничением (уточнением) следующим образом

subtype имя подтипа **is** базовый тип [уточнение];

Например, пусть базовый тип определен так

```
Type big_integer is range 0 to 2000;
```

Тогда подтип small_integer может быть определен так

```
Subtype small_integer is big_integer range 0 to 15;
```

Смешение типов в VHDL — это ошибка. Однако значения подтипа могут быть переданы типу (конверсия типов).

Пример.

```
Type big_integer is range 0 to 2000;
Subtype small_integer is big_integer range 0 to 15;
Signal intermediate : small_integer;
Signal final : big_integer;
. . .
Final <= intermediate * 5;  -- нет ошибки, конверсия типов
```

В данном фрагменте VHDL-кода нет ошибки, так как small_integer есть подтип базового типа big_integer.

Если же типы определяются независимо друг от друга, то присвоение значений одного типа другому невозможно, т. е. приводит к ошибке.

Пример.

```
Type big_integer is range 0 to 2000;
Type small_integer is range 0 to 15;
Signal intermediate : small_integer; Signal final : big_integer;
. . .
    Final <= intermediate * 5; -- ошибка, смешение типов
```

1.3. Декларации

Имеются три класса *объектов языка* VHDL:

- константы;
- сигналы;
- переменные.

С помощью объектов языка VHDL описываются объекты проекта, т. е. различные подсистемы проектируемой (моделируемой) цифровой системы. Поэтому, естественно, следует отличать объекты проекта от объектов языка VHDL

Декларация константы.

Общий вид конструкции, употребляемой при декларации констант одного и того же типа

constant список констант **:** тип [**:=** выражение] **;**

В VHDL *константы* подобны константам в других языках программирования.

Примеры декларации констант.

```
constant PERIOD:        time:= 100 ns;
constant PI:            real:= 3.14159;
constant WIDTH:         integer:= 32;
constant DEFAULT: bit_vector(0 to 3):= "0101";
constant a, b, c, d : integer := 5;
```

Каждая из декларированных в последней строке констант a, b, c, d равна 5.

Если символов «:=» нет в декларации константы, то константа называется *задержанной.* Такие константы могут быть в разделе деклараций пакета. Соответствующая полная декларация появляется в теле пакета.

Если в VHDL-коде символы «:=» следуют после выражения, то выражение принимает значение объекта, находящегося справа от данных символов, при этом, естественно, не должно быть смешения типов.

Декларация переменной

Общий вид конструкции, употребляемой при декларации переменных

variable список переменных **:** тип **[:=** выражение**];**

Переменная в VHDL подобна идентификатору, употребляемому в других языках программирования высокого уровня. Значение переменной может быть инициализировано и изменено ***немедленно*** после выполнения предложения, присваивающего переменной новое значение.

Примеры декларации переменных.

```
variable ROM, COLUMN:        integer range 0 to 31;
variable COUNT        :      positive := 100;
variable MEMORY :    TWO_DIMENSION (0 to 15, 0 to 31);
variable a, b, c, d: bit := '1';
```

Каждая из декларированных в последней строке битовых переменных a, b, c, d равна '1'.

Декларация сигнала

Общий вид конструкции, употребляемой при декларации сигналов одного типа

signal список сигналов **:** тип **[register | bus] [:=** выражение**];**

В необязательной опции **[register | bus]** может быть указано только одно из ключевых слов — либо слово **register,** либо слово **bus.**

Концепция *сигнала* в VHDL является одной из самых важных. Сигналы подобны (соответствуют) физическим линиям (проводникам), которые соединяют элементы схемы. Сигналы и переменные будут рассмотрены далее.

Примеры декларации сигналов.

```
signal CLK, RESETn:     bit;
signal COUNTER:         integer range 0 to 31;
signal RAM :      TWO_DIMENSION (0 to 15, 0 to 31);
signal INSTRUCTION:   bit_vector (15 downto 0);
signal a, b, c, d: bit := '0';
```

Как видно из последней строки примера, сигналам при декларации может быть присвоено начальное значение.

Декларация компонента

Общий вид конструкции, употребляемой при декларации компонента

```
component имя компонента
        generic (список параметров);
        port (список портов);
end component;
```

Список портов, начинающийся с ключевого слова port, определяет имя, направление (режим — mode) и тип каждого порта. Тип порта — это тип сигнала, ассоциированного с данным портом. *Направление порта* может быть: **in** — входной порт, **out** — выходной, **inout** — двунаправленный, **linkage, buffer**. Компоненты, имеющие направления портов linkage или buffer, в данной книге не будут рассматриваться. Компоненты декларируются в архитектурном теле прежде оператора begin. Ключевое слово **generic** служит для передачи параметров, чаще всего таких, как разрядность, число полюсов и других *настраиваемых* параметров. Примеры настраиваемых параметров будут даны позже. Компонент является частью проекта. Его реализация осуществляется соответствующим entity и одним либо несколькими архитектурными телами.

! В entity и декларируемом компоненте должны совпадать спецификации.

1.4. Интерфейс и архитектура объекта

Понятие *entity* определяется как «интерфейс объекта проекта». В *entity* описывается интерфейс между объектом проекта и окружением, в котором употребляется объект. «Внутренность» объекта в entity не описывается и может быть уподоблена «черному ящику».

Термин «архитектура» безотносительно к языку VHDL может быть определен как распределение функций, реализуемых системой, по отдельным ее уровням и точное определение границ между этими уровнями.

К термину «архитектура» близок термин «структура», понимаемый как совокупность элементов системы и связей между ними.

Архитектура — это структура системы на функциональном уровне ее описания.

Архитектурное тело (**architecture**) определяет тело объекта, т. е. раскрывает внутренность «черного ящика». В архитектурном теле описываются функции (поведение) либо структура объекта проекта. В приведенном ниже примере архитектурные тела RTL1, RTL2 задают поведение схемы, представленной на рис. 1.11. Архитектурные тела, в данном примере это RTL1, RTL2, могут быть различными, задавая одно и то же поведение.

Пример.

```
entity ANDOR is
        port (x1, x2, x3 :      in bit;
              f          :      out bit);
end ANDOR;

architecture RTL1 of ANDOR is
begin
  f <= (x1 and x2) or x3;
end RTL1;
architecture RTL2 of ANDOR is
  signal w : bit;
begin
  w <= x1 and x2;
  p1 : process (w, x3)
```

```
  begin
    f <= w or x3;
  end process p1;
end RTL2;
```

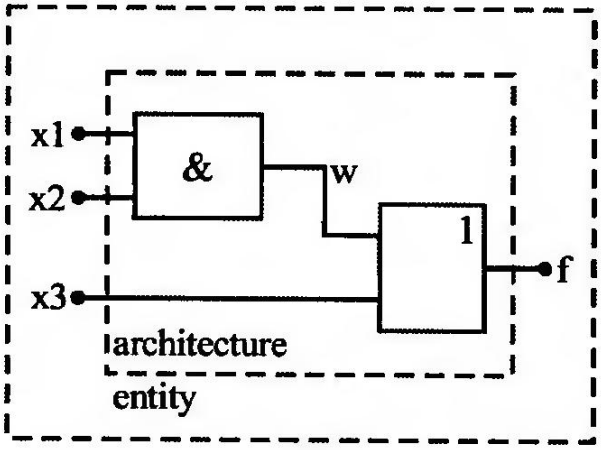


Рис. 1.11. Комбинационная логическая схема

В entity (в разделе деклараций) наряду с декларацией портов могут быть декларированы параметры. Для декларации таких параметров употребляется ключевое слово generic (настраиваемый). С помощью generic могут передаваться такие параметры, как ширина (разрядность) шины, число входных либо выходных полюсов, задержки элементов и т. д.

Пример декларации объекта с настраиваемым (изменяемым) параметром N.

```
entity Cate is
  generic
          (N:       Natural:= 4);
  port
          (Inputs: in bit_vector ( 1 to N);
           Result: out bit);
end Cate;
```

Изменяя параметр N, можно получать объект Cate с различным числом входов. При этом вносить другие изменения в VHDL-код нет необходимости.

Пример «минимальной» декларации объекта

```
entity TestBench is
end TestBench;
```

В гл.4. будет показано, что минимальная декларация объекта проекта используется при отладке VHDL-описания.

Пример использования generic для описания объекта Processor с переменной шириной шины (рис.1.12).

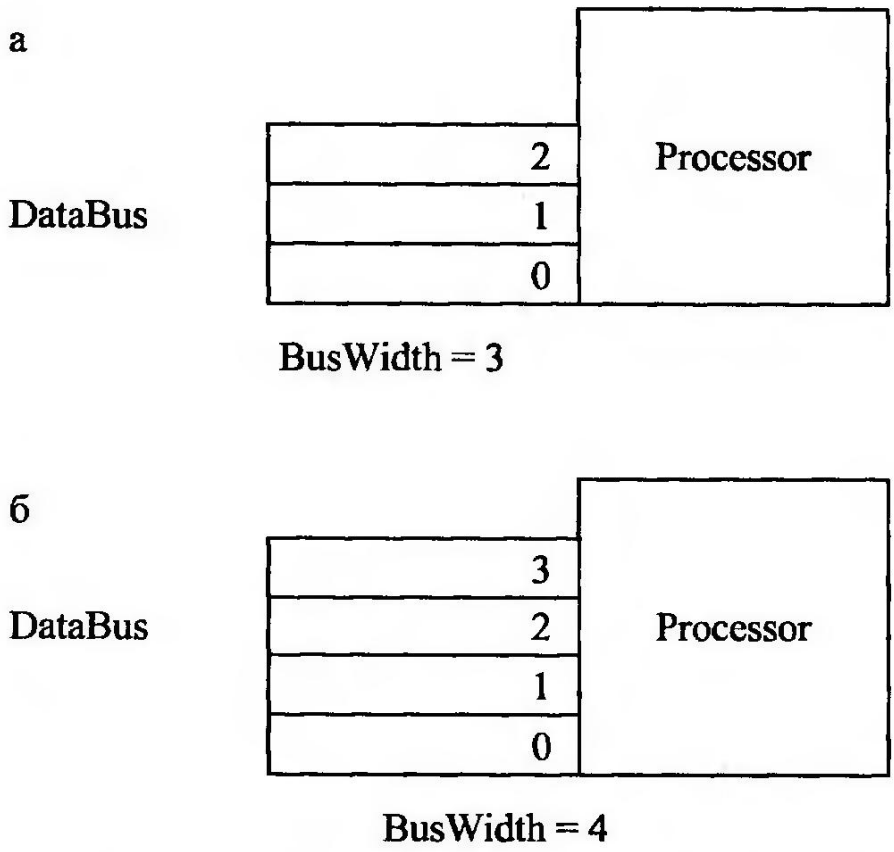


Рис. 1.12. Объект Processor с переменной шириной шины:
а — ширина шины равна 3; б — ширина шины равна 4

```
entity Processor is
  generic (BusWidth: Integer:= 3);
  port (DataBus: inout bit_vector (BusWidth –1 downto 0);
            . . . );
  end Processor;
```

1.5. Атрибуты

Атрибуты — это значения, связанные с поименованным элементом — объектом языка VHDL.

В VHDL имеются *предопределенные* (*predefined*) атрибуты и *определенные пользователем* (*user-defined*) атрибуты. Для построения моделей и моделирования важную роль играют атрибуты сигналов. Например, предопределенный атрибут **event** ассоциируется с каким-либо сигналом (например, с сигналом CLK). Атрибут записывается CLK**'event**. Этот атрибут имеет тип BOOLEAN с значением TRUE, когда значение CLK изменилось.

Атрибут S**'last_value** (прошлое значение S) — предыдущее значение, которое сигнал имел непосредственно перед последним изменением S. Относится к тому же самому типу, что и S.

Атрибут S**'transaction** имеет тип bit, атрибут изменяет свое значение в циклах моделирования, в которых происходит изменение (транзакция) S.

Атрибут S**'stable(T)** имеет тип BOOLEAN. Атрибут имеет истинное значение, если сигнал S стабилен в течение последних T единиц времени. Если T = 0, атрибут записывается S**'stable.**

Атрибуты сигналов, в том числе и такие, как S**'delayed**(t), S**'quiet**(t), S**'transaction**, S**'active**(t), S**'last_event**, S**'driving**, S**'driving_value** используются главным образом при моделировании.

Пример. На рис. 1.13 показано, как изменяются значения атрибутов ex**'transaction**, ex**'event**, ex**'last_value** при изменении сигнала ex.

Временная диаграмма (рис. 1.13) получена в результате моделирования следующего VHDL-кода.

```
entity signal_ex is
end signal_ex;
architecture beh of signal_ex is
signal ex, y1, y3 : bit;
signal y2 : boolean;
begin
ex <= '0' after 20 ns,
      '1' after 50 ns,
      '0' after 60 ns,
      '1' after 80 ns;
y1 <= ex'transaction; y2 <= ex'event; y3 <= ex'last_value;
end beh;
```

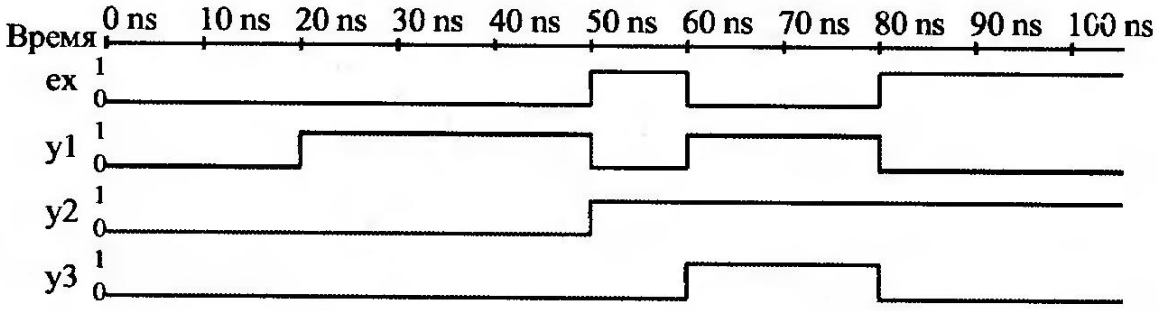


Рис. 1.13. Изменение сигнала ex и его атрибутов

На временной диаграмме (рис. 1.13) значение true сигнала y2 представляется единичным значением, значение false — нулевым значением. Рассмотрим на временной диаграмме (рис. 1.13) изменение значения y1 — атрибута 'transaction сигнала ex. В момент времени 20ns произошла транзакция сигнала ex (назначение сигнала '0' after 20 ns), поэтому атрибут изменил свое значение, следующее изменение произошло в момент времени 50 ns, т. е. тогда, когда произошло следующее изменение сигнала ex, и т. д. Атрибут 'event (сигнал y2) один раз изменил свое значение, а именно, после того, как первый раз изменилось значение сигнала ex. Атрибут 'last_value (сигнал y3) показывает предыдущее значение сигнала ex.

При моделировании и синтезе схем большее значение имеют следующие предопределенные атрибуты типа (табл. 1.4).

Таблица 1.4

Предопределенные атрибуты		
Атрибут объекта S	Тип атрибута	Результат
S'left	такой, как S	крайнее левое значение в S
S'right	– " –	крайнее правое значение в S
S'low	– " –	нижняя граница
S'high	– " –	верхняя граница
S'ascending	boolean	true, если S возрастающего диапа-зона, false — в противном случае
S'image(x)	string	текстовое представление значения x типа S
S'value(t)	базовый тип S	значение в S, представленное строкой t
S'pos(x)	integer	номер позиции объекта x в S
S'val(x)	базовый тип S	значение в позиции x в S
S'succ(x)	базовый тип S	значение в позиции на единицу боль-ше, чем x
S'pred	базовый тип S	значение в позиции на единицу мень-ше, чем x
S'leftof(x)	базовый тип S	значение левой от x позиции в S
S'rightof(x)	базовый тип S	значение правой от x позиции в S
S'base	базовый тип S	базовый тип S

С использованием атрибута **'high** в пакете STANDARD опре-деляются подтипы NATURAL, POSITIVE

subtype natural **is** integer **range** 0 **to** integer'high;

subtype positive **is** integer **range** 1 **to** integer'high;

Полезный набор атрибутов относится к массивам (табл.1.5).

Атрибуты A**'left(N)**, A**'right(N)**, A**'high(N)**, A**'low(N)** ассоции-руются с массивом объектов или конструируемым массивом подти-пов. Атрибуты возвращают значение N-го объекта в массиве. Пара-метр N является необязательным и по умолчанию равен 1. Атрибут TS**'base** возвращает базовый тип TS.

Таблица 1.5

Атрибуты типа «массив»

Атрибут	Результат
S'left	левая граница S
S'right	правая граница S
S'low	нижняя граница S
S'high	верхняя граница S
S'range(n)	диапазон индексов n-й размерности
S'reverse_range(n)	перевернутый по направлению и границам диапазон индексов n-й размерности
S'lenght(n)	длина диапазона индексов размерности n
S'ascending (n)	true, если S возрастающего диапазона, false — в противном случае

Пример.

Пусть тип type A_BYTE есть массив бит убывающего диапазона (7 **downto** 0). Тогда

A_BYTE'**left** = A_BYTE'**high** = 7;
A_BYTE'**low** = A_BYTE'**right** = 0;

В случае

type table **is array** (1 **to** 8) **of** bit;
variable array_1 : table := "00001111";

значением атрибута array_1'**left** является 1 — левая граница возрастающего диапазона.

Примеры.

1. Пусть

Type TWO_DIMENSION **is array** (natural **range** <>, natural **range** <>) **of** bit;

signal RAM: TWO_DIMENSION (0 **to** 10, 0 **to** 15);

Тогда RAM'**left**(2) = RAM'**low**(2) = RAM'**low**(1) = RAM'**low** = = RAM'**left** = RAM'**left**(1) = 0;
RAM'**high**(1) = RAM'**right**(1) = 10;
RAM'**high**(2) = RAM'**right**(2) = 15;

2. Пусть

Type new_values **is** (low, high, middle);

Тогда значением атрибута new_values'**pred** (high) является 'low'.

3. Ниже приведен тест для определения значений атрибутов типа vector, декларированного

type vector **is array** (7 **downto** 0) **of** bit;

как массив бит убывающего диапазона.

```
entity test_attr_vector is
end test_attr_vector;
architecture beh of test_attr_vector is
type vector is array (7 downto 0) of bit;
signal x : vector;
signal A, B, C, D : integer;
signal E : boolean;
signal F, G, H : integer;
begin
A <= vector'left;
B <= vector'right;
C <= vector'low;
D <= vector'high; E <= vector'ascending;
F <= vector'range;
G <= vector'reverse_range;
H <= vector'length; end beh;
```

В результате моделирования A = 7, B = 0, C = 0, D = 7, E = false, F = 7, G = 0, H = 8.

4. Ниже приведен тест для определения значений атрибутов перечислимого типа, состоящего из элементов a1, b1, a2, b2, a3, b3, w.

```
entity test_attr_scalar is
end test_attr_scalar;
architecture beh of test_attr_scalar is
type new_values is (a1, b1, a2, b2, a3, b3, w);
signal A, B, C, D : new_values;
signal H : integer;
signal K, L, M, N, P : new_values;
begin
A <= new_values'left;
B <= new_values'right;
C <= new_values'low;
D <= new_values'high;
H <= new_values'pos(b3);
K <= new_values'val(2);
-- L <= new_values'succ(w); -- bad
L <= new_values'succ(b3);
M <= new_values'pred(b2);
-- N <= new_values'leftof(a1); -- bad
N <= new_values'leftof(b1);
-- P <= new_values'rightof(w); -- bad
P <= new_values'rightof(b3);
end beh;
```

В строках, оформленных как комментарии, содержатся примеры ошибок: неправомерно пытаться определить значение атрибута 'leftof для элемента a1 (назначение сигнала N). Также неправомерно «заходить» за правую границу перечислимого типа, т. е. пытаться узнать значения атрибутов 'rightof, 'succ для элемента w (назначение сигналов L, P).

В результате моделирования A = a1, B = w, C = a1, D = w, H = 5, K = a2, L = w, M = a2, N = a1, P = w.

В некоторых системах моделирования реализация атрибутов может быть иная — в таких системах можно «заходить» за границы перечислимых типов.

Атрибуты, определенные пользователем

Если атрибут определяется пользователем, то сначала декларируется его тип, а затем определяется значение. Значение может быть сигналом либо другим объектом языка VHDL. Определенный пользователем атрибут при употреблении предваряется знаком ' (апостроф). Пример определенного пользователем атрибута **'two_length**

```
signal vect : bit_vector (0 to 5);
attribute two_length : integer;
attribute two_length of vect : signal is (vect'length) * 2;
. . .
C <= vect'two_length;
```

Сигнал C типа integer получит значение 12, так как длина vect'length массива vect равна 6.

Атрибуты, определяемые пользователями, могут быть введены не только для сигналов, но и для меток, компонент и т. д. [7].

1.6. Имена

Имена употребляются для обозначения декларируемых объектов.

Простые имена являются идентификаторами, такими как COUNTER, CLK.

Индексированные имена употребляются для обозначения элементов массива, например, RAM(1,3) — элемент двумерного массива, INSTRUCTION(5) — элемент одномерного массива.

Интервал имен одномерного массива: INSTRUCTION (7 **downto** 0).

Выборка имен используется для выбора полей записи или считывания, а также для выбора элемента библиотеки.

Пример.

DATE1.MONTH обозначает имя элемента, содержащегося в поле MONTH записи DATE.

WORK.DESIGN может обозначать объект проекта DESIGN в библиотеке WORK.

Имена атрибутов могут употребляться для обозначения предопределенных атрибутов или определенных пользователем атрибутов.

1.7. Операторы

Набор операторов (операций) в VHDL обеспечивает возможность работы с предусмотренными типами данных.

Список операций приведен в табл. 1.6.

Строки табл. 1.7 располагаются в порядке старшинства (от низшего к высшему) операторов. Операторы, находящиеся в одной строке, обладают одинаковым старшинством (приоритетом). Таким образом, операции нижней строки табл. 1.7 обладают наибольшим приоритетом и выполняются первыми, в частности, логический оператор not выполняется прежде других логических операторов.

Исходя из контекста VHDL-кода следует отличать оператор <= (назначение сигнала) и оператор <= (меньше либо равно). Следует также отличать унарные операции присвоения знака +, – от соответствующих бинарных операций сложения и вычитания.

Замечание. Для устранения неоднозначностей трактовки старшинства операций используются скобки.

Например, выражение «A nand B nand C» неверно (синтаксическая ошибка). Данное выражение не представляет трехвходовый элемент И-НЕ (трехвходовую NAND-ячейку). Правильная запись «not (A and B and C)». Запись «A nand (B nand C)» не есть то же самое, что «(A nand B) nand C».

Таблица 1.6

Операции языка VHDL

Обозначение	Название
not	логическое НЕ
and	логическое И
or	логическое ИЛИ
nand	логическое И-НЕ
nor	логическое ИЛИ-НЕ
xor	исключающее ИЛИ
xnor	эквивалентность (VHDL'93)
=	равно
/=	не равно
<	меньше
<=	меньше либо равно
>	больше
>=	больше либо равно
+	сложение, присвоение знака +
-	вычитание, присвоение знака -
&	конкатенация
*	умножение
/	деление
mod	модуль
rem	остаток
**	возведение в степень
abs	абсолютное значение

Таблица 1.7

Классификация операций

Класс операций	Операции					
Логические	and	or	nand	nor	xor	xnor (VHDL'93)
Сравнения	=	/=	<	<=	>	>=
Сложения и конкатенации	+	–	&			
Присвоение знака	+	–				
Умножения	*	/	mod	rem		
Смешанные	**	abs	not			

Примеры арифметических операторов.

Сложение (+)

```
RealX2 + 2.0     -- если RealX2 есть сигнал типа
                 -- вещественный, то число 2 должно
                 -- быть записано как вещественное
1us + 3ns        -- 1003ns
```

Вычитание (–)

```
8.33 – 5         --  неправильно, оба числа должны
                 --  быть одного типа
BusWidth – 1     --  допустимо, если BusWidth типа integer
```

Умножение (*)

```
4*SomeVal        --  допустимо, если SomeVal
                 --  типа Integer или Time
Mult*5ns         --  результат типа Integer либо Real
                 --  в зависимости от типа Mult
```

Деление (/)

```
CLK/2            --  тип CLK — Integer
5.0/2.0          --  результат — вещественное число 2.5
10ns/2ns         --  результат 5 типа Integer, но не Time
```

Модуль (mod)

6 mod 4	--	результат 2
6 **mod** (–4)	--	результат –2
(–6) **mod** 4	--	результат 2

Остаток деления (rem)

6 **rem** 4	--	результат 2
6 **rem** (–4)	--	результат 2
(-6) **rem** 4	--	результат –2

Экспонента (**)

c ** 0.5	--	неправильно в VHDL
A ** 2	--	эквивалентно A * A
B ** 3	--	эквивалентно B * B * B

Абсолютное значение (abs)

abs 1	--	результат 1
abs(–1)	--	результат 1
abs(5 * (–2))	--	результат 10

В стандарте [10] языка VHDL операции **rem** и **mod** определяются следующим образом.

Для операции L **rem** R должно выполняться соотношение

$$L = (L / R) * R + (L \textbf{ rem } R),$$

где L/R – целая часть частного, (L **rem** R) — результат выполнения операции **rem** (остаток). Остаток имеет *знак операнда L*.

Для операции L **mod** R должно выполняться соотношение

$$L = N * R + (L \textbf{ mod } R),$$

где N — некоторое целое число, (L **mod** R) — результат выполнения операции **mod**. Результат имеет *знак операнда R*.

Пример выполнения операций **rem**, **mod** языка VHDL:

13 **rem** 5 = 3,	13 **mod** 5 = 3;
13 **rem** (–5) = 3,	13 **mod** (–5) = –2;
(–13) **rem** 5 = –3,	(-13) **mod** 5 = 2;
(–13) **rem** (–5) = –3,	(-13) **mod** (–5) = –3.

Следует быть внимательным при переходе от битовых векторов к числам:

Bit_vector (0 **to** 7); -- возрастающий -- диапазон

1	0	1	1	1	1	0	0
0	1	2	3	4	5	6	7

Число **61** (десятичное), старший разряд справа

Bit_vector (7 **downto** 0); -- убывающий -- диапазон

1	0	1	1	1	1	0	0
7	6	5	4	3	2	1	0

Число **188** (десятичное), старший разряд слева

Логические операторы выполняются для следующих типов данных:

- boolean;
- bit, bit_vector;
- std_logic, std_logic_vector;
- std_ulogic, std_ulogic_vector.

Логические операторы **and, or, xor** имеют одинаковое старшинство и выполняются слева направо в выражениях. Оператор **not** имеет более высокое старшинство и выполняется прежде других операторов. В сложных логических выражениях порядок выполнения операторов регулируется скобками. Рекомендуем читателю применять скобки в затруднительных случаях. Например, для выражения

Z <= A **and not** B **or** C;

будет только отрицание B, в выражении

Z <= A **and not** (B **or** C);

будет отрицание подвыражения B **or** C, находящегося в скобках.

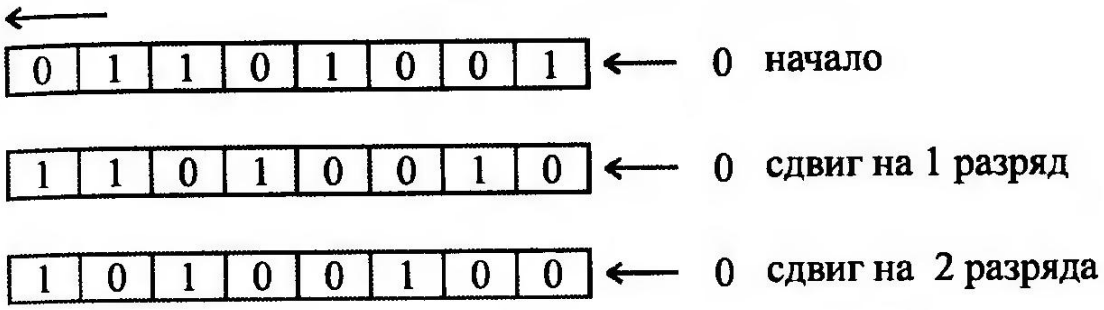
Операторы сдвига

Данные операторы поясним на примере семиразрядного битового вектора MyBus.

signal MyBus: bit_vector(7 **downto** 0) := "01101001";

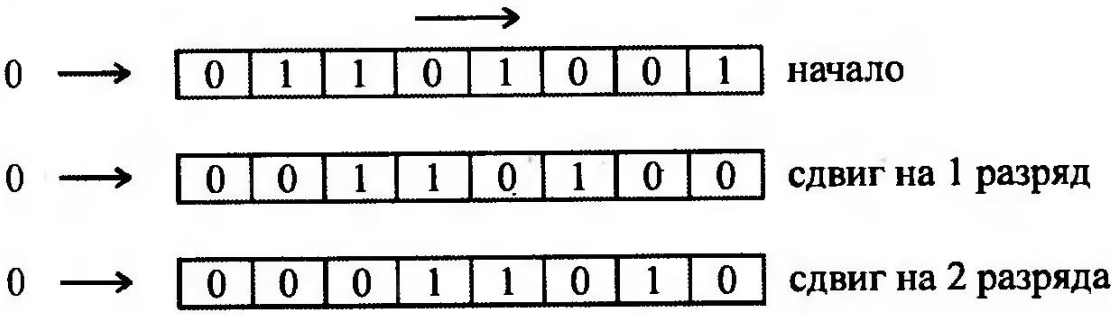
Логический сдвиг влево (Shift Left Logical)

оператор **sll**



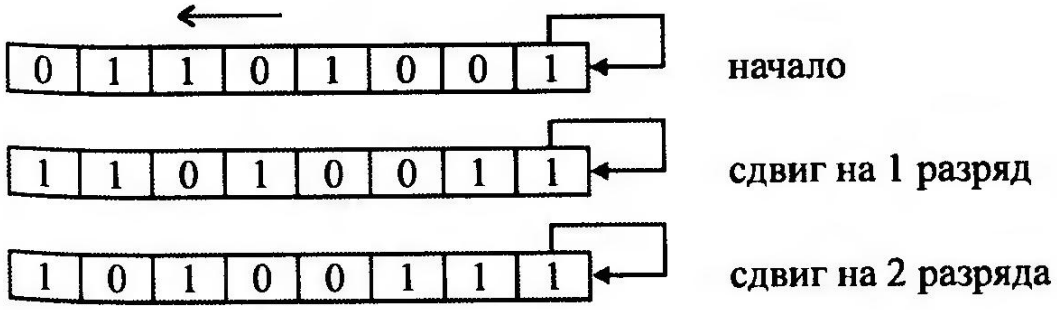
Логический сдвиг вправо (Shift Right Logical)

оператор **srl**



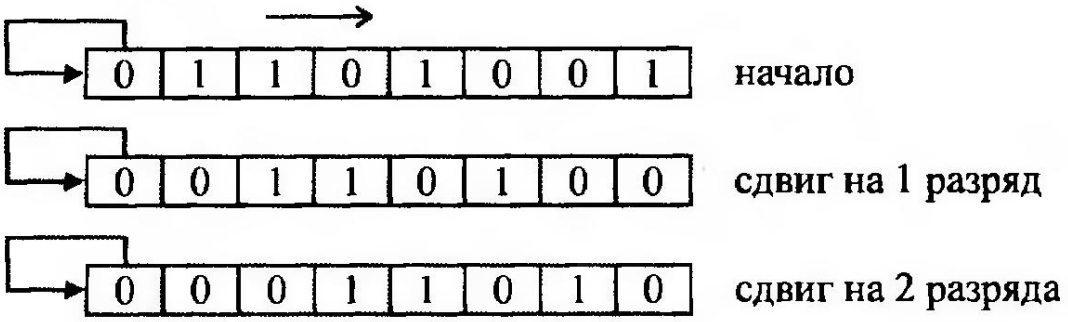
Арифметический сдвиг влево (Shift Left Arithmetic)

оператор **sla**



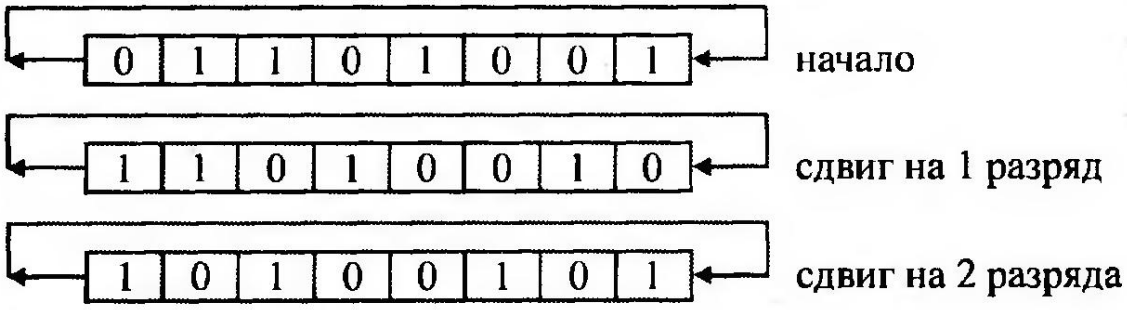
Арифметический сдвиг вправо (Shift Right Arithmetic)

оператор **sra**



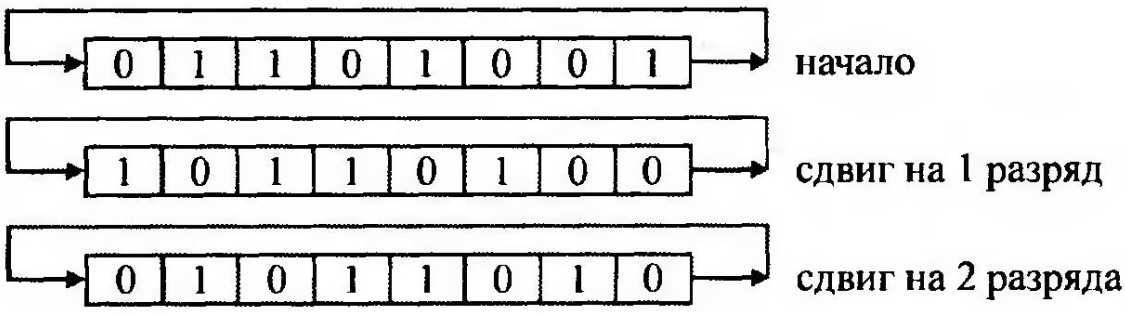
Вращение логическое влево (Rotate Left Logical)

оператор **rol**



Вращение логическое вправо (Rotate Right Logical)

оператор **ror**



Обычно операторы сдвига реализуются в виде функций, пример вызова такой функции см. в разд. 3.9.

! Обратите внимание на реализацию операторов сдвига. В некоторых системах моделирования и синтеза данные операторы могут быть реализованы в виде функций.

Оператор конкатенации

Оператор конкатенации обозначается через &.

Пример.

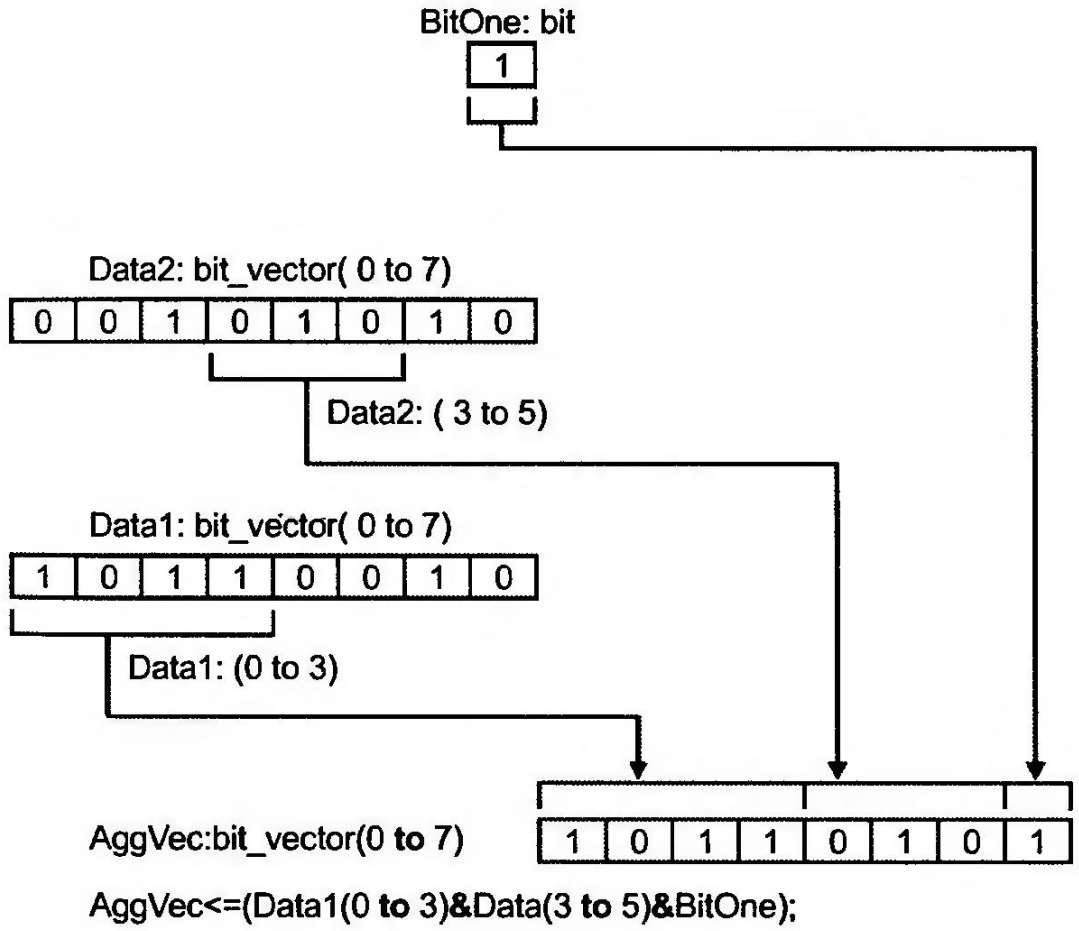


Рис. 1.14. Операция конкатенации

1.8. Понятие сигнала в языке VHDL

Логические сигналы в логических схемах передаются и обрабатываются параллельно.

На схеме показаны три логических блока [1].

Если предположить, что входной набор 1 (для логического блока 1) и входной набор 2 (для логического блока 2) активизируются одновременно, то логические блоки 1, 2 будут активизироваться также одновременно. Логический блок 3 будет активизирован, а через блоки 1, 2 могут проходить (конвейерным способом) измененные значения сигналов других входных наборов. Таким образом, поток сигналов может проходить через все блоки ***одновременно***.

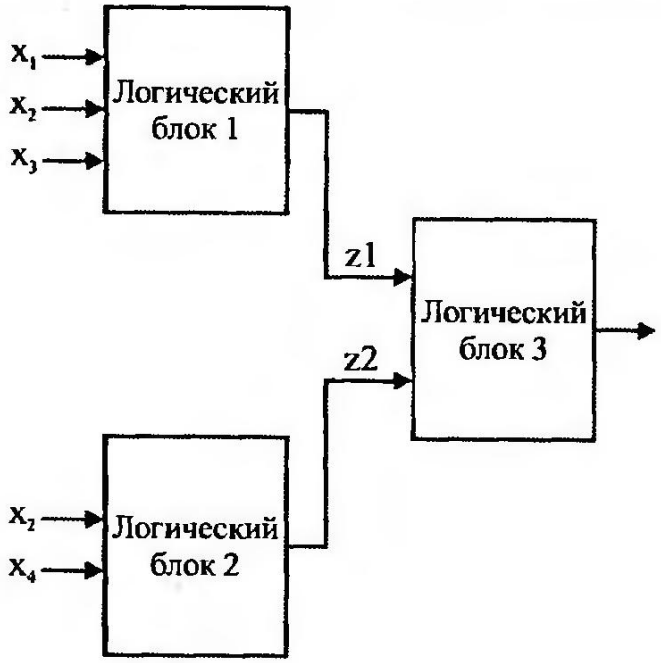


Рис. 1.15. Логическому блоку схемы соответствует процесс языка VHDL

Язык описания аппаратуры должен иметь средства для отображения такой одновременности. В языке VHDL это требование реализуется при помощи механизма процессов. Каждый *процесс* представляет некоторый логический блок моделируемой схемы, причем все процессы выполняются ***параллельно*** (во времени).

Три параллельных процесса для схемы (рис. 1.15) могут быть описаны следующим образом.

```
BLOCK1: process(X1, X2, X3)   -- X1, X2, X3 список
                              -- чувствительности блока 1
```

```
        begin
                -- операторы процесса BLOCK1
        end process BLOCK1;
BLOCK2: process(X2, X4)
                -- операторы процесса BLOCK2
        end process BLOCK2;
BLOCK3: process(Z1, Z2)
                -- операторы процесса BLOCK3
        end process BLOCK3;
```

В языке VHDL процесс активизируется, когда происходит изменение какого-либо сигнала в списке сигналов запуска этого процесса.

Список *сигналов запуска* — то же самое, что и *список чувствительности* (sensitivity list). В общем случае список сигналов запуска содержит входной набор сигналов соответствующего логического блока.

Рассмотрим схему (рис. 1.16).

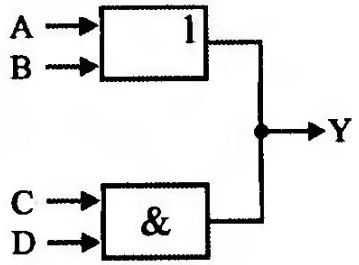


Рис. 1.16. Сигнал Y, имеющий два источника

Сигнал У ведет к двум источникам. Каждый источник сигнала имеет свой *драйвер*. Драйвер — это понятие языка VHDL, связанное моделированием. Драйвер можно представлять себе как контейнер [1], т. е. как какое-то хранилище — область памяти для хранения значения сигнала. Значения сигнала в данном хранилище изменяет только система моделирования во время процесса моделирования VHDL-кода. Моделирование является *событийным*. Событием яв-

ляется изменение какого-либо сигнала. Моделирование в любой временной точке осуществляется в два этапа: сначала изменяются драйверы сигналов, затем значения сигналов выдаются «наружу», т. е. туда, где они могут наблюдаться, например, на временных диаграммах.

! Можно определять только по одному драйверу на сигнал в процессе.

Так как в каждом процессе сигнал должен иметь *только один* источник, то, в случае, если имеется несколько источников сигнала, требуется специальная (разрешающая) функция, которая будет вычислять значение сигнала по значениям этого сигнала из нескольких драйверов. Например, сигнал Y (рис. 1.16) имеет два источника, и, соответственно, два драйвера. Для такого сигнала требуется разрешающая функция. Подробнее эта проблема будет рассмотрена далее (разд. 3.4).

Каждый процесс может быть в одном из трех состояний:

- выполняющийся;
- активный;
- приостановленный.

Выполняющийся — когда система моделирования выполняет процесс.

Активный — когда процесс является ожидающим, чтобы система моделирования его выполнила.

Приостановленный — когда он не является выполняющимся или активным.

Большинство ЭВМ, на которых работают системы моделирования, являются однопроцессорными. Параллелизм программно имитируется. Поэтому только один процесс из нескольких параллельных процессов является «выполняющимся» по существу. Реализация параллельных процессов происходит следующим образом.

Параллельные процессы, которые надо «одновременно» выполнять, выстраиваются в очередь. Система моделирования выбирает процесс из очереди активных процессов и выполняет процесс, тем самым исполняет предложения языка VHDL, относящиеся к данному процессу. Другие активные процессы выбираются поочередно.

Заметим, что выполненный процесс является приостановленным. Когда очередь активных процессов пуста, считается, что все параллельные процессы выполнились «одновременно», и может начаться следующий цикл моделирования. Приостановленный процесс может стать активным, когда изменился хотя бы один из драйверов сигналов из списка чувствительности этого процесса.

Рассмотрим простую двухуровневую логическую схему (см. рис. 1.11) и ее описание в языке VHDL в виде параллельных процессов.

```
entity ANDOR is
        port(   x1, x2, x3 :    in bit;
                        f :     out bit);
end ANDOR;

architecture example of ANDOR is
  signal w : bit;
begin
  p0 : w <= x1 and x2 after 10 ns;
  p1 : process (w, x3)
  begin
    f <= w or x3 after 20 ns;
  end process p1;
end example;
```

Во время фазы инициализации каждый сигнал из множества {x1, x2, x3} имеет значение 0. В процессе p0 будет вычислено w и процесс p0 приостановится. Затем вычисляется значение f через 30ns, так как 10ns выполняется процесс p0 и 20ns — процесс p1. Цикл моделирования начинается, когда один из сигналов изменится.

Временная диаграмма изображена на рис. 1.17. Заметим, что если сигнал x1 при декларации не имеет начального значения, то по умолчанию он принимает значение левой границы (x1'left) декларируемого типа. Например,

```
signal I : integer range 0 to 3;      -- при инициализации значение 0;
signal X : std_logic;                 -- при инициализации значение 'U'.
```

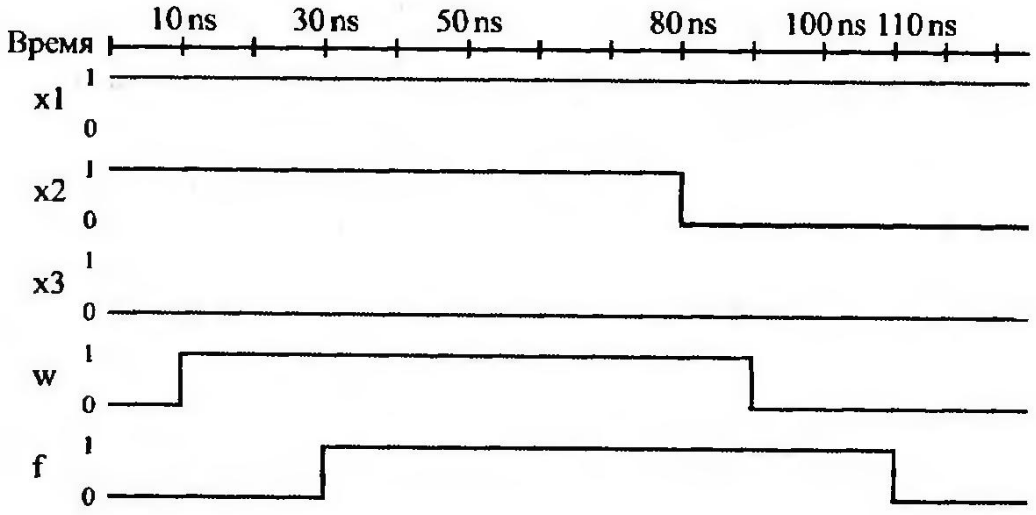


Рис. 1.17. Моделирование схемы ANDOR

1.9. Дельта-задержка

Изменим архитектурное тело example для entity ANDOR, удалив ключевые слова after и конкретные временные задержки, — тем самым логические операции and, or будут срабатывать «мгновенно».

```
architecture DELTA of ANDOR is
        signal w:bit;
begin
p0:     w<= x1 and x2;  -- нет слова after
p1:     process(w, x3)
  begin
  f<=w or x3;     -- нет слова after
  end process p1;
end DELTA;
```

Рассмотрим временную диаграмму (рис. 1.18).

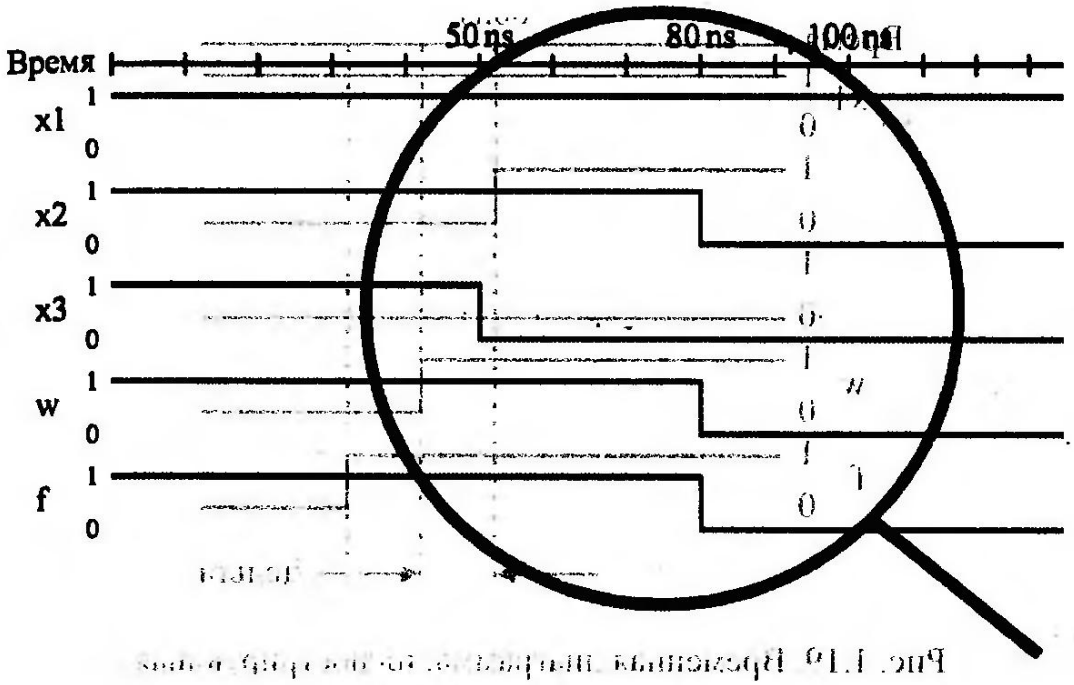


Рис. 1.18. Временная диаграмма (architecture DELTA)

В 80 ns входной сигнал x2 изменился, что послужило причиной изменения сигнала w и причиной изменения сигнала f. Эта ситуация очерчена лупой на временной диаграмме (рис.1.18). Все это случилось точно в то же время. Рассмотрим более подробно, что же произошло в момент времени 80 ns. В 80ns входной сигнал x2 изменяется. Начинается цикл моделирования. Активизируется процесс p0, выполняется, приостанавливается. Сигнал w изменяется в 0. Резонно сказать, что сигнал w изменился ***после*** сигнала x2. Мы будем ссылаться на это время как на время *дельта*, или *дельта-задержку* (рис. 1.19).

Изменение сигнала w служит причиной для процесса p1, который активизируется, выполняется, приостанавливается. Сигнал f изменяется после изменения сигнала w. Появляется другая дельта-задержка. В системах моделирования обычно на временных диаграммах дельта-задержки не показываются. Например, в системе моделирования ModelSim их можно видеть в специальном окне текстового (табличного) представления результатов моделирования.

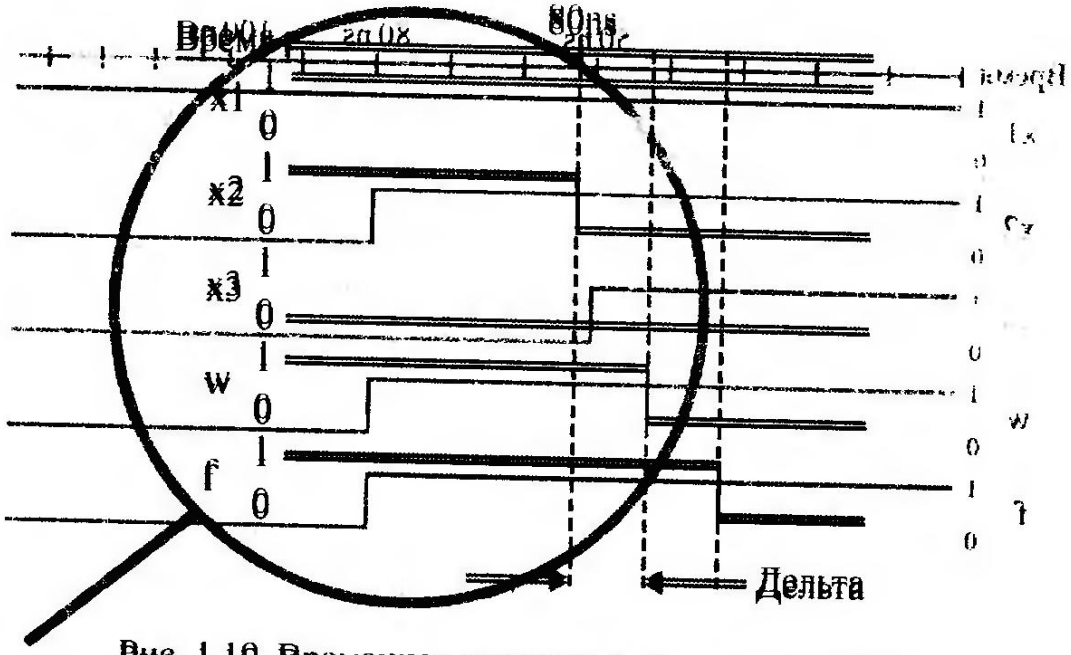


Рис. 1.19. Временная диаграмма, иллюстрирующая понятие «дельта-задержка»

Понятие дельта-задержки есть основное понятие моделирования в языке VHDL.

В одно и то же время (физическое) много сигналов могут изменяться и много процессов могут быть активными.

После того как все процессы пришли в состояние «приостановленный», система моделирования увеличивает время моделирования.

Итак, ответом на вопрос «Что представляет собой *дельта-задержка*, или просто *дельта*?» является следующее:

! ***Дельта*** есть один цикл прогона VHDL-модели.

VHDL-модель, имитирующая поведение цифровой системы, состоит из множества процессов. Во время прогона модели в данном цикле запускаются все процессы, входные параметры которых изменились с момента выполнения последнего цикла. После того как все процессы выполнены, данный цикл моделирования считается завершенным. Следующий цикл начнется после того, как в

очередной раз произойдет изменение какого-то входного сигнала для какого-то процесса. Этот период может составлять наносекунды времени моделирования или может просто означать, что мы перешли на следующий цикл моделирования, т. е. сделали временной шаг *дельта*.

Шаг величиной *дельта* образуется, например, при выполнении операторов назначения сигналов (в рассмотренном примере это сигналы w, f), не имеющих фразы **after**.

С точки зрения логической схемы *дельта* — это задержка на одном уровне (каскаде) логики, когда не задана конкретная задержка логического элемента.

Параллельные выполнения операторов (назначение сигналов) указываются при помощи символов **<=**. Символы **:=** означают процедурное выполнение и называются операторами *присваивания значения переменной.*

Требует ответа еще один важный вопрос: «Где в описаниях на языке VHDL могут быть использованы операторы присваивания значений переменным и операторы назначения сигналов?»

Ответ: «Операторы присваивания значений переменным разрешается использовать внутри процессов и в подпрограммах (функциях или процедурах)».

Оператор назначения сигналов может встречаться в любом месте выполняемого раздела архитектурного тела.

Природа оператора назначения сигнала, оператора присваивания переменной и блоков процессов дает разработчику VHDL-моделей ряд возможностей выбора [1].

В начале проектирования разработчик может использовать блок процесса, алгоритм которого реализуется при помощи операторов присваивания значений переменным, а последний оператор блока есть оператор назначения сигнала, вычисляющий выходной сигнал блока. Такой подход применяется на этапе алгоритмического проектирования, не ориентируясь на какую-то конкретную схему. Другой подход заключается в том, чтобы использовать операторы назначения сигналов. Этот подход называется иногда реализацией «потока данных» (data flow), он ориентирован на конкретную схему и более целесообразен на этапе логического проектирования.

УПРАЖНЕНИЯ

1. Что такое язык VHDL?

2. Для чего предназначен язык VHDL? Выберите правильный ответ.

a) только для проектирования заказных СБИС;

b) только для проектирования программируемых пользователями вентильных матриц;

c) для спецификации аппаратурной части проектируемой цифровой системы;

d) для спецификации системы перед разбиением ее на аппаратную и программную части;

e) для всех случаев a–d.

3. Что такое VHDL-описание, VHDL-код?

4. Что такое проект, лист проекта, примитив проекта?

5. Что такое иерархия проекта?

6. Что такое высокоуровневый синтез?

7. Что такое логический синтез?

8. Что такое структура схемы, функция схемы, поведение схемы?

9. Что такое IEEE?

10. Что такое VHDL-компилятор, VHDL-анализатор, система моделирования VHDL-кода?

11. Опишите технологию проектирования цифровых систем с использованием VHDL.

12. Можно ли на языке VHDL написать программу нахождения факториала натурального числа?

13. Что такое «синтезируемое подмножество» языка VHDL?

14. Правильно ли утверждение: «VHDL имеет много возможностей для моделирования аналоговых схем» ?

15. Когда были приняты стандарты языка VHDL?

16. Используя примитивы or2, and2, inv, составьте VHDL-код объекта add1.

17. Какие из следующих операторов являются неправильными и почему?

A_A, A_, 9U, A%B, P8_3, c4, wait, in, OU_T

18. Какие из следующих литералов являются неправильными и почему?

12_ _000, 1E-0, 2#1101#, 5#44#, #3#, 16# F#E3

19. Какие из следующих литералов с плавающей точкой являются неправильными и почему?

1_ 0.00, 100_.0, 1.1E-2, 2H#10.11#, 16 #F #E2, 1_0.0.

20. Какие из следующих литералов типа «строка бит» являются неправильными и почему?

2"110", B"1101", O"047", H"Abcd", 10"99", "0101"

21. Укажите правильные и неправильные идентификаторы

a) Decoder_1
b) _Decoder_1
c) 2FFT
d) sig_N
e) Not_Ack
f) Not-Ask
g) Sig_#N
h) FFT

22. Найдите ошибки в следующем VHDL-коде:

```
entity EXP is
end EXP;

architecture RTL of EXP is
  signal A, B, C, D, E, F : bit;
  signal X, Y, Z, S, T, R : bit;
begin
  R <= A and B and C;
  S <= B or C or D;
  T <= A and B or D;
  Y <= C nor E nor F;
```

```
    X <= A and not B and not C;
    Z <= F nand E nand B;
end RTL;
```

Исправьте их, проверьте результат с помощью VHDL-анализатора.

23. Правильно ли утверждение: «Комментарий в языке VHDL начинается и оканчивается двумя дефисами »?

24. Для каждого из утверждений ответьте, является оно правильным или ложным:

a) все элементы массива должны быть одного и того же типа;

b) значение true типа boolean эквивалентно значению 0 типа bit;

c) константы могут употребляться в выражениях.

25. Как специфицируются скалярные типы? Выберите правиль-ный ответ:

a) употребляя только понятие диапазона значений;

b) употребляя только понятие перечисления элементов;

c) употребляя a) и b) одновременно.

26. Какой тип данных, возвращаемых операторами сравнения?

27. Правильно ли, что в языке VHDL разрешено умножение вещественного числа на целое? А если один из операндов имеет тип time?

28. Какой из трех операторов отличается от двух других?

a) Z <= **not** X **and not** Y;

b) Z <= **not** (X **and** Y);

c) Z <= **not** X **and** Y;

29. Правильно ли утверждение: «Константе может быть присвоено новое значение, если оно эквивалентно предыдущему значению»?

30. Правильно ли утверждение: «Когда константа декларируется, то достаточно специфицировать ее значение, потому что тип константы косвенным образом определяется через ее значение»?

31. Правильно ли, что логические операторы могут употребляться только для типа bit ?

32. Как определяются арифметические операции для типа bit? Правильно ли, что операторы всегда определяются для отдельных (конкретных) типов данных языка VHDL ?

33. Можно ли в языке VHDL узнать (проверить) предыдущее значение сигнала, предыдущее значение переменной ? Если «да», то как это делается ?

34. Задан фрагмент VHDL-кода.

Variable TIME1, TIME2 : time;

a) TIME1 := TIME2 * 2,5;

b) TIME1:= TIME2/4;

c) TIME1:= 3.6 ns + TIME2;

d) TIME1:= TIME2 * 6.62 ns;

Укажите правильные и неправильные строки.

35. Что такое сигнал в языке VHDL? Чему соответствует сигнал в логической схеме?

36. Чем сигнал отличается от переменной? Как записывается оператор присвоения значения переменной?

37. Что такое параллельный процесс?

38. В каких состояниях бывают параллельные процессы?

39. Что такое дельта-задержка?

40. В каких разделах VHDL-кода может встретиться:

– оператор назначения сигнала;

– оператор присвоения значения переменной?

41. В какой части VHDL-кода необходимо указывать тип сигнала, который декларируется? Выберите правильный ответ:

a) когда сигнал декларируется;

b) когда сигнал употребляется первый раз в коде;

c) в пакете;

d) нет необходимости декларировать тип сигнала.

42. Когда (с точки зрения взаимодействия процессов) сигнал корректируется, т. е. получает то новое значение, которое приобретет в результате операции назначения сигнала, находящейся внутри процесса?

43. Промоделируйте «вручную» параллельные процессы

A<=X * Y;

B<=A + Z;

используя понятие дельта-задержки. Запишите VHDL-код и промоделируйте его с помощью системы моделирования.

44. Рассмотрите рис. 1.20. Назовите внешние и внутренние сигналы системы A. Ответьте, где декларируются внешние сигналы системы A, где декларируются внутренние сигналы системы A. Укажите интерфейсы для системы A и для подсистем B, C, D.

45. Рассмотрите систему, состоящую из телевизора и переносного кнопочного пульта, с помощью которого осуществляется управление телевизором. Укажите entity и architecture для данной системы.

46. Рассмотрите персональный компьютер, как систему, состоящую из системного блока, монитора и клавиатуры. Укажите entity и architecture для данной системы.

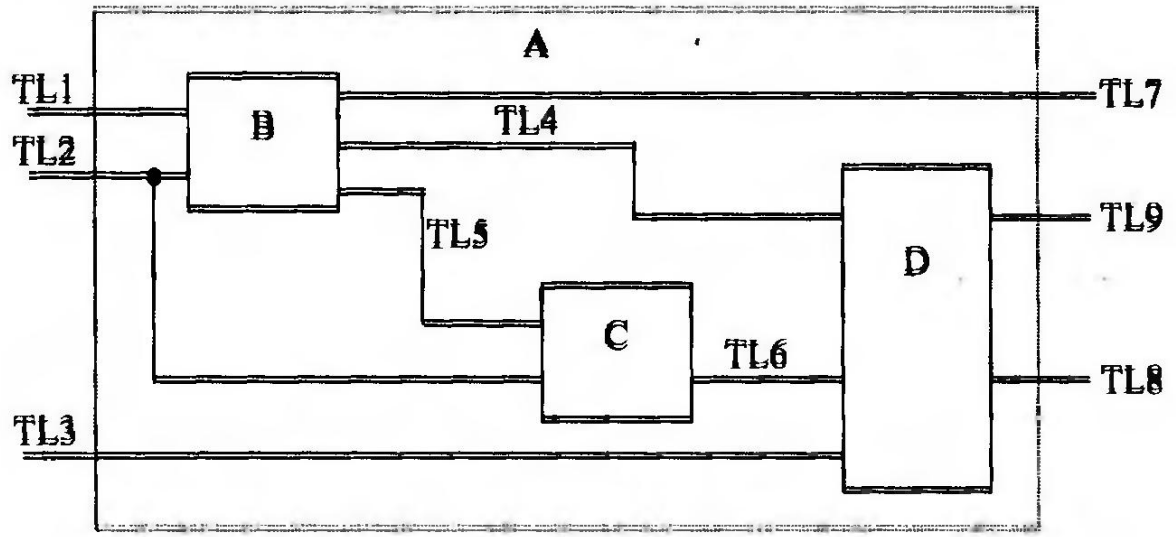


Рис. 1.20. Структура цифровой системы A