3 ОСНОВНІ ОПЕРАТОРИ МОВИ VHDL

3.1 Основи функціонування апаратно-орієнтованої частини алгоритмічного ядра мови VHDL

Алгоритмічне ядро VHDL ґрунтується на принципах функціонування паралельної мультипроцесорної моделі обчислень. Нижній рівень моделі становить віртуальний процесорний елемент ВПЕ, а верхній – сукупність ВПЕ, які обмінюються інформацією за допомогою запрограмованої системи міжпроцесорних зв'язків. До ВПЕ належать арифметично-логічний пристрій АЛП, пам'ять програми ОЗУП, пам'ять даних ОЗУД, джерела ІС і приймачі ПС сигналів (див. рис. 3.1).

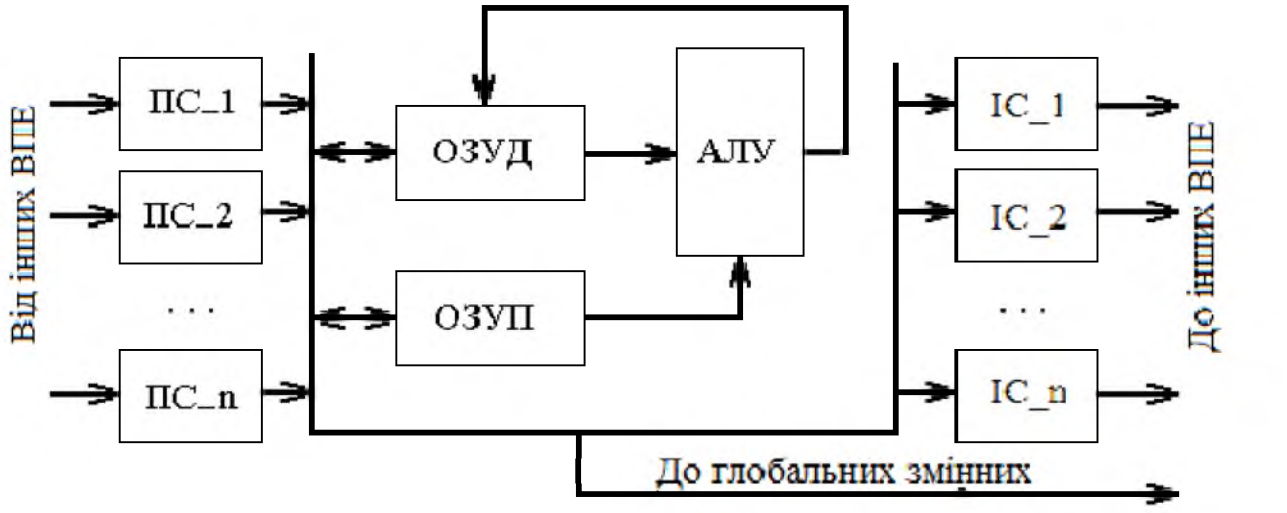


Рисунок 3.1 – Структурна схема віртуального процесорного елемента ВПЕ

АЛУ виконує стандартний набір арифметичних і логічних операцій для різних типів даних з урахуванням розрядності операндів і багатозначного логічного подання розрядів. У ході виконання операцій відбувається перевірка коректності результату з фіксацією помилок (поділ на нуль, вихід за межі діапазону, подання числа і т.д.).

В ОЗУП зберігається код програми у вигляді сукупності команд, які виконуються послідовно, і оператори, що зупиняють процес обчислень.

В ОЗУД міститься значення локальних змінних.

Джерела і приймачі сигналів (ІС, ПС) служать для зв'язку ВПЕ із зовнішнім світом. Причому, джерела генерують сигнали тільки в момент зупинки ВПЕ.

Для передачі значень в довільні моменти часу використовується механізм глобальних, поділюваних (shared) змінних, доступних для використання у всіх ВПЕ. Ці дані надходять у ВПЕ по спеціальній шині глобальних змінних.

Таким чином, верхній рівень моделі обчислювача VHDL складає архітектура, заснована на безлічі ВПЕ, об'єднаних зв'язками, по яких передаються сигнали, і загальна шина даних для передачі глобальних змінних (див. рис. 3.2).

Кожен ВПЕ виробляє виконання послідовної програми, що називається, в рамках моделі обчислювача VHDL, процесом. Відповідно, кількість ВПЕ дорівнює кількості процесів. Основними принципами функціонування процесів є:

- паралельність – всі процеси виконуються паралельно;
- одночасно виконуються процеси, що утворюють фронт хвилі запусків процесів, що пересувається з деяким кроком, що називається дельта-затримкою;
- паралельні оператори VHDL перетворюються в еквівалентні оператори процесів;
- поза тілом оператора процесу, змінні, за винятком глобальних, не доступні.

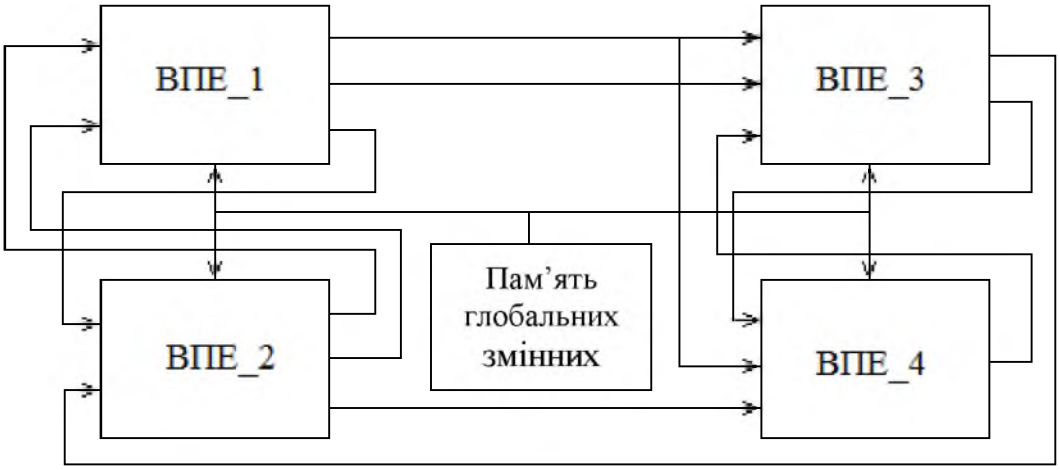


Рисунок 3.2 – Ілюстрація взаємодії декількох ВПЕ

Реалізацію алгоритмів на даній моделі доцільно проводити на спеціалізованій обчислювальній системі, в якій кожен ВПЕ виконується на спеціально виділеному процесорі. Ідеалізована модель обчислювача має бути реалізована на мультипроцесорній системі з кількістю процесорів, що дорівнює кількості процесів. Даний підхід концептуально простий, але вимагає надзвичайно високих апаратних ресурсів. Тому існуючі VHDL симулятори реалізуються на однопроцесорних ПЕОМ, у яких всі обчислювальні процеси по черзі розділяють один і той самий апаратний ресурс – центральний процесорний пристрій.

Розглянемо детально принципи функціонування VHDL симулятора.

Всі процеси, описані в VHDL програмі, можуть перебувати в трьох різних станах: сплячі, готові до виконання і такі, що виконуються.

До категорії сплячих належать процеси, для виконання яких немає готових вхідних даних. Ці процеси знаходяться в стані очікування зміни одного з вхідних сигналів, тобто в сплячому режимі.

В ході зміни одного з вхідних сигналів сплячий процес переводиться в розряд готових до виконання процесів, що утворюють чергу, з якої один з готових процесів (обраний довільним чином) пересилається в мікропроцесор для виконання.

Виконуваний процес виконує свої оператори згідно з можливостями функціонування ВПЕ на обчислювальних ресурсах мікропроцесора. При досягненні оператора зупинки процесу (**wait**) процес зупиняється. Сформовані ним сигнали запам'ятовуються і пересилаються іншим процесам. Після цього даний процес пересилається в чергу сплячих процесів. Під час надходження нових значень вхідних сигналів цикл роботи процесу повторюється.

Ядро симулятора VHDL є особливий виконуваний процес, який є диспетчером для інших процесів: веде лічильник часу моделювання, формує черги сплячих і готових до виконання процесів, вибирає процеси для виконання, передає сигнали від виконуваного процесу до решти, обслуговує звернення до дискової пам'яті і пам'яті глобальних змінних.

Під час розробки VHDL – описів необхідно враховувати фактори, що обмежують алгоритмічну складність і швидкодію моделей:

- максимальна кількість процесів у компільованій програмі визначається обсягом оперативної пам'яті обчислювальної системи;

- при заміні одного процесу іншим відбувається зміна контексту, яка може займати кілька тисяч тактів. Виходячи з цього, оптимізація моделі за критерієм швидкодії полягає в зменшенні кількості процесів;

- час роботи ядра симулятора нелінійно зростає зі збільшенням кількості процесів і сигналів, які активізують роботу процесів.

В ході синтезу VHDL-програми відбувається взаємно-однозначне відображення програмної моделі пристрою, в апаратну модель. Це означає, що кожному віртуальному ВПЕ ставиться у відповідність деякий спеціалізований процесорний елемент СПЕ, який реалізує функції, описані в програмі ВПЕ.

За принципом присвоювання значень сигналам розрізнюють два СПЕ: комбінаційні СПЕ і СПЕ з пам'яттю. Як випливає з назви, комбінаційні СПЕ є комбінаційна схема, стани виходів якої є логічними функціями від станів її входів. СПЕ з пам'яттю додатково до комбінаційної логіки мають в своїй структурі регістри для зберігання результатів і проміжних значень. Значення на виходах СПЕ є логічною функцією не тільки від станів входів, а й від даних, що зберігаються в регістрах. СПЕ з пам'яттю утворюються, якщо при виклику оператора процесу не всім сигналам або змінним присвоюються нові значення. Такі об'єкти мають зберігати свої попередні стани в елементах пам'яті.

Кожному паралельному оператору VHDL в ході синтезу ставиться у відповідність логічна схема, що виконує функції цього оператора. Так, оператори паралельного призначення сигналу, що реалізують операції

A <= **not** B, і C <= A + B синтезуються в інвертор і суматор відповідно (див. рис. 3.3). Розрядність наведених схемних елементів автоматично визначатиметься розрядністю вхідних даних. При переході до апаратної моделі час (затримка) реакції системи на вхідні дії є величиною, яка визначається характеристиками елементного базису, а не затримку, яка визначається програмним способом.

а) б)

Рисунок 3.3 – Ілюстрація синтезу паралельних операторів:
а) A <= **not** B; і б) C <= A + B.

Незважаючи на гадану простоту відображення програмних конструкцій на апаратні ресурси, в ході синтезу VHDL - коду накладається ряд обмежень, пов'язаних з особливостями елементної бази ПЛІС. Елементи і конструкції мови, які не мають відповідних еквівалентів у схемотехнічному базисі, утворюють несинтезовану безліч операторів VHDL. Так, сьогодні сигнали і змінні типу з плаваючою комою, глобальні змінні, файли, динамічні об'єкти, оператори виведення повідомлень на консоль (**assert**, **report**) і ряд інших не можуть бути безпосередньо відображені в апаратуру.

Деякі значення типу *std_ulogic* ('U', 'X', '-') також не можуть бути подані на схемному рівні, а інші 'H', 'L' відображаються в логічні "1" і "0" відповідно.

Стиль програмування також відіграє істотну роль в ході синтезу. Так, не можуть бути синтезовані такі алгоритмічні конструкції, для яких неможливо визначити конкретну комбінаційну схему на етапі компіляції. Наприклад, якщо для оператора циклу, який відображається в багаторівневу комбінаційну схему, на період компіляції не визначено кількість ітерацій, то кількість елементів, що входять до складу синтезованого схемотехнічного блоку, також не може бути визначено.

Стилем для синтезу є такий принцип складання VHDL-описів, при дотриманні якого всі алгоритмічні конструкції можуть бути відображені в апаратуру. У загальному випадку необхідно враховувати, що існує кілька поширених компіляторів-синтезаторів VHDL (Active HDL, Foundation Express, Orcad Express, Leonardo), розроблених різними фірмами-виробниками. Набір несинтезованих операторів VHDL істотно залежить від компілятора-синтезатора і, як правило, скорочується при переході до більш нових версій. Додатково САПР характеризуються наборами атрибутів, які керують синтезом, бібліотеками процедур і функцій, можливостями оптимізації та адаптації до конкретної елементної бази.

3.2 Основи синтаксису мови VHDL. Поняття об'єкта моделювання

Вихідний модуль програми мовою VHDL складається з послідовностей операторів, що описують функціонування схеми, записаних з урахуванням таких правил:

- кожен оператор є послідовністю слів, що містять літери англійського алфавіту, цифри і знаки пунктуації;
- слова поділяються будь-якою кількістю пробілів, символів табуляції і переведення рядка;
- оператори поділяються символами ';';
- коментарі в тексті програми відокремлюються послідовністю з двох символів '-';
- в конструкціях мови допустимо використання трьох класів об'єктів: констант, змінних, сигналів, які можуть набувати значень точно визначених типів;
- безпосередній обмін даними між об'єктами різних типів неможливий (необхідно використовувати функції перетворення типів).

Необхідно відзначити, що під час розробки VHDL - програм, алгоритмічні конструкції мають бути точно співвіднесені з реальною роботою схемних модулів і формованими сигналами. Цей принцип розробки апаратно-орієнтованих архітектур є найбільш фундаментальним для створення діючих VHDL - описів.

3.3 Структура опису об'єкта моделювання на VHDL

Під **об'єктом моделювання** в мові VHDL розуміють закінчене алгоритмічне подання деякої цифрової системи, що включає описи інтерфейсу і архітектури. Узагальнена структура опису об'єкта моделювання наведена на рисунку 3.4.

В інтерфейсній частині визначаються параметри налаштування, а також описуються входи і виходи модельованої цифрової системи.

У розділі архітектури міститься VHDL-опис функціонування об'єкта моделювання. Опис архітектури об'єкта може бути виконано на структурному або поведінковому (функціональному) рівнях. У деклараційній частині архітектури описуються об'єкти, підпрограми та компоненти, які є внутрішніми для об'єкта моделювання.

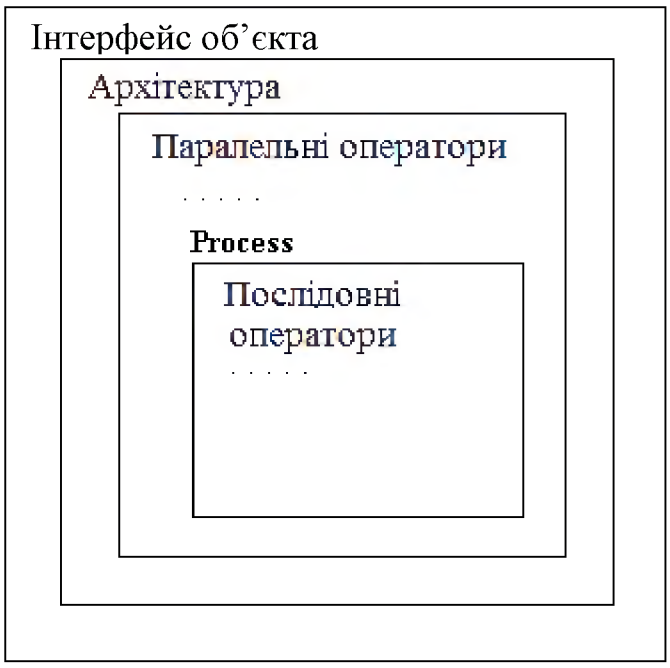


Рисунок 3.4 – Структура опису об'єкта моделювання

У реальній цифровій системі безліч компонентів одночасно формують значення певних сигналів. Тому, в основу алгоритмічного ядра VHDL покладені принципи безперервності та паралельності функціонування цифрових пристроїв. Відповідно, у виконавчій частині архітектури міститься опис роботи об'єкта моделювання у вигляді паралельних конструкцій мови VHDL.

Приклад:

signal A, B, C: *bit*;

. . .

A <= **not** B after 5 ns;

B <= **not** C after 10 ns;

На відміну від традиційних мов програмування, заснованих за принципом послідовного виконання програми, наведений фрагмент коду, який складався з паралельних операторів призначення сигналу, виконуватиметься так: при зміні сигналу C, сигналу B буде присвоєно нове значення через 10 нс; відповідно сигнал A, що залежить від B, отримає нове значення через 5 нс після зміни сигналу B (через 15 нс після зміни сигналу C).

У багатьох випадках функціонування складних обчислювальних блоків зручно описувати за допомогою послідовних алгоритмічних конструкцій, що включають оператори умовного переходу (**if**), вибору (**case**), циклу (**loop**) і т.д. Тому в VHDL введений спеціалізований паралельний оператор **process**, що дозволяє описувати паралельні процеси, які містять програмний код, що виконується послідовно.

3.4 Опис інтерфейсу об'єкта моделювання

Інтерфейс об'єкта моделювання описується в розділі **entity** і задає специфікацію параметрів настройки (**generic**) і зовнішніх (вхідних і вихідних) сигналів (**ports**). На даному етапі об'єкт моделювання подається "чорним ящиком" із зазначенням характеристик інтерфейсу. Формат опису інтерфейсної частини об'єкта моделювання:

entity *ім'я об'єкта моделювання* **is**
[generic (список параметрів, що настроюються); **]**
[port (список зовнішніх сигналів); **]**
end *ім'я об'єкта моделювання*;

Після ключового слова **entity** вказується ідентифікатор – ім'я об'єкта моделювання, яке має бути унікальним у рамках проекту.

Параметри настройки (**generic**) визначають наявність в схемному об'єкті керуючих входів, за допомогою яких може проводитися настройка примірників об'єктів залежно від конкретних параметрів. Таким чином можуть зазначатися часові затримки, розміри внутрішніх буферів, розрядність шин і т.д. Механізм параметрів, що настроюються, дозволяє адаптувати розроблені VHDL-описи цифрових систем до умов конкретного застосування.

Розглянемо приклад: нехай розроблений проект деякого пристрою з затримкою спрацьовування вентилів 5 нс. Припустимо, що при переході на нову технологію, що забезпечує більш високу швидкодію, затримка спрацьовування вентилів зменшиться до 3 нс. Якщо параметри затримки в VHDL - описі пристрою задані як стандартні константи (**constant**), то необхідно буде змінити їх значення у всіх деклараціях, що в проектах з розвиненою ієрархічною структурою виконати вкрай складно. Використання ж параметрів, що настроюються (**generic**), дозволяє створювати проекти, функціонування яких залежить від цих узагальнених констант, що змінюють значення залежно від конкретної реалізації.

Необхідно зазначити, що настроювальні параметри (**generic**) є константними значеннями. Їх можна ставити тільки в інтерфейсній частині опису об'єкта моделювання і не можна змінювати в ході виконання програми. Механізм використання параметрів, що настроюються (**generic**) в архітектурі об'єкта моделювання, буде детально розглянуто в наступних підрозділах.

Як було показано в підрозділі 2.11, зовнішні вхідні і вихідні сигнали об'єкта моделювання називають портами. У секції **port** опису інтерфейсу об'єкта моделювання в круглих дужках мають розташовуватися декларації всіх портів із зазначенням режиму роботи і типу переданих даних. Типи даних мають бути заздалегідь відомі: описані до декларації **entity** вище по тексту, або внесені в окремий бібліотечний файл, що підключається перед описом інтерфейсу об'єкта моделювання. Декларації портів з різними характеристиками записуються через роздільник '**;**'. Ідентифікатори декількох портів з однаковими специфікаціями можуть розташовуватися в одному описі через кому.

Приклад VHDL-опису інтерфейсу об'єкта моделювання *shem_1*, що зображений на рис. 3.5:

entity *shem_1* **is**
port (A, B, C, D: **in** *bit*; E, F: **out** *bit*);
end *shem_1*;

Оскільки в позначенні пристрою не вказані настроювальні константи, то в описі інтерфейсу об'єкта секція **generic** відсутня. Пристрій виконує логічну функцію or над парами вхідних сигналів. Тому доцільно (за відсутності прямих вказівок) декларувати інтерфейсні сигнали об'єкта моделювання, як такі, що двійковий тип *bit*, ('0', '1').

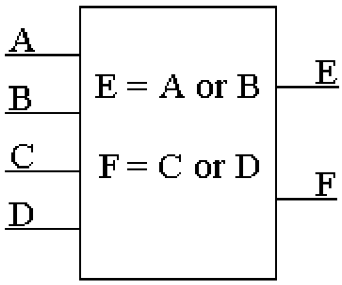


Рисунок 3.5 – Умовне позначення логічного пристрою

Необхідно зазначити, що секції **generic** і **port** не є обов'язковими. В об'єкті моделювання, в загальному випадку, можуть бути відсутніми настроювані параметри і / або інтерфейсні сигнали.

3.5 Особливості архітектури об'єкта моделювання

У мові VHDL термін архітектура (**architecture**) визначає принципи реалізації об'єкта моделювання. Таким чином, в архітектурному тілі розкривається функціональна сутність об'єкта моделювання, поданого "чорним ящиком" в інтерфейсній частині **entity**. Способи (стилі) завдання специфікації функціональної і часової роботи: структурний, поведінковий і потоковий, будуть детально розглянуті нижче. Формат подання архітектурного тіла об'єкта моделювання має такий вигляд:

architecture *ім'я архітектури* **of** *ім'я об'єкта моделювання* **is**
[Деклараційний блок]
begin
виконуваний блок (паралельні оператори VHDL);
end [**architecture**] [*ім'я архітектури*];

Після службового слова *architecture* вказується унікальний ідентифікатор *ім'я архітектури*. Специфікатор *ім'я об'єкта моделювання* дозволяє зв'язати опис архітектури з інтерфейсом об'єкта моделювання, заданому в частині **entity**. У деклараційного блоці архітектури допустимо описувати:

- типи даних (**type**);

- константи (**constant**);
- внутрішні сигнали (**signal**);
- глобальні змінні (**shared variable**);
- компоненти (**component**);
- процедури (**procedure**);
- функції (**function**).

Описи типів даних, констант, внутрішніх сигналів і глобальних змінних наводяться у відповідних форматах, розглянутих у попередньому розділі.

Для структурного уявлення архітектури об'єкта моделювання необхідно описати склад вхідних до нього компонентів, кожен з яких, в сою чергу, є об'єктом моделювання. Для включення одного об'єкта до складу іншого, його необхідно декларувати як компонент (**component**). Причому, оголошення компонента має повністю збігатися з описом відповідного йому інтерфейсу об'єкта моделювання (**entity**) з урахуванням портів і параметрів, що настроюються. Формат декларації компонента наведено нижче:

component *ім'я компонента* [**is**]
[**generic** (список параметрів, що настроюються);]
[**port** (список зовнішніх сигналів);]
end component [*ім'я компонента*];

Після службового слова **component** записується ідентифікатор *ім'я компонента*, який, фактично, визначає тип компонента і має повністю збігатися з ім'ям відповідного об'єкта моделювання. Далі, після необов'язкового службового слова is прямують список параметрів, що настроюються (**generic**), і опису інтерфейсних сигналів (**port**), ідентичні опису інтерфейсу (**entity**) відповідного об'єкта моделювання. Завершується опис компонента закриванням операторної дужки **end component** з необов'язковим зазначенням *імені компонента*.

Формати оголошення процедур і функцій будуть розглянуті нижче у відповідних підрозділах.

Декларацiйна частина архітектури може бути відсутньою.

У виконуваному блоці архітектури (після відкривання операторної дужки **begin**) містяться паралельні оператори VHDL, призначені для опису функціонування системи в термінах паралельних процесів. Завершується опис архітектури об'єкта моделювання закриванням операторної дужки **end** з необов'язковим зазначенням службового слова **architecture** й імені архітектури (для підвищення читаності VHDL опису рекомендується використовувати повний формат закривання операторної дужки **end**).

Приклад опису архітектури об'єкта моделювання shem_1 (див.
рис. 3.5), оголошеного в попередньому підрозділі, наводиться нижче:

architecture *arch_1* **of** *shem_1* **is**
begin
E <= A **or** B;

F <= C **or** D;

end architecture *arch_1*;

У даному прикладі функції об'єкта моделювання реалізовані за допомогою операторів паралельного призначення сигналу <=.

3.6 Паралельні оператори

У виконуваному блоці архітектури можуть знаходитися паралельні оператори VHDL, за допомогою яких описується робота об'єкта моделювання. Паралельні оператори задають паралельну в часі поведінку (функціонування) об'єкта моделювання.

Як було показано в попередніх підрозділах, **порядок виконання паралельних операторів не залежить від їхнього розташування всередині виконуваного блоку архітектури.**

Паралельні оператори активізуються зміною значень, що входять до їх складу сигналів, які, фактично, пов'язують різні функціональні блоки проектованого пристрою.

До паралельних операторів VHDL належать:

- оператор паралельного призначення сигналу;
- оператор умовного паралельного призначення сигналу;
- оператор вибіркового паралельного призначення сигналу;
- оператор конкретизації компонента;
- оператор генерації компонентів;
- оператор паралельного сполучення;
- оператор паралельного виклику процедури;
- оператор блоку;
- оператор процесу.

3.7 Оператор паралельного призначення сигналу

Формат оператора паралельного призначення сигналу:

Результуючий сигнал <= [специфікація затримки] вираз [after T];

Оператор паралельного призначення сигналу за синтаксисом подібний зі стандартними операторами присвоювання в інших мовах програмування, але має власну апаратно-орієнтовану специфіку. Обчислення результуючого значення сигналу задається виразом у правій частині, а специфікація затримки визначає інтервал часу T, через який сигнал прийме дане значення.

Приклади запису:

A <= B **or** (C **and** D) **after** 20 ns;

E <= B **or** F;

У першому рядку, присвоювання нового значення сигналу A виконується через 20 ns після зміни одного з вхідних сигналів (B, C, D), а в другому випадку – значення сигналу E присвоюється через мінімальний інтервал часу Δ (дельта - затримку) після зміни вхідних сигналів (B, F).

Розглянемо детальніше ще один приклад використання оператора паралельного призначення сигналу.

C <= '1', '0' after 10 ns, 1 after 20 ns;

Часові діаграми сигналу C наводяться в таблиці 3.1 та на рис. 3.6

Таблиця 3.1 – Часова діаграма сигналу C

t, ns	C
0	‘1’
10	‘0’
20	‘1’

Оператор паралельного призначення сигналу активізується тільки при зміні значень сигналів у виразі. Залежно від специфікації затримки реалізуються різні механізми інерційності присвоювання нового значення сигналу.

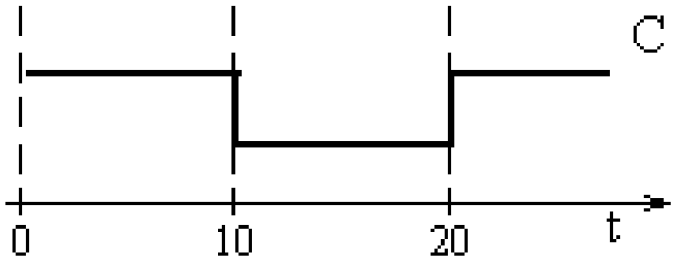


Рисунок 3.6 – Часова діаграма сигналу C

3.8 Поняття дельта-затримки в ході призначення сигналу

Цифрові пристрої завжди мають кінцеву інерційність. Тому, якщо в явному вигляді не вказані параметри затримки, кожен оператор паралельного призначення сигналу, що подається в ході синтезу логічним вентилем, виконує дії не миттєво, а через деякий мінімальний часовий інтервал, званий дельта-затримкою Δ.

Розглянемо приклад, у якому поданий фрагмент коду, що складається з операторів паралельного призначення сигналу:

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

. . .

A <= **not** B;

B <= C **or** D;

F <= **not** A;

Дана послідовність операторів може бути синтезована в схему з послідовним включенням логічними вентилями (див. рис. 3.7). Очевидно, що при постійних вхідних сигналах (C, D), на виході схеми встановлюються також постійні значення. При зміні сигналів C або D в момент часу t0 послідовно змінюються значення сигналів B, A і F (спочатку виконується 2-й рядок фрагмента, потім 1-й, а потім 3-й). Причому кожен вентиль вносить одну дельта-затримку. Так, сигнал B отримує нове значення в момент часу $t_0 + \Delta$, сигнал A – в момент часу $t_0 + 2\Delta$, а сигнал F – в момент часу $t_0 + 3\Delta$.

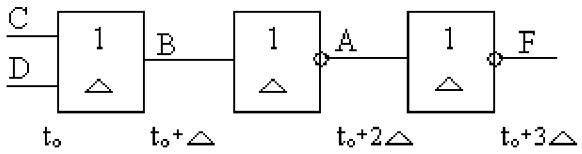


Рисунок 3.7 – Синтезована логічна схема, подана операторами паралельного призначення сигналу без специфікації затримки

Дельта - затримка є елементарним циклом моделювання проекту і дозволяє вирішувати заплановані на один і той самий фізичний час зміни значень сигналів. З точки зору логічної схеми дельта - затримка – це затримка на одному рівні логічних вентилів, коли не задані конкретні параметри затримок. Поняття Δ затримки є одним з основних понять моделювання в мові VHDL.

Приклад розглянутого VHDL-коду з явною вказівкою параметрів затримки в операторах паралельного призначення сигналу наведено нижче:

A <= **not** B **after** 10 ns;

B <= C **or** D **after** 20 ns;

F <= **not** A **after** 5;

Логічна схема, синтезована з даного фрагменту, наводиться на рис. 3.8. Кожен вентиль вносить часову затримку, відповідну величині параметра, зазначеного після слова **after**. При зміні сигналів C або D у момент часу t_0 сигнал B отримує нове значення у момент часу t_0 + 20 ns, сигнал A – в момент часу t_0 + 30 ns, а сигнал F – у момент часу t_0 + 35 ns.

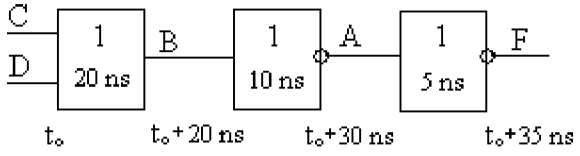


Рисунок 3.8 – Синтезована логічна схема, подана операторами паралельного призначення сигналу з явною вказівкою параметрів затримок

VHDL - конструкції з явною вказівкою значень у параметрах затримки використовуються тільки для моделювання роботи розроблюваного проекту. Реальна поведінка схеми з урахуванням часу спрацьовування компонентів визначається тільки на етапі фізичного моделювання.

3.9 Інерційна, режекційна і транспортна затримки в ході призначення сигналу

У загальному випадку в операторах призначення сигналу можна встановлювати 3 види затримки: інерційну, транспортну і режекційну.

Інерційна (**inertial**) затримка (задається за замовчуванням) моделює поведінку реального логічного вентиля при зміні значень сигналів на його входах. Приклад інерційної затримки з параметром 10 ns і її еквівалентна форма запису на VHDL за замовчуванням:

A <= inertial D after 10 ns; === A <= D after 10 ns;

Якщо зміна значення вхідного сигналу утримується менше тривалості інтервалу, зазначених вище в параметрі затримки, то логічний вентиль не реагує на зміну вхідного сигналу (це пов'язано з кінцевим часом спрацьовування реальних логічних вентилів і визначається тривалістю перехідних процесів при заряді / розряді ємнісних елементів).

Транспортна (**transport**) затримка моделює процес поширення сигналів по провідниках (лініях зв'язку) – будь-яка зміна вхідного сигналу повторюється через час, рівний параметрам затримки. Приклад транспортної затримки з параметром 10 ns

B <= transport D after 10 ns;

Режекційна (**reject**) затримка найбільш точно дозволяє моделювати реальні фізичні характеристики логічних компонентів. Так, у разі простою інерційної затримки в 10 ns, вентиль, наприклад, не реагуватиме на сигнал тривалістю 9.99 ns, що в реальних умовах практично не реально. Тому, при вказівці режекційної затримки необхідно вказувати два параметри: максимальну тривалість відсікання імпульсів, і час затримки вихідного сигналу щодо вхідного. Розглянемо приклад запису режекційної затримки, за якої схема не реагуватиме на короткі імпульси тривалістю менше **4 ns**, імпульси більшої тривалості повторюватимуться вентилем через **10 ns**:

C <= reject 4 ns inertial D after 10 ns;

За допомогою режекційної затримки можна найбільш гнучко задати часові параметри роботи компонента.

Часові діаграми для наведених прикладів затримок сигналу наводяться на рисунку 3.9.

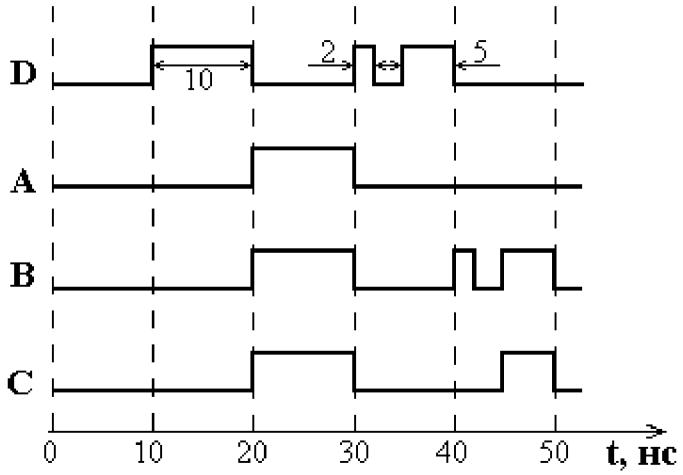


Рисунок 3.9 – Часові діаграми для сигналів A, B і C, що моделюють інерційну, транспортну і режекційну затримки сигналу D відповідно

3.10 Оператор умовного паралельного призначення сигналу

<приймач сигналу> <= [transport] [<модель затримки>] <прогнозування поведінки> when <умова> else <прогнозування поведінки>;

приклад застосування оператора

```
Q <= 1 when A = '1' else
     2 when A = '0' else
     3 when A = 'Z' else 4;
```

3.11 Оператор вибіркового паралельного призначення сигналу

with <ключовий вираз> select
<приймач> <= [quarded] [transport] [<модель затримки>]
<прогнозування поведінки> when <варіант>,"
<прогноз поведінки> when <варіант>;

приклад застосування оператора, мультиплексор 4-1.

```
with S select
Q <= D0 when 0,
D1 when 1,
D2 when 2,
D3 when 3;
```

3.12 Оператор конкретизації компонента

Структурний опис архітектури становить структуру об'єкта, як композицію з компонент, з'єднаних між собою і обмінюються сигналами. Функції, що реалізуються компонентами, в явному вигляді на відміну від попередніх прикладів у структурному описі не вказуються. Структурний опис включає опис інтерфейсу компоненту, з яких складається схема, і їх зв'язків. Повні (інтерфейс + архітектура) описи об'єктів – компоненти мають бути раніше поміщені в проектну бібліотеку, підключену до структурного опису архітектури.

Оператор конкретизації компонента відіграє основну роль в ході реалізації ієрархічного проектування. Його синтаксис:

\ Оператор вставки компонента \ :: = \ мітка примірника елемента \: \ вставляється елемент \ [**generic map** (\ зв'язування настроювальної константи \ {, \ зв'язування настроювальної константи \});] [**port map** (\ зв'язування порту \ {, \ зв'язування порту \})];

Дія цього оператора полягає в підстановці замість себе одного примірника компонента (вставляється елемент – компонент) або об'єкта (що вставляється елемент – об'єкт проекту), або компонента, зазначеного в конфігурації (в бібліотеці). Якщо вставляється компонент, то він має бути оголошений в цій архітектурі.

Приклад оператора генерації компонента для опису елемента «I-HE»:

```
COMPONENT I_NE
GENERIC (T: TIME);
PORT (X1, X2: IN BIT; Y: OUT BIT);
END COMPONENT;

K1: I_NE GENERIC (10 ns) PORT MAP (x1 => A, x2 => B, y => C);

K2: I_NE GENERIC (10 ns) PORT MAP (D, E, F);
ENTITY I_NE IS
GENERIC (T: TIME);
PORT (X1, X2: IN BIT; Y: OUT BIT);
END I_NE;
ARCHITECTURE BEH_I_NE OF I_NE IS
BEGIN
Y <= NOT (X1 AND X2) AFTER T;
END ARCHITECTUR BEH_I_NE;
```

Приклад компонента К1, що згенерований з підключенням зовнішніх сигналів до внутрішніх з ключовими відповідностями, а К2 – згенерований за позиційною відповідністю сигналів – з відповідним порядком перерахування внутрішніх сигналів під час їхнього опису в декларативній частині компонента.

3.13 Оператор генерації компонентів

Оператор генерації

Даний оператор використовується для генерації схем, що мають регулярні елементи, наприклад, суматорів. У випадку, якщо об'єкт включає в себе багато однотипних компонентів, оператори конкретизації компонентів для них мають схожу структуру. У моделі вони займають багато місця і ускладнюють її розуміння. Оператор генерації generate дозволяє в цьому випадку компактно описати модель. Він дозволяє організувати цикл з параметром і одноразовим параметричним описом компонента всередині циклу, на базі якого будуть згенеровані описи для всієї групи однотипних компонентів.

<мітка>: for <параметр генерації> generate
<паралельні оператори>
end generate;
або
<мітка>: if <умова генерації> generate
<паралельні оператори>
end generate;

Відмінності паралельного оператора генерації і послідовних операторів **for**, **if**.

1. Оператор генерації – це паралельний оператор, **for**, **if** – послідовні оператори.

2.Оператор генерації не має фраз else, elsif.

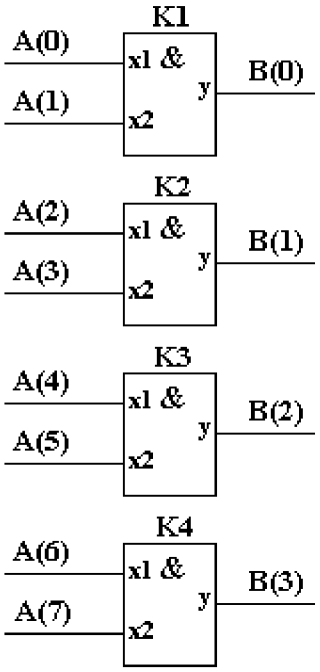
3. Для оператора генерації необхідна мітка.

4. Важливо відзначити, що параметр генерації – змінна-лічильник циклу окремо не декларується в розділі описів архітектури, а також, що всередині оператора генерації можуть застосовуватися тільки паралельні оператори.

Розглянемо приклад схеми, зображеної на рис. 3.10. Схема виконує операції логічного «І» над суміжними лініями векторного сигналу А. Основний фрагмент програми, що описує ці дії за допомогою оператора GENERATE:

```
Architecture arh1 of shem is
Component kand
Port (x1, x2: in bit; y: out bit);
End component;
Signal A: bit_vector (7 downto 0);
Signal B: bit_vector (3 downto 0);
Begin
komp: for i in 0 to 3 generate
```

```
k: kand port map (x1 => a (i * 2); x2 => a (i * 2 + 1); y => b (i));
end generate komp;
End arhitecture;
```



-

Рисунок 3.10 – Синтезована схема під час використання оператора generate при генерації компонентів K1 –K4 з функціями логічного «І»

3.14 Оператор паралельного виклику процедури

Підпрограми в VHDL, як і в інших алгоритмічних мовах, забезпечують, по-перше, структуризацію опису проекту, а по-друге, є засобом економії часу проектувальника, дозволяючи замінити кілька описів подібних фрагментів алгоритму одним оголошенням підпрограми і відповідними викликами в основному тексті.

Кожна підпрограма, яка використовується в проектному модулі, має бути представлена тілом підпрограми в розділі декларацій цього модуля або проектного модуля, ієрархічно старшого по відношенню до даного. Розрізняють два види підпрограм – процедури (procedure) і функції (function).

Обидва види містять в своєму тілі набір послідовних операторів, які задають сукупність дій, виконуваних після виклику цієї підпрограми. Процедура повертає результати або шляхом безпосереднього перетворення об'єктів, визначених у викликаючій програмі (глобальних сигналів або змінних), або за рахунок зіставлення об'єктів через список відповідностей. Функція ж визначає єдине значення, яке використовується у виразах, до яких включено виклик цієї функції.

Специфікація підпрограми визначає її інтерфейс – ім'я, вхідні і вихідні дані. Вхідні дані підпрограми специфікуються напрямком in, вихідні –

напрямком out, а дані, які сприймаються підпрограмою і повертаються у викликаючу програму зміненими, – як inout. Вказівка – категорії елемента списку (constant, variable або signal) забезпечує контроль коректності використання підпрограми. За замовчуванням визначено, що зіставляючий об'єкт належить до категорії variable.

Невідповідність типу або категорії фактичного або формального параметра є помилкою.

У розділі декларацій підпрограми можуть визначатися локальні, тобто певні тільки в тілі підпрограми, об'єкти: вкладені підпрограми, типи і підтипи даних, змінні, константи, атрибути. Розділ операторів містить тільки послідовні оператори.

Мова VHDL, на відміну від традиційних мов програмування, розрізняє послідовний і паралельний виклик підпрограми.

Синтаксично вони однакові, але різна їхня локалізація і правила виконання. Виклик функції трактується як паралельний, якщо входить в паралельний оператор, найчастіше – в оператор паралельного присвоювання, і як послідовний, якщо входить в послідовний оператор.

Оператор виклику процедури є послідовним, якщо локалізована в тілі процесу або тілі іншої підпрограми. В інших випадках оператор виклику підпрограми інтерпретується як паралельний оператор. Одна і та сама підпрограма може викликатися як паралельним, так і послідовним оператором. Паралельний виклик процедури фактично еквівалентний процесу, тіло якого збігається з тілом процедури з точністю до позначень, а список ініціалізаторів містить вхідні фактичні параметри оператора виклику.

Нижче наведено кілька вузлів підсумовування і віднімання в різній формі.

```
architecture ADD_EXAMPLES of SOMETHIN is procedure ADD_SUBB
(signal A, B: in integer;
 signal OPERATION: in std_logic;
signal RESULT: out integer) is begin if OPERATION = '0' then RESULT <= A
+ B;
else RESULT <= A-B;
end if;
end ADD_SUBB;
….
 begin
FIRST: ADD_SUBB (X1, Y1, CONTROLE, Z1);
SECOND: ADD_SUBB (A => X1, B => Y1, RESULT => Z2, OPERATION
=> CONTROLE);
```

Виклики процедур з мітками FIRST і SECOND ілюструють альтернативні способи запису списків відповідності. Перша форма запису списку асоціацій відповідає позиційному порівнянню фактичних і формальних параметрів підпрограм, а друга – порівнянню з іменем. Позиційне співставлення вимагає точного збігу порядку запису фактичних параметрів у списках відповідності та порядку запису формальних параметрів в інтерфейсному списку підпрограми.

Якщо який-небудь параметр не використовується або використовується значення входу за замовчуванням, відповідна позиція в списку наголошується як порожня. При зіставленні по імені порядок запису не має значення, важливо лише збіг імені формального параметра з ім'ям, зазначеним у декларації підпрограми.

Важливе значення у підпрограмі має оператор повернення return. У процедурі цей оператор припиняє її виконання, передаючи управління викликаючій програмі. Якщо виконаний оператор return у процедурі, викликаної послідовним оператором, то після нього виконується оператор викликаючої програми, наступний за оператором виклику.

Як приклад розглянемо опис функції LOG1, що обчислює результат логічного виразу LOG1 = (X and Y) or Z. Параметри (X, Y, Z) і повертаюче значення (визначене після ключового слова return) мають тип BIT. Ця функція може бути описана таким чином:

```
function LOG1 (signal X, Y, Z: BIT) return BIT is
begin
return (X and Y) or Z;
end LOG1;
```

Також розглянемо приклад опису функції, що виконує перетворення типу BIT у тип BOOLEAN (дані типи не еквівалентні).

```
function LOG2 (signal X: BIT) return BOOLEAN is
begin
if X = '1' then return true;
else return false;
end if;
end LOG2;
```

VHDL - код з текстом підпрограми може знаходитися в розділі декларацій архітектурного тіла основної програми, або в пакеті (бібліотеці).

При виклику підпрограм вказуються ім'я функції та список фактичних параметрів.

Приклад використання в архітектурі процедури і функції, що розглянуті вище:

```
Architecture arh1 of shem is
Signal S1, S2, S3: bit_vector (7 downto 0);
Signal S4: bit;
Signal S5: boolean;
procedure VEC (SIGNAL X, Y: in BIT_VECTOR (7 downto 0);
SIGNAL Z: out BIT_VECTOR (7 downto 0)) is begin
For i in 0 to 7 loop
Z (i) <= X (i) + У (i)
End loop;
End VEC;
function LOG2 (signal X: BIT) return BOOLEAN is
begin
```

```
if X = '1' then return true;
else return false;
end if;
end LOG2;
Begin
VEC (S1, S2, S3);
S5 <= LOG2 (S4);
End arh1;
```

У схемах можуть виникати ситуації, коли кілька вихідних сигналів об'єднуються в один. В результаті виходить сигнал (shared signal), що формується декількома джерелами (сигнал, який має декілька драйверів). У цьому випадку виникає необхідність визначення результуючого значення для цього сигналу. Наприклад, у реальних цифрових пристроях на рівні фізичних сигналів реалізується формування результуючого значення сигналу, що на рівні логічних сигналів проявляється як виконання сигналами-джерелами деякої логічної операції, так званої операції монтажної логіки (wired logic). У моделі мовою VHDL необхідно запрограмувати такі логічні операції, для чого використовують механізм застосування особливих роздільних функцій (resolution function). Спільні сигнали в моделі мають бути декларовані як сигнали спеціального роздільного типу (resolved type). На відміну від звичайних сигналів, при декларації цих сигналів вказується не тільки тип, а й функція, на базі якої визначатимуться значення сигналу.

Signal name: [resolution_function_name] type_mark [range].

Ідентифікатор type_mark задає ім'я типу для обумовленого сигналу.

Ідентифікатор Resolution_function_name – ім'я функції, що використовується для визначення значення сигналу.

Функція вирішення колізій, асоційована з таким сигналом, викликається щоразу, коли будь-які з джерел виконує оператор присвоювання нового значення сигналу. Сигнал існує безперервно в модельному часу, і те, що інші джерела в даний момент не змінювали значення на своїх вихідних портах, які пов'язані з даними сигналом, не означає, що там немає ніяких значень. На цих виходах зберігаються колишні значення. Відповідно, функція вирішення колізій має бути написана так, щоб порядок аналізу значення сигналу від різних джерел не впливав би на результат роботи функції.

В описі обчислюваного сигналу вказується тільки ім'я роздільної функції. Сама функція описується окремо. Одна роздільна функція може використовуватися для багатьох сигналів.

Приклад роздільної функції, що реалізує монтажне "АБО" (Wired "or") згідно з підключенням компонентів за схемою, зображено на рис. 3.11.

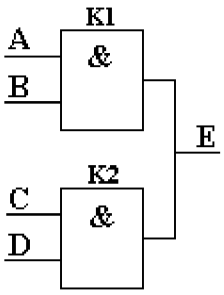


Рисунок 3.11 – Приклад схеми, що реалізує операцію монтажної логіки

```
Architecture arh1 of shem is
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;
Signal E: wired_or bit;
begin
K1: process (A,B) begin
E<=A and B;
End process K1;
K2: process (C,D) begin
E<=C and D;
End process K2;
```

Драйвери сигналу, зазначених вище параметром inputs, визначаються атрибутом 'range сигналу (inputs'range). Функція Wired_or сканує значення всіх драйверів сигналу Y і повертає '1', якщо хоча б один з них дорівнює '1', інакше '0'.

3.15 Оператор блоку

Оператор блоку BLOCK, аналогічно оператору PROCESS, є складовим оператором, тіло якого включає кілька операторів, але, в даному випадку, паралельних. Оператори тіла блоку, як і інші паралельні оператори, забезпечують можливість подання паралелізму в моделювальній системі. Ці оператори ініціюються не за послідовним, а за подієвим принципом, а результати їхнього виконання стають доступні іншим операторам як включених в блок, так і розміщених в інших блоках або "індивідуально", тільки після виконання усіх операторів, ініційованих однією подією.

У цьому сенсі оператори, включені в блок, не відрізняються від "індивідуальних" паралельних операторів.

Об'єднання операторів у блоки забезпечує такі можливості:

– структуризація тексту опису, тобто можливість явного і наочного виділення сукупності операторів, що описують закінчений функціональний вузол;

– можливість оголошення в блоці локальних типів, сигналів, підпрограм і деяких інших локальних об'єктів;

– можливість приписування всім або деяким операторам блоку загальних умов ініціалізації.

Спрощені правила запису оператора блоку визначені так:

<Позначка блоку>: BLOCK [(охоронний вираз)] [IS]
[<Розділ декларацій блоку>]
BEGIN
<Розділ операторів блоку>
END BLOCK [<мітка блоку>];

Найбільш специфічними аспектами блокової організації є поняття охоронного виразу і охороняється оператором присвоювання.

Охоронний вираз – це будь-який вираз логічного типу, аргументами якого є сигнали. Будь-яка зміна сигналів, що входять в охоронний вираз, викликає обчислення значення цього виразу і присвоєння отриманого значення зумовленої змінної QUARD. Область дії змінної QUARD – все тіло блоку, і воно може використовуватися як звичайна логічна змінна у вкладених операторах блоку. Охоронний оператор присвоювання використовує значення змінної QUARD без явної вказівки умови в програмі. Якщо QUARD = '0', то виконання операторів присвоювання, що містять ключове слово QUARDED, в такому блоці заборонено.

Розглянемо однорозрядний суматор.

```
ENTITY add1_e IS
PORT (b1, b2, enable: IN BIT;
c1, s1: out BIT);
END add1_e;

ARCHITECTURE struct OF add1_e is
BEGIN

p0: BLOCK (enable = '1')
BEGIN
s1 <= GUARDED (b1 XOR b2);
c1 <= GUARDED (b1 AND b2);
END BLOCK p0;

END struct;
```

Охоронним виразом блоку є вираз enable = 1. Якщо цей вислів приймає значення TRUE (істина), то охоронювані конструкції (призначення сигналів) виконуються, тобто однорозрядний суматор складає числа, якщо ж значення виразу є FALSE (неправда), то охоронювані призначення сигналів не виконуються, тобто суматор не складає числа b1, b2. Охорона призначення сигналів здійснюється зазначенням ключового слова GUARDED.

Як інший приклад охоронного блоку наведемо приклад опису D-тригера з асинхронним скиданням у вигляді блоку з охоронним виразом (clk = '1' or clr = '1').

```
D_LATCH: block (clk = '1' or clr = '1')
begin
Q <= guarded '0' when clr = '1';
else D when clk = '1';
else Q;
end block D_LATCH;
```

У даному прикладі clk – вхід синхронізації, clr – асинхронне скидання, D – вхід даних, Q – вихід тригера. Коли охоронний вираз має значення «неправда», то сигнал Q в лівій частині зберігає своє колишнє значення. Сигнал асинхронного скидання clr має пріоритет по відношенню до сигналу clk.

3.16 Оператор процесу

Оператор process – паралельний оператор, що дозволяє реалізувати незалежну послідовну поведінку певної частини проекту за допомогою послідовних операторів. Оператор процесу виконує функцію операторних дужок для послідовних операторів. Цей оператор визначається саме як складний оператор паралельного типу. Під складеним оператором розуміють оператор, який має тіло, що містить кілька вкладених операторів. Тому, цей оператор і є фактично операторними дужками для послідовних операторів VHDL. Оператор процесу починає виконуватися при зміні сигналів, що входять у список ініціалізаторів (за відсутності такого списку – безумовно після виконання усіх вкладених операторів), а результати його виконання доступні іншим паралельним операторам тільки після призупинення процесу.

Всі процеси виконуються паралельно і фактично є програмами, кожна з яких виконується на відповідному віртуальному процесорному елементі. Процес неможливо розташувати в процес, оскільки там є місце тільки для послідовних операторів. Тому, вкладені процеси, на відміну від, наприклад, вкладених циклів, неможливі. Процеси обмінюються сигналами, які

виконують синхронізацію процесів і переносять значення між ними. Якщо над сигналами визначена функція дозволу, то виходи джерел сигналу можуть об'єднуватися, сигнали можна оголошувати в процесах. Тіло процесу може мати розділ описів Declaration part і виконавчий розділ, який розташовується між ключовими словами begin і end: Оголошеними в процесі можуть бути: оголошення і тіло підпрограми, оголошення типу і підтипу, оголошення константи, змінної, файла, псевдоніма, оголошення та специфікація атрибута, оголошення групи. Те, що оголошено в деклараційній частині процесу, має область дії (видимість), яка обмежена даним процесом.

Спрощений формат оператора:
process [(список сигналів чутливості)]
[Деклараційна частина]
Begin
……
Виконавча частина
……
end process;

У дужках при моделюванні можна вказувати список чутливості – перелік сигналів, зміна значень одного з яких призводить до активізації процесу. В ході синтезу список чутливості процесу ігнорується (процес реагує на зміну всіх сигналів, присутніх у схемі, а саме, що входять до виконуючої частини). Приклади процесів будуть наведені нижче при розгляді послідовних операторів VHDL.

3.17 Послідовні оператори

Послідовні оператори в мові VHDL можуть розташовуватися виключно в тілі процесу. Вони подібні стандартним загальноалгоритмічним операторам мов високого рівня. Послідовні оператори називають також операторами реєстрової логіки, на використанні яких ґрунтується поведінкова форма подання проектів цифрових пристроїв. До послідовних операторів мови VHDL належать:

- оператор послідовного присвоєння сигналу;
- -оператор послідовного присвоєння змінної;
- оператор очікування;
- оператор послідовного умовного призначення сигналу;
- оператор послідовного вибіркового призначення сигналу;
- оператор циклу з додатковими операторами;
- пустий оператор.

3.18 Оператор послідовного присвоєння сигналу

Цей оператор <= співпадає за синтаксисом з оператором паралельного присвоєння сигналу і, фактично, відрізняється від нього розташуванням у тілі процесу.

Нижче наведено приклад послідовного оператора присвоєння

Приклад оператора процесу:
process (X1, X2, X3)
begin
Y <= X1 and (X2 or X3);
end process;

Цей приклад фактично еквівалентний оператору паралельного присвоєння сигналу, який розташований в архітектурі за межами процесу

Y <= X1 and (X2 or X3);

Виконання оператора присвоювання сигналу означає тільки обчислення його виразу, і лише призначення сигналу. Саме ж присвоювання значення сигналу, фактично, виконується в момент зупинки процесу за очікуванням події. Тому, якщо в одному процесі є кілька присвоювань одному сигналу, то справжнє присвоювання виконується тільки в момент зупинки процесу. Якщо перед зупинкою процесу виконувалося читання цього сигналу (наприклад, участь його як операнд у правій частині логічного виразу), то буде прочитано значення, що присвоєне при минулому запуску процесу.

Наприклад, згідно з виконанням послідовностей операторів, що наведені у таблиці 3.2, при паралельному та послідовному присвоєнні значень сигналам наглядатиметься однаковий результат за той самий час, але до нього реалізовуватимуться окремі шляхи (див. табл. 3.3).

Таблиця 3.2 – Приклади фрагментів паралельних і послідовних операторів присвоєння сигналів

Паралельні оператори	Послідовні оператори
A<=B;	Process (A,B,C, D) begin
B<=C;	A<=B;
C<=D;	B<=C;
	C<=D;
	End;

Таблиця 3.3 – Часові діаграми під час виконання фрагментів паралельних та послідовних операторів присвоєння сигналів

Модельний час	Часові діаграми							
	Реалізація паралельних операторів				Реалізація послідовних операторів			
	A	B	C	D	A	B	C	D
t	1	2	3	4	1	2	3	4
t+Δ	1	2	3	0	2	3	4	0
t+2Δ	1	2	0	0	3	4	0	0
t+3Δ	1	0	0	0	4	0	0	0
t+4Δ	0	0	0	0	0	0	0	0

3.19 Оператор послідовного присвоєння змінної

Присвоєння змінної виконується за допомогою оператора := і суттєво відрізняється від присвоєння значення сигналу та виконується негайно.

Приклад оператора процесу з присвоєнням значення змінної наведено нижче

```
process (X1, X2, X3)
variable A, B, C: integer;
begin
C := A*B;
end process;
```

3.20 Оператор очікування

Цей оператор **wait** припиняє виконання процесу і, як говорилося вище, в цей момент зупинки виконуються присвоювання значень сигналам, далі процес продовжує виконання під час появи події, яке вибирається цим оператором.

Існують чотири форми запису оператора **wait**:

- **wait**; під час запису цього оператора без параметрів, процес припиняє роботу на весь останній час моделювання. Фактично, такий процес виконується тільки один раз за час роботи програми і може бути корисним, наприклад, під час роботи з файлами, завантаження інілізаційних даних з постійних запам'ятовучих пристроїв та аналогічних завдань;

- конструкція оператора **wait on** із списком сигналів, наприклад, **wait on S1**, **S2**; призупиняє виконання процесу до моменту, доки не зміниться один із сигналів (S1, S2), зазначених у списку. Ця форма запису оператора wait може бути еквівалентною оператору process із списком чутливості сигналів, при цьому для коректного присвоєння значень сигналам оператор wait on потрібно розміщати наприкінці тіла процесу, а не на початку.

- Наступні форми запису процесів еквівалентні

```
-  process (B, C)                -  process
-  begin                         -  begin
-  A<=B or C;                    -  A<=B or C;
-  end process;                  -  wait on B, C
                                 -  end process;
```

Конструкція оператор **Wait until** умовний вираз, наприклад, **Wait until = = '1'**; призупиняє виконання процесу до того, доки виконуватиметься призначена умова, тобто виконання процесу почнеться при переключенні значення сигналу S в '0'.

- Конструкція оператора **WAIT for** період часу, наприклад, **WAIT for 100 ns**; встановлює максимальний час очікування (time out), після закінчення якого (100ns) процес відновлює своє виконання.

- Допускається кілька конструкцій оператора WAIT об'єднувати в один вираз. Наприклад, вираз WAIT on X, Y until (Z = 0); призупинить процес до моменту зміни значень сигналів X і Y; після цього буде перевірено умова Z = 0, і, якщо результат перевірки буде false, процес поновиться.

Якщо оператор WAIT використовується всередині процесу, то цей процес не повинен мати власного списку чутливості сигналів.

3.21 Оператор послідовного умовного призначення сигналу

Цей умовний оператор **if** залежно від заданих умов виконує послідовності операторів, причому від умови залежить, яка з цих послідовностей операторів виконується.

Повна форма запису оператора if має такий вигляд:

```
if умовний вираз_1 then
  послідовність операторів
[elsif умовний вираз_2 then
  послідовність операторів]
[else
  послідовність операторів]
end if;
```

Оператор допускає наявність будь-якого числа фраз elsif. Конструкції elsif і else є необов'язковими.

Оператор дозволяє синтезувати мультиплексори. Наприклад, найпростіший, що синтезований на рис. 3.12, за таким фрагментом

```
if C=1 then D<=A;
        else D<=B;
end if;
```

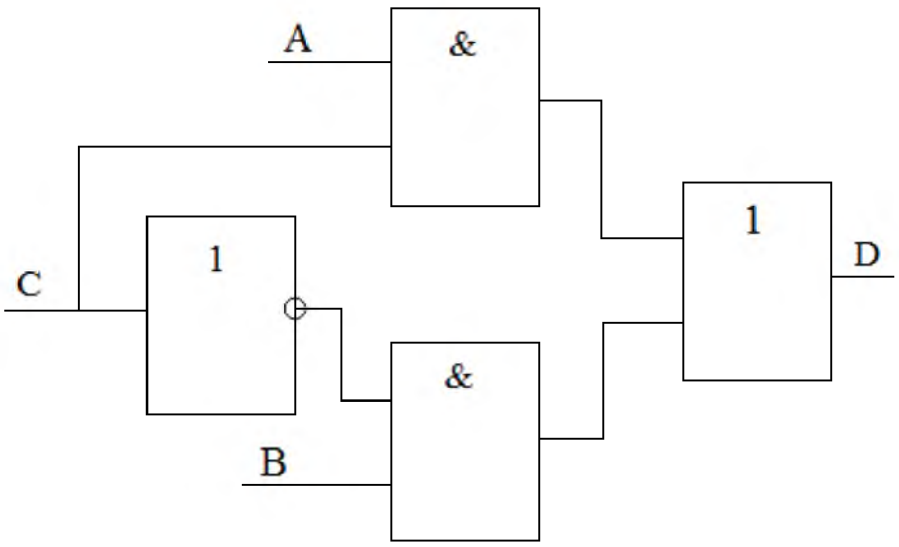


Рисунок 3.12 – Синтезований найпростійший мультиплексор

Приклад опису RS-тригера

```
IF s = '1' THEN q <= '1';
ELSIF r = '1'THEN q <=' 0 ';
END IF;
```

3.22 Оператор вибору

Оператор вибору **case** здійснює вибір оператора на основі значення керуючого виразу, а потім виконує обраний оператор або групу операторів. Як керуючий вираз може використовуватися будь-який дискретний тип або одновимірний масив символів.

```
case <ключовий вираз> IS
when <варіант_1>  => <оператор> [<оператор>];
[when <варіант_2>  => <оператор>[ <оператор> ];
[when <others> => <оператор>[ <оператор> ];];
end case;
```

Оператор case також дозволяє синтезувати мультиплексори. Розглянемо приклад

```
signal C: bit_vector [1 downto 0];
        A,B, C, Q: bit;
……
case C is
```

```
when "00" => Q <= A;
when "01" => Q <= B;
end case;
```

Цей фрагмент буде синтезовано в схему, що наведено на рис. 3.13. Ця комбінаційна схема додатково містить, на перший погляд, несподіване рішення – елемент пам'яті D, який утримує попередні значення сигналу Q' при невизначених комбінаціях вхідного сигналу C («10» та «11», при C(1)=1). Цей паразитний елемент пам'яті вносить затримку у розповсюдженні сигналу, що може бути критичним під час роботи на високих частотах. Щоб уникнити цього, потрібно розглядати всі комбінації вхідного сигналу за рахунок конструкції others, особливо це стосується, наприклад, типу std_logic, кожний розряд при якому приймає 9 значень. Приклад коректного застосування оператора case та відповідна синтезована схема на рис. 3.14 наведені нижче.

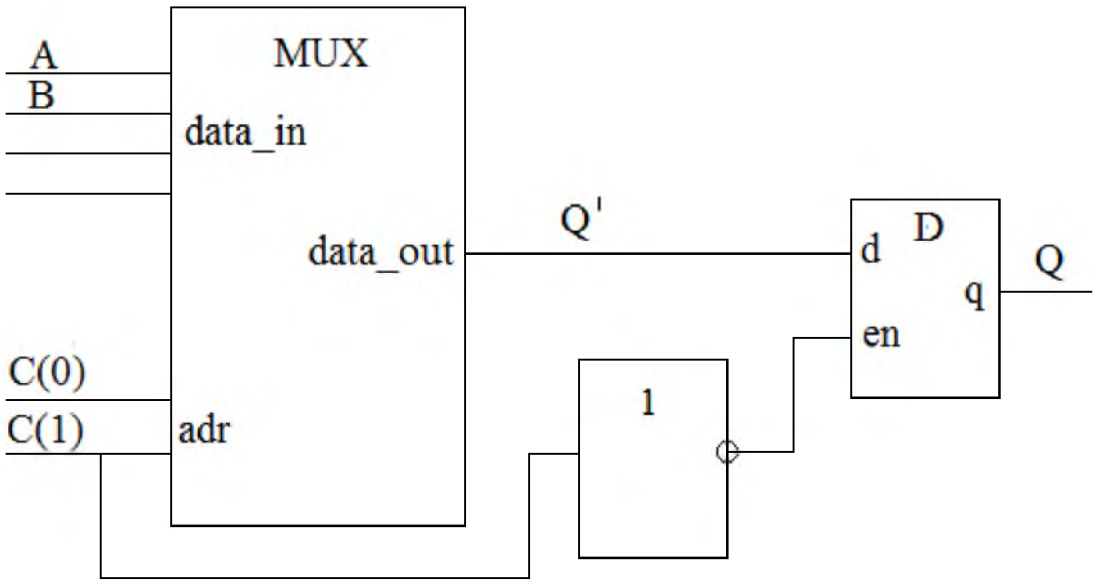


Рисунок 3.13 – Синтезований мультиплексор з паразитним елементом пам'яті

```
case C is
when "00" => Q <= A;
when "01" => Q <= B;
when others => Q <= B
end case;
```

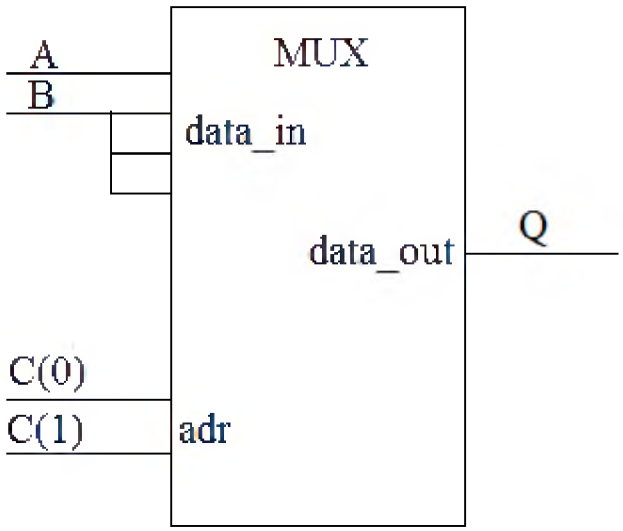


Рисунок 3.14 – Синтезований мультиплексор

3.23 Організація циклів

Оператори циклу дозволяють виконувати послідовність операторів більше одного разу. Оператор циклу складається з заголовка і тіла циклу. Будь-який оператор циклу в мові VHDL закінчується ключовими словами end loop (з міткою або без).

[<Мітка циклу:] [<ітераційна схема>] loop
<Оператори тіла цикла>
end loop [<мітка циклу>];

Оператор for дозволяє організувати цикл з параметром.

for <парам.циклу> in S1 to S2 loop
<Оператори тіла циклу>
end loop;

приклад:
for i in 1 to 4 loop
Y (i) <= A (i) and B (i);
end loop;
Необхідно відмітити, що пераметр циклу окремо не декларується, наприклад, як змінна типу integer та має бачимість тільки у тілі циклу.

Оператор while дозволяє організувати цикл з передумовою.

```
while <передумова>  loop
  <Оператори тіла циклу>
end loop;
```

Цикл while виконуватиметься, доки вираз у передумові матиме значення «true». При цьому необхідно передбачити можливість модифікації ідентифікаторів, що входять до передумови, в тілі циклу. Наприклад, цикл while з передумовою за значенням сигналу C.

```
while C='1' loop
Y (i) <= A (i) and B (i);
 C<=C and D;
end loop;
```

Цикл з безкінцевою кількістю ітерацій можна організувати в ході використання оператора loop без параметрів, наприклад.

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

Оператори циклу синтезуються, як правило, у регулярні структури однакових схемних елементів.

У мові VHDL не передбачено наявність операторів умовного і безумовного переходів. Мітки в програмі мають фактично тільки інформаційне значення. Тому для здійснення управління ходом виконання операторів у тілі циклу передбачені оператори NEXT і EXIT;

Оператор NEXT <умова> здійснює завершення поточної ітерації циклу під час виконання умови.

Оператор EXIT <умова> здійснює вихід з тіла циклу під час виконання умови.

Наприклад,

```
SUM: = 0;
for I in 1 to 10 loop
next when I = 5;
 exit when SUM> 100;
 SUM: = SUM + A (I);
end loop;
```

У цьому прикладі п'ята ітерація циклу пропускається, а вихід із тіла циклу виконується при перевищенні значення ідентифікатора SUM величини 100.

3.24 Пустий оператор

Використовується простим записом у послідовній частині VHDL-програми.

null

Оператор **null** не представляє дій. Він застосовується, щоб точно специфікувати, що немає дій. Типове застосування, наприклад, в операторі **case**, щоб визначити дії у всіх випадках вибору.

3.25 Функції перетворювання типів

Найчастіше більшість арифметичних операторів застосовуються до сигналів типу integer. Це не завжди зручно, тому що в реальних схемах використовуються тільки два типи даних: std_logic та std_logic_vector. При цьому над ними також необхідно здійснювати арифметичні операції. Для цього необхідно ввести функції перекладу сигналів з типу integer у std_logic_vector та навпаки. Для цього доцільно використовувати пакети з бібліотеки ieee. use ieee.std_logic_arith.all; use ieee.std_logic_unsigned.all;. У цих бібліотеках існують перекладні функції з integer в std_logic_vector і зворотні перетворення.

Функція

conv_integer (std_logic_vector) return integer;

дозволяє виконати перетворення аргумента типу std_logic_vector в значення типу integer

Функція

conv_std_logic_vector(integer, n) return std_logic_vector;

дозволяє виконати перетворення аргумента типу integer в значення типу std_logic_vector, другий параметр функції n задає розрядність перетворення.

Наприклад

```
signal a: integer range 0 to 255;
signal v: std_logic_vector (7 downto 0);
……..
a<= conv_integer (v);
……..
v<= conv_std_logic_vector(a, n);
```

У цьому прикладі використовуються взаємні перетворення восьми розрядних сигналів a типу integer та сигналу v типу std_logic_vector.

3.26 Принципи роботи з файлами

Файловий тип даних було розглянуто у підрозділі 2.10. У цілому, робота з файлами в мові VHDL дуже подібна із аналогічними діями у високорівневих мовах програмування, наприклад, Pascal та C. При цьому задається файлова змінна, яка зв'язується із фізичним файлом процедурою file_open. При цьому вказується режим доступу до файла, читання даних (read_mode), або запис даних (write_mode). Процедури доступу до файла також стандартні, read – для запису, write – для читання. При цьому першим параметром, як зазвичай, вказується файлова змінна, а далі, безпосередньо, ідентифікатор даних. Виклик процедури file_close закриває роботу із файлом. Приклад запису та читання даних із файлів a та b наведено нижче. Треба звернути увагу, що оператори роботи із файлами є послідовними та використовуються у тілі процесу. У прикладі процес завершується оператором wait без параметрів і виконуватиметься тільки один раз за час моделювання.

```
entity fileprog is
end fileprog;
architecture arh1 of fileprog is
type f_type is file of integer;
file a: f_type;
file b: f_type;
signal w, v: integer;
begin

  process
variable v1, v2, v3, v4: integer;
variable bol: boolean;

begin
file_open (a, "c: \ vhdl \ orcad \ tt.dat", write_mode);
v1: = 1;
v2: = 2;
write (a, v1);
write (a, v2);
file_close (a);

file_open (b, "c: \ vhdl \ orcad \ tt.dat", read_mode);
v1: = 1;
v2: = 2;
read (b, v3);
read (b, v4);
v <= v3;
```

```
W <= v4;
file_close (b);
wait;
  end process;
end architecture;
```

3.27 Особливості роботи паралельних та послідовних операторів

За синтаксом і правилами виконання безумовне паралельне присвоювання збігається з послідовним присвоєнням значення сигналу. Варіанти різняться за локалізацією в програмі і характеризуються різними умовами виконання. Найбільш істотні відмінності:

– паралельне присвоювання локалізується в загальному розділі архітектурного тіла, а послідовне – тільки в тілі процесу;

– послідовне присвоювання сигналу виконується після того, як ініційовано виконання процесу і виконані всі попередні оператори в тілі процесу;

– оператор паралельного присвоювання виконується відразу (з точки зору модельного часу) після зміни сигналів у правій частині цього оператора.

В обох випадках результати присвоєння спочатку фіксуються в драйвері сигналу і передаються сигналу, тобто зміни можуть впливати на інші оператори, тільки після виконання всіх операторів і процесів, що ініційовані однією подією, або через інтервал модельного часу, який заданий опцією **after**.

Перелельне умовне присвоювання і присвоювання за вибором багато в чому схожі з послідовними умовним оператором і оператором вибору, відповідно – дії, що описуються, виконуються за певних умов. Відмінність, крім єдиних для всіх паралельних і послідовних операторів властивостей, полягає в тому, що послідовні умовний оператор і оператор вибору є складовими, тобто умова може задавати в них виконання послідовності дій, а в операторах паралельного присвоювання – тільки привласнення одного значення.

3.28 Типи опису архітектур об'єкта в мові VHDL

У мові VHDL існують два основних рівні опису архітектури об'єктів – поведінковий і структурний.

На поведінковому рівні опис об'єктів проекту видається в VHDL у вигляді набору паралельних процесів. Організація процесів забезпечується введенням оператора процесу і оператора паралельного присвоювання сигналів, який також є процесом. Це визначає дві форми опису об'єктів на поведінковому рівні – універсальну (процесову), та спрощену (потокову).

У процесній формі опис об'єкта проводиться за допомогою процесів. Процес в VHDL визначає незалежну повторювану послідовність операторів і є поведінкою деякої частини проекту. Після того, як останній оператор послідовності буде виконаний, виконання починається знову з першого оператора процесу. Універсальність процесорної форми поведінкової архітектури є в можливості використання широкого набору загально-алгоритмічних послідовних операторів мови VHDL. Поведінковий рівень опису об'єктів проекту є базовим, оскільки будь-які схеми, принаймні на найнижчому рівні, представлені елементами, реалізація яких виконана на рівні їхньої поведінки. Основним оператором тут є оператор process.

У потоковій формі опису об'єкта проекту його архітектуру подано у вигляді набору паралельних операцій з використанням механізму дельта-затримки та ін. – фактично, спрощена форма поведінкового опису на рівні виконання міжрегістрових передавань. Основним оператором тут є оператор паралельного присвоєння сигналу <=.

На структурному рівні об'єкт поданий у вигляді ієрархії пов'язаних компонентів, на нижчому рівні компоненти реалізуються за допомогою поведінкового опису, як це наведено у прикладі в підрозділі 2.2. У структурному описі після оголошення архітектури ідуть оголошення компонентів, що використовуються в архітектурі. У тілі опису архітектури для створення екземплярів компонентів використовуються пропозиції конкретизації компонентів. Пропозиція конкретизації компонента починається з мітки, за якою іде ім'я компонента, а потім, якщо це необхідно, оператори призначення карти параметрів (generic map) під час моделювання схеми (без синтезу) і карти портів (port map). Крім конкретизації кожного компонента окремо, існує можливість множинної конкретизації компонента за допомогою оператора generic:

Також допускається розміщення в одному архітектурному тілі змішаного (mixed) опису архітектури, тобто можливе одночасне використання процесної, потокової і структурної форм, що в деяких випадках значно спрощує реалізацію об'єкта моделювання, або синтезу.

Розглянемо описи архітектури простішої комбінаційної схеми, що зображено на рис. 3.15, різними стилями опису архітектури.

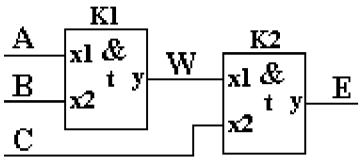


Рисунок 3.15 – Приклад простішої комбінаційної схеми

Приклад застосування поведінкової процесової архітектури:

ENTITY Comb_1 IS

```
PORT(A,B,C: IN BIT; E:OUT BIT);
END comb_1;
ARCHITECTURE STRUCT_1 OF Comb_1 IS
SIGNAL W: BIT;
SIGNAL W: BIT;
BEGIN
PROCESS (A, B)
BEGIN
W<=A AND B;
END PROCESS;
PROCESS (C, W)
BEGIN
E<=C AND W;
END PROCESS;
END ARCHITECTURE STRUCT_1;
```

Приклад застосування поведінкової потокової архітектури:

```
ENTITY Comb_2 IS
PORT(A,B,C: IN BIT; E:OUT BIT);
END comb_2;
ARCHITECTURE STRUCT_1 OF Comb_2 IS
SIGNAL W: BIT;
SIGNAL W:BIT;
BEGIN
W<=A AND B AFTER 2 NS;
E<=C AND W AFRER 5NS;
END ARCHITECTURE STRUCT_2;
```

Далі буде розглянуто опис схеми на структурному рівні. Щоб опис схеми міг застосовуватися у великому числі проектів, він має бути адаптованим та налаштованим до умов конкретного застосування. Основними параметрами, що настроюються, як правило, є час затримки компонента і розрядність вхідних / вихідних сигналів. Для настройки цих параметрів застосовується механізм параметрів, що настроюються generic.

Опис настроювання Generic – констант має розташовуватися в розділі опису об'єкта. Після ключового слова Generic у круглих дужках мають бути вказані настроювані параметри. Приклад опису об'єкта наводиться нижче:

```
Entity obj1 is
Оператор настройки параметра (може змінюватися)
Generic (sw_time: time: = 5 ns);
Порти описують вхідні і вихідні сигнали
Port (portlist)
End obj1.
```

При цьому початкові значення для настроювання констант задаються за допомогою опреатора **:=**.

В ході опису компонент для настройки параметрів необхідно застосовувати оператор конкретизації generic map (parameter_list). Формати запису оператора generic map (parameter_list) збігаються з ключовим і позиційним форматами оператора конткретізації портів компонента Port_map. При цьому початкові значення параметрів, що настроюються, які вказуються через оператор **:=**, ігноруються.

Структурний опис архітектури дозволяє описати систему як сукупність компонентів – підсистем, об'єднаних сигналами. Підсистеми також можуть бути подані сукупністю підсистем.

Щоб один об'єкт був включений до складу іншого об'єкта, його необхідно декларувати як компонент.

Декларація компонента

COMPONENT component_name

[GENERIC (parameter_list);]

PORT (port_list);

END COMPONENT [name];

У структурному описі архітектури відбувається "складання" системи з окремих компонентів за допомогою операторів конкретизації компонента:

Component_label: COMPONENT name

[Generic map (parameter_list)]

port map (port_list);

Ключова відповідність generic – параметри і порти задаються за допомогою таких операторів:

COMPONENT component_name

[GENERIC (parametr_name1: type, parametr_name2: type);]

PORT (port_name1: type; port_name2: type, port_name3: type);

End component;

.....

label1: component_name generic map (parametr_name1 => value1, parametr_name2 => value2 ...)

Port map (port_name => signal, port_name => signal ...);

Для конкретизації значень використовується оператор =>.

Позиційна відповідність портів задається у вигляді:

label1: component_name [generic map (value 1, value2 ...)]

Port map (signal1, signal2 ...);

При ключовій відповідності значення generic-параметрів і сигналів в операторі конкретизації компонента задаються в порядку опису параметрів і портів в описі компонента. Приклад опису схеми, в якій конкретизація компонента К1 задається за допомогою ключової, а К2 за допомогою позиційної відповідності наводиться нижче (затримки у константах GENERIC застосовуються із наведеного вище прикладу):

```
ENTITY Comb_3 IS
PORT(A,B,C: IN BIT; E:OUT BIT);
END comb_3;
ARCHITECTURE STRUCT OF Comb IS
COMPONENT I
GENERIC (t: time);
PORT(X1,X2:IN BIT;
Y:OUT BIT);
END COMPONENT;
SIGNAL W:BIT;
BEGIN
K1: I GENERIC MAP (t=>2 ns) PORT MAP(x1=>A, x2=>B, y=>W);
K2: I GENERIC MAP (5 ns) PORT MAP(C, W, E);
END STRUCT;
ENTITY I IS
GENERIC (t: time);
PORT(X1,X2: IN BIT; Y:OUT BIT);
END I;
ARCHITECTURE BEHI OF I IS
BEGIN
Y<=X1 AND X2 AFTER t;
END BEHI;
```

Для полегшення процесу проектування опис об'єкта моделювання і різні описи його архітектури зазвичай розміщують в бібліотеках. Бібліотеки можуть бути використані в різних проектах, що дозволяє оптимально організувати використання вже розроблених об'єктів. Крім об'єктів в бібліотеках можуть перебувати описи констант, змінних, типів, процедур і функцій.

Формат операції посилання на бібліотеку наводиться нижче:

```
library library_name;
use library_name.package_name.all;
```

Службове слово ALL, вказує, що використовуються всі об'єкти з бібліотеки. В іншому випадку необхідно вказати ім'я конкретного об'єкта.

Наприклад, якщо є бібліотека lib1 і компонент NE, що міститься в ній, то наступний приклад ілюструє використання компонента NE в структурному описі архітектури об'єкта.

```
library lib1;
use lib1.all;
architecture arh1 of .....
K1: entity NE port map (...)
....
```

В ході побудови повнорозмірних моделей на VHDL доводиться мати справу з великою кількістю програмних об'єктів. Тому доцільно об'єднати логічно пов'язані між собою описи компонентів, типів, змінних, констант,

функцій і процедур. Для цього в VHDL введений механізм пакетів (package). Посилання на пакет, що розміщується в бібліотеці, здійснюється у відповідність з форматом:

library library_name;
use library_name.package_name.all;

Як приклад можна розглянути посилання на стандартний пакет std_logic_1164, що входить до складу бібліотеки ieee:

Library IEEE;
use ieee.std_logic_1164.all;

В ході опису реальних схем можна застосовувати як стандартні, так і спеціалізовані (комерційні) бібліотеки та пакети.

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Назвіть основні паралельні оператори в мові VHDL.
2. Назвіть основні послідовні оператори в мові VHDL.
3. Назвіть основні особливості оператора process.
4. Поясніть механізм роботи процесів у мові VHDL.
5. Поясніть особливості поведінкового опису архітектури об’єкта моделювання мовою VHDL.
6. Поясніть особливості потокового опису архітектури об’єкта моделювання мовою VHDL.
7. Поясніть особливості структурного опису архітектури об’єкта моделювання мовою VHDL.
8. Поясніть принципи роботи з файлами у мові VHDL.
9. Які особливості існують в ході реалізації паралельних та послідовних оперторів у мові VHDL.
10. Назвіть основні типи опису архітектур об'єкта в мові VHDL.
11. Поясніть особливості роботи оператора wait.
12. Поясніть відмінність між паралельними та послідовними операторами VHDL, наведить приклади.