Типы данных

Язык VHDL основан на концепции строгой типизации данных, т. е. любой единице информации в программе обязательно присваивается имя, и для нее должен быть определен тип. Определение информационной единицы размещается в разделе деклараций программного модуля, в котором оно используется, или иерархически предшествующего модуля. Тип данных определяет набор значений объектов, отнесенных к этому типу, а также набор допустимых преобразований этих данных. Данные разных типов несовместимы в одном выражении.

Данные, используемые в программах, относятся к одной из категорий: *константы, переменные и сигналы*. Различие между сигналами и переменными определяется далее. Декларация объектов языка VHDL имеет следующий синтаксис:

<декларация объектов> :: =

<категория> <имя>«, <имя>» :<тип> [:=<выражение>];

<категория> ::= CONSTANT |VARIABLE | SIGNAL.

Одна декларация может определять несколько объектов. Выражение в декларации должно совпадать по типу с декларируемым объектом и задает значения константы либо начальные значения сигналов и переменных. Простые примеры:

CONSTANT a: integer:=15;

VARIABLE b,c: BIT;

SIGNAL d,e : DOUBLE_WORD;

В последнем примере предполагается, что тип DOUBLE_WORD был ранее определен пользователем.

С помощью объектов языка VHDL описываются объекты проекта, т.е. различные подсистемы проектируемой (моделируемой) системы, например компоненты. О декларации компонентов речь пойдет ниже.

Язык VHDL предопределяет некоторый базовый набор типов данных, которые не требуют объявления в программе пользователя. Кроме того, пользователь может определять свои типы данных. Различают скалярные типы данных и агрегатные типы. Объект, отнесенный к скалярному типу, рассматривается как законченная единица информации. Агрегат представляет упорядоченную совокупность скалярных единиц, объединенных одинаковым именем. Классификация типов данных VHDL приведена на рис. 1.

Предопределенные типы данных

Сначала остановимся на предопределенных типах.

<предопределенные типы>:: =

INTEGER | REAL | BIT | BOOLEAN | CHARACTER | STRING | TIME | BIT_VECTOR| SEVERITY_LEVEL | FILE_OPEN_STATUS | FILE_OPEN_KIND

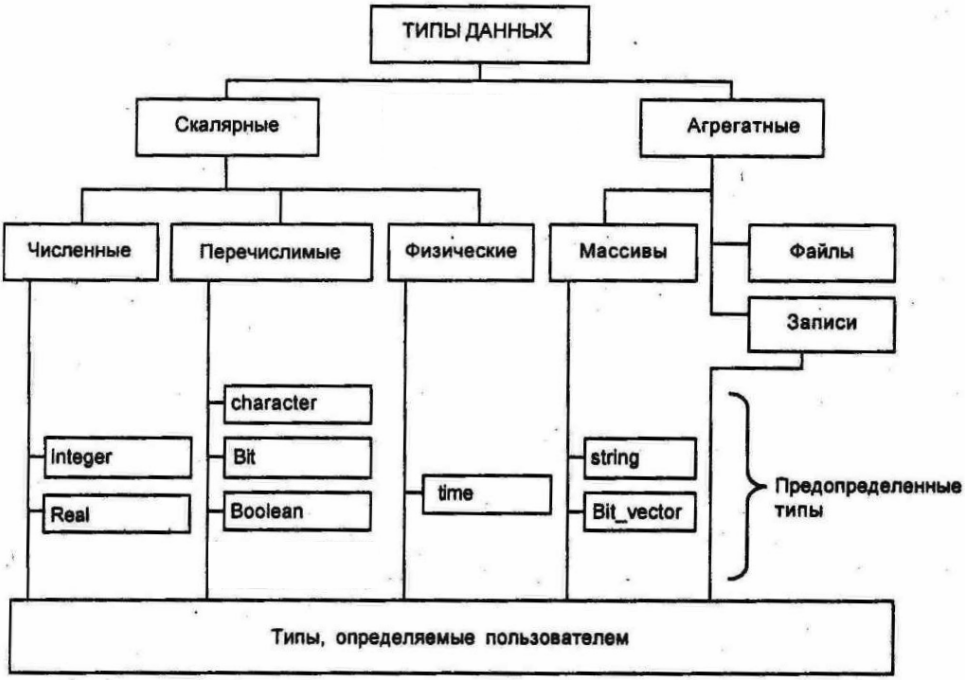

Рис. 1

Типы INTEGER и **REAL** определяют численные данные — целые и действительные соответственно. Диапазон представления чисел может зависеть от реализации, но стандартными считаются диапазоны $\{-2^{31}+1, +2^{31}-1\}$ для типа **INTEGER** и $\{10^{38}, -10^{38}\}$ для **REAL**. Над этим типом данных определены арифметические операции:

\+ - сложение или повторение;

-.- вычитание или инверсия;

* - умножение;

/ - деление;

mod - число по модулю (5 mod 3 = 2);

rem - остаток от деления;

abs- модуль (абсолютное значение числа);

** - возведение в степень.

Определены для этих кодов также и операции арифметического отношения =, /=,<, <=, >=,>, которые дают результат типа **BOOLEAN**. В арифметических выражениях предполагаются традиционные способы определения старшинства операций, включая применение скобок.

Данные типа BIT могут принимать значения из множества {'0', '1'}. На данных типа BIT определены логические операции:

NOT - инверсия;

OR - операция ИЛИ;

NOR - операция ИЛИ-НЕ;

AND - операция И;

NAND - операция И-НЕ;

XOR - неравнозначность;

XNOR - равнозначность.

Операции определены по правилам положительной логики (a AND b дает значение '1'**,** только если оба члена выражения равны '1' и т. д.).

Данные **типа BOOLEAN** также могут принимать два значения: {**TRUE, FASLSE**}, и на них определены те же операции, что и над данными типа **BIT**. Разница между типами **BIT** и **BOOLEAN** состоит в том, что первые применяются для представления уровней логических сигналов в аппаратуре, а вторые для представления обобщенных условий, например результатов сравнения. Так, если переменная select_e определена как бит, то нельзя записать условный оператор в виде

IF select_e THEN ...

следует записывать

IF select_e='1' THEN...

Если бы переменная select_e была определена как **BOOLEAN,** то, наоборот, первый вариант был бы допустим, а второй нет.

Тип CHARACTER объединяет все символы, определенные в используемой операционной системе - буквы, цифры, специальные символы. В тексте программы символьная константа записывается как стандартный символ, заключенный в одинарные кавычки ('a', 'b', и т. п.). Отметим, что символы '0' и '1' имеют двойное назначение — и как символ, и как логическое значение. В каждом конкретном случае тип определяется по контексту.

Тип TIME - время - служит для задания задержек элементов и времени приостанова процессов при моделировании. Запись временной константы имеет вид:

<целое> <единица измерения времени>

Определены следующие единицы измерения времени:

fs - фемтосекунда;

ps - пикосекунда;

ns - наносекунда;

us - микросекунда;

ms - миллисекунда;

s - секунда.

Над данными типа "время" определены операции отношения, сложения и вычитания, а также умножения и деления на целое.

Тип SEVERITY_LEVEL задает следующее множество значений: {note, warning, error, failure} и используется для управления работой компилятора или программы моделирования. С помощью переменных и констант этого типа в операторах **ASSERT** определяются действия, которые следует выполнить при обнаружении некоторых условий. Фактическая трактовка действий в стандарте не оговорена и оставлена на усмотрение разработчиков системы моделирования.

Типы FILE_OPEN_STATUS и **FILE_OPEN_KIND** обеспечивают возможность контроля процедур обмена между программой моделирования и файловой системой инструментального компьютера.

Типы STRING и **BIT_VECTOR** относятся к агрегатным и фактически определены как неограниченный массив символов и массив битов соответственно. Более подробно правила работы с массивами и их элементами рассмотрены далее. В тексте программы строковая константа заключается в двойные кавычки.

Пользователь имеет возможность определить собственнее типы, используя *декларацию типа:*

<декларация типа> ::= TYPE <имя типа> **IS** <определение типа>;

<определение типа> :: = <определение перечислимого типа> |

<определение ограниченного типа> | <определение физического типа> | <определение типа массивов> | <определение типа записей>.

Скалярные типы, вводимые пользователем

Определение *перечислимого типа* имеет вид:

<Определение перечислимого типа>::=

(перечислимое значение «, перечислимое значение »)

<перечислимое значнание> ::=

<идентификатор> | <символьная константа>.

Примеры:

TYPE state **IS** (S0,S1,S2,S3);

Может представлять, например, набор допустимых состояний системы, для каждого состояния определяются выполняемые действия и правила перехода в другое состояние.

TYPE colour **IS** (white, black, red, green, blue, yellow, argenda);

Набор цветов. Переменные этого типа могут применяться, например, для управления выводом на дисплей как в сеансах моделирования, так и в реальных устройствах.

TYPE octal_digit **IS** ('0', '1', '2' ,'3');

Определение *численных типов пользователя* целесообразно, во-первых, для контроля совместимости данных в программах, а во-вторых, для точного задания разрядности слов, представляющих данные в проектируемом объекте. В общем случае определение ограниченного типа подчиняется синтаксическому правилу:

<Определение ограниченного типа> ::= [<базовый тип>] <диапазон>

<диапазон>:: = RANGE <ограничение> <направление><ограничение> | **RANGE <>**

<направление> ::= DOWNTO | **TO**

Направление (**TO** - увеличение, **DOWNTO** - уменьшение) должно быть согласовано с соотношением ограничений.

Примеры:

TYPE Unsigned_short **IS** integer **RANGE** 0 **TO** 255;

TYPE my_data **IS** integer **RANGE** -2**(n-1) +1 **TO** 2**(n-1) -1;

TYPE input_level **IS RANGE** -10.0 **TO** +10.0;

Тип Unsigned short объединяет целые положительные числа, которые могут быть представлены в байтовом формате.

Тип my_data объединяет целые в диапазоне, который объявляет пользователь через разрядность данных *n*. В этом случае пользователь точно указывает компилятору число разрядов, необходимое для представления данных, обеспечивая экономию ресурсов микросхемы по сравнению с неограниченным типом.

При объявлении типа input level базовый тип явно не задан, тип ограничений устанавливается в соответствии с типом их фактических значений.

Пусть пользователь в одном проекте вводит два типа:

TYPE data **IS** integer **RANGE** 0 **TO** 15;

TYPE controle **IS** integer **RANGE** 0 **TO** 15;

Хотя с точки зрения представления эти типы равноценны, оказывается, что управляющие сигналы controle и сигналы данных data несовместимы, что облегчает контроль корректности описания.

Физические типы

Наряду с предопределенным типом **TIME** пользователь может определить другие физические типы, которые будут отражать физические (механические, электрические или иные) свойства носителя информации/

<определение физического типа> ::= RANGE <диапазон>
UNITS
<имя базовой единицы>
« <имя вторичной единицы> = <значение единицы>; »
END UNITS [<имя типа>];

Пример:

TYPE voltage **IS RANGE** -5E6 **TO** +5E6;
UNITS uV; -- базовая единица - микровольт
mV= 1000 uV; -- милливольт
V=1000 mV; -- вольт
END UNITS voltage;

Введение такого типа позволяет описывать и моделировать сопряжение цифровой логической схемы с аналоговыми источниками.

Массивы и записи

Массив, как и в других языках, — это набор данных, объединенных общим именем и различаемых по порядковым номерам (индексам). Для того чтобы ввести объект типа "массив", необходимо предварительно объявить соответствующий тип на основе следующих синтаксических правил:

<определение типа массива» ::=
ARRAY (<диапазон> «, <диапазон> ») **OF** <тип элемента массива>

Диапазон задает множество допустимых значений индекса. Число измерений массива формально не ограничено. Если диапазон задан конструкцией **RANGE<>**, то это является объявлением неограниченного массива. В этом случае определяется не диапазон значений индекса, а только тип индексной переменной. Такое определение предполагает задание диапазона при определении конкретного экземпляра объекта, относимого к такому ти-

пу, например, при вызове подпрограмм. В подобных случаях диапазон устанавливается динамически в соответствии с диапазоном подставляемого фактического параметра.

Примеры:

TYPE ram **IS ARRAY** (length-1 **DOWNTO** 0) **OF** integer **RANGE** 2**width-l **DOWNTO** 0;

TYPE ram1 **IS ARRAY** (length-1 **DOWNTO** 0, width-1 **DOWNTO** 0) **OF** std_logic;

TYPE ram2 **IS ARRAY** (integer **RANGE<>**, integer **RANGE<>**) **OF** std_logic;

Во всех приведенных декларациях объявляется в сущности одно и то же, а именно матрица ячеек памяти емкостью length слов по width разрядов в каждом, причем предполагается, что эти параметры были ранее определены. Однако выполнено это разными способами, а значит, и ссылаться на эти типы следует по-разному, ram и ram1 определены как ограниченные типы массивов, ram - как одномерный массив целых, а ram1 — как двумерный массив битов. ram2 определен как неограниченный тип и требует задания границ индексов при декларации объектов выбранного типа.

Декларации объектов, принадлежащих приведенным типам, могут выглядеть следующим образом:

VARIABLE ram_instance: ram;

VARIABLE raml_instance: ram1;

VARIABLE ram2_ins: ram2 (length-1 **DOWNTO** 0,width-1 **DOWNTO** 0);

При обращении к элементам массива в программе индексы помещаются в скобках следом за именем массива. Тип индексного выражения должен соответствовать типу индекса, объявленного при декларации типа массива. При обращении к элементу многомерного массива индексные выражения записываются через запятые в порядке, определенном в декларации типа.

ram2_ins (y,5) := x0; -- x0 определено как бит; y - целое;

z <= raml_instance (Y) -- Y и z - целые.

Для одномерных массивов определено несколько групповых операций, в которых массив рассматривается как единое целое. Это, прежде всего, операция конкатенации & (объединение строк). Например, приведенная ниже последовательность операторов присваивает сигналу b значение "11011001".

a:="1001";

b <= "1101" & a;

Здесь a и b — строки или битовые векторы, причем a - переменная, а b - сигнал.

Операции сдвига, которые будут рассмотрены, определены для одномерных массивов типа **BIT** или **BOOLEAN** и записываются следующим образом:

<имя массива> <символ операции сдвига> <целое>

Целое в записи выражения для сдвига определяет число разрядов, на которые осуществляется сдвиг кода.

Запись - эта структура данных, каждая информационная единица которой, называемая полем записи, имеет индивидуальное имя и может быть индивидуального типа. Записи удобны для агрегатирования различных данных, характеризующих один объект. Для использования записей как переменных сначала надо объявить соответствующий тип:

<определение типа записи> ::=

RECORD <список полей записи>:< тип»;

«<список полей записи>:< тип>;»

END RECORD;

Пример. Определим тип pixel, представляющий цветовые составляющие отображения точки на экране в формате FULL COLOR (полная цветопередача), предусматривающем восьмиразрядное представление трех цветовых составляющих.

TYPE pixel **IS RECORD** red, green, blue: integer **RANGE** 0 **TO** 255;

END RECORD;

Тогда тип "видеопамять" может быть определен как

TYPE video_ram **IS ARRAY**(integer **RANGE<>**,integer **RANGE <>**) **OF** pixel;

Экземпляр видеопамяти будет определяться, например, следующим образом:

SIGNAL VRAM : video_ram (679 **DOWNTO** 0, 839 **DOWNTO** 0);

Этот экземпляр может сохранять информацию об изображении размером 680 строк по 840 элементов в строке. Выборка значения красной составляющей верхнего левого элемента изображения из такой памяти описывается оператором

Out_red <= VRAM (0,0).red;

Подтипы

Специфическим понятием языка VHDL является подтип. Объекты, отнесенные к подтипу, сохраняют совместимость с данными типа, из которого выделяется подтип так называемого базового типа. Однако введение подтипа:

✓ определяет множество допустимых значений данных подтипа как подмножество допустимых значений базового типа;

✓ позволяет вводить дополнительные функции преобразования, определяемые только для данных подтипа.

Синтаксис декларации подтипа следующий:

<декларация подтипа> ::=

SUBTYPE <имя подтипа> **IS** <имя базового типа или подтипа> [<ограничение>] ;

Пример:

SUBTYPE bit_in_word_number **IS** integer **RANGE** 31 **DOWNTO** 0;

Определен подтип типа integer. Данные этого подтипа предполагается использовать для индексации бита в 32-разрядном коде. Данные совместимы с данными типа integer. Однако присвоение этим данным значений вне указанного диапазона вызывает сообщение об ошибке.