4.1. ОБЩИЕ СВЕДЕНИЯ

Языки описания аппаратуры (Hardware Description Languages) являются набором формальных записей, которые могут быть использованы на всех этапах разработки цифровых электронных систем Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком, он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры так же, как и для передачи данных о проекте, модификации и сопровождения Наиболее универсальным и распространенным языком описания аппаратуры является VHDL На этом языке возможно как поведенческое, так структурное и потоковое описание цифровых схем

Язык VHDL используется во многих системах для моделирования цифровых схем, проектирования программируемых логических интегральных микросхем, базовых матричных кристаллов, заказных интегральных микросхем

С точки зрения программиста язык VHDL состоит как бы из двух компонент — общеалгоритмической и проблемно-ориентированной

Общеалгоритмическая компонента VHDL — это язык, близкий по синтаксису и семантике к современным языкам программирования типа Паскаль, С и др Язык относится к классу строго типизированных Помимо встроенных (пакет STANDART) простых (скалярных) типов данных целый, вещественный, булевый, битовый, данных типа время, данных типа ссылка (указатель) пользователь может вводить свои типы данных (перечисление, диапазон и др)

Помимо скалярных данных можно использовать агрегаты массивы array, в том числе и битовые векторы bit_vector, и символьные строки string, записи record, файлы file

Последовательно выполняемые (последовательные) операторы VHDL могут использоваться в описании процессов, процедур и функций Их состав включает.

- оператор присваивания переменной (=),
- последовательный оператор назначения сигналу (<=),
- последовательный оператор утверждения (assert),
- условный (if),
- выбора (case),
- цикла (loop),
- пустой оператор (null),
- оператор возврата процедуры — функции (return),
- оператор последовательного вызова процедуры.

Язык поддерживает концепции пакетного и структурного программирования Сложные операторы заключены в операторные скобки if- end if, process- end process, case- end case, loop- end loop и т. д

Различаются локальные и глобальные переменные Область "видимости" локальных переменных ограничена пределами блока (процессного, процедурного, оператора блока, оператора описания архитектуры)

Фрагменты описаний, которые могут независимо анализироваться компилятором и при отсутствии ошибок помещаться в библиотеку проекта (рабочую библиотеку Work), называются проектными пакетами design unit Такими пакетами могут быть объявление интерфейса объекта проекта entity, объявление архитектуры architecture, объявление конфигурации configuration, объявление интерфейса пакета package и объявление тела пакета package body

Модули проекта, в свою очередь, можно разбить на две категории первичные и вторичные К первичным пакетам относятся объявления пакета, объекта проекта, конфигурации К вторичным — объявление архитектуры, тела пакета Один или несколько модулей проекта могут быть помещены в один файл, называемый файлом проекта (design file)

Каждый проанализированный модуль проекта помещается в библиотеку проекта (design library) и становится библиотечным модулем (library unit)

Каждая библиотека проекта в языке VHDL имеет логическое имя (идентификатор)

По отношению к сеансу работы с VHDL- системой существует два класса рабочих библиотек проекта. рабочие библиотеки и библиотеки ресурсов

Рабочая библиотека — это библиотека WORK, с которой в данном сеансе работает пользователь и в которую помещается пакет, полученный в результате анализа пакета проекта

Библиотека ресурсов — это библиотека, содержащая библиотечные модули, ссылка на которые имеется в анализируемом модуле проекта

В каждый конкретный момент времени пользователь работает с одной рабочей библиотекой и произвольным количеством библиотек ресурсов

Модули, как и в обычных алгоритмических языках, — это средство выделения из ряда программ и подпрограмм общих типов данных, переменных, процедур и функций, позволяющее упростить, в частности, процесс их замены

Так же, как в описаниях проектируемых систем разделяются описания интерфейсов и тел, в VHDL у пакета разделяются описание интерфейса и тела пакета По умолчанию предусмотрено подключение стандартных пакетов STANDART и TEXT 10 Пакет STANDART, в частности, содержит описание булевых операций над битовыми данными и битовыми векторами Нестандартные пакеты реализуются пользователями, желающими более точно отобразить свойства описываемых ими объектов Например, можно в пользовательском пакете переопределить логические операции И, ИЛИ и НЕ и перейти от булевого (0, 1) к многозначному (1, 0, X, Z) алфавиту моделирования

Проблемно-ориентированная компонента позволяет описывать цифровые системы в привычных разработчику понятиях и терминах Сюда можно отнести

- понятие модельного времени now,
- данные типа time, позволяющие указывать время задержки в физических единицах,
- данные вида сигнал signal, значение которых изменяется не мгновенно, как у обычных переменных, а с указанной задержкой, а также специальные операции и функции над ними,
- средства объявления объектов entity и их архитектур architecture

Если говорить про операторную часть проблемно-ориентированной компоненты, то условно ее можно разделить на средства поведенческого описания аппаратуры (параллельные процессы и средства их взаимодействия), средства потокового описания (описание на уровне межрегистровых передач) – параллельные операторы назначения сигнала (<=) с транспортной transport или инерциальной задержкой передачи сигналов и средства структурного описания объектов (операторы конкретизации компонент с заданием карт портов port map и карт настройки generic map, объявление конфигурации и т д)

Параллельные операторы VHDL включают

- оператор процесса process,
- оператор блока block,
- параллельный оператор назначения сигналу <=,

- оператор условного назначения сигналу when,
- оператор селективного назначения сигналу select,
- параллельный оператор утверждения assert,
- параллельный оператор вызова процедуры,
- оператор конкретизации компоненты port map,
- оператор генерации конкретизации generate.

Как видно из этого перечня, последовательные и параллельные операции назначения, вызова процедуры и утверждения различаются контекстно, то есть внутри процессов и процедур они последовательные, внепараллельные

Базовым элементом описания систем на языке VHDL является блок Блок содержит раздел описаний данных и раздел параллельно исполняемых операторов Частным случаем блока является описание архитектуры объекта В рамках описания архитектуры могут использоваться внутренние, вложенные блоки Наряду со всеми преимуществами блочной структуры программы и ее соответствия естественному иерархическому представлению структуры проекта операторы блока языка VHDL позволяют устанавливать условия охраны (запреты) входа в блок Только при истинности значения охранного выражения управление передается в блок и инициирует выполнение операторов его тела

4.2. АЛФАВИТ ЯЗЫКА

Как и любой другой язык программирования, VHDL имеет свой алфавит – набор символов, разрешенных к использованию и воспринимаемых компилятором В алфавит языка входят

1 Латинские строчные и прописные буквы

A, B, . . , Z и a, b, , z

2 Цифры от 0 до 9

3. Символ подчеркивания "_" (код ASCII номер 95)

Из символов, перечисленных в пп 1–3 (и только из них!) могут конструироваться идентификаторы в программе Кроме того, написание идентификаторов должно подчиняться следующим правилам

- идентификатор не может быть зарезервированным словом языка;
- идентификатор должен начинаться с буквы,
- идентификатор не может заканчиваться символом подчеркивания "_",
- идентификатор не может содержать двух последовательных символов подчеркивания "__"

Примеры корректных идентификаторов

cont, clock2, full_add

Примеры некорректных идентификаторов

1clock, _adder, add__sub, entity

Следует отметить, что прописные и строчные буквы не различаются, т е. идентификаторы clock и CLOCK являются эквивалентными

4. Символ "пробел" (код 32), символ табуляции (код 9), символ новой строки (коды 10 и 13)

Данные символы являются разделителями слов в конструкциях языка. Количество разделителей не имеет значения Таким образом, следующие выражения для компилятора будут эквивалентны

```
count.=2+2;
count := 2 + 2  ,
count :=        2
 +
 2;
```

5 Специальные символы, участвующие в построении конструкций языка

+ - * / = < > . , ( ) . , # ' " |

6. Составные символы, воспринимаемые компилятором как один символ.

<= >= => = /=

Разделители между элементами составных символов недопустимы

4.2.1. Комментарии

Признаком комментария являются два символа тире ("--") Компилятор игнорирует текст, начиная с символов "--" до конца строки, т е комментарий может включать в себя символы, не входящие в алфавит языка (в частности, русские буквы)

4.2.2. Числа

В стандарте языка определены числа как целого, так и вещественного типа Однако средства синтеза ПЛИС допускают применение только целых чисел Целое число в VHDL может быть представлено в одной из четырех систем счисления двоичной, десятичной, восьмеричной и шестнадцатеричной Конкретные форматы написания числовых значений будут описаны далее при рассмотрении различных типов языка.

К разновидности числовых значений можно отнести также битовые строки

4.2.3. Символы

Запись символа представляет собой собственно символ, заключенный в одиночные кавычки Например

'A', '*', ' '

В средствах синтеза ПЛИС область применения символов ограничена использованием их в качестве элементов перечислимых типов

4.2.4. Строки

Строки представляют собой набор символов, заключенных в двойные кавычки Чтобы включить двойную кавычку в строку, необходимо ввести две двойные кавычки Например

" A string"

"A string in a string ""A string"" "

4.3. ТИПЫ ДАННЫХ

Подобно высокоуровневым языкам программирования, VHDL является языком со строгой типизацией Каждый тип данных в VHDL имеет определенный набор принимаемых значений и набор допустимых операций В языке предопределено достаточное количество простых и сложных типов, а также имеются средства для образования типов, определяемых пользователем.

Необходимо отметить, что в данном пособии рассматриваются не все типы данных, определенные в стандарте, а только те, которые поддерживаются средствами синтеза ПЛИС

4.3.1. Простые типы

Следующие простые типы являются предопределенными

1. BOOLEAN (логический) — объекты данного типа могут принимать значения FALSE (ложь) и TRUE (истина)
2 INTEGER (целый) — значения данного типа представляют собой 32-разрядные числа со знаком Объекты типа INTEGER могут содержать значения из диапазона $-(2^{31}-1)...2^{31}-1$ (-2147483647 2147483647).
3 BIT (битовый) — представляет один логический бит Объекты данного типа могут содержать значение '0' или '1'

4. STD_LOGIC (битовый) — представляет один бит двнных Объекты данного типа могут принимать 9 состояний Данный тип определен стандартом IEEE 1164 для замены типа BIT.
5 STD_ULOGIC (битовый) – представляет один бит данных Объекты данного типа могут принимать 9 состояний. Данный тип определен стандартом IEEE 1164 для замены типа BIT (см. Примечание).
6. ENUMERATED (перечислимый) – используется для задания пользовательских типов.
7. SEVERITY_LEVEL – перечислимый тип, используется только в операторе ASSERT
8. CHARACTER – символьный тип.

Примечание

В действительности тип STD_ULOGIC является базовым типом для типа STD_LOGIC, т. е объекты обеих типов могут принимать одно и то же множество значений и имеют одинаковый набор допустимых операций. Единственное различие между типами заключается в том, что для типа STD_ULOGIC не определена функция разрешения (resolving function) В языке VHDL функция разрешения используется для определения значения сигнала, имеющего несколько источников (драйаеров).

Пример: Выходы двух буферов с тремя состояниями подключены к одной цепи X. Пусть выход одного буфера установился в состояние 'Z', а выход другого – в состояние '1'. Функция разрешения определяет, что в этом случае значение сигнала X будет равно '1'.

Поскольку для типа STD_ULOGIC не определена функция разрешения, сигналы этого типа могут иметь только один источник

Далее рассматривается только тип STD_LOGIC, однако все сказанное будет справедливо и для типа STD_ULOGIC На практике, в подавляющем большинстве случаев достаточно использования типа STD_LOGIC

4.3.2. Сложные типы

Из всей совокупности сложных типов, определенных в стандарте языка, для синтеза логических схем используются только массивы (тип ARRAY) и записи (тип RECORD) Однако тип RECORD поддерживается не всеми средствами синтеза и в данном пособии рассмотрен не будет.

Следующие типы-массивы являются предопределенными

1. BIT_VECTOR — одномерный массив элементов типа BIT
2. STD_LOGIC_VECTOR — одномерный массив элементов типа STD_LOGIC.
3. STD_ULOGIC_VECTOR — одномерный массив элементов типа STD_ULOGIC
4. STRING — одномерный массив элементов типа CHARACTER

Направление и границы диапазона индексов не содержатся в определении указанных типов и должны быть указаны непосредственно при объявлении объектов данных типов

4.3.3. Описание простых типов

Тип BOOLEAN

Тип BOOLEAN является перечислимым типом Объект данного типа может принимать значения FALSE (ложь) и TRUE (истина), причем FALSE эквивалентно 0, а TRUE эквивалентно 1

Все три типа объектов в VHDL (константы, переменные и сигналы) могут иметь тип BOOLEAN Таким объектам может быть присвоено только значение типа BOOLEAN

Пример

```
PROCESS (a, b)
 VARIABLE cond : BOOLEAN;
BEGIN
 cond := a > b;
 IF cond THEN
    output <= '1';
 ELSE
    output <= '0';
END IF;
END PROCESS;
```

Операторы отношения

Значения типа BOOLEAN могут участвовать в выражениях Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа BOOLEAN и одномерных массивов, содержащих элементы типа BOOLEAN. Результат выражения также имеет тип BOOLEAN (Как для всех перечислимых типов, операции сравнения над одномерными массивами типа BOOLEAN производятся поэлементно, начиная с крайнего левого элемента)

Логические операторы

Для операндов типа BOOLEAN и одномерных массивов, содержащих элементы типа BOOLEAN, определены все логические операции (AND, OR, NAND, NOR, XOR и NOT) Тип и размер операндов должны быть одинаковыми Тип и размер результата такой же, как тип и размер операндов

Оператор конкатенации

Оператор конкатенации также определен для операндов типа BOOLEAN и одномерных массивов, содержащих элементы типа BOOLEAN Результат выражения представляет собой одномерный массив, содержащий элементы типа BOOLEAN; размер массива равен сумме размеров операндов

Другие операторы

Другие операции над операндами типа BOOLEAN не определены

Тип INTEGER

Стандарт VHDL определяет тип INTEGER для использования в арифметических выражениях По умолчанию объекты типа INTEGER имеют размерность 32 бита и представляют целое число в интервале $-(2^{31}-1)$ $2^{31}-1$ (–2147483647 2147483647) Стандарт языка позволяет также объявлять объекты типа INTEGER, имеющие размер меньше 32 бит, используя ключевое слово RANGE, ограничивающее диапазон возможных значений

SIGNAL X INTEGER RANGE –127 TO 127

Данная конструкция определяет X как 8-битное число.

Кроме того, можно определить ограниченный целый тип, используя следующую конструкцию

TYPE *имя_типа* **IS RANGE** *диапазон_индексов*;

диапазон_индексов определяется следующим образом.

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*

Пример

```
TYPE byte_int 0 TO 255;
TYPE signed_word_int is range -32768 TO 32768,
TYPE bit_index is range 31 DOWNTO 0;
```

Значения типа INTGER записываются в следующей форме:

[основание #] разряд { [_] разряд} [#]

По умолчанию "основание" принимается равным 10. Допустимыми также являются значения 2, 8, 16.

При записи числа допускается использование одиночных символов подчеркивания, которые не влияют на результирующее значение.

Пример

```
CONSTANT min : INTEGER := 0;
CONSTANT group : INTEGER := 13_452; - - эквиввлентно 13452
CONSTANT max : INTEGER := 16#FF#;
```

Допустимое использование

Операторы отношения

Значения типа INTEGER могут участвовать в этих выражениях Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа INTEGER и одномерных массивов, содержащих элементы типа INTEGER. Результат выражения имеет тип BOOLEAN.

Арифметические операторы

Операторы +, –, ABS допустимы для операндов типа INTEGER. Результат выражения имеет тип INTEGER.

Операторы *, /, MOD, REM допустимы в следующих случаях:

- если оба операнда являются константами (CONSTANT),
- если второй операнд является константой и его значение равно 2^n, где $n = 0, 1, 2, 3...$

Применение операторов *, /, MOD, REM недопустимо, если оба операнда являются сигнвлами (SIGNAL) или переменными (VARIABLE)

Оператор возведения в степень (**), как правило, не поддерживается средствами синтеза

Другие операторы

Другие операции над операндами типа INTEGER не определены

Тип BIT

Объект данного типа может принимать значение '0' (лог. 0) или '1' (лог. 1).

Примечание

Стандартом IEEE 1164 определена замена типа BIT на более гибкий тип STD_LOGIC Поэтому использование типа BIT в новых разработках не рекомендуется

Допустимое использование

Операторы отношения

Значения типа BIT могут участвовать в выражениях Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа BIT и одномерных массивов, содержащих элементы типа BIT. Результат выражения имеет тип BOOLEAN. (Как для всех перечислимых типов, операции сравнения над одномерными массивами типа BIT производятся поэлементно, начиная с крайнего левого элемента)

Логические операторы

Для операндов типа BIT и одномерных массивов, содержащих элементы типа BIT, определены все логические операции (AND, OR, NAND, NOR, XOR и NOT). Тип и размер операндов должны быть одинаковыми. Тип и размер результата такой же, как тип и размер операндов.

Оператор конкатенации

Оператор конкатенации также определен для операндов типа BIT и одномерных массивов, содержащих элементы типа BIT Результат выражения представляет собой одномерный массив, содержащий элементы типа BIT, размер массива равен сумме размеров операндов

Другие операторы

Другие операции над операндами типа BIT не определены

Тип STD_LOGIC

Типы STD_LOGIC является перечислимым типом. Объекты типа STD_LOGIC могут принимать 9 значений: '0', '1', 'Z', '–', 'L', 'H', 'U', 'X', 'W'.

Для синтеза логических схем используются только первые четыре:

'0' – логический "0";
'1' – логическая "1";
'Z' – третье состояние;
'–' – не подключен.

Допустимое использование

Операторы отношения

Значения типа STD_LOGIC могут участвовать в выражениях Операторы отношения (=, /=, <, <=, >, >=) определены для операндов типа STD_LOGIC и одномерных массивов, содержащих элементы типа STD_LOGIC Результат выражения имеет тип BOOLEAN. (Как для всех перечислимых типов, операции сравнения над одномерными массивами типа STD_LOGIC производятся поэлементно, начиная с крайнего левого элемента)

Логические операторы

Для операндов типа STD_LOGIC и одномерных массивов, содержащих элементы типа STD_LOGIC, определены все логические операции (AND, OR, NAND, NOR, XOR и NOT). Тип и размер операндов должны быть одинаковыми. Тип и размер результата такой же, как тип и размер операндов.

Оператор конкатенации

Оператор конкатенации также определен для операндов типа STD_LOGIC и одномерных массивов, содержащих элементы типа STD_LOGIC Результат выражения представляет собой одномерный массив, содержащий элементы типа STD_LOGIC, размер массива равен сумме размеров операндов

Другие операторы

Другие операции над операндами типа STD_LOGIC не определены

Перечислимый тип

Перечислимый тип — это такой тип данных, при котором количество всех возможных значений конечно Строго говоря, все описанные выше типы являются перечислимыми.

Применение перечислимых типов преследует две цели

- улучшение смысловой читаемости программы,
- более четкий и простой визувльный контроль значений

Наиболее часто перечислимый тип используется для обозначения состояний конечных автоматов

Перечислимый тип объявляется путем перечисления названий элементов-значений Объекты, тип которых объявлен как перечислимый, могут содержать только те значения, которые указаны при перечислении.

Элементы перечислимого типа должны быть идентификаторами или символами, которые должны быть уникальными в пределах одного типа Повторное использование названий элементов в других перечислимых типах разрешается.

Объявление перечислимого типа имеет вид:

TYPE *имя_типа* **IS (** *название_элемента* [**,** *название_элемента*] **);**

Пример

```
Type State_type IS (stateA, stateB, stateC);
VARIABLE State : State_type;
.
.
.
State := stateB
```

В данном примере объявляется переменная *State*, допустимыми значениями которой являются *stateA*, *stateB*, *stateC*.

Примеры предопределенных перечислимых типов:

```
TYPE SEVERITY_LEVEL IS (NOTE, WARNING, ERROR, FAILURE);
TYPE BOOLEAN IS (FALSE, TRUE),
TYPE BIT IS ('0', '1');
TYPE STD_LOGIC IS ('U', 'X', '0', '1', 'Z', 'W',
'L', 'H', '-'),
```

Любой перечислимый тип имеет внутреннюю нумерацию первый элемент всегда имеет номер 0, второй — номер 1 и т. д Порядок нумерации соответствует порядку перечисления

Допустимое использование

Операторы отношения

Значения определенных пользователем перечислимых типов могут участвовать в выражениях Операторы отношения (=, /=, <, <=, >, >=) определены как для перечислимых типов, так и для одномерных массивов, содержащих элементы этих типов Результат выражения имеет тип BOOLEAN

Оператор конкатенации

Оператор конкатенации определен для операндов, имеющих перечислимый тип, и одномерных массивов, содержащих элементы перечислимого типа При этом оба операнда должны быть одного типа Результат выражения представляет собой одномерный массив, тип элементов которого равен типу операндов, размер массива равен сумме размеров операндов

Другие операторы

К операндам перечислимых типов применим оператор указания типа Данный оператор используется для уточнения типа объекта в случае, если одно и то же название элемента используется различными типами

Пример

```
TYPE COLOR IS (green, red, tellow, orange);
TYPE FRUIT IS (orange, apple, pear);
VARIABLE VC . COLOR,
VARIABLE VF · FRUIT,

VC  = COLOR'orange,
VF .= FRUIT'orange,
```

Другие операции над операндами перечислимых типов, определенных пользователем, не определены

Тип SEVERITY_LEVEL

Переменные этого типа используются только в операторе ASSERT и игнорируются при синтезе логических схем

Переменные типа SEVERITY_LEVEL могут принимать следующие значения NOTE, WARNING, ERROR и FAILURE

Тип CHARACTER

Перечислимый тип Значением объекта данного типа может быть любой символ из набора ASCII (128 первых символов).

Массивы

Массив (тип "массив") является сложным типом Массив представляет собой упорядоченную структуру однотипных данных Массив имеет диапазон индексов, который может быть возрастающим либо убывающим На любой элемент массива можно сослаться используя его индекс Несмотря на то, что стандартом языка допускается использование массивов любой размерности, для синтеза ПЛИС используются только одномерные (поддерживаются всеми средствами синтеза) и двумерные (поддерживаются ограниченным числом средств синтеза) массивы

Также можно сослаться на часть одномерного массива, используя вместо индекса диапазон индексов

Существуют две разновидности типа "массив" ограниченный (*constrained*) и неограниченный (*unconstrained*)

Объявление ограничечного типа определяет границы диапазона индексов (число элементов массива) в каждом измерении при определении типа

Объявление неограниченного типа не определяет границы диапазона индексов В этом случае границы диапазона устанавливаются при объявлении конкретного экземпляра объекта данного типа

Объявление ограниченного типа "массив" имеет вид

TYPE *имя_типа* **IS**
  ARRAY (*диапазон_индексов* [, *диапазон_индексов*])
    OF *тип_элемента*;

диапазон_индексов может определяться двумя способами

1) явным заданием границ диапазона

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*

2) с использованием идентификатора ограниченного подтипа В этом случае значения границ подтипа являются значениями границ индекса массива Описание подтипов см далее

Объявление неограниченного типа "массив" имеет вид

TYPE *имя_типа* **IS**
  ARRAY (*тип_индекса*
    [, *тип_индекса*])
    OF *тип_элемента*;

тип_индекса определяется следующим образом

подтип **RANGE <>**

где *подтип* может быть

INTEGER – индекс находится в диапазоне $-(2^{31}-1)$ $2^{31}-1$,

NATURAL – индекс находится в диапазоне 0 $2^{31}-1$,

POSITIVE – индекс находится в диапазоне 1 $2^{31}-1$,

Примеры

1) Объявление ограниченного массивного типа

TYPE word IS ARRAY (31 DOWNTO 0) OF STD_LOGIC,

TYPE register_bank IS ARRAY (byte RANGE 0 TO 132) OF INTEGER,

TYPE memory IS ARRAY (address) OF word, — двумерный массив

2) Объявление неограниченного массивного типа

TYPE logic IS ARRAY (INTEGER RANGE <>) OF BOOLEAN,

3) Объявление двумерного массива

TYPE Reg IS ARRAY (3 DOWNTO 0) OF STD_LOGIC_VECTOR(7 DOWNTO 0),

TYPE transform IS ARRAY (1 TO 4, 1 TO 4) OF BIT,

Как было сказано, в языке имеется несколько предопределенных типов "массив" Их объявления выглядят следующим образом

TYPE STRING IS ARRAY (POSITIVE RANGE <>) OF CHARACTER,

TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE <>) OF BIT,

TYPE STD_LOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF STD_LOGIC,

TYPE STD_ULOGIC_VECTOR IS ARRAY (NATURAL RANGE <>) OF STD_ULOGIC,

Объявление объекта типа "неограниченный массив" должно содержать ограничения на индекс Диапазон изменения индексов может быть ограничен:

1) с использованием ключевых слов TO или DOWNTO:

```
TYPE data_memory_type IS ARRAY (INTEGER RANGE <>)
OF BIT;
. . .
VARIABLE data_ memory : data_memory_type(0 TO 255);
```

2) путем использования диапазона начального значения.

```
CONSTANT part_id : STRING :="M38006".
```

СТРОКИ, БИТОВЫЕ СТРОКИ И АГРЕГАТЫ

Строки, битовые строки, агрегаты (*strings*, *bit strings*, *aggregates*) используются для конструирования значений объектов массивных типов Они могут использоваться в любом месте, где допускается значение типа массив, например, как начальное значение константы или операнд в выражении.

Строковая запись может быть использована для представления значений как объектов некоторых предопределенных типов (*string*, *bit_vector*, *std_logic_vector*), так и для любого одномерного массива, элементы которого имеют тип *character*, например:

```
TYPE bit6 IS ('U', '0', '1', 'F', 'R', 'X');. . . .
TYPE bit6_data IS ARRAY (POSITIVE RANGE<>) OF bit6;
. . .
SIGNAL data_bus : bit6_data(15 DOWNTO 0);
.
.
data_bus <= "UUUUUUUUFFFFFFFF";
```

VHDL позволяет компактно описывать битовые строки (значение типа *bit_vector* или *std_logic_vector*) в базисе 2, 8 и 16. Например

```
CONSTANT clear : bit_vector .= B"00_101_010";
CONSTANT empty : bit_vector := O"052";
CONSTANT null  : bit_vector .= X"2A";
```

Все три константы имеют одно и то же значение Отметим, что символы подчеркивания могут использоваться в любом месте битовой строки для облегчения чтения Расширенными цифрами (extended digits) для шестнадцатеричного представления являются буквы от A до F, причем могут использоваться как прописные, так и строчные буквы

Массивные агрегаты используются для присваивания объектам типа массив значений Массивные агрегаты формируются при помощи позиционной (positional) записи, поименованной (named) записи или комбинации этих двух форм Рассмотрим пример

Предположим, что имеются следующие описания

```
TYPE ArayType IS ARRAY (1 TO 4) OF CHARACTER;
. . .
VARIABLE Test : ArrayType;
```

и требуется, чтобы переменная *Test* содержала элементы 'f', 'o', 'o', 'd' в указанном порядке

Позиционная запись имеет вид

```
Test .= ('f', 'o', 'o', 'd');
```

Агрегат в данном случае записывается как список значений элементов, разделенных запятыми Первое значение назначается элементу с наименьшим значением индекса (крайнему левому)

Поименованная запись имеет вид

```
Test := (1=>'f', 3=>'o', 4=>'d', 2=>'o');
```

В этом случае агрегат также является списком, элементы которого разделены запятыми, однако элементы списка имеют формат:

позиция => значение

Порядок перечисления элементов при поименованной записи не имеет значения.

Комбинированная запись имеет вид

Test := ('f', 'o', 4=>'d', 3=>'o');

В этом случае сначала записываются элементы, присваиваемые с использованием позиционной записи, а оставшиеся элементы присваиваются с использованием поименованной записи.

При формировании агрегата с использованием поименованной (или комбинированной) записи вместо номера позиции можно указывать ключевое слово **OTHERS**, которое определяет значение для всех элементов, которые еще не были описаны в агрегате. Например

```
Test := ('f', 4=>'d', OTHERS=>'0');
```

Подтипы

Использование подтипов позволяет объявлять объекты, принимающие ограниченный набор значений из диапазона, допустимого для базового типа.

Подтипы применяются в двух случаях

1) Подтип может ограничить диапазон значений базового скалярного типа (ограничение по диапазону) В этом случае объявление подтипа выглядит следующим образом·

SUBTYPE *имя_подтипа* **IS** *имя_базового_типа* **RANGE** *диапазон_индексов*;

диапазон_индексов определяется следующим образом

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*.

Пример

Предположим, что разработчик желает создать сигнал *A* типа *severity* и что *A* может принимать значения только *OKAY*, *NOTE* и *WARNING*

TYPE sevirity IS (OKAY, NOTE, WARNING, ERROR, FAILURE),

SUBTYPE go_status IS severity RANGE OKAY TO WARNING,

SIGNAL A · go_status,

Базовый тип и ограничение диапазона могут быть включены непосредственно в объявление объекта Объявление сигнала A, эквивалентное приведенному выше будет выглядеть следующим образом

SIGNAL A severity RANGE OKAY TO WARNING,

Другие примеры

SUBTYPE pin_count IS INTEGER RANGE 0 TO 400;

SUBYUPE digits IS character range '0' TO '9',

Подтипы, объявленные таким образом, могут также участвовать в описании ограниченных массивных типов (см *"Массивы"*)

2) Подтип может определить границы диапазона индексов для неограниченного (*unconstrained*) массивного типа В этом случае объявление подтипа выглядит следующим образом

SUBTYPE *имя_подтипа* **IS** *имя_базового_типа*
(диапазон_индексов [, *диапазон_индексов*]);

диапазон_индексов определяется следующим образом

m **TO** *n*

n **DOWNTO** *m*

где *m*, *n* – целочисленные константы, *m* <= *n*

Такое использование подтипа может быть удобно при наличии большого числа объектов некоторого типа с одинаковыми ограничениями на индексы.

Пример

```
TYPE bit6_data IS ARRAY (POSITIVE RANGE <>) OF bit6;
SUBTYPE data_store IS bit6_data (63 DOWNTO 0);
SIGNAL A_reg, B_reg,C_reg : data_store;
VARIABLE temp : data_store;
```

В языке имеются два предопределенных числовых подтипа *natural* и *positive*, которые определены как:

```
SUBTYPE NATURAL IS INTEGER RANGE 0 TO highest_
integer;
SUBTYPE POSITIVE IS INTEGER RANGE 1 TO highest_
integer;
```

4.4. ОПЕРАТОРЫ VHDL

4.4.1. Основы синтаксиса

Исходный текст программы на языке VHDL состоит из последовательностей операторов, записанных с учетом следующих правил.

- каждый оператор — это последовательность слов, содержащих буквы английского алфавита, цифры и знаки пунктуации,
- слова разделяются произвольным количеством пробелов, табуляций и переводов строки,
- операторы разделяются символами ",",
- в некоторых операторах могут встречаться списки объектов, разделяемые символами "," или ","

Комментарии могут быть включены в текст программы с помощью двух подряд идущих символов "—". После появления этих символов весь текст до конца строки считается комментарием

Для указания системы счисления для констант могут быть применены спецификаторы:

B – двоичная система счисления, например B"0011",

O – восьмеричная система счисления, например O"3760";

H – шестнадцатеричная система счисления, например H"F6A0".

4.4.2. Объекты

Объекты являются контейнерами для хранения различных значений в рамках модели Идентификаторы объектов могут содержать буквы, цифры и знаки подчеркивания Идентификаторы должны начинаться с буквы, не должны заканчиваться знаком подчеркивания, и знаки подчеркивания не могут идти подряд Прописные и строчные буквы в VHDL не различаются Все объекты должны быть явно объявлены перед использованием, за исключением переменной цикла в операторе ***for***, которая объявляется по умолчанию

Каждый объект характеризуется типом и классом Типы разделяются на предопределенные в языке VHDL и определяемые пользователем Тип показывает, какого рода данные может содержать объект Класс показывает, что можно сделать с данными, содержащимися в объекте. В языке VHDL определены следующие классы объектов.

- ***Constant*** — константы. Значение константы определяется при ее объявлении и не может быть изменено Константы могут иметь любой из поддерживаемых типов данных
- ***Variable*** — переменные. Значение, хранимое в переменной, меняется везде, где встречается присваивание данной переменной. Переменные могут иметь любой из поддерживаемых типов данных
- ***Signal*** — сигналы. Сигналы представляют значения, передаваемые по проводам и определяемые присвоением сигналов (отличным от присвоения переменных). Сигналы могут иметь ограниченный набор типов (обычно *bit*, *bit_vector*, *std_logic*, *std_logic_vector*, *integer*, и, возможно, другие, в зависимости от среды разработки).

Повторное использование присваивания сигналов в наборе параллельных операторов не допускается. В наборе последовательных операторов такое присваивание допустимо и даст значение сигнала, соответствующее последнему по порядку присваиванию

Синтаксис объявления объектов

```
Constant ( name [, name] ) : Type [ ( index_range [
, index_range ] ) ] := initial_value;
Variable ( name [, name] ) : Type [ ( index_range [
, index_range ] ) ] [ := initial_value ];
Signal ( name [, name] ) : Type [ ( index_range ) ];
```

Диапазон значений индексов задается в виде *int_value* **to** *int_value* или *int_value* **downto** *int_value*

4.4.3. Атрибуты

Атрибуты (или иначе свойства) определяют характеристики объектов, к которым они относятся Стандарт VHDL предусматривает как предопределенные, так и определяемые пользователем атрибуты, однако современные инструментальные средства в большинстве своем поддерживают только предопределенные атрибуты Для обращения к атрибутам объекта используется символ << ' >>(например *A1'left*)

В VHDL определены следующие атрибуты

'left – левая граница диапазона индексов массива,

'right – правая граница диапазона индексов массива,

'low – нижняя граница диапазона индексов массива,

'high – верхняя граница диапазона индексов массива,

'range – диапазон индексов массива,

'reverse_range – обращенный диапазон индексов массива,

'length – ширина диапазона индексов массива

4.4.4. Компоненты

Объявление компонента определяет интерфейс к модели на VHDL (***entity*** и ***architecture***), описанной в другом файле Обычно объявление компонента совпадает с соответствующим объявлением ***entity*** Они могут различаться только значениями по умолчанию Эти значения используются, когда какой-либо из выводов компонента остается не присоединенным (ключевое слово ***open***) при установке компонента в схему

Оператор объявления компонента может находиться внутри объявления ***architecture*** или в заголовке пакета (***package***) Соответствующие компоненту объявления ***entity*** и ***architecture*** не обязательно должны существовать в момент анализа схемы В момент моделирования или синтеза должны существовать объявления entity и architecture для компонентов, которые не только объявлены, но и установлены в схему Это позволяет, например, конструктору задать объявления библиотечных элементов, а реальное их описание (объявления entity и architecture) задавать по мере использования этих элементов в конструкции.

Объяаление компонента записывается следующим образом

```
Component name
 [ port( port_list ); ]
end component;
```

4.4.5. Выражения

Выражения могут содержать следующие операторы: преобразование типа, ***and***, ***or***, ***nand***, ***nor***, ***xor***, =, /=, <, <=, >, >=, +, -, &, *, /, ***mod***, ***rem***, ***abs***, ***not***.

В зависимости от избранной САПР при синтезе может поддерживаться подмножество приведенных выше операторов. Порядок вычисления выражений определяется приоритетом операторов:

and, ***or***, ***nand***, ***nor***, ***xor*** – самый низкий приоритет
=, /=, <, <=, >, >=
+, -, & (бинарные)
+, - (унарные)
*, /, ***mod***, ***rem***
abs, ***not*** – высший приоритет

Операторы с более высоким приоритетом выполняются раньше Чтобы изменить такой порядок используются скобки

При моделировании (но не при синтезе) схемы возможно также описание формы сигнала в виде выражения. Записывается оно следующим образом·

```
Value_expression [ after time_expression ]
( , value_expression [ after time_expression ] }
```

4.4.6. Операторы

С помощью операторов описывается алгоритм, определяющий функционирование схемы. Они могут находиться в теле функции, процедуры или процесса.

Wait until *condition;*

Приостанавливает выполнение процесса, содержащего данный оператор до момента выполнения условия.

Signal <= expression

Оператор присваивания сигнала устанавливает его значение равным выражению справа

Variable := expression

Оператор присваивания устанавливает значение переменной равным выражению справа.

Procedure_name (parameter { , parameter })

Оператор вызова процедуры состоит из имени процедуры и списка фактических параметров

```
If condition then
 Sequence_of_statements
( Elsif condition then
 Sequence_of_statements }
[ else
 sequence_of_statements ]
end if ;
```

Оператор ***If*** используется для ветвления алгоритма по различным условиям.

```
Case expression is
 When choices_list => sequence_of_statements;
 ( When choices_list => sequence_of_statements; }
 When others => sequence_of_statements;
End case;
```

Оператор ***casa*** подобно оператору ***If*** задает ветвление алгоритма Значения в списках разделяются символом "|". Когда значение выражения встречается в одном из списков значений, выполняется соответствующая последовательность операторов Если значение выражения не присутствует ни в одном из списков, то выполняется список операторов, соответствующий ветви ***whan others***.

[loop_label :]
for *loop_index_variable* ***in*** *range* ***loop***
sequence_of_statements
and loop *[loop_label] ;*

Оператор цикла позволяет многократно выполнить последовательность операторов. Диапазон значений задается в виде *value1* ***to*** *value2* или *value1* ***downto*** *value2*. Переменная цикла последовательно принимает значения из заданного диапазона. Количество итераций равно количеству значений в диапазоне.

Return *expression ;*

Этот оператор возвращает значение из функции

Null

Пустой оператор, не выполняет никаких действий.

4.5. ИНТЕРФЕЙС И ТЕЛО ОБЪЕКТА

Полное VHDL-описание объекта состоит как минимум из двух отдельных описаний· описание интерфейса объекта и описание тела объекта (описание архитектуры).

Интерфейс описывается в объявлении объекта **entity declaration** и определяет входы и выходы объекта, его входные и выходные порты **ports** и параметры настройки **generic**. Параметры настройки отражают тот факт, что некоторые объекты могут иметь управляющие входы, с помощью которых может производиться настройка объектов, в частности задаваться время задержки

Например, у объекта Q1 три входных порта X1, X2, X3 и два выхода У1, У2 Описание его интерфейса на VHDL имеет вид·

```
Entity Q1 is
      Port (X1, X2, X3: in real; Y1, Y2: out real);
End Q1.
```

Порты объекта характеризуются направлением потока информации Они могут быть:

- входными (in);
- выходными (out);
- двунаправленными (inout),
- двунаправленными буферными (buffer);
- связными (linkage).

А также имеют тип, характеризующий значения поступающих на них сигналов:

- целый (integer),
- вещественный (real);
- битовый (bit);
- символьный (character)

Тело объекта специфицирует его структуру или поведение. Его описание по терминологии VHDL содержится в описании его архитектуры **architecture.**

VHDL позволяет отождествлять с одним и тем же интерфейсом несколько архитектур Это связано с тем, что в процессе проектирования происходит проработка архитектуры объекта переход от структурной схемы к электрической принципиальной, от поведенческого к структурному описанию

Средства VHDL для отображения структур цифровых систем базируются на представлении о том, что описываемый объект **entity** представляет собой структуру из компонент **component,** соединяемых друг с другом линиями связи. Каждая компонента, в свою очередь, является объектом и может состоять из компонент низшего уровня (иерархия объектов) Взаимодействуют объекты путем передачи сигналов **signal** по линиям связи Линии связи подключаются к входным и выходным портам компонент В VHDL сигналы отождествляются с линиями связи

Имена сигналов и имена линий связи совпадают (они отождествляются) Для сигналов (линий), связывающих компоненты друг с другом, необходимо указывать индивидуальные имена

Описание структуры объекта строится как описание связей конкретных компонент, каждая из которых имеет имя, тип и карты портов Карта портов **port map** определяет соответствие портов компонент поступающим на них сигналам, можно интерпретировать карту портов как разъем, на который приходят сигналы и в который вставляется объект-компонента

Принятая в VHDL форма описания связей конкретных компонент имеет следующий вид

```
Имя. тип связи (сигнал, порт).
```

Например, описание связей объекта Q1, представленного на рис 3, выглядит следующим образом

```
K1· SM port map (X1, X2, S),
K3. M port map (S, Y1),
K2: SM port map (S, X3, Y2),
```

Здесь K1, K2, K3 — имена компонент, SM, M — типы компонент, X1, X2, X3, S, Y1,Y2 — имена сигналов, связанных с портами

Полное VHDL описание архитектуры STRUCTURA объекта Q1 имеет вид

```
Architecture STRUCTURA of Q1 is
Component SM port (A, B  in real; C. out real),
End component;
Component M port (E: in real, D: out real);
End component,
Signal S. real;
Begin
K1. SM port map (X1, X2, S),
K3  M port map (S, Y1),
K2: SM port map (S, X3, Y2),
End  STRUCTURA;
```

Средства VHDL для отображения поведения описываемых архитектур строятся на представлении их как совокупности параллельно взаимодействующих процессов Понятие процесса **process** относится к базовым понятиям языка VHDL

Архитектура включает в себя описание одного или нескольких параллельных процессов Описание процесса состоит из последовательности операторов, отображающих действия по переработке информации Все операторы внутри процесса выполняются последовательно. Процесс может находиться в одном из двух состояний — либо пассивном, когда процесс ожидает прихода сигналов запуска или наступления соответствующего момента времени, либо активном — когда процесс исполняется

Процессы взаимодействуют путем обмена сигналами

В общем случае в поведенческом описании состав процессов не обязательно соответствует составу компонент, как это имеет место в структурном описании

Поведение VHDL- объектов воспроизводится на ЭВМ, и приходится учитывать особенности воспроизведения параллельных процессов на однопроцессорной ЭВМ Особая роль в синхронизации процессов отводится механизму событийного воспроизведения модельного времени **now.**

Когда процесс вырабатывает новое значение сигнала перед его посылкой на линию связи, говорят, что он вырабатывает будущее сообщение **transaction** С каждой линией связи (сигналом) может быть связано множество будущих сообщений. Множество сообщений для сигнала называется его драйвером **driver**

Таким образом, драйвер сигнала — это множество пар время – значение (множество планируемых событий)

VHDL реализует механизм воспроизведения модельного времени, состоящий из циклов На первой стадии цикла вырабатываются новые значения сигналов На второй стадии процессы реагируют на изменения сигналов и переходят в активную фазу Эта стадия завершается, когда все процессы перейдут снова в состояние ожидания. После этого модельное время становится равным времени ближвйшего запланированного события, и все повторяется

Особый случай представляет ситуация, когда в процессах отсутствуют операторы задержки Для этого в VHDL предусмотрен механизм так называемой дельта-задержки

В случае дельта-задержек новый цикл моделирования не связан с увеличением модельного времени В приведенном выше примере новое значение сигнала У1 вырабатывается через дельта-задержку после изменения сигнала S

Другая способность VHDL- процессов связана с так называемыми разрешенными **resolved** сигналами Если несколько процессов изменяют один и тот же сигнал, (сигнал имеет несколько драйверов), в описании объектов может указываться функция разрешения Эта функция объединяет значения из разных драйверов и вырабатывает одно Это позволяет, например, учесть особенности работы нескольких элементов на общую шину

В языке VHDL для наиболее часто используемых видов процессов — процессов межрегистровых передач — введена компактная форма записи

Полное описание архитектуры POVEDENIE объекта Q1 в этом случае имеет следующий вид

```
Architecture POVEDENIE of Q1 is
Signal S. real,
Begin
Y1<=S;
Y2<=S+X3 after 10 ns;
S<=X3+X2 after 10 ns,
End POVEDENIE,
```

4.5.1. Описание простого объекта

Для иллюстрации возможностей VHDL рассмотрим пример проектирования простой комбинационной схемы, назовем ее объект F Объект проекта F имеет два входа — A1 и A2 и два выхода — B1 и B2

4.5.2. Объявление объекта проекта F

```
Entity F is
Port (A1, A2: in BIT, B1, B2: out BIT)
```

Сигналы принимают значения 1 или 0 в соответствии с таблицей истинности

Входы		Выходы	
A1	A2	B1	B2
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

4.5.3. Поведенческое описание архитектуры

Вариант описания архитектуры BEHAVIOR объекта F использует условный оператор **If** языка VHDL и учитывает, что только при обоих входах A1 и A2, равных 1, выходы B1 = 1 и B2 = 0 В остальных случаях наоборот — B1 = 0 и B2 = 1.

```
Architecture BEHAVIOR of  F is
Begin
Process
Begin
Wait on (A1, A2)
If  (A1='1') and (A2='1')
Then B1<='1', B2<='0',
End if,
End process;
End;
```

В каждом процессе может быть только 1 оператор **wait on.** Второй вариант поведенческого описания архитектуры объекта F, назовем его BEHAVIOR_F, использует выбор **case** языка VHDL и учитывает то свойство функции F, что для первых трех строк ее значение не меняется В заголовке процесса указан список чувствительности процесса **process (A1, A2).** Это указание эквивалентно оператору **wait on (A1, A2)** в начале описания процесса

```
Architecture BEHAVIOR_F of F is
Begin
Process (A1,A2);
Begin
--&-операция
case  (A1& A2) is
--первые три строки таблицы
when "00"/ "01"/ "10"=> B1<='0', B2<='1'
--последняя строка таблицы
when "11" => B1<='1'; B2<='0'
end case
end process
end BEHAVIOR_F,
```

4.5.4. Потоковая форма

В процессе проектирования объекта F могут быть предложены различные варианты его функциональных схем.

Описание архитектуры объекта F может быть таким

```
Architecture F_A of F is
Begin
--каждому вентилю сопоставлен оператор назначения
сигнала
B1<= A1 and A2,
B2<= not (A1 and A2),
End,
```

Здесь каждому элементу сопоставлен процесс, отображающий последовательность преобразования входной информации и передачи ее на выход Процесс представлен в форме оператора параллельного назначения сигнала. Операторы назначения сигнала (<=) срабатывают параллельно при изменении хотя бы одного из сигналов в своих правых частях

Другой вариант описания архитектуры F_B. Здесь вентили включены последовательно

```
Architecture F_B of F is
Signal X: bit
Begin
B2<= not (X),
X <= A1 and A2,
B1 <= X;
End;
```

Промежуточный сигнал X введен в описание архитектуры F_B объекта F потому, что в описании интерфейса объекта F порт B1 объявлен выходным, то есть с него нельзя считывать сигнал и запись B2<= not(B1) была бы не корректной

Сигнал B2 вырабатывается только после изменения сигнала X Оператор B2<= not(X) сработает только тогда, когда изменится сигнал X, то есть после оператора X<= A1 and A2, т к он реагирует только на изменение сигнала в своей правой части С учетом задержки E1=10 нс , а E2=5нс описание архитектуры будет иметь вид

```
Architecture F_B_TIME of F is
Signal X. bit
Begin
--задержка на B1- 10 нс
--задержка на B2- 5 нс
B1<=X,
B2<= not (X) after 5 ns,
X<= A1 and A2 after 10 ns,
End;
```

Через 10 нс после изменения одного из входных сигналов (A1 или A2) может измениться выходной сигнал B1, и с задержкой 5 нс после него изменится B2

4.5.5. Структурное описание архитектуры

Описание архитектуры представляет собой структуру объекта как композицию компонент, соединенных между собой и обменивающихся сигналами Функции, реализуемые компонентами в яв-

ном виде, в отличие от предыдущих примеров в структурном описании не указываются Структурное описание включает описание интерфейсов компонент, из которых состоит схема их связей. Полные описания объектов-компонент (интерфейс + архитектура) должны быть ранее помещены в проектируемую библиотеку, подключенную к структурному описанию архитектуры

```
Library work;
 use work all
--подключение рабочей библиотеки work, содержащей
описание объекта соответствующего компоненте INE2
Architecture CXEM_F_C of F is
--ниже интерфейсы компоненты INE2
Component INE2
Port (X1, X2: in bit; Y: out bit);
End component;
--ниже описание связей экземпляров компонент
Signal X. bit;
Begin
E1: INE2 port map (A1, A2, X);
E2· INE2 port map (X, X, B1);
B2<= X;
End,
```

В описании архитектуры CXEM_F_C объекта F сначала указан интерфейс компонент, из которых строится схема Это компоненты типа INE2 с двумя входными и одним выходным портом Затем после *begin* идут операторы конкретизации компонент Для каждого экземпляра компоненты следуют ее имя или карта портов, указывающая соответствие портов экземпляра компоненты поступающим на них сигналам Например, для компоненты по имени E1 типа INE2 на порт X1 подан сигнал A1, на порт X2 — сигнал A2 Порядок конкретизации безразличен, так как это параллельные операторы Для того чтобы описание F было полным, в данном случае в рабочей библиотеке проекта work необходимо иметь описание интерфейса и архитектуры некоторого объекта, сопоставляемого компоненте INE2 Обозначим этот объект в библиотеке как LA3 Его описание

```
Entity LA3 is
Port (X, Y· in bit; Z: out bit);
End LA3,
Architecture DF_LA3 of LA3 is
Begin
Z<= not (X and Y) after 10ns;
End,
```

У объекта LA3 может быть насколько архитектур В примерах дан вариант потокового описания архитектуры DF_LA3 объекта LA3, который содержит оператор назначения сигналу Z инверсного значения конъюнкции сигналов X и Y с задержкой 10 нс

4.6. ОПИСАНИЕ КОНФИГУРАЦИИ

Конфигурацию можно рассматривать как аналог списка компонентов для проекта Оператор конфигурации (идентифицируемый ключевым словом "configuration") определяет, какие реализации должны быть использованы, и позволяет изменять связи компонентов в вашем проекте во время моделирования и синтеза

Конфигурации не являются обязательными, независимо от того, насколько сложен описываемый проект При отсутствии конфигурации, стандарт VHDL определяет набор правил, который обеспечивает конфигурацию по умолчанию; например, в случае, когда предусмотрено более одной реализации для блока, последняя скомпилированная реализация получит приоритет и будет связана с объектом

Обозначение типа компоненты в описании архитектуры CXEM_F_C объекта F и обозначение соответствующего объекта проекта в библиотеке могут не соответствовать друг другу Связывание обозначений осуществляется в форме объявления конфигурации Для того чтобы задать информацию о том, что использованная при описании архитектуры CXEM_F_C объекта F компонента INE2 соответствует библиотечному объекту LA3 и варианту его архитектуры под названием DF_LA3, необходимо объявить конфигурацию **configuration** Конфигурация V1 указывает, что из рабочей библиотеки проекта **library work** для архитектуры CXEM_F_C объекта F для компонент с именами E1 и E2 типа INE2 следует использовать архитектуру DF_LA3 объекта LA3

```
Library WORK
--подключается рабочая библиотека проекта
 configuration V1 of F is
--конфигурация по имени V1 объекта F
use WORK. all;
 --используются все (all) компоненты библиотеки WORK
for CXEM_F_C
--для архитектуры CXEM_F_C
--компоненты E1, E2 соответствуют объекту LA3 с архи-
тектурой DF_LA3 из библиотеки WORK
for E1,E2. INE2
use entity LA3 (DF_LA3);
end for,
end for;
end V1;
```

4.7. ВЕКТОРНЫЕ СИГНАЛЫ И РЕГУЛЯРНЫЕ СТРУКТУРЫ

Одним из средств повышения компактности описаний цифровых устройств является использование векторных представлений сигналов и операций над ними Например, пусть некоторый объект FV выполняет ту же функцию, что и объект F, но над 20-разрядными двоичными векторами AV1 и AV2

Его описание определяет порты как двоичные векторы:

```
Entity FV is
Port (AV1, AV2: in bit_vector (1 to 20);
  BV1, BV2. out bit_vector (1 to 20));
End FV1;
```

Поведенческое описание архитектуры FV в потоковой форме использует операции над битовыми векторами.

```
Architecture BECHAV_FV of FV is
Begin
BV2<= not (AV1 and AV2);
BV1<= AV1 and AV2;
End BECHAV_FV;
```

Структурное описание архитектуры FV для варианта реализации объекта FV как совокупности объектов F, представленное ниже, выполнено с использованием оператора генерации конкретизации Это позволяет повысить компактность описаний регулярных фрагментов схем.

```
Architecture STRUCT_FV of FV is
Component F port (X1, X2. in bit; Y1, Y2: out bit),
End component;
Begin
—первая компонента конкретизирована обычным способом с использованием позиционного соответствия сигналов портам
K1: F port map (AV1 (1), AV2 (1), BV1 (1), BV2 (1));
—вторая компонента конкретизирована с использованием ключевого способа указания соответствия сигналов ее портам
K2: F port map (AV1 ( 2)=>X1, BV1 ( 2)=>Y1, AV2 (2)=>X2, BV2 (2) =>Y2 );
—компоненты K3 - K20 конкретизированы с использованием оператора генерации, позволяющего компактно описывать регулярные фрагменты схем
for I in 3 to 20 generate
K( I ): F port map ( AV1 ( 1 ), AV2 (1), BV1 (1), BV2 (1) );
End generate;
End STRUCT_FV,
```

4.8. ЗАДЕРЖКИ СИГНАЛОВ И ПАРАМЕТРЫ НАСТРОЙКИ

Объект с задержкой можно представить как бы состоящим из двух — идеального элемента и элемента задержки

В языке VHDL встроены две модели задержек — инерциальная и транспортная

Инерциальная модель предполагает, что элемент не реагирует на сигналы, длительность которых меньше порога, равного времени задержки элемента Транспортная модель лишена этого ограничения

Инерциальная модель по умолчанию встроена в оператор назначения сигнала языка VHDL Например, оператор назначения *Y<=X1 and X2 after 10 ns,* описывает работу вентиля 2И и соответствует инерциальной модели Указание на использование транспортной модели обеспечивается ключевым словом **transport** в правой части оператора назначения Например, оператор *YT<=transport X1 and X2 after 10 ns,* отображает транспортную модель задержки вентиля

Задержка может быть задана не константой, а выражением, значение которого может конкретизироваться для каждого экземпляра объекта, используемого как компонента Для этого ее следует задать как параметр настройки в описании интерфейса объекта

Приведенное ниже описание объекта 12 включает описание интерфейса и тела 12 с инерциальной задержкой, заданной как параметр настройки

```
Entity 12 is
—параметр настройки T по умолчанию равен 10 нс
Generic (T· time = 10 ns),
Port ( X1, X2: in bit; Y: out bit ),
End 12;
Architecture A1_inert of 12 is
Begin
Y<= X1 and X2 after T,
End A1_inert;
Architecture A1_transport of 12 is
Begin
Y<= transport X1and X2 after 10 ns;
End;
```

Ниже представлен вариант описания архитектуры, иллюстрирующей возможность использования параметра настройки (задержка E1 равна 5 нс, E2 — 20 нс) и возможность совмещения структурного и поведенческого описаний в одной архитектуре

```
Architecture MIX_8_a of F is
Component 12
Generic (T  time),
Port (X1, X2  in bit, Y  out bit );
End component,
Begin
E1  12 generic map (5 ns );
 Port map ( A1, A2, B1),
E2. B2<= not ( A1 and A2 ) after 20 ns;
End,
```

Более сложной представляется ситуация, когда необходимо отобразить в описании архитектуры объекта тот факт, что задержки фронта и среза сигналов не совпадают или зависят от путей прохождения сигналов в схеме и ее предыдущего состояния

Одним из вариантов описания инерциального поведения вентиля 2И с разными задержками фронта и среза может быть следующим

```
Architecture INERT of 12 is
Begin
Process ( X1, X2 );
Variable Z  bit,
Begin
—выход идеального вентиля
Z=X1 and X2,
 if Z=' 1' and Z'DELAYED=' 0' then
—срез
Y<=' 0' after 3 ns
End  if,
End process,
End,
```

Атрибут Z'DELAYED дает предыдущее значение сигнала

При описании более сложных ситуаций следует учитывать особенности реализации механизма учета задержек сигналов в VHDL С сигналом ассоциируется драйвер — множество сообщений о планируемых событиях в форме пар время-значение сигнала

В случае транспортной задержки, если новое сообщение имеет время, большее, чем все ранее запланированные, оно включается в драйвер последним В противном случае предварительно уничтожаются все сообщения, запланированные на большее время

В случае инерциальной задержки также происходит уничтожение всех сообщений, запланированных на большее время Однако разница в том что происходит анализ событий, запланированных на меньшее время, и если значение сигнала отличается от нового, то они уничтожаются

4.9. АТРИБУТЫ СИГНАЛОВ И КОНТРОЛЬ ЗАПРЕЩЕННЫХ СИТУАЦИЙ

Описания систем могут содержать информацию о запрещенных ситуациях, например недопустимых комбинациях сигналов на входах объектов, рекомендуемых длительностях или частотах импульсов и т п Например, в вентиле 2И возникает риск сбоя в ситуациях, когда фронт одного сигнала перекрывает срез другого

Средством отображения информации о запрещенных ситуациях в языке VHDL является оператор утверждения (оператор контроля, оператор аномалии) **assert** В нем, помимо контролируемого условия, которое не должно быть нарушено, т е. должно быть истинным, записывается сообщение **report** о нарушении и уровень серьезности ошибки **severity**

Для этого необходимо, чтобы в момент среза сигнала его задержанное на 10 нс значение было равно 1 Если оно равно 0, то длительность сигнала меньше 10 нс

Время предыдущего события в сигнале Z можно получить атрибутом LAST_EVENT

```
Architecture C1 of 12 is
Signal Z: bit ='0',
Begin
Process ( X1, X2 ),
Z<= X1 and X2;
Assert not (Z='0' and  not Z'STABLE and Z'DELAYED
(10 ns)= '0')
Report "риск сбоя в 1 в вентиле 12"
Severity worning;
Y<= transport Z after 10 ns,
End C1;
```

Более полное представление о предопределенных атрибутах сигналов можно получить из **Таблицы 4.1** Помимо предопределенных, пользователь может вводить дополнительные атрибуты для сигналов и других типов данных

Таблица 4.1. Предопределенные атрибуты сигналов

Пример	Тип результата	Пояснения
S'QUIET(T)	Boolean	TRUE, если сигнал S пассивен на интервале T
S'TRANSACTION	Bit	Инвертируется S каждый раз, когда S активен (изменяется)
S'STABLE(T)	Boolean	TRUE, если не было событий за интервал T
S'DELAYED(T)	Signal	Предыдущее значение S в момент NOW-T
S'ACTIVE	Boolean	TRUE, если сигнал активен
S LAST_ACTIVE	Time	Время, когда сигнал последний раз был активен
S EVENT	Boolean	TRUE, если происходит событие в S
S LAST_VALUE	Signal	Значение сигнала перед последним событием в нем
S'LAST_EVENT	Time	Время последнего события в S

4.10. АЛФАВИТ МОДЕЛИРОВАНИЯ И ПАКЕТЫ

Описание пакета VHDL задается ключевым словом **package** и используется, чтобы собирать часто используемые элементы конструкции для глобального применения в других проектах Пакет можно рассматривать как общую область хранения, используемую, чтобы хранить описания типов, констант и глобальные подпрограммы. Объекты, определенные в пределах пакета, можно использовать в любом другом проекте на VHDL, и можно откомпилировать в библиотеки для дальнейшего повторного использования

Пакет может состоять из двух основных частей: описания пакета и дополнительного тела пакета. Описание пакета может содержать следующие элементы

- объявления типов и подтипов;
- объявления констант,
- глобальные описания сигналов,
- объявления процедур и функций,
- спецификация атрибутов;
- объявления файлов;
- объявления компонентов,
- объявления псевдонимов,
- операторы включения

Пункты, появляющиеся в пределах описания пакета, могут стать видимыми в других проектах с помощью оператора включения.

Если пакет содержит описания подпрограмм (функций или процедур) или определяет одну или более задерживаемых констант (константы, чья величина не задана), то в дополнение к описанию необходимо тело пакета. Тело пакета (которое определяется с использованием комбинации ключевых слов "package body"), должно иметь то же имя, что и соответствующее описание пакета, но может располагаться в любом месте проекта (оно не обязано располагаться немедленно после описания пакета)

Отношение между описанием и телом пакета отчасти напоминает отношение между описанием и реализацией элемента (тем не менее, может быть только одно тело пакета для каждого описания пакета) В то время как описание пакета обеспечивает информацию, необходимую для использования элементов, определенных в пределах этого пакета (список параметров для глобальной процедуры или имя определенного типа или подтипа), фактическое поведение таких объектов, как процедуры и функции, должно определяться в пределах тела пакета

Приведенные выше описания объекта F базировались на стандартных средствах языка VHDL — сигналы представлялись в алфавите '1', ' 0 ', логические операции И, ИЛИ, НЕ также определялись в этом алфавите Во многих случаях приходится описывать поведение объектов в других алфавитах. Например, в реальных схемах сигнал кроме значений '1' и '0' может принимать значение высокого импеданса 'Z' (на выходе буферных элементов) и неопределенное значение 'X', например, отражая неизвестное начальное состояние триггеров

Переход к другим алфавитам осуществляется в VHDL с помощью пакетов. У пакета, как и у объекта проекта, различают объявление интерфейса **package** и объявление тела объекта **package body**.

Ниже приводится пример пакета P4 для описания объектов в четырехзначном алфавите представления сигналов ('X', '0', '1', 'Z'), X — значение не определено, Z — высокий импеданс В этом пакете приводится тип KONTAKT для представления сигналов в четырехзначном алфавите и определяются функции NOT и AND над ними

В ТТЛ-логике высокий импеданс на входе воспринимается как 1, что учитывается в таблице функции AND

Ниже следует объявление пакета P4

```
Package P4 is
--перечислимый тип
type KONTAKT is ('X', '0', '1', 'Z' );
function "NOT" (X: in KONTAKT) return KONTAKT;
function "AND" (X1, X2: in KONTAKT) return KONTAKT;
end P4;
```

Ниже следует объявление тела пакета P4

```
Package body P4 is
Function "NOT" (A: in KONTAKT) return KONTAKT is
Begin
  If A='X' then return 'X'
Else if A='1' then return '0'
Else if A='0' then return '1'
Else return 'Z'
End if;
End "NOT",
Function "AND" ( A1, A2: in KONTAKT ) return KONTAKT is
Begin
 If (A1='0' ) or (A2='X' ) then return to '0'
Else if (A1= 'X' ) or (A2='0' ) or (A2= 'X') and
(A1='0' ) then return  'X'
Else return to'1'
End if,
End "AND";
End P4;
```

Пример использования пакета P4 при описании объекта F_P4 Этот объект отличается от F, т к у него другой интерфейс

```
--подключается ( use) пакет P4, все его функции (ALL)
use P4 ALL,
entity F_P4 is
port (A1, A2. in KONTAKT; B1, B2: out KONTAKT )
end F_P4,
Описание архитектуры F_P4_a
Architecture F_P4_a of  F is
Begin
B2<= not ( A1 and A2 );
B1<= A1 and A2,
End F_P4_a,
```

Из этого примера видно, что в ряде случаев изменение алфавита моделирования не требует внесения изменений в описания объектов

Например, переход к семизначному алфавиту ('0', '1', 'X', 'Z', 'F', 'S', 'R'), где тип KONTAKT имеет дополнительные значения F — фронт, S — срез, R — риск сбоя, потребует только создания нового пакета и подключения его к объявлению объекта F_P4 Изменение в других частях описаний объекта проекта не потребуется

4.11. ОПИСАНИЕ МОНТАЖНОГО "ИЛИ" И ОБЩЕЙ ШИНЫ

В цифровой аппаратуре используются монтажные ИЛИ (И) и двунаправленные шины на элементах с тремя состояниями выхода

Монтажное ИЛИ

```
Signal Y. WIRED_OR bit,
K1· process
Begin
Y<= X1 and X2
End process K1,
K2. process
Y<= X3 and X4,
End process K2,
```

Общая шина на элементах с трехстабильными выходами

```
Type A3 is ('0', '1', 'Z');
Signal D. CHIN A3;
C1: process
Begin
 If E1='0' then D<=' Z'
Else D<= X1 and X2
End if;
End process C1;
C2. process
 If E2='0' then D<='Z'
Else D<= X3 and X4
End process C2;
```

Если каждой компоненте (K1, K2) схемы сопоставить процесс, то имеем два параллельных процесса, каждый из которых вырабатывает свой выходной сигнал

В языке VHDL предусмотрен механизм разрешения конфликтов, возможных в подобных ситуациях, когда сигнал имеет несколько драйверов Функция разрешения обычно описывается в пакете, а ее имя указывается при описании соответствующего сигнала Например, тело функции разрешения WIRED_OR (монтажное ИЛИ) имеет следующий вид

```
Function WIRED_OR (INPUTS  bit_vector) return bit is
Begin
For I in INPUTS'RANGE loop
If INPUTS( I ) = '1' then
Return '1' ;
End if;
End loop;
Return '0',
End,
```

Драйверы сигнала INPUTS неявно рассматриваются как массив, границы которого определяются атрибутом RANGE

Функция сканирует драйверы сигнала и, если хоть один из них равен '1', возвращает значение '1', иначе 0'

Функция разрешения SHIN для шины на элементах с тремя состояниями выходами может быть такой

```
Type A3 is ('0', '1', 'Z'),
 Type VA3 is array ( integer range <> of A3 ),
Function SHIN (signal X. VA3 ) return A3 is
Variable VIXOD. A3.= 'Z',
Begin
For I in X'RANGE loop
  I f X( I) /= 'Z' then
VIXOD.= X ( I);
Exit;
End if;
End loop,
Return VIXOD,
End SHIN,
```

Предполагается, что может быть включен, (то есть, не равен 'Z') только один из драйверов входных сигналов

4.12. ОРГАНИЗАЦИИ, ПОДДЕРЖИВАЮЩИЕ РАЗВИТИЕ VHDL

Министерство обороны США в начале 80-х годов финансировало разработку многоуровнего языка VHDL, стандартизовало его и обязало своих поставщиков цифровых микросхем представлять в составе документации их описание на VHDL. Это можно рассматривать как важный, но только первый шаг к обязательности формальных моделей для всех видов выпускаемой электронной техники В связи с возлагаемой на VHDL особой ролью, интерес к нему в США и в Европе огромен, созданы Американская и Европейская группы, занимающиеся всем комплексом вопросов, связанных с внедрением VHDL, как-то.

- уточнение семантики языка,
- разработка методологии описания различных классов ЦУ,
- разработка внутренних форматов представления VHDL-моделей в САПР для обеспечения совместимости разрабатываемых продуктов,
- создание анализаторов, позволяющих контролировать синтаксис и семантику VHDL-моделей,
- создание справочно-обучающих систем и резидентных справочников по VHDL, позволяющих писать VHDL-модели под управлением и контролем системы,
- создание мощных систем моделирования, использующих в качестве входного языка VHDL.

Спонсорами работ по развитию VHDL являются Air Force Wright Aeronautical Laboratories, Avionics Laboratory, Air Force Systems Command, United States Air Force, Wright-Patterson Air Force Base , Ohio 45433.

В России работы по языку VHDL поддерживаются Российским научно-исследовательским институтом информационных систем (РосНИИИС), Московским институтом электронного машиностроения (кафедра "Специализированные вычислительные комплексы" МИЭМ), Томским политехническим университетом (кафедра "Вычислительной техники"), Международным центром по информатике и электронике, НИИ "Квант", Ассоциацией заинтересованных в применении VHDL