2 МОВА ОПИСУ АПАРАТУРИ VHDL. ОСНОВИ СИНТАКСИСУ, ТИПИ ДАНИХ, КЛАСИ ОБ'ЄКТІВ

2.1 Діючі стандарти мови VHDL

Як було показано в попередньому розділі, традиційний підхід до проектування складних цифрових систем за допомогою булевих рівнянь призводить до суттєвого ускладнення етапу розробки, зниження наочності й інформативності. Тому, стає доцільним проводити проектування, грунтуючись на текстовому алгоритмічному описі пристрою. Найбільш поширеним засобом такого подання проектів є мова опису апаратури VHDL (Very high speed integrated circuits Hardware Description Language), який використовується протягом ряду років найбільшими виробниками електронних систем.

Розробка мови VHDL як стандартизованого засобу для опису електронних схем проводилася на замовлення Міністерства оборони США. У 1987 році було введено перший стандарт VHDL ANSI/IEEE Std 1076-1987. Основний стандарт VHDL, орієнтований на проектування цифрових схем, був затверджений в 1993 році (ANSI/IEEE Std 1076-1993). Останнім стандартом мови є ANSI/IEEE Std 1076.1-1999, що включає розширення, спрямовані на опис аналогових і цифро-аналогових систем.

Синтезована підмножина мови VHDL була затверджена в стандарті IEEE P1076.6/D1.12, 1998. Даний державний стандарт на переносимість проектів цифрових систем з однією САПР в іншу і орієнтований на одноманітність опису алгоритмічних конструкцій.

Основними перевагами VHDL, порівняно з іншими мовами опису апаратури, є:

- висока поширеність і універсальність;
- доступність VHDL – орієнтованих САПР для проведення моделювання та синтезу цифрових систем;
- порівняно короткий період розробки;
- гнучкість створених проектів, підлаштовуватися під конкретні завдання споживача;
- можливість описувати проекти на функціональному, структурному і змішаному рівнях з різним ступенем деталізації;
- наочність алгоритмічних конструкцій;
- можливість проведення універсального тестування проекту;
- простота документування проекту.

Основні можливості VHDL ілюструються на рис. 2.2.

Мова VHDL містить загальноалгоритмічні і проблемно-орієнтовані конструкції. Загальноалгоритмічні конструкції роблять VHDL близьким за синтаксисом і семантикою до сучасних мов програмування (Pascal, C) і містять

оператори присвоювання, умови, вибору, циклу, механізми роботи з підпрограмами і т.д. У проблемно-орієнтованій частині визначено конструкції мови, що дозволяють описувати цифрові системи в загальноприйнятих розробниками поняттях і термінах, таких як часові затримки у фізичних одиницях, засоби оголошення об'єктів і архітектур, паралельна модель обчислень, клас об'єктів типу **signal** (сигнал) та ін.

Спочатку VHDL замислювався як мова для моделювання електронних схем. Однак проблеми моделювання відступають на другий план порівняно з основною метою проектувальника електронної апаратури – генерування схеми за її формальним описом за допомогою алгоритмічних конструкцій. Сьогодні за допомогою спеціально розроблених програмних пакетів VHDL - код може бути перетворений в схемотехнічне рішення на базі структур FPGA. Такі САПР називаються VHDL - синтезаторами, а набір операторів і конструкцій VHDL, перетворюються в схемні вузли – синтезуються підмножиною мови VHDL.

Рисунок 2.1 – Основні можливості мови VHDL

2.2 Ознайомлювальний проект цифрового пристрою на основі VHDL – опису

Введення в мову VHDL доцільно починати з прикладу опису найпростішого цифрового пристрою за допомогою базових мовних конструкцій. Такий підхід дозволяє отримати перше уявлення про предметну галузь, стилі і можливості VHDL описів, не вникаючи в зайві деталі.

Отже, розглянемо комбінаційну схему, зображену на рис. 2.2. Схема складається з двох двоходових компонентів *K1* і *K2*, виконують логічну функцію *&* ("І"), і одного двоходового компонента *K3*, що виконує логічну функцію *1* ("АБО"). Під компонентом у VHDL розуміють окремий незалежний функціональний блок, під сигналами – з'єднувальні дроти. Розглядаючи даний пристрій як єдиний схемотехнічний модуль, можна виділити його інтерфейс, що характеризує зовнішні сигнали *A, B, C, D, E*, на яких можуть бути присутніми значення низького "0" або високого "1" рівнів напруги (двійкова логіка). Сигнали *A, B, C, D* є вхідними, а *E* – вихідним сигналом схеми.

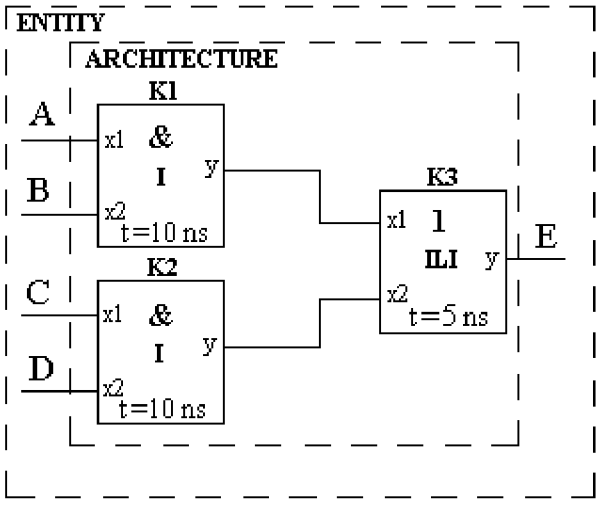


Рисунок 2.2 – Приклад найпростішої комбінаційної схеми

Опис інтерфейсної частини пристрою на VHDL відбувається в блоці *ENTITY* (див. рис. 2.2):

ENTITY Comb IS

PORT (A, B, C, D: IN BIT; E: OUT BIT);

END comb;

У цьому фрагменті *Comb* – ім'я пристрою, в дужках після службового слова *PORT* позначає інтерфейсні сигнали, вказані імена зовнішніх сигналів їх режими роботи (*IN/OUT*) і їх тип – *BIT* (двійкова логіка).

Під архітектурою *ARCHITECTURE* в VHDL розуміють структуру пристрою у вигляді сукупності компонентів *COMPONENT* і внутрішніх з'єднань *SIGNAL*. Декларацiйна частина архітектури *Struct* розглянутого пристрою *Comb* наведена нижче:

ARCHITECTURE Struct OF Comb IS

COMPONENT I

PORT(X1,X2:IN BIT; Y:OUT BIT);

END COMPONENT;

COMPONENT ILI

PORT(X1,X2:IN BIT; Y:OUT BIT);

END COMPONENT;

SIGNAL W,V:BIT;

У цій частині програми описуються присутні в пристрої типи компонентів з їх інтерфейсними сигналами, а також внутрішні сигнали схеми. Ім'я типу *I* відповідає компоненту, що виконує логічну функцію *&*, ім'я типу *ILI* – компоненту з функцією *1*. Інтерфейсними сигналами для компонентів є їхні зовнішні висновки (порти), позначені на схемі *x1*, *x2*, *y*, і описувані в дужках після службового слова *PORT* аналогічно блоку *ENTITY*. Внутрішні сигнали *SIGNAL* пристрою, позначені *V* і *W*, пов'язують виходи примірників компонентів *K1* і *K2* типу *I* з входами компонента *K3* типу *ILI*. Ці сигнали, як і зовнішні, мають тип *BIT*.

Безпосередньо після службового слова *BEGIN* розміщується операційна частина архітектури пристрою, що складається, в даному прикладі, з трьох операторів конкретизації компонента *PORT MAP*:

```
BEGIN
K1: I PORT MAP (x1 => A, x2 => B, y => W);
K2: I PORT MAP (x1 => C, x2 => D, y => V);
K3: ILI PORT MAP (x1 => W, x2 => V, y => E)
END ARCHITECTURE Struct;
```

У цьому фрагменті вказується розміщення компонентів і вказуються зв'язку між їхніми висновками (портами). Оператор конкретизації компонента починається з мітки, що визначає ім'я компонента в схемі (див. рис. 2.2). Потім, через двокрапку вказується тип компонента (*I, ILI*), ключові слова *PORT MAP*, у дужках після яких вказується відповідність інтерфейсних сигналів (портів) компонентів сигналів схеми. Останній рядок фрагмента вказує на завершення розділу опису архітектури *STRUCT* на пристрої *Comb*.

Для завершення опису схеми необхідно задати логічні функції, що виконуються компонентами кожного типу. За аналогією з розглянутим вище прикладом, кожен компонент описується як окремий пристрій: спочатку в блоці *ENTITY* йде розділ описів інтерфейсних сигналів *Ports*, а потім слід код відповідного архітектурного тіла:

```
ENTITY I IS
PORT (X1, X2: IN BIT; Y: OUT BIT);
END I;
ARCHITECTURE BEHI OF I IS
BEGIN
Y <= X1 AND X2 AFTER 10 NS;
END BEHI;
ENTITY ILI IS
PORT (X1, X2: IN BIT; Y: OUT BIT);
END ILI;
ARCHITECTURE BEHILI OF ILI IS
BEGIN
Y <= X1 OR X2 AFTER 5 NS;
END BEHILI;
```

Опис функції, виконуваної компонентом, проводиться у розділі архітектури, наприклад:

```
Y <= X1 OR X2 AFTER 5 NS,
```

де <= оператор паралельного призначення сигналу, а після службового слова *AFTER* вказується часова затримка спрацьовування компонента – інтервал часу, після закінчення якого на виході схеми з'явиться результат роботи функції.

Повний текст одного з варіантів VHDL - описів розглянутого пристрою наведено нижче:

```
ENTITY Comb IS
PORT(A,B,C,D: IN BIT; E:OUT BIT);
END comb;
ARCHITECTURE Struct OF Comb IS
COMPONENT I
PORT(X1,X2:IN BIT; Y:OUT BIT);
END COMPONENT;
COMPONENT ILI
PORT(X1,X2:IN BIT; Y:OUT BIT);
END COMPONENT;
SIGNAL W,V:BIT;
BEGIN
K1: I PORT MAP(x1=>A,x2=>B,y=>W);
K2: I PORT MAP(x1=>C,x2=>D,y=>V);
K3: ILI PORT MAP(x1=>W,x2=>V,y=>E);
END ARCHITECTURE Struct;
ENTITY I IS
PORT(X1,X2: IN BIT; Y:OUT BIT);
END I;
ARCHITECTURE BEHI OF I IS
BEGIN
Y<=X1 AND X2 AFTER 10 NS;
END BEHI;
ENTITY ILI IS
PORT(X1,X2: IN BIT; Y:OUT BIT);
END ILI;
ARCHITECTURE BEHILI OF ILI IS
BEGIN
Y<=X1 OR X2 AFTER 20 NS;
END BEHILI;
```

Дослідити роботу схеми можна, вивчивши часові діаграми вхідних і вихідних сигналів. Для цього необхідно створити, так званий, вектор тестів – спеціальну програму, що дозволяє вичерпно протестувати пристрій. У найпростішому випадку необхідно подати на входи схеми комбінацію значень сигналів, наприклад:

A <= '1'; B <= '1'; C <= '1'; D <= '1'; або *A <= '0'; B <= '1'; C <= '1'; D <= '0'*; і т.д.

Програма моделювання виконає побудову відповідних часових діаграм.

Розглянутий приклад ілюструє основні принципи моделювання пристроїв на VHDL і містить основні структурні елементи мови: інтерфейсну частину, розділ опису архітектури схеми і бібліотеку вхідних компонентів.

2.3 Алфавіт мови

Алфавіт VHDL – набір символів, дозволених до використання і сприймаються компілятором. В алфавіт входять:

• латинські малі та великі літери:

A, B, C,. . ., X, Y, Z, a, b, c,. . ., X, y, z;

• цифри від **0** до **9**;

• символи підкреслення "_"; пробілу, табуляції, нового рядка;

• спеціальні символи, що використовуються для побудови конструкцій мови: **+ – * / = < > . , : ; ( ) # ‘ “ |** ;

• складові символи, які сприймаються компілятором як поодинокі: **<= => >= := /=** – (прогалини й інші роздільники між елементами складених символів неприпустимі).

Під час запису VHDL коду немає відмінності між великими та стічними літерами латинського алфавіту (як у мові Pascal). Використання символів кирилиці в конструкціях мови, за винятком коментарів, не допускається.

2.4 Лексичні елементи мови VHDL

Програма мовою VHDL є послідовністю роздільних лексичних елементів: ключових (зарезервованих) слів, ідентифікаторів, літералів, роздільників і коментарів. Розглянемо докладно правила запису лексичних елементів.

Ключові слова. За аналогією з іншими мовами програмування в VHDL є набір ключових слів, наведений в таблиці 2.1.

Таблиця 2.1 – Список ключових слів мови VHDL Std 1076-1993

abs	access	after	alial	all	and
architecture	array	assert	attribute	begin	block
body	buffer	Bus	case	component	configuration
constant	disconnect	downto	else	elsif	end
entity	exit	file	for	function	generate
generic	group	Guarded	if	impure	in
inertial	inout	is	label	library	linkage
literal	loop	map	mod	nand	new
next	nor	not	null	of	on
open	or	others	out	package	port
postponed	procedure	process	pure	range	record
reject	register	rem	report	return	rol
ror	select	severity	shared	signal	sla
sll	sra	srl	subtype	then	to
transport	type	unaffected	units	until	use
variable	wait	when	while	with	xnor
xor					

Ідентифікатори можуть містити великі та малі символи латинського алфавіту (від A до Z і від a до z), цифри (від 0 до 9) і символ підкреслення "_". В ході запису ідентифікаторів великі та малі символи алфавіту еквівалентні, використання символів кирилиці не допускається. Ідентифікатор має бути записаний за такими правилами:

- починатися з літерного символу;
- не може закінчуватися символом підкреслення "_";
- не може містити поспіль два і більше символів підкреслення;
- не може бути ключовим словом мови.

Приклади коректного запису ідентифікаторів наведені нижче:

Clk еквівалентно **CLK** еквівалентно **clk**

Adr4 еквівалентно **ADR4** еквівалентно **adr4**

Enabled_1 еквівалентно **ENABLED_1** еквівалентно **enabled_1**.

Типові помилки під час запису ідентифікаторів:

9adr – ідентифікатор починається з цифри;

clk_ – ідентифікатор закінчується на символ "_";

Sum @ adr – ідентифікатор містить неприпустимий символ "@";

adr___en – ідентифікатор містить поспіль три символи "_";

Port – ідентифікатором є ключове слово (див табл. 2.1).

Додатково в VHDL введений механізм подання **розширених ідентифікаторів**, які можуть включати будь-яку послідовність символів, укладених в / /. Наприклад, **/ 65535 /** є допустимим ідентифікатором.

Літерали. У мові VHDL передбачено використання літералів таких типів: базових, десяткових, символьних, рядкових, бітових рядків. У стандарті мови VHDL ANSI/IEEE Std 1076.1-1999 введені можливості роботи як з цілими, так і речовими числами. Однак останні не підтримуються засобами для синтезу ПЛІС.

Базові літерали використовуються для подання чисел у системах числення з довільною основою. В ході запису базового літерала необхідно спочатку вказати основні системи числення, а потім значення, обмежене з обох сторін символами "#". У разі експоненційної форми подання числа символами "#" обмежується тільки мантиса. Для поліпшення сприйняття багатозначних числових значень між цифрами можуть перебувати символи підкреслення "_". Приклад подання числа 143_{10} у формі запису базових літералів:

2 # 10001111 # еквівалентно **2 # 1000_1111 #** в двійковій системі;

8 # 217 # еквівалентно **8 # 2_1_7 #** у вісімковій системі;

10 # 143 # еквівалентно **10 # 1_43 #** у десятковій системі;

16 # 8F # еквівалентно **16 # 8f #** у шістнадцятковій системі;

Десяткові літерали можуть бути цілими і дійсними та можуть записуватися як в стандартній, так і в експоненційній формі. Символ експоненти може бути рядковим або великим (**E** або **e**). Мінус перед значенням експоненти вказує на дійсне число. Приклади запису десяткових літералів:

цілі: **500, 5_00, 5_0_0, 5E2, 5e2**;

дійсні: **52.8, 5.28E1, 5.28e1**;

Символьні літерали можуть містити одиночний символ VHDL -алфавіту, укладений між апострофами. Під час запису символьних літералів враховується регістр символів, наприклад:

'C' і **'c'** - різні значення.

Рядковий літерал є набором (рядком) символів, укладених у лапки, які є рядковими дужками. Приклади:

"ABC", "Port_1", "Expression value".

Приклад некоректного запису рядкового літерала:

'ABCD' – використовуються апострофи замість подвійних лапок.

Літерал **бітовий рядок** використовується як додатковий засіб для подання числових значень. Бітові рядки формуються у вигляді послідовності літер, вкладених у лапки, і можуть бути двійковими, вісімковими і шістнадцятковими. Перед бітовими рядками ставляться символи – покажчики основи системи числення **b, o, x** відповідно, реєстр яких не має значення. Символ підкреслення не є значущим. Бітові рядки характеризуються величиною (довжиною) значення, яка виражається в бітах. Приклади:

b "1101" – бітовий рядок у двійковому поданні, довжина в бітах – 4;

o "527" – бітовий рядок у вісімковому поданні;

еквівалентний значенню **b "101_010_111"**, довжина в бітах – 9;

x "29" – бітовий рядок у шістнадцятковому поданні;

еквівалентний значенню **b "0010_1001"**, довжина в бітах – 8.

В ході запису бітових літералів у двійковому поданні покажчик системи числення b можна опускати.

Роздільники. Роздільником між командами VHDL є символ "**;**", який слід давати в обов'язковому порядку після запису кожної команди. Роздільний запис лексичних елементів мови забезпечується пробілами, кількість яких може бути довільною.

Коментарі. Ознакою коментаря є два символи тире, записані підряд ("--"). Компілятором ігноруються символи, такі за ознакою коментаря до кінця рядка. Коментарі можуть містити символи, що не входять в алфавіт VHDL, наприклад, символи кирилиці.

Для прикладу виконаємо лексичний аналіз рядка VHDL - коду, що містить пропозицію:

C <= A or *B;* -- порозрядна логічна операція "**or**" над сигналами A і B, яка містить лексичні елементи: "C", "<=", "A", "or", "B" , ";".

Елементи "**A**", "**B**", "**C**" є ідентифікаторами;

or – ключове слово (код логічної операції "АБО");

<= – складовий роздільник (оператор призначення сигналу);

; – роздільник;

- – коментар;

додатково як роздільники використовуються пробіли.

Мітки (label) ставляться перед операторами і відокремлюються двокрапкою:

ім'я мітки: оператор ...

У мові VHDL **мітки** служать виключно для описових функцій, з метою спрощення ідентифікації певних мовних конструкцій. Команд переходу на мітку (в явному вигляді) в синтаксисі VHDL немає.

2.5 Класифікація типів даних у мові VHDL

Типом даних називається поіменна множина значень із загальними ознаками. VHDL належить до класу суворо типізованих мов. Це означає, що кожен об'єкт мови має бути оголошений зі своїм типом і даному об'єкту можуть присвоюватися значення тільки відповідного типу. Суворе узгодження типів забезпечує VHDL - проектам додаткову надійність і економію часу в ході налагодження.

Кожен тип даних у VHDL має певні набори прийнятих значень і допустимих дій (операцій) над ними. Синтаксис оголошення типу даних у загальному випадку подається в такий спосіб:

type ім'я типу **is** опис типу,

де напівжирним шрифтом виділені ключові слова **type** і **is**.

Наслідком суворої типізації VHDL є ефект, який полягає в тому, що об'єкти, які належать до типів даних з різними іменами, але однаковими описами, не вважаються приналежними одному типу і не можуть обмінюватися значеннями.

Стандартом мови зумовлено значна кількість простих і складних типів даних (див. рис. 2.3), введені вказівні та файлові типи, а також є засоби для створення типів даних, визначених розробником.

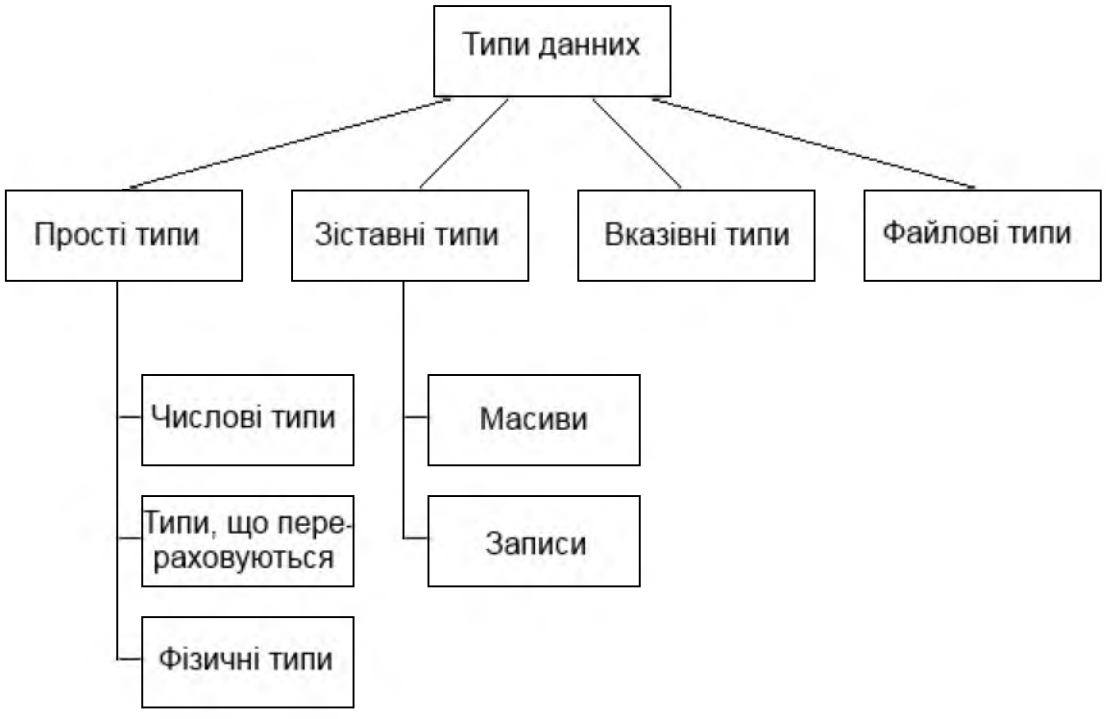


Рисунок 2.3 – Класифікація типів даних мови VHDL

2.6 Прості типи даних

Прості (скалярні) типи даних використовуються в VHDL для опису чисел, символів, значень сигналів, часових інтервалів й інших фізичних об'єктів.

У VHDL для подання **числових даних** використовуються **цілі і дійсні типи**. Формат оголошення числового типу:

type *ім'я типу* **is range** діапазон значень;

де діапазон значень подається у вигляді конструкції з використанням ключових слів **to** або **downto**:

вираз_1 **to / downto** вираз_2,

що визначає початкову і кінцеву величини діапазону прийнятих значень, які мають бути обчислені в процесі компіляції. Ключове слово **to** служить для визначення зростаючого діапазону (коли ліва межа менше, ніж права), а **downto** – для визначення діапазону, що зменшується (коли ліва межа більше ніж права). При ініціалізації, об'єкти числових типів, за замовчуванням, приймають початкові значення рівні лівій межі діапазону.

Стандартом VHDL для подання числових даних передбачені типи даних **integer** (цілий) і **real** (дійсний).

Тип **integer** (type **integer** is range -2147483648 to 2147483647) використовується для подання цілих чисел у діапазоні -2^{31} ... $2^{31}-1$.

Тип **real** (type **real** is range -1.0E38 to 1.0E38) має діапазон значень від $-1.0 \cdot 10^{38}$ до $1.0 \cdot 10^{38}$ і використовується для подання дійсних чисел. Однак даний тип не підтримується засобами синтезу ПЛІС.

Перелічуваний тип визначається списком (перерахуванням) усіх можливих значень даного типу. Формат оголошення цього типу:

type *ім'я типу* is (значеніе_0, [значення_1, ..., значення n-1]);

Елементи списку значень нумеруються при компіляції зліва - направо, починаючи з нуля. Наприклад, оголошення типу **control**

type *control* **is** (CLK, EN, SYN);

означає, що прийняті значення кодуватимуться: CLK – 0, EN – 1, SYN – 2.

У разі, якщо опис різних типів включає однакові імена (ефект перевантаження імен), то для однозначного визначення до якого типу даних належить значення необхідно явну вказівку типу даних за допомогою такої конструкції:

type *control_1* **is** (STROBE, BUSY, ACK, EN, AE);

type *control_2* **is** (STROBE, BUSY, ACK);

. . .

control_1 '(BUSY); -- Значення BUSY типу *control_1*;

control_2 '(BUSY); -- Значення BUSY типу *control_2*;

де спочатку вказується ім'я типу, а потім, через апостроф, значення, вміщене в стандартні дужки.

У VHDL стандартним переліком типів є: символьний тип (*character*), логічний (*boolean*), бітовий (*bit*), і стандартний логічний тип (*std_ulogic*).

Тип ***character*** підтримує повний набір символів восьмирозрядних кодувань ISO, що включає латиницю, кирилицю, цифрові, службові, псевдографічні і керуючі символи. Визначення типу має такий вигляд:

type *character* **is** (nul, soh ,…, ' ', '!', ..., '0', '1', 'A', 'B', ..., 'a', 'b', ..., '╕ ' , '╣', ..., 'а', 'б', ..., 'а', 'б', ..., 'ю', 'я');

Логічний тип даних ***boolean***, що приймає значення *false* і *true* (хибність й істина), використовується для подання результатів умовних виразів, до складу яких можуть входити оператори порівняння і логічні команди. Його визначення має такий вигляд:

type *boolean* **is** (*false*, *true*);

Тип ***bit*** призначений для опису логічних рівнів сигналів у цифрових схемах і приймає значення "0" і "1", які **не еквівалентні** *false* і *true* типу *boolean*, а також 0 і 1 типу *integer*. Визначення типу має такий вигляд:

type *bit* **is** ('0', '1');

Значення типу bit поміщаються в одинарні лапки. Застосування логічних операторів до об'єктів типу bit здійснюється за принципом позитивної логіки.

Тип **std_ulogic** (standart unresolve logic) є розширенням типу bit у термінах багатозначної логіки (стандарт IEEE 1164) і дозволяє виконувати моделювання процесів проходження сигналів у реальних електронних схемах. Формат опису типу має такий вигляд:

type *std_ulogic* **is** ('U' ,' X ',' 0 ',' 1 ',' Z ',' W ',' L ',' H ',' - ');

Розшифровка значень типу *std_ulogic* наведена в таблиці 2.2. На практиці найчастіше використовується шестизначна логіка ('Ú', 'X', '0', '1', 'Z', '-'). При цьому невідоме значення 'X' не еквівалентне невизначеному '-', яке широко використовується під час проведення логічного синтезу і оптимізації схеми. Так, для отримання кращого результату при мінімізації булевих функцій деякі невизначені значення замінюються на "0" або "1".

Тип даних *std_ulogic* не входить в пакет standart і для його використання необхідно підключити пакет std_logic_1164 бібліотеки IEEE, записавши такий код:

library IEEE;

use IEEE. std_logic_1164. **all**;

Таблиця 2.2 – Перелік значень типу даних std_ulogic

Прийняті значення	Опис
‘U’	не ініціалізовано
‘X’	невідоме значення, сильне джерело
‘0’	сильний 0
‘1’	сильна 1
‘Z’	високий імпеданс
‘W’	невідоме значення, слабке джерело
‘L’	слабкий 0
‘H’	слабка 1
‘–’	невизначене значення

Фізичні типи даних використовуються уявлення фізичних величин. Дані фізичного типу визначаються своїм значенням і одиницею виміру. Визначення фізичного типу включає в себе первинний модуль, у якому визначається основна одиниця виміру, і (не обов'язково) вторинні модулі, в яких визначаються додаткові одиниці виміру і їх відношення до основної одиниці:

type ім'я типу **is range** діапазон значень
units
первинний модуль;
[Вторинний модуль_1;]
[Вторинний модуль_2;]
...
[Вторинний модуль_n;]
end units;

Як приклад розглянемо формат опису підтримуваного в VHDL стандартного фізичного типу даних time, призначеного для опису часу в ході моделювання цифрових схем:

type time **is range** 0 **to** 1E20
units
fs;
ps = 1000 fs;
ns = 1000 ps;
us = 1000 ns;
ms = 1000 us;
sec = 1000 ms;
min = 60 sec;
hr = 60 min;
end units;

де fs (фемтосекунда 10^{-15} с) – ім'я первинного модуля, інші одиниці (ps, ns, us, ms, sec, hr) утворюють вторинний модуль і є похідними від базової величини. Величина fs є мінімальним фізичним часом, яке може бути враховано в ході моделювання.

Розглянемо особливості арифметичних операцій з об'єктами фізичного типу даних.

- Операндами в ході виконання операцій додавання і віднімання можуть бути тільки об'єкти одного фізичного типу; результатом є величина, що має тип вхідних операндів.

- Над значенням фізичного типу можуть бути виконані операції множення або ділення на ціле або дійсне число; результатом є величина фізичного типу.

- Допустима операція ділення, при якій операнди можуть бути однакового фізичного типу; результатом операції буде величина цілого типу.

2.7 Підтипи даних

Часто в ході опису моделі доцільно працювати з об'єктами, які можуть набувати значень у суворо певній галузі з безлічі можливих значень будь-якого типу. Тип таких об'єктів можна визначити на основі базового типу. А новостворена підмножина значень базового типу називається підтипом (**subtype**).

Формат опису підтипу на основі базового типу має вигляд:

subtype *ім'я підтипу* **is** *ім'я базового типу* **range** початкове значення **to/downto** кінцеве значення;

Всі операції, застосовні до об'єктів базового типу, допустимі і для об'єктів його підтипів. Однак при переповненні результату операції виникатиме помилка.

Прикладом опису підтипу може служити оголошення підтипу *byte*, що приймає значення базового типу integer у діапазоні від 0 до 255:

subtype *byte* **is** *integer* **range** 0 **to** 255;

У VHDL введені вбудовані підтипи даних: *natural* (натуральний), *positive* (позитивний), *delay_length* (інтервал часу), *std_logic* (стандартна логіка). Перші два є похідними від базового типу integer і мають такі визначення:

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

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

де діапазон прийнятих значень зверху обмежений максимальним значенням типу *integer* за допомогою конструкції, що містить атрибут high, зазначений через апостроф після імені базового типу. Детально атрибути скалярних типів даних будуть розглянуті в підрозділі 2.12.

Наведені підтипи даних *natural* і *positive* доцільно використовувати в ході індексації елементів масивів.

Для підтипу *delay_length* базовим є фізичний тип даних *time*:

subtype *delay_length* **is** *time* **range** 0 fs **to** time'high;

Цей підтип даних містить невід'ємні значення типу *time* і введений для зручності подання часових інтервалів.

Підтип даних *std_logic* є дозволеним (**resolved**) підтипом базового типу *std_ulogic*:

subtype *std_logic* **is resolved** *std_ulogic*;

Термін *дозволений* (resolved) означає те, що для сигналів цього типу даних може існувати кілька джерел, а результуючий логічний рівень може бути отриманий за допомогою спеціальної роздільної функції. Детально ці питання розглядаються в підрозділі 3.12.

2.8 Складові типи даних

Масив є безліччю елементів однакового типу. Позиція кожного елемента масиву визначається індексом. Формат визначення типу n - вимірного масиву має такий вигляд:

type *ім'я типу масиву* **is array** (*діапазон значеній_1, ...,*

[*Діапазон значень_n*]) **of** *ім'я базового типу*;

де діапазон значень подається у вигляді конструкцій:

значення_1 **to** / **downto** *значення_2*,

якщо початкове і кінцеве значення визначені заздалегідь, і

тип індексу range <>,

якщо точний діапазон прийнятих значень заздалегідь не відомий, причому *тип індексу* визначає максимально можливі межі діапазону індексів. В останньому випадку в ході опису об'єкта обов'язково треба задати конкретні значення для визначення інтервалу індексів.

Як приклад розглянемо оголошення типу *vector* – одновимірного масиву з 100 елементів типу *integer* з діапазоном індексів від 0 до 99, і типу *matrix* – квадратної матриці з 10 рядків і 10 стовпців з діапазоном індексів від 1 до 10 і речовим типом елементів (*real*).

type *vector* **is array** (0 **to** 99) **of** *integer*;

type *matrix* **is array** (1 **to** 10, 1 **to** 10) **of** *real*;

При невідомомій заздалегідь кількості елементів опис типу matrix_2 матиме вигляд:

type *matrix_2* **is array** (*natural* **range** <>, *natural* **range** <>) **of integer**;

Елементи масиву, в свою чергу, можуть бути масивами. Так, можна оголосити масив array_1, елементи якого матимуть тип vector, оголошений в попередньому прикладі:

type *array_1* **is array** (*natural* **range** <>) **of** *vector*;

Для обробки інформації, що подається у вигляді двійкового коду, в VHDL введені стандартні рядкові типи даних: ***bit_vector*** і ***std_logic_vector***, що дозволяють компактно визначати одновимірні бітові масиви з необмеженим розміром

type *bit_vector* **is array** (*natural* **range** <>) **of** *bit*;

type *std_logic_vector* **is array** (*natural* **range** <>) **of** *std_logic*;

Згідно з наведеними формальними специфікаціями векторні об'єкти характеризуються кількістю елементів (розрядністю або шириною шини) і базовим типом елементів. Для елементів масивів ***bit_vector***, ***std_logic_vector*** базовими є типи даних ***bit*** і ***std_logic*** відповідно. Літерали типу ***bit_vector*** і ***std_logic_vector***, що становлять бітові рядки, поміщаються в подвійні лапки. Розрядність об'єкта, оголошеного як масив типу ***bit_vector*** або ***std_logic_vector***, вказується діапазоном натуральних чисел за допомогою вже відомої конструкції, укладеної в стандартні круглі дужки:

(*Значення_1* **to** / **downto** *значення_2*),

причому, тип діапазону індексів (зростаючий або спадаючий) має бути явно заданий для однозначного визначення ваги ліній (бітів) шини.

Запис (**record**) визначає складовий тип даних, що складається з ряду полів різних типів і служить для об'єднання в одну структуру різнорідної інформації. Формат оголошення запису має такий вигляд:

type *ім'я запису* **is record**

ім'я_поля_1: ім'я_типу;

[Ім’я_поля_2: ім’я_типу;]

. . .

[Ім’я_поля_n: ім’я_типу;]

end record *ім'я запису*;

Запис може складатися з одного або більше елементів різних типів. Кожен елемент характеризується своїм ім'ям і типом даних. Наприклад, розглянемо тип даних *date_timer*, що містить 16-розрядне поле *date* типу bit_vector (15 downto 0), 32-розрядне поле *timer* типу bit_vector (31 downto 0), і однорозрядне поле дозволу доступу *en* типу bit:

type *date_timer* **is record**

date: bit_vector (15 downto 0);

timer: bit_vector (31 downto 0);

en: bit;

end record *date_timer*;

2.9 Вказівний тип даних (**access**)

Скалярні та складові типи даних дозволяють працювати з об'єктами точно заданої розмірності. Однак під час розробки деяких додатків можуть виникати ситуації, коли необхідно працювати з даними, розмір яких заздалегідь не відомий, або потрібно описати складні взаємини між окремими структурами. Для цього в VHDL за аналогією з іншими мовами програмування введений вказівний тип даних **access**, опис якого виглядає так:

type *ім'я _вказівного_типу* **is access** *ім'я типу*;

Параметр *ім'я _типу* може приймати значення будь-якого типу або підтипу даних за винятком файлового і розрізнених типів даних.

Як приклад розглянемо оголошення вказівного *типу natural_pointer*, що дозволяє забезпечити доступ до об'єкта типу *natural*:

type *natural_pointer* **is access** *natural*;

Об'єкт, що належить типу *natural_pointer*, може містити значення-посилання на об'єкт типу natural, але не на об'єкти інших типів даних.

2.10 Файлові типи даних

В ході опису цифрових систем на VHDL файли можуть використовуватися для зберігання вхідних і вихідних даних, а також часто використовуваних у програмі значень. Спеціально для роботи з файлами в VHDL введений спеціальний тип даних – **file** (файл). Формат опису файлового типу наводиться нижче:

type *ім'я _файлового_типу* **is file of** *ім'я_типу_елементів_файла*;

Як приклад розглянемо оголошення файлового типу *file_1*, елементами якого є цілі числа типу *integer*:

type *file_1* **is file of** *integer*.

2.11 Класи об'єктів в VHDL

Об'єкти в VHDL служать для зберігання різних значень в рамках моделі. Для однозначності подання даних кожен об'єкт повинен мати своє власне ім'я – ідентифікатор, що записується за правилами, зазначеними у підрозділі 2.4. Всі об'єкти мають бути явно визначені перед їхнім використанням.

Кожен об'єкт характеризується **типом** і **класом**.

Тип показує, якого роду дані містить об'єкт і може бути стандартним або певним користувачем (див. підрозділ 2.5).

Клас визначає операції, які можна проводити над цими об'єктами. У мові VHDL введено 3 класи об'єктів: **константи**, **змінні** та **сигнали**. Перші два класи мають прямі аналоги у класичних мовах програмування, таких як Fortran, Pascal, C. Клас сигналів спеціально введений для моделювання цифрових схем.

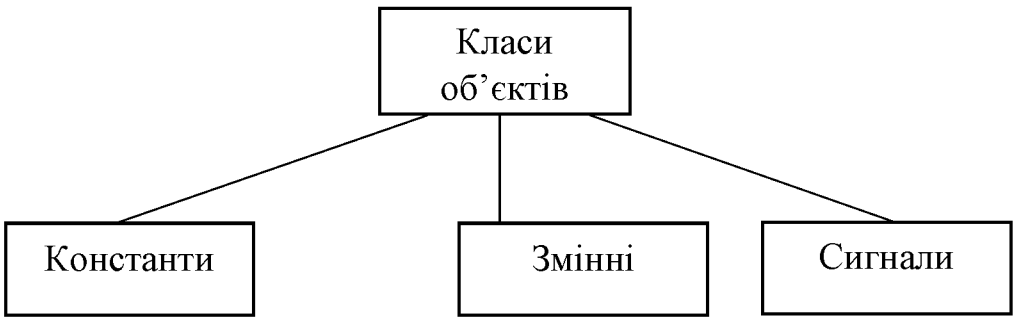


Рисунок 2.4 – Класи об'єктів мови VHDL

Константи (**constant**) – об'єкти, значення яких задається в ході опису і не змінюється в процесі роботи програми. Константи можуть мати будь-який з розглянутих у попередніх підрозділах тип даних. Формат опису константи виглядає так:

constant *ідентифікатор*: *тип*: = значення;

Наприклад, оголошення константи K типу integer, що приймає значення 10, має вигляд:

constant K: *integer*: = 10;

Змінні (**variable**) – об'єкти, значення яких можуть змінюватися в процесі виконання програми. Як і константи, змінні можуть мати будь-який з підтримуваних типів даних. Формат опису змінної:

variable *ідентифікатор*: **тип** [: = початкове значення];

Початкове значення для змінної вказувати не обов'язково. Наприклад, декларація змінної V типу bit, з присвоєнням початкового значення '1', записується так:

variable V: *bit*: = '1';

У разі оголошення кількох змінних однакового типу їх можна вказувати після ключового слова **variable** через кому:

variable *ідентифікатор_1* [..., *ідентифікатор_2*]: *тип*;

Для присвоювання значень змінним використовується складений оператор **:=**. У правій частині цього оператора присвоювання можуть записуватися вирази, що містять константи, змінні та сигнали. Наприклад, сума значень змінних A і B типу *integer* може бути присвоєна змінній C так:

variable A, B, C: *integer*;

. . .

C: = A + B;

Сигнали (**signal**) – спеціалізовані об'єкти мови VHDL, введені для опису роботи електронних елементів і, що становлять значення, передаються по лініях зв'язку. Сигнали, на відміну від констант і змінних, характеризуються не тільки значенням, а й моментом модельного часу, за якого дане значення існує. Використання цих характеристик сигналу дозволяє проводити аналіз часових залежностей в ході функціонування системи. Станом сигналу прийнято вважати два параметри: *значення сигналу / момент модельного часу*. Сигнал може змінювати своє значення за ходом модельного часу. Для моделювання реальних процесів передачі даних з використанням сигналів у VHDL вводиться поняття ***транзакції*** – внутрішньої інструкції системи моделювання зі зміни конкретного сигналу в заданий момент модельного часу. Послідовність значень сигналу, що відповідають різним моментам модельного часу протягом деякого інтервалу, формує *часову діаграму* сигналу (*waveform*).

Сигнали можуть бути еквівалентні як одиночним лініям зв'язку, по яких передається одне двійкове значення логічного рівня, так і шинним структурам, в яких передається інформація та може надаватися комбінацією деяких довічних значень (векторні сигнали) в будь-який момент часу. Сигнали в VHDL (див. рис. 2.5) поділяються на внутрішні (**signal**) і зовнішні (**port**).

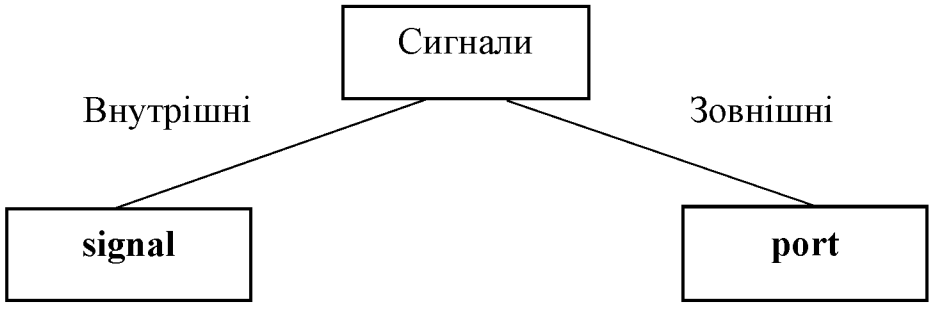


Рисунок 2.5 – Класифікація сигналів у мові VHDL

Внутрішні сигнали (**signal**) відображають лінії зв'язку всередині схеми і описуються у форматі:

signal *ідентифікатор*: *тип* [: = початкове значення];

Приклад декларації сигналу A типу *integer* з початковим значенням 100:

signal A: ***integer***: = 100;

При оголошенні декількох внутрішніх сигналів однакового типу їх можна вказувати після ключового слова **signal**, розділяючи ідентифікатори через кому:

signal *ідентифікатор_1* [..., *ідентифікатор_2*]: **тип**;

Приклад декларації сигналів A, B, C і D типу *bit*:

signal A, B, C, D: *bit*;

Зовнішні сигнали (**port**) служать для забезпечення інтерфейсу системи з зовнішнім середовищем. Зовнішні сигнали характеризуються режимом і типом і описуються після службового слова **port** у стандартних круглих дужках:

port ([**signal**] *ідентифікатор*: ***режим*** *тип*);

Ключове слово **signal** у ході опису портів, як правило, не вказують.

Режим (mode) роботи порту характеризує напрямок передачі інформації і визначає, чи є сигнал вхідним (***in***), вихідним (***out***), або двонаправленим (***inout***).

Тип (type) порту характеризує значення, які можуть бути присутніми на відповідній лінії зв'язку.

Приклад опису вхідного порту A і вихідного порту B типу ***bit*** наводиться нижче:

port (*A*: ***in*** *bit*; *B*: ***out*** *bit*);

Якщо кілька інтерфейсних сигналів мають однакові режим і тип, то їхні описи можна об'єднати в одній специфікації, розділяючи ідентифікатори через кому. Наприклад, декларація двонапрямлених портів *C, D, E і F* типу *integer* може виглядати так:

port (*C, D, E, F*: ***inout*** *bit*);

Для присвоювання значень сигналам використовується складений оператор призначення <=, який залежно від програмного контексту може виконувати паралельне або послідовне присвоєння. У правій частині цього оператора присвоювання можуть записуватися вирази, що містять константи, змінні і сигнали, за винятком вихідних портів, що мають режим роботи ***out***.

приклад:

```
port (A, B: in bit;
C, D: out bit;);
. . .
C <= A and B;      -- Коректний запис оператора призначення сигналу;
D <= not C;        -- неправильний запис: у правій частині оператора
                      призначення
                   -- сигналу записана операція заперечення над сигналом C
                   -- має режим out.
```

Для виключення такої ситуації необхідно організувати обчислення з введенням допоміжного внутрішнього сигналу або змінної.

Головною відмінністю оператора призначення сигналу <= від оператора присвоювання значення змінної **:=** є те, що зміна сигналу відбувається не в поточний момент модельного часу, а через деякий інтервал (затримку) по осі часу. Детально це питання буде розглянуто в підрозділі 3.3.

2.12 Атрибути об'єктів

У VHDL типи даних додатково характеризуються властивостями, званими **атрибутами**, які визначають параметри об'єктів, такі як межі діапазону прийнятих значень, тип індексації елементів масиву і т.д. Фактично, атрибут формується викликом спеціальної функції, що повертає його значення. Реалізація цього принципу прихована від програміста. Звернення до атрибутів об'єктів має такий синтаксис:

ім'я типу або об'єкта 'ім'я атрибута.

Зумовлені атрибути для скалярних типів даних наводяться в таблиці 2.3. Наприклад, для скалярного типу HUNDRED, що приймає значення на множині цілих чисел у діапазоні від 1 до 100, значення атрибутів будуть такими:

```
type HUNDRED is range 1 to 100;
. . .
HUNDRED'left = 1;        -- визначення лівої межі діапазону;
HUNDRED'right = 100;     -- визначення правої межі діапазону;
HUNDRED'low = 1;         -- найменше значення діапазону;
HUNDRED'high = 100;      -- найбільше значення діапазону;
HUNDRED'ascending = true; -- зростаючий діапазон.
```

Таблиця 2.3 – Атрибути числових типів даних: N_T - Name_type (ім'я типу або об'єкта)

Позначення	Опис
N_T**'left**	Величина на лівій межі інтервалу значень типу.
N_T**'right**	Величина на правій межі інтервалу значень типу.
N_T**'low**	Найменше значення в типі.
N_T**'high**	Найбільше значення в типі.
N_T**'ascending**	Величина приймає значення ***true***, якщо діапазон значень типу заданий від меншого до більшого або перерахуванням, ***false*** – у протилежному випадку.

Для фізичних типів даних та тих, що перераховуються на додаток до атрибутів, наведених у таблиці 2.3, зумовлені спеціальні атрибути, наведені в таблиці 2.4.

Таблиця 2.4 – Атрибути фізичних типів даних та тих, що перераховуються: N_T - Name_type (ім'я типу або об'єкта)

Позначення	Опис
N_T**'pos**(*value*)	Номер позиції значення *value* в ряду прийнятих значень типу N_T
N_T**'val**(*n*)	Величина на позиції *n* в ряду прийнятих значень типу N_T
N_T**'succ**(*value*)	Значення наступного за *value* елемента типу N_T
N_T**'pred**(*value*)	Значення попереднього за *value* елемента типу N_T

Значення основних атрибутів для типу CPU, що перераховуються, що приймає значення (*PENTIUM, CELERON, ATLON, DURON*), наведені нижче:

type CPU **is** (*PENTIUM, CELERON, ATLON, DURON*);

. . .

CPU'**left** = *PENTIUM*; -- визначення лівої межі діапазону;

CPU'**right** = *DURON*; -- визначення правої межі діапазону;

CPU'**pos** (*PENTIUM*) = 0; -- визначення номера позиції аргументу;

CPU'**val** (2) = *ATLON*; -- визначення значення за індексом;

CPU'**succ** (*PENTIUM*) = **CELERON**; -- наступне значення;

CPU'**pred** (*CELERON*) = **PENTIUM**; -- попереднє значення.

Для отримання параметрів масивів у VHDL зумовлені спеціальні атрибути, наведені в таблиці 2.5.

Таблиця 2.5 – Атрибути масивів: N_A - Name_Array (ім'я масиву), параметр *i* – вказує номер вимірювання (для багатовимірних масивів)

Позначення	Опис
N_A'**left(*i*)**	Величина на лівій межі індексів масиву N_A за вказаним виміром *i*
N_A'**right(*i*)**	Величина на правій межі індексів масиву N_A за вказаним виміром *i*
N_A'**low(*i*)**	Величина на нижній межі індексів масиву N_A за вказаним виміром *i*
N_A'**high(*i*)**	Величина на верхній межі індексів масиву N_A за вказаним виміром *i*
N_A'**range(*i*)**	Інтервал індексів масиву N_A за вказаним виміром *i*
N_A'**reverse_range(*i*)**	Інтервал індексів масиву N_A за вказаним виміром *i*, зазначений у зворотному порядку
N_A'**length(*i*)**	Довжина інтервалу індексів масиву N_A за вказаним виміром *i*
N_A'**ascending(*i*)**	Величина типу boolean, що приймає значення true, якщо заданий зростаючий діапазон індексів масиву N_A за вказаним виміром *i*, false – при спадному діапазоні.

Розглянемо тип MAS, що становить матрицю цілих чисел з діапазонами індексів (1 ... 10, 15 ... 0). Атрибути даного типу визначатимуть, як:

type MAS **is** array (1 to 10, 15 downto 0);

. . .

MAS'**left (1) = 1**; -- визначення лівої межі діапазону № 1;

MAS'**right (1) = 10**; -- визначення правої межі діапазону №1;

MAS'**left (2) = 15**; -- визначення лівої межі діапазону № 2;

MAS'**right (2) = 0**; -- визначення правої межі діапазону № 2;

MAS'**low (1) = 1**; -- визначення нижньої межі діапазону № 1;
MAS'**high (1) = 10**; -- визначення верхньої межі діапазону № 1;
MAS'**low (2) = 0**; -- визначення нижньої межі діапазону № 2;
MAS'**high (2) = 15**-- -- визначення верхньої межі діапазону № 1;
MAS'**range (1) = 1 to 10**; -- діапазон індексів з вимірювання № 1;
MAS'**range (2) = 15 downto 0**; -- діапазон індексів з вимірювання № 2
MAS'**length (1) = 10**; -- довжина діапазону індексів з вимірювання № 1;
MAS'**length (2) = 16** -- довжина діапазону індексів з вимірювання № 2;
MAS'**ascending (1) = true**; -- зростаючий діапазон № 1;
MAS'**ascending (2) = false** -- регресний діапазон № 2.

Як параметр у атрибутів векторних типів вказується номер вимірювання, за замовчуванням номер вимірювання вважається рівним.

Окремий клас властивостей об'єктів становлять атрибути сигналів, які використовуються для отримання інформації про події, що відбуваються із сигналами. Перелік атрибутів сигналів з описом наведено в таблиці 2.6.

Таблиця 2.6 – Атрибути сигналів: N_S – Name_Signal (ім'я сигналу), параметр *t* – вказує часовий інтервал

Позначення	Опис
N_S'**delayed** (***T***)	Сигнал, приймає значення N_S, але із затримкою на часовий інтервал T.
N_S'**stable**(***T***)	Сигнал типу *boolean*, приймає значення *true*, якщо протягом часового інтервалу **T** сигнал N_S стабільний.
N_S'**quiet(*T*)**	Сигнал типу *boolean*, приймає значення *true*, якщо протягом часового інтервалу **T** до сигналу N_S не було звернень.
N_S'**transaction**	Сигнал типу *bit*, перемикається з 0 в 1 (або з 1 в 0) при кожному зверненні до сигналу N_S.
N_S'**active**	Величина типу *boolean*, приймає значення *true*, якщо в поточному циклі моделювання виконується звернення до сигналу N_S.
N_S'**event**	Величина типу *boolean*, приймає значення *true*, якщо в поточному циклі моделювання відбувається зміна сигналу N_S.
N_S'**last_active**	Величина, що дорівнює інтервалу часу, що пройшов з моменту останнього звернення до сигналу N_S.
N_S'**last_event**	Величина, що дорівнює інтервалу часу, що пройшов з моменту останньої зміни значення сигналу N_S.
N_S'**last_value**	Величина, що дорівнює значенню сигналу N_S у попередньому циклі моделювання.

Розглянемо атрибути сигналу C (що є логічною сумою значень сигналів A і B. Сигнали C, D_5, Q_5, S_5 і T формуються таким чином:

C <= A **or** B;

D_5 <= C'**delayed** (5 ns);

Q_5 <= C'**quiet** (5 ns);

S_5 <= C'**stable** (5 ns);

T <= C'**transaction**;

Часові діаграми сигналів D_5, Q_5, S_5 і T наведені на рисунку 2.6.

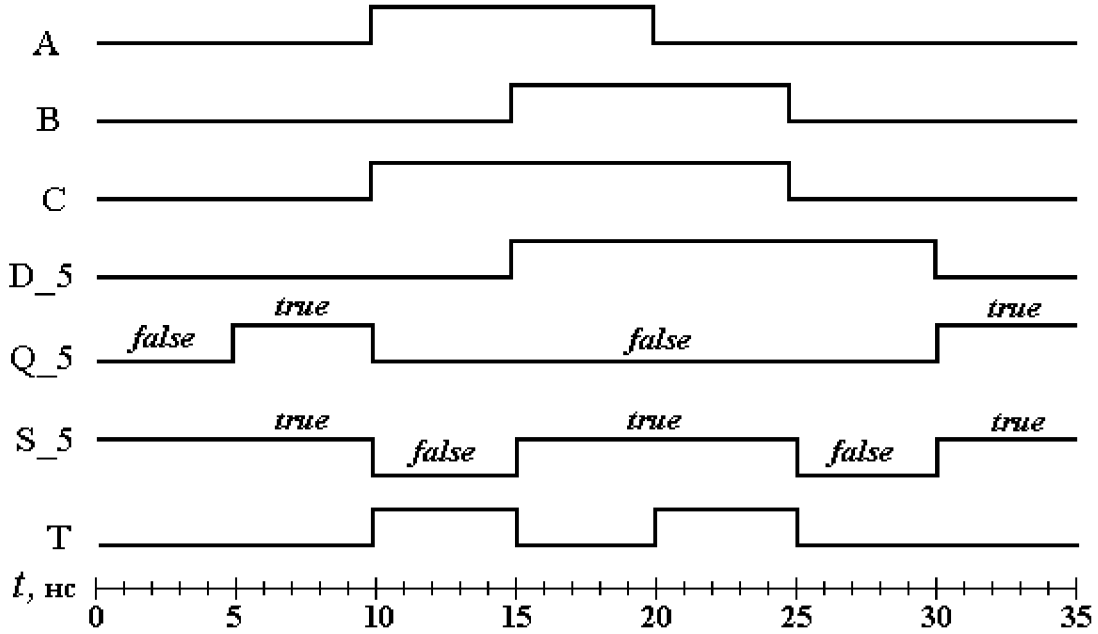


Рисунок 2.6 – Ілюстрація значень атрибутів сигналу C: D_5, Q_5, S_5, T

Атрибут C'**delayed** (5 ns) формує сигнал D_5, в точності повторює форму C, але із затримкою на 5 нс.

Значення сигналу Q_5, формованого атрибутом C'**quiet** (5 ns), дорівнюватиме *true*, якщо сигнал C не мав трансакцій протягом 5 нс., і так само *false* перші 5 нс після кожної трансакції C.

Значення сигналу S_5, формованого атрибутом C'**stable** (5 ns), дорівнює *true*, якщо сигнал C стабільний протягом інтервалу часу 5 нс., і так само *false* перші 5 нс після зміни значення C.

Атрибут C'**transaction** формує сигнал T, перемикається в момент часу, коли відбувається транзакція (обчислення нового значення) сигналу C, тобто коли відбуваються зміни сигналів A чи B, що викликають процедуру обчислення нового значення сигналу C.

2.13 Операції у виразах

Для обчислення значень сигналів у VHDL вводяться вирази, які виконують арифметичні або логічні обчислення над одним або декількома

операндами. Вирази використовуються в операторах призначення сигналу, присвоювання значення змінної, при ініціалізації об'єктів, можуть бути операндами і параметрами виклику підпрограм. Набір операцій забезпечує можливість працювати тільки з певними типами даних. В ході запису виразів для явної вказівки пріоритету операцій необхідно використовувати стандартні круглі дужки.

Операції в VHDL можна розділити на 3 групи: арифметичні; логічні, що включають операції зсуву і конкатенації; і порівняння.

Арифметичні операції застосовуються для об'єктів типів *integer*, *real* і *одновимірних бітових векторів*. В останньому випадку необхідно підключати спеціальні бібліотеки, такі як **ieee.std_logic_unsigned** і **ieee.std_logic_arith**. У виразах мають бути присутні операнди однакових типів даних, за винятком спеціально обумовлених форматом операцій (наприклад, див. підрозділ 2.6.). Перелік арифметичних операцій зі стислими описами наведений в таблиці 2.7.

приклад:

signal A, B, C, D, E, F, I, K: *integer*;

. . .

A <= (B + C) / D + I * F - K;

Порядок виконання операцій: підсумовування в дужках, ділення, множення, додавання, віднімання.

Таблиця 2.7 – Арифметичні операції в мові VHDL: R – результат, OP1 – перший операнд, OP2 – другий операнд

Позначення	Опис
R=OP1 + OP2	Додавання OP1 і OP2. Операнди і результат є числовими типами.
R= – OP	Використання унарної операції "мінус" привласнює результату інверсне значення операнда OP.
R=OP1 – OP2	Віднімання.OP2 з OP1 Операнди і результат є числовими типами.
R=OP1 * OP2	Множення OP1 на OP2. Операнди і результат є числовими типами.
R=OP1 / OP2	Розподіл OP1 / OP2. Операнди і результат є числовими типами.
R=OP1 ** OP2	Зведення OP1 в ступінь OP2. Операнди і результат є числовими типами.
R=**abs**(OP)	Отримання модуля числа OP. Операнд і результат є числовими типами.
R=OP1 **mod** OP2	Отримання залишку від ділення OP1 на OP2. Операнди і результат є числовими типами.

Якщо в правій частині арифметичних виразів присутні операнди різних числових типів, то їх можна перетворити за допомогою операторів приведення типів **real (x)** і **integer (x)**, які конвертують значення аргументу x у значення типів *real* і *integer* відповідно.

Приклад:

variable A, B: *integer*;

variable C: *real*;

. . .

A: = B + **integer** (C);

У даному прикладі в ході обчислення значення змінної A, що приймає значення цілого типу *integer*, використовується змінна C дійсного типу *real*, для перетворення значення якого у величину цілого типу застосовується функція **integer** (C).

Логічні операції мають найнижчий пріоритет під час обчислення виразів. Операндами логічних операцій можуть бути значення типів *bit*, *boolean*, і *одновимірних бітових векторів*. Не допускається використовувати операнди різних типів в одному логічному виразі. Перелік логічних операцій із стислими описами наведений в таблиці 2.8.

Для однозначного визначення порядку обчислень в логічних виразах необхідно використовувати дужки, наприклад:

signal A, B, C, D, E, F: *bit*;

. . .

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

Спочатку будуть виконані операції **and**, потім **or** і на завершення - **not**.

Таблиця 2.8 – Логічні операції в мові VHDL

Позначення	Опис
1	2
R=OP1 **and** OP2	Логічне "І". Операнди і результат можуть бути типу *boolean bit* або одновимірними бітовими масивами.
R=OP1 **or** OP2	Логічне "АБО". Операнди і результат можуть бути типу *boolean bit* або одновимірними бітовими масивами.
R=OP1 **xor** OP2	Логічне виключає "АБО". Операнди і результат можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами.
R=OP1 **nand** OP2	Логічне "І - НЕ". Операнди і результат можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами.
R=OP1 **nor** OP2	Логічне "АБО - НЕ". Операнди і результат можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами.

Продовження табл. 2.8

1	2
R= OP1 **xnor** OP2	Логічне виключає "АБО - НЕ". Операнди і результат можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами.
R= **not** OP1	Логічне заперечення. Операнд та результа можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами.

Операції зсуву визначені для бітових векторів і виконують зсув бітів вектора на число розрядів, що задаються значенням типу *integer*. Формат операцій зсуву:

тип операції ідентифікатор, кількість розрядів

Як і в багатьох мовах програмування, в VHDL присутні операції логічного, арифметичного і циклічного зрушень (див. таблицю 2.9).

Таблиця 2.9 – Операції зсуву в мові VHDL

R=OP **sll** N	Логічний зсув вліво OP на N позицій. OP і R – можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами, N – типу *integer*.
R= OP **srl** N	Логічний зсув вправо OP на N позицій. OP і R – можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами, N – типу *integer*.
R= OP **sla** N	Арифметичний зсув вліво OP на N позицій. OP і R – можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами, N – типу *integer*.
R= OP **sra** N	Арифметичний зсув вправо OP на N позицій. OP і R – можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами, N – типу *integer*.
R= OP **rol** N	Циклічний зсув вліво OP на N позицій. OP і R – можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами, N – типу *integer*.
R= OP **ror** N	Циклічний зсув вправо OP на N позицій. OP і R – можуть бути типу *boolean*, *bit* або одновимірними бітовими масивами, N – типу *integer*.
R= OP1 **&** OP2	Конкатенація (злиття) OP1 і OP 2. Операнди і результат є одновимірними масивами.

Операції циклічного зсуву виконують зсув **sll** (вліво) **srl** (вправо) бітів вектора на величину, яка подається значенням другого параметра. При цьому з протилежного боку у вектор додаються '0' (див. приклад на рис. 2.7).

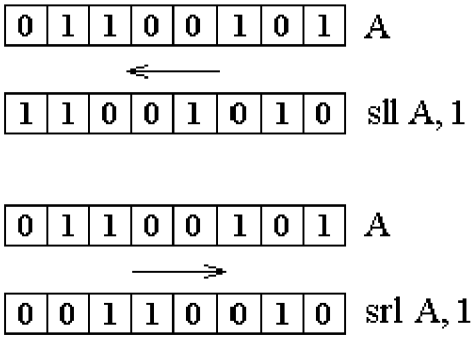


Рисунок 2.7 – Ілюстрація роботи операторів логічного зсуву над вектором A: **sll** (вліво) **srl** (вправо)

Операції арифметичного зсуву виконують зсув **sla** (вліво) **sra** (вправо) бітів вектора на величину, яка подається значенням другого параметра. При цьому з протилежного боку у вектор копіюється колишнє значення даного розряду. (див. приклад на рисунку 2.8).

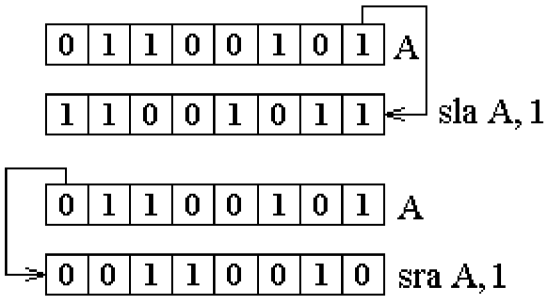


Рисунок 2.8 – Ілюстрація роботи операторів арифметичного зсуву над вектором A: **sla** (вліво) **sra** (вправо)

Операції циклічного зсуву виконують зсув **rol** (вліво) **ror** (вправо) бітів вектора на величину, яка подається значенням другого параметра. При циклічному зсуві вліво (rol) в правий біт переноситься значення крайнього лівого розряду; при циклічному зсуві вправо (ror) у лівий біт переноситься значення крайнього правого розряду (див. приклад на рис. 2.9).

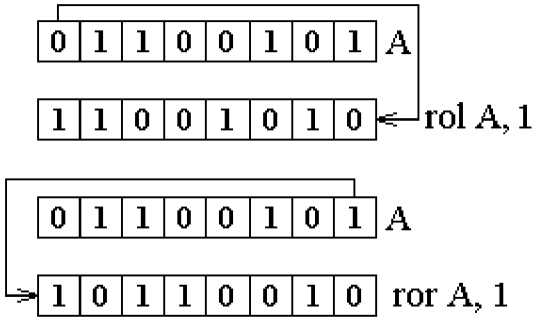


Рисунок 2.9 – Ілюстрація роботи операторів циклічного зсуву над вектором A: **sll** (вліво) **srl** (вправо)

Оператор конкатенації & (не плутати з кон'юнкцією "I") виконує злиття двох об'єктів типу бітових векторів. Як приклад розглянемо фрагмент коду:

```
C: bit_vector (7 downto 0);
A, B: bit_vector (3 downto 0);
. . .
C<=A & B;
```

Вихідні дані і результат операції наведені на рисунку 2.9.

У граничному випадку конкатенація може проводитися над об'єктами типу *bit*.

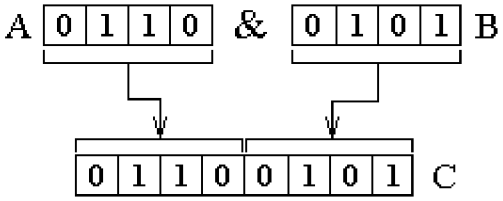


Рисунок 2.9 – Ілюстрація роботи оператора конкатенації **&** над векторами A і B

Операції порівняння виконуються над операндами однакового типу і повертають значення логічного типу *boolean*. Операції рівності = і нерівності / = виконуються над усіма типами даних, за винятком файлового. Операції порівняння>,> =, <, <= виконуються над об'єктами *integer*, *real* і *перелічуваних* типів, а також над одновимірними векторами, що складаються з елементів зазначених типів. Перелік операцій порівняння з описами наведено в таблиці 2.10.

Таблиця 2.10 – Операції порівняння в мові VHDL

Позначення	Опис
OP1 = OP2	Рівність OP1 і OP2. Операнди можуть бути будь-якого типу даних, крім **file**. Результатом операції є значення типу *boolean*.
OP1 /= OP2	Нерівність OP1 і OP2. Операнди можуть бути будь-якого типу даних, крім **file**. Результатом операції є значення типу *boolean*.
OP1 < OP2	OP1 менше OP2. Операнди можуть бути будь-якого типу даних, крім файлового. Результатом операції є значення типу *boolean*.
OP1 <= OP2	OP1 менше або дорівнює OP2. Операнди можуть бути будь-якого типу даних, крім **file**. Результатом операції є значення типу *boolean*.
OP1 > OP2	OP1 більше або дорівнює OP2. Операнди можуть бути будь-якого типу даних, крім **file**. Результатом операції є значення типу *boolean*.
OP1 >= OP2	OP1 більше або дорівнює OP2. Операнди можуть бути будь-якого типу даних, крім **file**. Результатом операції є значення типу *boolean*.

Під час порівняння елементів перелічуваних типів порівнювати не самі значення, а їх порядкові номери (див. підрозділ 2.6). Так, елемент, що стоїть в ряду правіше (старший), вважається великим. Наприклад, для фрагмента:

type *control_2* **is** (EN, STROBE, BUSY, ACK);

variable A, B: *control_2*;

. . .

A: = STROBE;

B: = ACK;

значення умовного виразу **B> A** дорівнюватиме *true*, оскільки змінна B містить значення ACK з порядковим номером 3, а змінна A – значення STROBE з порядковим номером 1.

Під час порівняння векторних об'єктів попарно порівнюються елементи векторів у напрямку зліва направо. Якщо пара елементів неоднакова, то вектор з великим значенням елемента вважається, відповідно, великим. Якщо відповідні елементи ідентичні, то розглядається наступна пара елементів. Наприклад, під час порівняння бітових векторів:

variable A, B: *bit_vector* (7 **downto** 0);

. . .

A: = "10111011";

B: = "10101011";

Результат A> B дорівнюватиме *true*, оскільки у четвертому розряді вектор A містить '1', а вектор B - "0".

Для використання розглянутих операцій над об'єктами довільних (нестандартних) типів даних у VHDL застосовується механізм **перевантаження** операцій. При цьому дії, що виконуються при виклику операції, залежать від типу операндів. Так, для більшості операцій знайдуться еквівалентні функції з такою ж назвою, певні в таких стандартних пакетах, як **IEEE.std_arith**, **IEEE.numeric_std**. Дані бібліотеки дозволяють виконувати операції над об'єктами типів, похідних від std_ulogic, таких як std_logic і std_logic_vector.

2.14 Основні прийоми роботи з векторними типами даних

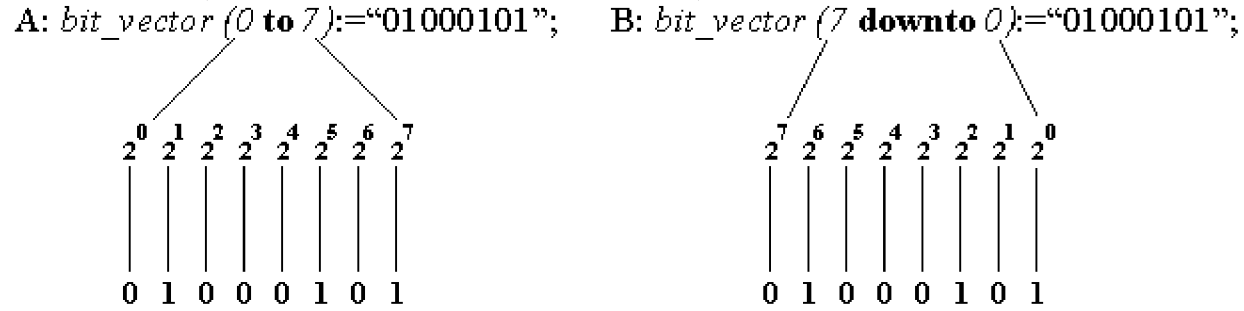
Для опису шинних стуктур в VHDL широко використовуються векторні типи даних bit_vector і std_logic_vector. Останній є найбільш універсальним (див. підрозділ 2.6), а тип bit_vector використовується для опису простих функціональних моделей. Надалі загальні питання роботи з бітовими векторами розглядатимуться на основі типу bit_vector, а специфічні – на основі тип даних std_logic_vector.

Інформація про вагу кожного розряду вектора міститься в завданні діапазону індексів. Розглянемо опис восьмирозрядних векторних сигналів A і B, яким задані однакові початкові значення у вигляді бітових рядків:

signal A: *bit_vector* (0 **to** 7): = "01000101";

signal B: *bit_vector* (7 **downto** 0): = "01000101";

На перший погляд оголошення сигналів A і B ідентичні, але в VHDL істотне значення має порядок завдання діапазону індексів, який вказує на вагу розрядів. У разі використання зростаючого діапазону (0 **to** 7) записується зліва нульовий біт, що є молодшим розрядом вектора, а крайній правий біт – старшим розрядом (див. рис. 2.10, а). В ході використання спадного діапазону (7 **downto** 0) нульовий біт, що записується праворуч, є молодшим розрядом вектора, а крайній зліва біт – старшим розрядом (див. рис. 2.10, б). У позиційних системах числення прийнято записувати літерали, вага розрядів яких спадає в напрямку зліва направо. Тому форма подання вектора зі зменшенням діапазону (див. рис. 2.10, б) є більш наочною і коректною. Помилка при виборі напрямку завдання діапазону індексів призводить до важковловимого побічного ефекту, що полягає в неправильній інтерпретації числових значень: так, для вектора A (див. рис. 2.10, а), заданого за допомогою зростаючого діапазону, десятковий еквівалент числа дорівнює 162 ("01000101" $\rightarrow$ $10100010_2 = 162_{10}$), а для вектора B (див. рис. 2.10, б), заданого за допомогою спадного діапазону, десятковий еквівалент числа дорівнює 69 ($01000101_2 \rightarrow 01000101 = 69_{10}$).



а) б)

Рисунок 2.10 – Ілюстрація вказівки а) зростаючого і б) спадного діапазонів для векторних типів даних *bit_vector*, *std_logic_vector*

Задати значення об'єктів векторних типів можна або шляхом привласнення об'єкту літерала, зазначеного в подвійні лапки, або поелементно – шляхом привласнення кожному біту рядки значення, відповідного базового типу (*bit* або *std_logic*) і вміщеного в одинарні лапки. При зверненні до окремих бітів вектора вказується його ідентифікатор і номер біта, взятий у круглі дужки. Наступні форми запису присвоювання значень вектору еквівалентні:

signal A: *bit_vector* (3 **downto** 0); **signal** A: *bit_vector* (3 **downto** 0);

. . . . . .

A <= "тисяча сто один"; A (0) <= '1';

A (1) <= '0';
A (2) <= '1';
A (3) <= '1';

Окремим бітам вектора можна присвоювати значення об'єктів базових типів (*bit*, *std_logic*). Приклад:

signal A: bit_vector (2 **downto** 0);
signal B, C, D: bit;
. . .
A (0) <= B;
A (1) <= C;
A (2) <= D;

У даному прикладі нульовому, першому і другому бітам вектора A присвоюються значення сигналів B, C і D. Можливо і зворотне присвоювання. При цьому з метою скорочення запису можна використовувати агрегати – операції над наборами об'єктів простих типів, взятих у круглі дужки. Наступні форми привласнення еквівалентні:

Стандартне побітове присвоювання	використання агрегату
signal A: *bit_vector* (3 **downto** 0);	**signal** A: *bit_vector* (3 **downto** 0);
signal B, C, D, E: *bit*;	**signal** B, C, D, E: *bit*;
. . .	. . .
B <= A (0);	(B, C, D, E) <= A;
C <= A (1);	
D <= A (2);	
E <= A (3);	

Під час присвоєння значень одного вектора іншому і виконання логічних (побітових) операцій, за винятком конкатенації, операнди повинні мати однакову довжину. Приклади правильних і помилкових операцій призначення сигналів у ході виконання простих і логічних операцій над векторними об'єктами наводяться нижче:

signal A, B, C: *bit_vector* (7 **downto** 0);
signal D, E: *bit_vector* (3 **downto** 0);
A <= B; -- правильне призначення сигналу;
D <= A; -- помилка, невідповідність розрядності приймача і джерела;
A <= B **and** C; -- коректний запис логічної операції;
A <= B **or** E; -- помилка, невідповідність розрядності операндів;

У деяких випадках необхідно призначити значення тільки частині розрядів вектора (сектору). Ця операція виконується за допомогою конструкції, що містить ідентифікатор вектора і діапазон індексів, вказаний в круглих дужках за допомогою зарезервованих слів **to / downto**. Приклад:

signal A: *bit_vector* (7 **downto** 0);
signal B, C: *bit_vector* (3 **downto** 0);
. . .
A (3 **downto** 0) <= B **and** C;
A (7 **downto** 4) <= "1011";

Універсальне присвоювання значень векторних об'єктів можна виконати на базі агрегатних описів. При цьому елементам векторів ставляться у відповідність значення об'єктів простих типів. Якщо привласнення відбувається в порядку нумерації елементів вектора, то така відповідність називається позиційною. Якщо відбувається поіменне присвоювання значень кожному елементу вектора, то має місце ключова відповідність. Допускається використовувати змішані форми прив'язки об'єктів, в яких застосовується як ключові, так і позиційні відповідності. Зарезервоване слово **others** позначає інші (які не вказані) елементи вектора і має використовуватися останнім у списку відповідностей. Розглянемо приклади:

Стандартне присвоєння початкового значення за допомогою бітового рядка:

variable A: *bit_vector* (7 **downto** 0): = "00010111";

Використання позиційної відповідності:

variable A: *bit_vector* (7 **downto** 0): = ('0', '0', '0', '1', '0', '1', '1', '1');

Використання ключової відповідності (варіант № 1):

variable A: *bit_vector* (7 **downto** 0): = (0 | 1 | 2 | 4 => '1', 3 | 5 | 6 | 7 => '0',);

Використання ключової відповідності (варіант № 2):

variable A: *bit_vector* (7 **downto** 0): = (2 **downto** 0 => '1', 4 => '1', **others** => '0');

Використання ключової відповідності (варіант № 3):

variable A: *bit_vector* (7 **downto** 0): = (0 | 1 | 2 | 4 => '1', **others** => '0',);

Використання змішаної форми відповідності:

variable A: *bit_vector* (7 **downto** 0): = ('0', '0', '0', '1', **others** => '0',);

Значення векторних об'єктів, що мають однакову розрядність, але різний індексний порядок, можна привласнювати один одному. При цьому порядок проходження бітів буде змінений на зворотний. Приклад:

signal A: *bit_vector* (7 **downto** 0): = "00010101";

signal B: *bit_vector* (0 **to** 7);

. . .

B <= A;

У результаті операції призначення сигналу вектору B буде присвоєно значення "10101000".

2.15 Принципи роботи з багатовимірними масивами

Для опису наборів елементів однакового типу використовуються об'єкти, оголошені як масиви (див. підрозділ 2.8). Для роботи з масивом необхідно:

- визначити тип масиву (діапазони індексів, базовий тип);
- оголосити об'єкт даного типу;
- задати значення елементів масиву.

Описи діапазонів індексів відокремлюються один від одного комами. Звернення до елементу багатовимірного масиву здійснюється шляхом зазначення імені об'єкта і всіх індексів, укладених у круглі дужки.

Наприклад, для оголошення сигналу M, що є двовимірним масивом цілих чисел з діапазонами індексів 0..4 і 0..2 можна задати тип matrix, а потім описати M, як:

type *matrix* **is array** (0 **to** 4, 0 **to** 2) **of** *integer*;

signal M: *matrix*: = ((1,2,3), (4,5,6), (7, 8, 9), (10,11,12), (13,14,15));

Початкові значення елементів масиву M присвоюються стандартним чином після складеного оператора **:=** шляхом завдання відповідностей, розглянутих у попередньому розділі.

У результаті масив M матиме такий вигляд:

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

З огляду на наведений розподіл, елемент масиву M (3,2) знаходиться на перетині 4-го рядка і 3 стовпці і відповідно дорівнює 8.

Присвоєння значення в програмі елементу такого масиву матиме вигляд:

M (1,2) <= 100;

причому складовою оператор <= показує, що M належить до класу сигналів (***signals***).

Карта розподілу пам'яті під елементи даного масиву наведена на рис. 2.11. Елементи зберігаються в пам'яті по рядках: при фіксованому значенні номера рядка (індекс, що вказується зліва) зберігаються в пам'яті елементи з послідовно збільшуючими номерами стовпця (індекс, що вказується праворуч). Наведений послідовний принцип розподілу пам'яті легко узагальнити на масиви, що мають розмірність більше двох.

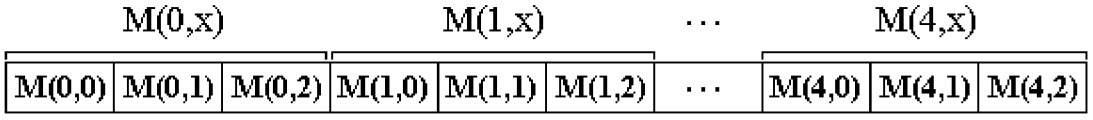


Рисунок 2.11 – Ілюстрація розподілу пам'яті під елементи масиву M (0..4, 0..2).

Робота з багатовимірними масивами підтримують не всі компіляторами VHDL, наприклад, Foundation Express. Для усунення цієї проблеми необхідно описувати типи одновимірних масивів, елементи яких, у свою чергу, будуть масивами. Наприклад, наведений вище двовимірний масив M (0..4, 0..2) можна описати так:

type *vector* **is array** (0 **to** 2) **of** *integer*;

type *matrix* **is array** (0 **to** 4) **of** *vector*;

signal M: *matrix*: = ((1,2,3), (4,5,6), (7, 8, 9), (10,11,12), (13,14,15));

Звернення до елементу цього масиву матиме вигляд:

M (4) (2) <= 20.

Розподіл пам'яті під елементи даного масиву і, відповідно, процедура присвоєння початкових значень виконуватимуться так само, як і при прямому оголошенні двовимірного масиву. Область значень першого індексу масиву M перебуватиме в діапазоні 0..4, другого – в діапазоні 0...2.

При оголошенні типу масиву його розмір (діапазони індексів) можна залишити невизначеним. У цьому випадку опис об'єкта даного типу має містити конкретні значення кордонів діапазонів індексів.

Приклад оголошення розглянутого вище двовимірного масиву M (0..4, 0..2), тип якого описаний з невизначеними значеннями діапазонів індексів наведено нижче:

type *matrix* **is array** (*natural* **range** <>, *natural* **range** <>) **of** *integer*;

signal M: *matrix* (0 **to** 4, 0 **to** 2): = ((1,2,3), (4,5,6), (7, 8, 9), (10,11,12), (13 , 14,15));

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Перелічіть і поясніть основні можливості мови VHDL.
2. Що включають в себе поняття загальноалгоритмічних і проблемно-орієнтованих конструкцій мови VHDL?
3. Перелічіть основні лексичні елементи мови VHDL.
4. Наведіть основні правила запису ідентифікаторів і літералів у мові VHDL.
5. Охарактеризуйте поняття типу даних у мові VHDL. Синтаксис оголошення типу даних.
6. Поясніть класифікацію типів даних у мові VHDL.
7. Прості типи даних. Який принципи використання стандартних числових типів даних?
8. Прості типи даних. Який принцип роботи з переліком і фізичними типами даних.
9. Охарактеризуйте поняття підтипу даних у мові VHDL. Синтаксис оголошення, принципи використання підтипів даних.
10. Перелічіть складні типи даних. Синтаксис оголошення, принципи роботи з масивами і записами в мові VHDL.
11. Який принцип роботи з вказівним і файловим типами даних у мові VHDL.
12. Перелічіть основні класи об'єктів в мові VHDL. Синтаксис оголошення констант і змінних.
13. Охарактеризуйте поняття сигналу в мові VHDL.

14. Перелічіть онутрішні і зовнішні сигнали. Правила оголошення та використання.

15. У чому основна відмінність сигналів від змінних у мові VHDL?

16. Перелічіть атрибути об'єктів. Механізм роботи, принципи використання.

17. Операції у виразах. Наведіть і поясніть основні арифметичні і логічні операції.

18. У чому полягають відмінності між групами операторів зсуву?

19. Операції порівняння. Поясніть механізм порівняння векторних об'єктів.

20. Поясніть основні принципи роботи з векторними типами даних.

21. Як вказівки діапазону індексів впливають на вагу розрядів об'єкта векторного типу даних?

22. Поясніть правила оголошення та принципи роботи з багатовимірними масивами.