ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ЯЗЫК ОПИСАНИЯ АППАРАТУРЫ ЦИФРОВЫХ СИСТЕМ — VHDL

ОПИСАНИЕ ЯЗЫКА

МОСКВА
1995

gost.e-ecolog.ru

Предисловие

1 РАЗРАБОТАН И ВНЕСЕН Российским научно-исследовательским институтом информационных систем (РосНИИ ИС) и Всероссийской ассоциацией организаций, заинтересованных в применении языка VHDL (ВАЯПС).

2 ПРИНЯТ и ВВЕДЕН В ДЕЙСТВИЕ Постановлением Госстандарта России от 13.03.95. . . N 129.

3 Настоящий стандарт содержит полный аутентичный текст стандарта "IEEE Std 1076—87. IEEE Standard VHDL. Language Reference Manual" (IEEE Std 1076—87. Стандарт Института инженеров по электротехнике и радиоэлектронике США на Язык Описания Аппаратуры Сверхскоростных Интегральных Схем. Руководство пользователя языка). При обнаружении разночтений приоритет имеет IEEE Std 1076—87.

4 ВВЕДЕН ВПЕРВЫЕ

© ИПК Издательство стандартов, 1995

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Госстандарта России

СОДЕРЖАНИЕ

0.1 Область применения . . . 1
0.2 Нормативные ссылки . . . 1
0.3 Введение . . . 1
0.4 Определения . . . 2
1 Объекты проекта и конфигурации . . . 14
1.1 Объявления объектов . . . 15
1.1.1 Заголовок объекта . . . 15
1.1.1.1 Параметры настройки . . . 16
1.1.1.2 Порты . . . 16
1.1.2 Раздел объявлений объекта . . . 17
1.1.3 Раздел операторов объекта . . . 18
1.2 Архитектурные тела . . . 18
1.2.1 Раздел объявлений архитектурного тела . . . 19
1.2.2 Раздел операторов архитектурного тела . . . 19
1.3 Объявления конфигурации . . . 20
1.3.1 Конфигурация блока . . . 21
1.3.2 Конфигурация компонента . . . 22
2 Подпрограммы и пакеты . . . 23
2.1 Объявления подпрограмм . . . 23
2.1.1 Формальные параметры . . . 24
2.1.1.1 Передача параметров класса constant и variable . . . 24
2.1.1.2 Передача параметров класса signal . . . 25
2.2 Тела подпрограмм . . . 25
2.3 Совмещение подпрограмм . . . 26
2.3.1 Совмещение операций . . . 27
2.4 Функции разрешения . . . 27
2.5 Объявления пакетов . . . 28
2.6 Тела пакетов . . . 29
2.7 Правила согласования . . . 30
3 Типы . . . 31
3.1 Скалярные типы . . . 31
3.1.1 Перечисляемые типы . . . 32
3.1.1.1 Предопределенные перечисляемые типы . . . 32
3.1.2 Целые типы . . . 33
3.1.2.1 Предопределенные целые типы . . . 33
3.1.3 Физические типы . . . 33
3.1.3.1 Предопределенные физические типы . . . 35
3.1.4 Плавающие типы . . . 35
3.1.4.1 Предопределенные плавающие типы . . . 35
3.2 Составные типы . . . 36
3.2.1 Индексируемые типы . . . 36
3.2.1.1 Ограничения индекса и дискретные диапазоны . . . 37
3.2.1.2 Предопределенные индексируемые типы . . . 38
3.2.2 Структурные типы . . . 39
3.3 Ссылочные типы . . . 39
3.3.1 Неполные описания типов . . . 40
3.3.2 Размещение и уничтожение объектов . . . 41
3.4 Файловые типы . . . 41
3.4.1 Файловые операции . . . 41
4 Объявления . . . 42
4.1 Объявления типов . . . 42
4.2 Объявления подтипов . . . 43
4.3 Объекты . . . 44
4.3.1 Объявления объектов . . . 44
4.3.1.1 Объявления констант . . . 44
4.3.1.2 Объявления сигналов . . . 45
4.3.1.3 Объявления переменных . . . 46
4.3.2 Объявления файлов . . . 47
4.3.3 Объявления интерфейсов . . . 47
4.3.3.1 Списки интерфейсов . . . 49
4.3.3.2 Списки сопоставлений . . . 49

4 3.4 Объявление дополнительного имени . . . 50
4.4 Объявление атрибутов . . . 51
4.5 Объявления компонентов . . . 52
5 Спецификации . . . 52
5.1 Спецификация атрибута . . . 52
5.2 Спецификация конфигурации . . . 53
5.2.1 Связывающее указание . . . 54
5.2.1.1 Аспект объекта проекта . . . 54
5.2.1.2 Аспект отображения параметров настройки и аспект отображения портов . . . 55
5.2.2 Неявное связывающее указание . . . 55
5.3 Спецификация отключения . . . 56
6 Имена . . . 57
6.1 Имена . . . 57
6.2 Простые имена . . . 58
6.3 Составные имена . . . 58
6.4 Индексируемые имена . . . 59
6.5 Сечения . . . 59
6.6 Атрибуты . . . 59
7 Выражения . . . 60
7.1 Выражения . . . 60
7.2 Операторы . . . 60
7.2.1 Логические операторы . . . 61
7.2.2 Операторы отношения . . . 61
7.2.3 Аддитивные операторы . . . 62
7.2.4 Мультипликативные операторы . . . 63
7.2.5 Прочие операторы . . . 64
7.3 Операнды . . . 65
7.3.1 Литералы . . . 65
7.3.2 Агрегаты . . . 66
7.3.2.1 Агрегаты структур . . . 66
7.3.2.2 Агрегаты массивов . . . 66
7.3.3 Вызовы функций . . . 67
7.3.4 Квалифицированные выражения . . . 67
7.3.5 Преобразование типа . . . 68
7.3.6 Генераторы . . . 69
7.4 Статические выражения . . . 69
7.5 Универсальные выражения . . . 71
8 Последовательные операторы . . . 71
8.1 Оператор ожидания . . . 72
8.2 Оператор утверждения . . . 72
8.3 Оператор назначения сигнала . . . 73
8.3.1 Изменение планируемой выходной формы сигнала . . . 74
8.4 Оператор присваивания переменной . . . 75
8.4.1 Присваивание индексируемой переменной . . . 76
8.5 Оператор вызова процедуры . . . 76
8.6 Условный оператор . . . 76
8.7 Оператор выбора . . . 76
8.8 Оператор цикла . . . 77
8.9 Оператор перехода . . . 78
8.10 Оператор выхода . . . 78
8.11 Оператор возврата . . . 78
8.12 Пустой оператор . . . 79
9 Параллельные операторы . . . 79
9.1 Оператор блока . . . 79
9.2 Оператор процесса . . . 80
9.2.1 Драйверы . . . 81
9.3 Параллельный вызов процедуры . . . 81
9.4 Параллельный оператор утверждения . . . 82
9.5 Параллельный оператор назначения сигнала . . . 82
9.5.1 Условное назначение сигнала . . . 84
9.5.2 Выборочное назначение сигнала . . . 85
9.6 Оператор конкретизации компонента . . . 85
9.6.1 Конкретизация компонента . . . 86
9.7 Оператор генерации . . . 88
10 Область действия и видимость . . . 88
10.1 Область объявлений . . . 88
10.2 Область действия объявлений . . . 89

10.3 Видимость . . . 89
10.4 Описания использования . . . 91
10.5 Контекст разрешения совмещения . . . 92
11 Модули проекта и их анализ . . . 92
11.1 Модули проекта . . . 92
11.2 Библиотеки проекта . . . 93
11.3 Описание контекста . . . 93
11.4 Порядок анализа . . . 94
12 Предвыполнение и выполнение . . . 94
12.1 Предвыполнение иерархии проекта . . . 94
12.2 Предвыполнение заголовка блока . . . 95
12.2.1 Описание параметров настройки . . . 95
12.2.2 Описание отображения параметров настройки . . . 95
12.2.3 Описание портов . . . 95
12.2.4 Описание отображения портов . . . 95
12.3 Предвыполнение раздела объявлений . . . 95
12.3.1 Предвыполнение объявления . . . 95
12.3.1.1 Объявления и тела подпрограмм . . . 96
12.3.1.2 Объявления типов . . . 96
12.3.1.3 Объявления подтипов . . . 96
12.3.1.4 Объявления объектов . . . 96
12.3.1.5 Объявления дополнительных имен . . . 97
12.3.1.6 Объявления атрибутов . . . 97
12.3.1.7 Объявления компонентов . . . 97
12.3.2 Предвыполнение спецификации . . . 97
12.3.2.1 Спецификации атрибутов . . . 97
12.3.2.2 Спецификации конфигураций . . . 97
12.3.2.3 Спецификация отключения . . . 97
12.4 Предвыполнение раздела операторов . . . 97
12.4.1 Операторы блока . . . 97
12.4.2 Операторы генерации . . . 98
12.4.3 Операторы конкретизации компонентов . . . 98
12.4.4 Другие параллельные операторы . . . 98
12.5 Динамическое предвыполнение . . . 99
12.6 Выполнение модели . . . 99
12.6.1 Распространение значений сигналов . . . 99
12.6.2 Изменение неявных сигналов . . . 101
12.6.3 Цикл моделирования . . . 102
13 Лексические элементы . . . 102
13.1 Набор символов . . . 102
13.2 Лексические элементы, разделители и ограничители . . . 104
13.3 Идентификаторы . . . 104
13.4 Абстрактные литералы . . . 105
13.4.1 Десятичные литералы . . . 105
13.4.2 Базированные литералы . . . 105
13.5 Символьные литералы . . . 106
13.6 Строковые литералы . . . 106
13.7 Битово-строковые литералы . . . 106
13.8 Комментарии . . . 107
13.9 Зарезервированные слова . . . 107
13.10 Возможные замены символов . . . 108
14 Предопределенное окружение языка . . . 108
14.1 Предопределенные атрибуты . . . 108
14.2 Пакет STANDARD . . . 114
14.3 Пакет TEXTIO . . . 115
Приложение А Сводка синтаксических правил . . . 118

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ЯЗЫК ОПИСАНИЯ АППАРАТУРЫ ЦИФРОВЫХ СИСТЕМ VHDL

Описание языка

VHSIC Hardware Description Language.
Language reference manual

Дата введения 1996—01—01

0.1 ОБЛАСТЬ ПРИМЕНЕНИЯ

Настоящий стандарт распространяется на программные и технические средства проектирования радиоэлектронной аппаратуры (РЭА) и больших интегральных схем (БИС).

Стандарт определяет синтаксис и семантику языка VHDL (Very high speed integrated circuit Hardware Description Language), который предназначен для формального представления цифровых систем (ЦС) различного уровня функциональной (вычислительный комплекс, ЭВМ, устройство, узел) и (или) конструктивной (шкаф, стойка, блок, плата, БИС) сложности на различных уровнях детализации (алгоритм функционирования, регистровые передачи, вентильная схема). VHDL-описание ЦС используется для фиксации представления ЦС в процессе ее проектирования, для изучения ЦС в процессе ее эксплуатации и ремонта.

Стандарт применяется в автоматизированных системах проектирования РЭА и БИС.

0.2 НОРМАТИВНЫЕ ССЫЛКИ

В настоящем стандарте использованы ссылки на следующие стандарты:

ГОСТ 27463—87 Системы обработки информации. 7-битные кодированные наборы символов

ГОСТ 27465—87 Системы обработки информации. Символы. Классификация, наименование и обозначение

ГОСТ Р 34.303—92 Системы обработки информации. 8-битные кодированные наборы символов

0.3 ВВЕДЕНИЕ

Описание языка VHDL дано при помощи контекстно-независимого синтаксиса вместе с контекстно-зависимыми синтаксическими и семантическими требованиями, представленными в повествовательной форме. Контекстно-независимое описание синтаксиса дано в упрощенном варианте форм Бэкуса-Наура, а именно:

а) слова, состоящие из строчных букв алфавита и содержащие в ряде случаев символ подчеркивания, используются для обозначения синтаксических категорий, например:

formal_port_list.

Если имя категории используется не по назначению (то есть не для описания синтаксических правил), то символы подчеркивания заменяются пробелами. Для данного примера это будет

formal port list;

б) слова, выделенные жирным шрифтом, используются для обозначения служебных (зарезервированных) слов языка VHDL, например:

array;

в) вертикальная черта служит для разделения альтернативных употреблений, если последние не заключены вместе с этим символом в фигурные скобки. В противном случае вертикальная черта является частью конструкции, например:

```
letter_or_digit::= letter | digit
choice :: = choice { | choice }
```

г) квадратные скобки используются для выделения дополнительных (необязательных) употреблений, например два следующих правила являются эквивалентными:

```
return_statement :: = return [expression];
return_statement :: = return;
```

Издание официальное

д) фигурные скобки используются для выделения повторяющихся употреблений. Повторение может иметь место 0 и более раз; повторение раскрывается в рекурсивной форме слева направо. Например, два следующих правила являются эквивалентными:

term :: = factor {multiplying_operator factor}
term :: = factor | term multiplying_operator factor |

е) если имя какой-нибудь синтаксической категории частично слева выделено курсивом, то это эквивалентно невыделенной части имени категории и рассматривается как дополнительная семантическая информация. Например, *type*_name и *subtype*_name для понятия name являются эквивалентными;

ж) выделение жирным шрифтом терминов в тексте указывает на определение этих терминов;

з) термин simple_name используется в каждом случае употребления идентификатора, который уже обозначает некоторое объявленное понятие.

В некоторых разделах данного документа содержатся примеры и замечания. Примеры иллюстрируют возможные формы употребления описанных конструкций. Замечания используются для выделения последовательностей правил, описанных в данном разделе или в другом месте, и не должны рассматриваться как часть определения языка.

0.4 ОПРЕДЕЛЕНИЯ

В настоящем стандарте применяются следующие термины:

Агрегат (Aggregate) — Вычисление агрегата вырабатывает значение составного типа. Это значение задается указанием значений каждого из элементов агрегата. Для указания сопоставления значения и элемента может быть использовано либо позиционное сопоставление, либо именованное сопоставление (7.3.2)

Активный драйвер (Active Driver) — Драйвер считается активным в течение цикла моделирования, в котором он принимает новое значение, независимо от того, отличается или нет это значение от предыдущих значений драйвера (12.6.1)

Анализ (Analysis) — Анализ файла проекта, содержащего VHDL-описание, подразумевает синтаксический и семантический анализ этого описания и добавление промежуточной формы представления модулей проекта в библиотеку проекта (11.1)

Анонимный (Anonymous) — Ряд имен создается неявно; простое имя такого описания не всегда определено и в этом случае такое описание считается анонимным. Базовый тип числового типа или индексируемого типа является анонимным типом; аналогично, объект, указываемый ссылочным значением, является анонимным (4.1)

Атрибут (Attribute) — Атрибут определяет некоторую характеристику именованного понятия. Некоторые атрибуты являются предопределенными атрибутами и могут быть одним из следующих видов: типом, диапазоном, значением, сигналом или функцией. Остальные атрибуты определяются пользователем и всегда являются константами (4.4)

ASCII — Американский стандартный код обмена информацией. Пакет Standard содержит определение типа Character, представляющего набор символов ASCII (3.1.1, 14.2)

Библиотека (Library) — См. Библиотека проекта (design library).

Библиотека проекта (Design Library) — Библиотека проекта представляет собой зависящее от реализации средство, позволяющее хранить промежуточные представления проанализированных описаний (11.2)

Библиотека ресурсов (Resource Library) — Библиотека ресурсов — библиотека, содержащая библиотечные модули, ссылка на которые имеется в анализируемом модуле (11.2)

Библиотечный модуль (Library Unit) — Библиотечным модулем является промежуточная форма представления проанализированного модуля проекта (11.1)

Блок (Block) — Блок представляет часть иерархии проекта. Блоком является либо внешний блок, либо внутренний блок (1)

Бокс (Box) — Символ <> (называемый боксом) в определении подтипа индекса указывает на неопределенный диапазон (различные объекты этого типа необязательно должны иметь одинаковые границы и направления) (3.2.1)

Верхняя граница (Upper Bound) — Для диапазона L **to** R или L **downto** R наибольшая из величин L или R называется верхней границей диапазона (3.1)

Вид (Mode) — Вид порта или параметра задает направление прохождения информации через этот порт или параметр. Значениями вида являются **in**, **out**, **inout**, **buffer** или **linkage** (4.3.3)

Видимый (Visible) — Объявление идентификатора считается видимым в конкретном месте текста, если согласно правилам видимости объявление определяет возможный смысл появления этого идентификатора. Видимость объявления может достигаться косвенно (то есть при помощи расширенных имен) или непосредственно (то есть при помощи простых имен) (10.3)

Внешний блок (External Block) — Внешним блоком является блок, определяемый объектом проекта (1)

Внутренний блок (Internal Block) — Вложенный блок в модуле проекта, определяемый оператором блока, является внутренним блоком (1)

Возобновляться (Resume) — Процесс, возобновляющийся в данном цикле моделирования, становится готовым для выполнения и выполнится в конце данного цикла моделирования (12.6.3)

Восходящий (Ascending) — Диапазон L **to** R называется восходящим диапазоном (3.1)

Входы (Inputs) — Сигналы, идентифицируемые самым длинным статическим префиксом каждого имени сигнала, входящего в качестве первичного в каждое выражение (отличное от выражения времени) внутри параллельного оператора назначения сигнала, являются входами этого оператора (9.5)

Выполняться (Execute) — Считается, что процесс выполняется, если он выполняет действия, заданные алгоритмом, описанным в разделе операторов этого процесса (12.6)

Выражение (Expression) — Выражение определяет вычисление значения (7.1)

Выражение защиты (Guard Expression) — Выражение защиты представляет собой выражение, сопоставляемое с оператором блока, которое управляет назначением защищенных сигналов внутри этого блока (4.3.1.2., 9.1)

Выражение начального значения (Initial Value Expression) — Выражение начального значения задает начальное значение, присваиваемое переменной или константе (4.3.1.3)

Генератор (Allocator) — Генератор является операцией, используемой для создания анонимных переменных объектов, доступных через ссылочные значения (3.3, 7.3.6)

Глобальное статическое выражение (Globally Static Expression)	— Глобальное статическое выражение является выражением, которое может быть вычислено в процессе предвыполнения иерархии проекта, в которой оно находится. Локально статическое выражение также является глобально статическим (7.4)
Глобальное статическое первичное (Globally Static Primary)	— Глобальным статическим первичным является локальное статическое первичное или одна из определенных групп первичных, являющихся глобально статическими (7.4)
Диапазон (Range)	— Диапазон задает подмножество значений скалярного типа (3.1)
Диапазон индекса (Index Range)	— Диапазон значений, принадлежащих диапазону, соответствующему некоторому индексу, является диапазоном индекса (3.2.1)
Дискретный диапазон (Discrete Range)	— Дискретным диапазоном является диапазон, границы которого имеют дискретный тип (3.2.1.1)
Дискретный массив (Discrete Array)	— Дискретным массивом является одномерный массив, элементы которого имеют дискретный тип (7.2.2)
Дискретный тип (Discrete Type)	— Дискретным типом является перечисляемый тип или целый тип (3.1)
Дополнительное имя (Alias)	— Альтернативное имя объекта (4.3.4)
Драйвер (Driver)	— Драйвер сигнала представляет собой вместилище проектируемой выходной формы этого сигнала. Значение сигнала является функцией текущих значений его драйверов. Каждый процесс, содержащий значение сигнала, неявно содержит драйвер этого сигнала. Оператор назначения сигнала оказывает влияние только на сопоставленный с ним драйвер
Зависеть от библиотечного модуля (Depend)	— Модуль проекта, содержащий явно или неявно ссылку на другие библиотечные модули, зависит от этих модулей. Эта зависимость оказывает влияние на допустимый порядок анализа модулей проекта
Зависеть от значения сигнала (Depend)	— Текущее значение неявного сигнала R считается зависимым от текущего значения другого сигнала S, если R обозначает неявный сигнал S'STABLE(T), S'QUIET(T) или S'TRANSACTION, или, если R обозначает неявный сигнал GUARD, а S является любым другим неявным сигналом, имя которого задано внутри выражения защиты, определяющего текущее значение R (12.6.2)
Задающее значение (Driving Value)	— Задающим значением сигнала является значение, которое этот сигнал поставляет в качестве источника другого сигнала (12.6.1)
Защита (Guard)	См. выражение защиты (guard expression)
Защищенное назначение (Guarded Assignment)	— Защищенным назначением является параллельный оператор назначения сигнала, содержащий опцию guarded (9.5)
Защищенный сигнал (Guard Signal)	— Сигнал, объявленный с использованием описателей register или bus является защищенным сигналом. Для таких сигналов назначение внутри оператора назначения защищенного сигнала имеет специальный смысл (4.3.1.2)
Защищенный целевой объект (Guarded Target)	— Целевой объект оператора назначения сигнала, состоящий исключительно из защищенных сигналов, является защищенным целевым объектом. Целевой объект, состоящий только из незащищенных сигналов, является незащищенным целевым объектом (9.5)

Идентифицировать (Identify) — Считается, что имя в элементе сопоставления агрегата, используемого в назначении или присваивании целевого объекта, идентифицирует сигнал или переменную и каждый подэлемент этого сигнала или переменной (8.3., 8.4)

Иерархия проекта (Design Hierarchy) — Иерархия проекта представляет собой иерархию объектов проекта, получаемую в результате успешной декомпозиции объекта проекта на подкомпоненты, которые, в свою очередь, также декомпозируются (1)

Изменяться (Update) — Считается, что значение сигнала изменяется или явно, если этот сигнал используется в качестве целевого объекта в операторе назначения сигнала; или косвенно, если этот сигнал сопоставлен с объектом интерфейса, вид которого **out**, **buffer**, **inout** или **linkage**, или один из подэлементов сигнала изменяется (4.3.3)

Именованное сопоставление (Named Association) — Считается, что элемент сопоставления является именованным, если формальный указатель в нем задан явно (4.3.3.2, 7.3.2)

Имя (Name) — Каждая форма объявления сопоставляет с описываемым понятием идентификатор. Только внутри области действия этого объявления существуют места, в которых возможно использовать этот идентификатор для ссылки на сопоставленное с ним объявленное понятие; такие места определяются правилами видимости. В таких местах считается, что этот идентификатор является именем этого понятия (4, 6.1)

Имя единицы (Unit Name) — В объявлении физического типа каждое объявление единицы (как базовой, так и вторичной) определяет имя единицы (3.1.3)

Индексируемый тип (Array Type) — Значение индексируемого типа состоит из элементов, имеющих один и тот же подтип (а, следовательно, один и тот же тип). Каждый элемент однозначно отличается индексом (для одномерного массива) или последовательностью индексов (для многомерного массива). Каждый индекс должен быть значением дискретного типа и находиться в соответствующем диапазоне индекса (3.2.1)

Инерционная задержка (Inertial Delay) — Инерционная задержка представляет собой модель задержки, используемую при моделировании переключательных схем; импульс, длительность которого меньше, чем время переключения схемы, не будет передан. Эта задержка является умалчиваемой для операторов назначения сигнала (8.3)

Интервал времени блокировки (Timeout Interval) — Интервал времени блокировки, задаваемый описанием времени блокировки оператора ожидания (until предложение), определяет максимальное время, на которое процесс приостанавливается этим оператором (8.1)

Источник (Source) — Источник сигнала поставляет значение сигнала. Источником может быть либо драйвер, либо порт экземпляра компонента, с которым этот сигнал сопоставлен, либо собрание источников (составной источник) (4.3.1.2)

Константа (Constant) — Константой является объект, значение которого изменить нельзя (4.3.1.1)

Конфигурация (Configuration) — Конфигурация описывает, каким образом экземпляры компонентов в конкретном блоке связываются с объектами проекта для того, чтобы определить компоновки объектов проекта при формировании полного проекта (1, 1.3)

Локально статическое выражение (Locally Static Expression) — Выражение, которое может быть вычислено в течение анализа модуля проекта, в котором оно используется, является локально статическим выражением (7.4)

Локально статическое имя (Locally Static Name) — Считается, что имя является локально статическим, если каждое выражение в этом имени является локально статическим (6.1)

Локально статическое первичное (Locally Static Primary) — Локально статическое первичное — это одна из определенной группы первичных, включающая литералы, ряд констант и ряд атрибутов (7.4)

Множество чувствительности (Sensitivity Set) — Множеством чувствительности оператора ожидания является множество сигналов, к которым этот оператор чувствителен. Явно множество чувствительности задается описанием чувствительности (on предложение); или оно может подразумеваться описанием условия (until предложение) (8.1)

Модель (Model) — Результатом предвыполнения иерархии проекта является модель, которая может быть выполнена с целью моделирования проекта, представленного этой моделью (12.6)

Модуль проекта (Design Unit) — Модуль проекта может быть независимо проанализирован и помещен в библиотеку проекта. Модулем проекта является объявление объекта, объявление архитектуры, объявление конфигурации, объявление пакета или объявление тела пакета (11.1)

Нарушение утверждения (Assertion Violation) — Нарушение утверждения возникает в том случае, когда вычисление условия в операторе утверждения дает ложное значение (8.2)

Неограниченный подтип (Unconstrained Subtype) — Подтип считается неограниченным, если он связан с условием, которое не налагает никаких ограничений (4.3)

Неполная константа (Deferred Constant) — Неполной константой является константа, описанная (с помощью объявления неполной константы) в объявлении пакета и не имеющая значения; неполная константа имеет соответствующее полное описание в соответствующем теле пакета, определяющее значение этой константы (4.3.1.1)

Неполное описание типа (Incomplete Type Declaration) — Неполное описание типа используется для определения рекурсивных ссылочных типов (3.3.1)

Непосредственная область действия (Immediate Scope) — Непосредственной областью действия объявления, стоящей непосредственно внутри данной области объявлений, является часть области действия, распространяющаяся от начала объявления до конца области объявлений (10.2)

Непосредственно внутри (Immediately Within) — Считается, что объявление стоит непосредственно внутри области объявлений, если область является самой внутренней областью, объемлющей это объявление, не учитывая при этом область объявлений (если она есть), сопоставленную с самим объявлением (10.1)

Несопоставленный формальный параметр (Unassociated Formal) — Формальный параметр, не имеющий сопоставленного с ним фактического параметра, является несопоставленным (5.2.1.2)

Неявное выражение (Default Expression) — Неявное выражение обеспечивает неявное значение, используемое для формального параметра настройки, порта или параметра, если этот объект интерфейса не сопоставлен. Неявное выражение также используется для обеспечения начального значения сигналов и их драйверов (4.3.1.2, 4.3.3)

Неявный сигнал (Implicit Signal) — Любой из сигналов S'Stable(T) или S'Quiet(T), а также любой неявный сигнал GUARD является неявным сигналом (12.6.2)

Нижняя граница (Lower Bound) — Для диапазона L **to** R или L **downto** R наименьшее из значений L и R называется нижней границей диапазона (3.1)

Нисходящий (Descending) — Диапазон L (downto) R называется нисходящим диапазоном (3.1)

Область действия (Scope) — Областью действия объявления является часть текста, в которой объявление может быть видимо. Эта часть определяется правилами видимости и совмещения (10.2)

Обозначать (Denote) — В тех местах, где объявление видимо, идентификатор, заданный в этом объявлении, обозначает понятие, описанное в этом объявлении

Объект (Object) — Понятие, содержащее значение конкретного типа, является объектом. Существуют три класса объектов: константы, сигналы и переменные (4.3)

Объект проекта (Design Entity) — Объявление объекта вместе с сопоставленным ему типом архитектуры определяет объект проекта. Различные объекты проекта могут использовать одно и то же объявление объекта, что позволяет описывать различные компоненты с одинаковым интервалом или различные представления одного и того же компонента (1)

Объявление (Declaration) — Объявление определяет понятие и сопоставляет идентификатор (или другое обозначение) с этим понятием. Такое сопоставление имеет эффект внутри области текста, называемой областью действия объявления. Внутри области действия объявления существуют места, где можно использовать этот идентификатор для ссылки на сопоставленное с ним описанное понятие. В таких местах этот идентификатор является простым именем этого понятия; в свою очередь считается, что это имя обозначает сопоставленное с ним понятие

Объявление объекта (Entity Declaration) — Объявление объекта устанавливает интерфейс между конкретным объектом проекта и средой, в которой он используется. Оно также содержит объявления и операторы, составляющие часть этого объекта. Конкретное объявление объекта может быть использовано множеством объектов проекта, имеющих различную архитектуру. Таким образом объявление объекта потенциально может представлять класс объектов проекта, имеющих одинаковый интерфейс (1, 1.1)

Ограничение (Constraint) — Ограничение определяет (необязательно собственное) подмножество значений типа. Существуют ограничения индекса, ограничения диапазона и ограничения размера (3)

Ограничение диапазона (Range Constraint) — Ограничение диапазона задает диапазон значений в типе (3.1, 3.1.2)

Ограничение индекса (Index Constraint) — Ограничение индекса устанавливает диапазон индекса для каждого индекса в индексируемом типе и, тем самым, границы массива (3.2.1.1)

Омограф (Homograph) — Каждое из двух объявлений считается омографом другого, если оба они имеют один и тот же идентификатор, и совмещение разрешено для одного из двух. Если совмещение разрешено для обоих объявлений, то каждое из двух является омографом другого, если они имеют один и тот же идентификатор, символ оператора или символьный литерал, а также профиль параметров и типа результата (10.3)

Отдельное объявление объекта (Single Object Declaration) — Объявление объекта называется отдельным, если список идентификаторов в этом объявлении содержит один идентификатор (4.3.1)

Ошибка (Error) — Ошибка является условием, при котором исходное описание становится недействительным. Если ошибка обнаружена во время анализа модуля

проекта, то для этого исходного описания не будет создан соответствующий библиотечный модуль. Ошибка выполнения вызывает прекращение процесса моделирования (11.4)

Ошибочный (Erroneous) — Термин "ошибочный" используется в отношении ошибочного условия, которое не всегда может быть обнаружено (2.1.1.1)

Параллельный оператор (Concurrent Statement) — Параллельные операторы выполняются асинхронно без определенной относительной последовательности. Параллельные операторы используются для описания потока данных и структуры (9)

Параметр (Parameter) — Параметр — это константа, переменная или сигнал, объявленные в списке интерфейса спецификации подпрограммы. Характеристики класса объектов, к которому данный параметр принадлежит, также являются характеристиками этого параметра. Кроме того, параметр имеет сопоставленный ему вид, задающий направление прохождения данных через этот параметр (2.1.1)

Параметр генерации (Generate Parameter) — Параметр генерации объявляется оператором генерации (9.7)

Параметр настройки (Generic) — Параметр настройки является константой, описанной в объявлении компонента или в объявлении объекта. В отличие от констант, значение параметра настройки может быть задано извне либо в операторе конкретизации компонента, либо в спецификации конфигурации (1.1.1.1)

Параметр цикла (Loop Parameter) — Параметр цикла объявляется самим оператором цикла (8.8)

Пассивный (Quiet) — В данном цикле моделирования сигнал, не являющийся активным, считается пассивным (12.6.1)

Пассивный процесс (Passive Process) — Считается, что оператор процесса является пассивным процессом, если ни сам процесс, ни какая-либо из процедур, для которых этот процесс является родителем, не содержат оператор назначения сигнала (9.2)

Первичный (Primary) — Один из элементов, составляющих выражение, называется первичным (7.1)

Переменная (Variable) — Переменная — объект с единственным текущим значением (4.3.1.3)

Перечисляемый тип (Enumeration Type) — Перечисляемый тип представляет собой тип, значения которого определяются значением списка или перечислением этих значений. Значения представляются интервалами перечисления (3.1, 3.1.1)

Плавающий тип (Floating Point Types) — Плавающие типы обеспечивают приближенное представление действительных чисел (3.1, 3.1.4)

Планируемая выходная форма сигнала (Projected Output Waveform) — Планируемая выходная форма сигнала состоит из последовательности, содержащей одну или более транзакций, представляющих текущее и планируемые будущие значения драйвера (9.2.1)

Подключен (Connected) — Формальный порт, сопоставленный с фактическим портом или сигналом, считается подключенным. Формальный порт, сопоставленный с зарезервированным словом **open**, считается неподключенным (1.1.1.2)

Подразумевать (Imply) — Считается, что связывающее указание в спецификации конфигурации подразумевает объект проекта, обозначаемый непосредственно, косвенно или по умолчанию (5.2.1.1)

Подтип (Subtype) — Подтип — тип вместе с наложенным ограничением (3)

Подтип индекса (Index Subtype)	— Подтипом индекса для конкретной позиции индекса в массиве является подтип, обозначаемый меткой типа в соответствующем определении подтипа индекса (3.2.1)
Подтип результата (Result Subtype)	— Подтипом результата функции является подтип значения, возвращаемого этой функцией (2.1)
Подходить (Appropriate)	— Считается, что префикс подходит для типа, если тип этого префикса является рассматриваемым типом или тип префикса является ссылочным типом, который указывает на рассматриваемый тип (6.1)
Подэлемент (Subelement)	— Термин "подэлемент" используется взамен термина "элемент", когда необходимо обозначить либо элемент, либо элемент другого элемента. В тех местах, где подэлементы исключены, используется термин "элемент" (3)
Позиционное сопоставление (Positional Association)	— Считается, что элемент сопоставления является позиционным, если он не содержит явного формального указателя; в этом случае фактический указатель в данной позиции списка сопоставления относится к элементу интерфейса, стоящему в той же позиции в списке интерфейса (4.3.3.2, 7.3.2)
Полное объявление (Full Declaration)	— Полное объявление константы представляет собой объявление константы, стоящее в теле пакета и имеющее тот же идентификатор, что и неполное объявление константы, стоящее в соответствующем объявлении пакета. Полным объявлением типа является объявление, соответствующее неполному объявлению типа (2.6)
Полностью связан (Fully Bound)	— Экземпляр компоненты является полностью связанным, если связывающее указание для этого экземпляра подразумевает и объект и архитектуру (5.2.1.1)
Полный контекст (Complete Context)	— Полным контекстом является либо объявление, либо спецификация, либо оператор (10.5)
Порт (Port)	— Портом является сигнал, описанный в списке интерфейса объявления объекта или в списке интерфейса объявления компонента. Кроме характеристик сигналов, порт имеет сопоставленный ему вид; этот вид ограничивает направления прохождения данных, допустимые для этого порта (1.1.1.2, 4.3.1.2)
Последовательный оператор (Sequential Statement)	— Последовательные операторы выполняются последовательно один за другим. Они используются для алгоритмического описания (8)
Предвыполнение (Elaboration)	— Процесс, при помощи которого объявление достигает своего эффекта, называется предвыполнением объявления. После своего предвыполнения объявление считается предвыполненным. До окончания своего предвыполнения объявление считается (а также и до предвыполнения) не предвыполненным (12)
Преобразуемый (Convertible)	— Операнд является преобразуемым, если существует неявное преобразование типа операции в данный тип (7.3.5)
Принадлежать диапазону (Belong)	— Считается, что значение V принадлежит диапазону, если отношения (нижняя граница диапазона < = V) и (V < = верхняя граница диапазона) оба являются истинными (3.1)
Принадлежать подтипу (Belong)	— Считается, что значение принадлежит подтипу конкретного типа, если оно принадлежит этому типу и удовлетворяет наложенному ограничению (3)
Приостанавливать (Suspend)	— Процесс, который прекратил свое выполнение и ожидает событие или истечение временного интервала, считается приостановленным (12.6.3)

Простое имя (Simple Name) — Простое имя понятия — это либо идентификатор, сопоставленный этому понятию при его объявлении, либо другой идентификатор, сопоставленный этому понятию в объявлении дополнительного имени (6.2)

Профиль типа параметров (Parameter Type Profile) — Считается, что два списка этих параметров имеют один и тот же профиль типа параметров, если и только если они содержат одинаковое количество параметров и в каждой позиции списков соответствующие параметры имеют один и тот же базовый тип (2.3)

Профиль типа параметров и результата (Parameter and Result Type Profile — Считается, что две подпрограммы имеют один и тот же профиль типа параметров и результата, если и только если обе они имеют один и тот же профиль типа параметров и либо обе они являются функциями с одинаковым базовым типом результата, либо ни одна из них не является функцией (2.3)

Пустая транзакция (Null Transaction) — Транзакция, получаемая вычислением пустого элемента формы сигнала, является пустой транзакцией (8.3.1)

Пустое сечение (Null Slice) — Сечение, дискретный диапазон которого есть пустой диапазон, является пустым сечением (6.5)

Пустой диапазон (Null Range) — Пустой диапазон — это диапазон, задающий пустое подмножество значений; диапазон L to R является пустым диапазоном, если L>R; диапазон L downto R является пустым диапазоном, если L<R (3.1)

Пустой массив (Null Array) — Если в ограничении индекса массива какой-либо из дискретных диапазонов определяет пустой диапазон, то такой массив является пустым массивом, не имеющим компонентов (3.2.1.1)

Пустой элемент формы сигнала (Null Waveform Element) — Пустой элемент формы сигнала используется для отключения драйвера защищенного сигнала (8.3.1)

Рабочая библиотека (Working Library) — Библиотека Проекта, в которую помещается результат анализа (библиотечный модуль) модуля проекта, является рабочей библиотекой (11.2)

Разрешение (Resolution) — Разрешение — процесс выявления разрешенного значения разрешенного сигнала, основанный на значениях множества источников этого сигнала (2.4, 4.3.1.2)

Разрешенное значение (Resolved Value) — Разрешенным значением сигнала является результат функции разрешения, сопоставленной с этим сигналом, определяемый как функция от множества входов, представленных источниками этого сигнала (2.4, 4.3.1.2)

Разрешенный сигнал (Resolved Signal) — Разрешенный сигнал — сигнал, который имеет сопоставимую ему функцию разрешения (4.3.1.2)

Распространяется (Extend) — Если считается, что часть текста в области объявлений с разобщенными частями распространяется от некоторой особой точки раздела объявлений до конца этой области, то эта часть является соответствующим подмножеством этой области объявлений (и не включает промежуточных сегментов объявлений, стоящих между объявлением интерфейса и соответствующим объявлением тела) (10.1)

Регистр (Register) — Регистр — это один из видов защищенного сигнала. Регистр сохраняет свое последнее значение даже в том случае, если все его драйверы отключены (4.3.1.2)

Регулярная структура (Regular Structure) — Регулярная структура состоит из экземпляров одного или более компонентов, устроенных и взаимосвязанных (при помощи сигналов)

в повторяющемся режиме. Каждый экземпляр может иметь характеристики, зависящие от его позиции внутри группы экземпляров. Регулярные структуры могут быть представлены через использование оператора генерации (9.7)

Родитель (Parent) — Считается, что процесс или подпрограмма являются родителем данной процедуры, если этот процесс или подпрограмма содержат оператор вызова процедуры этой процедуры или ее родителя (2.2)

Самый длинный статический префикс (Longest Static Prefix — Самым длинным статическим префиксом имени сигнала является само имя, если оно является статическим именем сигнала; в противном случае им является самый длинный префикс имени, являющегося статическим именем сигнала (6.1)

Связанный (Bound) — Метка считается связанной, если она идентифицирована в списке экземпляров спецификации конфигурации (5.2)

Сечение (Slice) — Сечением является одномерный массив, состоящий из последовательности следующих друг за другом элементов другого одномерного массива (6.5)

Сигнал (Signal) — Сигнал — объект, имеющий историю своих прошлых значений. Сигнал может иметь множество драйверов, каждый из которых имеет свое текущее значение и планируемые будущие значения. Термин "сигнал" относится к объектам, описанным либо при помощи объявлений сигнала, либо при помощи объявлений порта (4.3.1.2)

Символьный тип (Character Type) — Перечисляемый тип является символьным типом, если, по крайней мере, один из его литералов перечисления является символьным литералом (3.1.1)

Скалярный тип (Scalar Type) — Скалярный тип — тип, значения которого не имеют элементов. Скалярными типами являются: целые, плавающие, физические и перечисляемые типы (3, 3.1)

Скрытый (Hidden) — Объявление может быть скрыто в своей области действия, если последняя содержит омограф этого объявления. Скрытое объявление не является непосредственно видимым (10.3)

Слева от (To the Left Of) — Считается, что значение V1 находится слева от значения V2 внутри данного диапазона, если оба они принадлежат этому диапазону и V2 следует за V1 (в случае восходящего диапазона) или V2 предшествует V1 (в случае нисходящего диапазона) (3.1)

Событие (Event) — Считается, что на сигнале произошло событие, если текущее значение этого сигнала изменилось в результате замещения этого сигнала его эффективным значением (12.6.1)

Совместимый (Compatible) — Ограничение диапазона совместимо с подтипом, если каждая граница этого диапазона принадлежит этому подтипу или если это ограничение диапазона определяет пустой диапазон. Ограничение индекса совместимо с индексируемым типом, если и только если ограничение, определяемое каждым дискретным диапазоном в этом ограничении индекса, совместимо с соответствующим подтипом индекса в этом индексируемом типе (3.1, 3.2.1)

Совмещение (Overloading) — Идентификаторы или литералы перечисления, обозначающие два разных понятия, считаются совмещенными. Совмещенными также могут быть литералы перечисления, подпрограммы и предопределенные операторы (2.3.1, 3.1.1)

Согласование (Conform) — Считается, что две спецификации программы согласуются, если (исключая ряд допустимых минимальных вариаций) они представлены одной и той же последовательностью лексических элементов, и

соответствующим лексическим элементам дается правилами видимости одинаковое значение. Согласование аналогично определено для объявления неполных констант (2.7)

Соответствующий элемент (Matching Element) — Соответствующими элементами являются элементы двух значений составного типа, определенных таким образом, что они подходят друг другу при использовании их в ряде логических операций и операциях отношения (7.2.2)

Сопоставленный драйвер (Associated Driver) — Сопоставленный драйвер для оператора назначения сигнала — это отдельный драйвер этого сигнала в (явном или эквивалентном) операторе процесса, содержащем этот оператор назначения (9.2.1)

Составной тип (Composite Type) — Составной тип — тип, значения которого состоят из элементов. Имеются два класса составных типов: индексируемый и структурный типы (3, 3.2)

Спецификация (Specification) — Спецификация сопоставляет дополнительную информацию с ранее объявленным понятием. Существуют четыре вида спецификаций: спецификации атрибута, спецификации инициализации, спецификации конфигурации и спецификации отключения (5)

Спецификация подпрограммы (Subprogram Specification) — Спецификация подпрограммы задает обозначение подпрограммы, все формальные параметры подпрограммы (если они есть) и тип результата (в случае, если подпрограмма является функцией) (2.1)

Список интерфейса (Interface List) — Список интерфейса описывает объекты интерфейса, требуемые подпрограммой, компонентом, объектом проекта или оператором блока (4.3.3.1)

Список интерфейса настройки (Generic Interface List) — Список интерфейса настройки определяет локальные или формальные константы настройки (1.1.1.1, 4.3.3.1)

Список интерфейса параметров (Parameter Interface List) — Список интерфейса параметров описывает параметры для подпрограммы. Он может содержать объявления констант интерфейса, объявления переменных интерфейса, объявления сигналов интерфейса или комбинацию этих объявлений (4.3.3.1)

Список интерфейса портов (Port Interface List) — Список интерфейса портов описывает порты блока, компонента или объекта проекта. Он состоит только из объявлений сигналов интерфейса (1.1.1.2, 4.3.1.1)

Список сопоставления (Association List) — Список сопоставления устанавливает соответствие между именами локальных или формальных портов или параметров и локальными или фактическими именами и выражениями (4.3.3.2)

Ссылочный тип (Access Type) — Значение ссылочного типа может указывать на объект, созданный генератором (3.3)

Статический (Static) — См. локально статический и глобально статический

Статический диапазон (Static Range) — Статическим диапазоном является диапазон, границы которого представлены статическими выражениями (7.4)

Статическое имя (Static Name) — Имя считается статическим, если каждое выражение, входящее в это имя как часть (например, выражение индекса), является статическим (6.1)

Статическое имя сигнала (Static Signal Name) — Статическое имя сигнала — это статическое имя, обозначающее сигнал (6.1)

Структурный тип (Record Type) — Структурный тип является составным типом. Значения структурного типа состоят из именованных элементов (3.2.2, 7.3.2.1).

Текущее значение (Current Value) — Текущим значением драйвера является компонент значения транзакции, компонент времени которой не превышает текущего времени моделирования (9.2.1)

Тело архитектуры (Architecture Body) — Тело архитектуры описывает внутреннюю организацию или функционирование объекта проекта. Каждое тело архитектуры, сопоставленное с конкретным объявлением объекта проекта, определяет уникальный объект проекта. Архитектура может быть использована для описания поведения, данных или структуры объекта проекта (1, 1.2)

Тип (Type) — Тип — множество значений и операций над ними (3)

Транзакция (Transaction) — Транзакция представляет собой совокупность, состоящую из двух компонентов: значения и времени. Компонент значения представляет (текущее или) будущее значение драйвера; компонент времени представляет относительную задержку до того момента, когда это значение станет текущим (9.2.1)

Транспортная задержка (Transport Delay) — транспортная задержка является дополнительной моделью задержки для назначения сигнала. Транспортная задержка является характеристикой устройств (таких как линии передачи), которые выставляют бесконечно малую частоту срабатывания: любой импульс передается независимо от того, насколько мала его длительность (8.3)

Удовлетворять (Satisfy) — Считается, что значение удовлетворяет ограничению, если это значение находится в подмножестве, определяемом этим ограничением (3, 3.2.1.1)

Указываемый подтип (Designated Subtype) — Указываемым подтипом ссылочного типа является подтип, определяемый указанием подтипа в описании этого ссылочного типа (3.3)

Указываемый тип (Designated Type) — Указываемым типом ссылочного типа является базовый тип подтипа, определяемого указанием подтипа в описании этого ссылочного типа (3.3)

Указывать (Designate) — Считается, что непустое ссылочное значение указывает на объект (3.3)

Укороченная операция (Short_Circuit Operation) — Укороченная операция — операция, для которой правый операнд вычисляется только в том случае, когда левый имеет определенное значение. Укороченными операциями являются предопределенные логические операции **and**, **or**, **nand** и **nor**, определенные для операндов типов BIT и BOOLEAN (7.2)

Универсальный действительный (Universal_Real) — Плавающие литералы — литералы анонимного предопределенного типа, называемого *универсальный_действительный* в данном руководстве (3.1.4, 7.3.1)

Универсальный целый (Universal_Integer) — Целые литералы — литералы анонимного предопределенного типа, называемого *универсальный_целый* в данном руководстве (3.1.2, 7.3.1)

Упорядоченный слева направо (Left_to_Right Order) — Последовательность значений конкретного диапазона является упорядоченной слева направо, если каждое значение в этой последовательности стоит левее следующего значения в этой последовательности внутри этого диапазона (за исключением последнего значения) (3.1)

Файловый тип (File Type) — Файловые типы обеспечивают доступ к файлам, находящимся в среде главной системы (3, 3.4)

Файл проекта (Design File) — Файл проекта представляет собой последовательность, состоящую из одного или более модулей проекта (11.1)

Фактический параметр (Actual) — Фактический параметр это либо выражение, либо порт, либо сигнал, либо переменная, сопоставленные с формальным портом, формальным параметром или формальным параметром настройки (1.1.1.2, 4.3.3.2, 5.2.1.2)

Физический тип (Physical Type) — Физический тип используется для представления измерений некоторой величины (3.1, 3.1.3)

Формальный параметр (Formal) — Формальным параметром является либо формальный порт или формальный параметр настройки объекта проекта, либо формальный параметр подпрограммы (2.1, 2.1.1, 4.3.3.2, 5.2.1.2)

Форма сигнала (Waveform) — Форма сигнала состоит из серий транзакций. Каждая транзакция представляет будущее значение драйвера сигнала. Все транзакции в форме сигнала упорядочены по времени,то есть одна транзакция возникает перед другой в том случае, если первая представляет значение, запланированное на более ранний срок, чем значение, представленное другой транзакцией (8.3)

Функция разрешения (Resolution Function) — Определяемая пользователем функция, вычисляющая разрешенное значение разрешенного сигнала, называется функцией разрешения (2.4, 4.3.1.2)

Целый тип (Integer Type) — Значения целого типа представляют целые числа внутри заданного диапазона (3.1, 3.1.2)

Числовой тип (Numeric Type) — Числовым типом является либо целый, либо плавающий, либо физический тип (3.1)

Читать (Read) — Считается, что значение объекта читается, если имеется ссылка на его значение или на его атрибут (4.3.3)

Шина (Bus) — Шина является одним из видов защищенного сигнала. Шина принимает задаваемые пользователем значения, когда все ее источники отключены (4.3.3, 4.3.1.2)

Экземпляр (Instance) — Оператор конкретизации компонента представляет экземпляр соответствующего компонента. Каждый экземпляр компонента может иметь различные фактические параметры, сопоставляемые с его локальными портами или параметрами настройки (9.6.1)

Элемент сопоставления (Association Element) — Элемент сопоставления связывает фактический или локальный параметр с локальным или формальным параметром (4.3.3.2)

Эффективное значение (Effective Value) — Эффективное значение конкретного сигнала представляет собой значение, получаемое вычислением ссылки на этот сигнал внутри выражения (12.6.1)

Ядро модели (Kernel Process) — Ядро модели осуществляет выполнение операций ввода/вывода, распространение значений сигналов и изменение значений неявных сигналов (таких, как S'Stable(N)); а также выявление происходящих событий, выполнение соответствующих процессов в ответ на эти события (12.6)

1 ОБЪЕКТЫ ПРОЕКТА И КОНФИГУРАЦИИ

Объект проекта (design entity) является первичной абстракцией аппаратных средств в VHDL. Он представляет собой часть аппаратного проекта, который имеет четко определенные входы и выходы и который выполняет четко определенную функцию. Объект проекта может представлять всю проектируемую систему, некоторую подсистему, плату, кристалл, макро-ячейку, логический элемент или любой другой

уровень абстракции, находящийся между перечисленными. *Конфигурация (configuration)* может быть использована для описания того, как объекты проекта должны быть соединены для создания полного проекта.

Объект проекта может быть описан в терминах иерархии *блоков (blocks)*, каждый из которых представляет часть всего проекта. Блоком верхнего уровня в такой иерархии является сам объект проекта; такой блок является *внешним (external)* блоком, который располагается в библиотеке и может быть использован в качестве компонента в других проектах. Вложенные блоки в этой иерархии являются *внутренними (internal)* блоками, описанными с помощью операторов блока (см. 9.1).

Объект проекта может также быть описан в терминах взаимосвязанных компонентов. Каждый компонент объекта проекта может быть связан с объектом проекта более низкого уровня для того, чтобы определить структуру или поведение этого компонента. Результатом успешной декомпозиции объекта проекта в компоненты и связывания этих компонентов с другими объектами проекта, которые в свою очередь могут быть декомпозированы таким же способом, является иерархия объектов проекта, представляющих весь проект в целом. Такая совокупность объектов проекта называется *иерархией проекта (design hierarchy)*. Само связывание, необходимое для описания иерархии проекта, может быть задано в конфигурации объекта верхнего уровня в этой иерархии.

В данном разделе даны способы описания объектов проекта и конфигураций. Объект проекта описывается объявлением объекта (entity declaration) вместе с соответствующим архитектурным телом (architecture body). Конфигурация описывается объявлением конфигурации (configuration declaration).

1.1. Объявления объектов

Объявление объекта описывает интерфейс между данным объектом проекта и средой, в которой он используется. Оно также может содержать объявления и операторы, являющиеся составной частью этого объекта. Данное объявление объекта может совместно использоваться многими объектами проекта, имеющими различную архитектуру. Таким образом, объявление объекта может потенциально представлять некоторый класс объектов проекта с одним и тем же интерфейсом.

Пример —

```
entity_declaration :: =
  entity identifier is
    entity_header
    entity_declarative_part
  [ begin
    entity_statement_part ]
  end [ entity_simple_name ];
```

Заголовок объекта (entity header) и раздел объявлений (entity declarative part) состоят из описательных элементов, присущих каждому объекту проекта, чей интерфейс описывается данным объявлением объекта. Раздел операторов объекта (entity statement part), если он имеется, задает параллельные операторы, присутствующие в каждом объекте проекта такого класса.

Если в конце описания объекта используется простое имя (simple name), то оно должно совпадать с идентификатором (identifier) этого описания.

1.1.1 Заголовок объекта

Заголовок объекта содержит объявления объектов, используемых для связи между объектом проекта и окружающей средой.

```
entity_header :: =
    [ formal_generic_clause ]
    [ formal_port_clause ]

generic_clause :: =
    generic (generic_list);

port_clause :: =
    port (port_list);
```

Список параметров настройки (generic list) в формальном описании настройки (formal generic clause) задает общие константы, значения которых определяются окружающей средой. Список портов (port list) в формальном описании портов (formal port clause) задает входные/выходные порты объекта проекта.

В определенных случаях имена общих констант и портов, объявленных в заголовке объекта, становятся видимыми вне объекта проекта (см. 10.2 и 10.3).

Примеры

1 Объявление объекта, содержащее только объявление портов:

```
entity Full_Adder is D
    port (X, Y, Cin : in Bit; Cout, Sum: out Bit);
end Full_Adder;
```

2 Объявление объекта, содержащее также объявления параметров настройки:

```
entity AndGate is
    generic
      (N: Natural : = 2);
    port
      (Inputs: in Bit_Vector (1 to N);
       Result: out Bit);
end AndGate;
```

3 Объявление объекта, ничего не содержащее:

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

1.1.1.1 *Параметры настройки*

Параметры настройки обеспечивают канал для передачи статической информации в блок из окружающей среды. Нижеследующее относится как к внешним блокам, описанным как объекты проекта, так и к внутренним блокам, описанным как операторы блока.

Пример —

generic_list :: = *generic*_interface_list

Параметры настройки задаются при помощи списка интерфейса настройки (generic interface list); списки интерфейса описаны в 4.3.3.1.

Каждый элемент интерфейса в таком списке задает формальный параметр настройки. Значение формального параметра настройки может быть задано соответствующим фактическим значением в списке соответствия параметров настройки. Если для данного формального параметра настройки не задано соответствующее фактическое значение, но задано выражение для неявного значения этого параметра, то значением параметра является значение этого выражения. Считается ошибкой, если для формального параметра настройки не задано фактическое значение, а в соответствующем элементе интерфейса отсутствует выражение для неявного значения.

Примечание — Параметры настройки могут быть использованы для управления характеристиками структуры, обработки данных и поведения блока или просто для документирования. В частности, параметры настройки могут быть использованы для указания размера портов, количества подкомпонентов в блоке, временных характеристик блока или даже физических характеристик проекта таких, как температура, емкость, размещение и пр.

1.1.1.2 *Порты*

Порты обеспечивают каналы для динамической связи между блоком и окружающей средой. Нижеследующее относится как к внешним блокам, описанным как объекты проекта, так и к внутренним блокам, описанным как операторы блока.

port_list :: = *port*_interface_list

Порты блока задаются списком интерфейса портов (port interface list); списки интерфейса описаны в 4.3.3.1. Каждый элемент интерфейса в этом списке задает формальный порт. Порты блока могут быть сопоставлены с сигналами из окружающей среды, в которой этот блок используется, с целью взаимодействия с другими блоками в этой среде. Сам порт является сигналом (см. 4.3.1.2), таким образом формальный порт некоторого блока может быть сопоставлен с портом объемлющего блока. Порт или сигнал, сопоставленный с конкретным формальным портом, называется *фактическим* (*actual*), соответствующим этому формальному порту (см. 4.3.3.2). Фактический порт или сигнал должен обозначаться статическим именем (см. 6.1).

Если после того, как конкретное описание является предвыполненным (см. раздел 12), формальный порт сопоставлен с фактическим портом, то в зависимости от вида формального порта (см. 4.3.3) накладываются следующие ограничения:

1) Для формального порта вида **in** сопоставляемый фактический порт может быть только портом вида **in, inout** или **buffer.**

2) Для формального порта вида **out** сопоставляемый фактический порт может быть только портом вида **out** или **inout.**

3) Для формального порта вида **inout** сопоставляемый фактический порт может быть только портом вида **inout.**

4) Для формального порта вида **buffer** сопоставляемый фактический порт может быть только портом вида **buffer.**

5) Для формального порта вида **linkage** сопоставляемый фактический порт может быть портом любого вида.

Порт вида **buffer** может иметь по крайней мере один источник (см. 4.3.1.2). Более того, любой фактический порт, сопоставляемый с формальным портом вида **buffer**, может иметь по крайней мере один источник.

Если формальный порт сопоставляется с фактическим портом или сигналом, то считается, что этот порт *подключен* (*connected*). Если вместо этого формальный порт сопоставляется с зарезервированным словом open, то считается, что этот порт *отключен* (*unconnected*). Порт вида in может быть отключен только в том случае, если его объявление содержит неявное выражение (см. 4.3.3). Порт любого вида, отличного от in, может быть отключенным, если его тип не является неограниченным индексируемым типом.

1.1.2 *Раздел объявлений объекта*

Раздел объявлений конкретного объекта содержит объявления, общие для всех объектов проекта, чей интерфейс определяется этим объявлением объекта.

```
entity_declarative_part :: =
    {entity_declarative_item}

entity_declarative_item :: =
    subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | signal_declaration
  | file_declaration
  | alias_declaration
  | attribute_declaration
  | attribure_specification
  | disconnection_specification
  | use_clause
```

Имена, описанные в сегментах объявления (declarative_item) конкретного объявления объекта, видимы внутри тел соответствующих объектов проекта, а также внутри определенных частей соответствующего объявления конфигурации.

Пример

```
entity ROM is
        port ( Addr : in Word;
               Data : out Word;
               Sel : in Bit);
        type Instruction is array (1 to 5) of Natural = ;
        type Program is array (Natural range < >) of
               Instruction;
        use Work.OpCodes.all, Work.RegisterNames.all;

constant ROM_Code : Program : =
    (
        (STM, R14, R12, 12, R13),
        (LD, R7, 32, 0, R1),
        (BAL, R14, 0, 0, R7),
```

```
            •
            •
            •
          и т.д.
    );
end ROM;
```

1.1.3 *Раздел операторов объекта*

Раздел операторов объекта содержит параллельные операторы, общие для всех объектов проекта этого класса.

```
entity_statement_part :: =
    {entity_statement}
entity_statement :: =
    concurrent_assertion_statement
    | passive_concurrent_procedure_call
    | passive_process_statement
```

Раздел операторов объекта может содержать только три вида параллельных операторов: параллельный оператор утверждения, параллельный оператор вызова процедуры и оператор процесса. Причем последние два вида должны использоваться только в пассивной форме (см. 9.2). Указанные операторы могут быть использованы для контролирования рабочих режимов или характеристик объекта проекта.

Пример —

```
entity Latch is

    port (Din:  in  Word;
          Dout: out Word;
          Load: in  Bit;
          Clk:  in  OBit);
    constant Setup: Time : = 12ns;
    constant PulseWidth: Time : = 50ns;

    use Work.TimingMonitors.all;
begin
    assert Clk = '1' or Clk'Delayed'Stable (PulseWidth);
    CheckTiming (Setup, Din, Load, Clk);
end;
```

1.2. Архитектурные тела

Архитектурное тело описывает тело объекта проекта. Оно задает взаимосвязь между входами и выходами объекта проекта и может быть выражено в терминах структуры, потока данных или поведения. Спецификация тела может быть полной или частичной.

```
architecture_body :: =
    architecture identifier of entity_name is
        architecture_declarative_part
    begin
        architecture_statement_part
    end [ architecture_simple_name ];
```

Идентификатор описывает простое имя архитектурного тела. Это имя выделяет архитектурное тело из совокупности тел, сопоставленных с одним и тем же объявлением объекта.

Имя объекта описывает имя объявления объекта, определяющего интерфейс объекта проекта. Для конкретного объекта проекта объявление объекта и сопоставленное с ним архитектурное тело должны находиться в одной и той же библиотеке.

Если в конце архитектурного тела используется простое имя, то оно должно повторять идентификатор этого тела.

Для конкретного объявления объекта может существовать более чем одно соответствующее архитектурное тело, каждое из которых описывает отличное тело с одним и тем же интерфейсом, и таким образом представляет (вместе с этим объявлением) отдельный объект проекта с тем же интерфейсом.

Примечание — Два архитектурных тела, сопоставленные с разными объявлениями объекта, могут иметь одинаковые простые имена, даже если все они находятся в одной и то же библиотеке.

1.2.1 *Раздел объявлений архитектурного тела*

Раздел объявлений архитектурного тела содержит объявления, доступные для применения внутри блока, определяемого объектом проекта.

```
architecture_declarative_part :: =
    { blok_declarative_item }

blok_declarative_item :: =
    subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | signal_declaration
  | file_declaration
  | alias_declaration
  | component_declaration
  | attribute_declaration
  | attribute_specification
  | configuration_specification
  | disconnection_specification
  | use_clause
```

Все виды объявлений описаны в разделе 4, все виды спецификаций описаны в разделе 5. Описание использования, которое делает внешне описанные имена видимыми внутри блока, дано в разделе 10.

1.2.2 *Раздел операторов архитектурного тела*

Раздел операторов содержит операторы, описывающие внутреннюю организацию и/или функционирование блока, определяемого объектом проекта

```
architecture_statement_part :: =
        [ concurent_statement ]
```

Все операторы, входящие в состав раздела операторов, являются параллельными, выполняющимися асинхронно по отношению друг к другу. Описание параллельных операторов приведено в разделе 9.

Примеры

1 Тело объекта Full_Adder

```
architecture DataFlow of Full_Adder is
  signal A, B: Bit;
begin
  A<= X xor Y;
  B<= A and Cin;
  Sum<= A xor Cin;
  Cout<= B or (X and Y);
end DataFlow;
```

2 Тело объекта TestBench

```
library Test;
  use Test.Components.all
architecture Structure of TestBench is
  component Full_Adder
    port (X, Y, Cin: Bit; Cout, Sum: out Bit);

signal A, B, C, D, E, F, G: Bit;
signal OK: Boolean;

begin

UUT:
  Full_Adder port_map (A, B, C, D, E);
```

```
Generator:
  AdderTest port_map (A, B, C, F, G);
Comparator:
  AdderCheck port_map (D, E, F, G, OK);

end Structure;
```

3 Тело объекта AndGate

```
architecture Behavior of AndGate is
begin
process (Inputs)
      variable Temp: Bit;
  begin
      Temp : = '1';
      for i in Inputs'Range loop
        if Inputs(i) = '0' then
          Temp : = '0';
          exit;
        end if;
      end loop;
      Result <= Temp after 10ns;
    end process;
end Behavior;
```

1.3 Объявления конфигурации

Связывание экземпляров компонентов с объектами проекта осуществляется спецификацией конфигурации (см. 5.2); указанные спецификации применяются в разделе объявлений блока, в котором создаются соответствующие экземпляры компонентов. В определенных случаях возникает потребность оставить связывание экземпляров не заданным в конкретном блоке и отложить его на более поздний срок. Механизм задания таких отложенных связываний обеспечивается объявлением конфигурации.

```
configuration_declaration :: =
  configuration identifier of entity_name is
    configuration_declarative_part
    block_configuration
  end [ configuration_simple_name ];
    configuration_declarative_part :: =
  { configuration_declarative_item }
configuration_declarative_item :: =
  use_clause
  | attribure_specification
```

Имя объекта (entity name) задает имя объявления объекта, определяющее объект проекта, стоящий на вершине иерархии проекта. Для конфигурации конкретного объекта проекта объявление конфигурации и соответствующее объявление объекта должны располагаться в одной и той же библиотеке.

Если в конце объявления конфигурации стоит идентификатор, то он должен повторять идентификатор, стоящий вначале этого объявления.

Примечание — Объявление конфигурации достигает своего эффекта всецело в результате предвыполнения (см. раздел 12). Для объявления конфигурации не существует никаких, связанных с ним, динамических семантик.

Конкретная конфигурация может быть использована для определения другой, более сложной конфигурации.

Пример

- - архитектура микропроцессора:

```
architecture Structure_View of Processor is
  component ALU port (...) end component;
  component MUX port (...) end component;
  component Latch port (...) end component;
begin
  A1: ALU port map (...);
```

```
    M1: MUX port map (...);
    M2: MUX port map (...);
    M3: MUX port map (...);
    L1: Latch port map (...);
    L2: Latch port map (...);
  end Structure_View;

  - - конфигурация микропроцессора:

  library TTL, Work;
  configuration V2_27_87 of Processor is
    use Work.all;
    for StructureView
      for A1: ALU
        use configuration TTL.SN74LS181;
      end for;
      for M1, M2, M3: MUX
        use entity Multiplex4 (Behavior);
      end for;
      for all: Latch
        - - используется неявная конфигурация
      end for;
    end for;
  end V4_27_87;
```

1.3.1 *Конфигурация блока*

Описание конфигурации блока определяет конфигурацию некоторого блока. Таким блоком может быть либо внутренний блок, определяемый оператором блока, либо внешний блок, определяемый объектом проекта.

```
block_configuration :: =
  for block_specification
    { use_clause }
    { configuration_item }
  end for;

block_specification :: =
  architecture_name
  | block_statement_label
  | generate_statement_label [(index_specification)]

index_specification :: =
  discrete_range
  | static_expression

configuration_item :: =
  block_configuration
  | component_configuration
```

Спецификация блока (block specification) идентифицирует внутренний или внешний блок, по отношению к которому употребляется данная конфигурация блока (block configuration).

Если конфигурация блока стоит непосредственно внутри объявления конфигурации (configuration declaration), то спецификация блока в этой конфигурации блока должна быть представлена в виде имени архитектуры, а имя архитектуры должно обозначать тело объекта проекта, интерфейс которого определяется объявлением объекта, обозначаемым именем объекта, стоящим в объемлющем объявлении конфигурации.

Если конфигурация блока стоит непосредственно внутри конфигурации компонента (component configuration), то соответствующий компонент должен быть полностью связан (см. 5.2.1.1), спецификация блока такой конфигурации блока должна быть представлена в виде имени архитектуры, а имя архитектуры должно обозначать то же архитектурное тело, с которым связываются соответствующие компоненты.

Если конфигурация блока стоит непосредственно внутри другой конфигурации блока, то спецификация блока первой должна быть представлена в виде метки оператора блока или оператора генерации, а метка

должна обозначать оператор блока или оператор генерации, заключенный непосредственно внутри блока, обозначаемого спецификацией блока объемлющей конфигурации блока.

Для любого имени, являющегося меткой оператора блока (block statement label), стоящего внутри конкретного блока, соответствующая конфигурация блока может использоваться в виде элемента конфигурации (configuration item) внутри конфигурации блока, соответствующей этому блоку.

Для любого имени, являющегося меткой оператора генерации (generate statement label), стоящего внутри конкретного блока, одна или более соответствующих конфигураций блока могут использоваться в виде элементов конфигурации внутри конфигурации блока, соответствующей этому блоку. Такие конфигурации блока употребляются в отношении неявных блоков, генерируемых вышеуказанным оператором генерации. Если такая конфигурация блока содержит спецификацию индекса (index specification) в виде дискретного диапазона, то эта конфигурация блока употребляется в отношении тех неявных операторов блока, которые генерируются в соответствии с заданным диапазоном значений соответствующего индекса генерации. Если такая конфигурация блока содержит спецификацию индекса в виде статического выражения, то эта конфигурация блока употребляется только для того неявного оператора блока, который генерируется в соответствии с заданным значением соответствующего индекса генерации. Если в такой конфигурации отсутствует спецификация индекса, то она употребляется в отношении всех неявных блоков, генерируемых соответствующим оператором генерации.

Аналогично, для каждого экземпляра компонента, стоящего внутри блока, соответствующего конкретной конфигурации блока, подразумевается использование неявной конфигурации компонента, если явная конфигурация компонента для этого экземпляра отсутствует. Подразумевается также, что вышеуказанные неявные конфигурации блока и компонента в виде элементов конфигурации стоят после всех явных элементов конфигурации в этой конфигурации блока.

Считается ошибкой, если в конкретной конфигурации блока для одного и того же блока или экземпляра компонента определено более одного элемента конфигурации.

Примечание — Исходя из правил, описанных выше и в разделе 10, простое имя, видимое косвенно в конце раздела объявлений конкретного блока, также видимо косвенно в пределах любого элемента конфигурации, заключенного в соответствующую конфигурацию блока. Если такое имя видимо непосредственно в конце конкретного раздела объявлений блока, то оно также будет видимо непосредственно в соответствующих элементах конфигурации, за исключением того случая, когда в соответствующем объявлении конфигурации используется описание использования (use clause) для другого объявления с таким же простым именем, и область действия этого описания охватывает все или часть этих элементов конфигурации. Если такое описание использования применяется, то это имя непосредственно видимо внутри соответствующих элементов конфигурации, за исключением тех мест, которые подпадают под область действия дополнительного описания использования (тех мест, где никакое имя не будет непосредственно видимо).

Если подразумевается, что внутри конфигурации блока используется неявный элемент конфигурации, то этот элемент конфигурации никогда не будет включать в себя явные элементы конфигурации.

Примеры

1 Конфигурация блока для объекта проекта

```
for Work.ShiftReg          - - имя архитектуры
    - -спецификации конфигурации
    - -для блоков и компонентов
    - -внутри ShiftReg
end for;
```

2 Конфигурация блока для оператора блока

```
for B1                     - - метка блока
    - -спецификации конфигурации
    - -для блоков и компонентов
    - -внутри блока B1
end for;
```

1.3.2 *Конфигурация компонента*

Конфигурация компонента определяет конфигурацию одного и более экземпляров компонента в соответствующем блоке.

```
component_configuration :: =
  for component_specification
    [ use binding_indication; ]
    [ block_configuration ]
  end for;
```

Спецификация компонента (component specification) (см. 5.2) идентифицирует экземпляры компонента, в отношении которых данная конфигурация компонента применяется. Конфигурация компонента, стоящая

непосредственно внутри конкретной конфигурации блока, применяется в отношении экземпляров компонента, используемых непосредственно внутри соответствующего блока.

Считается ошибкой, если и явная спецификация конфигурации (в архитектурном теле) и конфигурация компонента, содержащая связывающее указание (binding indication) (в объявлении конфигурации), применяются к одному и тому же экземпляру компонента.

Если конфигурация компонента содержит связывающее указание (см. 5.2.1), то эта конфигурация компонента предполагает спецификацию конфигурацию для экземпляров компонента, к которым она применяется. Неявная спецификация конфигурации имеет ту же самую спецификацию компонента и то же самое связывающее указание, что и сама конфигурация компонента.

Если конкретный экземпляр компонента является несвязанным с соответствующим блоком, то любая явная конфигурация компонента для этого экземпляра, не содержащая явного связывающего указания, будет содержать неявное умалчиваемое связывающее указание (см. 5.2.2). Аналогично, если конкретный экземпляр компонента является не связанным с соответствующим блоком, то любая неявная конфигурация компонента для этого экземпляра будет содержать неявное умалчиваемое связывающее указание.

В пределах конкретной конфигурации компонента (неявной или явной) подразумевается, что существует неявная конфигурация блока для объекта проекта, к которому привязывается соответствующий экземпляр компонента, при условии, что не используется явная конфигурация блока и что соответствующий экземпляр компонента полностью связан.

Примеры

1 Конфигурация компонента с связывающим указанием

```
for all: 10Port
  use entity StdCells.PadTriState4 (StdCells.DataFlow)
    - - для блоков и компонентов
    - - внутри блока B1
  end for;
    port map (Pout=>A, Pin=>B, IO=>Dir, Vdd=>Pwr, Gnd=>Gnd);
end for;
```

2 Конфигурация компонента, содержащая конфигурации блоков

```
for D1: DSP
    - - связывание, заданное в объекте проекта, или неявное for Filterer
          - - спецификации конфигурации для компонентов end for;
          for Processor
          - - спецификации конфигурации для компонентов
    end for;
end for;
```

2 ПОДПРОГРАММЫ И ПАКЕТЫ

Подпрограммы определяют алгоритмы для вычисления значений или представления поведения. Они могут быть использованы как вычислительные ресурсы для конвертирования значений различных типов, для определения функции разрешения выходных значений, задающих общий сигнал, или для определения отдельных частей процесса. Пакет представляет собой средства определения тех или иных ресурсов таким образом, который позволяет различным модулям проекта совместно использовать одни и те же объявления.

Существуют две формы подпрограмм: процедуры и функции. Вызов процедуры является оператором; вызов функции является выражением и возвращает одно значение. Описание подпрограммы может быть задано двумя частями, объявлением подпрограммы, описывающим соглашение по ее вызову, и телом подпрограммы, описывающим ее выполнение.

Пакеты также могут быть описаны двумя частями: объявлением пакета, описывающим видимое содержимое пакета, и телом пакета, описывающим внутренние детали. В частности, тело пакета содержит тела подпрограмм, описанных в объявлении пакета.

2.1 Объявления подпрограмм

Объявление подпрограмм (subprogram declacation) описывает процедуру или функцию в зависимости от начального зарезервированного слова.

```
subprogram_declacation :: =
    subprogram_specification;

subprogram_specification :: =
    procedure designator [(formal_parameter_list)]
```

```
| function designator [(formal_parameter_list)]
    return type_mark

designator :: = identifier | operator_symbol

operator_symbol :: = string_literal
```

Спецификация процедуры задает ее обозначение и ее *формальные параметры*, если они есть. Спецификация функции задает ее обозначение, ее формальные параметры (если они есть) и подтип возвращаемого значения *(тип результата)*.

Обозначение процедуры (designator) — это всегда идентификатор. Обозначение функции - это либо идентификатор, либо символ оператора (operator symbol). Обозначение функции в виде символа оператора используется для совмещения оператора. Последовательность символов, представленных символом оператора, должна иметь вид оператора, принадлежащего к одному из шести классов операторов, определенных в 7.2. При этом пробелы недопустимы, а буквы могут быть прописными или строчными.

Примечание — Все подпрограммы могут вызываться рекурсивно.

2.1.1 *Формальные параметры*

Список формальных параметров (formal parameter list) в спецификации подпрограммы (subprogram specification) описывает формальные параметры этой подпрограммы.

```
formal_parameter_list :: = parameter_interface_list
```

Формальными параметрами подпрограмм могут быть константы, переменные или сигналы. Во всех трех случаях вид параметра определяет способ доступа к этому формальному параметру внутри подпрограммы. Вид формального параметра вместе с его классом может также определять, как такой доступ должен быть реализован.

Единственными допустимыми значениями вида формальных параметров процедуры являются **in**, **out** и **inout**. Если значением вида является **in**, и класс объекта не задан явно, то в качестве последнего подразумевается **constant**. Если значением вида является **out** или **inout**, и класс объекта явно не задан, то в качестве последнего принимается **variable**.

Единственным допустимым значением вида формальных параметров функции является **in** (независимо от того, задан ли он явно или неявно). В качестве класса объекта можно использовать только **constant** или **signal**. Если класс объекта явно не задан, то подразумевается **constant**.

В вызове подпрограммы формальному параметру класса **signal** должен сопоставляться фактический параметр того же класса. Формальному параметру класса **variable** должен сопоставляться фактический параметр того же класса. Формальному параметру класса **constant** должно сопоставляться выражение.

Примечание — Атрибуты фактического параметра в подпрограмму не передаются, поэтому ссылки на атрибут формального параметра допустимы только в том случае, если этот формальный параметр имеет такой атрибут и эти ссылки возвращают значение этого атрибута, сопоставленного с этим формальным параметром.

2.1.1.1 *Передача параметров класс constant и variable*

Для параметров класса **constant** или **variable** в вызов подпрограммы (или из него) пересылаются только значения фактических или формальных параметров. В данном разделе описаны способ таких пересылок, а также связанные с ним привилегии доступа, допускаемые для параметров вышеуказанных классов.

Для параметров скалярного типа или ссылочного типа значение передается копированием. В начале каждого вызова (если режим параметра **in** или **inout**) значение фактического параметра копируется в сопоставленный с ним формальный параметр. После завершения выполнения тела подпрограммы (если режим параметра **out** или **inout**) значение формального параметра копируется обратно в сопоставленный ему фактический параметр.

Для параметров индексируемого или структурного типа передача значения может быть выполнена копированием как для скалярных типов. При передаче копированием параметра с режимом **out** должен копироваться диапазон каждой позиции индекса фактического параметра, а также все его подэлементы. Альтернативно этот эффект может быть достигнут передачей по ссылке, при которой каждое использование формального параметра (с целью чтения или модификации его значения) рассматривается как использование сопоставленного ему фактического параметра на время выполнения вызова подпрограммы. Язык не определяет, какой из этих двух механизмов должен быть адаптирован для передачи параметров, а также используется ли один и тот же механизм при разных вызовах одной и той же подпрограммы. Выполнение подпрограммы является ошибочным, если его эффект зависит от того, какой механизм выбран реализацией.

Для параметра (класса variable) файлового типа язык не определяет четкого механизма передачи значения, но ссылка к формальному параметру должна быть эквивалентна ссылке к фактическому параметру. Считается ошибкой, если элемент сопоставления задает сопоставление фактического параметра формальному параметру файлового типа и этот элемент сопоставления содержит функцию преобразования типа.

Примечание — В пределах тела подпрограммы формальный параметр подчинен любому ограничению, вытекающему из указания подтипа, заданного в спецификации этого параметра. Для формального параметра неограниченного индексируемого типа диапазоны для каждой позиции индекса получаются из фактического параметра, и формальный параметр ограничен этими диапазонами.

Для параметров индексируемого или структурного типов правила передачи значений предполагают, что если к фактическому параметру такого типа есть обращение только по одному каналу, то эффект вызова подпрограммы не зависит от того, используется или нет реализацией копирование для передачи параметров. Если таких каналов много (например, если другому формальному параметру сопоставлен тот же фактический параметр), то значение формального параметра является неопределенным после модификации фактического параметра за счет модификации другого формального параметра. Программа, использующая такое неопределенное значение, является ошибочной

2.1.1.2 *Передача параметров класса signal*

Для формального параметра класса signal в вызов подпрограммы передаются ссылки на сигнал, драйвер этого сигнала или все вместе.

Для параметра класса signal с режимом in или inout фактический сигнал сопоставляется с соответствующим формальным сигналом в начале каждого вызова. С этого момента на время выполнения тела подпрограммы ссылка на формальный сигнал в пределах выражения эквивалентна ссылке на фактический сигнал.

Считается ошибкой, если атрибуты STABLE, QUIET и DELAYED формального сигнала любого режима читаются в пределах подпрограммы.

Оператор процесса содержит драйвер для каждого фактического сигнала, сопоставленного с формальным сигналом вида out и inout в вызове подпрограммы. Аналогично подпрограмма содержит драйвер для каждого формального сигнала режима out или inout, объявленного в спецификации этой подпрограммы.

Для параметра класса signal вида inout или out драйвер фактического сигнала сопоставляется соответствующему драйверу формального параметра в начале каждого вызова. С этого момента на время выполнения тела подпрограммы присваивание драйверу формального сигнала эквивалентно присваиванию драйверу фактического сигнала.

Если фактический сигнал сопоставляется формальному сигналу любого вида, то первый должен обозначаться статическим именем сигнала. Считается ошибкой, если формальная или фактическая часть элемента сопоставления, используемого для сопоставления фактического сигнала формальному сигналу, содержит функцию преобразования типа.

Примечание — В пределах тела подпрограммы формальный сигнал подчинен любому ограничению, вытекающему из указания подтипа, заданного в спецификации параметра. Для формального сигнала неограниченного индексируемого типа границы получаются из фактического сигнала и формальный параметр ограничен этими границами.

Из вышеуказанных правил следует, что процедура с параметром класса signal и видом out или inout, вызываемая процессом, завершается до того, как любое присваивание этому параметру внутри этой процедуры имеет эффект. Присваивания драйверу формального сигнала эквивалентны присваиванием непосредственно драйверу фактического сигнала, содержащемуся в процессе, вызывающем эту процедуру.

2.2 Тела подпрограмм

Тело подпрограммы (subprogram body) задает выполнение подпрограммы.

```
subprogram_body :: =
    subprogram_specification is
        subprogram_declarative_part
    begin
        subprogram_statement_part
    end [ designator ];

subprogram_declarative_part :: =
    { subprogram_declarative_item }

subprogram_declarative_item :: =
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
```

```
    | constant_declaration
    | variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_ declaration
    | attribute_specification
    | use_clause

subprogram_statement_part :: =
    { sequential_statement }
```

Объявление подпрограммы необязательно. При его отсутствии в роли объявления выступает спецификация подпрограммы (subprogram specification) в теле подпрограммы. Для каждого объявления подпрограммы должно быть соответствующее тело. Если задано и объявление подпрограммы и тело подпрограммы, то спецификация подпрограммы в теле подпрограммы должна согласовываться (см. 2.7) со спецификацией подпрограммы в объявлении подпрограммы. Более того, и объявление, и тело подпрограммы должно стоять непосредственно в пределах одной и той же области объявлений.

Если в конце тела подпрограммы стоит обозначение (designator), то оно должно повторять обозначение этой подпрограммы.

Алгоритм, реализуемый подпрограммой, определяется последовательностью операторов, стоящих в разделе операторов подпрограммы (subprogram statement part).

Выполнение подпрограммы активизируется вызовом подпрограммы. Выполнение подпрограммы заключается в выполнении последовательности операторов после предварительной установки соответствия между формальными и фактическими параметрами. После завершения выполнения делается возврат в место вызова (и выполняется необходимое обратное копирование формальных параметров в фактические).

Процесс или подпрограмма считается *родительской* для конкретной процедуры, если этот процесс или подпрограмма содержит оператор вызова процедуры, относящийся к этой процедуре или к родителю этой процедуры.

Если функция является родительской для конкретной процедуры и процедура содержит ссылку на объект класса signal или variable, то объект должен быть описан внутри области объявлений, связанной с этой функцией, или внутри области объявлений, связанной с указанной процедурой. Аналогично, если функция содержит ссылку на объект класса signal или variable, то объект должен быть описан внутри области объявлений, связанной с этой функцией.

Из правил видимости следует, что объявление подпрограммы обязательно, если вызов этой подпрограммы текстуально располагается перед телом подпрограммы, и это объявление само должно стоять перед вызовом.

Вышеуказанные правила в отношении функций, а также тот факт, что параметры функции могут быть только вида in, предполагают, что функция не имеет никакого другого эффекта, кроме вычисления возвращаемого значения. Следовательно, гарантируется, что функция, активизируемая явно в процессе преобразования объявления, или функция, активизируемая неявно во время цикла моделирования, не оказывает никакого эффекта на другие объекты в описании.

2.3 Совмещение подпрограмм

Считается, что два списка формальных параметров имеют (formal parameter list) один и тот же *профиль* типа *параметров*, если и только если они содержат одинаковое количество параметров и в каждой позиции списков соответствующие параметры имеют один и тот же базовый тип. Считается, что две подпрограммы имеют один и тот же *профиль типа параметров* и *результата*, если и только если обе они имеют один и тот же профиль типа параметров и, либо обе они являются функциями с одинаковым базовым типом результата, либо ни одна из них не является функцией.

Конкретное обозначение подпрограммы может быть использовано в нескольких спецификациях подпрограмм. В этом случае это обозначение считается совмещенным; подпрограммы, имеющие такое обозначение, также считаются совмещенными и, следовательно, совмещаются друг с другом. Если две подпрограммы совмещаются друг с другом, одна из них может скрыть другую при единственном условии, что обе подпрограммы имеют одинаковый профиль типа параметров и результата.

Вызов совмещенной подпрограммы является неоднозначным (и следовательно недопустимым), если имя подпрограммы, количество сопоставлений параметров, типы и порядок фактических параметров, имена формальных параметров (при использовании именованных сопоставлений) и тип результата (для функций) не позволяют идентифицировать точно одну (совмещенную) спецификацию подпрограммы.

Примеры

```
1 - - Объявления совмещенных подпрограмм:
      procedure Write (F: inout Text; Value: Integer);
      procedure Write (F: inout Text; Value: String);

      procedure Check (Setup: Time; signal D: Data;
                       signal C: Clock);
      procedure Check (Hold: Time; signal C: Clock;
                       signal D: Data);

2 - - Вызовы совмещенных подпрограмм:
      Write (Sys_Output, 12);

      Write (Sys_Error, "Фактический выход не соответствует ожидаемому");
      Check (Setup=>10ns, D=>Bus, C=>Clk1);

      Check (Hold=>5ns, D=>Bus, C=>Clk2);
      Check (15ns, Bus, Clk);    - - двусмысленно, если
                                 - - Data'Base = Clock'Base
```

Примечание — В понятие профиля типа параметров и результата не входят имена параметров, классы параметров, вид параметров, подтипы параметров или неявные выражения (а также их наличие или отсутствие).

Неоднозначности могут (но необязательно) возникать, когда фактические параметры вызова совмещенной подпрограммы сами являются вызовами совмещенных функций, литералами или агрегатами. Неоднозначность может (но необязательно) также возникнуть, когда несколько совмещенных подпрограмм, принадлежащих различным пакетам, оказываются видимыми. Такие неоднозначности могут быть разрешены двумя способами: для некоторых или всех фактических параметров и для результата (если он есть) может быть использовано квалифицированное выражение; или имя подпрограммы может быть выражено более точно как расширенное имя (см. 6.3).

2.3.1 *Совмещение операций*

Объявление функции, обозначение которой является символом оператора, используется для совмещения оператора. Последовательность символов, представляющая знак оператора, должна совпадать с одним из операторов, принадлежащих к одному из шести классов операторов, определенных в 7.2.

Спецификация подпрограммы унарного оператора должна иметь единственный параметр. Спецификация подпрограммы бинарного оператора должна иметь два параметра; в каждом использовании этого оператора первый параметр выступает в роли левого операнда, а второй - в роли правого операнда.

Для операторов "+" и "—" допускается совмещение - как унарного, так и бинарного операторов.

Примечание — Совмещение оператора равенства не влияет на выборку альтернатив в операторе выбора или в операторе выборочного назначения сигнала.

Совмещение операторов, выполняющихся по укороченной схеме, таких как *and*, не подразумевает, что функция, обозначенная таким оператором, будет активизироваться по такой же схеме.

Функции, совмещающие символы операторов, могут также вызываться с использованием обычной нотации вызова функции в отличие от нотации оператора.

Примеры

```
1 type MVL is ('0', '1', 'Z', 'X')

2 function "and" (L, R: MVL) return MVL;
  function "or" (L, R: MVL) return MVL;
  function "not" (R: MVL) return MVL;

3 signal Q, R, S: MVL;

4 Q <= 'X' or '1';
  R <= "or" ('O', 'Z');
  S <= (Q and R) or not S;
```

2.4 Функции разрешения

Функция разрешения — это функция, которая определяет, каким образом значения нескольких источников конкретного сигнала должны быть разрешены в отдельное значение для этого сигнала. Функции разрешения сопоставляются сигналам, требующим разрешения, включением имени функции

разрешения в объявления этих сигналов или объявления их подтипов. Сигнал с сопоставленной ему функцией разрешения называется разрешенным сигналом (см. 4.3.1.2).

Функция разрешения должна иметь один входной параметр в виде одномерного неограниченного массива, тип элементов которого совпадает с типом элементов разрешенного сигнала. Подтип индекса этого массива должен удовлетворять числу источников любого сигнала, разрешаемого этой функцией. Тип возвращаемого значения функции также должен совпадать с типом сигнала.

Функция разрешения, сопоставленная разрешенному сигналу, определяет *разрешенное значение* этого сигнала как функцию совокупности входов от совокупности их источников. Если разрешенный сигнал относится к некоторому составному типу и подэлементам этого типа также сопоставлены функции разрешения, то последние не оказывают никакого эффекта на процесс определения разрешенного значения этого сигнала.

Функции разрешения активизируются неявно в течение каждого цикла моделирования, в котором соответствующие разрешенные сигналы активны (см. 12.6.1). Каждый раз, когда активизируется функция разрешения, ей передается значение индексируемого типа, каждый элемент которого определяется соответствующим источником разрешенного сигнала, за исключением тех источников, которые являются драйверами, значения которых определяются пустыми транзакциями (см. 8.3.1.). Такие драйверы называются *отключенными*. Для определенных активизаций (особенно для тех, которые влекут разрешение источников сигнала, объявленного как шина) функция разрешения может быть активизирована с входным параметром в виде пустого массива; такое случается, когда все источники шины являются драйверами и все они отключены. В этом случае функция разрешения должна возвращать значение, представляющее такое состояние шины.

Пример

```
function WIRED_OR (Inputs: BIT_VECTOR) return BIT
    is constant FloatValue: Bit := '0';
begin
    if Inputs'Length = 0 then
            - - случай, когда все драйверы шины являются
                отключенными
            return   FloatValue;
    else
            for I in Inputs'Range loop
                if Inputs (I) = '1' then
                    return '1';
                end if;
            end loop;
            return '0';
    end if;
end;
```

2.5 Объявления пакетов

Объявление пакета (package declaration) определяет интерфейс для пакета. Область действия объявления внутри пакета может быть расширена на другие модули проекта.

```
package_declaration :: =
    package identifier is
        package_declarative_part
    end [package_simple_name];

package_declarative_part :: =
    {package_declarative_item}

package_declarative_item :: =
    subprogram_declaration
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | signal_declaration
    | file_declaration
    | alias_declaration
    | component_declaration
    | attribute_declaration
```

```
    | attribute_specification
    | disconnection_specification
    | use_clause
```

Если в конце объявления пакета стоит простое имя (simple name), то оно должно повторять идентификатор этого объявления.

Элементы, описанные внутри объявления пакета, становятся видимыми косвенно внутри конкретного модуля проекта везде, где имя этого пакета видимо в этом модуле. Указанные элементы могут также быть непосредственно видимыми при помощи соответствующего описания использования (см. 10.4).

Примечание — Наличие тела пакета для всех пакетов необязательно. В частности, тело пакета необязательно, если в объявлении пакета не содержатся описания подпрограмм или неполных констант.

Подпрограмма, написанная на другом языке, может быть доступна заданием ее интерфейса в виде описания подпрограммы внутри объявления пакета, не имеющего соответствующего тела. Тело такой подпрограммы должно быть сопоставлено объявлению ее интерфейса способом, зависящим от реализации. Например, встроенные функции, обеспечиваемые конкретной моделирующей программой, могли бы быть объявлены именно таким способом. Предполагается, что подпрограммы на других языках, объявленные таким путем, реализуют семантику, подразумеваемую их описаниями интерфейсов.

Примеры

1 Объявление пакета, не требующее тела пакета:

```
package TimeConstants is
    constant tPLH:    Time : = 10ns;
    constant tPHL:    Time : = 12ns;
    constant tPLZ:    Time : = 7ns;
    constant tPZL:    Time : = 8ns;
    constant tPHZ:    Time : = 8 ns;
    constant tPZH:    Time : = 9ns;
end TimeConstants;
```

2 Объявление пакета, которое может иметь тело пакета

```
package TriState is
    type Tri is ('0', '1', 'Z', 'E');

    function BitVal (Value: Tri) return Bit;
    function TriVal (Value: Bit) return Tri;
    type TriVector is array (Natural range < >) of Tri;
    function Resolve (Sources: TriVector) return Tri;
end TriState;
```

2.6 Тела пакетов

Тело пакета (package body) определяет тела подпрограмм или значения неполных констант, описанных в объявлении пакета.

```
package body :: =
    package body package_simple_name is
        package_body_declarative_part
    end [package_simple_name];

package_body_declarative_part :: =
    {package_body_declarative_item}

package_body_declarative_item :: =
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | file_declaration
    | alias_declaration
    | use_clause
```

Простое имя (simple name), стоящее в начале тела пакета, должно повторять идентификатор пакета. Если в конце тела пакета стоит простое имя, то оно должно совпадать с идентификатором в объявлении пакета. Наряду с такими элементами как объявление константы тело пакета может содержать другие объявления, способствующие определению тел подпрограмм, описанных в объявлении пакета. Элементы, объявленные в теле пакета, не могут быть видимыми вне этого тела.

Если конкретное объявление пакета содержит описание неполной константы (см. 4.3.1.1), то в соответствующем теле пакета должно появиться описание константы с тем же идентификатором. Такое описание объекта называется *полным* описанием неполной константы. Указание подтипа в полном описании должно согласовываться с указанием подтипа в неполном описании константы.

В пределах объявления пакета, содержащего описание неполной константы, и внутри тела этого пакета до окончания соответствующего полного описания использование имени, обозначающего отложенную константу, допускается только в неявных выражениях для локальных параметров настройки, локальных портов или формальных параметров. Результат вычисления выражения, ссылающегося на неполную константу до предвыполнения соответствующего полного описания, языком не определен.

Пример —

```
package body TriSlate is
    function BitVal (Value: Tri) return Bit is
        constant Bits: Bit_Vector : = "0100";
    begin
        return Bits (Tri'Pos(Value));
    end;

    function TriVal (Value: Bit) return Tri is
    begin
        return Tri'Val (Bit'Pos(Value));
    end;

function Resolve (Sources: TriVector) return Tri is
    Variable V: Tri : = 'Z';
begin
    for i in Sources'Range loop
        if Sources(i) /= 'Z' then
            if V = 'Z' then
               V : = Sources(i);
            else
               return 'E';
            end if;
        end if;
    end loop
    return V;
  end;

end TriState;
```

2.7 Правила согласования

Всякий раз, когда правила языка либо требуют, либо допускают появление спецификации конкретной подпрограммы более чем в одном месте, в каждом таком месте разрешены следующие вариации:

1) Числовой литерал может быть замещен другим числовым литералом, если и только если оба они имеют одинаковое значение.

2) Простое имя может быть заменено расширенным именем, в котором это простое имя используется как суффикс, если и только если в обоих случаях смысл простого имени определяется одним и тем же объявлением.

Считается, что две спецификации подпрограммы согласуются, если (исключая комментарии и вышеуказанные допустимые вариации) обе они представлены одной и той же последовательностью лексических элементов и соответствующим лексическим элементам правилами видимости дается одинаковый смысл.

Согласование также определено для указаний подтипа в объявлениях неполных констант.

Примечание — Простое имя может быть заменено расширенным именем, даже если это простое имя является префиксом составного имени. Например, составное имя Q.R может быть заменено на P.Q.R, если Q объявлено непосредственно внутри P.

Следующий пример содержит спецификации, которые не согласуются из-за того, что не представлены одной и той же последовательностью лексических элементов.

```
procedure P (X, Y: INTEGER);
procedure P (X: INTEGER; Y: INTEGER);
procedure P (X, Y : in INTEGER);
```

3 ТИПЫ

В данной главе описаны различные категории типов, предусматриваемые языком, а также особые типы, которые являются предопределенными. Описание всех предопределенных типов содержится в пакете STANDARD, объявление которого содержится в разделе 14.

Тип характеризуется множеством значений и множеством операций. Множество операций над типом подразумевает явно описанные подпрограммы, имеющие параметры или результат этого типа. В это множество входят также предопределенные операторы (см. 7.2). Операции, обусловленные этими операторами, становятся неявно объявленными для конкретного описания типа сразу после этого описания независимо от того, имеются ли за этим описанием другие явные описания.

Имеются четыре класса типов. *Скалярные* типы — это целые, плавающие, физические типы и типы, определяемые перечислением их значений; значения этих типов не имеют элементов. *Составные* типы — это индексируемые и структурные типы; значения этих типов состоят из значений элементов. *Ссылочные* типы обеспечивают доступ к объектам конкретного типа. *Файловые* типы обеспечивают доступ к объектам, содержащим последовательность значений конкретного типа.

Множество возможных значений объекта конкретного типа может подчиняться условию, называемому *ограничением* (сюда также входит случай, когда ограничение не устанавливает никаких границ); считается, что значение *удовлетворяет* ограничению, если оно удовлетворяет соответствующему условию. *Подтип* — это тип вместе с ограничением; считается, что значение *принадлежит подтипу* конкретного типа, если оно принадлежит этому типу и удовлетворяет ограничению; этот тип называется *базовым типом* этого подтипа. Тип является подтипом самого себя; такой подтип называется *неограниченным;* он связан с условием, не налагающим никаких границ. Базовым типом типа является сам тип.

Множество операций, определенных для подтипа конкретного типа, включает в себя операции, определенные для этого типа; однако операции присваивания для объекта, имеющего конкретный подтип, действительны только для тех значений, которые принадлежат этому подтипу. Дополнительные операции, такие как квалифицирование (в квалифицированном выражении), явно определяются описанием подтипа.

Термин *подэлемент* используется в настоящем стандарте вместо термина элемент для обозначения либо элемента, либо элемента другого элемента или подэлемента. В местах, где другие подэлементы исключены, используется термин элемент.

Конкретный тип не должен иметь подэлементы, тип которых это сам тип. Имя класса типов в данном руководстве используется как квалификатор для объектов и значений, имеющих тип рассматриваемого класса. Например, термин "индексируемый объект" используется для объектов, типом которых является индексируемый тип; аналогично, термин "ссылочное значение" используется для значения ссылочного типа.

Примечание — Множество значений подтипа — это подмножество значений базового типа. Подмножество не обязано быть собственным подмножеством.

3.1 Скалярные типы

Скалярные типы включают в себя перечисляемые *типы, целые типы, физические типы и плавающие типы.* Перечисляемые и целые типы называются *дискретными типами.* Целые, плавающие и физические типы называются *числовыми типами.* Все скалярные типы упорядочены; другими словами, все операции отношения предопределены для их значений. Каждое значение дискретного или физического типа имеет номер позиции в виде целого значения.

```
scalar_type_definition :: =
        enumeration_type_definition
      | integer_type_definition
      | floating_type_definition
      | physical_type_definition

range_constraint :: = range range

range :: = range_attribute_name
    | simple_expression direction simple_expression

direction :: = to | downto
```

Диапазон (range) задает подмножество значений скалярного типа. Диапазон является *пустым* диапазоном, если заданное подмножество пусто.

Диапазон **L to R** называется *восходящим* диапазоном; если L > R, то этот диапазон пуст. Диапазон **L downto R** называется *нисходящим* диапазоном; если L < R, то этот диапазон пуст. Меньшая из величин L и R называется *нижней границей*, а большая — *верхней границей* диапазона.

Считается, что значение V *принадлежит диапазону*, если отношения (нижняя граница <= V) и (V <= верхняя граница) верны и сам диапазон не является пустым. Операторы >, < и <= в вышестоящих определениях являются предопределенными операторами применимого скалярного типа.

Считается, что значение V1 стоит левее значения V2 внутри конкретного диапазона, если оба они принадлежат этому диапазону и либо V2 следует за V1, если диапазон является восходящим; либо V2 предшествует V1, если диапазон является нисходящим. Последовательность значений конкретного диапазона является *упорядоченной слева направо*, если каждое значение в этой последовательности стоит левее следующего значения в этой последовательности внутри этого диапазона (за исключением последнего значения).

Если ограничение диапазона (range constraint) используется в указании подтипа, то тип выражений (а также тип границ атрибута-диапазона) должен совпадать с базовым типом, задаваемым обозначением типа в этом указании подтипа. Ограничение диапазона *совместимо* с подтипом, если каждая граница этого диапазона принадлежит этому подтипу, или это ограничение диапазона определяет пустой диапазон. В противном случае ограничение диапазона несовместимо с подтипом.

Направление ограничения диапазона совпадает с направлением диапазона, используемого в этом ограничении.

Примечание — Правила индексирования и итерации используют значения дискретных типов.

3.1.1 *Перечисляемые типы*

Описание перечисляемого типа определяет перечисляемый тип.

```
enumeration_type_definition :: =
        (enumeration_literal, {, enumeration_literal})

enumeration_literal :: = identifier | character_literal
```

Идентификаторы и символьные литералы, задаваемые описанием перечисляемого типа, не должны повторяться. Каждая спецификация литерала перечисления (enumeration_literal) является объявлением соответствующего литерала перечисления.

Перечисляемый тип будет рассматриваться как символьный тип, если хотя бы один из его литерала перечисления представлен символьным литералом.

Каждый литерал перечисления порождает уникальное перечисляемое значение. Предопределенная зависимость следования между перечисляемыми значениями вытекает из порядка соответствующих *номеров* позиций. Номер позиции значения первого перечисляемого литерала в списке есть 0; номер позиции каждого последующего литерала перечисления в этом списке на 1 больше, чем номер позиции предшествующего литерала.

Если один и тот же идентификатор или символьный литерал задан в более чем одном описании перечисляемого типа, то соответствующие литералы считаются *совмещенными*. В любом месте, где встречается совмещенный литерал, тип этого литерала должен быть определяемым из контекста.

Каждое описание перечисляемого типа определяет восходящий диапазон.

Примеры

```
1 type MULTI_LEVEL_LOGIC is (LOW, HIGT, RISING, FALLING, AMBIGUOUS);

2 type BIT is ('1', '0');

3 type SWITCH_LEVEL is ('0', '1', 'X'); — совмещение
                                          '0' и '1'
```

3.1.1.1 *Предопределенные перечисляемые типы*

Предопределенными перечисляемыми типами являются CHARACTER, BIT, BOOLEAN и SEVERITY_LEVEL.

Предопределенный тип CHARACTER является символьным типом, значениями которого являются 128 символов ASCII. Каждый из 95 графических символов в этом множестве обозначается соответствующим символьным литералом.

Объявления предопределенных типов CHARACTER, BIT, BOOLEAN и SEVERITY_LEVEL содержатся в пакете STANDARD, описанном в разделе 14.

Примечание — Нетрафические элементы предопределенного типа CHARACTER представлены аббревиатурой, соответствующей аббревиатуре непечатных символов ASCII (за исключением тех, которые указаны в разделе 14)

Тип BOOLEAN может быть использован для описания логических схем как с высоким, так и с низким активным уровнем в зависимости от конкретных выбранных функций конвертирования в или из типа BIT

3.1.2 *Целые типы*

Описание целого типа определяет целый тип, множество значений которого включает множество из заданного диапазона.

integer_type_definition :: = range_constrait

Описание целого типа (integer type definition) определяет как тип, так и подтип этого типа. Сам тип является анонимным типом, диапазон которого устанавливается реализацией этот диапазон должен полностью покрывать диапазон, заданный в описании целого типа. Подтип — это именованный подтип указанного анонимного базового типа, при этом в качестве имени этого подтипа используется имя, заданное в соответствующем объявлении типа, а диапазон подтипа — это заданный диапазон.

Каждая граница в ограничении диапазона, используемого в описании целого типа, должна быть представлена в виде локального статического выражения некоторого целого типа, но при этом необязательно, чтобы обе границы имели один и тот же целый тип (допускаются и отрицательные границы).

Целые литералы — это литералы анонимного предопределенного типа, называемого в данном руководстве как *универсальный целый*. Другие целые типы не имеют литералов. Однако для каждого целого типа существует неявное преобразование, которое преобразует значение типа *универсальный целый* в соответствующее значение (если оно есть) этого целого типа (см. 7.3.5).

Номер позиции целого значения — это соответствующее значение типа *универсальный целый*.

Для всех целых типов предопределены одни и те же арифметические операции (см. 7.2). Считается ошибкой, если выполнение такой операции (в частности, неявное преобразование) не может привести к получению правильного результата (то есть, если значение, относящееся к математическому результату, не является значением целого типа).

Пределы ограничения диапазона целых типов, отличных от *универсального целого*, зависят от реализации. Реализация должна допускать объявление любого целого типа, диапазон которого находится в пределах от минус 2147483647 до плюс 2147483647 включительно.

Примеры

```
1 type TWOS_COMPLEMENT_INTEGER is range —32768 to 32767;
2 type BYTE_LENGTH_INTEGER is range 0 to 255;
3 type WORD_INDEX is range 31 downto 0;
4 subtype HIGTH_BIT_LOW is BYTE_LENTH_INTEGER range 0 to 127;
```

3.1.2.1 *Предопределенные целые типы*

Единственным предопределенным целым типом является тип INTEGER. Диапазон типа INTEGER зависит от реализации, но при этом гарантируется, что он имеет пределы от минус 2147483647 до плюс 2147483647. Тип определен с восходящим диапазоном.

Примечание — Диапазон типа INTEGER в конкретной реализации может быть выявлен при помощи атрибутов 'LOW и 'HIGH.

3.1.3 *Физические типы*

Значения физического типа представляют измерения некоторой величины. Любое значение физического типа — это целочисленное значение, кратное базовой единице измерения для этого типа.

```
physical_type_definition :: =
    range_constraint
      units
        base_unit_declaration
        { secondary_unit_declaration }
      end units;

base_unit_declaration :: = identifier;

physical_literal :: = [ abstract_literal ] unit_name
```

Описание физического типа (physical type definition) определяет как тип, так и подтип этого типа. Сам тип является анонимным типом, диапазон которого устанавливается реализацией; этот диапазон

должен полностью покрывать диапазон, заданный в описании физического типа. Подтип - это именованный подтип указанного анонимного базового типа, при этом в качестве имени этого подтипа используется имя, заданное в соответствующем объявлении типа, а диапазон подтипа — это заданный диапазон.

Каждая граница в ограничении диапазона, используемого в описании физического типа, должна быть представлена в виде локального статического выражения некоторого целого числа, но при этом необязательно, чтобы обе границы имели один и тот же целый тип (допускаются и отрицательные границы).

Каждое объявление единицы (либо базовой, либо вторичной) определяет *имя единицы*. Имена единиц, стоящие в объявлениях вторичных единиц (secondary unit declaration), должны быть явно или неявно определены в терминах целочисленных, кратных базовой единице объявления типа, в котором они стоят.

Абстрактный литерал (если задан), являющийся составной частью физического литерала, стоящего в объявлении вторичной единицы, должен быть представлен в виде целого литерала.

Физический литерал, состоящий исключительно из имени единицы, эквивалентен 1 (целого типа), за которой следует это имя.

Для каждого значения физического типа существует соответствующей номер позиции. Номер позиции значения, соответствующего имени единицы, — это количество базовых единиц, представляемых этим именем единицы. Номер позиции значения, соответствующего физическому литералу, имеющему своей составной частью абстрактный литерал, — это наибольшее целое, не большее произведения этого абстрактного литерала на номер позиции сопутствующего имени единицы.

Для всех физических типов предопределены одни и те же арифметические операции (см. 7.2). Считается ошибкой, если выполнение такой операции не может привести к правильному результату (то есть, если значение, относящееся к математическому результату, не является значением физического типа).

Пределы ограничения диапазона физических типов зависят от реализации. Реализация должна допускать объявление любого физического типа, диапазон которого находится в пределах от минус 2147483647 до плюс 2147483647 включительно.

Примеры

```
1 type TIME is range —1E18 to 1E18
        units
            fs;              - - фемтосекунда
            ps = 1000 fs; - - пикосекунда
            ns = 1000 ps; - - наносекунда
            us = 1000 ns; - - микросекунда
            ms = 1000 us; - - миллисекунда
            sec = 1000 ms; - - секунда
            min = 60 sec; - - минута
end units;

2 type DISTANCE is range 0 to 1E16
        units
        - - базовая единица:
            A;                    - - ангстрем
        - - меры длины:
            nm = 10 A;          - - нанометр
            um = 1000 nm;      - - микрометр или микрон
            mm = 1000 mm;      - - миллиметр
            cm = 10 mm;        - - сантиметр
            m = 1000 mm;        - - метр
            km = 1000 m;        - - километр
        - - Английские меры длины:
            mil = 254000 A;      - - миллидюйм
            inch = 1000 mil;     - - дюйм
            ft = 12 iuch;          - - фут
            yd = 3 ft;             - - ярд
            fm = 6 ft;             - - сажень
            mi = 5280 ft;          - - миля
            lg = 3 mi;             - - лига
        end units;
```

```
3 x: distance; y: time; z: integer;
  x : = 5A+13ft—27inch;
  y : = 3ns+5min
  z ; = ns/ps;
  x : = y/10;
```

Примечание — Следствием вышестоящих определений является следующее: если L не входит в диапазон, заданный описанием физического типа, то имя базовой единицы, стоящее отдельно, не является допустимым литералом физического типа.

Для преобразований между абстрактными значениями и физическими значениями могут быть использованы атрибуты POS и VAL.

3.1.3.1 *Предопределенные физические типы*

Единственным предопределенным физическим типом является тип TIME. Диапазон этого типа зависит от реализации, но при этом гарантируется, что он находится в пределах от минус 2147483647 до плюс 2147483647. Тип определен с восходящим диапазоном. Все спецификации задержек должны быть типа TIME. Объявление типа TIME содержится в пакете STANDARD (см. раздел 14).

По умолчанию базовая единица типа TIME (1 фемтосекунда) является *пределом разрешения* для типа TIME. Любое значение типа TIME меньшее, чем этот предел, усекается до 0 временных единиц. Реализация может допускать, чтобы при конкретном выполнении моделирования (см. 12.6) в качестве предела расширения была выбрана вторичная единица типа TIME. Более того, реализация может ограничивать точность представления значений типа TIME и результатов выражений типа TIME при условии, что значения, имеющие такую же малую величину, что и предел разрешения, могут быть представлены в рамках этого ограничения. Считается ошибкой, если конкретная единица типа TIME стоит в любом месте внутри иерархии проекта, определяющей модель для исполнения, и номер позиции этой единицы меньше, чем номер позиции вторичной единицы, выбранной в качестве предела разрешения для типа TIME в течение выполнения этой модели.

Примечание — Выбирая вторичную единицу типа TIME в качестве предела разрешения для того типа, можно достичь более длительного периода времени моделирования с меньшей точностью или наоборот, более короткого периода времени моделирования с более высокой точностью.

3.1.4 *Плавающие типы*

Плавающие типы обеспечивают приближение к действительным числам. Плавающие типы имеют пользу в моделях, в которых точная характеризация вычисления с плавающей точкой не имеет особой важности или решающего значения.

```
floating_type_definition :: = range_constraint
```

Описание плавающего типа (floating_type_definition) определяет как тип, так и подтип этого типа. Сам тип является анонимным типом, диапазон которого определяется реализацией; этот диапазон должен полностью покрывать диапазон, заданный в описании плавающего типа. Подтип - это именованный подтип указанного анонимного базового типа, при этом в качестве имени этого подтипа используется имя, заданное в соответствующем описании типа, а диапазон подтипа — это заданный диапазон.

Каждая граница в ограничении диапазона, используемого в описании плавающего типа, должна быть представлена в виде локального статического выражения некоторого плавающего типа, но при этом необязательно, чтобы обе границы имели один и тот же плавающий тип (допускаются и отрицательные границы).

Плавающие литералы — это литералы анонимного предопределенного типа, называемого в данном руководстве как *универсальный действительный*. Другие плавающие типы не имеют литералов. Однако для каждого плавающего типа существует неявное преобразование, которое преобразует значение типа *универсальный действительный* в соответствующее значение (если оно есть) этого плавающего типа (см. 7.3.5).

Для всех плавающих типов предопределены одни и те же арифметические операции (см. 7.2). Считается ошибкой, если выполнение такой операции не приводит к получению правильного результата (то есть, если значение, относящееся к математическому результату, не является значением плавающего типа). В случае операций над плавающими типами к реализации не предъявляются требования к обнаружению указанной выше ошибки, так как выявление состояний переполнения, возникающих в результате операций с плавающими типами, для большинства машин является трудновыполнимым.

Пределы ограничения диапазона плавающих типов, отличных от *универсального действительного*, зависят от реализации. Реализация должна допускать объявление любого плавающего типа, диапазон которого находится в пределах от минус 1E38 до плюс 1E38 включительно. Представление плавающего типа должно включать как минимум шесть десятичных цифр точности.

3.1.4.1 *Предопределенные плавающие типы*

Единственным предопределенным плавающим типом является тип REAL. Диапазон типа REAL зависит от применяемой машины, но при этом гарантируется, что он имеет пределы от минус 1E38 до плюс 1E38 включительно. Тип определен с восходящим диапазоном.

Примечание — Диапазон типа REAL в конкретной реализации может быть выявлен при помощи атрибутов 'LOW и 'HIGH.

3.2 Составные типы

Составные типы используются для определения собраний значений. Они включают в себя как массивы значений (собрание значений однородного типа), так и структуры значений (собрание значений потенциально неоднородных типов).

```
composite_type_definition :: =
    | array_type_definition
    | record_type_definition
```

Объект составного типа представляет собой собрание объектов, каждый из которых представляет собой один элемент этого составного объекта. Составной тип может содержать элементы только скалярного, составного или ссылочного типов; элементы файловых типов в составном типе недопустимы. Таким образом, объект составного типа в конечном счете представляет собой собрание объектов скалярного или ссылочного типа, один для каждого несоставного подэлемента этого составного объекта.

3.2.1 *Индексируемые типы*

Индексируемый объект является составным объектом, состоящим из элементов, имеющих один и тот же подтип. Имя элемента массива используется вместе с одним или более значениями индексов, принадлежащих к заданному дискретному типу. Значением индексируемого объекта является составное значение, состоящее из значений его элементов.

```
array_type_definition :: =
    unconstrained_array_definition
    | constrained_ARRAY_definition
unconstrained_array_definition :: =
    array ( index_subtype_definition)
            { index_subtype_definition } )
    of element_subtype_indication

constrained_array_definition :: =
    array index_constraint of element_subtype_
                                        indication

index_subtype_definition :: =
    type_mark range < >

index_constraint :: = (discrete_range {,discrete_range})

discrete_range :: = discrete_subtype_indication
    | range
```

Индексируемый объект характеризуется числом индексов (размерностью массива), типом, позицией и диапазоном каждого индекса, а также типом и возможными ограничениями его элементов. Порядок индексов имеет значение.

Одномерный массив имеет индивидуальный элемент для каждого возможного значения индекса. Многомерный массив имеет индивидуальный элемент для каждой возможной последовательности индексов, которая может быть составлена выборкой единственного значения для каждого индекса (в заданном порядке). Возможные значения для конкретного индекса — это все значения, принадлежащие соответствующему диапазону; этот диапазон значений называется *диапазоном индекса.*

Описание неограниченного индексируемого типа (unconstrained array definition) определяет индексируемый тип и имя, обозначающее этот тип. Для каждого объекта, имеющего такой тип, количество индексов, тип и позиция каждого индекса, а также подтип элементов такие же, как в описании типа. *Подтип индекса* для конкретной позиции индекса — это (по определению) подтип, обозначенный меткой типа (type mark) в соответствующем определении подтипа индекса (index subtype definition). Значения левой и правой границ каждого диапазона индекса не определены, но должны принадлежать к соответствующему подтипу индекса; аналогично, направление каждого диапазона индекса не определено. Символ < > (называемый *боксом*), стоящий в описании подтипа индекса, указывает на неопределенный

диапазон (различные объекты этого типа необязательно должны иметь одинаковые границы и направление диапазона).

Описание ограниченного индексируемого типа (constrained array definition) определяет как индексируемый тип, так и подтип этого типа:

1) Индексируемый тип является неявно объявленным анонимным типом; этот тип определяется (неявным) описанием неограниченного индексируемого типа, в котором указание подтипа элементов (element subtype indication) копируется из описания ограниченного индексируемого типа, и в котором метка типа каждого описания подтипа индекса обозначает подтип, определенный соответствующим дискретным диапазоном (discrete range).

2) Индексируемый подтип — это подтип, полученный наложением ограничения индекса (index constraint) на этот индексируемый тип.

Если для объявления типа используется описание ограниченного индексируемого типа, то простое имя, описываемое этим объявлением, обозначает индексируемый подтип.

Направление дискретного диапазона совпадает с направлением диапазона (range) или указания дискретного подтипа (discrete subtype indication), которые определяют этот дискретный диапазон.

Примеры

1 - - Примеры объявлений ограниченных массивов

а) type MY_WORD is array (0 to 31) of BIT;
- - слово памяти с восходящим диапазоном

б) type DATA_IN is array (7 downto 0) of FIVE_LEVEL_LOGIC;
- - входной порт с нисходящим диапазоном

2 - - Пример объявлений неограниченных массивов

type MEMORY is array (INTEGER range < >) of MY_WORD;
- - массив памяти

3 - - Примеры объявлений индексируемых объектов

а) signal DATA_LINE : DATA_IN;
- - определяет входную шину данных

б) variable MY_MEMORY : MEMORY (0 to 2**n — 1);
- - определяет память из 2**n 32-разрядных слов

Примечание — Правила в отношении объявлений ограниченных типов означают, что объявление ограниченного индексируемого типа такое, как

```
type T is array (POSITIVE range MIN to MAX) of ELEMENT;
```

эквивалентно следующей последовательности объявлений:

```
subtype index_subtype is POSITIVE range MIN to MAX;
type array_type is array (index_subtype range < >) of
                                        ELEMENT;

subtype T is array_type (index_subtype);
```

где *index_subtype* и *array_type* являются анонимными. Следовательно, T — это имя подтипа и все объекты, объявленные с этой меткой типа, являются массивами, имеющими один и тот же диапазон индекса.

3.2.1.1 *Ограничения индекса и дискретные диапазоны*

Ограничение индекса определяет диапазон индекса для каждого индекса в индексируемом типе и тем самым границы соответствующего массива.

Для дискретного диапазона, границы которого заданы в виде числовых литералов или атрибутов, и используемого в описании ограниченного массива, подразумевается неявное преобразование значения каждой границы в предопределенный тип INTEGER, а типом обоих границ (до неявного преобразования) является тип *универсальный целый*. В противном случае обе границы должны быть одного и того же дискретного типа, отличного от типа *универсальный целый;* этот тип должен быть определяемым независимо от контекста, но с учетом того факта, что он должен быть дискретным и что обе границы должны иметь один и тот же тип. Эти правила применимы также к дискретному диапазону, используемому в схеме итерации или в схеме генерации.

Если ограничение индекса стоит после метки типа в указании подтипа, то тип или подтип, обозначаемый этой меткой, не должен, в свою очередь, налагать ограничение индекса. Метка типа должна обозначать либо неограниченный индексируемый тип, либо ссылочный тип, указывающий на такой же индексируемый тип. В обоих случаях ограничение индекса должно обеспечивать дискретный диапазон для каждого индекса этого индексируемого типа, а тип каждого дискретного диапазона должен совпадать с типом соответствующего индекса.

Ограничение индекса *совместимо* с типом, обозначаемым меткой типа, если и только если ограничение, определяемое каждым дискретным диапазоном, совместимо с соответствующим подтипом индекса. Если какой-нибудь из дискретных диапазонов определяет пустой диапазон, то любой массив, ограниченный таким способом, является *пустым массивом*, не имеющим компонентов. Значение массива *удовлетворяет* ограничению индекса, если в каждой позиции индекса это значение и это ограничение имеют один и тот же диапазон индекса. (Необходимо учесть, что присваивание и некоторые другие операции над массивами вызывают неявное преобразование типа).

Диапазон индекса для каждого индекса индексируемого объекта устанавливается следующим образом:

1) Для переменной или сигнала, описанных объявлением объекта, указание подтипа в соответствующем объявлении объекта должно определять ограниченный индексируемый подтип (а следовательно, диапазон индекса для каждого индекса этого объекта). Те же требования верны для указания подтипа в объявлении элемента в описании индексируемого типа, если тип элемента структуры является индексируемым типом, и для указания подтипа элемента в описании индексируемого типа, если тип элемента является индексируемым типом.

2) Для константы, описанной объявлением объекта, диапазоны индекса определяются инициализирующим значением, если подтип константы неограничен; в противном случае они определяются этим подтипом (в этом случае инициализирующее значение представляет собой результат неявного преобразования типа).

3) Для атрибута, значение которого задано спецификацией атрибута, диапазоны индекса определяются выражением, заданным в спецификации, если подтип атрибута является неограниченным; в противном случае они определяются этим подтипом (в этом случае значение атрибута является результатом неявного преобразования подтипа).

4) Для любого индексируемого объекта, на который имеется ссылочное значение, диапазоны индекса определяются оператором, создающим этот объект (см. 7.3.6).

5) Для объекта интерфейса, объявленного с указанием подтипа, определяющим ограниченный индексируемый подтип, диапазоны индекса определяются этим подтипом.

6) Для формального параметра подпрограммы, имеющего неограниченный индексируемый тип, диапазоны индекса получаются из соответствующего элемента сопоставления в применяемом вызове подпрограммы.

7) Для формального параметра настройки объекта проекта, имеющего неограниченный индексируемый тип, диапазоны индекса получаются из соответствующего элемента сопоставления в описании соответствия параметров настройки применяемого (явно или неявно) связывающего указания.

8) Для формального порта объекта проекта, имеющего неограниченный индексируемый тип, диапазоны индекса получаются из соответствующего элемента сопоставления в описании сопоставления портов применяемого (явно или неявно) связывающего указания.

9) Для локальных параметров настройки компонента, имеющего неограниченный индексируемый тип, диапазоны индекса получаются из соответствующего элемента сопоставления в описании соответствия параметров настройки применяемого оператора конкретизации компонента.

10) Для локального порта компонента, имеющего неограниченный индексируемый тип, диапазоны индекса получаются из соответствующего элемента сопоставления в описании соответствия портов применяемого оператора конкретизации компонента.

Если диапазоны индекса для объекта интерфейса получаются из соответствующего элемента сопоставления, то они устанавливаются либо фактической частью, либо формальной частью этого элемента сопоставления, в зависимости от вида этого объекта:

1) Для объекта интерфейса вида **in**, **inout** или **linkage**, если фактическая часть содержит функцию преобразования типа, то тип результата этой функции должен быть ограниченным индексируемым типом, а диапазоны индекса получаются из этого ограниченного подтипа; в противном случае диапазоны индекса получаются из объекта или значения, представленного фактическим обозначением.

2) Для объекта интерфейса вида **out**, **buffer**, **inout** или **linkage**, если формальная часть содержит функцию преобразования типа, то подтип параметра этой функции должен быть ограниченным индексируемым типом, а диапазоны индекса получаются из этого ограниченного подтипа; в противном случае диапазоны индекса получаются из объекта, представленного фактическим обозначением.

Для объекта интерфейса вида **inout** или **linkage** диапазоны индекса, устанавливаемые по первому правилу, должны быть идентичны диапазонам индекса, устанавливаемым по второму правилу.

3.2.1.2 *Предопределенные индексируемые типы*

Предопределенными индексируемыми типами являются STRING и BIT_VECTOR, описанные в пакете STANDARD в разделе 14.

Значениями предопределенного типа STRING являются одномерные массивы предопределенного типа CHARACTER, индексируемые значениями предопределенного типа POSITIVE:

```
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
type STRING is array (POSITIVE range < >) of CHARACTER;
```

Значениями предопределенного типа BIT_VECTOR являются массивы предопределенного типа BIT, индексируемые значениями предопределенного типа NATURAL:

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
type BIT_VECTOR is array (NATURAL range < >) of BIT;
```

Примеры

```
1 varitable MESSAGE:STRING (1 to 17) : = "THIS IS A MESSAGE";
2 signal LOW_BYTE : BIT_VECTOR (1 to 7);
```

3.2.2 *Структурные типы*

Структурный тип — это составной тип, объекты которого состоят из именованных элементов. Значением структурного объекта является составное значение, состоящее из значений его элементов.

```
record_type_definition :: =
    record
        element_declaration
        { element_declaration }
    end record

element_declaration :: =
    identifier_list : element_subtype_definition;

identifier_list :: = identifier {,identifier}

element_subtype_definition :: = subtype_indication
```

Каждое объявление элемента (element definition) описывает элемент структурного типа. Идентификаторы всех элементов структурного типа должны быть уникальными. Использование имени, обозначающего элемент структуры, недопустимо внутри описания структурного типа, которое содержит объявление этого элемента.

Объявление элемента, содержащее несколько идентификаторов, эквивалентно последовательности отдельных объявлений элементов. Каждое отдельное объявление элемента описывает элемент структуры, подтип которого задается описанием подтипа элемента (element definition).

Описание структурного типа (record type definition) создает структурный тип; он состоит из объявлений элементов, стоящих в том же порядке, что и в описании этого типа.

Пример

```
type DATE is
    record
        DAY : INTEGER range 1 to 31;
        MONTH : MONTH_NAME;
        YEAR : INTEGER range 0 to 4000;
    end record;
```

3.3 Ссылочные типы

Объект, описанный объявлением объекта, создается предвыполнением этого объявления и обозначается простым именем или какой-либо другой формой имени. В отличие от этого объекты, создаваемые вычислением *генераторов* (см. 7.3.6), не имеют простых имен. Доступ к такому объекту достигается ссылочным значением, формируемым генератором; в этом случае считается, что ссылочное значение указывает на этот объект.

```
access_type_definition :: = access subtype_indication
```

Для каждого ссылочного типа существует литерал null, имеющий пустое ссылочное значение, не указывающее ни на какой объект вообще. Пустое значение ссылочного типа является неявным начальным значением этого типа. Другие значения типа получаются вычислением специальной операции этого типа, называемой генератором. Каждое такое ссылочное значение указывает на объект, подтип которого определяется указанием подтипа (subtype indication) в описании этого ссылочного типа; этот подтип называется *указываемым подтипом;* базовый тип этого подтипа называется *указываемым типом.* Указываемый подтип не может быть файловым типом.

Объект, описанный с ссылочным типом, должен быть объектом класса переменная. Объект, указываемый ссылочным значением, всегда является объектом класса переменная.

Единственной формой ограничения, допускаемой после имени ссылочного типа, стоящего в указании подтипа, является ограничение индекса. Ссылочное значение принадлежит соответствующему подтипу ссылочного типа, если это значение является либо пустым значением, либо значение указываемого объекта удовлетворяет ограничению.

Примеры

```
1 type ADDRESS is access MEMORY;
2 type BUFFER_PTR is access BUFFER;
```

Примечание — Ссылочное значение, формируемое генератором, может быть присвоено различным переменным соответствующего ссылочного типа. Следовательно может существовать более одной переменной ссылочного типа, указывающей на один и тот же объект, созданный генератором. Ссылочное значение может указывать только на объект, созданный генератором; в частности, оно не может указывать на объект, описанный объявлением объекта.

Если ссылочное значение указывает на объект, имеющий индексируемый тип, то границы этого объекта задаются явно или неявно в соответствующем генераторе.

3.3.1 *Неполные описания типов*

На типы, указываемые ссылочными типами, не существует конкретных ограничений. В частности, тип элемента указываемого типа может быть, в свою очередь, другим ссылочным типом или даже тем же самым ссылочным типом. Это позволяет описывать взаимозависимые и рекурсивные ссылочные типы. Описание таких типов требует предварительного неполного описания типа для одного или более типов.

```
incomplete_type_declaration :: = type identifier;
```

Для каждого неполного описания типа (incomplete type definition) должно существовать соответствующее полное описание типа с тем же идентификатором. Это описание должно стоять после соответствующего неполного описания, но в том же разделе объявлений (при этом необязательно, чтобы неполное и полное описания типа следовали текстуально друг за другом).

До завершения соответствующего полного описания типа имя, обозначающее тип, может быть использовано только в качестве метки типа в указании подтипа описания ссылочного типа; никакие ограничения в этом указании не допускаются.

Пример рекурсивного типа

```
type CELL;                    - - описание неполного типа
type LINK is access CELL;
type CELL is
   record
      VALUE : INTEGER;
      SUCC : LINK;
      PRED : LINK;
   end record;
variable HEAD : LINK : = new CELL' (0,nul,nul);
variable NEXT : LINK : = HEAD.SUCC;
```

Примеры взаимосвязи ссылочных типов

```
1 type PART; - - неполные описания типов
  type WIRE;

2 type PART_PTR is access PART;
  type WIRE_PTR is access WIRE;

3 type PART_LIST is array (POSITIVE range < >) of PART_PTR;
  type WIRE_LIST is array (POSITIVE range < >) of WIRE_PTR;
```

```
4 type PART_LIST_PTR is access PART_LIST;
  type WIRE_LIST_PTR is access WIRE_LIST;

5 type PART is
      record
         PART_NAME : STRING;
         CONNECTIONS : WIRE_LIST_PTR;
      end record;

6 type WIRE is
      record
         WIRE_NAME : STRING;
         CONNECTS : PART_LIST_PTR;
      end record;
```

3.3.2 *Размещение и уничтожение объектов*

Объект, указываемый ссылочным значением, размещается генератором для этого типа. Генератор является первичным в выражении. Генераторы описаны в 7.3.6. Для каждого ссылочного типа операция уничтожения неявно объявляется непосредственно после полного описания типа для этого типа. Эта операция позволяет явно освободить память, занимаемую указываемым объектом.

Если имеется следующее описание ссылочного типа:

```
type AT is access T;
```

то этим подразумевается наличие операции, неявно объявленной после этого описания:

```
procedure DEALLOCATE (P: inout AT);
```

Процедура DEALLOCATE принимает в качестве единственного параметра переменную заданного ссылочного типа. Если значением этой переменной является пустое значение для заданного ссылочного типа, то операция не имеет эффекта. Если значением этой переменной является ссылочное значение, указывающее на объект, то участок памяти, занимаемый этим объектом, освобождается и может быть использован в последующих созданиях объекта с использованием генератора. Ссылочный параметр P устанавливается в пустое значение для заданного типа.

3.4 Файловые типы

Описание файлового типа определяет файловый тип. Файловые типы используются для определения объектов, представляющих файлы в среде вычислительной системы. Значение файлового объекта - это последовательность значений, содержащихся в файле вычислительной системы.

```
file_type_definition :: = file of type_mark
```

Метка типа в описании файлового типа определяет подтип значений, содержащихся в файле. Метка типа может обозначать либо ограниченный, либо неограниченный подтип. Базовым типом этого подтипа не может быть файловый или ссылочный тип. Если базовый тип является составным типом, то он не должен содержать подэлементы ссылочного типа. Если базовым типом является индексируемый тип, то он должен быть одномерным индексируемым типом.

Примеры

```
1 file of STRING    - - Описание файлового типа, который
                    - - может содержать неограниченное
                    - - количество строк

2 file of NATURAL - - Описание файлового типа, который
                    - - может содержать только неотри-
                    - - цательные целые значения
```

3.4.1 *Файловые операции*

Для объектов файлового типа обеспечиваются три операции. Если имеется следующее описание типа:

```
type FT is file of TM;
```

в котором метка типа TM обозначает скалярный, именуемый или ограниченный индексируемый тип, то этим подразумевается наличие следующих операций, неявно объявленных после этого описания типа:

```
procedure READ (F: in FT; VALUE: out TM);
procedure WRITE (F: out FT; VALUE: in TM);
function ENDFILE (F: in FT) return BOOLEAN;
```

Процедура READ ищет следующее значение в файле. Процедура WRITE добавляет значение в конец файла. Функция ENDFILE возвращает значение FALSE, если последующая операция READ над входным файлом может отыскать следующее значение в этом файле; в противном случае она возвращает значение TRUE. Функция ENDFILE всегда возвращает значение TRUE для выводного файла.

Для описания файлового типа, в котором метка типа обозначает неограниченный индексируемый тип, объявляются те же операции, за исключением того, что операция READ объявляется несколько иначе:

```
procedure READ (F: in FT; VALUE: out TM; LENGTH: out
NATURAL);
```

Операция READ для такого типа выполняет то же действие, что и для других типов, но в дополнение они возвращают через параметр LENGTH значение, содержащее фактическую длину индексируемого значения, прочитанного этой операцией. Если объект, сопоставленный формальному параметру VALUE, короче этой длины, то читается только та порция индексируемого значения, которая может быть помещена в этот объект (эта порция является возвращаемым значением операции READ), остальная часть этого значения теряется. Если объект, сопоставляемый формальному параметру VALUE, длиннее, чем эта длина, то возвращаются все значения целиком, а оставшиеся элементы этого объекта остаются незадействованными в этой операции.

Ошибка возникает, когда операция READ выполняется над файлом F, а функция ENDFILE(F) в этой точке возвращает значение TRUE.

Примечание — Предопределенный пакет TEXTIO обеспечивает форматированный ввод/вывод кодов ASCII. Он содержит описание типа TEXT (файлового типа, представляющего файлы со строками символов ASCII переменной длины) и типа LINE (ссылочного типа, указывающего на такие строки). Операции READ и WRITE, которые добавляют или извлекают данные из отдельной строки, также описаны в пакете TEXTIO. В пакете TEXTIO содержатся описания дополнительных операций, обеспечивающих чтение и запись целых строк и определение состояния текущей строки или самого файла. Описание пакета TEXTIO содержится в разделе 14.

4 ОБЪЯВЛЕНИЯ

Язык определяет различные виды понятий, которые описываются явно или неявно с помощью объявлений.

```
declaration :: =
    | type_declaration
    | subtype_declaration
    | object_declaration
    | file_declaration
    | interface_declaration
    | alias_declaration
    | attribute_declaration
    | component_declaration
    | entity_declaration
    | configuration_declaration
    | subprogram_declaration
    | package_declaration
```

Для каждой формы объявления (declaration) правила языка устанавливают конкретную текстуальную область, называемую *областью действия* этого объявления. Каждая форма объявления сопоставляет идентификатор с описываемым понятием. Только внутри области действия объявления существуют места, где можно использовать этот идентификатор для ссылки на сопоставленное с ним описанное понятие; такие места определяются правилами видимости. Считается, что в таких местах этот идентификатор является *именем* понятия; в свою очередь считается, что это имя *обозначает* сопоставленное с ним понятие.

Данный раздел содержит описание объявлений типов и подтипов, различных видов объявлений объектов, объявлений дополнительных имен, объявлений атрибутов и объявлений компонентов. Другие виды объявлений описаны в разделах 1 и 2.

Объявление имеет эффект в результате процесса предвыполнения. Предвыполнение объявлений описано в разделе 12.

4.1 Объявления типов

Объявление типа (type declaration) описывает тип.

```
type_declaration :: =
    full_type_declaration
    | incomplete_type_declaration
    full_type_declaration :: =
    type identifier is type_definition;

type_definition :: =
    scalar_type_definition
    | composite_type_definition
    | access_type_definition
    | file_type_definition
```

Типы, получаемые предвыполнением отдельных описаний типа (type definition), являются различными типами. Предвыполнение описания скалярного или ограниченного индексируемого типа создает одновременно базовый тип и подтип этого базового типа.

Простое имя, стоящее в объявлении типа, обозначает описываемый тип при условии, что это объявление не описывает одновременно базовый тип и подтип этого базового типа. В противном случае простое имя обозначает подтип, а базовый тип является анонимным. Тип считается *анонимным*, если он не имеет простого имени. В пояснительных целях в данном руководстве для обозначения анонимного типа используется псевдоимя, выделенное курсивом. Это имя также используется в тех местах, где синтаксис языка обычно требует идентификатор.

Примечание — Два объявления типа всегда описывают два различных типа, даже если оба они текстуально идентичны. Так например, два следующих объявления целого типа описывают различные типы:

```
1 type A is range 1 to 10;
2 type B is range 1 to 10;
```

Это также верно для объявлений других классов типов.

Различные формы описаний типа представлены в разделе 3.

Примеры объявлений типов также содержатся в данном разделе.

4.2 Объявления подтипов

Объявление подтипа (subtype declaration) описывает подтип.

```
subtype_declaration :: =
    subtype identifier is subtype_indication;

subtype_indication :: =
    [resolution_function_name] type_mark[constraint]

type_mark :: =
    type_name
    | subtype_name

constraint :: =
    range_constraint
    | index_constraint
```

Метка типа (type mark) обозначает либо тип, либо подтип. Если метка типа является именем типа, то такая метка обозначает этот тип, а также соответствующий неограниченный подтип. Базовым типом метки типа является, по определению, базовый тип этого типа или подтипа, обозначаемого этой меткой.

Указание подтипа (subtype indication) определяет подтип базового типа метки типа.

Если указание подтипа содержит имя функции разрешения (resolution function name), то любой сигнал, объявленный с этим подтипом, будет разрешаться, по необходимости, заданной функцией (см. 2.4). Имя функции разрешения не имеет никакого эффекта в объявлениях файлов, дополнительных имен, атрибутов или других подтипов.

Если указание подтипа не содержит ограничения (constraint), то такой подтип эквивалентен подтипу, обозначаемому меткой типа. Условие, налагаемое ограничением, — это условие, получаемое вычислением выражений и диапазонов, составляющих это ограничение. В соответствующих разделах для каждой формы ограничения содержатся правила, устанавливающие совместимость. Эти правила таковы, что если ограничение совместимо с подтипом, то условие, налагаемое этим ограничением, не может противоречить

любому другому условию, уже наложенным этим подтипом на его значения. Возникает ошибка, если любая проверка совместимости не имеет успеха.

Направление указания дискретного подтипа совпадает с направлением ограничения диапазона (range constraint), используемого как ограничение в указании подтипа. Если ограничение не задано, и метка типа обозначает подтип, то направление указания подтипа совпадает с направлением обозначаемого подтипа. Если ограничение не задано и метка типа обозначает тип, то направление указания подтипа совпадает с направлением диапазона, используемого в описании обозначаемого типа. Направление дискретного подтипа совпадает с направлением соответствующего указания подтипа.

Указание подтипа, обозначающее ссылочный или файловый тип, не может содержать функцию разрешения. Единственным допустимым ограничением в указании подтипа, обозначающем ссылочный тип, является ограничение индекса (index constraint) (только в том случае, если указываемый тип является индексируемым типом).

Примечание — Объявление подтипа не описывает новый тип.

4.3 Объекты

Объект — это понятие, которое содержит (имеет) значение конкретного типа. Объектом может быть следующее:

1) Объект, описанный объявлением объекта.
2) Файл, описанный объявлением файла.
3) Параметр оператора цикла или генерации.
4) Формальный параметр подпрограммы.
5) Формальный порт объекта проекта.
6) Формальный параметр настройки.
7) Локальный порт.
8) Локальный параметр настройки.
9) Элемент или сечение другого объекта.
10) Значение объекта, указываемое значением ссылочного типа.

Имеются три класса объектов: константы, сигналы и переменные. Класс явно объявляемого объекта задается зарезервированным словом, которое должно стоять в начале объявления этого объекта. Для конкретного объекта составного типа каждый подэлемент этого объекта сам является объектом того же класса, что и составной объект. Значением составного объекта является агрегат, составленный из значений его подэлементов.

Объекты, описанные объявлениями объектов и объявлениями файлов, доступны для использования в пределах блоков, процессов, подпрограмм или пакетов. Параметры цикла или генерации объявляются неявно соответствующими операторами и доступны для использования только внутри этих операторов. Другие объекты, описанные объявлениями интерфейса, образуют каналы для обмена значениями между независимыми частями описания.

4.3.1 *Объявления объектов*

Объявление объекта описывает объект заданного типа.

```
object_declaration :: =
    constant_declaration
  | signal_declaration
  | variable_declaration
```

Объявление объекта называется *отдельным объявлением объекта*, если стоящий в нем список идентификаторов содержит единственный идентификатор; объявление объекта называется *множественным объявлением объекта*, если стоящий в нем список идентификаторов содержит два или более идентификаторов. Множественное объявление объекта эквивалентно последовательности соответствующего числа отдельных объявлений объектов. Для каждого идентификатора в списке эта эквивалентная последовательность содержит отдельное объявление объекта, состоящее из самого идентификатора, за которым следует двоеточие и все то, что стоит справа от двоеточия в множественном объявлении объекта. Порядок в эквивалентной последовательности соответствует порядку в списке идентификаторов.

Аналогичная эквивалентная последовательность имеет место для объявлений объектов интерфейса (см. 4.3.3).

4.3.1.1 *Объявления констант*

Объявление константы описывает *константу* заданного типа.

```
constant_declaration :: =
    constant identifier_list :
        subtype_indication [: =expression ];
```

Если в объявлении константы присутствует символ присваивания ": =", за которым следует выражение, то это выражение задает значение этой константы. Значение константы не может быть изменено после того, как произошло предвыполнение ее объявления.

Если в объявлении константы отсутствует символ присваивания и следующее за ним выражение, то такое объявление описывает *неполную константу*. Такое объявление константы может использоваться только в объявлении пакета. Соответствующее полное объявление константы должно стоять в теле этого пакета.

Формальные параметры подпрограмм вида in могут быть константами, а локальные и формальные параметры настройки всегда являются константами; объявления этих объектов описаны в 4.3.3. Параметр цикла является константой внутри соответствующего оператора цикла. Аналогично, параметр генерации является константой внутри соответствующего оператора генерации. Подэлемент или сечение константы является константой.

Считается ошибкой, если объявление константы описывает константу файлового или ссылочного типа.

Примеры

```
1 constant TOLERANCE : DISTANCE : = 1.5nm;
2 constant PI : REAL : = 3.141592;
3 constant CYCLE_TIME : TIME : = 100ns;
4 constant Propagation_Delay;
```

4.3.1.2 *Объявление сигналов*

Объявление сигнала описывает сигнал заданного типа.

```
signal_declaration :: =
    signal identifier_list : subtype_indication
                    [signal_kind ] [ : = expression ];
signal_kind :: = register | bus
```

Если в объявлении сигнала или в объявлении подтипа, используемого для описания этого сигнала, используется имя функции разрешения, то эта функция сопоставляется с описанным сигналом. Такой сигнал называется *разрешенным сигналом*.

Если в объявлении сигнала используется вид сигнала (signal kind), то сигналы, описанные таким способом, являются *защищенными* сигналами указанного вида. Каждый подэлемент защищенного сигнала составного типа также является защищенным сигналом. Присваивание значений защищенному сигналу происходит под управлением *выражений защиты* (или *защит*), вырабатывающих логическое значение. Когда конкретная защита имеет значение FALSE, то драйверам соответствующих защищенных сигналов неявно присваивается пустая транзакция (см. 8.3.1) с целью отключения этих драйверов. Для задания времени, по прошествии которого происходит отключение этих драйверов, используется спецификация отключения (см. 5.3).

Если объявление сигнала содержит символ присваивания, за которым следует выражение, то тип последнего должен совпадать с типом сигнала. Такое выражение считается *неявным выражением*. Неявное выражение определяет неявное *значение*, сопоставленное с сигналом или с каждым скалярным подэлементом составного сигнала. Для сигнала, описанного со скалярным подтипом, значение неявного выражения является неявным значением этого сигнала. Для сигнала, описанного с составным подтипом, каждый скалярный подэлемент значения неявного выражения является неявным значением соответствующего подэлемента этого сигнала.

При отсутствии неявного выражения, неявное значение для скалярного сигнала некоторого скалярного подтипа T или для каждого скалярного подэлемента составного сигнала, которые сами являются сигналами некоторого скалярного подтипа T, определяется как T'LEFT.

Считается ошибкой, если объявление сигнала описывает сигнал файлового или ссылочного типа. Также считается ошибкой, если защищенный сигнал скалярного типа не является ни разрешенным сигналом, ни подэлементом разрешенного сигнала.

Сигнал может иметь один или более *источников*. Для сигнала скалярного типа источником может быть либо драйвер (см. 9.2.1), либо порт вида **out**, **inout**, **buffer** или **linkage** экземпляра компонента, с которым этот сигнал сопоставлен. Для сигнала составного типа каждый составной источник является совокупностью скалярных источников, по одному на каждый скалярный элемент этого сигнала. Считается ошибкой, если после предвыполнения описания сигнал имеет множество источников, не являясь при этом разрешенным сигналом.

Если подэлемент разрешенного сигнала составного типа сопоставлен как фактический параметр в описании определения портов (либо в операторе конкретизации компонента, либо в связывающем указании), а соответствующий формальный порт имеет вид **out**, **inout**, **buffer** или **linkage**, то каждый скалярный подэлемент этого сигнала должен быть сопоставлен точно один раз с подобным формальным

портом в том же описании распределения портов таким образом, чтобы совокупность соответствующих формальных портов, взятых вместе, составляла один источник этого сигнала. Если подэлемент разрешенного сигнала составного типа сопоставлен как фактический параметр в описании распределения портов, то это эквивалентно тому, что каждый подэлемент этого сигнала сопоставлен в этом же описании.

Если подэлемент разрешенного сигнала составного типа имеет драйвер в конкретном процессе, то каждый скалярный подэлемент этого сигнала должен иметь драйвер в этом же процессе, а совокупность всех таких драйверов, взятых вместе, составляет один источник этого сигнала.

Неявное значение, сопоставленное со скалярным сигналом, определяет компонент значения транзакции, которая является начальным содержимым каждого драйвера (если они есть) этого сигнала. Компонент времени этой транзакции не определен, но подразумевается, что эта транзакция уже имела место вследствие начала моделирования.

Примеры

```
1 signal S: STANDARD.BIT_VECTOR (1 to 10);
2 signal CLK1, CLK2 : TIME;
3 signal OUTPUT : WIRED_OR MultivaluedLogic;
```

Примечание — Порты любого вида также являются сигналами. Термин *сигнал* используется в данном руководстве для ссылки на объекты, описанные либо объявлениями сигналов, либо объявлениями портов; термин *порт* используется для ссылки на объекты, описанные только объявлениями портов.

Сигналы получают начальное значение инициализацией их драйверов, начальные значения драйверов затем распространяются по соответствующей цепи для установки начальных значений сигналов, составляющих эту цепь (см. 12.6.3).

Значение сигнала может быть косвенно изменено оператором назначения сигнала (см. 8.3); такие назначения оказывают влияние на будущие значения этого сигнала.

4.3.1.3 *Объявления переменных*

Объявление переменной описывает *переменную* заданного типа.

```
variable_declaration :: =
    variable identifier_list : subtype_indication
                                [: = expression];
```

Если объявление переменной содержит символ присваивания, за которым следует выражение, то это выражение задает начальное значение для описываемой переменной; тип выражения должен совпадать с типом переменной. Указанное выражение считается *выражением начального значения.*

Если выражение начального значения стоит в объявлении переменной, то начальное значение устанавливается этим выражением всякий раз, когда происходит предвыполнение этого объявления. При отсутствии выражения начального значения используется неявное начальное значение. Неявное значение переменной скалярного подтипа T определяется как значение, вырабатываемое атрибутом T'LEFT. Неявное начальное значение переменной составного типа определяется как агрегат, составленный из неявных начальных значений всех ее скалярных подэлементов, каждый из которых в свою очередь является переменной скалярного типа. Неявное начальное значение переменной ссылочного типа определяется как значение **null** для этого типа.

Считается ошибкой, если объявление переменной описывает переменную файлового типа.

Примечание — Значение переменной может быть изменено оператором присваивания (см. 8.4); эти присваивания имеют мгновенный эффект. Параметры процедуры вида **in** могут быть файловыми переменными; параметры процедуры вида **out** или **inout** могут быть переменными любого вида.

Переменная, объявленная внутри конкретной процедуры, сохраняет свое существование до завершения этой процедуры и возврата в вызывающую программу. Для процедур, содержащих операторы ожидания, переменная может сохранять свое существование от одного момента моделирования до другого, и таким образом значение этой переменной временно поддерживается. Для процессов, которые не имеют завершения, все переменные сохраняют свое существование с начала моделирования до его окончания.

Примеры

```
1 variable INDEX : INTEGER range 0 to 99 : = 0;
      - - начальное значение устанавливается выражением
      - - начального значения

2 variable COUNT : POSITIVE;
      - - начальное значение есть POSITIVE'LEFT или 1

3 variable MEMORY:BIT_MATRIX (0 to 7, 0 to 1023);
      - - начальное значение есть агрегат из начальных
      - - значений каждого подэлемента.
```

4.3.2 *Объявления файлов*

Файловый объект создается объявлением файла. Такой объект относится к классу переменных; операции с файловыми объектами ограничены по сравнению с операциями, доступными для других объектов этого класса. В частности, для файловых объектов не разрешена операция присваивания.

```
file_declaration :: =
    file identifier : subtype_indication is
                      [mode] file_logical_name;

file_logical_name :: = string_expression
```

Указание подтипа (subtype indication) в объявлении файла должно описывать подтип файла. Единственными допустимыми видами файла являются виды **in** и **out**.

Логическое имя файла (file logical name) должно быть представлено выражением типа STRING. Значение этого выражения интерпретируется как логическое имя для файла в главной вычислительной системе. Реализация должна обеспечивать некоторый механизм для сопоставления логического имени файла с конкретным физическим файлом. Такой механизм языком не определяется.

Логическое имя файла идентифицирует внешний файл в главной файловой системе, сопоставляемый с файловым объектом. Это сопоставление обеспечивает механизм либо для импортирования данных, содержащихся во внешнем файле в течение моделирования, либо экспортирования данных, полученных в течение моделирования во внешний файл.

Если в объявлении файла задан вид **in**, то содержимое такого внешнего файла может читаться в течение моделирования. В этом случае файловый объект может читаться, но не модифицироваться, более чем одним процессом. Если в объявлении файла задан вид **out**, то содержимое такого внешнего файла может записываться в течение моделирования. В этом случае файловый объект может модифицироваться, но не читаться, одним или более процессами. Если вид не задан, то по умолчанию принимается **in**.

Если с одним и тем же логическим именем файла сопоставлено множество файловых объектов и каждый файловый объект описывается объявлением файла с видом **out**, то значения, выводимые в каждый файловый объект, выводятся во внешний файл, идентифицируемый этим логическим именем. Язык не определяет ни порядок, в котором эти значения выводятся во внешний файл, ни количество создаваемых при этом внешних файлов.

Если формальный параметр подпрограммы имеет файловый тип, то он должен сопоставляться с фактическим параметром, являющимся файловым объектом. Файловый объект конкретного вида может быть передан формальной файловой переменной только соответствующего вида.

Примечание — Все внешние файловые объекты, сопоставляемые с одним и тем же внешним файлом, должны иметь один и тот же базовый тип.

4.3.3 *Объявления интерфейсов*

Объявление интерфейса (interface declaration) описывает объект интерфейса заданного типа. Объекты интерфейса включают в себя:

— константы, используемые как параметры настройки объекта проекта, компонента или блока или как константные параметры подпрограмм;

— сигналы, используемые в качестве портов объекта проекта, компонента или блока или в качестве параметров-сигналов подпрограмм;

— переменные, используемые в качестве параметров — переменных подпрограмм.

```
interface_declaration :: =
    interface_constant_declaration
    | interface_signal_declaration
    | interface_variable_declaration

interface_constant_declaration :: =
    [constant] identifier_list : [in] subtype_indication
                                 [:=static_expression]

interface_signal_declaration :: =
    [signal] identifier_list : [mode] subtype_indication
                               [bus] [:= static
                                        _expression]
```

```
interface_variable_declaration :: =
    [variable] identifier_list : [mode] subtype
                                 _indication [:= static
                                 _expression]

mode :: = in | out | inout | buffer | linkage
```

Если в объявлении интерфейса вид (mode) явно не задан, то подразумевается in.

В объявлении константы интерфейса (interface constant declaration) или в объявлении сигнала интерфейса (interface signal declaration) указание подтипа (subtype indication) должно описывать подтип, который не является ни файловым, ни ссылочным типом.

Если объявление сигнала интерфейса содержит зарезервированное слово bus, то сигнал, описываемый этим объявлением, является защищенным сигналом класса bus.

Если объявление интерфейса содержит символ ":=", за которым следует выражение, то это выражение считается *неявным выражением* для объекта интерфейса. Тип этого выражения должен совпадать с типом соответствующего объекта интерфейса. Считается ошибкой, если неявное выражение стоит в объявлении интерфейса, вид которого linkage, или соответствующая метка типа (type mark) обозначает файловый тип.

В объявлении сигнала интерфейса неявное выражение определяет неявное значение (значения), сопоставляемое с сигналом интерфейса или его подэлементами. При отсутствии неявного выражения для сигнала или каждого скалярного подэлемента подразумевается неявное значение, устанавливаемое в соответствии с правилами объявления сигналов (см. 4.3.12). Такое значение используется для установки начального содержимого драйверов сигнала интерфейса, если они есть, как указано для объявлений сигналов.

Объект интерфейса обеспечивает канал для связи между окружающей средой и отдельной частью описания. Значение объекта интерфейса может быть установлено значением сопоставленного объекта или выражения, находящихся в этой среде; аналогично, значение объекта, находящегося в этой среде, может определяться значением сопоставленного с ним объекта интерфейса. Способы таких сопоставлений описаны в 4.3.3.2.

Считается, что значение объекта *читается*, если удовлетворяется одно из следующих условий:

1) Объект вычисляется, а также (косвенно) объект сопоставлен с объектом интерфейса вида in, inout или linkage.

2) Объект является сигналом и имя, обозначающее этот объект, используется в списке чувствительности оператора ожидания или оператора процесса.

3) Объект является сигналом и значение любого из его предопределенных атрибутов, таких как STABLE, QUIET, DELAYED, TRANSACTION, EVENT, ACTIVE, LAST_EVENT, LAST_ACTIVE или LAST_VALUE, читается.

4) Один из подэлементов объекта читается.

5) Объект является файлом и с этим файлом выполняется операция READ.

Считается, что значение объекта *изменяется*, если удовлетворяется одно из следующих условий:

1) Объект стоит в левой части оператора присваивания, а также (косвенно) если объект сопоставлен с объектом интерфейса вида out, buffer, inout или linkage.

2) Изменяется один из подэлементов объекта.

3) Объект является файлом и с этим файлом выполняется операция WRITE.

Изменяться могут объекты только класса сигнал или класса переменная. Переменная типа файл может быть изменена только выполнением операции WRITE; считается ошибкой, если файловая переменная стоит в левой части оператора присваивания.

Объект интерфейса имеет следующие виды:

1) **in.** Значение такого объекта интерфейса может только читаться. Любые атрибуты этого объекта интерфейса могут читаться, за исключением того, что значения атрибутов STABLE, QUIET, DELAYED и TRANSACTION параметра-сигнала не могут быть прочитаны внутри подпрограммы. Для файловых объектов допускается операция ENDFILE.

2) **out.** Значение такого объекта интерфейса может быть изменено. Допускается чтение атрибутов элемента интерфейса, отличных от предопределенных атрибутов STABLE, QUIET, DELAYED, TRANSACTION, EVENT, ACTIVE, LAST_EVENT, LAST_ACTIVE и LAST_VALUE. Для файлового объекта разрешена операция ENDFILE. Никакое другое чтение не допускается.

3) **inout.** Значение такого объекта интерфейса может быть прочитано либо изменено. Также разрешено чтение атрибутов этого объекта. Для файлового объекта разрешена операция ENDFILE.

4) **buffer.** Значение такого объекта интерфейса может быть прочитано либо изменено. Также разрешено чтение атрибутов этого объекта.

5) **linkage.** Значение такого объекта интерфейса может быть прочитано или изменено только подстановкой этого объекта в качестве фактического параметра для другого объекта интерфейса вида **linkage.** Никакие другие виды чтения и изменения не разрешены.

Примечание — Хотя сигналы видов inout и buffer имеют одинаковые характеристики в отношении того, могут ли они читаться или изменяться любым количеством источников, сигнал вида inout может изменяться любым количеством источников, в то время как сигнал вида buffer должен изменяться самое большее одним источником (см. 1.1-1.2).

Параметр подпрограммы файлового типа должен объявляться как параметр-переменная.

4.3.3.1 *Списки интерфейсов*

Список интерфейса (interface list) содержит объявления объектов интерфейса, требуемых подпрограммой, компонентом, объектом проекта или оператором блока.

```
interface_list :: =
    interface_element {; interface_element}

interface_element :: = interface_declaration
```

Список интерфейса параметров настройки состоит исключительно из объявлений констант интерфейса. Список интерфейса портов состоит исключительно из объявлений сигналов интерфейса. Список интерфейса параметров может содержать объявления констант интерфейса, объявления сигналов интерфейса, объявления переменных интерфейса (interface variable declaration) либо комбинацию этих объявлений.

4.3.3.2 *Списки сопоставлений*

Список сопоставлений (association_list) устанавливает соответствие между формальными или локальными параметрами настройки, портами или именами параметров с одной стороны и локальными или фактическими именами или выражениями с другой стороны.

```
association_list :: =
    association_element { , association_element }

association_element :: =
    [ formal_part = >] actual_part

formal_part :: =
    formal_designator
    | function_name ( formal_designator )

formal_designator :: =
    generic_name
    | port_name
    | parameter_name

actual_part :: =
    actual_designator
    | function_name ( actual_designator )

actual_designator :: =
    expression
    | signal_name
    variable_name
    open
```

Каждый элемент сопоставления (association element) в списке сопоставлений сопоставляет один фактический указатель (actual designator) с соответствующим элементом интерфейса в списке интерфейса (interface_list) объявления подпрограммы, объявления компонента, объявления объекта проекта или оператора блока. Соответствующий элемент интерфейса определяется либо по позиции, либо по имени.

Элемент сопоставления называется *именованным*, если формальный указатель (formal designator) используется явно; в противном случае имеет место *позиционное* сопоставление. Для позиционного сопоставления фактический указатель в конкретной позиции в списке сопоставлений относится к элементу интерфейса, стоящему в этой же позиции в списке интерфейса.

Именованные сопоставления могут задаваться в любом порядке, но если используются оба вида сопоставления в одном и том же списке сопоставлений, то все позиционные сопоставления должны быть заданы в первую очередь в их обычных позициях. Следовательно, как только появляется именованное сопоставление, то все оставшиеся сопоставления в этом списке должны быть именованными.

Формальная часть (formal part) именованного элемента сопоставления может быть в форме вызова функции, в котором единственным аргументом этой функции является сам формальный указатель. В этом случае имя функции должно обозначать функцию, единственный параметр которой имеет тот же тип, что и формальный указатель, и результат которой имеет тот же тип, что и соответствующий фактический указатель. Такая функция обеспечивает преобразование типа в случае, когда данные передаются от формального указателя фактическому указателю.

Аналогично, фактическая часть (actual part) именованного (или позиционного) элемента сопоставления может быть в форме вызова функции, в котором единственным аргументом этой функции является сам фактический указатель. В этом случае имя функции должно обозначать функцию, единственный параметр которой имеет тот же тип, что и фактический указатель, и результат которой имеет тот же тип, что и соответствующий формальный указатель. Такая функция обеспечивает преобразование типа в случае, когда данные передаются от фактического указателя формальному указателю.

Если вид формального указателя есть **in**, **inout** или **linkage** и фактический указатель не задан как **open**, то тип фактического указателя (после применения функции преобразования типа, если последняя задана в фактической части) должен совпадать с типом соответствующего формального указателя. Аналогично, если вид формального указателя есть **out**, **inout**, **buffer** или **linkage**, и фактический указатель не задан как **open**, то тип формального указателя (после применения функции преобразования типа, если последняя задана в формальной части) должен совпадать с типом соответствующего фактического указателя.

Для сопоставления сигналов с соответствующими формальными портами сопоставление формального указателя конкретного составного типа с фактическим указателем того же типа эквивалентно сопоставлению каждого скалярного подэлемента формального указателя с соответствующим подэлементом фактического указателя, при условии, что функция преобразования типа не задана ни в формальной части, ни в фактической части элемента сопоставления. Если функция преобразования типа задана, то считается, что формальный указатель целиком сопоставляется с фактическим указателем.

Аналогично, для сопоставления фактических указателей с соответствующими формальными параметрами подпрограммы сопоставление формального параметра конкретного составного типа с фактическим указателем того же типа эквивалентно сопоставлению каждого скалярного подэлемента формального параметра с соответствующим подэлементом фактического указателя. В каждом случае могут потребоваться различные механизмы передачи параметров, но в обоих случаях сопоставления будут иметь эквивалентный результат. Такой результат имеет место при условии, что ни один актуальный указатель не имеет доступ более чем по одному пути (см. 2.1.1.1).

Формальный указатель может быть либо явно объявленным объектом интерфейса, либо подэлементом этого объекта. В последнем случае для сопоставления формального и фактического указателей необходимо использовать именованное сопоставление. Более того, каждый подэлемент явно объявленного объекта интерфейса должен быть сопоставлен точно один раз с фактическим указателем в одном и том же списке сопоставлений, и все эти сопоставления должны быть представлены непрерывной последовательностью в этом списке. Каждый такой элемент сопоставления должен идентифицировать формальный указатель при помощи локально статического имени.

Если элемент интерфейса в списке интерфейса содержит неявное выражение для формального параметра настройки или для формального параметра вида **in**, то любой соответствующий список сопоставлений может не содержать элемент сопоставления для этого элемента интерфейса. Если этот элемент сопоставления не включается в список сопоставления, то в неявном элементе сопоставления для этого элемента интерфейса в качестве фактического выражения используется значение неявного выражения.

Примечание — Из вышеуказанных правил следует, что если элемент сопоставления опущен в списке сопоставлений с целью использования неявного выражения для соответствующего элемента интерфейса, то все последующие элементы сопоставления в этом списке должны быть именованными сопоставлениями.

Хотя неявное выражение может задаваться в элементе интерфейса, который описывает (локальный или формальный) порт, такое выражение не интерпретируется как значение неявного элемента сопоставления для этого порта, так как порты должны быть сопоставлены с сигналами в противоположность значениям. Вместо этого значение выражения используется для определения эффективного значения этого порта с течение моделирования, если порт оставлен несоединенным (см. 12.6.1).

4.3.4 *Объявление дополнительного имени*

Объявление дополнительного имени описывает альтернативное имя для существующего объекта.

```
alias_declaration :: =
    alias identifier : subtype_indication is name;
```

Идентификатор, задаваемый в объявлении дополнительного имени, обозначает объект, представляемый именем в этом объявлении. Дополнительное имя сигнала обозначает сигнал; дополнительное имя переменной обозначает переменную; дополнительное имя константы обозначает константу.

Имя (name) должно быть статическим именем (см. 6.1), обозначающим объект. Базовый тип имени, заданного в объявлении дополнительного имени, должен совпадать с базовым типом метки типа в указании подтипа; этот тип не должен быть многомерным индексируемым типом. Когда на объект, обозначаемый именем, имеется ссылка в виде дополнительного имени, описанного объявлением дополнительного имени, то объект рассматривается как объект, имеющий подтип, заданный указанием подтипа. То же самое имеет место для ссылок на атрибуты, в которых префикс обозначает дополнительное имя. Если указанный подтип является одномерным индексируемым подтипом, то этот подтип должен включать соответствующий элемент (см. 7.2.2) для каждого элемента объекта, обозначаемого этим именем.

Ссылка на элемент дополнительного имени неявно представляет собой ссылку на соответствующий элемент объекта, обозначаемого этим дополнительным именем. Ссылка на сечение дополнительного имени, состоящее из элементов e1, e2, ... en, является неявно ссылкой на сечение объекта, обозначаемое этим дополнительным именем, состоящее из соответствующих элементов для каждого из элементов от e1 до en.

Примеры

```
1 variable REAL_NUMBER: BIT_VECTOR (0 to 31) ;

2 alias SIGN: BIT is REAL_NUMBER (0) ;
      - - SIGN стал скалярным (BIT) значением

3 alias MANTISSA: BIT_VECTOR (23 downto 0) is
                              REAL_NUMBER (8 to 31) ;
      - - MANTISSA является 24-разрядным значением
      - - с диапазоном индекса от 23 до 0.
      - - необходимо отметить, что диапазоны индекса
      - - значений MANTISSA и REAL_NUMBER (8 to 31)
      - - имеют противоположные направления.
      - - ссылка на значение MANTISSA (23 downto 18)
      - - эквивалентна ссылке на значение REAL_NUMBER
        (8 to 13)

4 alias EXPONENT: BIT_VECTOR (1 to 7) is REAL_NUMBER
                                          (1 to 7) ;
      - - EXPONENT является 7-разрядным значением
      - - с диапазоном от 1 до 7
```

4.4 Объявление атрибутов

Атрибут является значением, функцией, типом, диапазоном, сигналом или константой. Любой из перечисленных видов атрибута может быть сопоставлен с одним или более понятиями в описании. Существуют две категории атрибутов, предопределенные атрибуты и атрибуты, определяемые пользователем. Предопределенные атрибуты являются источниками информации о различных понятиях в описании. Раздел 14 содержит описание всех предопределенных атрибутов. Предопределенные атрибуты вида сигнал не могут быть изменены.

Атрибуты, определяемые пользователем, являются константами произвольного типа. Такие атрибуты описываются объявлением атрибута (attribute declaration).

```
attribute_declaration :: =
    attribute identifier: type_mark;
```

Идентификатор, стоящий в объявлении атрибута, является указателем этого атрибута. Атрибут может быть сопоставлен с объектом проекта, архитектурой, конфигурацией, процедурой, функцией, пакетом, типом, подтипом, константой, сигналом, переменной, компонентом или меткой.

Метка типа (type mark) в объявлении атрибута должна обозначать подтип, не являющийся ни ссылочным ни файловым типом. Подтип может быть неограниченным.

Примеры

```
1 type COORDINATE is record X, Y: INTEGER end record;
2 type POSITIVE is INTEGER range 1 to INTEGER'HIGH;
3 attribute LOCATION: COORDINATE;
4 attribute PIN_NO: POSITIVE;
```

Примечание — Конкретное понятие E будет наследовать атрибут A, если и только если спецификация атрибута для значения атрибута A сопровождает объявление E. При отсутствии такой спецификации имя атрибута в форме E'A является недопустимым.

Атрибут, определяемый пользователем, сопоставляется с понятием, обозначаемым именем, заданным в объявлении, а не с самим именем. Следовательно, на атрибут объекта можно ссылаться при помощи дополнительного имени этого объекта (в отличие от ссылки при помощи объявленного имени объекта), используемого в качестве префикса имени атрибута. Атрибут, на который есть такая ссылка, ничем не отличается от атрибута (а следовательно и значения), на который ссылаются с использованием объявленного имени объекта в качестве префикса.

Определяемый пользователем атрибут порта, сигнала, переменной или константы некоторого составного типа, является атрибутом целиком порта, сигнала, переменной или константы, а не их элементов. Если необходимо сопоставить атрибут с каждым элементом некоторого составного типа, то сам атрибут может быть объявлен с составным типом таким образом, что для каждого элемента объекта будет существовать соответствующий элемент этого атрибута.

4.5 Объявления компонентов

Объявление компонента описывает виртуальный интерфейс объекта проекта, который может быть использован в операторе конкретизации компонента. Для сопоставления экземпляра компонента с объектом проекта, содержащимся в библиотеке, может быть использована конфигурация компонента или спецификация конфигурации.

```
component_declaration :: =
    component  identifier
    [ local_generic_clause ]
    [ local_port-clause ]
    end component;
```

Каждый элемент интерфейса в описании локальных параметров настройки (local generic clause) описывает локальный параметр настройки. Каждый элемент интерфейса в описании локальных портов (local port clause) описывает локальный порт.

5 СПЕЦИФИКАЦИИ

В данном разделе дано описание *спецификаций*, которые могут быть использованы для сопоставления дополнительной информации с VHDL-описанием. Спецификация сопоставляет дополнительную информацию с ранее объявленным понятием. Имеются три вида спецификаций: спецификации атрибутов, спецификации конфигураций и спецификации отключения.

Спецификации всегда связаны с уже существующими понятиями; таким образом конкретная спецификация должна следовать за или (в определенных случаях) быть заключена внутри объявления понятия, с которым она связана. Более того, спецификация должна всегда стоять либо непосредственно в том же разделе объявлений, в котором стоит объявление связанного с ним понятия, либо (в случаях, когда спецификации связаны с модулями проекта) непосредственно в разделе объявлений, сопоставленном с объявлением этого понятия.

5.1. Спецификация атрибута

Спецификация атрибута составляет определяемый пользователем атрибут с одним или более понятиями и определяет значение этого атрибута для этих понятий.

```
attribute_specification :: =
    attribute attribute_designator of
    entity_specification is expression;

entity_specification :: =
    entity_name_list : entity_class

entity_class :: =
      entity        | architecture  | configuration
    | procedure     | function      | package
    | type          | subtype       | constant
    | signal        | variable      | component
    | label

entity_mane_list :: =
    entity_designator {, entity_designator}
    | others
    | all

entity_designator :: = simple_name | operator_symbol
```

Указатель атрибута (attribute designator) должен обозначать атрибут. Список имен понятий (entity name list) идентифицирует те понятия, которые наследуют этот атрибут. Наследование определяется следующими правилами:

1) Если задан список указателей понятий (entity designator), то спецификация атрибута применяется в отношении тех понятий, которые обозначены этими указателями.

2) Если задано зарезервированное слово **others**, то спецификация атрибута применяется к понятиям указанного класса, которые описаны в непосредственно объемлющем разделе объявлений, при условии, что каждое такое понятие не задано явно в списке имен понятий предыдущей спецификации атрибута.

3) Если задано зарезервированное слово **all**, то спецификация атрибута применяется ко всем понятиям указанного класса, которые описаны в непосредственно объемлющем разделе объявлений.

Спецификация атрибута для конкретного класса понятий, стоящая в разделе объявлений, и в которой список имен понятий задан в виде зарезервированного слова **others** или **all**, должна быть последней спецификацией атрибута для указанного класса понятий в этом разделе объявлений. Никакое понятие, относящееся в заданному классу понятий, не может быть описано в указанном разделе объявлений после такой спецификации атрибута.

Выражение задает значение атрибута для каждого понятия, наследующего этот атрибут в результате выполнения спецификации этого атрибута. Тип выражения в спецификации атрибута должен совпадать с (или неявно конвертироваться в) меткой типа в соответствующем объявлении атрибута. Спецификация атрибута для модуля проекта (то есть объявления объекта, архитектуры, конфигурации или пакета) должна стоять непосредственно в разделе объявлений этого модуля. Спецификация атрибута для процедуры, функции, типа, подтипа, объекта (то есть константы, сигнала, переменной), компонента или помеченного понятия должна стоять внутри раздела объявлений, в котором находится объявление этой процедуры, функции, типа, подтипа, объекта, компонента или метки соответственно.

Для конкретного понятия значением определяемого пользователем атрибута этого понятия является значение, заданное в спецификации атрибута для этого понятия.

Считается ошибкой, если конкретный атрибут сопоставляется более одного раза с конкретным понятием. Аналогично считается ошибкой, если два различных атрибута с одинаковыми простыми именами сопоставлены с конкретным понятием.

Примеры

```
1 attribute PIN_NO of CIN : signal is 10;
2 attribute PIN_NO of COUT : signal is 5;
3 attribute LOCATION of ADDER1 : label is (10,15);
4 attribute LOCATION of others : label is (25,77);
5 attribute CAPACITANCE of all : signal is 15pF;
```

Примечание — Указатель понятия (entity designator) в виде символа оператора используется для сопоставления атрибута с совмещенным оператором.

Если спецификация атрибута используется, то она должна следовать за объявлением понятия, с которым этот атрибут сопоставляется, и предшествовать всем ссылкам на этот атрибут этого понятия. Спецификация атрибута разрешена только для определяемых пользователем атрибутов, но не для предопределенных атрибутов.

В списке имен понятий спецификации атрибута вместо имен понятий могут быть использованы дополнительные имена этих понятий, но в этом случае эта спецификация рассматривается как отдельная спецификация конкретного атрибута и все последующие спецификации этого атрибута, использующие объявленное имя этого понятия (или другие дополнительные имена), являются недействительными.

Спецификация атрибута переменной ссылочного типа сопоставляет атрибут с самой переменной, а не с указываемым объектом. Спецификация атрибута для одной из множества совмещенных подпрограмм, все из которых объявлены в одном и том же разделе объявлений, сопоставляет этот атрибут также и с каждой из указанных подпрограмм.

Определяемые пользователем атрибуты несут только локальную информацию и не могут быть использованы для передачи информации между описаниями. Например, сигнал X внутри архитектуры и порт Y компонента, находящегося внутри этой же архитектуры, могут иметь один и тот же атрибут A. Однако значения атрибутов X'A и Y'A никаким образом между собой не связаны. В частности, сопоставление сигнала X с портом Y в операторе конкретизации компонента не импортирует значение Y'A и не экспортирует значение X'A.

5.2 Спецификация конфигурации

Спецификация конфигурации сопоставляет связывающую информацию с метками компонентов, представляющими экземпляры конкретного компонента.

```
configuration_specification :: =
    for component_specification use binding_indication;

component_specification :: =
    instantiation_list : component_name
```

```
instantiation_list :: =
    instantiation_label {, instantiation_label}
    | others
    | all
```

Список экземпляров (instantiation list) идентифицирует те понятия, с которыми сопоставляется связывающая информация. Сопоставление выполняется по следующим правилам:

1) Если задан список меток экземпляров (instantiation label), то спецификация конфигурации применяется к соответствующим экземплярам компонента. Эти метки должны быть объявлены внутри непосредственно объемлющего раздела объявлений. Считается ошибкой, если эти экземпляры не являются экземплярами компонента, имя которого указано в спецификации компонентов (component specification).

2) Если задано зарезервированное слово **others**, то спецификация конфигурации применяется к экземплярам указанного компонента, метки которых объявлены в непосредственно объемлющем разделе объявлений при условии, что каждый такой экземпляр компонента уже не используется явно в списке экземпляров предыдущей спецификации конфигурации.

3) Если задано зарезервированное слово **all**, то спецификация конфигурации применяется ко всем экземплярам указанного компонента, метки которых объявлены в непосредственно объемлющем разделе объявлений.

Спецификация конфигурации, стоящая в разделе объявлений, и список экземпляров которой для данного компонента задан в виде зарезервированного слова **others** или **all**, должна быть последней такой спецификацией для этого компонента в этом разделе объявлений.

Предвыполнение спецификации конфигурации влечет сопоставление связывающей информации с метками, идентифицируемыми списком экземпляров. Метка, которая имеет сопоставленную с ней связывающую информацию, считается **связанной**. Считается ошибкой, если предвыполнение спецификации конфигурации влечет сопоставление связывающей информации с меткой компонента, которая уже связана.

5.2.1 Связывающее указание

Связывающее указание (binding indication) сопоставляет экземпляры компонента с конкретным объектом проекта. Оно также сопоставляет фактические параметры с формальными в интерфейсе объекта проекта.

```
binding_indication ::=
    entity_aspect
    [generic_map_aspect ]
    [port_map_aspect ]
```

Аспект объекта проекта (entity aspect) связывающего указания идентифицирует объект проекта, с которым сопоставляются экземпляры компонента. Аспект отображения параметров настройки (generic map aspect), если задан, идентифицирует выражения, сопоставляемые с формальными параметрами настройки в интерфейсе этого объекта проекта. Аналогично, аспект отображения портов (port map aspect) в связывающем указании идентифицирует сигналы, сопоставляемые с формальными портами в интерфейсе этого объекта проекта.

Если аспект отображения параметров настройки или аспект отображения портов не задан, то применяются правила умолчания (см. раздел *5.2.2*).

5.2.1.1 Аспект объекта проекта

Аспект объекта проекта идентифицирует конкретный объект проекта, сопоставляемый с экземплярами компонентов. Аспект объекта проекта может также задавать отложенное сопоставление.

```
entity_aspect ::=
    entity entity_name [(architecture_identifier) ]
    |configuration configuration_name
    |open
```

Первая форма аспекта объекта проекта идентифицирует отдельное объявление объекта проекта и (необязательно) соответствующее описание архитектуры. Если идентификатор архитектуры не задан, то считается, что непосредственно объемлющее связывающее указание *подразумевает* любой объект проекта, интерфейс которого определяется объявлением объекта, обозначенного именем объекта (entity name). В противном случае считается, что непосредственно объемлющее связывающее указание подразумевает объект проекта, состоящий из объявления объекта, обозначенного именем объекта и описания архитектуры, сопоставленного с этим объявлением, идентификатор архитектуры определяет простое имя, используемое в процессе предвыполнения иерархии проекта для выбора подходящего описания архитектуры. В этом случае соответствующие экземпляры компонента считаются *полностью связанными*.

Вторая форма аспекта объекта проекта идентифицирует объект проекта косвенно через конфигурацию. В этом случае считается, что этот аспект *подразумевает* объект проекта, стоящий в вершине иерархии проекта, которая определяется конфигурацией, обозначенной заданным именем.

Третья форма аспекта объекта проекта используется для задания отложенной идентификации объекта проекта. В этом случае считается, что непосредственно объемлющее связывающее указание не *подразумевает* никакого объекта проекта. Более того, это связывающее указание не должно содержать ни аспекта отображения параметров настройки, ни аспекта отображения портов.

Если идентификатор архитектуры используется в аспекте объекта проекта связывающего указания, используемого в конфигурации компонента, то этот идентификатор должен совпадать с простым именем описания архитектуры, сопоставленного с объявлением объекта, обозначенным соответствующим именем объекта.

5.2.1.2 *Аспект отображения параметров настройки и аспект отображения портов*

Аспект отображения параметров настройки сопоставляет значения с формальными параметрами настройки блока. Аналогично, аспект отображения портов сопоставляет сигналы с формальными портами блока. Нижеследующее верно как для внешних блоков, так и для внутренних блоков, определяемых операторами блока.

```
generic_map_aspect ::=
    generic map (generic_association_list)

port_map_aspect ::=
    port map (port_association_list)
```

В списке сопоставления портов (port association list) или в списке сопоставления параметров настройки (generic association list) можно использовать как именованное, так и позиционное сопоставление.

Далее в тексте используются следующие определения:

1) Термин *фактический параметр* используется для обозначения как фактического указателя, стоящего в элементе сопоставления портов, так и для фактического указателя, стоящего в элементе сопоставления списка сопоставления параметров настройки.

2) Термин *формальный параметр* используется для обозначения как формального указателя, стоящего в элементе сопоставления списка сопоставления портов, так и для формального указателя, стоящего в элементе сопоставления списка сопоставления параметров настройки.

Смысл аспектов сопоставления портов и параметров настройки заключается в сопоставлении фактических параметров с формальными параметрами интерфейса объекта проекта, подразумеваемого непосредственно объемлющем связывающим указанием. Каждый локальный порт или параметр настройки экземпляров компонента, к которым применяется объемлющая спецификация конфигурации, должен быть сопоставлен как фактический параметр с, по крайней мере, одним формальным параметром. Нельзя сопоставлять формальный параметр с более чем одним актуальным параметром.

Фактический параметр, сопоставляемый с формальным параметром настройки в аспекте отображения параметров настройки, должен быть выражением; фактический параметр, сопоставленный с формальным портом в аспекте отображения портов, должен быть сигналом.

На фактический параметр, сопоставляемый с формальным портом в аспекте сопоставления портов, налагается ряд ограничений; эти ограничения описаны в 1.1 | 2.

Формальный параметр для которого нет сопоставления с фактическим параметром, считается несопоставленным формальным параметром.

Примечание — Локальный параметр настройки (из объявления компонента) или формальный параметр настройки (из оператора блока или из объемлющего объекта проекта) может использоваться как фактический параметр в аспекте отображения параметров настройки. Аналогично, локальный порт (из объявления компонента) или формальный порт (из оператора блока или из объемлющего объекта проекта) может использоваться как фактический параметр в аспекте отображения портов.

5.2.2 *Неявное связывающее указание*

В некоторых случаях при отсутствии явного связывающего указания будет применяться неявное связывающее указание. Такое указание состоит из неявного аспекта объекта проекта, неявного аспекта отображения параметров настройки и неявного аспекта отображения портов.

Если нет ни одного видимого объявления объекта, простое имя которого совпадает с простым именем конкретизируемого компонента, то неявный аспект объекта проекта — это зарезервированное слово open. В противном случае, если такое объявление видимо, но не имеет сопоставленного с ним описания архитектуры, неявный аспект объекта проекта имеет вид

```
entity entity_name
```

в котором имя объекта — это простое имя конкретизированного компонента. В противном случае неявный аспект объекта проекта имеет вид

entity *entity*_name (*architecture*_indentifier)

в котором имя объекта — это простое имя конкретизируемого компонента, а идентификатор архитектуры совпадает с простым именем самого последнего проанализированного описания архитектуры, сопоставленного с этим объявлением объекта.

Неявное связывающее указание содержит неявный аспект отображения параметров настройки, если объект проекта, подразумеваемый аспектом объекта проекта, содержит формальные параметры настройки. Неявный аспект отображения параметров настройки сопоставляет каждый локальный параметр настройки в соответствующей конкретизации компонента (если он есть) с формальным параметром, имеющим то же простое имя. Считается ошибкой, если такой формальный параметр отсутствует или если его вид и тип не подходят для такого сопоставления. Все оставшиеся несопоставленные формальные параметры сопоставляются с фактическим указанием **open**.

Неявное связывающее указание содержит неявный аспект отображения портов, если объект проекта, подразумеваемый аспектом объекта проекта, содержит формальные порты. Неявный аспект отображения портов сопоставляет каждый локальный порт в соответствующей конкретизации компонента (если он есть) с формальным портом, имеющим то же простое имя. Считается ошибкой, если такой формальный порт отсутствует, или если его вид и тип не подходят для такого сопоставления. Все оставшиеся несопоставленные формальные порты сопоставляются с фактическим указанием **open**.

Если явное связывающее указание нуждается в аспекте отображения параметров настройки, а объект проекта, подразумеваемый аспектом объекта проекта, содержит формальные параметры настройки, то в этом связывающем указании подразумевается наличие неявного аспекта отображения параметров настройки. Аналогично, если явное связывающее указание нуждается в аспекте отображения портов, а объект проекта, подразумеваемый аспектом объекта проекта, содержит формальные порты, то в этом связывающем указании подразумевается неявный аспект отображения портов.

5.3. Спецификация отключения

Спецификация отключения определяет временную задержку, которая должна использоваться в неявном отключении драйверов защищенного сигнала в операторе назначения защищенного сигнала.

```
disconnection_specification ::=
    disconnect guarded_signal_specification after
    time_expression;

guarded_signal_specification ::=
    guarded_signal_list : type_mark

signal_list ::=
    signal_name {, signal_name}
    | others
    | all
```

Спецификация защищенных сигналов (guarded signal specification) содержит список сигналов, идентифицирующий сигналы, для которых определяется неявная задержка отключения. Определение подчинено следующим правилам:

1) Если задан список имен сигналов, то каждое имя в этом списке должно быть локально статическим именем, обозначающим защищенный сигнал, и спецификация отключения в этом случае применяется ко всем указанным в этом списке сигналам. Эти сигналы должны быть описаны в непосредственно объемлющем разделе объявлений.

2) Если задано зарезервированное слово **others**, то спецификация отключения применяется к драйверам любого сигнала указанного типа, описанным в непосредственно объемлющем разделе объявлений, при условии, что каждый такой сигнал уже явно не задан в списке сигналов предыдущей спецификации отключения.

3) Если задано зарезервированное слово **all**, то спецификация отключения используется для всех сигналов указанного типа, описанных в непосредственно объемлющем разделе объявлений.

Спецификация отключения, список сигналов которой представлен зарезервированным словом **others** или **all** для конкретного типа и которая задана в разделе объявлений, должна быть последней такой спецификацией для этого типа в этой области объявлений. Ни один защищенный сигнал не может быть описан в этом разделе объявлений после такой спецификации отключения.

Выражение времени (time expression) в спецификации отключения должно быть статическим и вырабатывать неотрицательное значение.

Считается ошибкой, если более чем одна спецификация отключения применяется к драйверам одного и того же сигнала.

При отсутствии спецификации отключения для конкретного скалярного сигнала S типа T подразумевается неявная спецификация отключения следующего вида:

disconnect S : T **after** 0ns;

Таким образом, для каждого защищенного сигнала всегда определена неявная задержка отключения либо явной спецификацией отключения, либо неявной.

6 ИМЕНА

6.1 Имена

Имена могут обозначать описанные понятия. Понятия могут быть описаны явно или неявно. Имена также могут обозначать объекты, указываемые ссылочными значениями, и подэлементы или сечения составных объектов и значений. Наконец, имена могут обозначать атрибуты любого из вышеуказанных понятий.

```
name ::=
  simple_name
  |operator_symbol
  |selected_name
  |indexed_name
  |slice_name
  |attribute_name

prefix ::=
  name
  |function_call
```

Некоторые формы имени (индексируемые или составные имена, сечения и имена атрибутов) содержат *префикс*, который может быть именем или вызовом функции. Если префикс имени является вызовом функции, то это имя обозначает элемент, сечение или атрибут, либо результат вызова функции, либо 1, если результат есть ссылочное значение объекта, указываемого этим результатом. Вызовы функции описаны в 7.3.3.

Если типом префикса является ссылочный тип, то этот префикс не должен быть именем, обозначающим формальный параметр вида **out** или его подэлемент.

Считается, что префикс *соответствует* типу, если:

1) тип префикса — это рассматриваемый тип.

2) тип префикса — это ссылочный тип, указываемый тип которого есть рассматриваемый тип.

Вычисление имени определяет понятие, обозначаемое этим именем. Вычисление имени, имеющего префикс, влечет вычисление префикса, являющегося соответствующим именем или вызовом функции. Если типом префикса является ссылочный тип, то вычисление префикса влечет определение объекта, указываемого соответствующим ссылочным значением. В этом случае считается ошибкой, если значением этого префикса является пустое ссылочное значение.

Имя считается *статическим именем*, если каждое выражение, входящее в состав этого имени (например, в качестве выражения индекса), является статическим выражением. Более того, имя считается локально *статическим именем*, если каждое выражение, входящее в состав этого имени, является локально статическим выражением. *Статическое имя сигнала* — это статическое имя, обозначающее сигнал. *Самый длинный статический префикс* имени сигнала — это само имя, если это имя является статическим именем сигнала; в противном случае — это самый длинный префикс имени, являющегося статическим именем сигнала.

Примеры

1) S (C,2) - - статическое имя: C — статическая константа.
2) R (I **to** 16) - - нестатическое имя: I — сигнал.
   - - R — самый длинный статический префикс,
   - - сечение R (I **to** 16).
3) T (n) - - статическое имя: n — константа настройки.
4) T (2) - - локальное статическое имя.

6.2 Простые имена

Простое имя (simple name) понятия — это либо идентификатор, сопоставленный своим объявлением с этим понятием, либо другой идентификатор, сопоставленный объявлением дополнительного имени с этим понятием. В частности, простым именем объекта, конфигурации, пакета, процедуры или функции является идентификатор, стоящий в соответствующем объявлении объекта, объявлении конфигурации, объявлении пакета, объявлении процедуры или объявлении функции соответственно. Простым именем архитектуры является имя, определяемое идентификатором этого архитектурного тела.

simple_name :: = identifier

Вычисление простого имени не имеет никакого другого эффекта, кроме определения понятия, обозначаемого этим именем.

6.3 Составные имена

Составное имя (selected name) используется для обозначения понятия, объявление которого стоит либо внутри объявления другого понятия, либо внутри библиотеки проекта.

selected_name :: = prefix.suffix
suffix :: = simple_name
| character_literal
| operator_symbol
| all

Составное имя может быть использовано для обозначения элемента структуры, объекта, обозначаемого ссылочным значением, или понятия, объявление которого заключено внутри другого наименованного понятия, в частности, внутри библиотеки или пакета. Более того, составное имя может быть использовано для обозначения всех понятий, объявления которых заключены внутри библиотеки или пакета.

Суффикс (suffix) составного имени, используемого для обозначения элемента структуры, должен быть простым именем, обозначающим элемент структурного объекта или значение. Префикс (prefix) должен соответствовать типу этого объекта или значения.

Суффикс составного имени, используемого для обозначения объекта, указываемого ссылочным значением, должен быть зарезервированным словом all. Префикс должен принадлежать к ссылочному типу.

Остальные формы составного имени называются *расширенными именами*. Префиксом расширенного имени не может быть вызов функции.

Расширенное имя обозначает первичный модуль, заключенный в библиотеке проекта, если его префикс обозначает эту библиотеку, а суффикс является простым именем первичного модуля, объявление которого заключено в этой библиотеке. Расширенное имя обозначает все первичные модули, заключенные в библиотеке, если его префикс обозначает эту библиотеку, а суффикс является зарезервированным словом all. Для вторичных модулей, в частности, для архитектурного тела, расширенное имя недопустимо.

Расширенное имя обозначает понятие, объявленное в пакете, если его префикс обозначает этот пакет, а суффикс является простым именем, символьным литералом или символом оператора понятия, объявление которого встречается непосредственно в этом пакете. Расширенное имя обозначает все понятия, объявленные в пакете, если его префикс обозначает этот пакет, а суффикс является зарезервированным словом all.

Расширенное имя обозначает понятие, объявленное непосредственно внутри наименованной конструкции, если его префикс обозначает конструкцию, являющуюся объектом, архитектурой, программой, оператором блока, оператором процесса или оператором цикла, а суффикс является простым именем, символьным литералом или символом оператора понятия, объявление которого встречается непосредственно внутри этой конструкции. Такая форма расширенного имени допускается только внутри самой конструкции.

Примеры

1	INSCRUCTION.OPCODE	- - элемент OPCODE структуры - - INSTRUCTION
2	PTR.all	- - объект, указываемый PTR
3	TTL.SN74LS221	- - модуль проекта, заключенный в библиотеке
4	CMOS.all	- - все модули проекта, заключенные в - - библиотеке CMOS
5	MEASUREMENTS.VOLTAGE	- - понятие, объявленное в пакете

```
6   STANDARD.all          - - все понятия, объявленные в пакете
                          - - STANDARD
7   P.DATA                - - понятие DATA, объявленное в
                          - - процессе P
```

6.4 Индексируемые имена

Индексируемое имя (indexed name) обозначает элемент массива.

indexed_name :: = prefix (expression { , expression })

Префикс индексируемого имени должен соответствовать индексируемому типу. Выражения задают значения индекса для элемента; для каждой позиции индекса в массиве должно быть одно такое выражение. При вычислении индексируемого имени вычисляются префикс и все выражения. Считается ошибкой, если значение индекса не принадлежит соответствующему диапазону индекса этого массива.

Примеры

```
1   REGISTER_ARRAY(5)     - - элемент одномерного массива
2   MEMORY_CELL(1024,7)   - - элемент двумерного массива
```

6.5 Сечения

Сечение (slice name) обозначает одномерный массив, составленный из последовательности рядом стоящих элементов другого одномерного массива. Сечение сигнала является сигналом; сечение переменной является переменной; сечение константы является константой; сечение значения является значением.

slice_name :: = prefix (discrete_range)

Префикс сечения должен соответствовать одномерному индексируемому объекту. Базовый тип этого объекта является базовым типом этого сечения.

Границы дискретного диапазона (discrete range) определяют границы сечения и должны принадлежать типу индексируемого массива. Сечение является *пустым сечением*, если дискретный диапазон является пустым, или, если направление дискретного диапазона не совпадает с направлением диапазона индекса объекта, обозначаемого префиксом.

При вычислении сечения вычисляются префикс и дискретный диапазон. Считается ошибкой, если какая-либо из границ дискретного диапазона не принадлежит диапазону индекса массива, обозначаемого префиксом, если только сечение не является пустым сечением. (Границы пустого сечения могут не принадлежать подтипу индекса).

Примеры

```
1  signal R15: BIT_VECTOR(0 to 31);
2  constant DATA: BIT_VECTOR(31 downto 0);
3  R15(0 to 31)          - - сечение с восходящим диапазоном
4  DATA(24 downto 1)     - - сечение с нисходящим диапазоном
5  DATA(24 to 25)        - - пустое сечение
```

Примечание — Если A является одномерным массивом, то имя A(N to N) или A(N downto N) является сечением, содержащим один элемент; типом сечения является базовый тип массива A. С другой стороны, A(N) является элементом массива A и имеет соответствующий тип элемента.

6.6 Атрибуты

Атрибут (attribute name) обозначает значение, функцию, тип диапазон, сигнал или константу, сопоставленную с некоторым понятием.

attribute_name :: =
prefix'attribute_designator[(static_expression)]
attribute_designator :: = *attribute*_simple_name

Применяемость обозначений атрибутов (attribute designator) зависит от префикса. Смысл префикса атрибута должен определяться независимо от обозначения атрибута и независимо от того факта, что этот префикс является префиксом атрибута.

Если обозначение атрибута определяет предопределенный атрибут, то статическое выражение (static expression) либо должно, либо может быть использовано, что зависит от описания этого атрибута (см. раздел 14); в противном случае оно не должно присутствовать.

Примеры

```
1   REGISTER'LEFT(1)      - - самая левая граница индекса массива
                          - - REGISTER
2   OUTPUT'FANOUT         - - число сигналов, управляемых портом
                          - - OUTPUT
3   CLK'DELAYED(5ns)      - - сигнал CLK, задержанный на 5 нс
```

7 ВЫРАЖЕНИЯ

7.1 Выражения

Выражение — это формула, определяющая вычисление некоторого значения.

```
expression :: =
    relation { and relation }
  | relation { or relation }
  | relation { xor relation }
  | relation [ nand relation ]
  | relation [ nor relation ]

relation :: =
    simple_expression [ relational_operator simple_expression ]

simple_expression :: =
    [ sign ] term { adding_operator term }

term :: =
    factor { multiplying_operator factor }

factor :: -
    primary [ ** primary ]
    | abs primary
    | not primary

primary :: =
    name
    | literal
    | aggregate
    | function_call
    | qualified_expression
    | type_conversion
    | allocator
    | ( expression )
```

Каждое первичное (primary) имеет значение и тип. Единственными допустимыми формами имени (name), используемыми в качестве первичного, являются атрибуты, вырабатывающие значение и имена, обозначающие объекты или значения. Если имя обозначает объект, то значением первичного является значение этого объекта.

Тип выражения (expression) зависит только от типов операндов и применяемых операторов; для совмещенного операнда или совмещенного оператора выявление типа операнда или идентификация совмещенного оператора зависят от контекста (см. 10.5). Для каждого предопределенного оператора типы операндов и результата описаны в 7.2.

Примечание — Синтаксические правила в отношении выражения, содержащего логические операторы, позволяют строить последовательность из операторов **and**, **or** или **xor**, так как соответствующие операции являются ассоциативными; для операторов **nand** и **nor** такая последовательность недопустима, так как соответствующие операции не являются ассоциативными.

7.2 Операторы

Ниже определены операторы, которые могут быть использованы в выражениях. Каждый оператор представлен своим классом операторов, в котором все операторы имеют один и тот же уровень старшинства; классы операторов представлены в порядке возрастания старшинства.

```
logical_operator          : : =   and  |  or  |  nand  |  nor  |  xor
relational_operator       : : =   =  |  /=  |  <  |  <=  |  >  |  >=
adding_operator           : : =   +  |  —  |  &
sign                      : : =   +  |  —  |
multiplying_operator      : : =   *  |  /  |  mod  |  rem
miscellaneous_operator    : : =   **  |  abs  |  not
```

Операторы более высокого уровня старшинства сопоставляются со своими операндами в первую очередь по сравнению с операторами более низкого уровня старшинства. В последовательности операторов одного и того же уровня старшинства сопоставление с операндами происходит в текстуальном порядке слева направо. Старшинство операторов является фиксированным и не может быть изменено пользователем, но наряду с этим для управления порядком сопоставления операторов и операндов могут быть использованы круглые скобки.

В общем случае операнды в выражении вычисляются до их сопоставления с операторами. Для некоторых операций операнд, стоящий справа, вычисляется в том случае, если и только если операнд, стоящий слева, имеет определенное значение. Такие операции называются *укороченными* операциями. Логические операции **and**, **or**, **nand** и **nor**, определенные для операндов типов BIT и BOOLEAN, все являются укороченными операциями. Более того, они являются единственными укороченными операциями.

7.2.1 *Логические операторы*

Логические операторы **and**, **or**, **nand**, **nor**, **xor** и **not** определены для предопределенных типов BIT и BOOLEAN. Они также определены для любого одномерного массива, тип элементов которого есть BIT или BOOLEAN. В последнем случае для бинарных операторов **and**, **or**, **nand**, **nor**, или **xor** операнды должны быть массивами одинаковой длины, операция выполняется над соответствующими элементами этих массивов, а результат является массивом, диапазон индекса которого совпадает с диапазоном индекса левого операнда. Для унарного оператора **not** операция выполняется над каждым элементом операнда, а результат является массивом, диапазон индекса которого совпадает с диапазоном индекса операнда.

В следующих таблицах определены результаты логических операций. Символ T представляет значение TRUE для типа BOOLEAN и '1' для типа BIT; символ F представляет значение FALSE для типа BOOLEAN и '0' для типа BIT;

A	B	A **and** B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A **or** B
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A **xor** B
T	T	F
T	F	T
F	T	T
F	F	F

A	B	A **nand** B
T	T	F
T	F	T
F	T	T
F	F	T

A	B	A **nor** B
T	T	F
T	F	F
F	T	F
F	F	T

A	**not** A
T	F
F	T

Для укороченных операций **and**, **or**, **nand** и **nor** над типами BIT и BOOLEAN правый операнд вычисляется только в том случае, если значения левого операнда не достаточно для определения результата операции. Для операции **and** и **nand** правый операнд вычисляется только в том случае, если значение левого операнда есть T; для операции **or** и **nor** правый операнд вычисляется только в том случае, если значение левого операнда есть F.

Примечание — Все бинарные логические операторы принадлежат к классу операторов самого низкого уровня старшинства. Унарный логический оператор **not** принадлежит к классу операторов с самым высоким уровнем старшинства.

7.2.2 *Операторы отношения*

Операторы отношения включают проверки на равенство, неравенство и упорядоченность операндов. Операнды каждого оператора отношения должны иметь одинаковый тип. Тип результата каждого оператора отношения есть предопределенный тип BOOLEAN.

Оператор	Операция	Тип операндов	Тип результата
=	равенство	любой тип	BOOLEAN
/=	неравенство	любой тип	BOOLEAN
< < = > > =	упорядоченность	любой скалярный тип или дискретный индексируемый тип	BOOLEAN

Операторы равенства и неравенства (= и /=) определены для всех типов, отличных от файловых типов. Оператор равенства возвращает значение TRUE, если два операнда равны, или в противном случае значение FALSE. Оператор неравенства возвращает значение FALSE, если два операнда равны, или в противном случае значение TRUE.

Два скалярных значения одного и того же типа равны, если и только если эти значения являются одним и тем же. Два составных значения одного и того же типа равны, если и только если для каждого элемента левого операнда имеется *соответствующий элемент* правого операнда (и наоборот) и значения соответствующих элементов равны (в соответствии с предопределенным оператором равенства для типа элемента). В частности, два пустых массива одного типа всегда равны. Два значения ссылочного типа равны, если и только если они оба указывают на один и тот же объект, или оба они равны пустому значению.

Для двух структурных значений соответствующими элементами являются те элементы, которые имеют один и тот же идентификатор элементов. Для двух одномерных массивов соответствующими элементами являются те элементы (если они есть), значения индексов которых соответствуют, то есть: левые границы диапазонов индексов определены таким образом, что они соответствуют; если два элемента соответствуют, то элементы, непосредственно стоящие справа, также определены таким образом, что они соответствуют. Для двух многомерных массивов соответствующими элементами являются те элементы, индексы которых соответствуют последовательно по позициям.

Операторы упорядоченности определены для любого скалярного типа и для любого дискретного индексируемого типа. Дискретный массив — это одномерный массив, элементы которого имеют дискретный тип. Каждый оператор возвращает значение TRUE, если соответствующее отношение удовлетворяется, или в противном случае значение FALSE.

Для скалярных типов упорядоченность определена в терминах относительных значений. Для дискретных индексируемых типов отношение < (меньше чем) определено таким образом, что левый операнд меньше чем правый операнд, если и только если:

1) Левый операнд является пустым массивом, а правый операнд таковым не является;

2) оба операнда не являются пустыми массивами и удовлетворяется одно из следующих условий:

а) самый левый элемент левого операнда меньше чем самый левый элемент правого операнда,

б) самый левый элемент левого операнда равен самому левому элементу правого операнда, а остаток левого операнда меньше чем остаток правого операнда (остаток состоит из оставшихся элементов, стоящих справа от самого левого элемента, и может быть пустым).

Отношение <= (меньше или равно) для дискретных индексируемых типов определено как исключающая дизъюнкция результатов операторов < и = для тех же двух операндов. Отношения > (больше) и >= (больше или равно) определены как дополнения операторов <= и < для тех же двух операндов соответственно.

7.2.3 *Аддитивные операторы*

Аддитивные операторы + и — предопределены для любого числового типа и имеют обычный смысл. Оператор конкатенации & предопределен для любого одномерного индексируемого типа

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата
+	сложение	любой числовой тип	тот же тип	тот же тип
—	вычитание	любой числовой тип	тот же тип	тот же тип
&	конкатенация	любой индексируемый тип	тот же индексируемый тип	тот же индексируемый тип
		любой индексируемый тип	тип элемента	тот же индексируемый тип
		тип элемента	любой индексируемый тип	тот же индексируемый тип
		тип элемента	тип элемента	любой индексируемый тип

Для операции конкатенации возможны три случая:

1) Если оба операнда являются одномерными массивами, то результатом конкатенации является одномерный массив, длина которого равна сумме длин операндов, состоящий из элементов левого операнда (в порядке слева направо), за которыми следуют элементы правого операнда (в том же порядке). Левой границей этого результата является левая граница левого операнда, если только он не является пустым массивом (в этом случае результатом операции является правый операнд). Направление результата совпадает с направлением левого операнда, если только он не является пустым массивом (в этом случае направление результата совпадает с направлением правого операнда).

2) Если только один из операндов является одномерным массивом, то результат конкатенации определяется правилами для случая (1), при условии, что вместо второго операнда используется неявный массив, единственным элементом которого является этот операнд. Левой границей такого массива является левая граница подтипа индекса этого массива, а его направление является восходящим (нисходящим), если подтип индекса является восходящим (нисходящим).

3) Если ни один из операндов не является одномерным массивом, то тип результата должен быть известен из контекста. Этот тип должен быть таким, чтобы каждый операнд являлся элементом неявного массива, и тип этого массива совпадал с типом результата. Подтип неявного массива определяется по правилам для случая (2), а результат конкатенации по правилам для случая (1).

Операторы знака + и — предопределены для любого числового типа и имеют обычный смысл. Для каждого из этих унарных операторов операнд и результат имеют одинаковый тип.

Примечание — В силу относительного предшествования знаков + и — в грамматике выражений операнд со знаком не должен следовать за мультипликативным оператором, оператором возведения в степень ** или операторами abs и not. Например, синтаксические правила не допускают написание следующих выражений:

1) A/+B - - недопустимое выражение
2) A**—B - - недопустимое выражение

Однако эти выражения могут быть переписаны в допустимой форме следующим образом:

1) A/(+B) - - допустимое выражение
2) A**(—B) - - допустимое выражение

7.2.4 *Мультипликативные операторы*

Операторы * и / предопределены для любого целого и для любого плавающего типа и имеют обычный смысл; операторы **mod** и **rem** предопределены для любого целого типа. Для каждого из этих операторов операнды и результат имеют один и тот же тип.

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата
*	умножение	любой целый тип	тот же тип	тот же тип
		любой плавающий тип	тот же тип	тот же тип
/	деление	любой целый тип	тот же тип	тот же тип
		любой плавающий тип	тот же тип	тот же тип
mod	модуль	любой целый тип	тот же тип	тот же тип
rem	остаток	любой целый тип	тот же тип	тот же тип

Целочисленное деление и остаток определены следующим отношением:

A=(A/B)*B+(A **rem** B),

где (A **rem** B) имеет знак операнда A и абсолютное значение меньшее, чем абсолютное значение операнда B. Целочисленное значение удовлетворяет следующему равенству:

(—A)/B=—(A/B)=A/(—B).

Результат операции деления по модулю таков, что (A **mod** B) имеет знак операнда B и абсолютное значение меньшее, чем абсолютное значение операнда B. Для некоторого целого значения N этот результат должен удовлетворять отношению:

A=B*N+(A **mod** B).

В дополнение к вышесказанному операторы * и / предопределены для любого физического типа.

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата
*	умножение	любой физический тип	INTEGER	совпадает с типом левого операнда
		любой физический тип	REAL	совпадает с типом левого операнда
		INTEGER	любой физический тип	совпадает с типом правого операнда
		REAL	любой физический тип	совпадает с типом правого операнда
/	деление	любой физический тип	INTEGER	совпадает с типом левого операнда
		любой физический тип	REAL	совпадает с типом левого операнда
		любой физический тип	тот же тип	*универсальный целый*

Умножение значения P физического типа T\Sp\s на значение I типа INTEGER эквивалентно следующему вычислению:

T\Sp\s'Val(T\Sp\s'Pos(P)*I)

Умножение значения P физического типа T\Sp\s на значение F типа REAL эквивалентно следующему вычислению:

T\Sp\s'Val(INTEGER(REAL(T\Sp\s'Pos(P))*F))

Деление значения P физического типа T\Sp\s на значение типа INTEGER эквивалентно следующему вычислению:

T\Sp\s'Val(T\sp\s'Pos(P)/I)

Деление значения P физического типа T\Sp\s на значение F типа REAL эквивалентно следующему вычислению:

T\Sp\s'Val(INTEGER(REAL(T\Sp\s'Pos(P))/F))

Деление значения P физического типа T\Sp\s на значение P2 того же физического типа эквивалентно следующему вычислению:

T\Sp\s'Pos(P)/T\Sp\s'Pos(P2)

7.2.5 *Прочие операторы*

Унарный оператор abs предопределен для любого числового типа.

Оператор	Операция	Тип операндов	Тип результата
abs	абсолютное значение	любой числовой тип	тот же числовой тип

Оператор возведения в степень ** предопределен для каждого целого типа и для каждого плавающего типа. В любом случае правый операнд, называемый экспонентой, имеет предопределенный тип INTEGER.

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата
**	возведение в степень	любой целый тип	INTEGER	совпадает с типом левого операнда
		любой плавающий тип	INTEGER	совпадает с типом левого операнда

Возведение в степень с использованием целой экспоненты эквивалентно умножению левого операнда самого на себя столько раз, сколько задано абсолютным значением экспоненты, и слева направо; если экспонента отрицательна, то результат представляет собой обратную дробь того, что было получено с абсолютным значением экспоненты.

Возведение в степень с отрицательной экспонентой допускается, если левый операнд имеет плавающий тип. Возведение в степень с нулевой экспонентой дает значение 1. Возведение в степень значения плавающего типа приблизительно.

7.3 Операнды

Операндами в выражении могут быть имена (обозначающие объекты, значения или атрибуты, производящие значения), литералы, агрегаты, вызовы функции, квалифицированные выражения, преобразования типа и генераторы. К тому же выражение, заключенное в круглые скобки, может быть операндом в другом выражении.

Имена определены в 6.1; остальные виды операндов определены в 7.3.1—7.3.6.

7.3.1 *Литералы*

Литерал — это либо числовой литерал, либо литерал перечисления, либо строковый литерал, либо битовый литерал, либо литерал null.

```
literal :: =
  numeric_literal
  | enumeration_literal
  | string_literal
  | bit_string_literal
  | null

numeric_literal :: =
  abstract_literal
  | physical_literal
```

Числовые литералы (numeric_literal) включают литералы абстрактных типов *универсальный_целый* и *универсальный_действительный*, а также литералы физических типов. Абстрактные литералы определены в разделе 13; физические литералы определены в 3.13.

Литералы перечисления (enumeration literal) — это литералы перечисляемых типов. Они включают как идентификаторы, так и символьные литералы. Перечисляемые литералы определены в 3.1.1.

Строковые (string literal) и битовые (bit string literal) литералы являются представлениями одномерных символьных массивов. Тип строкового или битового литерала должен определяться исключительно из контекста, в котором этот литерал используется, не учитывая при этом сам литерал, но используя тот факт, что типом этого литерала должен быть одномерный массив символьного типа (для строковых литералов) или типа BIT (для битовых литералов). Лексическая структура строковых и битовых литералов определена в разделе 13.

Для строковых и битовых литералов, представляющих непустые индексируемые значения, направление и границы этого значения определяются согласно правилам для позиционных агрегатов массивов, где число элементов агрегата эквивалентно длине (см. 13.6 и 15.7) строкового или битового литерала. Для строкового литерала, представляющего пустой массив, направление и самая левая граница этого массива определяются так же, как и для других строковых литералов. Если направление является восходящим, то самая правая граница — это значение, предшествующее (как установлено атрибутом PRED) значению левой границы; в противном случае самой правой границей является значение, следующее (как установлено атрибутом SUCC) за значением самой левой границы.

Символьные литералы, соответствующие графическим символам, содержащимся внутри строкового литерала или битового литерала, должны быть видимыми на месте строкового литерала.

Литерал null представляет пустое ссылочное значение для любого ссылочного типа.

Вычисление литерала производит соответствующее значение.

Примеры

1	3.14159_26536	- - литерал типа *универсальный_действительный*
2	5280	- - литерал типа *универсальный_целый*
3	10.7ns	- - литерал физического типа
4	0"4777"	- - литерал типа BIT_STRING
5	"54LS281"	- - литерал типа STRING
6	" "	- - строковый литерал, представляющий пустой
7		- - массив

7.3.2 *Агрегаты*

Агрегат объединяет одно или несколько значений в составное значение структурного или индексируемого типа.

```
aggregate :: =
  ( element_association { , element_association } )

element_association :: =
  [ choices => ] expression

choices :: = choice { | choice }

choice :: =
  simple_expression
  | descrete_range
  | element_simple_name
  | others
```

Каждое сопоставление элементов (element association) сопоставляет выражение с элементами. Сопоставление элементов считается *именованным*, если элементы задаются явно при помощи выборов; в противном случае сопоставление считается *позиционным*. В позиционном сопоставлении каждый элемент неявно задается позицией в соответствии с текстуальным порядком элементов в соответствующем объявлении типа.

В одном и том же агрегате (aggregate) могут быть использованы как именованные, так и позиционные сопоставления. При этом в начале должны задаваться все позиционные сопоставления (в текстуальном порядке), а затем все именованные сопоставления в любом порядке, за исключением того, что никакое сопоставление не может стоять после сопоставления **others**. Агрегаты, содержащие единственное сопоставление элементов, должны всегда задаваться с использованием именованного сопоставления для того, чтобы отличать их от выражений, заключенных в круглые скобки.

Сопоставление элементов, в котором выбор (choice) задан как простое имя (simple name), разрешен только в агрегате структуры. Сопоставление элементов, в котором выбор задан в виде простого выражения или дискретного диапазона, допустимо только в агрегате массива: простое выражение задает элемент с соответствующим значением индекса; дискретный диапазон задает элементы для каждого значения индекса в этом диапазоне. Сопоставление элементов с единственным выбором **others** допустимо в том случае, когда он задает все оставшиеся элементы (если они есть).

Каждый элемент значения, определяемого агрегатом, должен быть представлен раз и только раз в этом агрегате.

Тип агрегата должен определяться исключительно из контекста, в котором этот агрегат используется, исключая сам агрегат, но используя тот факт, что типом этого агрегата должен быть составной тип. Тип агрегата, в свою очередь, определяет требуемый тип для каждого из его элементов.

7.3.2.1 *Агрегаты структур*

Если типом агрегата является структурный тип, то имена элементов, заданные как выборы, должны обозначать элементы этого структурного типа. Если в качестве выбора агрегата структуры задан выбор **others** структуры, то он должен представлять по крайней мере один элемент. Сопоставление элементов с более чем одним выбором или с выбором **others** допускается только в том случае, если заданные элементы имеют один и тот же тип. Выражение, используемое в сопоставлении элементов, должно иметь тип сопоставляемых с ним элементов структуры.

7.3.2.2 *Агрегаты массивов*

Для агрегата одномерного индексируемого типа каждый выбор должен задавать значения, принадлежащие к типу индекса, а выражение, стоящее в каждом сопоставлении элементов, должно иметь тип элементов. Агрегат n-мерного индексируемого типа, где n больше 1, записывается как одномерный агрегат, в котором подтип индекса этого агрегата задан первой позицией индекса индексируемого типа, а выражение, заданное для каждого сопоставления элементов, является (n — 1)-мерным массивом или агрегатом массива. В многомерных агрегатах строковые и битовые литералы допускаются вместо одномерного массива символьного типа.

В противоположность последнему сопоставлению элементов с единственным выбором **others**, все остальные сопоставления элементов (если они есть) агрегата массива должны быть либо позиционными, либо именованными. Именованное сопоставление агрегата массива может иметь выбор, который не является локально статическим, или выбор в виде пустого диапазона при условии, что этот агрегат содержит единственное сопоставление элементов, которое, в свою очередь, имеет единственный выбор. Выбор others является локально статическим, если применяемое ограничение индекса является локально статическим.

Подтип агрегата массива, имеющего выбор others, должен определяться из контекста. Это означает, что такой агрегат может использоваться только:

1) Как фактический параметр, сопоставляемый с формальным параметром или формальным параметром настройки, объявленными с ограниченным индексируемым подтипом.

2) Как неявное выражение, определяющее неявное начальное значение порта, объявленного с ограниченным индексируемым подтипом.

3) Как результирующее выражение функции, где соответствующий тип результата функции является ограниченным индексируемым подтипом.

4) Как выражение значения в операторе присваивания, в котором левая часть является объявленным объектом и подтипом этого объекта является ограниченный индексируемый тип.

5) Как выражение, определяющее начальное значение константы или переменной, при этом эти объекты объявлены с ограниченным индексируемым типом.

6) Как выражение, определяющее начальное значение драйверов одного или более сигналов в спецификации инициализации, в которой соответствующий подтип является ограниченным индексируемым подтипом.

7) Как выражение, определяющее значение атрибута в спецификации атрибута, где этот атрибут объявлен с ограниченным индексируемым подтипом.

8) Как операнд в квалифицированном выражении, обозначение типа которого указывает на ограниченный индексируемый подтип.

9) Как подагрегат многомерного агрегата, при этом последний сам используется в одном из перечисленных контекстов.

Границы массива, который не имеет выбор others, определяются по следующим правилам. Если агрегат используется в одном из вышеуказанных контекстов, то направление подтипа индекса этого агрегата совпадает с направлением соответствующего ограниченного индексируемого подтипа; в противном случае направление подтипа индекса этого агрегата совпадает с подтипом индекса базового типа этого агрегата. Для агрегата, имеющего именованные сопоставления, самая левая и самая правая границы определяются направлением подтипа индекса этого агрегата и заданными наименьшим и наибольшим выборами. Для позиционного агрегата самая левая граница определяется применяемым ограничением индекса, если этот агрегат используется в одном из вышеуказанных контекстов; в противном случае самая левая граница представлена атрибутом S'LEFT, где S является подтипом индекса базового типа массива; в любом случае самая правая граница определяется направлением подтипа индекса и количеством элементом.

7.3.3 *Вызовы функций*

Вызов функции (function call) активизирует выполнение тела функции. Вызов задает имя активизируемой функции и фактические параметры, если они есть, сопоставляемые с формальными параметрами этой функции.

Результатом выполнения тела функции является значение, тип которого задается в объявлении активизируемой функции.

```
function_call :: =
  function_name [ ( actual_parameter_part ) ]

actual_parameter_part :: = parameter_association_list
```

Для каждого формального параметра функции вызов функции должен задавать точно один соответствующий фактический параметр. Этот фактический параметр задается либо явно элементом сопоставления в списке сопоставлений (association list), либо (при отсутствии такого элемента сопоставления) неявным выражением (см. 4.3.3.).

Вычисление вызова функции включает в себя вычисление выражений фактических параметров, заданных в вызове, и вычисление неявных выражений, сопоставляемых с формальными параметрами функции, которые не имеют сопоставленных с ними фактических параметров. В обоих случаях результирующее значение должно принадлежать подтипу сопоставляемого формального параметра. (Если формальный параметр принадлежит к неограниченному индексируемому типу, то этот формальный параметр заимствует подтип фактического параметра.) Тело функции выполняется с использованием значений фактических параметров и значений неявных выражений, взятых в качестве значений соответствующих формальных параметров.

7.3.4 *Квалифицированные выражения*

Квалифицированное выражение используется для явного утверждения типа и, возможно, подтипа операнда, представленного в виде выражения или агрегата.

```
qualified_expression :: =
    type_mark ' ( expression )
  | type_mark ' aggregate
```

Тип операнда и базовый тип обозначения типа (type mark) должны быть одним и тем же типом. Значением квалифицированного выражения является значение операнда. Вычисление квалифицированного выражения состоит из вычисления операнда и проверки на принадлежность его значения подтипу, заданному обозначением типа.

Примечание — Квалифицированное выражение может быть использовано для явного утверждения типа в тех случаях, когда тип литерала перечисления или агрегата нельзя установить по контексту.

7.3.5 *Преобразование типа*

Преобразование типа обеспечивает явное преобразование между родственными типами.

type_conversion :: = type_mark (expression)

Целевой тип преобразования типа — это базовый тип обозначения типа (type mark). Тип операнда преобразования типа должен определяться независимо от контекста (в частности, независимо от целевого типа). Более того, операнд преобразования типа не может быть литералом null, генератором, агрегатом или строковым литералом. В качестве операнда преобразования типа может использоваться выражение, заключаемое в круглые скобки, при условии, что это выражение имеет допустимую форму и без скобок.

Если обозначение типа задает подтип, то сначала выполняется преобразование в целевой тип, а затем проверка результата преобразования на принадлежность этому подтипу.

Явное преобразование типа допускается между родственными типами. В частности, допускается преобразование операнда конкретного типа в сам тип. Ниже приведены другие допустимые явные преобразования типа:

а) Абстрактные числовые типы.

Операнд может быть любого целого или плавающего типа. Значение операнда преобразуется в целевой тип, который также должен быть целым или плавающим. Преобразование плавающего значения в целый тип вызывает округление до ближайшего целого; в зависимости от расстояния до ближайших двух целых значений округление может быть в меньшую или в большую сторону.

б) Индексируемые типы.

Преобразование допустимо если и тип операнда и целевой тип являются индексируемыми типами, удовлетворяющими следующим условиям:

1) оба типа имеют одинаковую размерность;

2) в каждой позиции индексов типы индексов либо являются одним и тем же типом, либо взаимно преобразуемыми типами;

3) типы элементов являются одним и тем же типом.

Если обозначение типа задает неограниченный индексируемый тип, то для каждой позиции индекса границы результата получаются преобразованием границ операнда в соответствующий тип индекса целевого типа. Если обозначение типа задает ограниченный индексируемый подтип, то границы результата — это границы, налагаемые этим обозначением. В любом случае, значение каждого элемента результата — это значение соответствующего элемента операнда (см. 7.2.2).

В случае преобразования между числовыми типами считается ошибкой, если результат преобразования не удовлетворяет ограничению, налагаемому обозначением типа.

В случае преобразований между индексируемыми типами делается проверка того, что любое ограничение на подтип элемента, налагаемое индексируемым типом операнда и целевым индексируемым типом, это одно и то же ограничение. Если обозначение типа задает неограниченный индексируемый тип, то для каждой позиции индекса делается проверка, что границы результата принадлежат соответствующему подтипу индекса целевого типа. Если обозначение типа задает ограниченный индексируемый подтип, то делается проверка того, что для каждого элемента операнда существует соответствующий элемент целевого подтипа, и наоборот. Считается ошибкой, если какая-либо из этих проверок не имеет успеха.

В определенных случаях выполняются неявные преобразования типа. Неявное преобразование операнда типа *универсальный целый* в другой целый тип или операнда типа *универсальный действительный* в другой действительный тип может быть применено только в случае, если операнд является числовым литералом или атрибутом, либо если операнд является выражением, содержащим операцию деления значения физического типа на значение того же типа; такой операнд называется *преобразуемым* универсальным операндом. Неявное преобразование преобразуемого универсального операнда применяется, если и только если самый внутренний полный контекст определяет уникальный (числовой) целевой тип для неявного преобразования и не существует правильной интерпретации этого контекста без этого преобразования.

Примечание — Два индексируемых типа могут быть родственными, даже если соответствующие позиции индексов имеют разные направления.

7.3.6 *Генераторы*

Вычисление генератора создает объект и вырабатывает ссылочное значение, которое указывает на этот объект.

```
allocator :: =
  new  subtype_indication
  | new qualified_expression
```

Типом объекта, создаваемого генератором (allocator), является базовый тип обозначения типа, заданного либо в указании подтипа (subtype indication), либо в квалифицированном выражении (subtype expression). Для генератора, в котором используется указание подтипа, начальным значением создаваемого объекта является то же значение, которое задается неявно в объявлении переменной (неявное начальное значение) указанного подтипа. Для генератора, в котором используется квалифицированное выражение, начальное значение создаваемого объекта определяется этим выражением.

Тип ссылочного значения, возвращаемого генератором, должен определяться исключительно из контекста, но с использованием того факта, что возвращаемое значение является ссылкой на указанный в генераторе тип.

Единственной допустимой формой ограничения, используемого в указании подтипа генератора, является ограничение индекса. Если генератор содержит указание подтипа и если типом создаваемого объекта является индексируемый тип, то указание подтипа должно либо обозначать ограниченный подтип, либо содержать явное ограничение индекса. Указание подтипа, являющееся частью генератора, не должно содержать функцию разрешения.

Если типом создаваемого объекта является индексируемый тип, то создаваемый объект всегда ограничен. Если генератор содержит указание подтипа, то создаваемый объект ограничен этим подтипом. Если генератор содержит квалифицированное выражение, то создаваемый объект ограничен пределами начального значения, определяемого этим выражением. Для других типов подтипом создаваемого объекта является подтип, определяемый подтипом, используемым в описании ссылочного типа.

При вычислении генератора сначала предвыполняется указание подтипа или вычисляется квалифицированное выражение. Затем создается новый объект и ему присваивается начальное значение. В заключение возвращается ссылочное значение, которое указывает на созданный объект.

При отсутствии явного освобождения памяти реализация должна гарантировать, что любой объект, создаваемый вычислением генератора, будет существовать до тех пор, пока на этот объект или его подэлемент существует прямая или косвенная ссылка; то есть до тех пор, пока существует некоторое имя, с помощью которого можно ссылаться на этот объект.

Примечание — Для каждого ссылочного типа неявно объявлена процедуры Deallocate. Эта процедура обеспечивает механизм для явного освобождения памяти, занимаемой объектом, созданным при помощи генератора.

Реализация может (но не обязательно) обеспечивать освобождение памяти, занимаемой объектом, созданным при помощи генератора, как только все ссылки на него аннулируются.

Примеры

```
1       new NODE                                  - - принимает неявное начальное
                                                  - - значение
2       new NODE' (15ns, null)                    - - начальное значение задано
                                                  - - явно
3       new NODE' (Delay => 5ns, Next => Stack)
                                                  - - начальное значение задано
                                                  - - явно
4       new BIT_VECTOR'("00110110")               - - ограничен начальным
                                                  - - значением
5       new STRING(1 to 10)                       - - ограничен пределами индекса
6       new STRING                                - - неправильно: должно быть
                                                  - - задано ограничение
```

7.4 Статические выражения

Некоторые выражения называются *статическими*. Аналогично, статическими называются некоторые дискретные диапазоны, а обозначения типов для некоторых подтипов называют обозначающими статические подтипы.

Существует две категории статических выражений. Некоторые формы выражения могут быть вычислены в процессе анализа объекта проекта, в котором они используются; такое выражение называется *локально*

статическим, т.к. его значение зависит только от объявлений, которые являются локальными по отношению к этому объекту проекта или по отношению к пакетам, которые используются этим объектом проекта. Некоторые формы выражения могут быть вычислены в процессе предвыполнения иерархии проекта, в которой они используются; такое выражение называется *глобально статическим*, так как его значение может зависеть от объявлений, стоящих в других модулях проекта этой иерархии или от самого процесса предвыполнения. Считается, что локально статическое выражение также может быть глобально статическим.

Считается, что выражение будет локально статическим, если и только если каждый оператор в этом выражении обозначает предопределенный оператор, операнды и результат которого являются скалярными, и каждое первичное в этом выражении является *локально статическим первичным*, которое может быть одним из приведенного ниже:

1) литералом любого типа;

2) константой (за исключением неполной константы), явно описанной объявлением константы с использованием локально статического подтипа и инициализированной при помощи локально статического выражения;

3) вызовом функции, имя которой обозначает предопределенный оператор и фактические параметры которой представлены каждый локально статическим выражением;

4) предопределенным атрибутом локально статического поддтипа, являющимся значением;

5) предопределенным атрибутом локально статического подтипа, являющимся функцией, в которой фактические параметры представлены локально статическими выражениями;

6) определяемым пользователем атрибутом, значение которого определяется локально статическим выражением;

7) квалифицированным выражением, обозначение типа которого указывает на локально статический подтип и операнды которого представлены локально статическими выражениями;

8) локально статическим выражением, заключенным в круглые скобки;

Локально статическим диапазоном является диапазон, границы которого представлены локально статическими выражениями. Локально статическим ограничением диапазона является ограничение диапазона, диапазон которого является локально статическим. Локально статическим скалярным подтипом является либо скалярный базовый тип, либо скалярный подтип, формируемый наложением на локально статический подтип локально статического ограничения диапазона. Локально статическим дискретным диапазоном является либо локально статический подтип, либо локально статический диапазон.

Локально статическим ограничением индекса является ограничение индекса, в котором каждый подтип индекса соответствующего индексируемого типа является локально статическим, и в котором каждый дискретный диапазон является локально статическим. Локально статическим индексируемым подтипом является ограниченный индексируемый подтип, формируемый наложением на неограниченный индексируемый тип локально статического ограничения индекса.

Локально статическим подтипом является либо локально статический скалярный подтип, либо локально статический индексируемый подтип.

Считается, что выражение будет глобально статическим, если и только если каждый оператор в этом выражении обозначает предопределенный оператор и каждое первичное в этом выражении является *глобально статическим первичным*, которое может быть одним из приведенного ниже:

1) локально статическим первичным;

2) константой настройки;

3) параметром оператора генерации;

4) константой (включая неполную константу) явно описанной объявлением константы с использованием глобально статического подтипа и инициализированной при помощи статического выражения;

5) агрегатом глобально статического подтипа, сопоставления элементов которого содержат только глобально статические выражения;

6) вызовом функции, имя которой обозначает предопределенный оператор и фактические параметры которой представлены каждый глобально статическим выражением;

7) предопределенным атрибутом глобально статического подтипа, являющимся значением или диапазоном;

8) предопределенным атрибутом глобально статического подтипа, являющимся функцией, в которой фактические параметры представлены глобально статическими выражениями;

9) определяемым пользователем атрибутом, значение которого определяется глобально статическим выражением;

10) квалифицированным выражением, обозначение типа которого указывает на глобально статический подтип и операнд которого представлен глобально статическим выражением;

11) глобально статическим выражением, заключенным в круглые скобки.

Глобально статическим диапазоном является диапазон, границы которого представлены глобально статическими выражениями. Глобально статическим ограничением диапазона является ограничение

диапазона, диапазон которого является глобально статическим. Глобально статическим скалярным подтипом является либо скалярный базовый тип, либо скалярный подтип, формируемый наложением на глобально статический подтип глобально статического ограничения диапазона. Глобально статическим дискретным диапазоном является либо глобально статический подтип, либо глобально статический диапазон.

Глобально статическим ограничением индекса является ограничение индекса, в котором каждый подтип индекса соответствующего индексируемого типа является глобально статическим. Глобально статическим индексируемым подтипом является ограниченный индексируемый подтип, формируемый наложением на неограниченный индексируемый тип глобального статического ограничения индекса.

Глобально статическим подтипом является либо глобально статический скалярный подтип, либо глобально статический индексируемый подтип.

Примечание — Выражение, от которого требуется быть статическим, может быть либо локально статическим выражением, либо глобально статическим выражением. Аналогично, диапазон, ограничение диапазона, скалярный подтип, дискретный диапазон, ограничение индекса или индексируемый подтип, от которых требуется быть статическими, могут быть либо локально статическими, либо глобально статическими.

7.5 Универсальные выражения

Универсальным выражением является либо выражение, производящее результат типа *универсальный_целый*, либо выражение, производящее результат типа *универсальный_действительный*.

Для любого целого типа и для типа *универсальный_целый* предопределены одни и те же операции. Для любого плавающего типа и для типа *универсальный_действительный* предопределены одни и те же операции. В дополнение к этим операциям существуют еще следующие операции умножения и деления:

Оператор	Операция	Тип левого операнда	Тип правого операнда	Тип результата
*	умножение	*универсальный_действительный*	*универсальный_целый*	*универсальный_действительный*
		универсальный_целый	*универсальный_действительный*	*универсальный_действительный*
/	деление	*универсальный_действительный*	*универсальный_целый*	*универсальный_действительный*

Точность вычисления универсального выражения типа *универсальный_действительный* обязана быть не ниже точности самого точного предопределенного плавающего типа, обеспечиваемого реализацией, исключая сам *универсальный_действительный* тип. Более того, если это выражение является статическим выражением, то вычисление должно быть точным.

При вычислении операции нестатического универсального выражения применяются следующие правила. Если результат имеет тип *универсальный_целый*, то значения операндов и результат должны принадлежать диапазону целого типа, имеющего самый широкий диапазон, обеспечиваемый реализацией, исключая сам *универсальный_целый* тип. Если результат имеет тип *универсальный_действительный*, то значения операндов и результат должны принадлежать диапазону плавающего типа, имеющего самый широкий диапазон, обеспечиваемый реализацией, исключая сам *универсальный_действительный* тип.

8 ПОСЛЕДОВАТЕЛЬНЫЕ ОПЕРАТОРЫ

В данном разделе описаны различные формы последовательных операторов. Последовательные операторы используются для определения алгоритмов выполнения подпрограммы или процесса; они выполняются в том порядке, в котором заданы.

```
sequence_of_statements :: =
  { sequential_statement }

sequential_statement :: =
  wait_statement
  | assertion_statement
  | signal_assignment_statement
  | variable_assignment_statement
  | procedure_call_statement
  | if_statement
```

| case_statement
| loop_statement
| next_statement
| exit_statement
| return_statement
| null_statement

Некоторые последовательные операторы могут быть помечены. Используемые при этом метки неявно описываются в начале раздела объявлений самого внутреннего объемлющего оператора процесса или тела подпрограммы.

8.1 Оператор ожидания

Оператор ожидания (wait statement) вызывает приостановку выполнения оператора процесса или процедуры.

```
wait_statement :: =
  wait [ sensitivity_clause ]
    [ condition_clause ][ timeout_clause ];

sensitivity_clause :: = on sensitivity_list

sensitivity_list :: = signal_name { , signal_name }

condition_clause :: = until condition

condition :: = boolean_expression

timeout_clause :: = for time_expression
```

Описание чувствительности (sensitivity clause) определяет *множество чувствительности* оператора ожидания, т.е. множество сигналов, к которым этот оператор чувствителен. Каждое имя сигнала (signal name) в списке чувствительности (sensitivity list) идентифицирует конкретный сигнал как элемент этого множества. Каждое имя сигнала в списке чувствительности должно быть статическим именем и каждое имя должно обозначать сигнал, для которого чтение разрешено. Если описание чувствительности не задано, то множество чувствительности будет содержать сигналы, обозначаемые самым длинным статическим префиксом каждого имени сигнала, используемого как первичное в условии, заданном в описании условия (condition clause).

Если в списке чувствительности используется имя сигнала, обозначающее сигнал составного типа, то это эквивалентно тому, что в этом списке стоит имя каждого скалярного подэлемента этого сигнала.

Описание условия задает условие, которое должно удовлетворяться для продолжения выполнения процесса. Если описание условия не задано, то подразумевается описание until TRUE.

Описание времени блокировки (timeout clause) задает максимальное количество времени, в течение которого процесс будет оставаться приостановленным на данном операторе ожидания. Если описание времени блокировки не задано, то подразумевается описание for (STD.STANDARD.TIME'HIGH — STD.STANDARD.LOW). Считается ошибкой, если выражение времени в описании времени блокировки вырабатывает отрицательное значение.

Выполнение оператора ожидания влечет вычисление выражения времени (time expression) для выявления интервала времени блокировки. Оно также влечет приостановку выполнения соответствующего оператора процесса. При этом под последним подразумевается либо оператор процесса, содержащий этот оператор ожидания, либо родитель (см. 2.2) процедуры, содержащей этот оператор ожидания. Приостановленный процесс будет возобновлен сразу по истечении интервала времени блокировки.

Возобновление процесса также может произойти в результате события, происходящего на любом из сигналов, входящих в множество чувствительности оператора ожидания. Если такое событие происходит, то вычисляется условие, заданное в описании условия. Если значением этого условия является TRUE, то процесс будет возобновлен. Если значением условия является FALSE, то процесс остается приостановленным. Повторное приостановление не влечет перевычисления интервала времени блокировки.

Считается ошибкой, если оператор ожидания используется внутри функции или процедуры, родителем которой является функция. Более того, считается ошибкой, если оператор ожидания используется в явном операторе процесса, содержащем список чувствительности, или в процедуре, родителем которой является такой оператор процесса.

8.2 Оператор утверждения

Оператор утверждения (assertion statement) проверяет истинность заданного условия и формирует сообщение об ошибке, если это условие ложно.

```
assertion_statement :: =
  assert  condition
    [ report expression ]
    [ severity expression ];
```

Если описание сообщения задано, то оно должно содержать выражение предопределенного типа STRING, задающее текст сообщения. Если описание серьезности задано, то оно должно содержать выражение предопределенного типа SEVERITY_LEVEL, задающее уровень серьезности утверждения.

Описание сообщения задает строку сообщения, включаемую в сообщение об ошибке, генерируемое утверждением. При отсутствии описания сообщения в конкретном утверждении неявным значением строки сообщения является строка "Assertion violation". Описание серьезности задает уровень серьезности, сопоставляемый с утверждением. При отсутствии описания серьезности в конкретном утверждении неявным значением уровня серьезности является ERROR.

Вычисление оператора утверждения состоит из вычисления логического выражения, задающего условие. Если это выражение вырабатывает значение FALSE, то имеет место *нарушение утверждения*. Если это происходит, то вычисляются выражения в описании сообщения и в описании серьезности. Заданная строка сообщения и уровень серьезности (или соответствующие неявные значения) затем используются для формирования сообщения об ошибке.

Сообщение об ошибке состоит по крайней мере из:

1) указания, что данное сообщение сгенерировано утверждением;
2) значения уровня серьезности;
3) значения строки сообщения;
4) имени модуля проекта (см. раздел 11.1), содержащего это утверждение.

8.3 Оператор назначения сигнала

Оператор назначения сигнала (signal assignment statement) изменяет планируемые выходные формы сигнала, содержащиеся в драйверах одного или более сигналов (см. 9.2.1).

```
signal_assignment_statement :: =
  target <= [ transport ] waveform;

target :: =
  name
  | aggregate

waveform :: =
  waveform_element { , waveform_element }
```

Если целевой объект (target) оператора назначения сигнала представлен именем, то это имя должно обозначать сигнал, а базовым типом компонента значения каждой транзакции, производимой элементом формы сигнала (waveform element), стоящим в правой части оператора, должен быть базовый тип сигнала, обозначаемого этим именем. Такая форма оператора используется для присваивания значений, стоящих в правой части оператора, драйверам, сопоставленным с отдельным (скалярным или составным) сигналом.

Если целевой объект оператора назначения сигнала представлен в виде агрегата, то тип этого агрегата должен определяться из контекста, исключая сам агрегат, но используя тот факт, что типом этого агрегата должен быть составной тип. Базовым типом компонента значения каждой транзакции, производимой элементом формы сигнала, стоящим в правой части агрегата, должен быть базовый тип этого агрегата. Более того, выражение в каждом элементе сопоставления этого агрегата должно быть локально статическим именем, обозначающим сигнал. Такая форма оператора используется для присваивания подэлементов значений, стоящих в правой части оператора, драйверам, сопоставленным с сигналом, имя которого используется как соответствующий подэлемент этого агрегата.

Если целевой объект оператора назначения сигнала представлен в виде агрегата и выражение элемента сопоставления этого агрегата является именем сигнала, обозначающим конкретный сигнал, то этот сигнал и, следовательно, каждый подэлемент (если они есть) считаются *идентифицированными* этим сопоставлением элемента как объекты назначения этого оператора. Считается ошибкой, если конкретный сигнал или каждый подэлемент идентифицируется как объект назначения более чем одним сопоставлением элемента в таком агрегате. Более того, считается ошибкой, если элемент сопоставления в таком агрегате содержит выбор **others** или выбор в виде дискретного диапазона.

Правая часть оператора назначения сигнала может содержать зарезервированное слово **transport**. Это означает, что задержка, сопоставляемая с первым элементом формы сигнала, трактуется как *транспортная* задержка. Транспортная задержка является характеристикой устройств (таких, как линии передачи), которые имеют бесконечно большую частоту срабатывания: любой импульс передается независимо от того, насколько мала его длительность. Если слово **transport** не задано, то задержка трактуется как

инерционная задержка. Инерционная задержка является характеристикой переключательных схем: импульс, длительность которого меньше чем время переключения схемы, не будет передан.

8.3.1 *Изменение планируемой выходной формы сигнала*

Эффект выполнения оператора назначения сигнала определяется в терминах его влияния на планируемую выходную форму сигнала (см. 9.2.1), представляющую текущее и будущие значения драйверов сигналов.

```
waveform_element :: =
  value_expression [ after time_expression ]
  | null [ after time_expression ]
```

Будущее поведение драйвера(ов) для объекта назначения определяется транзакциями, производимыми вычислением элементов формы сигнала оператора назначения сигнала. Первый вид элемента формы сигнала используется для задания конкретного значения, которое будет назначено драйвером целевому объекту в указанное время. Вторая форма элемента формы сигнала используется для задания времени отключения драйвера сигнала, что приводит (по крайней мере временно) к прекращению присваивания значения целевому объекту. Такая форма элемента формы сигнала называется *пустым элементом формы сигнала.* Считается ошибкой, если целевым объектом оператора назначения сигнала, содержащего пустой элемент формы сигнала, не является защищенный сигнал.

Базовым типом выражения времени (time expression) в каждом элементе формы сигнала должен быть предопределенный тип TIME, определенный в пакете STANDARD. Если описание задержки не задано в элементе формы сигнала, то подразумевается неявное описание after 0ns. Считается ошибкой, если выражение времени в элементе формы сигнала вырабатывает отрицательное значение.

Вычисление элемента формы сигнала производит отдельную транзакцию. Компонент времени этой транзакции определяется суммированием значения текущего времени и значения выражения времени, заданного в элементе формы сигнала. Для первого вида элемента формы сигнала компонент значения этой транзакции определяется выражением значения, заданным в этом элементе. Для второго вида элемента формы сигнала компонент значения языком не определяется, но типом его должен быть тип целевого объекта. Транзакция, производимая вторым видом элемента формы сигнала, называется *пустой транзакцией.*

При вычислении оператора назначения сигнала, в котором целевой объект имеет скалярный тип, сначала вычисляется форма сигнала (waveform), стоящая справа. Вычисление формы сигнала состоит из вычисления каждого элемента, входящего в эту форму. Таким образом результатом вычисления формы сигнала является последовательность транзакций, каждая из которых относится к одному элементу формы сигнала, входящему в эту форму сигнала. Такие транзакции называются *новыми транзакциями.* Считается ошибкой, если последовательность новых транзакций не упорядочена по возрастанию в отношении времени.

Эта последовательность транзакций затем используется для изменения планируемой выходной формы сигнала, представляющей текущее и будущие значения драйвера, сопоставленного с упомянутым оператором назначения сигнала. Изменение планируемой выходной формы сигнала состоит из уничтожения 0 или более ранее вычисленных транзакций (называемых *старыми* транзакциями) из планируемой выходной формы сигнала и добавления новых транзакций, при этом изменение подчиняется следующим правилам:

1) Все старые транзакции, запланированные на время, равное или более позднее, чем время планирования самой ранней новой транзакции, выбрасываются из планируемой выходной формы.

2) Новые транзакции затем добавляются в конец планируемой выходной формы сигнала в порядке их планирования.

Если в соответствующем операторе назначения сигнала не задано зарезервированное слово transport, то начальная задержка рассматривается как инерционная задержка, и в этом случае происходит дальнейшее изменение планируемой выходной формы сигнала:

1) Все новые транзакции помечаются.

2) Старая транзакция помечается, если она непосредственно предшествует помеченной транзакции, и ее компонент значения такой же, как у этой помеченной транзакции.

3) Транзакция, определяющая текущее значение драйвера, помечается.

4) Все непомеченные транзакции (все из которых являются старыми транзакциями) выбрасываются из планируемой выходной формы сигнала.

С точки зрения помеченных транзакций считается, что любые две пустые транзакции, следующие одна за другой в планируемой выходной форме сигнала, имеют один и тот же компонент времени.

Выполнение оператора назначения сигнала, в котором целевой объект имеет составной тип, происходит аналогичным образом, за исключением того, что вычисление формы сигнала состоит из вычисления отдельной последовательности транзакций для каждого скалярного подэлемента, тип которого определяется

типом целевого объекта. Каждая такая последовательность состоит из транзакций, компоненты значения которых определяются значениями соответствующих скалярных подэлементов выражений значения в этой форме сигнала, а компонент времени определяется выражением времени, соответствующим выражению значения. Каждая такая последовательность затем используется для изменения планируемой выходной формы драйвера, соответствующего подэлементу целевого объекта. Это верно как для целевого объекта, заданного именем сигнала составного типа, так и для целевого объекта, заданного в виде агрегата.

Если конкретная процедура описана элементом объявления, который не содержится в операторе процесса, и в этой процедуре используется оператор назначения сигнала, то объектом назначения этого оператора должен быть формальный пример этой процедуры или родителя этой процедуры, или агрегат, составленный из таких формальных параметров.

Примечание — Если выражение значения (value expression), стоящее справа, является либо числовым литералом, либо атрибутом, вырабатывающим результат типа *универсальный_целый* или *универсальный_действительный*, то в этом случае выполняется неявное преобразование типа.

Вышеописанные правила гарантируют, что драйвер, на который воздействует оператор назначения сигнала, всегда является статически определяемым, если этот оператор стоит внутри конкретного процесса (включая случай, когда он стоит внутри процедуры, объявленной внутри этого процесса). В этом случае сам драйвер является драйвером, определенным этим процессом; в противном случае оператор назначения сигнала должен стоять внутри процедуры, а сам драйвер является драйвером, переданным в эту процедуру вместе с параметрами-сигналами этой процедуры.

8.4 Оператор присваивания переменной

Оператор присваивания переменной (variable assignment statement) изменяет текущее значение переменной на новое значение, заданное выражением. Переменная, обозначаемая именем, и выражение, стоящее в правой части оператора, должны быть одного и того же типа.

```
variable_assignment_statement :: =
  target : = expression;
```

Если целевой объект оператора присваивания представлен именем, то это имя должно обозначать переменную, а базовым типом выражения, стоящего в правой части, должен быть базовый тип переменной, обозначаемой этим именем. Такая форма оператора присваивания используется для присваивания значения, стоящего справа, отдельной (скалярной или составной) переменной.

Если целевой объект оператора присваивания представлен в виде агрегата, то тип этого агрегата должен определяться из контекста, исключая сам агрегат, но учитывая тот факт, что типом этого агрегата должен быть составной тип. Базовым типом выражения в правой части оператора присваивания должен быть базовый тип этого агрегата. Более того, выражение в каждом элементе этого агрегата должно быть статическим именем, обозначающим переменную. Такая форма оператора присваивания используется для присваивания каждого элемента значения, стоящего справа, переменной, имя которой используется как соответствующий подэлемент этого агрегата.

Если целевой объект оператора присваивания представлен в виде агрегата и локально статическое имя в элементе сопоставления этого агрегата обозначает конкретную переменную или другую переменную, подэлементом которой является эта переменная, то считается, что этот элемент сопоставления *идентифицирует* такую переменную как целевой объект оператора присваивания. Считается ошибкой, если эта переменная идентифицируется как объект присваивания более чем одним элементом сопоставления в одном и том же агрегате.

При выполнении оператора присваивания, целевой объект которого представлен именем переменной, сначала вычисляются имя переменной и выражение. Затем делается проверка на соответствие типа значения выражения подтипу переменной, за исключением того случая, когда переменная является массивом (в этом случае присваивание вызывает преобразование подтипа). Наконец, значение этого выражения становится новым значением этой переменной.

Выполнение оператора присваивания, целевой объект которого представлен в виде агрегата, происходит аналогичным образом, за исключением того, что вычисляется каждое из имен в агрегате и проверка подтипа выполняется для каждого подэлемента значения, стоящего справа, соответствующего одному из имен в этом агрегате. Значение этого подэлемента затем становится новым значением переменной, обозначаемой соответствующим именем.

Возникает ошибка, если вышеупомянутая проверка подтипа не имеет успеха.

Выявление типа целевого объекта оператора присваивания может потребовать выявление типа выражения, если целевой объект представлен именем, которое может быть интерпретировано как имя переменной, указываемой ссылочным значением, возвращаемым через вызов функции; и, аналогично, как элемент или сечение такой переменной.

Примечание — Если правая часть оператора присваивания представлена числовым литералом либо атрибутом, вырабатывающим результат типа *универсальный_целый* или *универсальный_действительный*, то в этом случае выполняется неявное преобразование типа.

Не допускается присваивание значения переменной файлового типа.

8.4.1 *Присваивание индексируемой переменной*

Если левая часть оператора присваивания представлена в виде имени, обозначающего индексируемую переменную (включая сечение), то значение, присваиваемое целевому объекту, неявно конвертируется в подтип этой индексируемой переменной; результат такого конвертирования подтипа становится новым значением этой индексируемой переменной.

Вышеуказанное означает, что новое значение каждого элемента индексируемой переменной задается соответствующим элементом (см. 7.2.2) индексируемого значения, получаемого вычислением выражения. В процессе конвертирования подтипа делается проверка на наличие для каждого элемента индексируемой переменной соответствующего элемента в этом индексируемом значении. Возникает ошибка, если эта проверка не имеет успеха.

Примечание — Неявное преобразование подтипа, описанное выше, для присваивания индексируемой переменной выполняется только для значения, стоящего в правой части и взятого как единое целое; конвертирование не выполняется для подэлементов, являющихся индексируемыми значениями.

8.5 Оператор вызова процедуры

Вызов процедуры активизирует выполнение тела процедуры.

```
procedure_call_statement :: =
  procedure_name [(actual_parameter_part)];
```

Имя процедуры задает активизируемое тело процедуры. Раздел фактических параметров, если задан, задает сопоставление фактических параметров с формальными параметрами процедуры.

Для каждого формального параметра процедуры вызов процедуры должен содержать точно один соответствующий фактический параметр. Этот фактический параметр задается либо явно элементом сопоставления в списке сопоставлений либо, при отсутствии такого элемента сопоставления, неявным выражением (см. 4.3.3).

Выполнение вызова процедуры включает вычисление выражений фактических параметров, заданных в этом вызове, и вычисление неявных выражений, сопоставленных с формальными параметрами этой процедуры, не имеющими сопоставленных с ними фактических параметров. (Если формальный параметр имеет неограниченный индексируемый тип, то этот параметр принимает подтип фактического параметра). Тело процедуры выполняется с использованием в качестве значений формальных параметров значений фактических параметров и значений неявных выражений.

8.6 Условный оператор

Условный оператор (if_statement) выбирает для выполнения одну или ни одной из заключенных в него последовательностей операторов (sequence of statement), в зависимости от значения одного или более соответствующих условий.

```
if_statement :: =
  if condition then
    sequence_of_statements
  { elsif condition then
    sequence_of_statements }

  [ else
    sequence_of_statements ]
  end if;
```

Выражение, задающее условие, должно иметь тип BOOLEAN.

Выполнение условного оператора состоит в последовательном вычислении условий, стоящих после слова if и после всех слов elsif (при этом заключительное else рассматривается как elsif TRUE then) до тех пор, пока одно из них не возвратит значение TRUE или все условия не возвратят значение FALSE. Если одно условие возвращает значение TRUE, то соответствующая ему последовательность операторов выполняется; в противном случае ни одна последовательность операторов не выполняется.

8.7 Оператор выбора

Оператор выбора (case statement) выбирает для выполнения одну последовательность операторов из числа альтернативных последовательностей операторов; выбираемая альтернатива определяется значением выражения.

```
case_statement :: =
  case expression is
```

```
    case_statement_alternative
    { case_statement_alternative }
  end case;

case_statement_alternative :: =
  when choices = >
    sequence_of_statements
```

Выражение должно иметь дискретный тип или одномерный символьный индексируемый тип (значения которого могут быть представлены строковыми или битово-строковыми литералами). Этот тип должен определяться независимо от контекста, в котором это выражение используется, но используя тот факт, что выражение должно иметь дискретный тип или одномерный символьный индексируемый тип. Каждый выбор в альтернативе оператора выбора (case statement alternative) должен иметь тот же тип, что и это выражение; список выборов задает значения выражения, при которых эта альтернатива будет выбрана.

Если в качестве выражения используется имя объекта, подтип которого является локально статическим (независимо от того, является ли он скалярным или индексируемым), то каждое значение этого подтипа должно быть представлено раз и только раз в множестве выборов оператора выбора, и никакие другие значения не допускаются; это правило также имеет место, если это выражение является квалифицированным выражением или преобразованием типа, в которых обозначение типа указывает на локально статический подтип.

Если типом выражения является одномерный символьный индексируемый тип, то это выражение должно быть именем объекта, подтип которого является локально статическим, или оно должно быть квалифицированным выражением или преобразованием подтипа, обозначение подтипа которых указывает локально статический подтип. В этом случае каждый выбор, стоящий в любой альтернативе оператора выбора, должен быть локально статическим выражением, значение которого имеет ту же длину, что и значение этого выражения.

Для других форм выражения каждое значение (базового) типа выражения должно быть представлено один и только один раз во множестве выборов и никакое другое значение не допускается.

Простое выражение и дискретные диапазоны, заданные в качестве выборов в операторе выбора, должны быть локально статическими. Выбор, определяемый дискретным диапазоном, символизирует все значения в соответствующем диапазоне. Выбор others допускается только в последней альтернативе, при этом он должен быть единственным выбором в этой альтернативе; этот выбор символизирует все значения (возможно и ни одного), которые не заданы в выборах предыдущих альтернатив. Простое имя элемента не допускается в качестве выбора в альтернативе оператора выбора.

Примечание — При выполнении оператора выбора для исполнения выбирается одна и только одна альтернатива, так как выборы являются исчерпывающими и взаимоисключающими. Квалификация выражения оператора выбора локально статическим подтипом часто бывает полезна для ограничения числа выборов, которые необходимо задавать явно.

Выбор others в операторе выбора требуется в случае, если типом выражения является тип *универсальный_целый* (например, если это выражение является целым литералом), так как это является единственным способом покрыть все значения этого типа.

Совмещение оператора "=" не влияет на семантику выполнения оператора выбора.

8.8 Оператор цикла

Оператор цикла (loop statement) содержит последовательность операторов, которая будет выполняться многократно 0 или более раз.

```
loop_statement :: =
    [ loop_label : ]
        [ iteration_scheme ] loop
            sequence_of_statements
    end loop [ loop_label ] ;

iteration_scheme :: =
  while condition
  | for loop_parameter_specification

parameter_specification :: =
  identifier in discrete_range
```

Если в конце оператора цикла используется метка, то она должна повторять метку в начале этого оператора.

Оператор цикла без схемы итерации (iteration scheme) задает повторяющееся выполнение последовательности операторов. Выполнение оператора цикла завершается при выходе из него вследствие выполнения оператора перехода, оператора выхода или оператора возврата.

Для оператора цикла со схемой итерации **while** перед каждым выполнением последовательности операторов вычисляется условие; если значением этого условия является TRUE, то последовательность операторов выполняется, если FALSE, то выполнение оператора цикла завершается.

Для оператора цикла со схемой итерации **for** спецификация параметра цикла (loop parameter specification) является объявлением *параметра цикла* с конкретным идентификатором. Параметр цикла является объектом, типом которого является базовый тип дискретного диапазона.

Внутри оператора цикла параметр цикла является константой. Следовательно, параметр цикла не может использоваться в качестве целевого объекта оператора присваивания. Аналогично, параметр цикла не может быть передан в качестве фактического параметра, соответствующего формальному параметру вида **out** или **inout** в списке сопоставления.

При выполнении оператора цикла со схемой итерации **for** сначала вычисляется дискретный диапазон (discrete range). Если этот диапазон является пустым диапазоном, то выполнение оператора цикла завершается; в противном случае последовательность оператора выполняется для каждого значения этого дискретного диапазона (при условии, что не был сделан выход из оператора цикла вследствие выполнения операторов перехода, выхода или возврата). Перед каждой такой итерацией параметру цикла присваивается соответствующее значение дискретного диапазона. Значения присваиваются в порядке слева направо.

8.9 Оператор перехода

Оператор перехода (next statement) используется для завершения выполнения одной из итераций объемлющего оператора цикла (далее — цикл). Завершение называется условным, если этот оператор содержит условие:

```
next_statement :: =
    next [ loop_label ] [ when condition ];
```

Оператор перехода, содержащий метку цикла, разрешен только внутри помеченного цикла и применяется именно к этому циклу; оператор перехода без метки цикла разрешен только внутри цикла и применяется к самому внутреннему объемлющему циклу (с меткой или без метки).

При выполнении оператора перехода сначала вычисляется условие, если он задано. Текущая итерация цикла завершается, если значением этого выражения является TRUE или условие не задано вообще.

8.10 Оператор выхода

Оператор выхода (exit statement) используется для завершения выполнения объемлющего оператора цикла (далее — цикл). Завершение является условным, если этот оператор содержит условие.

```
exit_statement :: =
  exit [ loop_label ] [ when condition ];
```

Оператор выхода, содержащий метку цикла, разрешен только внутри помеченного цикла и применяется именно к этому циклу; оператор выхода без метки цикла разрешен только внутри цикла и применяется к самому внутреннему объемлющему циклу (с меткой или без метки).

При выполнении оператора выхода сначала вычисляется условие, если оно задано. Выход из цикла затем имеет место, если значением этого условия является TRUE или условие не задано вообще.

8.11 Оператор возврата

Оператор возврата (return statement) используется для завершения выполнения самого внутреннего объемлющего тела функции или процедуры.

```
return_statement :: =
  return [ expression ];
```

Оператор возврата разрешен только внутри тела функции или процедуры и применяется к самой внутренней объемлющей функции или процедуре.

Оператор возврата, стоящий в теле процедуры, не должен содержать выражение. Оператор возврата, стоящий в теле функции, должен содержать выражение.

Значение этого выражения определяет результат, возвращаемый этой функцией. Типом этого выражения должен быть базовый тип обозначения типа, заданного после зарезервированного слова **return** в спецификации этой функции. Считается ошибкой, если выполнение функции завершается средством, отличным от оператора возврата.

При выполнении оператора возврата сначала вычисляется выражение (если оно задано) и делается проверка на принадлежность значения выражения подтипу результата. Если проверка имеет успех, то

на этом выполнение оператора возврата завершается; также заканчивается выполнение объемлющей подпрограммы. Возникает ошибка на месте оператора возврата, если проверка не имеет успеха.

Примечание — Если выражение является числовым литералом либо атрибутом, производящим результат типа *универсальный_целый* или *универсальный_действительный*, то выполняется неявное преобразование результата.

8.12 Пустой оператор

Пустой оператор (null statement) не вызывает никаких действий.

```
null_statement :: = null;
```

Выполнение пустого оператора не имеет никакого эффекта, кроме передачи управления на следующий оператор.

Примечание — Пустой оператор может быть использован для явного указания того, что никакие действия не выполняются при удовлетворении некоторых условий. Это, в частности, полезно при использовании оператора выбора, в котором все возможные значения выражения должны быть покрыты выборами; для определенных же выборов никаких действий не требуется.

9 ПАРАЛЛЕЛЬНЫЕ ОПЕРАТОРЫ

В данном разделе описаны различные формы параллельных операторов (concurrent statement). Параллельные операторы используются для определения взаимосвязанных блоков и процессов, которые совместно описывают полное поведение или структуру проекта. Параллельные операторы выполняются асинхронно в отношении друг друга.

```
concurrent_statement :: = - - параллельный оператор
    block_statement                    - - оператор блока
    | process_statement                - - оператор процесса
    | concurrent_procedure_call        - - параллельный вызов процедуры
    | concurrent_assretion_statement           - - параллельный оператор утверждения
    | concurrent_signal_assignment_statement   - - параллельный оператор назначения сигнала
    | component_instantiation_statement        - - оператор конкретизации компонента
    | generate_statement                       - - оператор генерации
```

Первичными параллельными операторами являются оператор блока (block statement), который группирует вместе другие параллельные операторы, и оператор процесса (process statement), который представляет отдельный независимый последовательный процесс. Остальные параллельные операторы обеспечивают удобную форму представления простых, общеиспользуемых форм процессов, а также структурной декомпозиции и регулярных описаний.

Внутри конкретного цикла моделирования реализация может обеспечивать выполнение параллельных операторов в параллельном или каком-либо другом порядке. Язык не определяет этот порядок. Описание, которое зависит от конкретного порядка выполнения параллельных операторов, является ошибочным.

Все параллельные операторы могут быть помечены. Метки этих операторов неявно объявляются в начале раздела объявлений самого внутреннего объемлющего объявления объекта, архитектурного тела или оператора блока.

9.1 Оператор блока

Оператор блока определяет внутренний блок, представляющий часть проекта. Блоки могут быть иерархически вложенными для обеспечения декомпозиции проекта.

```
block_statement :: =
    block_label:
    block [ ( guard_expression ) ]
          block_header
          block_declarative_part
        begin
          block_statement_part
        end block [ block_label ];

block_header :: =
    [ generic_clause
    [ generic_map_aspect; ] ]
    [ port_clause
    [ port_map_aspect; ] ]
```

```
block_declarative_part :: =
    { block_declarative_item }

block_statement_part :: -
    { concurrent_statement }
```

Если после зарезервированного слова block стоит выражение защиты* (guard expression), то в начале области объявлений (block declarative part) этого блока неявно описывается сигнал с простым именем GUARD предопределенного типа BOOLEAN, и это выражение определяет значение этого сигнала в любое конкретное время (см. 12.6.3). Типом выражения защиты должен быть тип BOOLEAN. Сигнал GUARD может быть использован для управления выполнением ряда операторов внутри блока (см. 9.5).

Неявный сигнал GUARD не должен иметь источник.

Если в операторе блока используется заголовок блока (block header), то он явно идентифицирует ряд значений или сигналов, которые должны быть импортированы из объемлющей среды в блок и сопоставлены формальным параметрам настройки или портам. Описание параметров настройки и описание портов определяют формальные параметры настройки и формальные порты блока (см. 1.1.1.1 и 1.1.1.2); аспекты отображения параметров настройки и портов определяют сопоставление фактических параметров настройки и портов формальным параметрам настройки и портам (см. 5.2.1.2). Такие фактические параметры настройки и порты вычисляются в контексте объемлющего раздела объявлений.

Если в конце оператора блока используется метка (block label), то она должна повторять метку этого блока.

Примечание — Значение сигнала GUARD всегда определяется в рамках области действия конкретного блока и не распространяется неявно на объекты проекта, связанные с компонентами, конкретизируемыми внутри этого блока. Сигнал GUARD может быть явно передан как фактический сигнал в конкретизацию компонента с целью распространения его значений на компоненты более низкого уровня.

Фактические параметры, стоящие в списке сопоставления портов конкретного блока, никогда не могут обозначать формальные порты этого блока.

9.2 Оператор процесса

Оператор процесса определяет независимый последовательный процесс, представляющий поведение некоторой части проекта.

```
process_statement :: =
    [ process_label: ]
      process [ (sensitivity_list ) ]
          process_declarative_part
      begin
          process_statement_part
      end process [ process_label ] ;

process_declarative_part :: =
{ process_declarative_item }

process_declarative_item :: =
    subprogram_declaration
    | subprogram_body
    | type_declaration
    | subtype_declaration
    | constant_declaration
    | variable_declaration
    | file_declaration
    | alias_declaration
    | attribute_declaration
    | attribute_specification
    | use_clause

process_statement_part :: =
    { suquential_statement }
```

Если после зарезервированного слова process стоит список чувствительности (sensitivity list), то подразумевается, что последним оператором в разделе операторов (process statement part) этого процесса является неявный оператор ожидания; этот оператор имеет форму

wait on sensitivity_list;

в которой список чувствительности — это тот же самый список, стоящий после слова process. Такой оператор процесса не может содержать явного оператора ожидания. Аналогично, если такой процесс является родителем процедуры, то эта процедура не может содержать оператора ожидания.

В списке чувствительности оператора процесса могут использоваться только статические имена сигналов (см. 6.1).

Если в конце оператора процесса стоит метка (process label), то она должна повторять метку этого оператора.

Выполнение оператора процесса состоит из повторяющегося выполнения последовательности операторов. После того, как последний оператор в этой последовательности будет выполнен, выполнение оператора процесса продолжается, начиная с первого оператора в этой последовательности.

Оператор процесса называется *пассивным процессом*, если ни сам процесс, ни любая процедура, для которой этот процесс является родительским, не содержат оператор назначения сигнала. Такой процесс или любой параллельный оператор, эквивалентный такому процессу, может использоваться в разделе операторов объявления объекта.

Примечание — Из вышеуказанных правил следует, что процесс, имеющий явный список чувствительности, всегда имеет в себе ровно один (неявный) оператор ожидания, и этот оператор стоит в конце последовательности операторов в разделе операторов этого процесса. Таким образом, процесс со списком чувствительности всегда находится в ожидании в конце своего раздела операторов; любое событие на сигналах, перечисленных в этом списке, влечет выполнение такого процесса с начала раздела операторов до конца, где процесс будет опять находиться в ожидании. Указанный процесс выполняется однажды весь в начале моделирования и приостанавливается при выполнении неявного оператора ожидания.

9.2.1 *Драйверы*

Каждый оператор назначения в операторе процесса определяет множество *драйверов* для ряда скалярных сигналов. Для конкретного скалярного сигнала S в операторе процесса существует отдельный драйвер при условии, что в этом операторе процесса имеется по крайней мере один оператор назначения сигнала и самый длинный статический префикс сигнала, стоящего в левой части этого оператора назначения, обозначает S или составной сигнал, в котором S является подэлементом. Считается, что каждый такой оператор назначения сигнала *сопоставляется* с этим драйвером. Выполнение оператора назначения влияет только на сопоставленный ему драйвер (или драйверы).

Драйвер скалярного сигнала представлен *планируемой выходной формой сигнала.* Планируемая выходная форма сигнала состоит из последовательности, состоящей из одной или более транзакций, каждая из которых состоит из двух частей: компонента значения и компонента времени. Для конкретной транзакции компонент значения представляет значение, которое драйвер сигнала принимает в некоторый момент времени, а компонент времени задает этот момент времени. Все транзакции упорядочены в отношении компонентов времени.

Драйвер всегда содержит по крайней мере одну транзакцию. Начальное содержимое драйвера, сопоставленного конкретному сигналу, определяется неявным значением, сопоставляемым этому сигналу (см. 4.3.1.2).

Для любого драйвера существует ровно одна транзакция, компонент времени которой не больше, чем текущее время моделирования. Компонент значения этой транзакции является *текущим значением* драйвера. Если в результате продвижения времени текущее время становится равным компоненту времени следующей транзакции, то первая транзакция выбрасывается из планируемой выходной формы сигнала, а следующая становится текущим значением драйвера.

9.3 Параллельный вызов процедуры

Параллельный вызов процедуры (concurrent procedure call) представляет процесс, содержащий соответствующий последовательный вызов процедуры.

```
concurrent_procedure_call :: =
    [ label : ] procedure_call_statement
```

Для любого параллельного вызова процедуры существует эквивалентный оператор процесса. Этот оператор не имеет списка чувствительности и раздела объявлений, а раздел операторов этого процесса состоит из оператора вызова процедуры (procedure call statement), за которым следует оператор ожидания. Оператор вызова процедуры состоит из того же имени процедуры и из того же раздела фактических параметров, что и параллельный вызов процедуры. Каждый формальный параметр процедуры, активизируемой параллельным вызовом процедуры, должен быть класса сигнал или константа.

Если в фактической части любого элемента сопоставления в параллельном вызове процедуры есть первичное, обозначающее сигнал, и этот сигнал сопоставляется формальному параметру вида in или inout, то эквивалентный оператор процесса включает в себя конечный оператор ожидания с описанием чувствительности, содержащим самый длинный статический префикс каждого имени сигнала, использу-

емого как первичное в фактической части и сопоставляемого такому формальному параметру; в противном случае эквивалентный оператор процесса содержит конечный оператор ожидания, не имеющий ни явного описания чувствительности, ни явного описания условия, ни явного описания времени блокировки.

Выполнение параллельного вызова процедуры эквивалентно выполнению эквивалентного оператора процесса.

Пример

```
    - - параллельный вызов процедуры
CheckTiming (tPLH, tPHL, Clk, D, Q);

    - - эквивалентный процесс
process
begin
    CheckTiming (tPLH, tPHL, Clk, D, Q);
    wait on Clk, D, Q;
end process;
```

Примечание — Параллельные вызовы процедуры представляют возможность объявлять процедуры, выражающие общеиспользуемые процессы, и довольно просто создавать такие процессы обычным вызовом процедуры в виде параллельного оператора. Оператор ожидания в конце раздела операторов эквивалентного оператора процесса позволяет вызывать процедуру без ее бесконечного зацикливания, даже если эта процедура необязательно предназначена для использования в качестве процесса (то есть она не содержит оператора ожидания). Такая процедура может сохранять свое существование с течением времени (и таким образом значения ее переменных могут удерживать свое состояние с течением времени), если самый внешний оператор этой процедуры является оператором цикла, и этот цикл содержит оператор ожидания.

Аналогично можно гарантировать, что такая процедура выполняется один раз в начале моделирования, если ее последний оператор является оператором ожидания, не имеющим ни описания чувствительности, ни описания условия, ни описания времени блокировки.

Значение неявного объявляемого сигнала GUARD не имеет влияния на вычисление параллельного вызова процедуры, если он не используется явно в одной из фактических частей раздела, фактических параметров этого параллельного вызова процедуры.

9.4 Параллельный оператор утверждения

Параллельный оператор утверждения (concurrent assertion statement) представляет пассивный оператор процесса, содержащий заданный оператор утверждения.

```
concurrent_assertion_statement :: =
  [ label: ] assertion-statement
```

Для любого параллельного оператора утверждения существует эквивалентный оператор процесса. Этот эквивалентный процесс не имеет ни списка чувствительности, ни раздела объявлений. Раздел операторов этого процесса содержит оператор утверждения (assertion statement), за которым следует оператор ожидания. Оператор утверждения содержит то же условие, описание сообщения и описания серьезности, что и параллельный оператор утверждения.

Если в логическом выражении, определяющем условие утверждения, есть первичное, обозначающее сигнал, то эквивалентный оператор процесса включает в себя конечный оператор ожидания с описанием чувствительности, содержащим самый длинный статический префикс каждого имени сигнала, используемого как первичное в этом выражении; в противном случае эквивалентный оператор процесса содержит конечный оператор ожидания, не имеющий ни явного описания чувствительности, ни явного описания условия, ни явного описания времени блокировки.

Выполнение параллельного оператора утверждения эквивалентно выполнению эквивалентного оператора процесса.

Примечание — Параллельные вызовы процедуры представляют возможность объявлять процедуры, выражающие общеиспользуемые процессы, и довольно просто создавать такие процессы обычным вызовом процедуры в виде параллельного оператора. Оператор ожидания в конце раздела операторов эквивалентного оператора процесса позволяет вызывать процедуру без ее бесконечного зацикливания, даже если эта процедура необязательно предназначена для использования в качестве процесса (то есть она не содержит оператора ожидания). Такая процедура может сохранять свое существование с течением времени (и таким образом значения ее переменных могут удерживать свое состояние с течением времени), если самый внешний оператор этой процедуры является оператором цикла, и этот цикл содержит оператор ожидания. Аналогично можно гарантировать, что такая процедура выполняется один раз в начале моделирования, если ее последний оператор является оператором ожидания, не имеющим ни описания чувствительности, ни описания условия, ни описания времени блокировки.

Значение неявного объявленного сигнала GUARD не имеет влияния на вычисление параллельного вызова процедуры, если он не используется явно в одной из фактических частей раздела, фактических параметров этого параллельного вызова процедуры.

9.5 Параллельный оператор назначения сигнала

Параллельный оператор назначения сигнала (concurrent_signal_assignment) представляет эквивалентный оператор процесса, который присваивает значения сигналам.

concurrent_signal_assignment_statement :: =
[label:] conditional_signal_assignment
| [label:] selected_signal_assignment

options :: = [guarded] [transport]

Существуют две формы параллельного оператора назначения сигнала. Отличительные характеристики каждой из этих форм описаны ниже.

Каждая форма может содержать одну или обе из двух опций guarded и transport. Опция guarded указывает на то, что оператор назначения сигнала выполняется, когда сигнал GUARD изменяет свое значение с FALSE на TRUE, или когда этот сигнал имеет значение TRUE и на одном из его входов происходит событие. (Таким сигналом может быть один из неявно объявленных сигналов GUARD, сопоставленных с операторами блока, имеющими выражением защиты; или им может быть явно объявленный сигнал типа BOOLEAN, видимый в точке параллельного оператора назначения сигнала). Опция transport указывает на то, что оператор назначения сигнала имеет транспортную задержку.

Если целевой объект параллельного назначения сигнала является именем, которое обозначает защитный сигнал (см. 4.3.1.2), или он представлен в форме агрегата, в котором выражение в каждом элементе сопоставления является статическим именем сигнала, обозначающим защищенный сигнал, то такой целевой объект называется защищенным целевым объектом. Если целевой объект параллельного назначения сигнала является именем, которое обозначает сигнал, не являющийся защищенным сигналом, или он представлен в виде агрегата, в котором выражение в каждом элементе сопоставления является статическим именем сигнала, не являющегося защитным сигналом, то такой целевой объект называется незащищенным целевым объектом. Считается ошибкой, если целевой объект параллельного назначения сигнала не является ни тем и ни другим.

Для любого параллельного оператора назначения сигнала существует эквивалентный оператор процесса, имеющий тот же смысл. Оператор процесса, эквивалентный параллельному оператору назначения сигнала, целевой объект которого представлен именем сигнала, строится следующим образом:

1) Если в параллельном операторе назначения сигнала задана метка, то такая же метка используется в операторе процесса.

2) Если в параллельном операторе назначения сигнала задана опция transport, то зарезервированное слово transport используется в каждом операторе назначения сигнала в операторе процесса.

3) Раздел операторов эквивалентного оператора процесса содержит преобразование сигнала, которое имеет вид: либо последовательного оператора назначения сигнала, либо условного оператора или оператора выбора, содержащего последовательные операторы назначения сигнала — один для каждой из альтернативных форм сигнала. Это преобразование сигнала определяет, какая из альтернативных форм сигнала должна быть присвоена выходным сигналам. При этом раздел операторов может содержать последовательность операторов отключения, которые предназначены для присвоения пустых транзакций целевому объекту параллельного назначения сигнала при выполнении определенных условий.

Если в параллельном операторе назначения сигнала используется опция guarded, то такое параллельное назначение сигнала называется защищенным назначением.

Если параллельный оператор назначения сигнала представляет собой защищенное назначение, и целевой объект параллельного назначения сигнала является защищенным целевым объектом, то раздел операторов эквивалентного оператора процесса имеет вид:

```
if GUARD then
    преобразование сигнала
else
    операторы отключения
end if;
```

В противном случае, если параллельный оператор назначения сигнала является защищенным назначением, но целевой объект параллельного назначения сигнала не является защищенным целевым объектом, то раздел операторов эквивалентного оператора процесса имеет вид:

```
if GUARD then
    преобразование сигнала
end if;
```

Наконец, если параллельный оператор назначения сигнала не является защищенным назначением и целевой объект параллельного назначения сигнала не является защищенным целевым объектом, то раздел операторов эквивалентного оператора процесса имеет вид:

преобразование сигнала

Считается ошибкой, если параллельное назначение сигнала является защищенным назначением, а целевой объект параллельного назначения сигнала является защищенным целевым объектом.

4) Если параллельный оператор назначения сигнала является защищенным назначением или если в каком-нибудь выражении (отличном от временного выражения) внутри параллельного оператора назначения сигнала имеется первичное, обозначающее сигнал, то оператор процесса содержит в качестве последнего оператора оператор ожидания с явным описанием чувствительности. Это описание чувствительности содержит самый длинный статический префикс каждого имени сигнала (если он есть), используемого как первичное в одном из вышеупомянутых выражений. Более того, если параллельный оператор назначения сигнала является защищенным назначением, то описание чувствительности также содержит простое имя GUARD. (Сигналы, идентифицированные этими именами, называются входами оператора назначения сигнала). В противном случае оператор процесса содержит в качестве последнего оператор ожидания без явного описания чувствительности, явного описания условия или явного описания времени блокировки.

Если в эквивалентном операторе процесса имеется последовательность операторов отключения, то эта последовательность содержит одно последовательное назначение сигнала для каждого скалярного подэлемента целевого объекта параллельного оператора назначения сигнала.

Для каждого такого последовательного назначения сигнала целевой объект этого назначения является соответствующим скалярным подэлементом целевого объекта параллельного назначения сигнала, а форма сигнала этого назначения является пустым элементом формы сигнала, выражение времени которого заданно принимает спецификацию отключения (см. 5.3).

Если целевой объект параллельного оператора назначения сигнала представлен в форме агрегата, то имеет место то же самое преобразование. Такой целевой объект может содержать только локально статические имена сигналов, и никакие два имени сигналов не могут идентифицировать один и тот же объект или подэлемент.

Считается ошибкой, если пустой элемент формы сигнала стоит в форме сигнала параллельного оператора назначения сигнала.

Выполнение параллельного оператора назначения сигнала эквивалентно выполнению эквивалентного оператора процесса.

Примечание — Параллельный оператор назначения сигнала, формы сигнала и целевой объект которого содержат только статические выражения, эквивалентен оператору процесса, конечный оператор ожидания которого не имеет явного описания чувствительности, и поэтому он будет выполнен один раз целиком в начале моделирования, а затем приостановлен постоянно.

9.5.1 *Условное назначение сигнала*

Условное назначение сигнала (conditional signal assignment) представляет оператор процесса, в котором преобразование сигнала задано условным оператором.

```
conditional_signal_assignment :: =
    target <= options condititonal_waveforms;

conditional_waveforms :: =
    { waveform when condition else }
    waveform
```

Для конкретного условного назначения сигнала существует соответствующий ему эквивалентный оператор процесса, как определено для любого параллельного оператора назначения сигнала. Если условное назначение сигнала имеет форму:

```
targen <= options waveform1 when condition1 else
                  waveform2 when condition2 else
                            .
                            .
                            .
          waveformN—1 when conditionN—1 else
          waveformN ;
```

то преобразование сигнала в соответствующем операторе процесса имеет вид

```
if condition1 then
    target <= [ transport ] waveform1;
elsif condition then
    target <= [ transport ] waveform2;
```

```
        •
        •
        •
    target <= [ transport ] waveformN—1;
else
    target <= [ transport ] waveformN;
end if;
```

Если условные формы сигнала (conditional waveforms) представлены единственной формой сигнала, то преобразование сигнала в соответствующем операторе процесса имеет вид

```
target <= [ transport ] waveform;
```

Характеристики форм сигнала и условий в условном операторе назначения должны быть такими, чтобы условный оператор в эквивалентном операторе процесса являлся допустимым оператором.

9.5.2 *Выборочное назначение сигнала*

Выборочное назначение сигнала (selected signal assignment) представляет оператор процесса, в котором преобразование сигнала задано оператором выбора.

```
selected_signal_assignment :: =
  with expression select
    target <= options selected_waveforms ;

selected_waveforms :: =
  { waveform when choices , }
    waveform when choices
```

Для конкретного выборочного назначения сигнала существует соответствующий эквивалентный оператор процесса, как определено для любого параллельного оператора назначения сигнала. Если выборочное назначение сигнала имеет форму

```
with expression select
target<= options waveform1 when choice_list1,
                 waveform2 when choice_list2,
                     •
                     •
                     •
                 waveformn—1 when choice_listn—1,
                 waveformn   when choice_listn;
```

то преобразование сигнала в соответствующем операторе процесса имеет вид

```
case expression is
    when choice_list1 =>
      target <= [transport] waveform1;
    when choice_list2 =>
      target <= [transport] waveform2;
              •
              •
              •
    when choice_listN—1 =>
      target <= [transport] waveformn—1;
    when choice_listN =>

      target <= [transport] waveformn;
    end case;
```

Характеристики выражения выбора, форм сигнала и выборов в операторе выборочного назначения должны быть такими, чтобы оператор выбора в эквивалентном операторе процесса являлся допущенным оператором.

9.6 Оператор конкретизации компонента

Оператор конкретизации компонента (component instantiation statement) определяет подкомпонент объекта проекта, в котором он используется, и сопоставляет сигналы с портами этого подкомпонента.

Этот подкомпонент является одним экземпляром класса компонентов, определяемого соответствующим объявлением компонента.

component_instantiation_statement :: =
*instantiation*_label: *component*_name
[generic_map_aspect]
[port_map_aspect];

Имя компонента (component name) должно представлять имя компонента, описанное в объявлении компонента.

Аспект отображения параметров настройки (generic map aspect), если задан, сопоставляет каждому локальному параметру настройки (или его подэлементу) в соответствующем объявлении компонента отдельный фактический параметр настройки.

Для каждого локального параметра настройки (или его подэлемента) должно быть задано точно одно сопоставление. Аналогично, аспект отображения портов (port map aspect), если задан, сопоставляет каждому локальному порту (или подэлементу) в соответствующем объявлении компонента отдельный фактический параметр. Для каждого локального порта (или его подэлемента) должно быть задано точно одно сопоставление. Аспекты отображения параметров настройки и аспекты отображения портов описаны в 5.2.1.2.

Примечание — Для связывания конкретного экземпляра компонента с объектом проекта и сопоставления локальных параметров настройки и локальных портов с формальными параметрами настройки и формальными портами этого объекта может быть использована спецификация конфигурации.

Оператор конкретизации компонента может быть использован для представления структурной организации проекта. Используя объявления компонентов, сигналов и операторы конкретизации компонентов, конкретный (внутренний или внешний) блок может быть описан в терминах подкомпонентов, взаимосвязанных при помощи сигналов.

Конкретизация компонента обеспечивает средство структурирования логической декомпозиции проекта. Точные структурные или поведенческие характеристики конкретного подкомпонента могут быть описаны позже. Конкретизация компонента также обеспечивает механизм многократного использования существующих проектов, находящихся в библиотеке проекта.

Спецификация конфигурации может связывать конкретный экземпляр компонента с существующим объектом проекта, даже если параметры настройки и порты в объявлении объекта точно не совпадают с параметрами настройки и портами компонента.

9.6.1 *Конкретизация компонента*

Оператор конкретизации компонента и соответствующая спецификация конфигурации, взятые вместе, предполагают, что иерархия блоков внутри объекта проекта, содержащего конкретизацию компонента, будет расширена уникальной копией блока, определенного другим объектом проекта. Аспект отображения параметров настройки и аспект отображения портов в операторе конкретизации компонента и в связывающем указании спецификации конфигурации идентифицируют соединения, которые должны быть сделаны с целью выполнения расширения.

Оператор конкретизации компонента эквивалентен паре вложенных операторов блока, сцепляющих иерархию блоков в модуле проекта, содержащем этот оператор конкретизации, с уникальной копией иерархии блоков, содержащейся в другом объекте проекта (то есть подкомпоненте). Внешний блок в этой паре представляет объявление компонента; внутренний блок представляет объект проекта, с которым этот компонент связывается. Каждый из этих блоков определяется оператором блока.

Заголовок оператора блока, соответствующего объявлению компонента, состоит из описания параметров настройки и описания портов (если они есть), заданных в этом объявлении компонента, за которыми следует аспект отображения параметров настройки и аспект отображения портов (если они есть), заданных в соответствующем операторе конкретизации компонента. Смысл любого идентификатора, стоящего в заголовке этого оператора блока, тот же, который придается соответствующему вхождению этого идентификатора в описание параметров настройки, описание портов, аспект отображения параметров настройки или аспект отображения портов соответственно.

Раздел операторов оператора блока, соответствующего объявлению компонента, состоит из вложенного оператора блока, соответствующего объекту проекта.

Заголовок оператора блока, соответствующего объекту проекта, состоит из описаний параметров настройки и портов (если они есть), заданных в объявлении объекта, определяющем интерфейс этого объекта проекта, за которым следуют аспекты отображения параметров настройки и портов (если они есть), используемых в связывающем указании, которое связывает экземпляр компонента с этим объектом. Раздел объявлений оператора блока, соответствующего объекту проекта, состоит из объявлений, стоящих в разделе объявлений объекта, за которым следуют объявления, стоящие в соответствующем архитектурном теле. Раздел операторов оператора блока, соответствующего объекту проекта, состоит из параллельных операторов, стоящих в разделе операторов объекта, за которыми следуют параллельные операторы, стоящие в разделе операторов соответствующего архитектурного тела.

Смысл любого идентификатора, стоящего в любом месте этого оператора блока, тот же, который придается соответствующему вхождению этого идентификатора в объявление объекта или архитектурное тело соответственно.

Пример — Пусть имеются следующие описания:

```
- - объявление компонента
component COMP port (A, B: inout BIT);
- - спецификация конфигурации
for C: COMP use
    entity X(Y)
    port map (P1 => A2, P2 => B);
- - оператор конкретизации компонента
C: COMP port map (A => S1, B => S2);
- - объявление объекта и архитектуры
entity X is
    port (P1, P2: inout BIT);
    constant Delay: Time: = 1ms;
begin
    Check Timing (P1, P2, 2 * Delay);
end X;
architecture Y of X is
    signal P3 : Bit;
begin
    P3 <= P1 after Delay;
    P2 <= P3 after Delay;
    B : block
     ...
       begin
     ...
       end block;
end Y;
```

тогда следующие операторы блока реализуют сцепление иерархии блоков, в которой объявлен компонент C с иерархией блоков, заключенной в объекте проекта X(Y)

```
C: block                              - - блок компонента
    port (A, B: inout BIT);           - - локальные порты
    port map (A => S1, B => S2);      - - связывание
                                      - - фактические/локальные

begin
    X: block                            - - блок объекта проекта
    port (P1, P2 : inout BIT);          - - формальные порты
    port map (P1 => A, P2 => B);        - - связывание
                                        - - локальных портов с формальными
    constant Delay : Time : = 1ms       - - объявление из объекта
    signal P3 : Bit ;                   - - объявление из архитектуры

begin
    Check Timing (P1, P2, 2 * Delay); - - оператор из объекта
    P3 <= P1 after Delay;               - - операторы из архитектуры
    P2 <= P3 after Delay;
    B: block                            - - внутренняя иерархия блоков
      ...
    begin
      ...
    end block B;
  end block X ;
end block C ;
```

Расширения иерархии блоков, предполагаемые операторами конкретизации компонентов, связанных с объектами проекта, реализуются в процессе предвыполнения иерархии проекта (см. раздел 12).

9.7 Оператор генерации

Оператор генерации (generate statement) обеспечивает механизм итеративного или условного предвыполнения части описания.

```
generate_statement :: =
  generate_label :
    generation_scheme  generate
      {concurrent_statement}
    end generate [generate_label] ;

generate_scheme :: =
  for generate_parameter_specification
  | if condition

label :: = identifier
```

Если в конце оператора генерации стоит метка, то она должна повторить метку оператора генерации (generate label).

Для оператора генерации со схемой генерации (generation scheme for) спецификация параметра генерации (generate parameter specification) является объявлением параметра генерации с заданным идентификатором. Параметр генерации является константой, типом которой является базовый тип дискретного диапазона, заданного в спецификации параметра генерации.

Предвыполнение оператора генерации описано в 12.4.2.

Пример —

```
B: block
  begin
    L1: CELL port map (Top, Bottom, A(0), B(0));
    L2: for I in 1 to 3 generate
        L3: for J in 1 to 3 generate
          L4: if I+J >4 generate
            L5: CELL port map (A(I—1), B(J—1), A(I), B(J));
          end generate;
        end generate;
      end generate;
    L6: for I in 1 to 3 generate
      L7: for J in 1 to 3 generate
        L8: if I+J < 4 generate
          L9: CELL port map (A(I+1), B(J+1), A(I), B(J));
        end generate;
      end generate;
    end generate;

end block B;
```

10 ОБЛАСТЬ ДЕЙСТВИЯ И ВИДИМОСТЬ

В данном разделе описаны правила, определяющие области действия и правила, определяющие видимость идентификаторов в различных точках текста описания. В формулировке этих правил используется понятие области объявлений.

10.1 Область объявлений

Область объявления — это часть теста описания. Отдельная область объявления формируется (текстуально) следующими конструкциями:

1) Объявление объекта, взятое вместе с соответствующим архитектурным телом.
2) Объявление конфигурации.
3) Объявление подпрограммы, взятое вместе с соответствующим телом подпрограммы.
4) Объявление пакета, взятое вместе с соответствующим телом (если оно есть).
5) Объявление структурного типа.
6) Объявление компонента.
7) Оператор блока.
8) Оператор процесса.
9) Оператор цикла.

10) Конфигурация блока.

11) Конфигурация компонента.

В каждом из вышеуказанных случаев считается, что область объявлений *сопоставляется* с соответствующим объявлением или оператором. Считается что объявление стоит *непосредственно внутри* области объявлений, если эта область является самой внутренней областью, объемлющей это объявление, не учитывая при этом область объявлений (если она есть), сопоставленную с самим объявлением.

Некоторые области объявлений состоят из раздельных частей. Каждая область объявлений, тем не менее, рассматривается как (логически) непрерывная часть текста описания. Следовательно, если какое-либо правило определяет часть текста как текст, *распространяющийся* от некоторой специфической точки области объявлений до конца этой области, то эта часть является соответствующим подмножеством этой области объявлений (и, следовательно, она не содержит промежуточных объявлений между объявлением интерфейса и соответствующим объявлением тела).

10.2 Область действия объявлений

Для каждой формы объявления правилами языка определяется некоторая часть текста описания, называемая *областью действия объявления*. Областью действия объявления также называется область действия любого понятия, описанного этим объявлением. Более того, если объявление сопоставляет некоторое обозначение (либо идентификатор, либо символьный литерал, либо символ оператора) с описываемым понятием, то эта часть текста также называется областью действия этого обозначения. Внутри области действия понятия и только там существуют места, в которых допускается использование сопоставленного обозначения в целях ссылки на описанное понятие. Такие места определяются правилами видимости и совмещения.

Область действия объявления, стоящего непосредственно внутри области объявлений, распространяется от начала этого объявления до конца области объявлений; эта часть области действия объявлений называется *непосредственной областью действия*. Более того, для каждого из объявлений, перечисленных ниже, область действия этого объявления распространяется за пределы непосредственной области действия.

1) Объявления, стоящие непосредственно внутри объявления пакета.
2) Объявления элемента в описании структурного типа.
3) Объявление формального параметра в объявлении подпрограммы.
4) Объявление локального параметра настройки в объявлении компонента.
5) Объявление локального порта в объявлении компонента.
6) Объявление формального параметра настройки в объявлении объекта.
7) Объявление формального порта в объявлении объекта.

При отсутствии отдельного объявления подпрограммы спецификация подпрограммы, заданная в теле подпрограммы, ведет себя как это объявление, и правило (3) в этом случае также применимо. В каждом из перечисленных случаев конкретное объявление стоит непосредственно внутри другого объявляющего объявления, и область действий первого распространяется до конца области действия второго.

В дополнение к вышеописанным правилам, область действия *любого* объявления, содержащего конец раздела объявлений конкретного блока (либо внешнего блока, определяемого объектом проекта, либо внутреннего блока, определяемого оператором блока) распространяется на объявление конфигурации, конфигурирующее этот блок.

Если конфигурация компонента используется как элемент объявлений непосредственно внутри конфигурации блока, конфигурирующей конкретный блок, и область действия конкретного объявления содержит конец раздела объявлений этого блока, то область действия этого объявления распространяется от начала до конца области объявлений, сопоставленной с этой конфигурацией компонента. Аналогичные правила применяются к конфигурации блока, используемой как элемент объявления непосредственно внутри другой конфигурации блока, при условии что первая конфигурирует внутренний блок. Более того, область действия описания использования распространяется аналогичным образом. Наконец, область действия библиотечного модуля, содержащегося в библиотеке проекта, распространяется вместе с областью действия логического имени библиотеки, соответствующего этой библиотеке проекта.

Примечание — Вышеописанные правила в отношении области действия применяются ко всем формам объявления. В частности, они применяются также к неявным объявлениям.

10.3 Видимость

Смысл появления идентификатора в конкретном месте текста определяется правилами видимости, а в случае совмещенных объявлений — правилами совмещения. Под термином идентификатор в данной главе понимается любой идентификатор, отличный от зарезервированного слова. Под термином "место" в данной главе понимаются те места, в которых может стоять лексический элемент (такой, как идентификатор). Под совмещенными объявлениями в данной главе понимают совмещенные объявления подпрограмм и литералов перечисления.

Для каждого идентификатора и в каждом месте текста правилами видимости устанавливается множество объявлений (с этим идентификатором), которое определяет возможные смыслы появления этого идентификатора. Объявление считается *видимым* в конкретном месте текста, если согласно правилам видимости это объявление определяет возможный смысл этого появления. При установлении смысла такого объявления возможны два случая:

1 Правилами видимости установлен *по крайней мере один* возможный смысл. В этом случае правила видимости являются достаточными для установления объявления, определяющего смысл появления идентификатора, или при отсутствии такого объявления — для установления того факта, что это появление недопустимо в этом месте.

2 Правилами видимости установлено *более одного* возможного смысла. В этом случае появление идентификатора допустимо в этом месте, если и только если точно *одно видимое* объявление приемлемо в этом контексте для правил совмещения.

Объявление видимо только в пределах определенной части действия; эта часть начинается сразу за этим объявлением, за исключением объявления модуля проекта, когда эта часть начинается непосредственно после зарезервированного слова is, стоящего после идентификатора этого модуля.

Видимость может быть либо косвенной, либо непосредственной. Объявление видимо *косвенно* в местах, определяемых следующим правилами:

1) Для первичного модуля, содержащегося в библиотеке: на месте суффикса в составном имени, префикс которого обозначает эту библиотеку.

2) Для архитектурного тела, сопоставленного с конкретным объявлением объекта: на месте спецификации блока в конфигурации блока для внешнего блока, интерфейс которого определяется этим объявлением объекта.

3) Для объявления, заданного в объявлении пакета: на месте суффикса в составном имени, префикс которого обозначает этот пакет.

4) Для объявления элемента, стоящего внутри объявления структурного типа: на месте суффикса в составном имени, префикс которого подходит для этого типа; также на месте выбора (перед составным разделителем =>) в именованном сопоставлении элемента, стоящем в агрегате этого типа.

5) Для предопределенного атрибута, который применяется к конкретному диапазону определения: на месте обозначения атрибута (после разделителя ') в имени атрибута, префикс которого принадлежит к этому диапазону определения.

6) Для определяемого пользователем атрибута: на месте обозначения атрибута (после разделителя ') в имени атрибута, префикс которого обозначает понятие, с которым этот атрибут сопоставлен.

7) Для объявления формального параметра в конкретном объявлении подпрограммы: на месте формального указателя (перед составным разделителем =>) в списке именованного сопоставления параметров в соответствующем вызове подпрограммы.

8) Для объявления локальных параметров настройки в конкретном объявлении компонента: на месте формального указателя (перед составным разделителем =>) в списке именованного сопоставления параметров настройки в соответствующем операторе конкретизации компонента; аналогично, на месте фактического указателя (после составного разделителя =>) в списке сопоставления параметров настройки в соответствующем связывающем указании.

9) Для объявления локального порта в конкретном объявлении компонента: на месте формального указателя (перед составным разделителем =>) в списке именованного сопоставления портов в соответствующем операторе конкретизации компонента; аналогично, на месте фактического указателя (после составного разделителя =>) в списке сопоставления портов в соответствующем связывающем указании.

10) Для объявления формального параметра настройки в конкретном объявлении объекта: на месте формального указателя (перед составным разделителем =>) в списке именованного сопоставления параметров настройки в соответствующем связывающем указании.

11) Для объявления формального порта в конкретном объявлении объекта: на месте формального указателя (перед составным разделителем =>) в списке именованного сопоставления портов в соответствующем связывающем указании.

Наконец, внутри области объявлений, сопоставленной с конструкцией, отличной от объявления структурного типа, любое объявление, стоящее непосредственно в этой области, видимо косвенно на месте суффикса в расширенном имени, префикс которого обозначает эту конструкцию.

Объявление считается *непосредственно видимым* внутри определенной части своей непосредственной области действия; эта часть распространяется до конца этой непосредственной области действия, исключая те места, где это объявление скрыто, что объясняется ниже. Объявление, стоящее непосредственно внутри видимой части пакета, можно сделать непосредственно видимым при помощи описания использования в соответствии с правилами, описанными в 10.4.

Объявление считается *скрытым* в рамках (всей или части) внутренней области объявлений, если эта внутренняя область объявлений содержит *омограф* этого объявления; внешнее объявление тогда является скрытым в рамках непосредственной области действия этого внутреннего омографа. Каждое из двух

объявлений считается омографом другого, если оба они имеют один и тот же идентификатор и совмещение разрешено самое большее для одного из двух. Если совмещение разрешено для обоих объявлений, то каждое из двух является омографом другого, если они имеют один и тот же идентификатор, символ оператора или символьный литерал, а также профиль параметров и типа результата.

В рамках спецификации подпрограммы каждое объявление, имеющее тот же идентификатор, что и эта подпрограмма, скрыто. В таких местах объявление не может быть видимо ни косвенно, ни непосредственно.

Два объявления, стоящие непосредственно в одной и той же области объявлений, не должны быть омографами, если только точно одно из них не является неявным объявлением предопределенной операции. В таких случаях предопределенная операция всегда скрыта другим омографом. В таких местах неявное объявление скрыто внутри всей области действия другого объявления; неявное объявление невидимо ни косвенно, ни непосредственно.

Всякий раз, когда объявление с определенным идентификатором видимо из конкретной точки, этот идентификатор и описанное понятие (если оно есть) также считаются видимыми из этой точки. Непосредственная видимость и косвенная видимость также определены для символьных литералов и символов оператора. Оператор видим непосредственно, если и только если непосредственно видимо соответствующее объявление этого оператора.

Пример —

```
L1: block
      signal A, B: Bit;
    begin
    L2: block
          signal B: Bit;              - - внутренний омограф B
        begin
          A <= B after 5ns;           - - означает L1.A <= L2.B
          B <= L1.B after 10ns;       - - означает L2.B <= L1.B
        end block;
        B <= A after 15 ns;           - - означает L1.B <= L1.A
    end block;
```

10.4 Описания использования

Описание использования (use clause) делает объявления, которые являются видимыми косвенно, видимыми непосредственно.

```
use_clause :: =
    use selected_name {, selected_name}
```

Каждое составное имя (selected name) в описании использования идентифицирует одно и более объявлений, которые станут потенциально непосредственно видимыми. Если суффикс составного имени является простым именем или символом оператора, то такое составное имя идентифицирует только объявление(я) этого простого имени или символа оператора, содержащегося в пакете или библиотеке, обозначаемой префиксом этого составного имени. Если суффиксом является зарезервированное слово all, то это составное имя идентифицирует все объявления, содержащиеся в пакете или библиотеке, обозначаемой префиксом этого составного имени.

Для каждого описания использования существует определенная область текста, называемая *областью действия* этого описания. Эта область начинается непосредственно за описанием использования. Если описание использования является элементом некоторой области объявлений, то область действия этого описания распространяется до конца этой области объявлений. Если описание использования стоит внутри описания контекста модуля проекта, то область действия этого описания использования распространяется до конца области объявлений, сопоставленной с этим модулем. Область действия описания использования может дополнительно быть распространена на объявление конфигурации (см. 10.2).

Для того, чтобы установить, какие объявления становятся непосредственно видимыми в конкретном месте при помощи описания использования, рассмотрим множество объявлений, идентифицированных всеми описаниями использования, области действия которых включают это место. Каждое объявление в этом множестве является потенциально видимым объявлением. Потенциально видимое объявление фактически становится непосредственно видимым, за исключением двух следующих случаев:

1) Потенциально видимое объявление не становится непосредственно видимым, если рассматриваемое место находится внутри непосредственной области действий омографа этого объявления.

2) Потенциально видимые объявления, имеющие одно и то же обозначение, не становятся непосредственно видимыми, если каждое из них не является либо литералом перечисления, либо объявлением подпрограммы.

Примечание — Вышеуказанные правила гарантируют, что объявление, которое становится непосредственно видимым при помощи описания использования, не может скрывать непосредственно видимое объявление, видимость которого установлена другим способом.

Если понятие X, объявленное в пакете P, становится потенциально видимым внутри пакета Q (то есть при помощи описания использования "use P.X;", вставленного в описание контекста пакета Q), а описание контекста модуля проекта R содержит описание использования "use Q.all;", то этим не подразумевается, что понятие X будет потенциально видимо в модуле R. Только те понятия, которые фактически объявлены в пакете Q, будут потенциально видимыми в модуле R (при отсутствии какого-либо другого описания использования).

10.5 Контекст разрешения совмещения

Совмещение определено для подпрограмм и литералов перечисления.

Для совмещенных понятий разрешение совмещения устанавливает фактический смысл, который имеет появление литерала перечисления всякий раз, когда правилами видимости установлено, что в месте появления приемлем более чем один смысл; разрешение совмещения также устанавливает фактический смысл появления оператора.

В таком месте рассматриваются все видимые объявления. Появление допустимо только в том случае, если существует точно одна интерпретация каждого составляющего самого внутреннего полного контекста; полным контекстом является либо объявление, либо спецификация, либо оператор.

При рассмотрении возможных интерпретаций полного контекста используются только синтаксические правила, правила области действия и видимости и правила, описанные ниже:

1) Любое правило, требующее, чтобы имя или выражение имело определенный тип, или имело тот же тип, что и другое имя или выражение.

2) Любое правило, требующее, чтобы тип имени или выражения относили к определенному классу типов; аналогично, любое правило, требующее, чтобы определенный тип был дискретным, целым, действительным, физическим, универсальным, символьным или логическим.

3) Любое правило, требующее, чтобы префикс соответствовал определенному типу.

4) Правила, требующие, чтобы тип агрегата устанавливался исключительно из объемлющего полного контекста. Аналогично, правила, требующие, чтобы тип префикса атрибута, тип выражения оператора выбора или тип операнда преобразования типа устанавливались независимо от контекста.

5) Правила для разрешения вызовов совмещенных подпрограмм; для неявных преобразований универсальных выражений; для интерпретации дискретных диапазонов с границами, имеющими универсальный тип; для интерпретации расширенного имени, префикс которого обозначает подпрограмму.

11 МОДУЛИ ПРОЕКТА И ИХ АНАЛИЗ

В настоящем разделе дано обсуждение общей организации описаний, а также их анализа и последующего определения их в библиотеке проекта.

11.1 Модули проекта

Некоторые конструкции могут быть независимо проанализированы и помещены в библиотеку проекта; такие конструкции называются *модулями проекта* (design units). Один или более модулей проекта в свою очередь составляют *файл проекта* (design file).

```
design_file :: = design_unit { design_unit }

design_unit :: = contex_clause library_unit

library_unit :: =
  primary_unit
  | secondary_unit

primary_unit :: =
  entity_declaration
  | configuration_declaration
  | package_declaration

secondary_unit :: =
  architecture_body
  | package_body
```

Модули проекта в фале проекта анализируются в текстуальном порядке их появления в этом файле. Результатом анализа является определение соответствующего библиотечного модуля в библиотеке проекта. *Библиотечный модуль* (library unit) — это либо первичный, либо вторичный модуль. Вторичный модуль — это отдельно анализируемое тело первичного модуля, полученного в результате предыдущего анализа.

Имя первичного модуля задается первым идентификатором, стоящим после начального зарезервированного слова в этом модуле. Из вторичных модулей имеются только архитектурные тела; имя архитектурного тела задается идентификатором, стоящим после зарезервированного слова architecture. Каждый первичный модуль в конкретной библиотеке должен иметь простое имя, уникальное в рамках библиотеки, а также каждое архитектурное тело, ассоциированное с конкретным объявлением объекта проекта, должно иметь простое имя — уникальное внутри множества имен архитектурных тел, ассоциированных с этим объявлением объекта.

Объявления объектов проекта, архитектурные тела и объявления конфигурации описаны в разделе 1. Объявления пакетов и тела пакетов описаны в разделе 2.

11.2 Библиотеки проекта

Библиотека проекта (design library) представляет собой зависящее от реализации средство хранения модулей проекта, которые ранее подвергались анализу. Конкретная реализация может позволять любое число библиотек проекта.

```
library_clause :: = library logical_name_list ;

logical_name_list :: = logical_name { , logical_name}

logical_name :: = identifier
```

Описание библиотеки (library clause) задает логические имена для библиотек проекта в главном окружении. Описание библиотеки появляется как часть описания контекста, стоящего в начале модуля проекта. Существует определенная область текста, называемая *областью видимости* (scope) описания библиотеки; эта область начинается непосредственно после описания библиотеки и простирается до конца области объявлений, ассоциированной с модулем проекта, в котором стоит это описание. Внутри этой области, за исключением тех мест, где оно скрыто, каждое логическое имя, заданное в описании библиотеки, обозначает библиотеку проекта в главном окружении.

Для конкретного логического имени библиотеки фактическое имя соответствующих библиотек проекта в главном окружении может быть (или нет) одинаковым. Конкретная реализация должна обеспечивать некоторый механизм для ассоциирования логического имени библиотеки с конкретной библиотекой. Такой механизм языком не определяется. Существуют два класса библиотек проекта: рабочие библиотеки и библиотеки ресурсов. *Рабочая библиотека* (working library) — это библиотека, куда помещается библиотечный модуль, полученный в результате анализа модуля проекта. *Библиотека ресурсов* (resource library) — это библиотека, содержащая библиотечные модули, ссылка на которые имеется в анализируемом модуле проекта. В процессе анализа конкретного модуля проекта могут участвовать только одна рабочая библиотека и любое число библиотек ресурсов (включая саму рабочую библиотеку).

Подразумевается, что каждый модуль проекта содержит неявно следующие разделы описания контекста:

```
library STD, WORK; use STD.STANDARD.all;
```

Логическое имя библиотеки STD обозначает библиотеку проекта, в которой содержится пакет STANDARD и пакет TEXTIO; оба они являются единственными стандартными пакетами, определяемыми языком (см. раздел 14). (Описание использования use делает все объявления, содержащиеся внутри пакета STANDARD, непосредственно видимыми в соответствующем модуле проекта; см. 10.4). Логическое имя библиотеки WORK обозначает текущую рабочую библиотеку, используемую в процессе анализа.

Вторичный модуль, соответствующий конкретному первичному модулю, может быть помещен только в ту библиотеку проекта, в которой расположен этот первичный модуль.

Примечание — Проект языка предполагает, что содержимое библиотек ресурсов, перечисленных в описаниях библиотеки, стоящих в описании контекста модуля проекта, будет оставаться неизменным в процессе анализа этого модуля (за исключением того случая, когда библиотека ресурса одновременно является и рабочей библиотекой)

Рекомендуется, чтобы библиотека STD содержала только те библиотечные модули, которые соответствуют модулям проекта, описанным как часть руководства по языку. Эта совокупность модулей может меняться со временем по мере развития языка; однако портабильность проектов будет улучшена, если для любой конкретной версии языка библиотека STD содержит известный набор библиотечных модулей.

11.3 Описание контекста

Описание контекста устанавливает начальную среду, в которой анализируется модуль проекта.

```
context_clause :: = { context_item }
```

```
context_item :: =
  library_clause
  | use_clause
```

Описание библиотеки задает логические имена библиотек, ссылка на которые может иметь место в любом модуле проекта. Описания библиотек даны в 11.2. Описание использования (use clause) делает определенные объявления непосредственно видимыми внутри модуля проекта. Описания использования даны в 10.4.

Зависимости между модулями проекта устанавливаются описаниями использования, то есть модуль проекта, который явно или неявно ссылается на другие библиотечные модули в описании использования, *зависит* (depends) от этих библиотечных модулей. Эти зависимости оказывают влияние на допустимый порядок анализа модулей проекта (см. 11.4).

Примечание — Правила использования описаний использования таковы, что независимо от того, используется имя библиотечного модуля один или более раз в применяемых описаниях использования или даже в одном конкретном описании использования, эффект будет один и тот же.

11.4 Порядок анализа

Правила, устанавливающие порядок, в котором модули проекта могут анализироваться, вытекают непосредственно из правил видимости, а именно:

1) Первичный модуль, на имя которого имеется ссылка внутри конкретного модуля проекта, должен анализироваться раньше этого модуля.

2) Первичный модуль должен анализироваться ранее любого соответствующего вторичного модуля.

Порядок анализа проекта не должен противоречить порядку, определенному вышеуказанными правилами.

Если в процессе анализа модуля проекта обнаруживается какая-либо ошибка, то попытка анализа отвергается, при этом не оказывается никакого эффекта на текущую рабочую библиотеку.

Конкретный библиотечный модуль зависит от изменения любого библиотечного модуля, имя которого используется внутри этого библиотечного модуля. Вторичный модуль потенциально зависит от изменения соответствующего ему первичного модуля. Если библиотечный модуль изменяется (например, в результате повторного анализа соответствующего модуля проекта), то все библиотечные модули, которые потенциально зависят от этого изменения, становятся устарелыми и должны быть повторно анализированы, прежде чем их можно будет использовать вновь.

12 ПРЕДВЫПОЛНЕНИЕ И ВЫПОЛНЕНИЕ

Процесс, при помощи которого объявление достигает своего эффекта, называется *предвыполнением* этого объявления. После своего предвыполнения объявление считается предвыполненным. До окончания своего предвыполнения (а также и до предвыполнения) объявление еще не считается предвыполненным.

Предвыполнение также определено для иерархий проектов, разделов объявлений, разделов операторов (содержащих параллельные операторы) и параллельных операторов. Предвыполнение таких конструкций необходимо, чтобы в конечном счете предвыполнить объявления, описанные внутри этих конструкций.

Для того, чтобы выполнить модель, иерархия проекта, определяющая эту модель, должна быть предвыполнена. Затем имеет место инициализация цепей. Наконец, происходит моделирование. Моделирование состоит из повторяющегося выполнения *цикла моделирования*, в течение которого осуществляется выполнение процессов и модификация цепей.

12.1 Предвыполнение иерархии проекта

Предвыполнение иерархии проекта создает множество процессов, взаимосвязанных при помощи цепей; это множество процессов и цепей может быть выполнено для моделирования поведения этого проекта.

Иерархия проекта может быть определена объектом проекта. Предвыполнение иерархии проекта, определенной этим способом, состоит из предвыполнения оператора блока, эквивалентного внешнему блоку, определяемому этим объектом проекта. Эквивалентный оператор блока определен в 9.6.1. Предвыполнение оператора блока определено в 12.4.1.

Иерархия проекта может быть также определена конфигурацией. Предвыполнение конфигурации состоит из предвыполнения оператора блока, эквивалентного внешнему блоку, определяемому объектом проекта, конфигурируемым этой конфигурацией.

Предвыполнение оператора блока влечет в первую очередь предвыполнение каждого еще непредвыполненного пакета, используемого в этом блоке. (Пакет используется внутри конкретной конфигурации, если имя этого пакета, либо стоящее отдельно либо являющееся префиксом в расширенном имени, используется внутри этой конфигурации). Аналогично, предвыполнение конкретного пакета влечет в первую очередь предвыполнение каждого еще непредвыполненного пакета, используемого внутри этого пакета. Предвыполнение пакета дополнительно включает предвыполнение раздела объявлений, стоящего

в объявлении пакета, за которым следует предвыполнение раздела объявлений, стоящего в соответствующем теле пакета, если оно есть. Предвыполнение раздела объявлений определено в 12.3.

12.2 Предвыполнение заголовка блока

Предвыполнение заголовка блока (block header) состоит из предвыполнения описания параметров настройки (generic clause), описания отображения параметров настройки (generic map clause), описания портов (port clause) и описания отображения портов (port map clause) в указанном порядке.

12.2.1 *Описание параметров настройки*

Предвыполнение описания параметров настройки состоит из предвыполнения каждого из эквивалентных отдельных объявлений параметров настройки, содержащихся в этом описании, в заданном порядке. Предвыполнение объявления параметра настройки состоит из предвыполнения указания подтипа (subtype indication) и последующего создания константы настройки (generic constant) этого подтипа.

Значение константы настройки не определено до тех пор, пока не будет предвыполнено последующее описание отображения параметров настройки, или, при отсутствии последнего, до тех пор, пока не будет вычислено неявное отображение, сопоставленное с этой константой настройки, определяющее значение этой константы.

12.2.2 *Описание отображения параметров настройки*

Предвыполнение описания отображения параметров настройки состоит из предвыполнения списка сопоставления параметров настройки (generic association list). Список сопоставления содержит неявный элемент сопоставления (association element) для каждой константы настройки, для которой нет явного сопоставления; фактическая часть (actual part) такого элемента сопоставления представляет собой неявное выражение (default expression), состоящее в объявлении этой константы.

Предвыполнение списка сопоставления параметров настройки состоит из предвыполнения каждого элемента сопоставления параметров настройки в этом списке. Предвыполнение элемента сопоставления параметров настройки состоит из предвыполнения формальной части (formal part) и вычисления фактической части. Константа настройки или ее подэлемент, обозначаемые формальной частью, затем инициализируются значением, получаемым при вычислении соответствующей фактической части.

12.2.3 *Описание портов*

Предвыполнение описания портов состоит из предвыполнения каждого из эквивалентных отдельных объявлений портов, содержащихся в этом описании, в заданном порядке. Предвыполнение объявления порта состоит из предвыполнения указания подтипа и последующего создания порта этого подтипа.

12.2.4 *Описание отображения портов*

Предвыполнение описания отображения портов состоит из предвыполнения списка сопоставления портов (port association list).

Предвыполнение списка сопоставления портов состоит из предвыполнения каждого элемента сопоставления портов в этом списке. Предвыполнение элемента сопоставления портов состоит из предвыполнения формальной части; порт или его подэлемент, обозначаемый этой частью,затем сопоставляется с сигналом, обозначаемым фактической частью. Такое сопоставление влечет проверку ограничений, налагаемых на сопоставления портов (см. 1.1.1.2.). Возникает ошибка, если эта проверка не имеет успеха.

Если порт типа in в своем объявлении содержит неявное выражение, но для него не задан элемент сопоставления, то это выражение вычисляется и полученное значение становится значением этого порта.

12.3 Предвыполнение раздела объявлений

Предвыполнение раздела объявлений состоит из предвыполнения входящих в него элементов объявлений (declarative item), если они есть, в том порядке, в котором они расположены. В определенных случаях предвыполнение элемента объявлений влечет вычисление выражений, стоящих внутри этого элемента объявлений. Значение каждого объекта, обозначаемого первичным в таком выражении, должно быть определено во время вычисления этого выражения.

Примечание — Из вышеуказанных правил следует, что имя сигнала, объявленное внутри блока, не может быть использовано в выражениях, стоящих в элементах объявлений внутри блока, так как значение сигнала не определено до тех пор, пока иерархия проекта не будет предвыполнена. Однако параметр-сигнал может быть использован в выражениях, стоящих в элементах объявлений, которые в свою очередь используются в разделе объявлений подпрограммы, но при условии, что эта подпрограмма вызывается только после начала моделирования, так как значение каждого сигнала будет определено только к этому моменту.

12.3.1 *Предвыполнение объявления*

Предвыполнение объявления создает объявляемый элемент.

Для каждого объявления правила языка (в частности правила области действия и видимости) таковы, что либо невозможно, либо недопустимо использовать конкретный элемент до предвыполнения объявления, описывающего этот элемент. Например, невозможно использовать имя типа для объявления объекта до того, как соответствующее объявление типа не будет предвыполнено. Аналогично, недопустимо вызывать подпрограмму до предвыполнения ее тела.

12.3.1.1 *Объявления и тела подпрограмм*

Предвыполнение объявления подпрограммы влечет предвыполнение списка интерфейса параметров (parameter interface list) этого объявления; это в свою очередь влечет предвыполнение указания подтипа каждого элемента интерфейса с целью установить подтип каждого формального параметра этой подпрограммы. Предвыполнение тела подпрограммы не имеет никакого другого эффекта, кроме утверждения того факта, что это тело может теперь использоваться для выполнения вызовов этой подпрограммы.

12.3.1.2 *Объявления типов*

Предвыполнение объявления типа в общем состоит из предвыполнения определения типа (type definition) и создания этого типа. Для ограниченного индексируемого типа предвыполнение еще состоит из предвыполнения эквивалентного анонимного неограниченного индексируемого типа, за которым следует предвыполнение именованного подтипа этого неограниченного типа.

Предвыполнение определения перечисляемого типа (enumeration type definiton) не имеет никакого другого эффекта, кроме создания соответствующего типа.

Предвыполнение определения целого, плавающего или физического типа (integer-type definition, floating-type definition, physical-type definition) состоит из предвыполнения соответствующего указания подтипа. Для определения физического типа каждое объявление единицы (unit declaration) в этом определении также предвыполняется. Предвыполнение объявления физической единицы не имеет никакого другого эффекта, кроме создания единицы, определяемой объявлением единицы.

Предвыполнение определения неограниченного индексируемого типа (unconstrained type definition) состоит из предвыполнения указания подтипа элементов (element subtype indication) этого индексируемого типа.

Предвыполнение определения структурного типа (record type definition) состоит из предвыполнения эквивалентных отдельных объявлений элементов (element declaration) в заданном порядке. Предвыполнение объявления элемента состоит из предвыполнения указания подтипа элемента (element subtype indication).

Предвыполнение определения ссылочного типа (access type definition) состоит из предвыполнения соответствующего указания подтипа.

12.3.1.3 *Объявления подтипов*

Предвыполнение объявления подтипа (subtype declaration) состоит из предвыполнения указания подтипа. Предвыполнение указания подтипа создает подтип. Если этот подтип не содержит ограничение (constrain), то это тот же подтип, который обозначается меткой типа (type mark). Предвыполнение указания подтипа, содержащего ограничение, состоит из следующего:

1) Сначала предвыполняется ограничение.

2) Затем делается проверка на совместимость этого ограничения с типом или подтипом, означаемым меткой типа (см. 3.1 и 3.2.1.1).

Предвыполнение ограничения диапазона (range constraint) состоит из вычисления этого диапазона. Вычисление диапазона определяет границы и направление этого диапазона. Предвыполнение ограничения индекса (index constraint) состоит из предвыполнения каждого из дискретных диапазонов (discrete range) в этом ограничении индекса в порядке, не определяемом языком. Предвыполнение ограничения размера (size constraint) состоит из вычисления выражения.

12.3.1.4 *Объявления объектов*

Предвыполнение объявления объекта (object declaration), описывающего объект, отличный от файлового объекта, происходит в следующем порядке:

1) Сначала предвыполняется указание подтипа. Тем самым устанавливается подтип этого объекта.

2) Если объявление объекта содержит явное инициализирующее выражение, то вычислением этого выражения получается начальное значение этого объекта. В противном случае устанавливается неявное начальное значение.

3) Объект создается.

4) Начальное значение присваивается объекту.

Инициализация такого объекта (либо всего объекта, либо его подэлемента) влечет проверку на принадлежность начального значения подтипу этого объекта. Для индексируемого объекта, описанного объявлением объекта сигнала, применяется неявное преобразование подтипа, как в случае оператора присваивания, если только этот объект не является константой, подтипом которой является неограниченный индексируемый тип.

Предвыполнение объявления файлового объекта состоит из предвыполнения указания подтипа, за которым следует создание этого объекта. Затем вычисляется логическое имя файла и файловому объекту сопоставляется соответствующий физический файл.

Примечание — Описанные правила применяются ко всем объявлениям объектов, за исключением объявлений портов и параметров настройки, предвыполнение которых описано в 12.2.1—12.2.4.

Выражение, инициализирующее константу, не обязательно должно быть статическим выражением

12.3.1.5 *Объявления дополнительных имен*

Предвыполнение объявления дополнительного имени (alias declaration) состоит из предвыполнения указания подтипа для утверждения подтипа, сопоставляемого с этим именем, за которым следует создание дополнительного имени как альтернативного для имени объекта, заданного в этом объявлении. Создание дополнительного имени для индексируемого объекта влечет проверку того, что этот подтип включает соответствующий элемент для каждого элемента этого объекта. Возникает ошибка, если эта проверка не имеет успеха.

12.3.1.6 *Объявления атрибутов*

Предвыполнение объявления атрибута (attribute declaration) не имеет никакого другого эффекта, кроме создания шаблона для определения атрибутов элементов.

12.3.1.7 *Объявления компонентов*

Предвыполнение объявления компонента (component declaration) не имеет никакого другого эффекта, кроме создания шаблона для конкретизации экземпляров компонента.

12.3.2 *Предвыполнение спецификации*

Предвыполнение спецификации (specification) не имеет никакого другого эффекта, кроме сопоставления дополнительной информации с ранее объявленным элементом.

12.3.2.1 *Спецификация атрибутов*

Предвыполнение спецификации атрибута (attribute specification) происходит в следующем порядке:

1) Для того, чтобы установить, какие элементы задействуются спецификацией атрибута, предвыполняется спецификация понятия (entity specification).

2) Для определения значения атрибута вычисляется выражение.

3) Создается новый экземпляр указанного атрибута и сопоставляется с каждым из задействованных элементов.

4) Каждому новому экземпляру атрибута присваивается полученное значение.

Присваивание значения экземпляру конкретного атрибута влечет проверку на принадлежность этого значения подтипу указанного атрибута. Для атрибута ограниченного индексируемого типа сначала применяется неявное преобразование подтипа, как в операторе присваивания. Для атрибута неограниченного индексируемого типа в таком преобразовании нет необходимости; ограничения этого атрибута определяются ограничениями самого значения.

Примечание — Выражение в спецификации атрибута необязательно должно быть статическим выражением.

12.3.2.2 *Спецификации конфигураций*

Предвыполнение спецификаций конфигураций (configuration specification) происходит в следующем порядке:

1) Предвыполняется спецификация компонента (component specification); тем самым определяется, какие экземпляры компонента задействуются этой спецификацией конфигурации.

2) Предвыполняется связывающее указание (binding indication); тем самым идентифицируется объект проекта, к которому задействованные экземпляры компонента будут привязаны.

3) С каждой меткой задействованного экземпляра компонента сопоставляется связывающая информация для последующего использования ее в конкретизации этих экземпляров.

Составной частью этого проекта предвыполнения является проверка на существование в указанной библиотеке проекта как объявления объекта (entity declaration), так и соответствующего архитектурного тела (architecture body), подразумеваемых связывающим указанием. Возникает ошибка, если эта проверка не имеет успеха.

12.3.2.3 *Спецификация отключения*

Предвыполнение спецификации отключения (disconnection specification) происходит в следующем порядке:

1) Предвыполняется спецификация защищенных сигналов (guarded signal specification); тем самым идентифицируются сигналы, задействованные этой спецификацией.

2) Вычисляется выражение времени (time expression); тем самым определяется время отключения для драйверов задействованных сигналов.

3) Время отключения сопоставляется с каждым задействованным сигналом для последующего использования при построении операторов отключения в эквивалентных процессах для защищенных назначений (guarded assignment) задействованных сигналов.

12.4 *Предвыполнение раздела операторов*

Параллельные операторы (concurrent statement), стоящие в разделе операторов (statement part) блока, должны быть предвыполнены до начала моделирования. Предвыполнение раздела операторов блока состоит из предвыполнения каждого параллельного оператора в заданном порядке.

12.4.1 *Операторы блока*

Предвыполнение оператора блока (block statement) состоит из предвыполнения заголовка блока (block header), если он задан, за которым следует предвыполнение раздела объявлений блока (block declarative part) и, наконец, предвыполнения раздела операторов блока (block statement part).

Предвыполнение оператора блока может происходить под управлением объявления конфигурации (configuration declaration). В частности, конфигурация блока (block configuration) внутри объявления конфигурации может являться источником последовательности дополнительных неявных спецификаций конфигураций, которые применяются в течение предвыполнения соответствующего оператора блока. Если оператор блока предвыполняется под управлением объявления конфигурации, то такая последовательность неявных спецификаций конфигураций, источником которой является конфигурация блока, предвыполняется как часть раздела объявлений этого блока, стоящая после всех остальных элементов объявлений в этом разделе.

Последовательность неявных спецификаций конфигурации, источником которой является конфигурация блока, включает в себя каждую из спецификаций конфигураций, подразумеваемых конфигурациями компонентов (см. 1.3.2), стоящими непосредственно внутри этой конфигурации блока; порядок спецификаций конфигураций совпадает с порядком конфигураций компонентов.

12.4.2 *Операторы генерации*

Предвыполнение оператора генерации (generate statement) состоит из замещения этого оператора множеством копий (возможно пустым) оператора блока, раздел операторов которого включает параллельные операторы, содержащиеся в этом операторе генерации.

Такие операторы блока считаются *представленными* этим оператором генерации. Затем каждый оператор блока предвыполняется.

Для оператора генерации со схемой генерации (generation scheme) for предвыполнение состоит из предвыполнения дискретного диапазона (discrete range), за которым следует генерация одного оператора блока для каждого значения в этом диапазоне. Все операторы блока при этом имеют следующий вид:

1) Метка оператора блока совпадает с меткой оператора генерации.

2) Раздел объявлений блока содержит единственное объявление константы, которое описывает константу с тем же простым именем, что и у применяемого параметра генерации (generate parameter); значением этой константы является значение параметра генерации, при котором произошла генерация этого конкретного блока. Тип этого объявления определяется базовым типом дискретного диапазона параметра генерации.

3) Раздел операторов блока содержит копию параллельных операторов, стоящих внутри оператора генерации.

Для оператора генерации со схемой генерации if предвыполнение состоит из вычисления логического выражения (boolean expression), за которым следует генерация точно одного оператора блока, если это выражение вырабатывает значение TRUE, или ни одного блока — в противном случае. В случае генерации оператор блока имеет следующий вид:

1) Метка блока совпадает с меткой оператора генерации.

2) Раздел объявления блока пустой.

3) Раздел операторов блока состоит из копий параллельных операторов, стоящих внутри оператора генерации.

Примечание — Повторение меток блока при использовании схемы генерации for не подразумевает многократное объявление этой метки. Последовательность операторов блока, представленных оператором генерации, составляет последовательность ссылок к одной и той же неявно объявленной метке.

12.4.3 *Операторы конкретизации компонентов*

Предвыполнение оператора конкретизации компонента (component instantiation statement) не имеет никакого эффекта, если только экземпляр компонента не является либо полностью связанным (fully bound) с объектом проекта, определенным объявлением объекта и архитектурным телом, либо связанным с конфигурацией этого объекта проекта. Если это так, то предвыполнение соответствующего оператора конкретизации компонента состоит из предвыполнения подразумеваемого оператора блока, представляющего экземпляр этого компонента, и (вместе с этим блоком) подразумеваемого оператора блока, представляющего объект проекта, к которому этот экземпляр привязан. Указанные операторы блока определяются в 9.6.1.

12.4.4 *Другие параллельные операторы*

Все другие параллельные операторы — это либо операторы процесса (process statement), либо операторы, для которых существует эквивалентный оператор процесса.

Предвыполнение оператора процесса происходит в следующем порядке:

1) Предвыполняется раздел объявлений процесса (process declarative part).

2) Создаются драйверы, требуемые этим оператором процесса.

3) Начальная транзакция, определяемая неявным значением, сопоставленным с каждым скалярным сигналом, управляемым этим оператором процесса, вставляется в соответствующий драйвер.

Предвыполнение всех параллельных операторов назначения сигнала (concurrent signal assignment statement) и параллельных операторов утверждения (concurrent assertion statement) состоит из построения эквивалентного оператора процесса, за которым следует его предвыполнение.

12.5 Динамическое предвыполнение

Выполнение некоторых конструкций, содержащих в отличие от параллельных операторов последовательные операторы, также влечет предвыполнение. Такое предвыполнение происходит во время выполнения модели.

Существуют три вида конструкций, в которых предвыполнение происходит динамически в процессе моделирования:

1) Выполнение оператора цикла со схемой итерации (iteration scheme) **for** влечет предвыполнение спецификации параметра цикла (loop parameter specification) до того, как начнется выполнение операторов, содержащихся в этом цикле (см. 8.8.). Такое предвыполнение создает параметр цикла (loop parameter) и вычисляет дискретный диапазон.

2) Выполнение вызова подпрограммы (subprogram call) влечет предвыполнение списка интерфейса параметров (parameter interface list) соответствующего объявления подпрограмм (subprogram declaration); это в свою очередь влечет предвыполнение каждого объявления интерфейса (interface declaration) с целью создания соответствующих формальных параметров. Затем фактические параметры сопоставляются с формальными параметрами. Наконец, предвыполняется раздел объявлений соответствующего тела подпрограммы (subprogram body) и происходит выполнение последовательности операторов, содержащихся в этом теле.

3) Вычисление генератора (allocator), содержащего указание подтипа, влечет предвыполнение этого указания до генерации создаваемого объекта.

Примечание — Из вышеописанных правил следует, что элементы объявлений (declarative item), стоящие внутри раздела объявлений тела подпрограммы, предвыполняются каждый раз, когда соответствующая подпрограмма вызывается; таким образом следующие одно за одним предвыполнения конкретного элемента объявлений, стоящего в таком месте, могут создавать элементы с различными характеристиками. Например, следующие одно за одним предвыполнения одного и того же объявления подтипа, стоящего в теле подпрограммы, могут создавать подтипы с различными ограничениями.

12.6 Выполнение модели

Результатом предвыполнения иерархии проекта является модель, которая может быть выполнена с целью моделирования проекта, представленного этой моделью. Моделирование влечет выполнение определенных пользователем процессов, взаимодействующих друг с другом и окружающей средой.

Ядро модели (kernel process) — это концептуальное представление процесса, который координирует активность определенных пользователем процессов при выполнении моделирования. Этот процесс осуществляет распространение значений сигналов и изменение неявных сигналов (таких, как S'STABLE(T)). Более того, на этом процессе лежит ответственность за обнаружение происходящих событий и осуществление выполнения подходящих процессов в ответ на эти события.

Для любого конкретного сигнала, явно объявленного в модели, ядро модели содержит переменную, представляющую текущее значение этого сигнала. Любое вычисление имени, обозначающего этот сигнал, использует текущее значение соответствующей переменной в ядре модели. При выполнении моделирования ядро модели изменяет эту переменную на основании текущих значений источников соответствующего сигнала.

Дополнительно ядро модели содержит переменную, представляющую текущее значение любого неявно объявленного сигнала GUARD, объявление которого появляется в результате использования защищающего выражения в конкретном операторе блока. Более того, ядро модели содержит драйвер и переменную, представляющие текущее значение любого сигнала вида S'Stable(T), для каждого префикса S и для каждого времени T, к которому есть ссылка в модели; то же самое верно для сигналов вида S'Quiet(T) и S'Transaction.

12.6.1 *Распространение значений сигналов*

По мере продвижения времени моделирования транзакции в планируемой выходной форме сигнала конкретного драйвера (см. 9.2.1.) будут каждая, последовательно друг за другом, становиться значением этого драйвера. Когда драйвер таким способом приобретает новое значение, независимо от того, отличается ли оно от предыдущего значения, считается, что этот драйвер *активен* в течение данного цикла моделирования. Сигнал считается *активным* в течение конкретного цикла моделирования, если:

1) один из его источников является активным;

2) один из его подэлементов является активным;

3) этот сигнал используется в формальной части (formal part) элемента сопоставления (association element) в списке сопоставления портов (port association list), и соответствующий фактический сигнал является активным;

4) этот сигнал является подэлементом разрешенного сигнала, и последний является активным.

Если сигнал конкретного составного типа имеет источник другого типа (и, следовательно, в соответствующем элементе сопоставления задана функция преобразования типа), то каждый скалярный подэлемент этого сигнала рассматривается как активный, если этот источник сам является активным. Аналогично, если порт конкретного составного типа сопоставляется с сигналом другого типа (и, следовательно, в соответствующем элементе сопоставления задана функция преобразования типа), то каждый скалярный подэлемент этого порта считается активным, если фактический сигнал сам является активным.

В дополнение к вышесказанному, неявный сигнал считается активным в течение конкретного цикла моделирования, если ядро модели изменило этот неявный сигнал в этом цикле.

Если сигнал не является активным в течение конкретного цикла моделирования, то этот сигнал считается *пассивным* (quiet) в течение этого цикла моделирования.

Ядро модели вычисляет два значения для некоторых сигналов в течение каждого цикла моделирования. *Задающее значение* конкретного сигнала — это значение, которое этот сигнал обеспечивает в качестве источника другого сигнала. *Эффективное значение* конкретного сигнала — это значение, получаемое вычислением ссылки на этот сигнал внутри выражения. Задающее и эффективное значения сигнала не всегда являются одним и тем же, особенно, когда в распространении значений сигнала участвуют функции разрешения и преобразования типа.

Для скалярного сигнала S задающее значение сигнала вычисляется следующим образом:

1) Если S не имеет источников, то задающим значением S является неявное значение, сопоставленное с S (см. 4.3.1.2.).

2) Если S имеет один источник в виде драйвера и S не является разрешенным сигналом (см. 4.3.1.2.), то задающим значением S является значение этого драйвера.

3) Если S имеет один источник в виде порта и S не является разрешенным сигналом, то задающим значением S является задающее значение формальной части элемента сопоставления, сопоставляющего S с этим портом (см. 4.3.3.2.). Задающее значение формальной части получается вычислением формальной части с использованием задающего значения сигнала, обозначаемого формальным указателем на месте формального указателя.

4) Если S является разрешенным сигналом, то задающее значение S совпадает с разрешенным значением S, полученным выполнением функции разрешения, сопоставленной с S; при этом эта функция вызывается с входным параметром, содержащим конкатенацию задающих значений источников S, за исключением значений драйверов S, текущее значение которых определяется пустой транзакцией (см. 8.3.1).

Для составного сигнала R задающее значение R эквивалентно агрегату из задающих значений каждого скалярного подэлемента сигнала R.

Для скалярного сигнала S эффективное значение S вычисляется следующим образом:

1) Если S является сигналом, описанным объявлением сигнала, портом вида **buffer** или неподключенным портом вида **inout**, то эффективное значение S совпадает с задающим значением S.

2) Если S является подключенным портом вида **in** или **inout**, то эффективное значение S совпадает с эффективным значением фактической части элемента сопоставления, сопоставляющего фактический сигнал с S (см. 4.3.3.2). Эффективное значение фактической части получается вычислением фактической части с использованием эффективного значения сигнала, обозначаемого фактическим указателем на месте фактического указателя.

3) Если S является неподключенным портом вида **in**, то эффективное значение S задается неявным значением, сопоставляемым с S (см. 4.3.1.2).

Для составного сигнала R эффективным значением R является агрегат, составленный из эффективных значений всех подэлементов сигнала R.

Для скалярного сигнала S и задающее, и эффективное значения должны принадлежать подтипу этого сигнала. Для составного сигнала R выполняется неявное преобразование подтипа в подтип сигнала R; для каждого элемента сигнала R должен существовать соответствующий элемент как в задающем, так и в эффективном значениях и наоборот.

Для того, чтобы изменить сигнал в течение конкретного цикла моделирования, ядро модели сначала вычисляет задающее и эффективное значения этого сигнала. Затем ядро изменяет переменную, содержащую текущее значение сигнала, на только что вычисленное эффективное значение следующим образом:

1) Если S является сигналом некоторого типа, отличного от индексируемого типа, то для изменения текущего значения S используется эффективное значение S. При этом делается проверка на принадлежность эффективного значения S подтипу сигнала S. Возникает ошибка, если эта проверка не имеет успеха. Наконец, эффективное значение сигнала S присваивается переменной, представляющей текущее значение сигнала S.

2) Если S является индексируемым сигналом (включая сечения массива), то эффективное значение S неявно преобразуется в подтип сигнала S. При этом делается проверка того, существует ли для каждого

элемента сигнала S соответствующий элемент в эффективном значении и наоборот. Возникает ошибка, если эта проверка не имеет успеха. Результат этого преобразования затем присваивается переменной, представляющей значение сигнала S.

Если при изменении сигнала текущее значение сигнала отличается от предыдущего значения, то считается, что на этом сигнале произошло *событие*. Это определение применяется к любому изменению сигнала, происходящему либо в соответствии с вышеописанными правилами, либо в соответствии с правилами изменения неявных сигналов, описанными в 12.6.2. Появление события может вызвать возобновление и последующее выполнение определенных процессов в течение цикла моделирования, в котором это событие произошло.

Для любого сигнала, отличного от сигнала вида register, вычисление задающего и эффективного значений этого сигнала, а также изменение текущего значения этого сигнала выполняется в каждом цикле моделирования, как описано выше. Изменение сигнала вида register происходит таким же образом в течение каждого цикла моделирования, за исключением тех циклов, в которых все источники этого сигнала имеют текущее значение, определяемое пустой транзакцией.

Изменение неявных сигналов S'STABLE(T), S'QUIET(T) и S'TRANSACTION для любого префикса S и любого времени T не подчиняется вышеописанным правилам; изменение этих сигналов подчинено правилам, описанным в 12.6.2.

Примечание — В цикле моделирования подэлемент составного сигнала может быть пассивным, хотя сам сигнал может быть активным.

Правила в отношении сопоставления фактических параметров с формальными (см. 4.3.3.2.* подразумевают, что если составной сигнал сопоставляется с составным портом вида out, inout или buffer и их формальная, их фактические части элемента сопоставления не содержат функцию преобразования типа, то каждый скалярный подэлемент формального параметра является источником соответствующего подэлемента фактического параметра. В этом случае конкретный подэлемент фактического параметра будет активным, если и только если активен соответствующий подэлемент формального параметра.

Алгоритм вычисления задающего значения скалярного сигнала S является рекурсивным. Например, если S является локальным сигналом, стоящим в списке сопоставления портов, то задающее значение сигнала S может быть получено только после того, как будет вычислено задающее значение соответствующей фактической части. Это может повлечь многократное выполнение указанного алгоритма.

Аналогично, алгоритм вычисления эффективного значения сигнала S является рекурсивным. Например, если формальный порт S вида in сопоставлен с фактическим сигналом A, то эффективное значение A должно быть вычислено до того, как может быть вычислено эффективное значение порта S. Фактический сигнал A может сам быть использованным как формальный порт в списке сопоставления портов.

Для портов вида out или linkage эффективное значение не задается, т.к. эти порты не могут читаться.

12.6.2 *Изменение неявных сигналов*

Ядро модели изменяет значение каждого неявного сигнала GUARD, сопоставленного с оператором блока, имеющим защищающее выражение. Аналогично, ядро модели изменяет значения каждого неявного сигнала S'STABLE(T), S'QUIET(T) или S'TRANSACTION для любого префикса S и любого времени T; это также влечет изменение драйверов сигналов S'STABLE(T) и S'QUIET(T).

Для любого неявного сигнала GUARD текущее значение этого сигнала модифицируется, если и только если соответствующее защищающее выражение содержит ссылку на сигнал S, и S является активным в течение текущего цикла моделирования. В этом случае неявный сигнал GUARD изменяется вычислением соответствующего защищающего выражения и присваиванием результата этого вычисления переменной, представляющей текущее значение этого сигнала.

Для любого неявного сигнала S'STABLE(T) текущее значение этого сигнала (а также текущее состояние соответствующего драйвера) модифицируется, если и только если истинно одно из следующих утверждений:

1) в текущем цикле моделирования на S произошло событие,

2) драйвер сигнала S'STABLE(T) является активным.

Если на сигнале S произошло событие, то сигнал S'STABLE(T) изменяется присваиванием значения FALSE переменной, представляющей текущее значение сигнала S'STABLE(T), а драйверу сигнала S'STABLE(T) присваивается форма сигнала TRUE after T. В противном случае, если драйвер сигнала S'STABLE(T) является активным, то S'STABLE(T) изменяется присваиванием текущего значения этого драйвера переменной, представляющей текущее значение сигнала S'STABLE(T). Наконец, если оба утверждения ложны, то ни переменная, ни драйвер не модифицируются.

Аналогично, для любого неявного сигнала S'QUIET(T) текущее значение этого сигнала (а также текущее состояние соответствующего драйвера) модифицируется, если и только если истинно одно из следующих утверждений:

1) сигнал S является активным,

2) драйвер сигнала S'QUIET(T) является активным.

Если сигнал S является активным, то сигнал S'QUIET(T) изменяется присваиванием значения FALSE переменной, представляющей текущее значение сигнала S'QUIET(T), а драйверу сигнала S'QUIET(T) присваивается форма сигнала TRUE after T. В противном случае, если драйвер сигнала S'QUIET(T) является активным, то сигнал S'QUIET(T) изменяется присваиванием текущего значения этого драйвера

переменной, представляющей текущее значение сигнала S'QUIET(T). Наконец, если оба утверждения ложны, то ни переменная, ни драйвер не модифицируются.

Для любого сигнала S'TRANSACTION текущее значение этого сигнала модифицируется, если и только если сигнал S является активным. Если это так, то сигнал S'TRANSACTION изменяется присваиванием значения выражения (not S'TRANSACTION) переменной, представляющей текущее значение сигнала S'TRANSACTION.

Текущее значение конкретного неявного сигнала R считается *зависимым* от текущего значения другого сигнала S, если верно одно из следующих утверждений:

1) R обозначает неявный сигнал GUARD и S является любым другим неявным сигналом, имя которого стоит в защищающем выражении, определяющем текущее значение сигнала R.

2) R обозначает неявный сигнал S'STABLE(T).

3) R обозначает неявный сигнал S'QUIET(T).

4) R обозначает неявный сигнал S'TRANSACTION.

Эти правила определяют частичную упорядоченность всех сигналов в модели. Изменение неявных сигналов ядром модели гарантируется таким образом, что если конкретный неявный сигнал R зависит от текущего значения другого сигнала S, то текущее значение сигнала S будет изменено в течение конкретного цикла моделирования до изменения текущего значения сигнала R.

Примечание — Вышеописанные правила подразумевают, что если драйвер сигнала S'STABLE(T* является активным, то новым текущим значением этого драйвера является значение TRUE. Более того, вышеописанные правила подразумевают, что если в течение конкретного цикла моделирования на сигнале S произошло событие и драйвер сигнала S'STABLE(T* стал активным в течение этого же цикла, то переменной, представляющей текущее значение сигнала S'STABLE(T*, будет присвоено значение FALSE и текущее значение драйвера сигнала S'STABLE(T* в течение этого цикла никогда не будет присвоено этому сигналу

12.6.3 *Цикл моделирования*

Выполнение модели состоит из фазы инициализирования, за которой следует повторяющееся выполнение операторов процесса, заданных в описании модели. Каждое такое повторение называется *циклом моделирования*. В каждом цикле вычисляются значения всех сигналов, заданных в описании. Если в результате этого вычисления на конкретном сигнале происходит событие, то операторы процесса, которые чувствительны к этому сигналу, будут возобновлены и выполнены в течение этого цикла.

Фаза инициализации состоит из следующих шагов:

1) Вычисляются задающее значение и эффективное значение каждого явно объявленного сигнала и текущее значение сигнала устанавливается в эффективное значение. Это значение рассматривается как значение сигнала, которое он имеет постоянно до начала моделирования.

2) Значение каждого неявного сигнала вида S'STABLE(T) или S'QUIET(T) устанавливается в значение TRUE.

3) Значение каждого неявного сигнала GUARD устанавливается в значение, полученное вычислением соответствующего защищающего выражения.

4) Каждый процесс в модели выполняется до тех пор, пока он не будет приостановлен.

В начале моделирования текущее время имеет значение 0 нс.

Цикл моделирования состоит из следующих шагов:

1) Если нет активных драйверов, то время моделирования продвигается к следующему времени, в которое драйверы станут активным или возобновится процесс. Моделирование завершается, если время моделирования достигло TIME'HIGH.

2) Каждый активный явный сигнал в модели изменяется. (В результате на сигналах могут произойти события.)

3) Каждый неявный сигнал в модели изменяется. (В результате на сигналах могут произойти события.)

4) Каждый процесс P, если P в настоящий момент чувствителен к сигналу S, и на S произошло событие в этом цикле моделирования, то процесс P возобновляется.

5) Каждый возобновленный процесс выполняется до его приостановки.

Примечание — Начальное значение любого неявного сигнала вида S'TRANSACTION не определено.

Изменение явных сигналов описано в 12.6.1; изменение неявных сигналов описано в 12.6.2.

Когда процесс возобновляется, то он добавляется в множество процессов, которые должны быть выполнены в течение текущего цикла моделирования. Однако ни один процесс фактически не начинает выполняться до наступления последнего шага цикла моделирования, к моменту наступления которого идентифицируются все процессы, выполнимые в этом цикле.

13 ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ

Текст описания состоит из одного или более файлов проекта. Каждый файл проекта представляет собой последовательность лексических элементов, каждый из которых составлен из символов; в настоящем разделе описаны правила построения лексических элементов.

13.1 Набор символов

В тексте VHDL-описания разрешено использовать только графические символы и символы управления форматом. Каждому графическому символу соответствует уникальный код из семиразрядного набора символов ISO (ГОСТ 27463) и знак визуального представления. Некоторые графические символы представлены различными знаками визуального представления в зависимости от национальной принадлежности набора символов ISO. Описание языка в настоящем стандарте использует графические символы набора ASCII (ANSI представления набора символов ISO).

```
graphic_character :: = basic_graphic_character
                     | lower_case_letter
                     | other_special_character

basic_graphic_character :: = upper_case_letter
                           | digit | special_character
                           | space_character

basic_character :: = basic_graphic_character
                   | format_effector
```

Базовый набор символов является достаточным для составления любого описания. Символы, входящие в каждую категорию базовых графических символов (basic graphic character), определены следующим образом:

а) заглавные буквы (upper case letters)

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

б) цифры (digits)

0 1 2 3 4 5 6 7 8 9

в) специальные символы (special characters)

" # & ' ( ) * + , — . / : ; < = > _ |

г) символ пробела (space character)

К символам управления форматом (format effector) относятся символы из набора ISO (и ASCII), называемые: горизонтальная табуляция, вертикальная табуляция, возврат каретки, подача строки, подача страницы.

В остальные категории графических символов входят следующие символы:

д) строчные буквы (lower case letters)

a b c d e f g h i j k l m n o p q r s t u v w x y z

е) прочие специальные символы (other special characters)

! $ % @ ? [\ ] ^ ` { } ~

Допустимая замена специальных символов: вертикальная черта (|), диез (#) и кавычки (") — определена в 13.10 настоящего раздела.

Примечание — Символ из набора ISO, соответствующий визуальному символу # в представлении ASCII, появляется как символ фунта стерлинга в французском, немецком и английском национальных стандартах. В любом случае разработка набора шрифтов визуальных символов (например, курсивного или подчеркнутого* не является частью стандарта ISO.

Значения аббревиатур, используемых в данном разделе, следующие: ANSI — Американский Национальный Институт по Стандартам, ASCII — Американский Стандартный код обмена информацией, ISO — Международная Организация по Стандартизации.

Для ссылки на специальные символы используются следующие имена:

символ	имя	символ	имя
"	кавычки	>	больше
#	диез	_	подчеркивание
&	амперсанд	|	вертикальная черта
'	апостроф	!	восклицательный знак
(	левая круглая скобка	$	доллар
*)	правая круглая скобка	%	процент
	звезда, умножить	?	вопросительный знак
+	плюс	@	коммерческое at
,	запятая	[	левая квадратная скобка
—	дефис, минус	\	обратный слэш
.	точка, период	]	правая квадратная скобка
/	слэш, разделитель	^	орфографический знак
:	двоеточие	`	тупое ударение
;	точка с запятой	{	левая фигурная скобка
<	меньше	}	правая фигурная скобка
=	равно	~	тильда

13.2 Лексические элементы, разделители и ограничители

Текст каждого модуля проекта является последовательностью отдельных лексических элементов. Каждый лексический элемент — это либо ограничитель, либо идентификатор (может быть служебным словом), либо абстрактный литерал, либо символьный литерал, либо строковый литерал, либо битово-строковый литерал, либо комментарий.

В некоторых случаях в явном виде необходим разделитель для разделения рядом стоящих лексических элементов (а именно, когда без разделения они интерпретируются как отдельный лексический элемент). Разделителем может быть либо символ пробела, либо символ управления форматом, либо конец строки. Символ пробела не является разделителем внутри комментария, строкового литерала или символьного литерала, включающего этот символ.

Конец строки всегда является разделителем. В языке не определено, что вызывает конец строки. Если для конкретной реализации конец строки задается одним или более символами, то эти символы должны быть символами управления форматом, отличными от символа горизонтальной табуляции. В любом случае последовательность, состоящая из одного или нескольких символов управления форматом, отличных от символа горизонтальной табуляции, должна вызывать по крайней мере один конец строки.

Между любыми двумя соседними лексическими элементами разрешено ставить один или более разделителей, а также перед первым лексическим элементом или после последнего лексического элемента в каждом модуле проекта. По крайней мере один разделитель необходим между идентификатором или абстрактным литералом и соседним идентификатором или абстрактным литералом. Ограничителем может быть один из следующих символов (входящих в базовый набор):

& ' ( ) * + , — . / : < = > |

или один из составных ограничителей, составленный из рядом стоящих специальных символов:

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

Каждый из специальных символов, перечисленных в списке односимвольных ограничителей, является простым ограничителем, за исключением того случая, когда этот символ используется в составном ограничителе или комментарии, строковом литерале, в символьном литерале или абстрактном литерале.

Примечание — Каждый лексический элемент должен помещаться на одной строке, т.к. конец строки рассматривается как разделитель. Кавычки, диез и подчеркивание, а также два рядом стоящих дефиса не являются ограничителями, но могут являться частью других лексических элементов.

В отношении составных ограничителей используются следующие имена:

ограничитель	имя
=>	стрелка
**	двойная звезда, возведение в степень
:=	присваивание переменной
/=	не равно
>=	больше или равно
<=	меньше или равно, а также назначение сигнала
<>	блок

13.3 Идентификаторы

Идентификаторы используются в качестве имен, а также в качестве зарезервированных слов.

identifier :: = letter { [underline] letter_or_digit }

letter_or_digit :: = letter | digit

letter :: = upper_case_letter | lower_case_letter

Все символы в идентификаторе существенны, включая символ подчеркивания (underline), стоящий между буквой (letter) и цифрой (digit) или между двумя соседними буквами, или двумя соседними цифрами. Идентификаторы, различающиеся только по использованию соответствующих заглавных (upper case letter) и строчных (lower case letter) букв, считаются одинаковыми.

Примеры

1 COUNT X C_OUT FFT Decoder
2 VHSIC X1 PageCount STORE_NEXT_ITEM

Примечание — Так как пробел является разделителем, то использование его недопустимо внутри идентификатора.

13.4 Абстрактные литералы

Имеются два класса абстрактных литералов: действительные литералы и целые литералы. Действительным литералом является абстрактный литерал, содержащий точку; целым литералом является абстрактный литерал без точки. Действительными литералами являются литералы типа *универсальный — действительный*. Целыми литералами являются литералы типа *универсальный целый*.

abstract_literal :: = decimal_literal | based_literal

13.4.1 *Десятичные литералы*

Десятичным литералом (decimal literal) является абстрактный литерал, выраженный в десятичной системе счисления (то есть база счисления — это точно 10).

decimal_literal :: = integer [. integer] [exponent]

integer :: = digit { [underline] digit }

exponent :: = E [+] integer | E — integer

Символ подчеркивания, стоящий между двумя соседними цифрами десятичного литерала, не оказывает влияния на значение этого абстрактного литерала. Буква E в выражении экспоненты (если оно используется) может быть написана в любой форме (заглавной или строчной) — значение выражения от этого не меняется.

Экспонента обозначает степень числа 10, на которую умножается значение десятичного литерала, взятого без выражения экспоненты, для получения значения этого литерала с экспонентой. Выражение экспоненты в целом литерале не должно содержать знак минус.

Примеры

1	12	0	1E6	123_456	- - целые литералы
2	12.0	0.0	0.456	3.14159_26	- - действительные - - литералы
3	1.34E—12	1.0E+6	6.023E+24		- - действительные литералы с экспонентой

Примечание — В абстрактных литералах допускаются незначащие нули. Так как пробел является разделителем, то он не разрешен в абстрактных литералах даже между составными частями экспоненты. Значение 0 экспоненты разрешено только в целых литералах.

13.4.2 *Базированные литералы*

Базированный литерал (based literal) — это абстрактный литерал, выраженный в форме, которая явно содержит базу счисления. База счисления может находиться в пределах от 2 до 16.

based_literal :: =
  base # based_integer [. based_integer] # [exponent]

base :: = integer

based_integer :: =
  extended_digit { [underline] extended_digit }

extended_digit :: = digit | letter

Символ подчеркивания, стоящий между двумя соседними цифрами в базированном литерале, не влияет на значение этого абстрактного литерала. База счисления (base) и экспонента должны быть выражены десятичным числом. Буквой, используемой в качестве расширенной цифры (extended digit), могут быть буквы от A до F, соответственно представляющие цифры от 10 до 15. Буква в базированном литерале (будь то буква E в экспоненте или расширенная цифра) может быть написана как в заглавной так и в строчной форме, — значение литерала от этого не меняется.

Предполагается обычный смысл базированного написания; в частности значение каждой расширенной цифры не должно превышать значения базы. Экспонента обозначает степень базы, на которую умножается значение базированного литерала, взятого без выражения экспоненты, для получения значения этого литерала с экспонентой. Выражение экспоненты в целом литерале не должно содержать знак минус.

Примеры

- - целые литералы
- - значения 255

```
2#1111_1111#  16#FF#    016#OFF#

- - целые литералы
- - значения 224

16#E#E1    2#1110_0000#

- - действительные
- - литералы значения
- - 4095.0

16#F.FF#E+2    2#1.1111_1111_111#E11
```

13.5 Символьные литералы

Символьный литерал формируется включением одного из 95 графических символов (включая символ пробела) между двумя символами апострофа. Символьный литерал содержит значение символьного типа.

character_literal :: = ' graphic_character '

Примеры

```
1   'A'
2   '*'
3   ' ' '
4   ' '
```

13.6 Строковые литералы

Строковые литералы формируются из последовательности графических символов (возможно и пустой), заключенной между двумя символами кавычки, используемыми как строковые скобки.

string_literal :: = " { graphic_character } "

Строковый литерал имеет значение, представляющее собой последовательность символьных значений, соответствующих графическим символам этого строкового литерала, исключая сами кавычки. Если необходимо включить символ кавычки в состав строкового литерала, то этот символ надо повторить подряд два раза на соответствующем месте. (Это означает, что строковый литерал, содержащий два рядом стоящих символа кавычки, никогда не интерпретируется как два рядом стоящих строковых литерала).

Длина строкового литерала равна количеству символьных значений в представленной последовательности. (Каждый удвоенный символ кавычки учитывается как один символ).

Примеры

```
1   "Setup time is too short"  - - сообщение об ошибке
2   ""                         - - пустой строковый литерал
3   " " "A" """"               - - три строковых литерала длиной 1.
4   "Characters such as $, % and } are allowed in stringliterals"
```

Примечание — Строковый литерал должен быть написан на одной строчке, т.к. он является лексическим элементом (см. 13.2). Более длинные литералы могут быть составлены при помощи операции конкатенации. Эта операция также может быть использована для получения строковых литералов, содержащих неграфические символы. Спецификация неграфических символов содержится в объявлении типа CHARACTER в пакете STANDARD. Ниже даны примеры использования операции конкатенации:

```
1) "FIRST PART OF A SEQUENCE OF CHARACTERS" &
2) "THAT CONTINUES ON THE NEXT LINE"
3) "sequence that includes the" & ACK & "control character"
```

13.7 Битово-строковые литералы

Битово-строковые литералы (bit-string literal) формируются из последовательности расширенных цифр, заключаемой между двумя символами кавычки, используемых как скобки строки битов, которой предшествует спецификатор базы (base specifier).

bit_string_literal :: = base_specifier " bit_value"

bit_value :: = extended_digit { [underline] extended_digit }

base_specifier :: = B | O | X

Символ подчеркивания, стоящий между двумя соседними цифрами в битово-строковом литерале, не влияет на значение этого литерала. Буквой, используемой в качестве расширенной цифры, могут быть буквы от A до F, представляющие соответственно цифры от 10 до 15. Буква в битово-строковом литерале

(будь то расширенная цифра или спецификатор базы) может быть написана как в заглавной, так и строчной форме — значение литерала от этого не меняется.

Если спецификатор базы счисления есть B, то в качестве расширенных цифр могут выступать только цифры 0 и 1; если O, то цифры от 0 до 7; и если X, то цифры от 0 до 9 и от A до F.

Битово-строковый литерал имеет значение, представляющее собой последовательность значений предопределенного типа BIT (то есть последовательность из '0' и '1'). Если указатель базы счисления есть B, то значение битово-строкового литерала есть сам литерал. Если указатель базы есть O (или X), то значением литерала является последовательность, полученная замещением каждой расширенной цифры последовательностью из трех (или четырех) значений предопределенного типа BIT.

Длина битово-строкового литерала равна числу значений типа BIT в представленной последовательности.

Примеры

1 X"FFF" - - эквивалентно B"1111_1111_1111"
2 O"777" - - эквивалентно B"111_111_111"
3 X"777" - - эквивалентно B"0111_0111_0111"

13.8 Комментарии

Комментарий начинается с двух рядом стоящих символов дефиса и ограничен концом строки. Комментарий может появиться в любой строке VHDL-описания. Наличие или отсутствие комментариев не оказывает влияния на правильность описания. Более того, комментарии не оказывают влияния на выполнение модуля моделирования; единственным назначением комментариев является повышение читабельности описания.

В комментарии допускаются любые символы любых алфавитов, реализуемых инструментальной ЭВМ.

Примеры

```
1  the last sentence above echoes the Algol 68 report end;
   - - processing of line is complete
2  a long comment may be split onto
3  two or more consecutive lines
4  - - - - - - - - - - - - — the first two hyphens start the comment
5  - - это комментарий
```

Примечание — Горизонтальная табуляция может быть использована в комментарии после двойного символа дефиса и это эквивалентно одному или более пробелов (см. 13.2).

13.9 Зарезервированные слова

Идентификаторы, перечисленные ниже, называются зарезервированными словами. Они зарезервированы в языке для специального назначения. Для читабельности руководства зарезервированные слова выделены в нем отчетливыми строчными буквами.

ABS	**ELSE**	**MOD**	**RETURN**
ACCESS	**ELSEIF**	**NAND**	**SELECT**
AFTER	**END**	**NEW**	**SEVERITY**
ALIAS	**ENTITY**	**NEXT**	**SIGNAL**
ALL	**EXIT**	**NOR**	**SUBTYPE**
AND	**FILE**	**NOT**	**THEN**
ARCHITECTURE	**FOR**	**NULL**	**TO**
ARRAY	**FUNCTION**	**OF**	**TRANSPORT**
ASSERT	**GENERATE**	**ON**	**TYPE**
ATTRIBUTE	**GENERIC**	**OPEN**	**UNITS**
BEGIN	**GUARDED**	**OR**	**UNTIL**
BLOCK	**IF**	**OTHERS**	**USE**
BODY	**IMPORT**	**OUT**	**VARIABLE**
BUFFER	**IN**	**PACKAGE**	**WAIT**
BUS	**INITIALIZE**	**PORT**	**WHEN**
CASE	**INOUT**	**PROCEDURE**	**WHILE**
COMPONENT	**IS**	**PROCESS**	**WITH**
CONFIGURATION	**LABEL**	**RANGE**	**XOR**
CONSTANT	**LIBRARY**	**RECORD**	
DISCONNECT	**LINKAGE**	**REGISTER**	
DOWNTO	**LOOP**	**REM**	
	MAP	**REPORT**	

Зарезервированное слово не должно использоваться как объявленный идентификатор.

Примечание — Зарезервированные слова, различающиеся только в использовании соответствующих заглавных и строчных букв, рассматриваются как идентичные (см 13.3). Зарезервированное слово range также используется как имя предопределенного атрибута.

13.10 Возможные замены символов

Для базовых символов вертикальная черта, диез, кавычка разрешены следующие замены:

— вертикальная черта (|) может быть заменена восклицательным знаком (!) при использовании в качестве ограничителя;

— диез (#) в базированном литерале может быть заменен двоеточием (:), при этом заменить надо оба знака в этом литерале;

— кавычки ("), используемые как ограничители в строковом литерале с обеих сторон, могут быть заменены на проценты (%). При этом необходимо заменить обе строковые скобки, а из последовательности символов исключить все символы кавычки. Каждый символ процента внутри последовательности символов должен быть удвоен. Удвоенный символ процента интерпретируется в этом случае как один символ.

Такие замены не изменяет смысл описания.

Примечание — Рекомендуется ограничивать замещение вышеуказанных символов для случаев, когда соответствующий визуальный символ недоступен. Необходимо иметь ввиду, что на некоторых устройствах вертикальная черта представлена как разорванная вертикальная черта и в этом случае замена не рекомендуется.

Правила использования идентификаторов и абстрактных литералов таковы, что строчные и заглавные буквы могут быть использованы без различия. Эти лексические элементы, таким образом, могут быть написаны с использованием только базового набора символов.

14 ПРЕДОПРЕДЕЛЕННОЕ ОКРУЖЕНИЕ ЯЗЫКА

В настоящем разделе описаны предопределенные атрибуты языка VHDL и пакеты, которые должны обеспечиваться всеми реализациями языка.

14.1 Предопределенные атрибуты

Предопределенные атрибуты обозначают значения, функции, типы и диапазоны, сопоставленные с различными видами понятий. Ниже дано описание этих атрибутов. Для каждого атрибута дается следующая информация:

— Вид атрибута: значение,тип, диапазон, функция или сигнал.
— Описание префиксов для каждого атрибута.
— Описание параметра или аргумента, если он есть.
— Результат вычисления атрибута и тип этого результата (если применяется).
— Налагаемые ограничения или комментарии.

T'BASE

Вид:	Тип.
Префикс:	Любой тип или подтип T.
Результат:	Базовый тип T.
Ограничения:	Этот атрибут допускается только в качестве префикса имени другого атрибута; например, T'BASE'LEFT.

T'LEFT

Вид:	Значение.
Префикс:	Любой скалярный тип или подтип T.
Тип результата:	Такой же тип, как T.
Результат:	Левая граница T.

T'RIGHT

Вид:	Значение.
Префикс:	Любой скалярный тип или подтип T.
Тип результата:	Такой же тип, как T.
Результат:	Правая граница T.

T'HIGH

Вид:	Значение.
Префикс:	Любой скалярный тип или подтип T.
Тип результата:	Такой же тип, как T.
Результат:	Верхняя граница T.

T'LOW

Вид:	Значение.
Префикс:	Любой скалярный тип или подтип T.
Тип результата:	Такой же тип, как T.
Результат:	Нижняя граница T.

T'POS(X)

Вид:	Функция.
Префикс:	Любой дискретный или физический тип или подтип T.
Параметр:	Выражение, типом которого является базовый тип T.
Тип результата:	*Универсальный_целый*.
Результат:	Номер позиции значения параметра.

T'VAL(X)

Вид:	Функция.
Префикс:	Любой дискретный или физический тип или подтип T.
Параметр:	Выражение, типом которого является базовый тип T.
Тип результата:	Базовый тип T.
Результат:	Значение, номер позиции которого является значением типа *универсальный_целый*, соответствующим X.

T'SUCC(X)

Вид:	Функция.
Префикс:	Любой дискретный или физический тип или подтип T.
Параметр:	Выражение, типом которого является базовый тип T.
Тип результата:	Базовый тип T.
Результат:	Значение, номер позиции которого на 1 больше, чем номер позиции значения параметра.
Ограничения:	Возникает ошибка, если значение X эквивалентно T'BASE'HIGH.

T'PRED(X)

Вид:	Функция.
Префикс:	Любой дискретный или физический тип или подтип T.
Параметр:	Выражение, типом которого является базовый тип T.
Тип результата:	Базовый тип T.
Результат:	Значение, номер позиции которого на 1 меньше, чем номер позиции значения параметра.
Ограничения:	Возникает ошибка, если значение X эквивалентно T'BASE'LOW.

T'LEFTOF(X)

Вид:	Функция.
Префикс:	Любой дискретный или физический тип или подтип T.
Параметр:	Выражение, типом которого является базовый тип T.
Тип результата:	Базовый тип T.
Результат:	Значение, находящееся слева от значения параметра в диапазоне значений T.
Ограничения:	Возникает ошибка, если значение X эквивалентно T'BASE'LEFT.

T'RIGHTOF(X)

Вид:	Функция.
Префикс:	Любой дискретный или физический тип или подтип T.
Параметр:	Выражение, типом которого является базовый тип T.
Тип результата:	Базовый тип T.
Результат:	Значение, находящееся справа от значения параметра в диапазоне значений T.
Ограничения:	Возникает ошибка, если значение X эквивалентно T'BASE'RIGHT.

A'LEFT[(N)]

Вид: Функция.

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерности A. По умолчанию принимается 1.

Тип результата: Тип, левой границы N-го диапазона индекса A.

Результат: Левая граница N-го диапазона индекса A. (Если A является дополнительным именем индексируемого объекта, то результатом является левая граница N-го диапазона индекса, взятого из объявления A,а не из объявления объекта).

A'RIGHT[(N)]

Вид: Функция.

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерности A. По умолчанию принимается 1.

Тип результата: Тип правой границы N-го диапазона индекса A.

Результат: Правая граница N-го диапазона индекса A. (Если A является дополнительным именем индексируемого объекта,то результатом является правая граница N-го диапазона индекса, взятого из объявления A, а не из объявления объекта).

A'HIGH[(N)]

Вид: Функция.

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерности A. По умолчанию принимается 1.

Тип результата: Тип N-го диапазона индекса A.

Результат: Верхняя граница N-го диапазона индекса A. (Если A является дополнительным именем индексируемого объекта, то результатом является верхняя граница N-го диапазона индекса, взятого из объявления A, а не из объявления объекта).

A'LOW[(N)]

Вид: Функция.

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерности A. По умолчанию принимается 1.

Тип результата: Тип N-го диапазона индекса A.

Результат: Нижняя граница N-го диапазона индекса A. (Если A является дополнительным именем индексируемого объекта,то результатом является нижняя граница N-го диапазона индекса, взятого из объявления A, а не из объявления объекта).

A'RANGE[(N)]

Вид: Диапазон.

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерность A. По умолчанию принимается 1.

Тип результата: Тип N-го диапазона индекса A.

Результат: Диапазон A'LEFT(N) to A'RIGHT(N), если N-й диапазон индекса является восходящим; или диапазон A'LEFT(N) downto A'RIGHT(N), если N-й диапазон индекса является нисходящим. Если A является дополнительным именем индексируемого объекта, то результат определяется N-м диапазоном индекса, взятым из объявления A, а не из объявления объекта.

A'REVERSE RANGE[(N)]

Вид: Диапазон:

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерности A. По умолчанию принимается 1.

Тип результата: Тип N-го диапазона индекса A.

Результат: Диапазон A'RIGHT(N) downto A'LEFT(N), если N-й диапазон индекса является восходящим; или диапазон A'RIGHT(N) to A'LEFT(N), если N-й диапазон индекса является нисходящим. Если A является дополнительным именем индексируемого объекта, то результат определяется N-м диапазоном индекса, взятым из объявления A, а не из объявления объекта.

A'LENGTH[(N)]

Вид: Значение.

Префикс: Любой префикс A, подходящий для индексируемого объекта или его дополнительного имени; или обозначающий ограниченный индексируемый подтип.

Параметр: Локально статическое выражение типа *универсальный_целый*, значение которого не должно превышать размерности A. По умолчанию принимается 1.

Тип результата: *Универсальный_целый.*

Результат: Количество значений в N-м диапазоне индекса A, то есть значение A'HIGH(N)—A'LOW(N)+1.

S'DELAYED[(T)]

Вид: Сигнал.

Префикс: Любой сигнал, обозначаемый статическим именем S.

Параметр: Статическое выражение типа TIME, производящее отрицательное значение. По умолчанию принимается 0 ns.

Тип результата: Базовый тип сигнала S.

Результат: Сигнал, эквивалентный сигналу S, задержанному на T единиц времени. Значение S'DELAYED(t) во время Tn всегда эквивалентно значению сигнала S во время Tn—t.

Конкретно: Пусть R имеет тот же подтип что и S, пусть T >= 0 ns и пусть P будет оператором процесса в форме:

```
P: process (S)
  begin
  R <= transport S after T;
  end process;
```

Предположим, что начальное значение R совпадает с начальным значением S, тогда атрибут 'DELAYED определен таким образом, что S'DELAYED(T)=R для любого T.

Примечание — S'DELAYED(0ns) неэквивалентно S, когда S только что изменился.

S'STABLE[(T)]

Вид:	Сигнал.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Параметр:	Статическое выражение типа TIME, производящее неотрицательное значение. По умолчанию принимается 0 ns.
Тип результата:	Тип Boolean.
Результат:	Сигнал со значением TRUE при условии, что в течение T единиц времени на сигнале S не было события; или сигнал со значением FALSE в противном случае (см. 12.6.2).

Примечание — S'STABLE(0ns)=(S'DELAYED(0ns)=S), и S'STABLE (0 ns) имеет значение FALSE только в том случае, когда S только что изменился.

S'QUIET[(T)]

Вид:	Сигнал.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Параметр:	Статическое выражение типа TIME, производящее неотрицательное значение. По умолчанию принимается 0 ns.
Тип результата:	Тип Boolean.
Результат:	Сигнал со значением TRUE при условии, что сигнал S был пассивным в течение T единиц времени; или сигнал со значением FALSE в противном случае (см. 12.6.2).

Для конкретного цикла моделирования S'QUIET(0ns) имеет значение TRUE, если и только если сигнал S является пассивным в этом цикле.

S'TRANSACTION

Вид:	Сигнал.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Тип результата:	Тип Bit.
Результат:	Сигнал, значение которого инвертируется по отношению к предыдущему значению этого сигнала в каждом цикле моделирования, в котором сигнал S становится активным.

S'EVENT

Вид:	Функция.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Тип результата:	Тип Boolean.
Результат:	Значение, указывающее произошло или нет событие на сигнале S.

Конкретно: Для скалярного сигнала S S'EVENT возвращает значение TRUE, если в течение текущего цикла моделирования на сигнале S произошло событие; в противном случае возвращается значение FALSE.

Для составного сигнала S S'EVENT возвращает значение TRUE, если в течение текущего цикла моделирования на каком-либо скалярном подэлементе сигнала S произошло событие; в противном случае возвращает значение FALSE.

S'ACTIVE

Вид:	Функция.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Тип результата:	Тип Boolean.
Результат:	Значение, указывающее активен или нет сигнал S.

Конкретно: Для скалярного сигнала S S'ACTIVE возвращает значение TRUE, если в течение текущего цикла моделирования сигнал S является активным; в противном случае возвращается значение сигнала FALSE.

Для составного сигнала S S'ACTIVE возвращает значение TRUE, если в течение текущего цикла моделирования какой-либо скалярный подэлемент сигнала S является активным; в противном случае возвращается значение FALSE.

S'LAST_EVENT

Вид:	Функция.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Тип результата:	Тип Time.
Результат:	Временной интервал, прошедший с момента последнего события на сигнале S.

Конкретно: Для скалярного сигнала S S'LAST_EVENT возвращает самое большое значение T (если оно существует) типа TIME, для которого S'DELAYED(T)'STABLE вырабатывает значение TRUE, в противном случае возвращается значение 0 ns.

Для составного сигнала S S'LAST_EVENT возвращает минимальное из значений R'LAST_EVENT для каждого скалярного подэлемента R сигнала S.

S'LAST_ACTIVE

Вид:	Функция.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Тип результата:	Тип Time.
Результат:	Интервал времени, прошедший с момента, когда сигнал S был последний раз активным.

Конкретно: Для скалярного сигнала S S'LAST_ACTIVE возвращает наибольшее значение T (если оно существует) типа TIME, для которого S'DELAYED(T)'QUIET вырабатывает значение TRUE, в противном случае возвращается значение 0 ns.

Для составного сигнала S S'LAST_ACTIVE возвращает минимальное из значений R'LAST_ACTIVE для каждого скалярного подэлемента R сигнала S.

S'LAST_VALUE

Вид:	Функция.
Префикс:	Любой сигнал, обозначаемый статическим именем S.
Тип результата:	Базовый тип S.
Результат:	Предыдущее значение сигнала S, непосредственно имевшее место до последнего изменения S.

Конкретно: Для скалярного сигнала S S'LAST_VALUE=S'DELAYED(T), где T >= 0 ns является наименьшим значением, при котором S'STABLE(T) вырабатывает значение FALSE. Если такого T не существует, то S'LAST_VALUE эквивалентно S.

Для составного сигнала S S'LAST_VALUE эквивалентно агрегату из предыдущих значений каждого элемента сигнала S.

Примечания

1 Если S'STABLE(T) имеет значение FALSE, то по определению для некоторого t, где 0 ns <= t <= T, S'DELAYED(T)/=S.

2 Если Ts является минимальным значением, при котором S'STABLE(Ts) вырабатывает значение FALSE, то для всех t, где 0 ns <= t <= Ts, S'DELAYED(T)=S.

B'BEHAVIOR

Вид:	Значение.
Префикс:	Любой блок, обозначенный меткой, или любой объект проекта, обозначенный именем архитектуры.
Тип результата:	Тип Boolean.
Результат:	Значение TRUE, если блок, определяемый оператором блока или объектом проекта, не содержит оператор конкретизации компонента. В противном случае возвращается значение FALSE.

B'STRUCTURE

Вид:	Значение.
Префикс:	Любой блок, обозначенный меткой, или любой объект проекта, обозначенный именем архитектуры.
Тип результата:	Тип Boolean.
Результат:	Значение TRUE, если блок, определяемый оператором блока или объектом проекта, не содержит непассивный оператор процесса или параллельный оператор, эквивалентом которого является непассивный оператор процесса. В противном случае возвращается значение FALSE.

Примечание — Взаимосвязь между значениями атрибутов LEFT, RIGHT, LOW, HIGH выражается следующей таблицей:

		Восходящий диапазон	Нисходящий диапазон
T'LEFT	=	T'LOW	T'HIGH
T'RIGHT	=	T'HIGH	T'LOW

Так как атрибуты S'EVENT, S'ACTIVE, S'LAST_EVENT, S'LAST_ACTIVE и S'LAST_VALUE являются функциями, а не сигналами, то они не могут вызывать выполнение процесса, хотя значение, возвращаемое такой функцией, может изменяться динамически.

В связи с этим в параллельных контекстах, таких как выражения защиты или параллельные операторы назначения сигнала, рекомендуется использовать эквивалентные атрибуты S'STABLE и S'QUIET, являющиеся сигналами, или выражения, содержащие эти атрибуты. Аналогично, функция STANDARD.NOW также не должна использоваться в параллельных процессах.

14.2 Пакет STANDARD

Пакет STANDARD предопределяет ряд типов, подтипов и функций. Предполагается, что в начале каждого модуля проекта присутствует неявное описание контекста, ссылающееся на этот пакет. Пакет STANDARD не может быть изменен пользователем.

```
package STANDARD is
- - предопределенные перечисляемые типы:
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');
type CHARACTER is (

        NUL,    SOH,    STX,    ETX,    EOT,    ENQ,    ACK,    BEL,
        BS,     HT,     LF,     VT,     FF,     CR,     SO,     SI,
        DLE,    DC1,    DC2,    DC3,    DC4,    NAK,    SYN,    ETB,
        CAN,    EM,     SUB,    ESC,    FSP,    GSP,    RSP,    USP,
        ' ',    '!',    '"',    '#',    '$',    '%',    '&',    ''',
        '(',    ')',    '*',    '+',    ',',    '-',    '.',    '/',
        '0',    '1',    '2',    '3',    '4',    '5',    '6',    '7',
        '8',    '9',    ':',    ';',    '<',    '=',    '>',    '?',
        '@',    'A',    'B',    'C',    'D',    'E',    'F',    'G',
        'H',    'I',    'J',    'K',    'L',    'M',    'N',    'O',
        'P',    'Q',    'R',    'S',    'T',    'U',    'V',    'W',
        'X',    'Y',    'Z',    '[',    '\',    ']',    '^',    '_',
        '`',    'a',    'b',    'c',    'd',    'e',    'f',    'g',
        'h',    'i',    'j',    'k',    'l',    'm',    'n',    'o',
        'p',    'q',    'r',    's',    't',    'u',    'v',    'w',
        'x',    'y',    'z',    '{',    '|',    '}',    '~',    DEL);

type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
- - предопределенные числовые типы:
type INTEGER is range определен-реализацией;
type REAL is range определен-реализацией;
- - предопределенный тип TIME;
type TIME is range определен-реализацией;
   units
      fs;                  - - фемтосекунда
      ps  =  1000 fs;      - - пикосекунда
      ns  =  1000 ps;      - - наносекунда
      us  =  1000 ns;      - - микросекунда
      ms  =  1000 us;      - - миллисекунда
      sec =  1000 ms;      - - секунда
      min =  60  sec;      - - минута
      hr  =  60  min;      - - час
   end units;

- - функция, возвращающая текущее время моделирования;

function NOW return TIME;

- - предопределенные числовые подтипы:
```

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;

subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;

- - предопределенные индексируемые типы:

type STRING is array (POSITIVE range < >) of CHARACTER;

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

end STANDARD;
```

Примечание — ASCII мнемоники для символов FS, GS, RS и US представлены в типе CHARACTER соответственно символами FSP, GSP, RSP, USP во избежание конфликта с единицами типа TIME.

14.3 Пакет TEXTIO

Пакет TEXTIO содержит объявления типов и подпрограмм, обеспечивающих форматированные операции ввода-вывода с символами ASCII.

```
package TEXTIO is

- - описание типов для ввода/вывода текстов

type LINE is access STRING;     - - LINE является указателем
                                - - значения типа STRING
type TEXT is file of STRING;    - - файл из ASCII записей
                                - - переменной длины
type SIDE is (RIGHT, LEFT);     - - для выравнивания выходных
                                - - данных внутри полей
subtype WIDHT is NATURAL;       - - для задания ширины выход-
                                - - ных полей

- - стандартные текстовые файлы

file INPUT; TEXT is in "STD_INPUT";

file OUTPUT; TEXT is in "STD_OUTPUT";

- - процедуры ввода для стандартных типов

procedure READLINE (F: in TEXT; L: out LINE);

procedure READ (L:inout LINE; VALUE:out BIT; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out BIT);

procedure READ (L:inout LINE; VALUE:out BIT_VECTOR; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out BIT_VECTOR);

procedure READ (L:inout LINE; VALUE:out BOOLEAN; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out BOOLEAN);

procedure READ (L:inout LINE; VALUE:out CHARACTER; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out CHARACTER;

procedure READ (L:inout LINE; VALUE:out INTEGER; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out INTEGER);

procedure READ (L:inout LINE; VALUE:out REAL; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out REAL);

procedure READ (L:inout LINE; VALUE:out STRING; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out STRING);

procedure READ (L:inout LINE; VALUE:out TIME; GOOD:out BOOLEAN);
procedure READ (L:inout LINE; VALUE:out TIME);
```

```
- - процедуры вывода для стандартных типов

procedure WRITELINE (F: out TEXT; L: in LINE);

procedure WRITE (L: inout LINE;    VALUE: in BIT;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);

procedure WRITE (L: inout LINE;    VALUE: in  BIT_VECTOR;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);

procedure WRITE (L: inout LINE;    VALUE: in BOOLEAN;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);

procedure WRITE (L: inout LINE;    VALUE: in CHARACTER;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);

procedure WRITE (L: inout LINE;    VALUE: in INTEGER;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);

procedure WRITE (L: inout LINE;    VALUE: in REAL;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);
            DIGITS: in NATURAL:=0);

procedure WRITE (L: inout LINE;    VALUE: in STRING;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0);

procedure WRITE (L: inout LINE;    VALUE: in TIME;
          JUSTIFIED: in SIDE:=RIGHT; FIELD: in WIDHT: = 0;
          UNIT: in TIME: = ns);

- - процедуры обработки позиции файла

function ENDLINE (L: in LINE) return BOOLEAN;

function ENDFILE (F: in TEXT) return BOOLEAN;

end TEXIO;
```

Процедуры READLINE и WRITELINE, описанные в пакете TEXTIO, читают и записывают целиком строки файла типа TEXT. Процедура READLINE вызывает чтение следующей строки из этого файла и возвращает в качестве значения параметра L ссылочное значение, указывающее на объект, представляющий эту строку. Если параметр L содержит непустое ссылочное значение перед вызовом процедуры, то объект, указываемый этим значением, будет уничтожен прежде, чем будет создан новый объект. Процедура WRITELINE вызывает записывание строки, указываемой ссылочным значением параметра L, в этот файл и возвращает через параметр L ссылочное значение, указывающее на пустую строку. Если перед вызовом процедуры параметр L содержит пустое ссылочное значение, то в файл записывается пустая строка.

Каждая процедура READ, описанная в пакете TEXTIO, читает данные из начала строки, указываемой параметром L, и модифицирует этот параметр таким образом, что при выходе он указывает на оставшуюся часть строки. Каждая процедура WRITE аналогично дописывает данные в конец строки, указываемой параметром L; в этом случае параметр L продолжает указывать на всю формируемую строку. Необходимо отметить, что операции записывания не помещают апострофы, окружающие отдельное символьное значение или двойные кавычки, окружающие строковое значение, не делают этого также и соответствующие операции чтения, которые уничтожают эти символы, если они появляются во входном файле.

Для каждого предопределенного типа данных имеются две процедуры READ, описанные в пакете TEXTIO. Первая из них имеет три параметра:

1) L — строка, из которой происходит считывание;
2) VALUE — значение, считываемое из этой строки;
3) GOOD — индикатор, указывающий на правильность завершения операции.

Например, операция READ(L, IntVal, OK) возвратит OK со значением FALSE, L без изменения и IntVal неопределенным, если IntVal является переменной типа INTEGER, а L указывает на строку "ABC". Индикатор успешного завершения операции, возвращаемый через параметр GOOD, позволяет процессу изящно устранить неожиданное противоречие во входном формате. Вторая форма операции чтения имеет всего два параметра L и VALUE. Если требуемый тип не может быть прочитан в VALUE

из строки L, то возникает ошибка. Так, например, операция READ(L, IntVal) приведет к ошибке, если IntVal имеет тип INTEGER, а L указывает на строку "ABC".

Для каждого предопределенного типа данных имеется одна процедура WRITE, описанная в пакете TEXTIO. Каждая из них имеет по крайней мере два параметра:

1) L — строка, куда записываются данные;

2) VALUE — записываемое значение.

Дополнительные параметры JUSTIFIED, FIELD, DIGITS и UNIT управляют форматированием вводимых данных. Каждая операция вывода дописывает данные в строку, форматированную рамками некоторого поля, которое имеет длину, достаточную по крайней мере для представления выводимого значения. Ширина требуемого поля задается параметром FIELD. Так как фактическая ширина поля будет всегда по крайней мере достаточной для хранения строкового представления выводимых данных, то неявное значение 0 для параметра FIELD вызывает вывод данных в поле,которое точно соответствует длине данных (то есть никакие ведущие или сопровождающие пробелы не выводятся). Параметр JUSTIFIED задает вид выравнивания выводимых данных внутри поля либо по правой границе, либо по левой границе; по умолчанию принимается выравнивание по правой границе.

Параметр DIGITS задает количество цифр справа от десятичной точки при выводе действительного числа; неявное значение 0 указывает на то, что число будет выводиться в стандартной форме, состоящей из нормализованной мантиссы и экспоненты (например: 1.079236E—2). Если DIGITS имеет ненулевое значение, то действительное число выводится в форме, состоящей из целой части, за которой следует '.' и затем дробная часть, количество цифр в которой равно значению DIGITS (например: 3.14159).

Параметр UNIT задает формат значения типа TIME. Значение этого параметра должно быть равно одной из единиц, объявленных в описании типа TIME; в результате значение формализуется как целый или действительный литерал, за которым следует имя самой единицы. Например, вызов процедуры WRITE (Line, 5ns, UNIT=>us) приведет к тому, что в строку, указываемую значением Line, добавится строковое значение "0.005us"; в то время как вызов процедуры WRITE(Line, 5ns) приведет к добавлению строкового значения "5ns" (так как по умолчанию значением параметра UNIT является значение ns).

В дополнение к вышеописанным процедурам для строк внутри текстового файла определен предикат ENDLINE. Для входного параметра L типа Line функция ENDLINE возвращает значение выражения (L'Length=0). Функция ENDFILE определяется для файлов типа TEXT неявным объявлением этой функции в рамках объявления файлового типа.

Примечание — Для переменной L типа Line атрибут L'Length возвращает текущую длину строки независимо от операции ввода-вывода, выполняемой над этой строкой. Для строки L в случае операции записывания атрибут L'Length возвращает количество символов, которое уже записано в эту строку, что эквивалентно номеру позиции последнего символа в этой строке. Для строки L в случае операции чтения атрибут L'Length возвращает количество символов, которое осталось в этой строке.

Выполнение операции ввода-вывода может изменить или даже уничтожить строковый объект, указываемый входным параметром L типа Line для этой операции. Таким образом, если значение переменной L типа Line присваивается другой ссылочной переменной и затем с L выполняется операция ввода-вывода, то в результате может появиться повисшая ссылка.

ПРИЛОЖЕНИЕ А
(справочное)

Сводка синтаксических правил

(Данное приложение не является частью стандарта
IEEE Std. 1076—1987, IEEE Standard VHDL)

Приложение содержит сводку синтаксических правил языка VHDL. Эти правила упорядочены по алфавиту в отношении нетерминальных имен стоящих слева. В круглых скобках указаны номера пунктов, в которых эти правила описаны.

```
abstract_literal :: =                                              (13.4)
  decimal_literal | based_literal

access_type_defifnition :: =                                       (3.3)
  access sybtype_indication

actual_designator :: =                                             (4.3.3.2)
  expression
  | signal_name
  | variable_name
  | open

actual_paramater_part :: =                                         (7.3.3)
  parameter_association_list

actual_part :: =                                                   (4.3.3.2)
  actual_designator
  | function_name ( actual_designator )

adding_operator :: =                                               (7.2)
  + | — | &

aggregate :: =                                                     (7.3.2)
( element_association { , element_association } )

alias_declaration :: =                                             (4.3.4)
  alias identifier : subtype_indication
  is name ;

allocator :: =                                                     (7.3.6)
  new  subtype_indication
  | new qualified_expression

architecture_body :: =                                             (1.2)
  architecture identifier of entity_name is
    architecture_declarative_part
  begin
    architecure_statement_part
  end [ architecure_simple_name ] ;
```

```
architecture_declarative_part :: =                                    (1.2.1)
{ block_declarative_item }

architecure_statement_part :: =                                       (1.2.2)
  { concurrent_statement }

array_type_definition :: =                                            (3.2.1)
  unconstrained_array_definition
  | constrained_array_definition

assertion_statement :: =                                              (8.2)
  assert condition
    [ report expression ]
    [ severity expression ] ;

association_element :: =                                              (4.3.3.2)
  [ formal_part => ] actual_part

association_list :: =                                                 (4.3.3.2)
  associaton_element { , association_element }

attribute_declaration :: =                                            (4.4)
  attribute identifier : type_mark ;

attribute_designator :: =                                             (6.6)
  attribute_simple_name

attribute_name :: =                                                   (6.6)
  prefix ' attribute_designator
      [ ( static_expression ) ]

attribute_specification :: =                                          (5.1)
  attribute attribute_desitgnator of
  entity_specification is expression ;

base :: =                                                             (13.4.2)
  integer

base_specifier :: =                                                   (13.7)
  B | O | X

base_unit_declaration :: =                                            (3.1.3)
  identifier ;

based_integer :: =                                                    (13.4.2)
  extended_digit { [underline] extended_digit }

based_literal :: =                                                    (13.4.2)
  base # based_integer [ . based_integer ]
     # [ exponent ]
```

```
basic_character :: =                                                    (13.1)
  basic_graphic_character | format_effector

basic_graphic_character :: =                                            (13.1)
  upper_case_letter | digit
  | special_character | space_character

binding_indication :: =                                                 (5.2.1)
  entity_aspect
  [ generic_map_aspect ]
  [ port_map_aspect ]

bit_string_literal :: =                                                 (13.7)
  base_specifier " bit_value "

bit_value :: =                                                          (13.7)
  extended_digit { [underline] extended_digit }

block_configuration :: =                                                (1.3.1)
  for block_specification
    { use_clause }
    { configuration_item }
  end for;

block_declarative_item :: =                                             (1.2.1)
  subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | signal_declaration
  | file_declaration
  | alias_declaration
  | component_declaration
  | attribute_declaration
  | attribute_specification
  | configuration_specification
  | disconnection_specification
  | use_clause

block_declarative_part :: =                                             (9.1)
  { block_declarative_item }

block_header :: =                                                       (9.1)
  [ generic_clause [ generic_map_aspect ; ] ]
  [ port_clause [ port_map_aspect ; ] ]

block_specification :: =                                                (1.3.1)
  architecture_name
  | block_statement_label
  | generate_statement_label
      [ ( index_specification ) ]
```

```
block_statement :: =                                                    (9.1)
  block_label :
    block_ [ ( guard_expression ) ]
        block_header
        block_declarative_part
    begin
        block_statement_part
    end block [ block_label ] ;

block_statement_part :: =                                               (9.1)
  { concurrent_statement }

case_statement :: =                                                     (8.7)
  case expression is
    case_statement_alternative
    { case_statement_alternative }
  end case;

case_statement_alternative :: =                                         (8.7)
  when choices =>
    sequence_of_statements

character_literal :: =                                                  (13.5)
  ' graphic_character '

choice :: =                                                             (7.3.2)
  simple_expression
  | discrete_range
  | element_simple_name
  | others

choices :: =                                                            (7.3.2)
  choice { | choice }

component_configuration :: =                                            (1.3.2)
  for component_specification
    [ use binding_indication ; ]
    [ block_configuration ]
  end for;

component_declaration :: =                                              (4.5)
  component identifier
    [ local_generic_clause ]
    [ local_port_clause ]
  end component ;

component_instantiation_statement :: =                                  (9.6)
  instantiation_label:
    component_name
      [ generic_map_aspect ]
      [ port_map_aspect ];
```

component_specification :: = (5.3)
  instantiation_list : *component*_name

composite_type_definition :: = (3.2)
  array_type_definition
  | record_type_definition

concurrent_assertion_statement :: = (9.4)
  [label :] assertion_statement

concurrent_procedure_call :: = (9.3)
  [label :] procedure_call_statement

concurrent_signal_assignment_statement :: = (9.5)
    [label :] conditional_signal_assignment
  | [label :] selected_signal_assignment

concurrent_statement :: = (9)
  block_statement
  | process_statement
  | concurrent_procedure_call
  | concurrent_assertion_statement
  | concurrent_signal_assignment_statement
  | component_instantiation_statement
  | generate_statement

condition :: = (8.1)
  *boolean*_expression

condition_clause :: = (8.1)
  until condition

conditional_signal_assignment :: = (9.5.1)
  target <= options conditional_waveforms ;

conditional_waveforms :: = (9.5.1)
  { waveform when condition else }
  waveform

configuration_declaration :: = (1.3)
  configuration identifier of *entity*_name is
    configuration_declarative_part
    block_configuration
  end [*configuration*_simple_name] ;

configuration_declarative_item :: = (1.3)
  use_clause
  | attribute_specification

configuration_declarative_part :: = (1.3)
  { configuration_declarative_item }

configuration_item :: = (1.3.1)
  block_configuration
  | component_configuration

configuration_specification :: = (5.2)
  for component_specification use
  binding_indication ;

constant_declaration :: = (4.3.1.1)
  constant identifier_list : subtype_indication
  [:= expression] ;

constrained_array_deifinition :: = (3.2.1)
  array index_constraint of_(
      *element*_subtype_indication

constraint :: = (4.2)
  range_constraint | index_constraint

context_clause :: = (11.3)
  { context_item }

context_item :: = (11.3)
  library_clause | use_clause

decimal_literal :: = (13.4.1)
  integer [. integer] [exponent]

declaration :: = (4)
  type_declaration
  | subtype_declaration
  | object_declaration
  | file_declaration
  | interface_declaration
  | alias_declaration
  | attribute_declaration
  | component_declaration
  | entity_declaration
  | configuration_declaration
  | subprogram_declaration
  | package_declaration

design_file :: = (11.1)
  design_unit { design_unit }

disign_unit :: = (11.1)
  context_clause library_unit

designator :: = (2.1)
  identifier | operator_symbol

direction :: = (3.1)
  to | downto

disconnection_specification :: = (5.3)
  disconnect guarded_signal_specification
  after *time*_expression ;

discrete_range :: = (3.2.1)
  *discrete*_subtype_indication | range

element_association :: = (7.3.2)
  [choices =>] expression

element_declaration :: = (3.2.2)
  identifier_list : element_subtype_definition ;

element_subtype_difinition :: = (3.2.2)
  subtype_indication

entity_aspect :: = (5.2.1.1)
  entity *entity*_name
      [(*architecture*_identifier)]
  | **configuration** *configuration*_name
  | **open**

entity_class :: = (5.1)
    entity | **architecture** | **configuration**
  | **procedure** | **function** | **package**
  | **type** | **subtype** | **constant (8**
  | **signal** | **variable** | **component**
  | **label**

entity_declaration :: = (1.1)
  entity identifier **is**
    entity_header
    entity_declarative_part
  [**begin**
      entity_statement_part]
    end [*entity*_simple_name] ;

entity_declarative_item :: = (1.1.2)
  subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | signal_declaration
  | file_declaration
  | alias_declaration
  | attribute_declaration
  | attribute_specification
  | disconnection_specification
  | use_clause

entity_declarative_part :: = (1.1.2)
  { entity_declarative_item }

```
entity_designator :: =                                                    (5.1)
  simple_name | operator_symbol

entity_header :: =                                                        (1.1.1)
  [ formal_generic_clause ]
  [ formal_port_clause ]

entity_name_list :: =                                                     (5.1)
  entity_designator { , entity_designator }
  | others | all

entity_specification :: =                                                 (5.1)
  entity_name_list : entity_class

entity_statement :: =                                                     (1.1.3)
  concurrent_assertion_statement
  | passive_concurrent_procedure_call
  | passive_process_statement

entity_statement_part :: =                                                (1.1.3)
  { entity_statement }

enumeration_literal :: =                                                  (3.1.1)
  identifier | character_literal

enumeration_type_definition :: =                                          (3.1.1)
  ( enumeration_literal { , enumeration_literal } )

exit_statement :: =                                                       (8.10)
  exit [ loop_label ] [ when condition ] ;

exponent :: =                                                             (13.4.1)
  E [+] integer | E — integer

expression :: =                                                           (7.1)
  relation    { and relation }
  | relation { or relation }
  | relation { xor relation }
  | relation [ nand relation ]
  | relation [ nor relation ]

extended_digit :: =                                                       (13.4.2)
  digit | leter

factor :: =                                                               (7.1)
  primary [ ** primary ]
  | abs primary
  | not primary

file_declaration :: =                                                     (4.3.2)
  file identifier : subtype_indication is
  [ mode ] file_logical name ;
```

file_logical_name :: = (4.3.2)
*string*_expression

file_type_definition :: = (3.4)
file **of** type_mark

floating_type_definition :: = (3.1.4)
range_constraint

formal_designator :: = (4.3.3.2)
*generic*_name
| *port*_name
| *parameter*_name

formal_parameter_list :: = (2.1.1)
*parameter*_interface_list

formal_part :: = (4.3.3.2)
formal_designator
| *function*_name (formal_designator)

full_type_declaration :: = (4.1)
type identifier **is** type_definition ;

function_call :: = (7.3.3)
*function*_name [(actual_parameter_part)]

generate_statement :: = (9.7)
*generate*_label :
generation_scheme **generate**
{ concurrent_statement }
end generate [*generate*_label] ;

generation_scheme :: = (9.7)
for *generate*_parameter_specification
| **if** condition

generic_clause :: = (1.1.1)
generic (generic_list) ;

generic_list :: = (1.1.1.1)
*generic*_interface_list

generic_map_aspect :: = (5.2.1.2)
generic map (*generic*_association_list)

graphic_character :: = (13.1)
basic_graphic_character | lower_case_letter
| other_special_character

guarded_signal_specification :: = (5.3)
*guarded*_signal_list : type_mark

identifier :: = (13.3)
letter { [underline] letter_or_digit }

identifier_list :: = (3.2.2)
identifier { , identifier }

if_statement :: = (8.6)
if condition then sequence_of_statements
{ elsif condition then "
sequence_of_statements }
[else sequence_of_statements]
end if;

incomplete_type_declaration :: = (3.3.1)
type identifier ;

index_constraint :: = (3.2.1)
(discrete_range { , discrete_range })

index_specification :: = (1.3.1)
discrete_range | *static*_expression

index_subtype_deifinition :: = (3.2.1)
type_mark range < >

indexed_name :: = (6.4)
prefix (expression { , expression })

instantiation_list :: = (5.2)
*instantiation*_label { , *instantiation*_label }
| others
| all _

integer :: = (13.4.1)
digit { [underline] digit }

integer_type_definition :: = (3.1.2)
range_constraint

interface_constant_declaration :: = (4.3.3)
[constant] identifier_list : [in]
subtype_indication [: = *static*_expression]

interface_declaration :: = (4.3.3)
interface_constant_declaration
| interface_signal_declaration
| interface_variable_declaration

interface_element :: = (4.3.3.1)
interface_declaration

interface_list :: = (4.3.3.1)
interface_element { ; interface_element }

interface signal_declaration :: = (4.3.3)
  [**signal**] identifier_list : [mode]
  subtype_indication [**bus**]
  [: = static_expression]

interface_variable_declaration :: = (4.3.3)
  [**variable**] identifier_list : [mode]
  subtype_indication [: = *static*_expression]

iteration_scheme :: = (8.8)
  while condition
  | **for** *loop*_parameter_specification

label :: = (9.7)
  identifier

letter :: = (13.3)
  upper_case_letter | lower_case_letter

letter_or_digit :: = (13.3)
  letter | digit

library_clause :: = (11.2)
  library logical_name_list ;

library_unit :: = (11.1)
  primary_unit | secondary_unit

literal :: = (7.3.1)
  numeric_literal
  | enumeration_literal
  | string_literal
  | bit_string_literal
  | **null**

logical_name :: = (11.2)
  identifier

logical_name_list :: = (11.2)
  logical_name { , logical_name }

logical_operator :: = (7.2)
  and | **or** | **nand** | **nor** | **xor**

loop_statement :: = (8.8)
  [*loop*_label :]
    [iteration_scheme] **loop**
      sequence_of_statements
    end loop [*loop*_label] ;

miscellaneous_operator :: = (7.2)
  ** | **abs** | **not**

mode :: = (4.3.3)
  in | **out** | **inout** | **buffer** | **linkage**

multiplying_operator :: = (7.2)
  * | / | **mod** | **rem**

name :: = (6.1)
  simple_name
  | operator_symbol
  | selected_name
  | indexed_name
  | slice_name
  | attribute_name

next_statement :: = (8.9)
  next [*loop*_label] [**when** condition] ;

null_statement :: = (8.12)
  null;

numeric_literal :: = (7.3.1)
  abstract_literal | physical_literal

object_declaration :: = (4.3.1)
  constant_declaration
  | signal_declaration
  | variable_declaration

operator_symbol :: = (2.1)
  string_literal

options :: = (9.5)
  [**guarded**] [**transport**]

package_body :: = (2.6)
  package body *package*_simple_name **is**
    package_body_declarative_part
  end [*package*_simple_name] ;

package_body_declarative_item :: = (2.6)
  subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | file_declaration
  | alias_declaration
  | use_clause

package_body_declarative_part :: = (2.6)
  { package_body_declarative_item }

```
package_declaration :: =                                        (2.5)
  package identifier is "
    package_declarative_part
  end [ package_simple_name ] ;

package_declarative_item :: =                                   (2.5)
  subprogram_declaration
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | signal_declaration
  | file_declaration
  | alias_declaration
  | component_declaration
  | attribute_declaration
  | attribute_specification
  | disconnection_specification
  | use_clause

package_declarative_part :: =                                   (2.5)
  { package_declarative_item }

parameter_specification :: =                                    (8.8)
  identifier in discrete_range

physical_literal :: =                                           (3.1.3)
  [ abstract_literal ] unit_name

physical_type_definition :: =                                   (3.1.3)
  range_constraint
  units
    base_unit_declaration
    { secondary_unit_declaration }
  end units

port_clause :: =                                                (1.1.1)
  port ( port_list ) ;

port_list :: =                                                  (1.1.1.2)
  port_interface_list

port_map_aspect :: =                                            (5.2.1.2)
  port map ( port_association_list )

prefix :: =                                                     (6.1)
  name | function_call
```

```
primary :: =                                                    (7.1)
  name
  | literal
  | aggregate
  | function_call
  | qualified_expression
  | type_conversion
  | allocator
  | ( expression )

primary_unit :: =                                               (11.1)
  entity_declaration
  | configuration_declaraton
  | package_declaraton

procedure_call_statement :: =                                   (8.5)
  procedure_name [ ( actual_parameter_part ) ] ;

process_declarative_item :: =                                   (9.2)
  subprogram_declaration
  | subprogram_body
  | type_declaration
  | subtype_declaration
  | constant_declaration
  | variable_declaration
  | file_declaration
  | alias_declaration
  | attribute_declaration
  | attribute_specification
  | use_clause

process_declarative_part :: =                                   (9.2)
  { process_declarative_item }

process_statement :: =                                          (9.2)
  [ process_label : ]
    process [ ( sinsitivity_list ) ]
      process_declarative_part
    begin
      process_statement_part
    end process [ process_label ] ;

process_statement_part :: =                                     (9.2)
  { sequential_statement }

qualified_expression :: =                                       (7.3.4)
    type_mark ' ( expression )
  | type_mark , aggregate

range :: =                                                      (3.1)
  range_attribute_name
  | simple_expression direction simple_expression
```

```
range_constraint :: =                                                    (3.1)
  range range

record_type_definition :: =                                              (3.2.2)
  record
    element_declaration
    { element_declaration }
  end record

relation :: =                                                            (7.1)
  simple_expression
    [ relational_operator simple_expression ]

relational_operator :: =                                                 (7.2)
    = | /= | < | <= | > | >=

return_statement :: =                                                    (8.11)
  return [ expression ] ;

scalar_type_definition :: =                                              (3.1)
  enumeration_type_definition
  | integer_type_definition
  | floating_type_definition
  | physical_type_definition

secondary_unit :: =                                                      (11.1)
  architecture_body | package_body

secondary_unit_declaraton :: =                                           (3.1.3)
  identifier = physical_literal ;

selected_name :: =                                                       (6.3)
  prefix . suffix

selected_signal_assignment :: =                                          (9.5.2)
  with expression select
    target < = options selected_waveforms ;

selected_waveforms :: =                                                  (9.5.2)
  { waveform when choices , }
  waveform when choices

sensitivity_clause :: =                                                  (8.1)
  on sensitivity_list

sensitivity_list :: =                                                    (8.1)
  signal_name { , signal_name }

sequence_of_statements :: =                                              (8)
  { sequential_statement }
```

(8)

```
sequential_statement :: =
  wait_statement
  | assertion_statement
  | signal_assignment_statement
  | variable_assignment_statement
  | procedure_call_statement
  | if_statement
  | case_statement
  | loop_statement
  | next_statement
  | exit_statement
  | return_statement
  | null_statement
```

(7.2)

```
sign :: =
  + | —
```

(8.3)

```
signal_assignment_statement :: =
  target <= [ transport ] waveform ;
```

(4.3.1.2)

```
signal_declaration :: =
  signal identifier_list : subtype_indication
    [ signal_king ] [ : = expression ] ;
```

(4.3.1.2)

```
signal_kind :: =
  register | bus
```

(5.3)

```
signal_list :: =
  signal_name { , signal_name }
  | others | all
```

(7.1)

```
simple_expression :: =
  [ sign ] term { adding_operator term }
```

(6.2)

```
simple_name :: =
  identifier
```

(6.5)

```
slice_name :: =
  prefix ( discrete_range )
```

(13.6)

```
string_literal :: =
  " { graphic_character } "
```

(2.2)

```
subprogram_body :: =
  subprogram_specification is
    subprogram_declarative_part
  begin
    subprogram_statement_part
  end [ designator ] ;
```

(2.1)

```
subprogram_declaration :: =
  subprogram_specification ;
```

```
subprogram_declarative_item :: =                                    (2.2)
  subprogram_declaration
  | subprogram_body
  | type_decla ation
  | subtype_declaration
  | constant_declaration
  | variable_declaration
  | file_declaration
  | alias_declaration
  | attribute_declaration
  | attribute_specification
  | use_clause

subprogram_declarative_part :: =                                    (2.2)
  { subprogram_declarative_item }

subprogram_specification :: =                                       (2.1)
  procedure designator
      [ ( formal_parameter_list ) ]
  | function  designator
      [ ( formal_parameter_list ) ]
    return type_mark

subprogram_statement_part :: =                                      (2.2)
  { sequential_statement }

subtype_declaration :: =                                            (4.2)
  subtype identifier is subtype_indication ;

subtype_indication :: =                                             (4.2)
  [ resolution_function_name ] type_mark
  [ constraint ]

suffix :: =                                                         (6.3)
  simple_name
  | character_literal
  | operator_symbol
  | all

target :: =                                                         (8.3)
  name | aggregate

term :: =                                                           (7.1)
  factor { multiplying_operator factor }

timeout_clause :: =                                                 (8.1)
  for time_expression

type_conversion :: =                                                (7.3.5)
  type_mark ( expression )

type_declaration :: =                                               (4.1)
  full_type_declaration
  | incomplete_type_declaration
```

```
type_definition :: =                                                    (4.1)
  scalar_type_definition
  | composite_type_definition
  | access_type_definition
  | file_type_definition

type_mark :: =                                                          (4.2
  type_name | subtype_name

unconstrained_array_definition :: =                                     (3.2.1)
  array ( index_subtype_definition
      { , index_subtype_definition } )
  of element_subtype_indication

use_clause :: =                                                         (10.4)
  use selected_name { , selected_name } ;

variable_assignment_statement :: =                                      (8.4)
  target : = expression ;

variable_declaration :: =                                               (4.3.1.3)
  variable identifier_list :
      subtype_indicator [ : = expression ] ;

wait_statement :: =                                                     (8.1)
  wait [ sensitivity_clause ] [ condition_clause ]
          [ timeout_clause ] ;

waveform :: =                                                           (8.3)
  waveform_element { , waveform_element }

waveform_element :: =                                                   (8.3.1)
  value_expression [ after time_expression ]
  | null [ after time_expression ]
```

УДК 681.3.06.:006.354 ОКС 35.060 П85 ОКСТУ 5001

Ключевые слова: автоматизированное проектирование, программные средства, язык моделирования, цифровая система

Редактор *Т.С. Шеко*
Технический редактор *О.Н. Власова*
Корректор *А.С. Черноусова*
Компьютерная верстка *Е.Н. Мартемьянова*

Сдано в набор 03.04.95. Подписано в печать 20.07.95. Усл. печ. л. 17,5. Усл. кр.-отт. 17,75. Уч.-изд. л. 16,74.
Тираж 340 экз. С2656. Зак. 1406.

ИПК Издательство стандартов
107076, Москва, Колодезный пер., 14.
Набрано в Издательстве на ПЭВМ.
Калужская типография стандартов, ул. Московская, 256.

gost.e-ecolog.ru