МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

О.Г. Аврунін, Т.В. Носова, В.В. Семенець

«ОСНОВИ МОВИ VHDL ДЛЯ ПРОЕКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ НА ПЛІС»

Навчальний посібник

ЗАТВЕРДЖЕНО
Вченою радою Харківського
національного університету
радіоелектроніки.
Протокол №5/14 від 10.04.2018

Харків 2018

УДК 004.432.2

Рецензенти:

А.Д. Черенков, д-р. техн. наук, професор, професор кафедри біомедичної інженерії та теоретичної електротехніки Харківського національного технічного університету сільського господарства ім. П. Василенка;

С.В. Павлов, д-р техн. наук, проф., проректор з наукової роботи, проф. кафедри біомедичної інженерії Вінницького національного технічного університету.

Аврунін О.Г. «Основи мови VHDL для проектування цифрових пристроїв на ПЛІС»: навч. посібник / О.Г. Аврунін, Т.В. Носова, В.В. Семенець. – Харків: ХНУРЕ, 2018. – 196 с.

ISBN 978-966-659-247-0
DOI: 10.30837/978-966-659-247-0

У даному навчальному посібнику розглянуто елементну базу сучасних ПЛІС, мову опису апаратури VHDL, основи синтаксису, типи даних, класи об'єктів, основні оператори мови VHDL. Кожний розділ супроводжується контрольними запитаннями та завданнями. Теоретичні розділи доповнено лабораторним практикумом та бібліотекою компонентів. Навчальний посібник також містить модульні контрольні завдання з дисципліни.

Рекомендується студентам усіх радіотехнічних спеціальностей для студентів денної та заочної форм навчання.

Основи мови VHDL для проектування цифрових пристроїв на ПЛІС
Аврунін О.Г., Носова Т.В., Семенець В.В.
DOI: 10.30837/978-966-659-247-0

УДК 004.432.2

ISBN 978-966-659-247-0

ЗМІСТ

ВСТУП 5
1 ЕЛЕМЕНТНА БАЗА СУЧАСНИХ ПЛІС 6
1.1 Актуальність проблеми розробки цифрових пристроїв на ПЛІС 6
1.2 Пристрої на основі програмованих логічних матриць ПЛМ 7
1.3 Пристрої на основі програмованої матричної логіки ПМЛ 9
1.4 Пристрої на основі складних програмованих логічних пристроїв СПЛП 9
1.5 Пристрої на базових матричних кристалах БМК 12
1.6 Пристрої на основі програмованих користувачем вентильних матриць FPGA 14
1.7 Перспективи розвитку архітектури ПЛІС 17
1.8 Системний підхід у ході проектування цифрових пристроїв на ПЛІС 18
1.9 Методика і засоби автоматизованого проектування цифрових пристроїв на ПЛІС 20
1.10 Можливості мов опису апаратури HDL 22
КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ 22
2 МОВА ОПИСУ АПАРАТУРИ VHDL. ОСНОВИ СИНТАКСИСУ, ТИПИ ДАНИХ, КЛАСИ ОБ'ЄКТІВ 24
2.1 Діючі стандарти мови VHDL 24
2.2 Ознайомлювальний проект цифрового пристрою на основі VHDL – опису 25
2.3 Алфавіт мови 29
2.4 Лексичні елементи мови VHDL 29
2.5 Класифікація типів даних у мові VHDL 32
2.6 Прості типи даних 33
2.7 Підтипи даних 36
2.8 Складові типи даних 36
2.9 Вказівний тип даних (**access**) 38
2.10 Файлові типи даних 38
2.11 Класи об'єктів у VHDL 39
2.12 Атрибути об'єктів 42
2.13 Операції у виразах 45
2.14 Основні прийоми роботи з векторними типами даних 51
2.15 Принципи роботи з багатовимірними масивами 54
КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ 56
3 ОСНОВНІ ОПЕРАТОРИ МОВИ VHDL 58
3.1 Основи функціонування апаратно-орієнтованої частини алгоритмічного ядра мови VHDL 58

3.2 Основи синтаксису мови VHDL. Поняття об'єкта моделювання 62
3.3 Структура опису об'єкта моделювання на VHDL..... 62
3.4 Опис інтерфейсу об'єкта моделювання..... 64
3.5 Особливості архітектури об'єкта моделювання..... 65
3.6 Паралельні оператори..... 67
3.7 Оператор паралельного призначення сигналу..... 67
3.8 Поняття дельта-затримки в ході призначення сигналу..... 68
3.9 Інерційна, режекційна і транспортна затримки в ході призначення сигналу..... 70
3.10 Оператор умовного паралельного призначення сигналу..... 71
3.11 Оператор вибіркового паралельного призначення сигналу 71
3.12 Оператор конкретизації компонента..... 72
3.13 Оператор генерації компонентів..... 73
3.14 Оператор паралельного виклику процедури..... 74
3.15 Оператор блоку..... 78
3.16 Оператор процесу..... 80
3.17 Послідовні оператори 81
3.18 Оператор послідовного присвоєння сигналу 82
3.19 Оператор послідовного присвоєння змінної 83
3.20 Оператор очікування..... 83
3.21 Оператор послідовного умовного призначення сигналу 84
3.22 Оператор вибору 85
3.23 Організація циклів..... 87
3.24 Пустий оператор..... 89
3.25 Функції перетворювання типів 89
3.26 Принципи роботи з файлами..... 90
3.27 Особливості роботи паралельних та послідовних операторів 91
3.28 Типи опису архітектур об'єкта в мові VHDL 91
КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ 96
4 ЛАБОРАТОРНИЙ ПРАКТИКУМ..... 97
Лабораторна робота № 1 105
Лабораторна робота № 2 116
Лабораторна робота № 3 121
Лабораторна робота № 4 126
Лабораторна робота № 5 132
Лабораторна робота № 6 139
БІБЛІОТЕКА КОМПОНЕНТІВ 145
МОДУЛЬНІ КОНТРОЛЬНІ ЗАВДАННЯ З ДИСЦИПЛІНИ 172
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 195

ВСТУП

У сучасних умовах, на порозі четвертої науково-технічної революції, в основі якої лежить концепція повністю мережного і адаптивного виробництва з інтелектуальними системами, є потреба в принципово нових підходах щодо розробки електронної апаратури. Однак, щоб повністю використовувати потенціал Індустрії 4.0, необхідні зміни на ринку праці відповідно до нових вимог та, відповідно, в концепціях навчання, оскільки підготовка досвідчених фахівців – це ключовий фактор успіху.

Проектування складних електричних схем – складний процес, який пов'язаний з необхідністю чіткого і вичерпного опису створених рішень, контролем функціонування на відповідність технічного завдання, труднощами супроводу і модернізації. Традиційно системи автоматизованого проектування САПР (CAD - Computer Aided Design) цифрових пристроїв забезпечені модулями схемного редагування проектів. Однак під час розробки пристроїв на програмованих логічних інтегральних схемах ПЛІС (Програмованих Логічних Інтегральних Схемах), складність яких оцінюється десятками і сотнями тисяч еквівалентних вентилів, схемне подання не забезпечує наочного сприйняття проекту. Тому, в останнє десятиріччя набули істотного розвитку засоби текстового опису цифрових пристроїв на ПЛІС, реалізовані у вигляді мов опису апаратури (Hardware Description Languages - HDL).

Мова опису апаратури Very high speed integrated circuits Hardware Description Language (VHDL) була розроблена в 1983 році на замовлення Міністерства оборони США з метою формального опису логічних схем для реалізації усіх етапів проектування електронних систем. Сьогодні VHDL, що відповідає стандарту ANSI/IEEE Std 1067-1993, є найбільш поширеною мовою опису апаратури в країнах Європи і США. Застосовується як для моделювання, так і для синтезу цифрових пристроїв, і поряд з мовами Verilog і AHDL є базовою під час розробки апаратури сучасних обчислювальних систем.

У пропонованому посібнику розглядаються питання розробки використання VHDL – описів для проектування цифрових систем на ПЛІС.

У першій частині розглядаються архітектури сучасних ПЛІС, а також системний підхід в ході проектування цифрових пристроїв.

У другій частині наводиться розгорнутий опис типів і конструкцій мови VHDL стандарту ANSI/IEEE Std 1067-1993.

У третій частині розглядаються оператори та принципи опису архітектур мовою VHDL.

У четвертій частині розглядається лабораторний практикум з використанням оригінальних лабораторних макетів на базі ПЛІС FPGA ACEX 1K EP1K50QC208 та програмного середовища розробки MAX+PLUS II.

1 ЕЛЕМЕНТНА БАЗА СУЧАСНИХ ПЛІС

1.1 Актуальність проблеми розробки цифрових пристроїв на ПЛІС

Сьогодні на світовому ринку електронних технологій різноманіття запропонованих схематичних рішень ділиться на 3 основних типи: мікропроцесори, замовні великі інтегральні схеми (ЗБІС) і програмовані логічні інтегральні схеми (ПЛІС).

Економічна ефективність виробництва БІС визначається, виходячи з вартості S виробництва однієї мікросхеми:

$$S = P_1 + \frac{P_2}{N}, \tag{1.1}$$

де P_1 – вартість виконання повного технологічного циклу під час виробництва однієї БІС, включаючи витрати на матеріали;

P_2 – витрати на проектування БІС;

N – кількість випущених пристроїв.

У формулі (1.1) параметр P_1, як правило, змінюється незначно. Тому очевидно, що істотно знизити загальну вартість виробництва БІС можна або шляхом організації їх великосерійного (масового) виробництва, що призводить до збільшення N, або зниження витрат P_2 на проектування БІС за допомогою застосування ефективних САПР.

Процесори і мікросхеми пам'яті великого обсягу є стандартними пристроями, вони не виготовляються для конкретної системи за спеціальним замовленням і їхні функції залишаються постійними в різних системах. Тому стандартні БІС/ЗБІС мають досить низьку вартість, що визначається масовим виробництвом, а також лідирують за рівнем інтеграції та швидкодії.

Однак у цифровій системі, поряд зі стандартними, присутні й нестандартні компоненти, специфічні для даної розробки. Це стосується блоків управління, пристроїв взаємодії компонентів і т.д. Традиційно, реалізація спеціалізованої частини системи пов'язана з використанням мікросхем малого і середнього рівня інтеграції, що супроводжується різким збільшенням кількості корпусів інтегральних схем, ускладненням монтажу, зниженням надійності і швидкодії системи. Водночас замовити для системи спеціалізовані інтегральні схеми високого рівня інтеграції важко, оскільки пов'язано з великими часовими фінансовими витратами.

У цьому випадку оптимальним рішенням є виробництво універсальних схем, які користувач міг би пристосувати (запрограмувати) для реалізації специфічних логічних функцій. Сьогодні реалізація спеціалізованих функцій здійснюється за допомогою БІС/ЗБІС з програмованою і репрограмованою структурою (програмованих логічних інтегральних схем – ПЛІС). За

принципом програмування ПЛІС діляться на напівзамовні і безпосередньо програмовані користувачем.

Виробництво напівзамовних БІС (ASIC – Application Specific Integrated Circuits) розбивається на дві стадії:

- на першій стадії проектується і випускається в масовій кількості кристал-заготовка – універсальний технічний засіб для реалізації найрізноманітніших пристроїв;

- на другій стадії відбувається реалізація на даному кристалі функцій конкретного пристрою, шляхом виконання додаткових зв'язків напиленням додаткових шарів на вже готову заготовку за індивідуальним рисунком міжз'єднань.

Таким чином, за допомогою декількох додаткових масок універсальний кристал масового випуску програмується на виконання спеціалізованих функцій. Така технологія виробництва обходиться набагато дешевше, ніж виготовлення повністю замовних БІС.

У безпосередньо програмованих користувачем БІС конфігурування кристала-заготовки для виконання спеціалізованих функцій здійснюється шляхом завдання програмних налаштувань через зовнішні контакти ПЛІС без напилення додаткових шарів на кристал. Готовий кристал, укладений в стандартний корпус, необхідно лише запрограмувати без виконання додаткових виробничих циклів. Це істотно підвищує ефективність розробки нестандартних пристроїв на ПЛІС.

1.2 Пристрої на основі програмованих логічних матриць ПЛМ

Вперше принцип розробки пристроїв на мікросхемах програмувальної логіки був реалізований у вигляді створення програмованих логічних матриць ПЛМ (PLA, Programmable Logic Array), що є першими представниками універсальних ПЛІС.

Ідея конструкції ПЛМ грунтується на тому, що будь-яку логічну функцію можна подати у вигляді диз'юнктивної нормальної форми (ДНФ). ДНФ є сумою кон'юнктивних членів, що обертають логічну функцію в 1, і може бути реалізована за допомогою логічних вентилів трьох типів: і (and), або (or), ні (not). Так, диз'юнктивна нормальна форма для функції Y, таблиця істинності якої зображена на рисунку 1.1, наводиться нижче:

$$Y = \overline{X_1}\,\overline{X_2}\,\overline{X_3} + X_1 X_2 \overline{X_3} + X_1 \overline{X_2} X_3 . \qquad (1.2)$$

Як випливає з формули (1.2), у внутрішній структурі ПЛМ мають знаходитися лінії з прямими й інвертованими вхідними сигналами, які можна подати на входи будь-якого вентиля (and), а виходи всіх (and) можуть бути приєднані до входів вентилів (or).

Основними параметрами ПЛМ (див. рис. 1.2) є кількість вхідних сигналів *m*, число термів *l* і число виходів *n*. Вхідні буфери (IB - Input Buffers) формують сигнали необхідної потужності для живлення матриці GAand і перетворять вхідні сигнали x у парафазні. З виходу IB прямі й інвертовані вхідні сигнали подаються на входи кон'юнктерів матриці GAand, на виході якої утворюються *l* термів T (кон'юнктивних членів). Число *l* формованих термів дорівнює числу кон'юнкторів у матриці GAand. Далі терми подаються на входи матриці GAor, тобто на входи диз'юнкторів, які формують вихідні функції. Число *n* реалізованих функцій F дорівнює числу диз'юнкторів у матриці GAor.

X3	X2	X1	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Рисунок 1.1 – Таблиця істинності для функції

$$Y = \overline{X_1}\,\overline{X_2}\,\overline{X_3} + X_1 X_2 \overline{X_3} + X_1 \overline{X_2} X_3$$

Сигнали з виходів матриці Gaor надходять у блок виведення OB (Output Block), який формує прямі й інверсні сигнали вихідних функцій F і забезпечує необхідну навантажувальну здатність виходів. Керуючий сигнал OE (Output Enable) дозволяє або забороняє вихід ПЛМ на зовнішні шини. Таким чином, за допомогою ПЛМ можна реалізувати систему *n* логічних функцій від m аргументів, що містить не більше *l* термів.

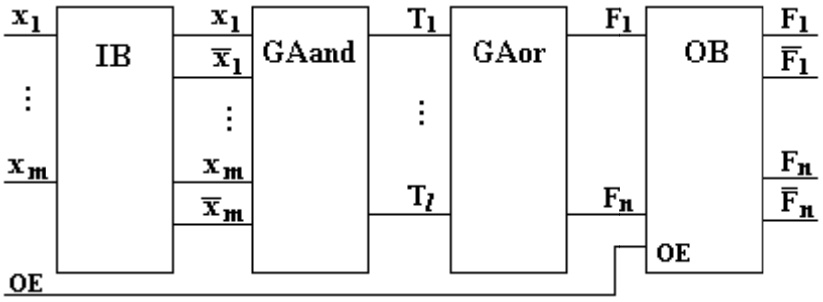


Рисунок 1.2 – Базова структура ПЛМ

ПЛМ випускаються як на основі біполярної технології, так і на МОП-транзисторах. У матрицях є системи горизонтальних і вертикальних зв'язків. В ході програмування у вузлах перетину ліквідуються або створюються елементи зв'язку. У першому випадку, в ході виробництва ПЛМ виконуються всі зв'язки в місці перетину провідників, а під час програмування непотрібні зв'язки видаляються. У другому випадку, специфікація зв'язків здійснюється за допомогою додатково виготовленої маски – фотошаблона. Як приклад на рис. 1.3 наводиться структурна схема ПЛМ з трьома входами і

двома виходами до виконання процесу програмування (відсутність перемички в матрицях GAand і GAor).

Для реалізації функцій:

$$F1 = X_1\overline{X_2}\,\overline{X_3} + \overline{X_1}\,\overline{X_2}\,\overline{X_3}:$$
$$F2 = X_1\overline{X_2}\,\overline{X_3} + X_1X_2X_3$$

виконується створення відповідних перемичок у матрицях GAand і Gaor (див. рис. 1.4). Прикладами таких ПЛІС можуть бути вітчизняні ІС К556РТ1, РТ2, РТ21.

1.3 Пристрої на основі програмованої матричної логіки ПМЛ

В ході реалізації найбільш типові практичні завдання логічної потужності ПЛМ часто використовується не повною мірою (системи перемикальних функцій не мають великих перетинів за однаковими термами). У цих випадках можливість використання виходів будь-яких кон'юнкторів будь-якими диз'юнкторами стає зайвим ускладненням схемотехнічної реалізації. Структури, в яких виходи елементів and жорстко пов'язані з елементами or, називаються **програмованою матричною логікою ПМЛ** (PAL – Programmable Array Logic). Порівняно з ПЛМ, структури ПМЛ мають меншу функціональну гнучкість, з огляду на фіксованій матриці Gaor, але їх виготовлення і використання спрощується. Приклад структурної схеми ПМЛ, що має *m* входів і *n* виходів наведено на рисунку 1.5. У даній схемі міститься 4*n елементів and, тому що кожному елементу or надається структура з чотирьох кон'юнкторів). Інтегральні схеми середнього ступеня інтеграції з ПМЛ випускаються низкою фірм-виробників, таких як INTEL, ALTERA, AMD, LATTICE та ін.

1.4 Пристрої на основі складних програмованих логічних пристроїв СПЛП

Наведені архітектури ПЛІС містять порівняно мало осередків і застосовуються для реалізації відносно простих пристроїв. Подальшим розвитком архітектури ПМЛ сьогодні є структури, що отримали назву **складних програмованих логічних пристроїв СПЛП** (CPLD - Complex Programmable Logic Devices). СПЛУ мають кілька матричних логічних блоків МЛБ (MLB - Matrix Logical Block), об'єднаних комутаційною матрицею КМ (PIA - Programmable Interconnect Array). Кожен матричний логічний блок є структурою типу ПМЛ – програмованою матрицею GAand і фіксованою матрицею GAor. Узагальнена структурна схема СПЛП (CPLD) наводиться на рисунку 1.6.

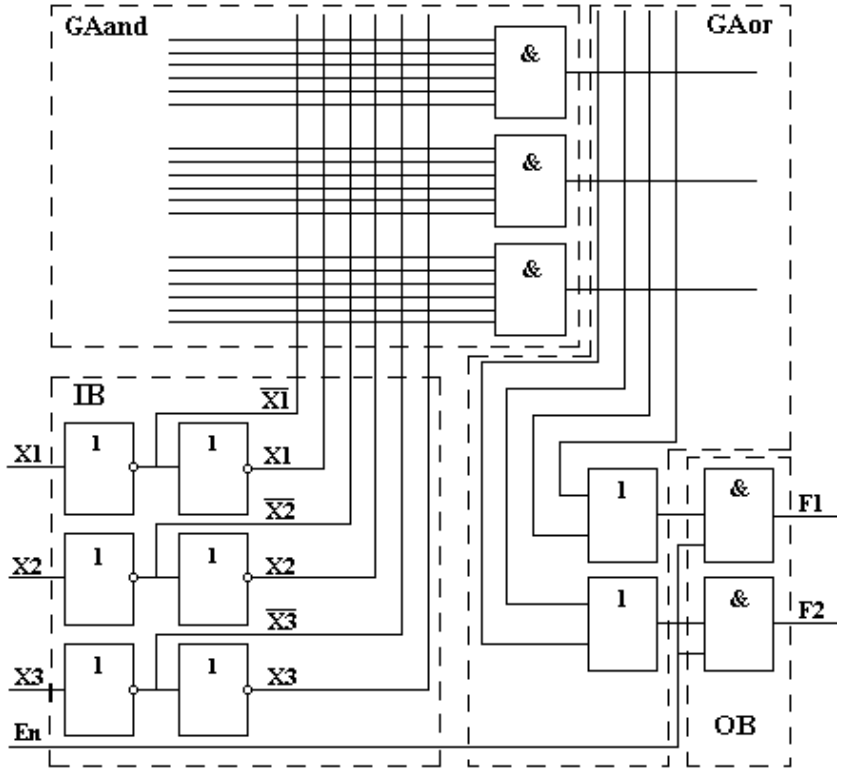


Рисунок 1.3 – Структурна схема ПЛМ до програмування

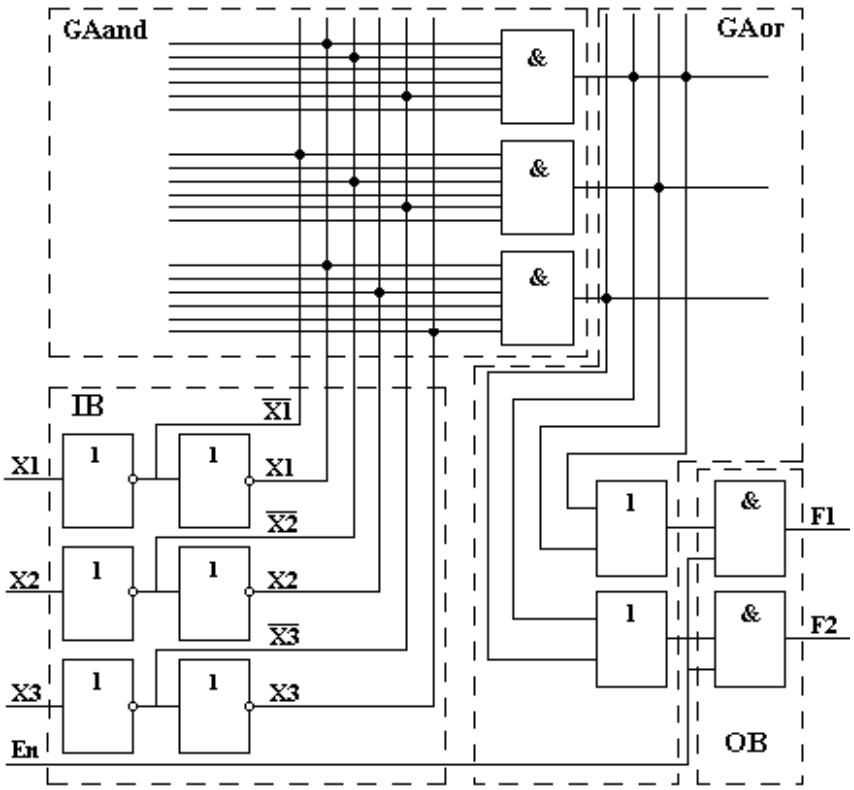


Рисунок 1.4 – Структурна схема ПЛМ після програмування

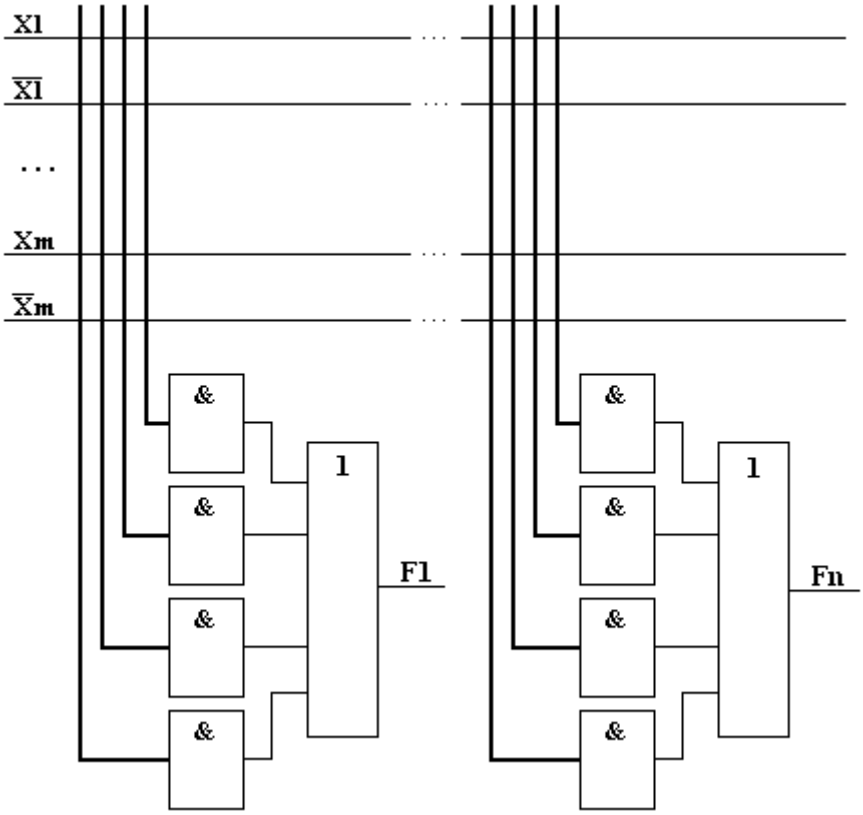


Рисунок 1.5 – Структурна схема ПМЛ

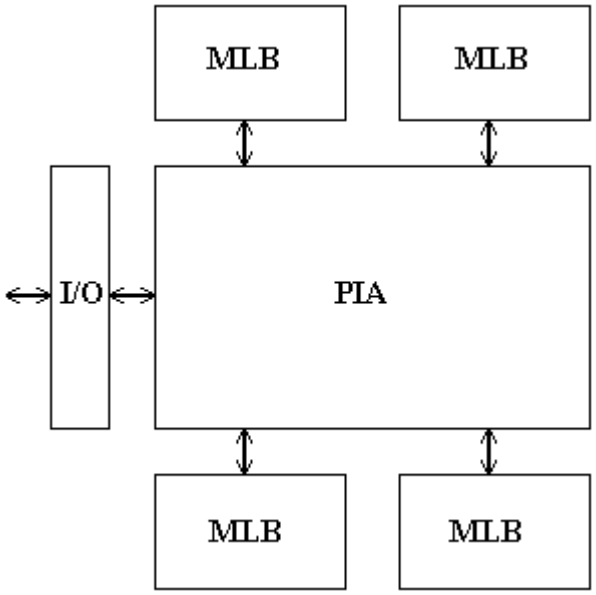


Рисунок 1.6 – Узагальнена структурна схема СПЛУ (CPLD)

Як перемички в матриці міжз'єднань використовуються МОП-транзистори з плаваючим затвором, як в електрично перепрограмуючому ПЗУ (EEPROM). Дані ПЛІС виробляються за технологією Flash – пам’яті.

У СПЛП (CPLD) використовується безперервна система ідентичних зв'язків, що дає можливість гарної передбачуваності затримок сигналів у провідниках. Це істотно полегшує проектування і виготовлення роботоспроможних схем високої швидкодії. ПЛІС типу CPLD, як правило, мають високу ступінь інтеграції (до

12000 вентилів, до 560 макроосередків). Відомими пристроями, що мають архітектуру СПЛП (CPLD), є мікросхеми MAX 5000 MAX 7000 фірми Altera. Мікросхеми типу CPLD доцільно використовувати для реалізації цифрових автоматів, керуючих і інтерфейсних схем.

1.5 Пристрої на базових матричних кристалах БМК

Альтернативою ПЛІС на програмованих логічних матрицях є архітектура пристроїв на базових матричних кристалах БМК (GA – Gate Array). Основою БМК є сукупність регулярно розташованих на кристалі базових осередків (БО) – наборів елементів схем, що регулярно повторюються по площі кристала і займають центральну область. На периферії розташовані базові осередки введення/виведення, орієнтовані на реалізацію зовнішніх зв'язків БМК. Таким чином, БМК є кристалом-заготовкою, який перетворюється в необхідну схему виконанням відповідних з'єднань. Споживач може реалізувати на основі БМК безлічі пристроїв певного класу, задавши для кристала той чи інший варіант рисунка міжз'єднань компонентів.

В ході проектування БМК намагаються оптимальним чином збалансувати число базових осередків, трасувальні ресурси кристала і число контактних майданчиків для підключення зовнішніх висновків. Трасувальні можливості БМК визначаються площею, що відводиться для міжз'єднувальних зв’язків в ортогональних напрямках, і кількістю шарів міжз'єднань. Недостатня трасувальна здатність веде до зменшення кількості задіяних у ході побудови схеми базових осередків. Надлишкова трасувальна спроможність призводить до нераціонального використання площі кристала, що знижує рівень інтеграції БМК і підвищує його вартість.

Проведемо уточнення термінології, використовуваної під час проектування БМК.

Базовим осередком (БО) є деякий набір схемних елементів (скомутованих або нескомутованих), регулярно повторюваних на певній площі кристала.

Базові осередки внутрішньої області БМК називаються **матричними базовими осередками** (МБО).

Базові осередки периферійної області кристала називаються **периферійними базовими осередками** (ПБО).

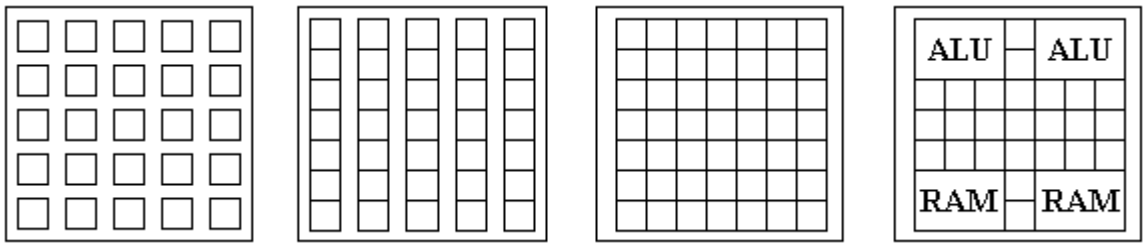
З елементів МБО може бути сформований один логічний елемент, а для реалізації складних функцій використовуються кілька осередків. При цьому з елементів МБО може бути сформовано будь-який функціональний вузол, а склад елементів осередку визначається схемою найскладнішого вузла.

Функціональний осередок (ФО) – функціонально закінчена схема, реалізована шляхом з'єднання елементів у межах однієї або декількох БО.

Канали трасування – області на БМК для можливого розміщення міжз'єднань.

За принципом розміщення базових осередків БМК діляться на канальні, безканальні і блокові (див. рис. 1.7). У канальних БМК (див. рис. 1.7, а) в центральній області кристала розташовані базові осередки (позначені квадратами) і канали трасування. У канальних БМК великі можливості зі створення зв'язків нівелюються низькою площею упаковки через значні затрати площі кристала на області міжз'єднань. Для підвищення ефективності упаковки базових осередків канали можуть виконуватися тільки вертикальними (див. рис. 1.7, б).

Пошук шляхів створення БМК високого рівня інтеграції призвів до безканальної архітектури БМК. Внутрішня область такого БМК містить щільно упаковані ряди базових осередків і не має фіксованих каналів для трасування міжз'єднань (див. рис. 1.7, в). У кристалі цього типу будь-яка область, в якій розташовані БО, може бути використана як для створення логічної схеми, так і для створення міжз'єднань. Внаслідок більш раціонального розташування зв'язків у безканальних БМК зменшується затримка передачі сигналу по зв'язках (довжини і паразитні ємності міжз'єднань зменшуються).



а) б) в) г)

Рисунок 1.7 – Структури БМК різних типів:
а) з горизонтальними і вертикальними каналами;
б) з вертикальними каналами; в) безканальні, г) блокові

Безканальні БМК реалізуються у варіантах «море вентилів» і «море транзисторів». У першому випадку кристал містить масив закінчених логічних елементів, у другому – масив транзисторів. Для підвищення ефективності реалізації на БМК функціонально-складних пристроїв, які мають не тільки комбінаційні схеми, але і значне число елементів пам'яті, в сучасні БМК високого ступеня інтеграції, крім базових осередків, включають спеціалізовані блоки пам'яті, помножувачі, схеми прискореного перенесення і т.д., реалізовані на рівні топології кристала-заготовки. Такі БМК називають **блоковими** (див. рис. 1.7, г).

БМК належать до категорії напівзамовних ПЛІС. Це означає, що в масовій кількості виробляються заготовки (напівфабрикати) БІС, а надання кристалам-заготовкам індивідуального характеру здійснюється на заключній стадії виробництва. В ході проектування цифрових пристроїв на БМК доцільно використовувати бібліотеки функціональних осередків, які містять готові

схемні рішення, що створюються розробником мікросхеми з урахуванням фізичних характеристик кристала.

Для оцінки логічної складності БМК вводиться поняття – **еквівалентного вентиля** – групи елементів БМК, відповідної за можливостями для реалізації логічної функції двовходового вентиля «І-НІ» (або «АБО-НІ»). Логічна складність сучасних БМК високого рівня інтеграції складає близько 10^4 еквівалентних вентилів і більше.

1.6 Пристрої на основі програмованих користувачем вентильних матриць FPGA

Подальшим розвитком архітектури БМК є **програмовані користувачем вентильні матриці** (FPGA – Field Programmable Gate Array). При цьому FPGA, на відміну від БМК, програмуються користувачем «на місці» без виконання додаткових виробничих циклів.

Внутрішня організація FPGA схожа з організацією канальних БМК (масив базових осередків на внутрішній площі кристала і масив периферійних осередків по периметру, трасувальні канали для завдання зв'язків між базовими осередками). Відмінність FPGA від БМК полягає в структурі цих компонентів кристала.

Розглянемо докладно топологію кристала ПЛІС (FPGA) серії Virtex фірми Xilinx (див. рис. 1.8). На площі кристала ПЛІС розміщені матриця конфігурованих логічних блоків КЛБ (CLB – Configure Logical Block), матриця відрізків міжз'єднань, покритих матрицями з польових транзисторів – перемичок МП (IB - Interconnect Block), блоки настроюваних ОЗУ (RAM); блоки введення/виведення сигналів (IOB - Input / Output Block) і периферійний канал міжз'єднань з мінімальною затримкою (VR - VersaRing), розміщені по периметру кристала. Для забезпечення достатніх можливостей маршрутизації міжз'єднань і мінімальної затримки при передачі сигналів використовується до дев'яти шарів металізації.

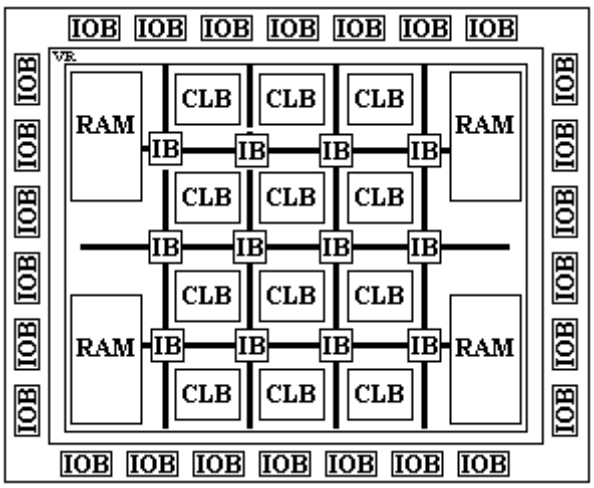


Рисунок 1.8 – Топологія кристала ПЛІС серії Virtex фірми Xilinx

Базовий осередок в архітектурі FPGA набуває вигляду КЛБ (CLB – Configurable Logical Block), основним логічним елементом якого є логічна таблиця ЛТ (LUT - Look-Up Table), що становить однобітний ОЗУ, призначена для зберігання логічної функції від n-аргументів. Фактично в пам'ять безпосередньо заноситься таблиця істиності булевої функції, причому набір значень аргументів використовується як адреса блоку пам'яті, за яким на вихід надходить значення функції. Для завдання таблиці істинності логічної функції від n аргументів потрібно однобітний блок пам'яті, який має 2^n осередків. Позначення чотиривходового блоку ЛТ (LUT) для завдання функції «І» від чотирьох аргументів (16 осередків пам'яті) за таблицею істинності наведено на рисунку 1.9.

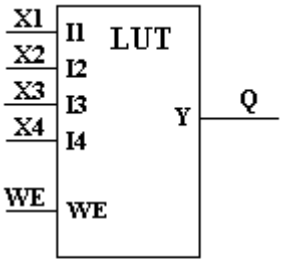
До складу КЛБ зазвичай входять кілька ЛТ, а також комбінаційні та тригерні схеми (див. рис. 1.10).

КЛБ характеризуються такими властивостями:

- **функціональність** (functionality) визначає наскільки великі логічні можливості одного КЛБ;

- **зернистість** (granularity) визначає наскільки функціонально і схемотехнічно простими будуть елементи, складові КЛБ.

Прикладом дрібнозернистого КЛБ може служити ЛБ фірми Crosspoint Solutions, що містить ланцюжки транзисторів; прикладом великозернистого КЛБ можуть служити ЛБ, у мікросхемах Xilinx XC4000, що містять 2 ЛТ на 4 аргумента, 1 ЛТ на 3 аргумента, 2 D - тригери, пов'язані через кілька мультиплексорів, а також логічні схеми вентильного рівня.



X4	X3	X2	X1	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

а) б)

Рисунок 1.9 – Позначення чотиривходового блоку LUT (а) і таблиці істинності функції «І» від чотирьох аргументів (б)

Дрібнозернистість КЛБ веде до більшої гнучкості їхнього використання, можливості реалізувати відтворювані функції різними способами. Водночас дрібнозернистість ускладнює систему міжз’єднань FPGA в зв'язку з великим числом програмованих точок зв'язку.

Мікросхеми FPGA, як правило, мають складну ієрархічну систему зв'язків, що включає зв'язок загального призначення (general-purpose interconnects), довгі

лінії (long lines) для передач на великі відстані з малою затримкою, прямі зв'язки (direct interconnects) і лінії тактування (clock lines).

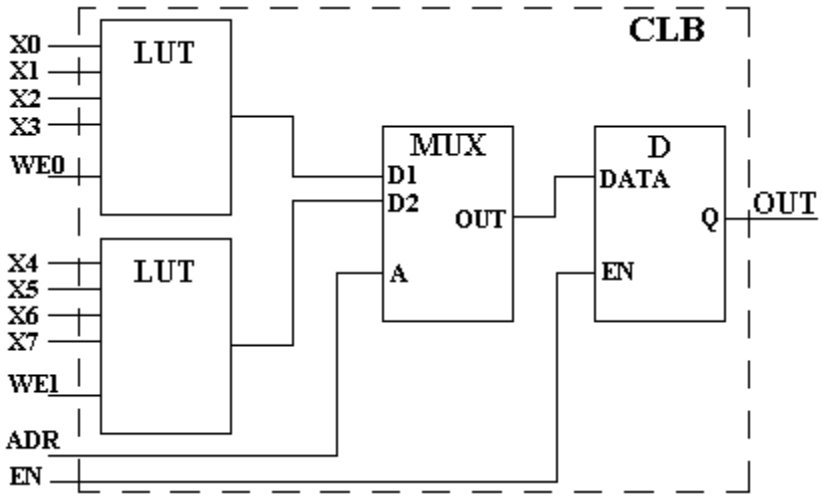


Рисунок 1.10 – Структурна схема КЛБ (CLB)

Універсальність і зручні налаштування КЛБ вимагають накладних витрат і не кращим чином позначаються на швидкодії пристроїв на FPGA. Підвищення технічних характеристик у цих пристроях забезпечується зануренням у кристал готових функціональних блоків (блоків пам'яті, помножувачів, процесорних ядер), спроектованих і оптимізованих на рівні схемотехніки і топології ЗБІС. Подібну структуру мають мікросхеми FPGA Xilinx Virtex-II Pro.

Незважаючи на обмежені функціональні та швидкісні можливості порівняно з напівзамовними ПЛІС, FPGA є перспективним середовищем реалізації цифрових пристроїв. Важливою перевагою FPGA є простота конфігурування мікросхеми. При включенні живлення сигнали конфігурації (прошивка), що зберігаються в ПЗУ, які стоїть поруч з ПЛІС, переписуються в спеціальний зсувний регістр ПЛІС, до виходів якого підключені затвори всіх транзисторів, що «програмують» (див. рис. 1.11).

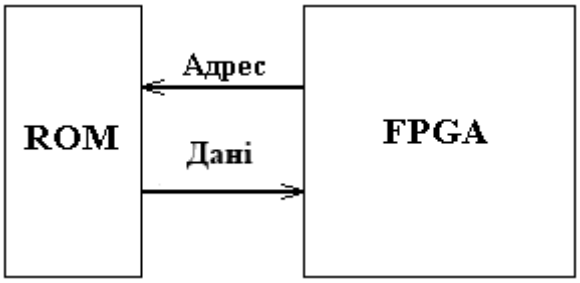


Рисунок 1.11 – Структурна схема цифрового пристрою на основі FPGA

Найбільш ефективно можливості мікросхем FPGA проявляються в ході використання бібліотек компонентів, що поставляються розробником ПЛІС і

оптимізованих за затримками і займаними ресурсами з урахуванням топологічних властивостей кристала.

Порівняно з СПЛУ (CPLD), FPGA мають переваги, пов’язані з необмеженою кількістю перепрограмування, більшою питомою ємністю вентилів на цент, малим енергоспоживанням, більш високою надійністю.

1.7 Перспективи розвитку архітектури ПЛІС

Розвиток архітектури ПЛІС йде по шляху створення комбінованих структур, що поєднують переваги FPGA і CPLD. Такі пристрої отримали назву ЗБІС програмованої логіки змішаної архітектури (FLEX - Flexible Logic Element Matrix). Зберігши ряд характеристик, присутніх у CPLD, мікросхеми FLEX мають логічні елементи табличного типу ЛТ (LUT), їхні логічні блоки розташовані у вигляді матриці з горизонтальними і вертикальними трасувальними каналами, що характерно для FPGA. При цьому наявність безперервних зв'язків, що типово для CPLD, дає гарну передбачуваність і малі величини затримок поширення сигналів. Характерними представниками мікросхем цього типу є ПЛІС фірми Altera FLEX 8000, FLEX 10K.

Перші ПЛІС, що виготовляються за технологією ТТЛШ (до середини 80-х років) характеризувалися високою швидкодією (менше 10 нс), низькою вартістю, великою споживчою потужністю, малим ступенем інтеграції, неможливістю перепрограмування. В кінці 80-х років набули широкого поширення ПЛІС, виконані за технологією КМОП. У таких ПЛІС роль сполучних елементів (перемичок) відіграють осередки пам'яті типу EPROM або EEPROM. Якщо в біполярних ПЛІС з'єднання розриваються шляхом звичайного запису перемички, то в КМОП-ПЛІС осередку програмуються за рахунок накопичення або видалення електричного заряду. Такі перемички можна не тільки розривати, але і відновлювати. Цей процес називається стиранням схеми. Залежно від типу елементів пам'яті розрізняють ПЛІС з ультрафіолетовим (УФ) стиранням (EPROM) і електричним стиранням (EEPROM). ПЛІС з УФ-стиранням виготовляються в керамічних корпусах з вікном. Стирання відбувається при опроміненні ПЛІС УФ-випромінюванням із заданими параметрами. Стирання ПЛІС типу EEPROM виконується шляхом подачі на схему певних електричних сигналів (15-25В). Впровадження технології КМОП дозволило значно збільшити ступінь інтеграції ПЛІС і досягти 10000 і більше вентилів. Споживання КМОП-схем становить близько 1 мА/МГц, а деякі ПЛІС мають режим мікроамперного споживання в статичному режимі.

Ряд технологічних удосконалень і зменшення топологічних норм проектування довели рівень інтеграції сучасних ЗБІС програмованої логіки до величин у кілька мільйонів еквівалентних вентилів при робочій тактовій частоті до 500 МГц. Завдяки перерахованим вище параметрам, на кристалі ЗБІС ПЛ стало можливим розмістити схему високої складності – систему, що включає мікропроцесор, пам'ять, схеми сполучення і т.д. Такі системи отримали назву

систем на кристалі. При цьому різні за функціональністю блоки реалізуються одними і тими ж апаратними засобами, з огляду на їхню програмованість. Системи різного призначення містять, як правило, типові компоненти, що ставить питання про необхідність введення в ЗБІС поряд зі структурами програмованої логіки спеціалізованих областей із заздалегідь визначеними функціями – апаратних ядер (Hardcores). Введення спеціалізованих апаратних ядер у ПЛІС, маючи ряд позитивних наслідків, призводить до зниження універсальності ПЛІС з Hardcores, порівняно зі стандартними мікросхемами програмованої логіки.

1.8 Системний підхід у ході проектування цифрових пристроїв на ПЛІС

Сьогодні для проектування складних цифрових пристроїв, що мають сотні тисяч і навіть мільйони компонентів, застосовується **системний підхід** (systematic methodology). Він полягає в тому, що проектована система може бути подана кількома різними за формою, адекватністю та ступенем подробиці моделями. Як цифрова система розглядається пристрій, що перетворює або зберігає інформацію.

Системний підхід до проектування цифрових пристроїв передбачає початковий опис всього пристрою у вигляді абстрактної структури, що задовольняє необхідні умови – специфікації. Потім ця структура (велика система) розбивається на підсистеми, кожна з яких реалізує незалежну функцію, але в сукупності ці підсистеми реалізують функції великої системи. На наступному етапі кожна підсистема може бути розбита на складові нижчого рівня. Такі ітерації виконуються до тих пір, доки на найнижчому рівні модель пристрою не складатиметься тільки з найпростіших, неподільних елементів.

Перевагою даного підходу є незалежність проектування кожної з підсистем. При цьому проектувальнику не потрібно працювати з усім обсягом інформації, а тільки з тим, який необхідний для створення конкретної частини проекту.

Основні етапи системного проектування цифрових пристроїв наведені на рисунку 1.12.

Першим етапом проектування є створення технічного завдання (ТЗ), яке зазвичай включає в себе вимоги до робочих характеристик системи, опис інтерфейсу, експлуатаційні характеристики системи (розсіювана потужність). Так само проводиться оцінка ефективності системи, орієнтовний розрахунок вартості готового виробу і т.д.

На основі ТЗ створюється попередня високорівнева функціонально-структурна схема системи з описом алгоритмічних основ роботи кожного компонента.

На третьому етапі отримана функціонально-структурна схема перетворюється в модель рівня реєстрових передач, у якій розглядаються описи регістрів, модулів пам'яті, операційних і керуючих автоматів.

Результатом наступного етапу проектування є створення логічних схем для кожного компонента. При цьому кожен компонент розглядається як система, що складається із сукупності найпростіших логічних вентилів.

Надалі опис логічного рівня перетворюється в схемний (розробляється принципова схема пристрою на рівні транзисторів).

Заключними етапами є проектування топології кристала, обчислення реальних фізичних властивостей пристрою, таких як площа кристала і розсіювана потужність. При цьому відбувається верифікація проектних норм, визначаються реальні фізичні параметри схеми і готується вичерпний опис пристрою для виконання виробничих циклів.

Рисунок 1.12 – Етапи системного проектування цифрових пристроїв

На кожному етапі проектування цифровий пристрій подається різними за повнотою опису і ступеня адекватності моделями, які можна розділити на 3 типи:

1 – **функціональні** (**поведінкові**), що описують принципи функціонування пристрою без урахування його технічної реалізації. На даному рівні формою подання пристрою є булеві рівняння, алгоритми, функції.

2 – **структурні** – дозволяють ввести в опис пристрою поняття структури для реалізації деякої функції. Це дає можливість розширити реалізацію поведінкової моделі до рівня логічних вентилів і підвищити адекватність опису пристрою.

3 – **фізичні** – орієнтовані на подання пристрої в конкретному схемотехнічному базисі, що дозволяє провести найбільш повну оптимізацію апаратної реалізації.

На всіх етапах системного проектування для опису пристрою використовуються компоненти. На найвищому рівні – це невелика кількість складних компонентів, таких як суматори або модулі пам'яті; на нижніх – величезне число елементарних компонентів, таких як логічні вентилі або транзистори. Кожному рівню відповідає своє формальне математичне описання, за яким можна передбачити поведінку розроблювального пристрою. При цьому вивчення фізичних характеристик проекту, а також оптимізацію параметрів схеми доцільно розглядати на нижніх рівнях проектування, хоча для цього потрібен довший час розробки та аналізу.

1.9 Методика і засоби автоматизованого проектування цифрових пристроїв на ПЛІС

Як було показано в попередньому розділі, сучасний рівень розробки цифрових пристроїв передбачає широке використання програмованих логічних інтегральних схем (ПЛІС). Проектування пристроїв високої складності на ПЛІС виконується тільки за допомогою спеціалізованих систем автоматизованого проектування (САПР). При цьому структурна схема основних етапів проектування, зображена на рис. 1.12, перетворюється до алгоритму автоматизованого проектування ПЛІС, наведеному на рисунку 1.13.

На концептуальному рівні проектування визначаються функції пристрою, безлічі вхідних і вихідних сигналів, можливості розбиття проекту на частини. Цей рівень проектування, практично, не пов'язаний з автоматизацією і його реалізація цілком покладається на розробника.

Результати концептуального синтезу вводяться в САПР, в якій виконується компіляція проекту – синтез пристрою в базисі бібліотеки моделей. Компіляція розбивається на ряд послідовних етапів: формування бази даних проекту, контроль з'єднань, логічна мінімізація проекту і т.д. Результатом компіляції є файл, який містить конфігураційну інформацію для заданої ПЛІС.

Скомпільований проект вимагає ретельної перевірки, тому за етапом синтезу йде етап аналізу за допомогою моделювання та теоретичної верифікації. Моделювання здійснюється на декількох рівнях, що характеризуються різним ступенем відображення властивостей реального об'єкта. Так, за допомогою функціонального моделювання можна перевірити правильність логічної структури пристрою. Часове моделювання дозволяє проводити тестування роботи пристрою з урахуванням затримок сигналів у компонентах без урахування остаточної топології трасування.

В результаті моделювання можуть виявитися помилки, які вимагають виправлень, що надає процесу проектування ітеративний характер з поверненнями до попередніх етапів і введення в проект потрібних змін.

Далі проводиться конфігурація ПЛІС, після чого можливе проведення етапу фізичного моделювання – перевірки реальної роботи спроектованого пристрою. При успішному завершенні фізичного моделювання пристрій готовий для установки в систему.

Застосування САПР для розробки пристроїв на ПЛІС вимагає ефективних, наочних, керованих і контрольованих засобів опису проекту. На сучасному етапі найбільш поширеними універсальними способами опису проектів є **графічний** і **текстовий**.

Рисунок 1.13 – Алгоритм автоматизованого проектування ПЛІС

Графічний спосіб заснований на поданні схеми проекту в базисі допустимих для обраної САПР бібліотечних елементів, наприклад, у базисі елементів стандартної серії ТТЛ (Ш). Основні переваги графічного способу – традиційність і наочність, пов'язані зі звичністю розробників до сприйняття зображень схем. Однак у ході проектування пристроїв на ПЛІС, складність яких оцінюється десятками і сотнями тисяч еквівалентних вентилів, різко втрачається наочність проекту навіть під час коректного застосування системного підходу.

Тому в останнє десятиріччя набули істотного розвитку кошти текстового опису цифрових пристроїв на ПЛІС, реалізовані у вигляді мов опису апаратури (Hardware Description Languages – HDL).

Сучасні мови опису апаратури допускають опис проектованого пристрою як з точки зору його поведінки (виконуваних функцій), так і з точки зору його структури. Ці можливості дозволяють подавати проект у формі текстового опису алгоритмів функціонування окремих компонентів пристрою в поєднанні з описом міжкомпонентних з'єднань.

Перевагами **текстового способу** опису проекту за допомогою мов опису апаратури є компактність, автоматизація більшості перетворень, можливість перенесення проекту з однієї апаратної платформи на іншу, простота документування.

1.10 Можливості мов опису апаратури HDL

Сьогодні засоби **HDL** розділяються на два основні різновиди – мови низького і високого рівнів.

Мови опису апаратури низького рівня за своїми командами безпосередньо орієнтовані на роботу з апаратними засобами і мають потенційні можливості для створення проектів з оптимізованими параметрами. Дані HDL, як правило, жорстко прив'язані до певної апаратури. За допомогою мов низького рівня, що враховують спеціалізовані особливості архітектури ПЛІС, істотно полегшується створення проектів з найкращими часовими характеристиками. Прикладами таких мов можуть служити AHDL (Altera HDL) і ABEL (Xilinx).

Мови опису апаратури високого рівня менш пов'язані з апаратними платформами і володіють більшою універсальністю. Найбільш поширеними мовами цієї групи є VHDL і Verilog. Ці мови, як і інші мови програмування високого рівня, дозволяють описати будь-який алгоритм у послідовній формі – за допомогою послідовності операторів присвоювання і прийняття рішень. Особливістю даних мов є наявність засобів відображення паралельно виконуваних апаратних дій, які подаються окремими паралельно виконуваними процесами із загальним ініціалізованим впливом. Найбільш поширеною мовою цього класу, специфікованою міжнародними стандартами, є мова опису апаратури VHDL.

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть принципи розробки цифрових пристроїв на напівзамовних і програмованих користувачем БІС.
2. Поясніть особливості архітектури програмованих логічних матриць (ПЛМ) і принципи проектування цифрових пристроїв на їхній основі.
3. Поясніть відмінність архітектури ПЛМ і ПМЛ.
4. Які сучасні тенденції розвитку архітектур ПЛМ і ПМЛ?
5. Поясніть принципи побудови пристроїв на БМК.

6. Чим визначається логічна складність пристроїв на БМК?

7. Поясніть переваги і недоліки різних структур БМК.

8. Поясніть принципи внутрішньої організації ПЛІС FPGA.

9. Поясніть структуру конфігураційного-логічного блоку FPGA.

10. Яку функцію виконує логічна таблиця LUT в FPGA.

11. Поясніть принцип конфігурації ПЛІС FPGA.

12. Сформулюйте сучасні концепції розвитку архітектур ПЛІС.

13. У чому полягає сутність системного підходу під час проектування цифрових пристроїв?

14. Які функціональні елементи мають утворювати систему на кристалі?

15. Перерахуйте основні етапи системного проектування цифрових пристроїв.

16. Як змінюється адекватність формального опису цифрового пристрою при переході від функціональної до структурної і до фізичної моделей.

17. У чому полягають особливості автоматизованого проектування цифрових пристроїв на ПЛІС?

18. Сформулюйте основні переваги та недоліки способів схемного і текстового уявлень цифрових пристроїв на ПЛІС.

19. Сформулюйте основні особливості мов опису апаратури високого і низького рівнів.

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.

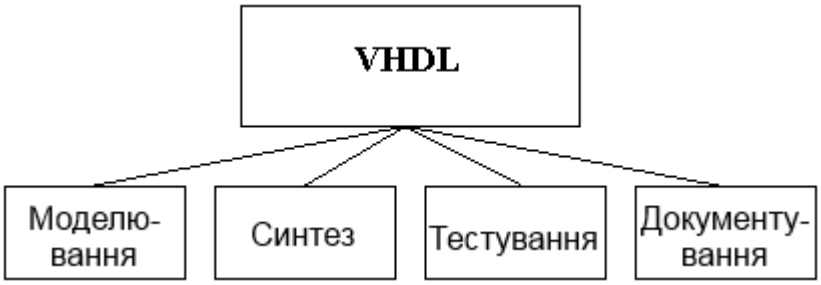


Рисунок 2.1 – Основні можливості мови VHDL

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 *К1* і *К2*, виконують логічну функцію *&* ("І"), і одного двоходового компонента *К3*, 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.

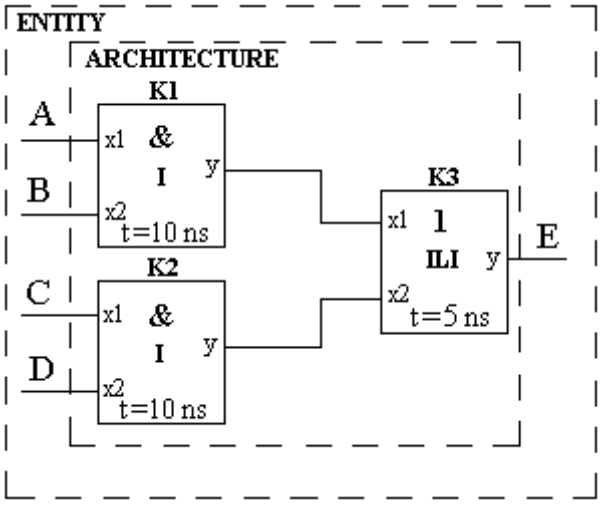


Рисунок 2.2 – Приклад найпростішої комбінаційної схеми

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.

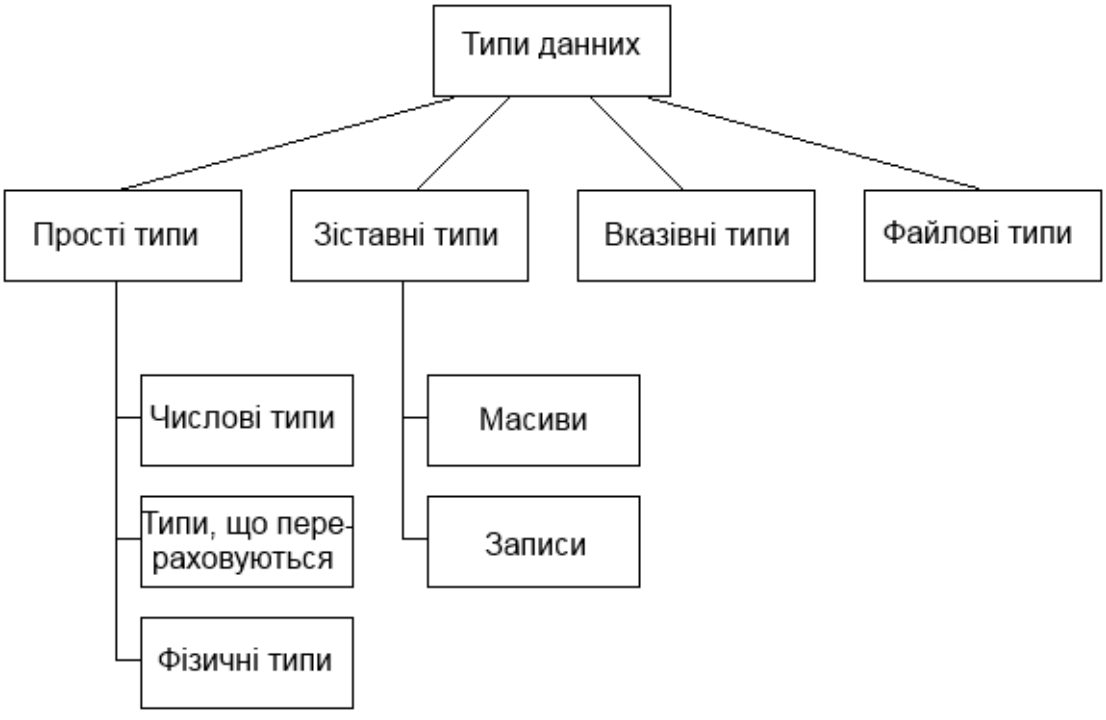


Рисунок 2.3 – Класифікація типів даних мови VHDL

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.

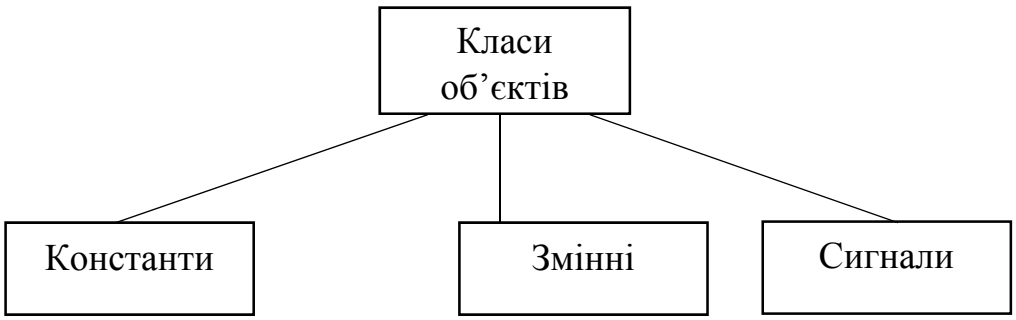


Рисунок 2.4 – Класи об'єктів мови VHDL

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**).

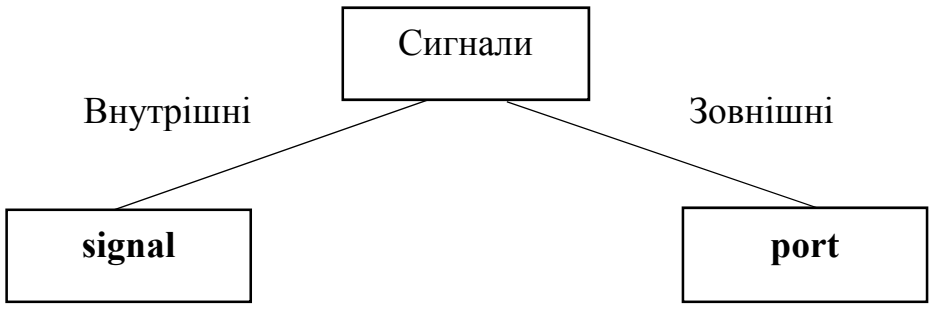


Рисунок 2.5 – Класифікація сигналів у мові VHDL

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.

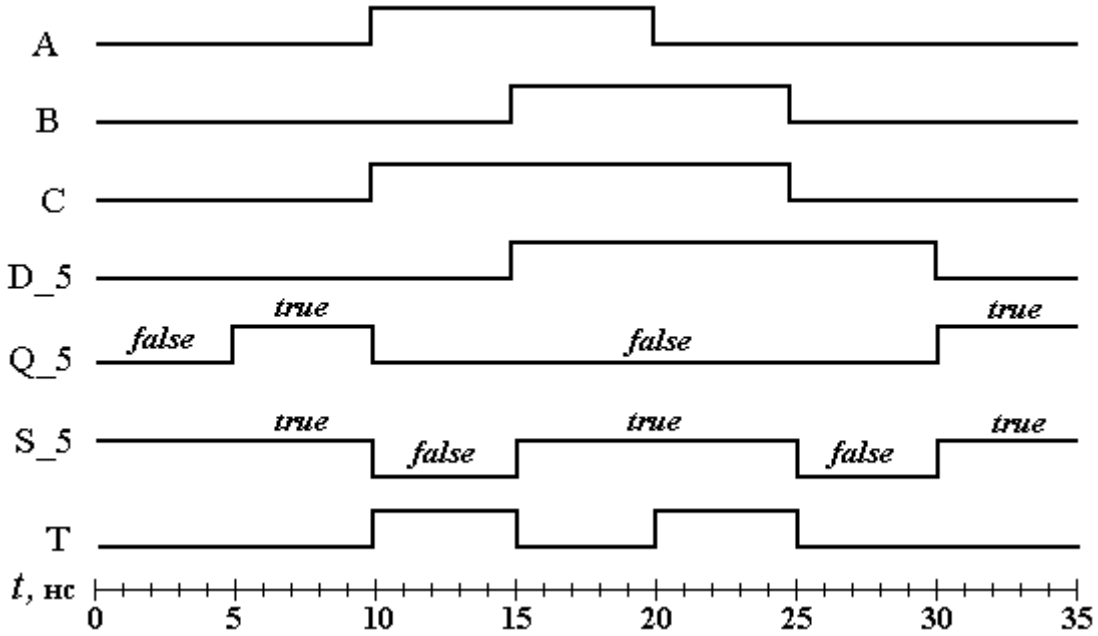


Рисунок 2.6 – Ілюстрація значень атрибутів сигналу C: D_5, Q_5, S_5, T

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).

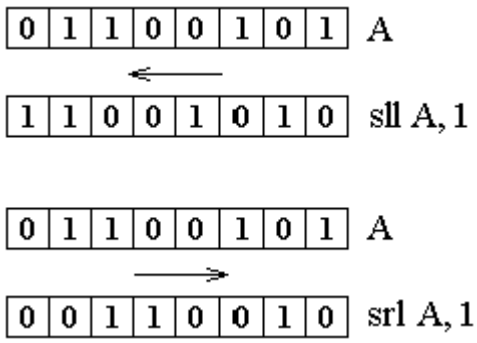


Рисунок 2.7 – Ілюстрація роботи операторів логічного зсуву над вектором A: **sll** (вліво) **srl** (вправо)

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).

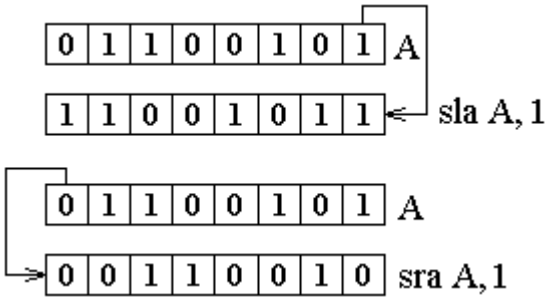


Рисунок 2.8 – Ілюстрація роботи операторів арифметичного зсуву над вектором A: **sla** (вліво) **sra** (вправо)

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).

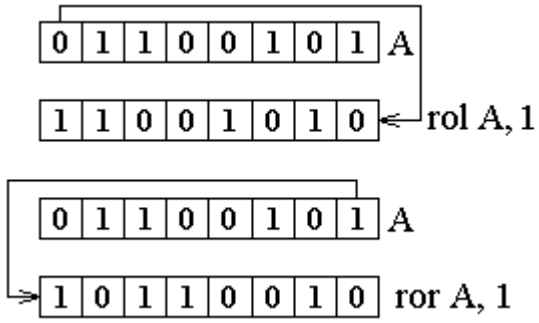


Рисунок 2.9 – Ілюстрація роботи операторів циклічного зсуву над вектором A: **sll** (вліво) **srl** (вправо)

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*.

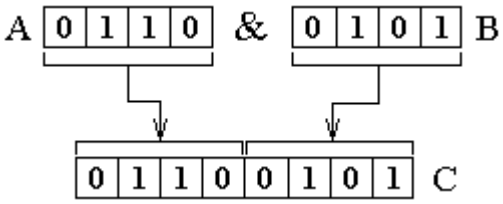


Рисунок 2.9 – Ілюстрація роботи оператора конкатенації **&** над векторами A і B

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

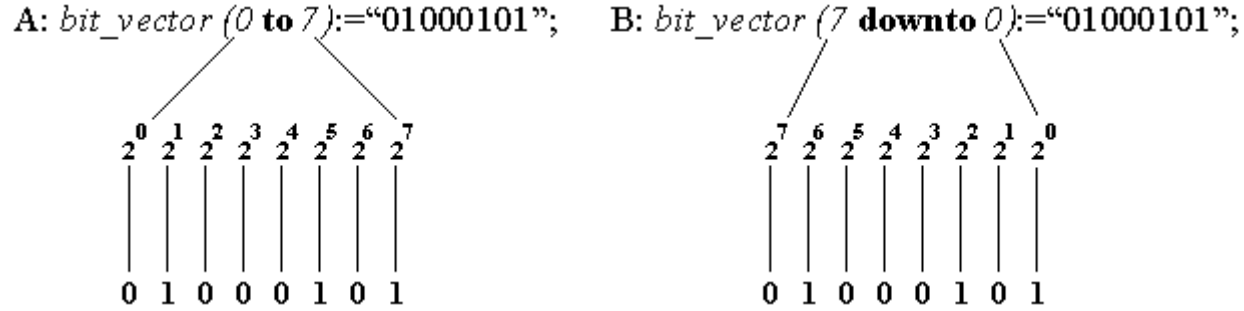
Для опису шинних стуктур в VHDL широко використовуються векторні типи даних bit_vector і std_logic_vector. Останній є найбільш універсальним (див. підрозділ 2.6), а тип bit_vector використовується для опису простих функціональних моделей. Надалі загальні питання роботи з бітовими векторами розглядатимуться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

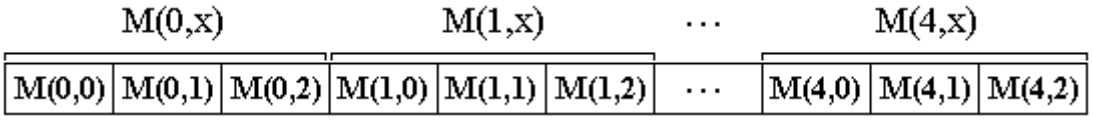


Рисунок 2.11 – Ілюстрація розподілу пам'яті під елементи масиву M (0..4, 0..2).

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.

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).

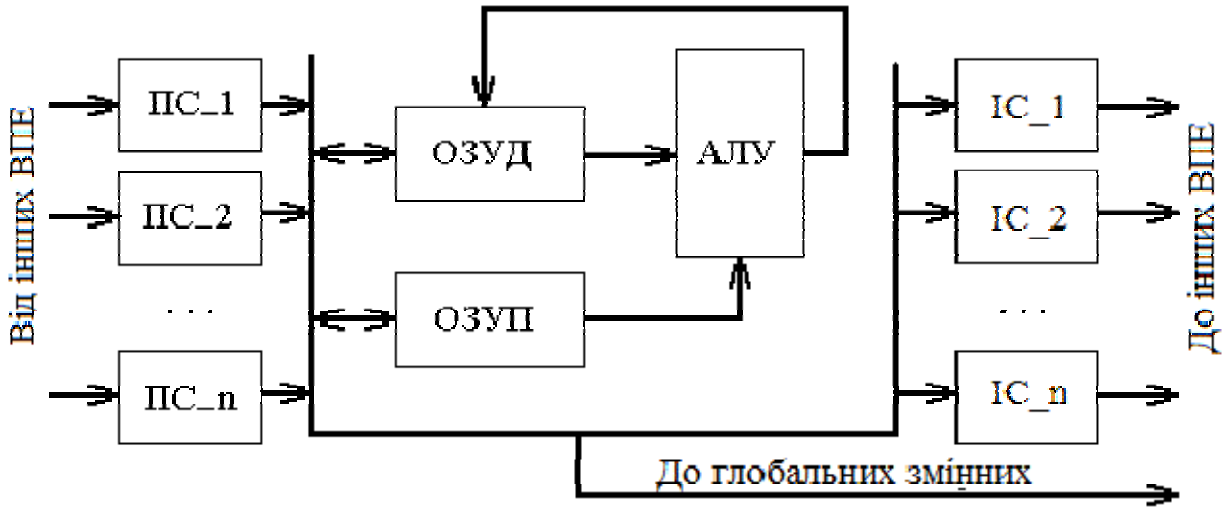


Рисунок 3.1 – Структурна схема віртуального процесорного елемента ВПЕ

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.

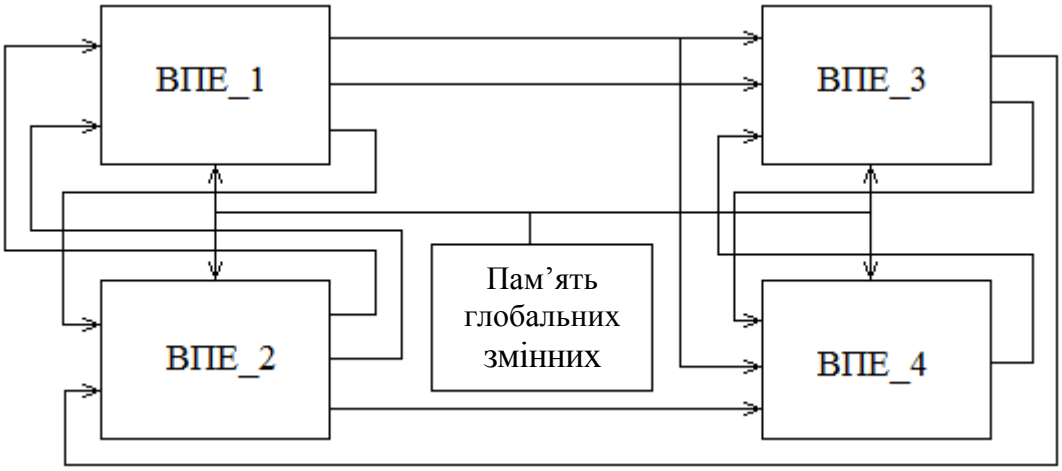


Рисунок 3.2 – Ілюстрація взаємодії декількох ВПЕ

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.

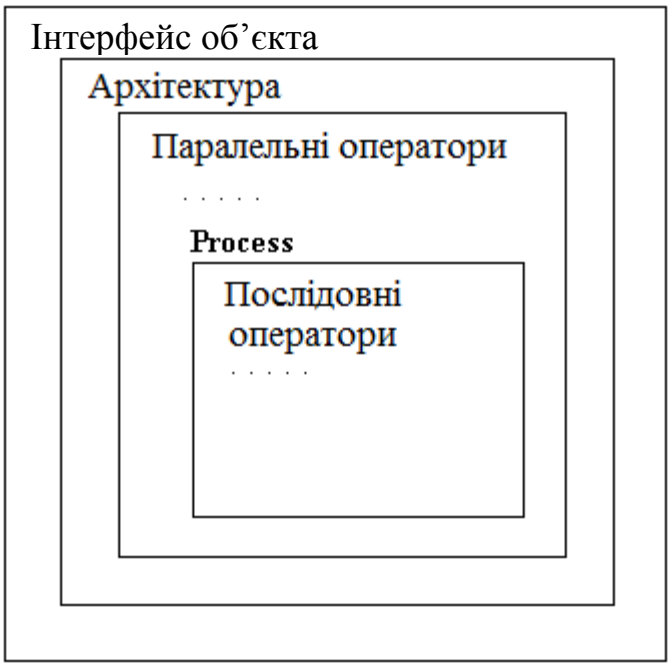


Рисунок 3.4 – Структура опису об'єкта моделювання

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').

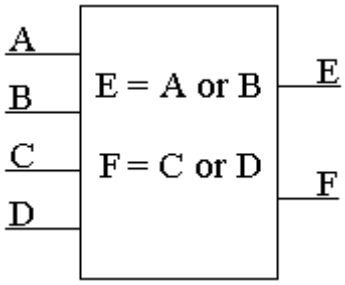


Рисунок 3.5 – Умовне позначення логічного пристрою

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.

Декларацийна 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.

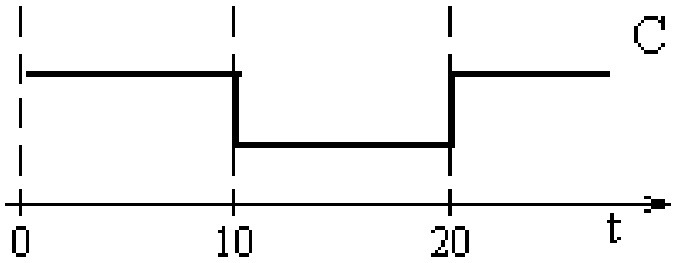


Рисунок 3.6 – Часова діаграма сигналу C

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 $t_0 + 3\Delta$.

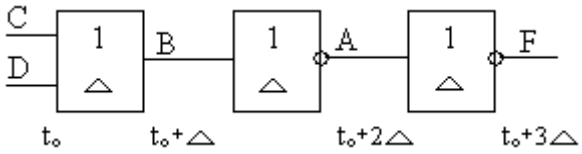


Рисунок 3.7 – Синтезована логічна схема, подана операторами паралельного призначення сигналу без специфікації затримки

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.

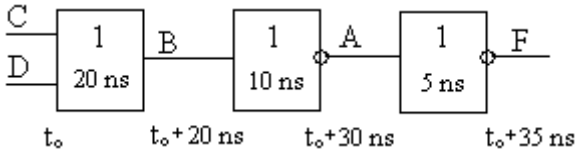


Рисунок 3.8 – Синтезована логічна схема, подана операторами паралельного призначення сигналу з явною вказівкою параметрів затримок

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.

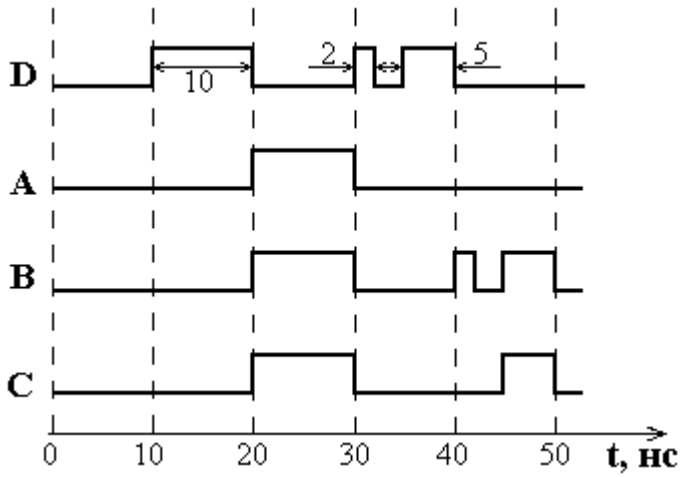


Рисунок 3.9 – Часові діаграми для сигналів A, B і C, що моделюють інерційну, транспортну і режекційну затримки сигналу D відповідно

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-НЕ»:

```
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.

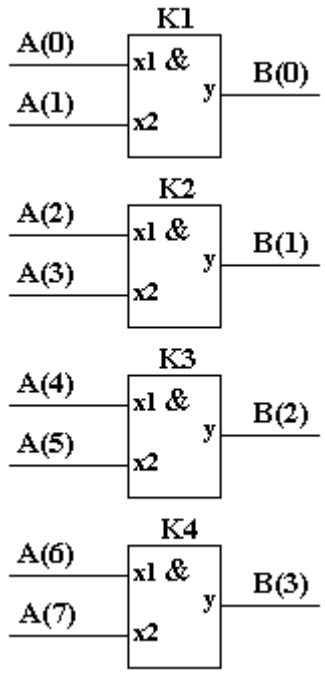
3. Для оператора генерації необхідна мітка.

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.

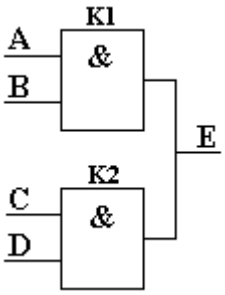


Рисунок 3.11 – Приклад схеми, що реалізує операцію монтажної логіки

```
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;
```

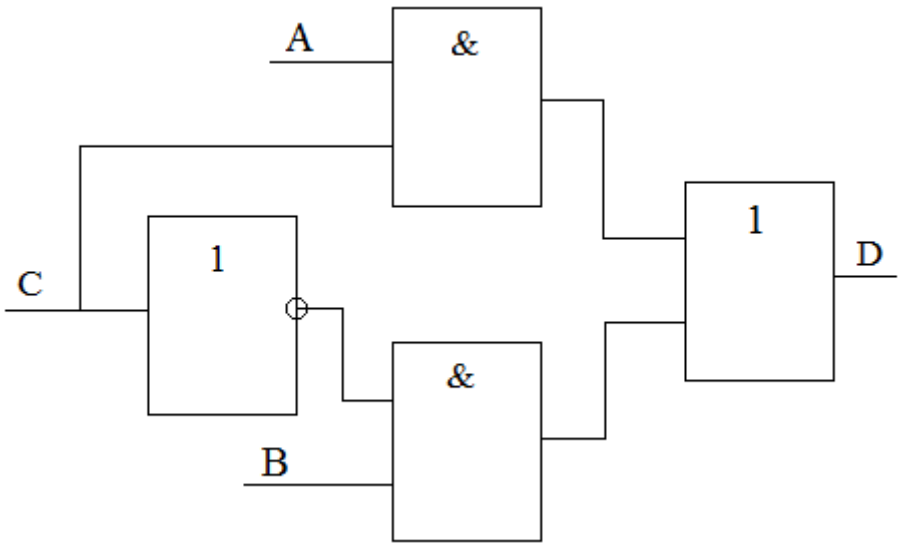


Рисунок 3.12 – Синтезований найпростійший мультиплексор

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.

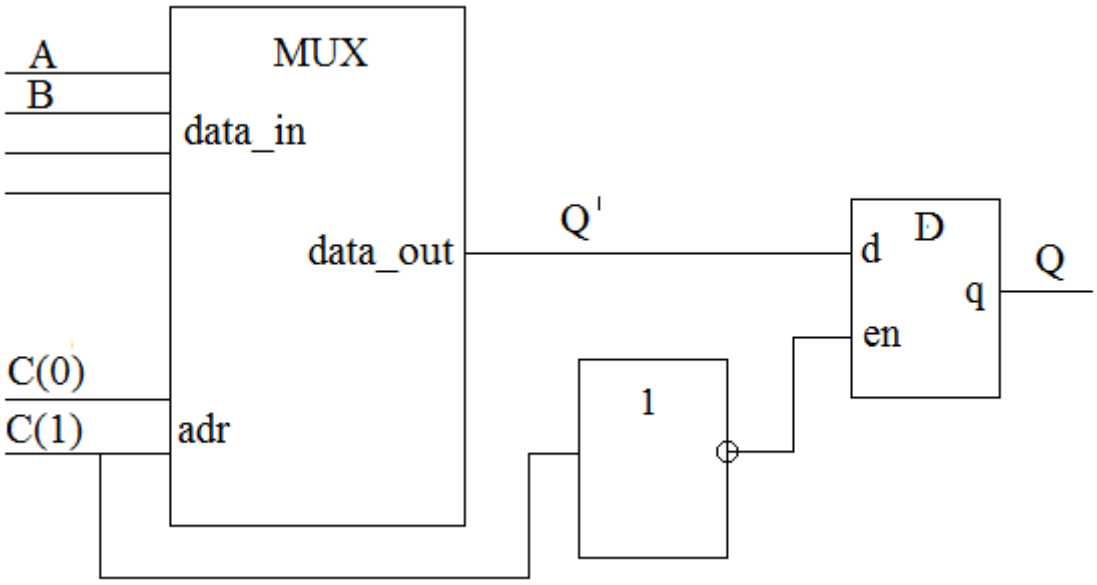


Рисунок 3.13 – Синтезований мультиплексор з паразитним елементом пам'яті

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

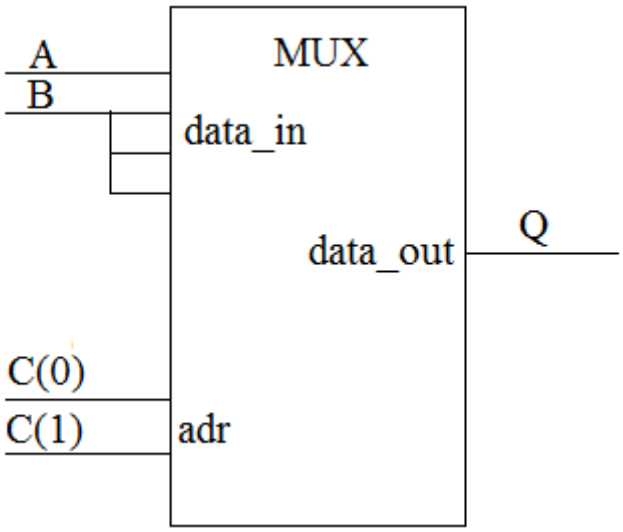


Рисунок 3.14 – Синтезований мультиплексор

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 параметр 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.

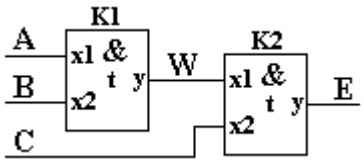


Рисунок 3.15 – Приклад простішої комбінаційної схеми

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.

4 ЛАБОРАТОРНИЙ ПРАКТИКУМ

Загальні положення. На сучасному етапі розвиток напівпровідникових технологій дозволив принципово змінити підхід до проектування цифрових пристроїв, за якоговсе частіше знаходять застосування програмувальні логічні інтегральні схеми (ПЛІС). У цей час найбільш широко використовуються ПЛІС, що програмуються користувачем безпосередньо (без виконання додаткових виробничих циклів) за допомогою спеціальних апаратно-програмних засобів. Це істотно підвищує ефективність розробки нестандартних пристроїв на ПЛІС. Такі ПЛІС у закордонній літературі одержали назву FPGA - Field Programmable Gate Array - програмовані полем вентильні матриці. FPGA є перспективним засобом реалізації цифрових пристроїв, важливою перевагою яких є простота конфігурування мікросхеми. При увімкненні живлення сигнали конфігурації (прошивання), що зберігаються в ПЗП, який розташований поруч із ПЛІС, переписуються в спеціальний зсувний регістр ПЛІС, до виходів якого підключені затвори всіх «програмуючих» транзисторів (див. рис. 4.1).

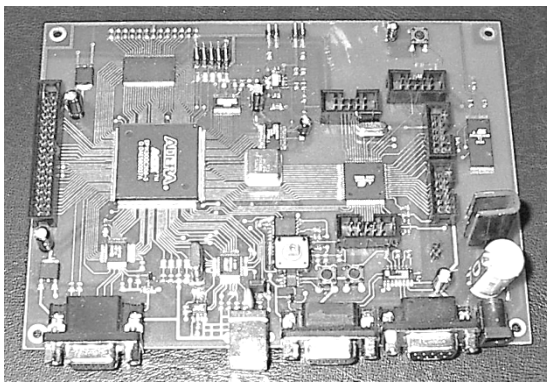


Рисунок 4.1 – Зовнішній вигляд друкованої плати лабораторного макета МЛ-2

Перевагами мікросхем FPGA порівняно із твердою логікою є:

- висока швидкодія;
- високий ступінь інтеграції, що дозволяє розмістити цифровий пристрій в одному кристалі й тим самим знизити час і витрати на трасування й виробництво друкованих плат;
- можливість перепрограмування безпосередньо в системі;
- скорочення часу розробки й виробництва цифрової частини пристрою;
- зручність і простота в налагодженні роботи пристрою (зокрема моделювання й аналіз роботи схеми за допомогою програмного забезпечення, що постачається фірмою-виготовлювачем ПЛІС);
- порівняно низька вартість (у перерахуванні на один логічний вентиль).

Сучасні ПЛІС надають розроблювачеві багато цікавих архітектурних і функціональних особливостей, що роблять можливим побудову на одному кристалі цифрових пристроїв різного призначення й складності.

Найбільш ефективно можливості мікросхем FPGA проявляються під час використання бібліотек компонентів, що постачаються розроблювачем ПЛІС і оптимізованих по затримках і займаних ресурсах з урахуванням топологічних властивостей кристала. Мікросхеми FPGA дозволяють проводити необмежену кількість циклів перепрограмування, мають мале енергоспоживання й досить високу надійність.

Основними виробниками ПЛІС на базі технології FPGA є фірми ALTERA і XILINX.

У циклі лабораторних робіт використовуватиметься лабораторний макет, заснований на мікросхемі FPGA ACEX 1K EP1K50QC208 і програмна система розробки цифрових пристроїв на ПЛІС MAX+PLUS II.

Сімейство ПЛІС ACEX 1K є на даний момент одним із найпопулярніших і є елементною базою для реалізації алгоритмів ЦОС, побудови складних пристроїв обробки даних і інтерфейсів сполучення. Це пояснюється тим, що, завдяки великій логічній місткості, зручній архітектурі, що включає убудовані блоки пам'яті, достатньо високій надійності й вдалому співвідношенню ціна - логічна місткість. ПЛІС FPGA ACEX 1K EP1K50QC208 містить 147 виводів, логічна місткість ПЛІС становить - 50000 еквівалентних вентилів, також містить 199000 системних вентилів і 40960 біт двопортової пам'яті.

Опис лабораторного робочого місця. Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3. Зовнішній вигляд передньої та інтерфейсної панелей макета наведено на рис. 4.1 і 4.2 відповідно; нумерація виводів ПЛІС із позначенням підключених модулів наведена в додатку А. Для кожного проекту необхідно створювати окремі каталоги, що обумовлено тим, що в процесі розробки проекту системою MAX+plus II створюється й підтримується безліч файлів, що стосуються поточного проекту. Насамперед, це файл проекту («**Project File**»), назва якого визначає назву проекту в цілому. Цей файл містить основну логіку й ієрархію проекту, оброблювану компілятором. Крім того, створюється ряд допоміжних файлів, пов'язаних із проектом, але не є частиною логіки проекту. Більша частина допоміжних файлів створюється й автоматично записується в каталог проекту в процесі вводу й компіляції проекту. Це, насамперед файли призначень і конфігурації (.ACF), файли звітів (.RPT), файли даних для функціонального моделювання й часового аналізу (.SNF), файли даних для програмування (.POF) і ряд інших. ***Назви цих файлів завжди збігаються з назвою проекту.*** Деякі допоміжні файли створюються користувачем: наприклад, для виконання функціонального моделювання створюється файл (.SCF), що містить опис початкових і поточних станів вхідних сигналів (входів) і перелік виходів, для яких мають бути визначені вихідні сигнали. Тому перед початком роботи над новим проектом слід створити робочий каталог проекту, при цьому ім'я каталогу можна вибирати довільно (тобто ім'я каталога може не збігатися з ім'ям).

Рисунок 4.2 – Зовнішній вигляд передньої панелі лабораторного макета МЛ-2

Таблиця 4.1 – Призначення виводів ПЛІС

Призначення виводу	Номер виводу	Коментар
Світлодіоди		
D0	86	Світлодіод0
D1	87	Світлодіод1
D2	88	Світлодіод2
D3	89	Світлодіод3
D4	90	Світлодіод4

Продовження табл. 4.1

D5	92	Світлодіод5
D6	93	Світлодіод6
D7	94	Світлодіод7
Кнопки		
F1	71	Кнопка 0
F2	73	Кнопка 1
F3	74	Кнопка 2
F4	75	Кнопка 3
F5	79	Кнопка 4
F6	80	Кнопка 5
F7	83	Кнопка 6
F8	85	Кнопка 7
CLK	183	Вивід тактового сигналу

На передній панелі лабораторного макета МЛ-2 використовуваними в роботі із ПЛІС є верхній блок світлодіодної індикації (ряд світлодіодів D0-D7, позначені як FPGA-LED) і верхній ряд кнопок, позначених F1 - F8. Номери виводів ПЛІС, до яких підключаються блоки світлодіодів і кнопок, а також тактового сигналу, наведені в таблиці 4.1.

Інші пристрої передньої панелі лабораторного макета МЛ-2 призначені для проектування складних комбінованих пристроїв на мікроконтролерах і ПЛІС і не використовуються в даному циклі лабораторних робіт. На інтерфейсній панелі лабораторного макета МЛ-2 розташовані інтерфейси для підключення зовнішніх пристроїв, а також інтерфейси програматора, електроживлення й клавіша увімкнення електроживлення (див. рис. 4.3). Підключення лабораторного макета МЛ-2 до ПЕОМ здійснюється через рознімач DB-25 паралельного інтерфейсу Centronics.

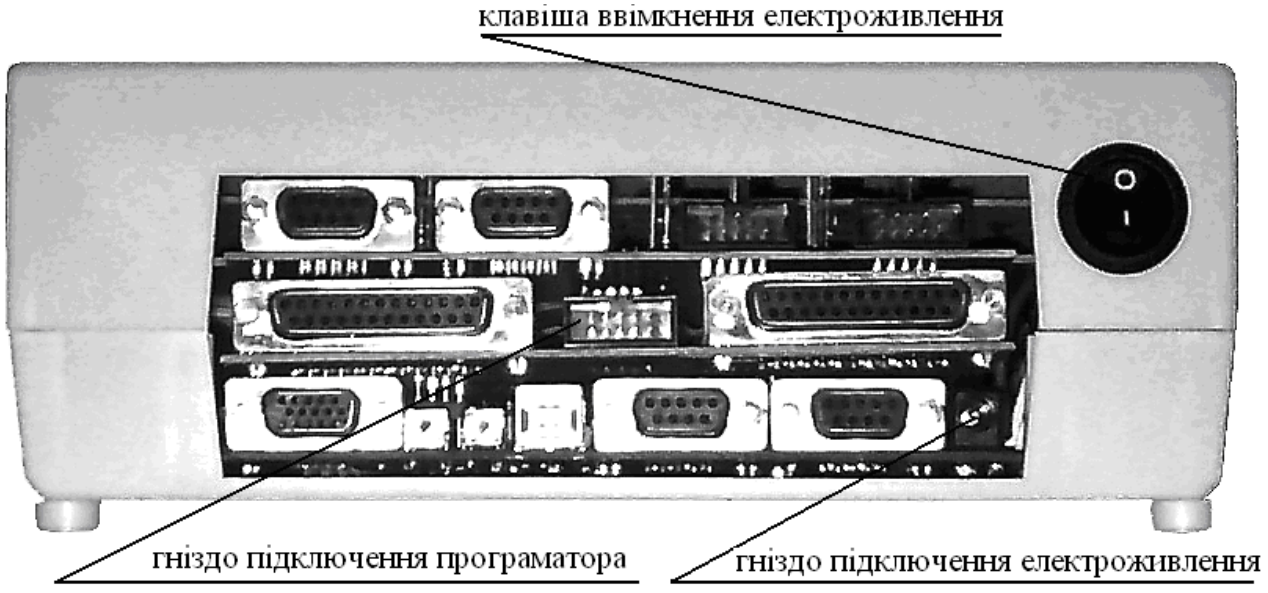


Рисунок 4.3 – Зовнішній вигляд інтерфейсної панелі лабораторного макета МЛ-2

Підключення світлодіодів до виводів ПЛІС лабораторного макета МЛ-2 здійснюється згідно з рис. 4.4, на якому показано підключення світлодіода D0 до 86-го виводу ПЛІС через струмообмежуючий резистор R1. На аноди світлодіодів (D0-D7) подається напруга + 5 V, катоди кожного світлодіода підключаються до відповідних виводів ПЛІС. У гілку кожного світлодіода послідовно підключається струмообмежуючий резистор номіналом 390 Ом. Світлодіоди D0 - D7 включаються при подачі рівня «логічного нуля» на відповідні виводи ПЛІС, які мають бути сконфігуровані на вивід (output).

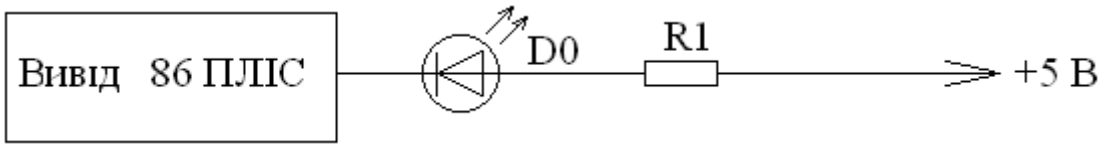


Рисунок 4.4 – Схема підключення світлодіода D0 до виводу ПЛІС

Підключення кнопок до виводів ПЛІС лабораторного макета МЛ-2 здійснюється згідно з рис. 4.5, на якому показано підключення кнопки F1 до 71-го виводу ПЛІС через струмообмежуючий резистор R1. При розімкнутих контактах кнопок на відповідному виводі ПЛІС перебуває рівень «логічної одиниці», при натисканні на кнопку на відповідному виводі встановлюється рівень «логічного нуля». Виводи ПЛІС, підключені до кнопок F1-F8, мають бути сконфігуровані на вхід (input). Розташування інтерфейсних рознімачів на задній панелі лабораторного макета МЛ-2 наведено на рисунку 4.6.

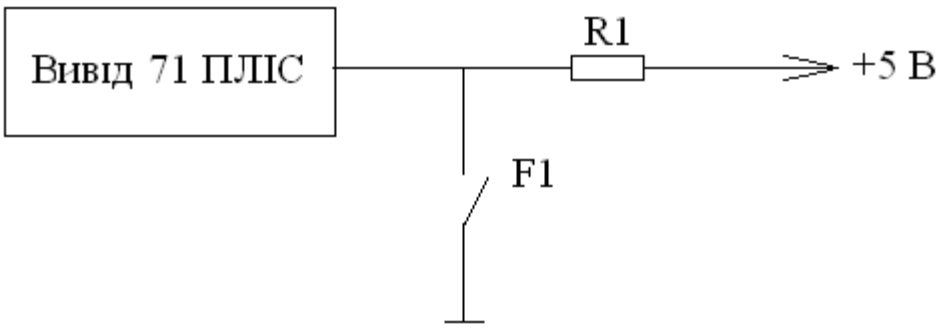


Рисунок 4.5 – Схема підключення кнопки F1 до виводу ПЛІС

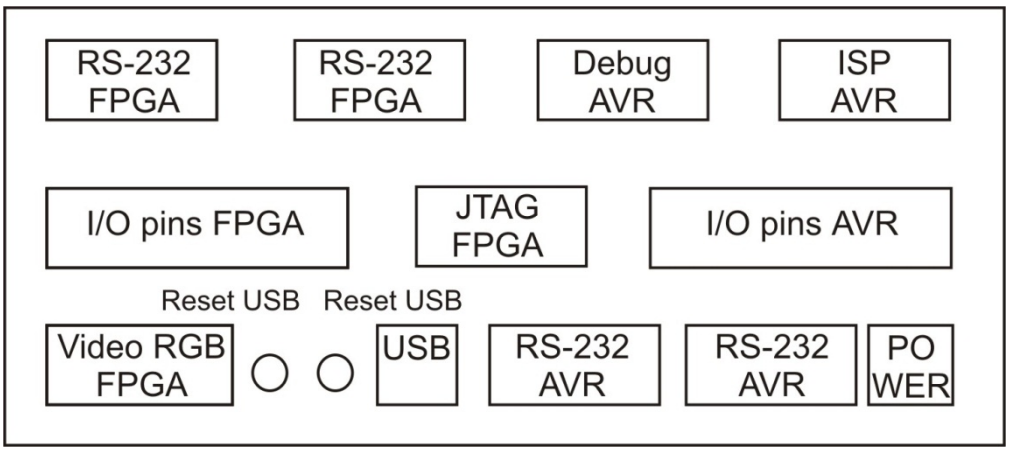


Рисунок 4.6 – Розташування інтерфейсних рознімачів на задній панелі лабораторного макета МЛ-2

У циклі лабораторних робіт використовується програмний пакет MAX+PLUS II, призначений для автоматизованого проектування цифрових пристроїв на ПЛІС.

Пакет MAX+PLUS II забезпечує виконання всіх етапів, необхідних для розробки й випуску готових виробів цифрових пристроїв на ПЛІС фірми Altera, що включають: створення проектів пристроїв; синтез структур і трасування внутрішніх зв'язків ПЛІС; підготовку даних для програмування або конфігурування ПЛІС (компіляцію); верифікацію проектів (функціональне моделювання й часовий аналіз); програмування або конфігурування ПЛІС.

Для реалізації всіх цих етапів до складу MAX+PLUS II входять наступні, пов'язані між собою програмні засоби.

Засоби для вводу проектів (редактори проектів):

Graphic Editor – графічний редактор, призначений для вводу проекту у вигляді схеми з'єднань символьних елементів, що витягуються з бібліотек пакета (стандартних або тих, що введені користувачем).

Waveform Editor – редактор часових діаграм (сигнальний редактор), що виконує подвійну функцію: на етапі вводу забезпечує введення логіки проекту у вигляді діаграм станів входів і виходів, а на етапі моделювання забезпечує введення діаграм тестових (еталонних) вхідних станів модулюючого пристрою й завдання переліку тестувальних виходів.

Text Editor – текстовий редактор, призначений для створення й редагування текстових файлів, що містять опис логіки проекту мовою опису пристроїв AHDL (Altera Hardware Description Language), або близьких до нього мов типу VHDL,Verilog.

Symbol Editor – символьний редактор, що дозволяє редагувати існуючі символи й створювати нові. Будь-який відкомпільований проект може бути згорнутим у символ, поміщений у бібліотеку й використаний як елемент у будь-якому іншому проекті.

Floorplan Editor – редактор зв'язків (порівневий планувальник), що на плані розташування основних логічних елементів дозволяє вручну розподіляти виводи ПЛІС (закріплювати виводи за конкретними вхідними й вихідними сигналами) і перерозподіляти деякі внутрішні ресурси ПЛІС.

Засіб для синтезу структури, трасування зв'язків, перевірки коректності проекту й локалізації помилок, формування файлів програмування або конфігурування ПЛІС, що входять у пакет компілятора (MAX+PLUS II Compiler):

Netlist Extractor – засіб, що забезпечує витяг списку з'єднань із вихідного файла подання проекту, створеного під час введення проекту.

Database Builder – засіб, призначений для побудови бази даних проекту.

Logic Synthesizer – засіб, що забезпечує перевірку коректності проекту за формальними правилами й синтез оптимальної структури проекту.

Partitioner – засіб, що забезпечує розбивку проекту на частини в тих випадках, коли ресурсів одного кристала (мікросхеми) недостатньо для реалізації проекту.

Fitter – трасувальник внутрішніх зв'язків, що забезпечує реалізацію синтезованої структури.

SNF Extractor – засіб, що забезпечує відображення параметрів проекту, необхідних для функціонального моделювання й часового аналізу.

Засіб для верифікації проектів включають:

Simulator – засіб, що разом з редактором тимчасових діаграм призначено для функціонального моделювання проекту з метою перевірки правильності логіки його функціонування.

Timing Analyzer – засіб, що забезпечує розрахунок тимчасових затримок від кожного входу до кожного, логічно пов'язаного з ним виходу.

Для програмування або конфігурування ПЛІС використовується MAX+PLUS II Programmer. Програмування й перепрограмування мікросхем, що мають убудовану систему програмування (ISP), може здійснюватися безпосередньо в складі кінцевого виробу через спеціальний кабель, що підключається або до LPT-Порту (Byte Blaster), або до COM-Порту (Bit Blaster) комп'ютера й технологічного 10 контактного з'єднувача інтерфейсу JTAG, установлюваного на платі виробу.

До складу САПР також входять три сервісних додатки:

Design Doctor – засіб, призначений для перевірки коректності проекту з використанням емпіричних правил.

Message Processor – процесор повідомлень, що забезпечує обробку, вивід на зображення й локалізацію (вказівка місця в проекті, до якого воно належить) повідомлень трьох типів: повідомлень про помилки (Error), попереджень (Warning) й інформаційних повідомлень (Info). Причину висновку того або іншого повідомлення можна з'ясувати через опцію Help on Message процесора повідомлень. За наявності повідомлень про помилки, компіляція проекту неможлива до їхнього повного усунення. За наявності попереджень компіляція успішно завершується, однак, наявність попередження свідчить про виявлення

проблеми, що може призвести до неправильної роботи пристрою. Інформаційні повідомлення потрібно тільки приймати до відома.

Hierarchy Display – забезпечує огляд ієрархічної структури проекту, що може складатися з безлічі складених у різних редакторах і згорнутих у символи проектів більш низьких рівнів, причому число рівнів не обмежується. Основний проект (проект самого верхнього рівня) має бути створений у графічному редакторі (якщо проект має тільки один рівень ієрархії, то він може створюватися в будь-якому редакторі).

Порядок виконання лабораторних робіт. На початку першого заняття всі студенти повинні пройти інструктаж з техніки безпеки й розписатися в журналі обліку виконання лабораторних робіт. Студенти, які не ознайомилися із правилами техніки безпеки, до виконання лабораторних робіт не допускаються.

Кожній лабораторній роботі (ЛР) може передувати самостійна підготовка студентів, у процесі якої потрібно вивчити методичні вказівки до лабораторної роботи, конспект лекцій і рекомендовані літературні джерела. Перед початком ЛР викладач перевіряє підготовленість студентів до виконання конкретної лабораторної роботи, що передбачає знання теоретичного матеріалу за темою роботи, мети й порядку виконання роботи, наявність попередніх варіантів виконання індивідуальних завдань.

Результати виконання ЛР відображаються у звіті, що може містити: назву лабораторної роботи, мету роботи, лістинги програм виконаних завдань і висновки. До початку наступної ЛР студент повинен подати викладачеві повністю оформлений звіт з попередньої роботи й захистити його. Залік з ЛР студент одержує після співбесіди з викладачем за темою виконаної роботи.

Лабораторна робота 1

ВИВЧЕННЯ СЕРЕДОВИЩА АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ MAX+PLUS II І РЕАЛІЗАЦІЯ НАЙПРОСТІШИХ ЛОГІЧНИХ ЕЛЕМЕНТІВ НА ПЛІС ЗА ГРАФІЧНИМ ОПИСАННЯМ

1.1 Мета роботи

Вивчення графічного інтерфейсу середовища автоматизованого проектування MAX+PLUS II і реалізації на ПЛІС функцій найпростіших логічних елементів з використанням графічного опису.

1.2 Опис лабораторної установки

Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3. Зовнішній вигляд передньої та інтерфейсної панелей наведені на рис. 4.1 і 4.2, нумерація виводів ПЛІС із позначенням підключених модулів наведені в таблиці 4.1.

1.3 Вказівки до організації самостійної роботи

Під час підготовки до лабораторної роботи необхідно проробити теоретичний матеріал за літературою [1–8] і за конспектом лекцій, ознайомитися зі структурою й принципами функціонування ПЛІС типу FPGA, можливостями графічного опису схем у середовищі MAX+PLUS II.

1.4 Порядок виконання роботи

Запустити пакет MAX+PLUS II, відкриється головне вікно MAX+PLUS II (див. рис. 1.1). У найвищому рядку відображається ім'я останнього проекту, з яким велася робота. Два наступні рядки є звичайними для Windows-програм: рядок основного меню й панель інструментів, у лівій частині якої розташовані звичайні інструменти Windows (New, Open, Save, Print, Cut, Copy, Paste, Undo), а в правій – специфічні інструменти пакета, за допомогою яких здійснюється запуск основних додатків пакета. Створюватимемо проект у графічному редакторі.

Розглянемо основні етапи синтезу найпростіших цифрових схем на ПЛІС за їхнім графічним описом на прикладі двовходового логічного елемента «І». Для цього після запуску системи MAX+PLUS II необхідно створити новий файл (див. рис. 1.2). У діалоговому вікні, що відкрилося **(File/**New), вибираємо **пункт Graphic Editor** file (див. рис. 1.3) і натискаємо кнопку ОК, при цьому автоматично відкривається вікно графічного редактора (див. рис. 1.4). Це вікно

має ряд додаткових пунктів основного меню й панель інструментів редакторів, розташовану вертикально з лівої сторони вікна. Збережемо новий файл проекту (через меню **File/Save**) під ім'ям pro_i (розширення присвоюється автоматично).

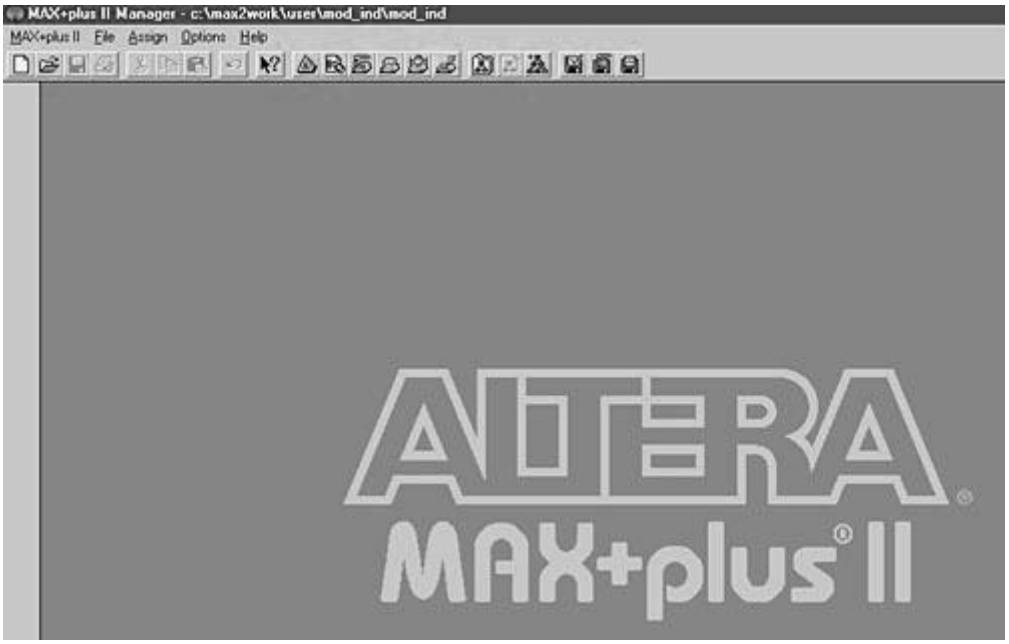


Рисунок 1.1 – Головне вікно MAX+PLUS II

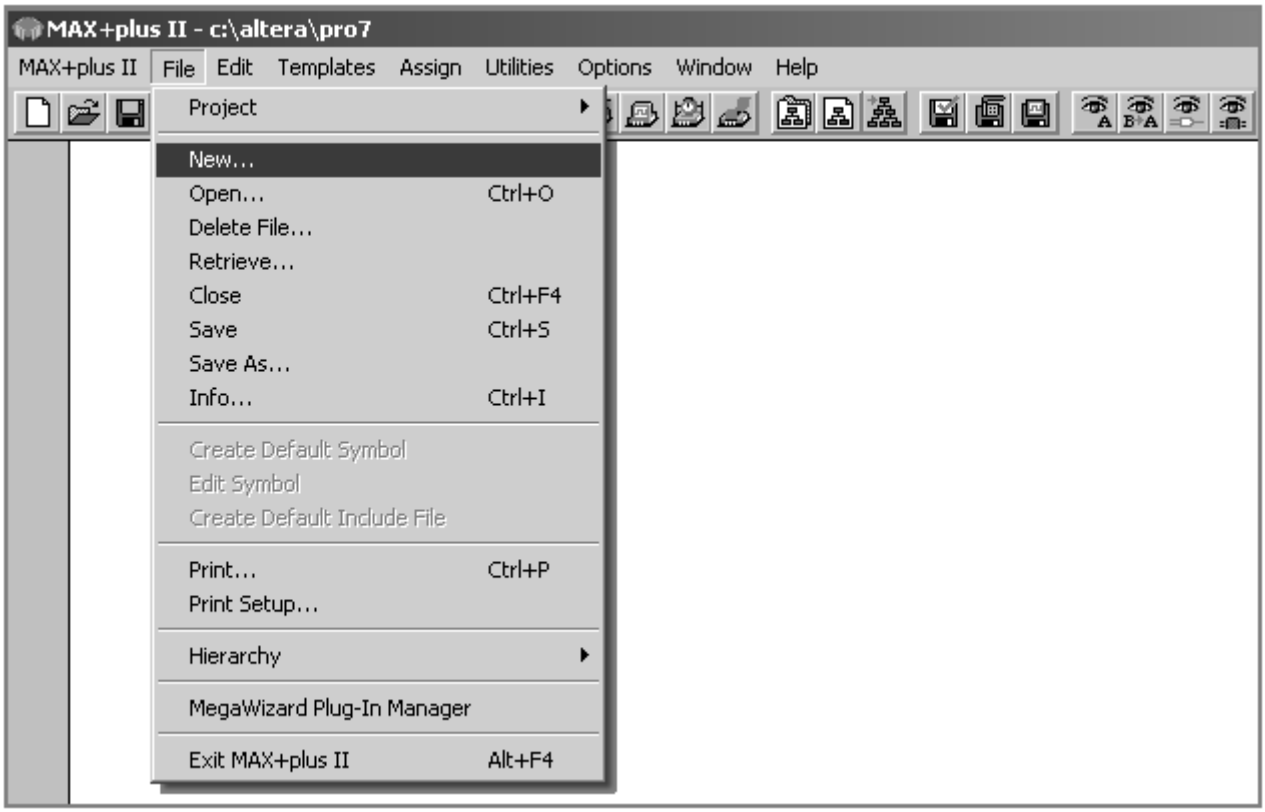


Рисунок 1.2 – Меню створення нового файла

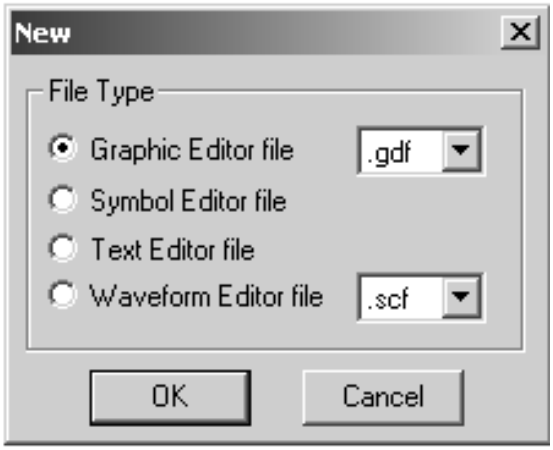


Рисунок 1.3 – Меню вибору графічного файла

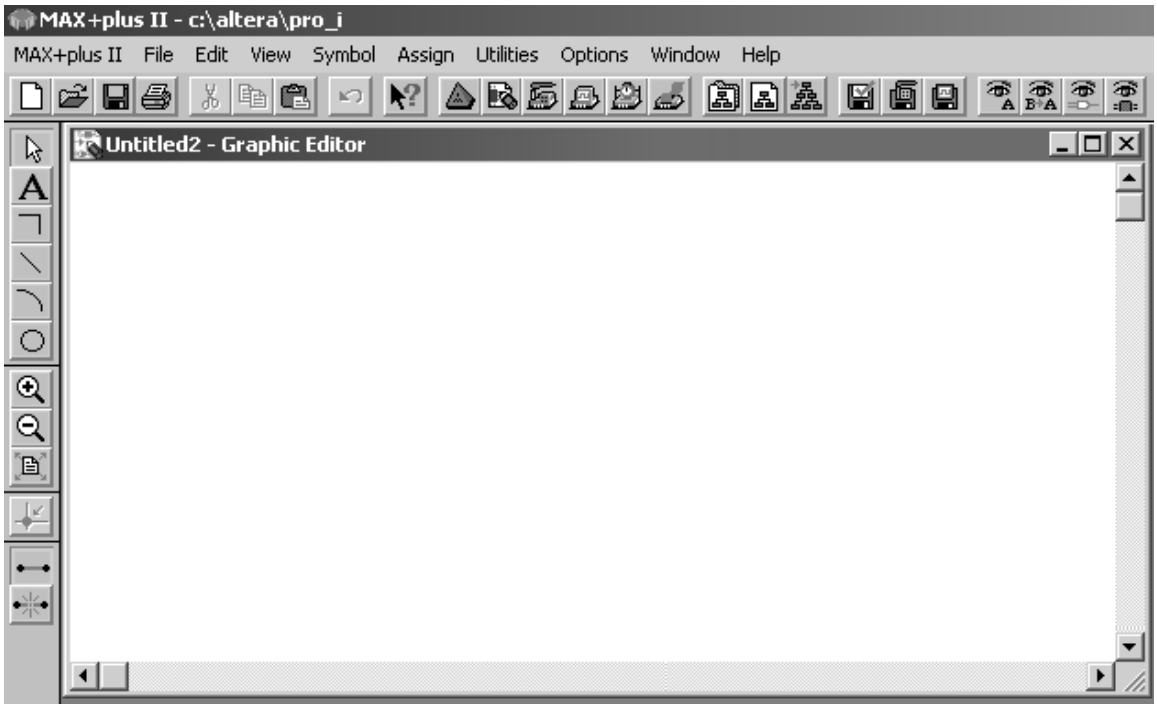


Рисунок 1.4 – Вікно графічного редактора

Ім'я файла проекту обов'язково прив'язується до ім'я проекту. Прив'язка ім'я файла до ім'я проекту здійснюється через підменю Project меню File головного меню робочого вікна вибором пункту: **Set Project to Current File** (див. рис. 1.5).

Подвійним натисканням лівої клавіші мишки на порожньому робочому полі графічного редактора викликається підменю **Enter Symbol** (рис. 1.6) вибору робочої бібліотеки й, безпосередньо, схемотехнічних вузлів, до складу яких входить великий набір основних логічних елементів, тригерів, елементів входу й виходу (input, output, bidir), а також допоміжні елементи: GND (логічний нуль), VCC (логічна одиниця) і т.д.

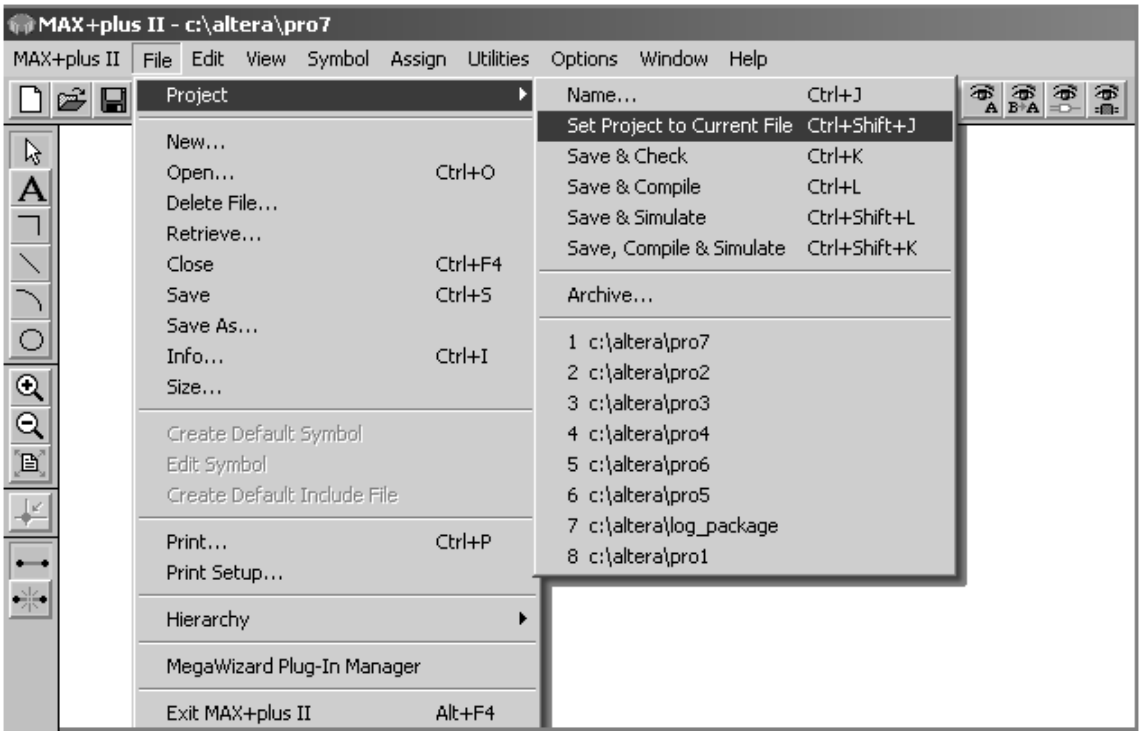


Рисунок 1.5 – Пункт підменю прив'язки ім'я файла до ім'я проекту

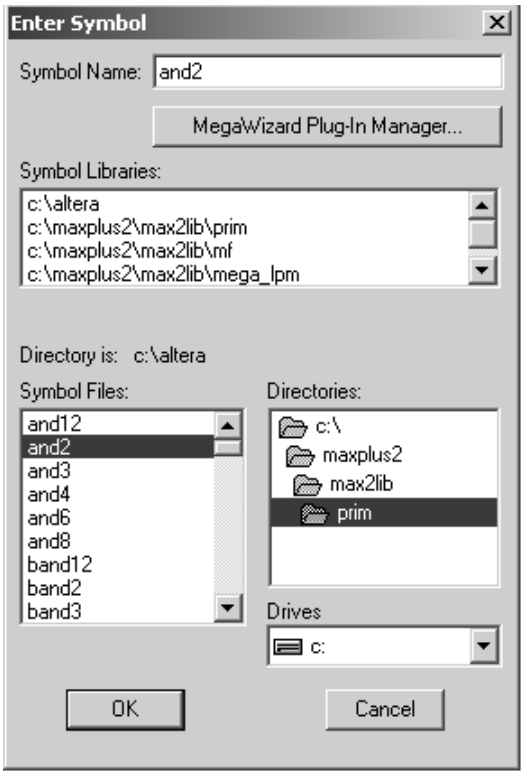


Рисунок 1.6 – Діалогове вікно Enter Symbol

Вибір необхідного елемента здійснюється шляхом вказівки необхідної бібліотеки у вікні **Symbol Libraries** (використовувана бібліотека prim) діалогового вікна **Enter Symbol**, після чого у вікні **Symbol Files** вказується необхідний елемент (and2).

Обраний елемент розміщується на робочому полі графічного редактора й може бути переміщений в інше місце. Переміщення елементів може проходити без збереження або зі збереженням з'єднань, проведених між елементами, залежно від стану кнопок Rubberbanding Functions on/off, розташованих у нижній частині панелі інструментів редактора.

З'єднання виводів символів елементів можна виконати двома способами.

Перший спосіб припускає проведення сигнальних ліній. Для цього курсор мишки сполучають із виводом елемента, при цьому він автоматично перетворюється в інструмент рисування ортогональних ліній (перехрестя), потім натискають ліву кнопку мишки й, утримуючи її, проводять лінію. За один прийом можна провести два ортогональних відрізки лінії. Якщо цього недостатньо, то процедуру можна повторити, починаючи з кінця проведеної лінії або з виводу іншого елемента.

Другий спосіб полягає в присвоєнні однакових імен тим виводам, які мають бути з'єднані між собою. Іменований вивід елемента називається вузлом (Node). Для присвоєння ім'я вузлу необхідно спочатку провести коротку сигнальну лінію. Оскільки після закінчення рисування лінії вона залишається виділена (виділені елементи позначаються червоним кольором або червоною рамкою), то ім'я вузла можна вводити відразу. Якщо лінія не виділена (має чорний колір), то необхідно виділити її одинарним клацанням лівої клавіші мишки в тому місці, де має починатися ім'я вузла й набрати на клавіатурі необхідне ім'я.

Встановлені елементи, групи елементів (будь-яку виділену область вікна) можна вилучати, переміщати, повертати й розмножувати (копіювати) у межах вікна редактора або через буфер переносити в інші вікна за правилами, прийнятими у Windows.

Частина інструментів редактора, призначеного для створення пояснюючих написів, рисунків і таблиць, які не є частиною проекту й не обробляються надалі компілятором.

Далі необхідно розмістити на робочому полі графічного редактора два компоненти введення сигналу input і один компонент виводу output, після чого необхідно з'єднати вхідні компоненти input із входами логічного елемента and2 і компонент output з виходом and2. Потім необхідно позначити інтерфейсні сигнали (наприклад, PIN_NAME1, PIN_NAME2 і pinout3), вибравши режим редагування текстової інформації схемних компонентів одиночним клацанням лівої клавіші мишки за відповідним підписом.

Далі виконується вибір типу ПЛІС шляхом виклику з пункту меню **Assign** підпункту **Device**, у якому необхідно ввести позначення типу використовуваної в лабораторному макеті ПЛІС (Device Family – ACEX1K; EP1K50QC 208-3, (див. рис. 1.7). Після цього в пункті меню **Assign** (підпункт

Pin/Location/Chip) необхідно привласнити інтерфейсним сигналам номери виводів ПЛІС. Вхідні лінії відповідатимуть кнопкам, наприклад. F1 - 71, F2 - 73, а вихід схеми – світлодіоду D0 - 86 (див. рис. 1.8).

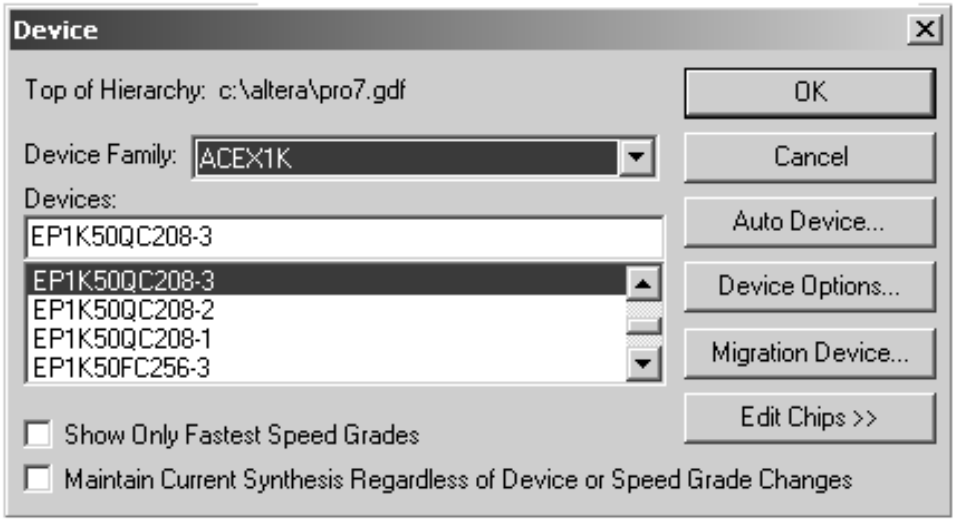


Рисунок 1.7 – Діалогове вікно вибору сімейства й типу ПЛІС

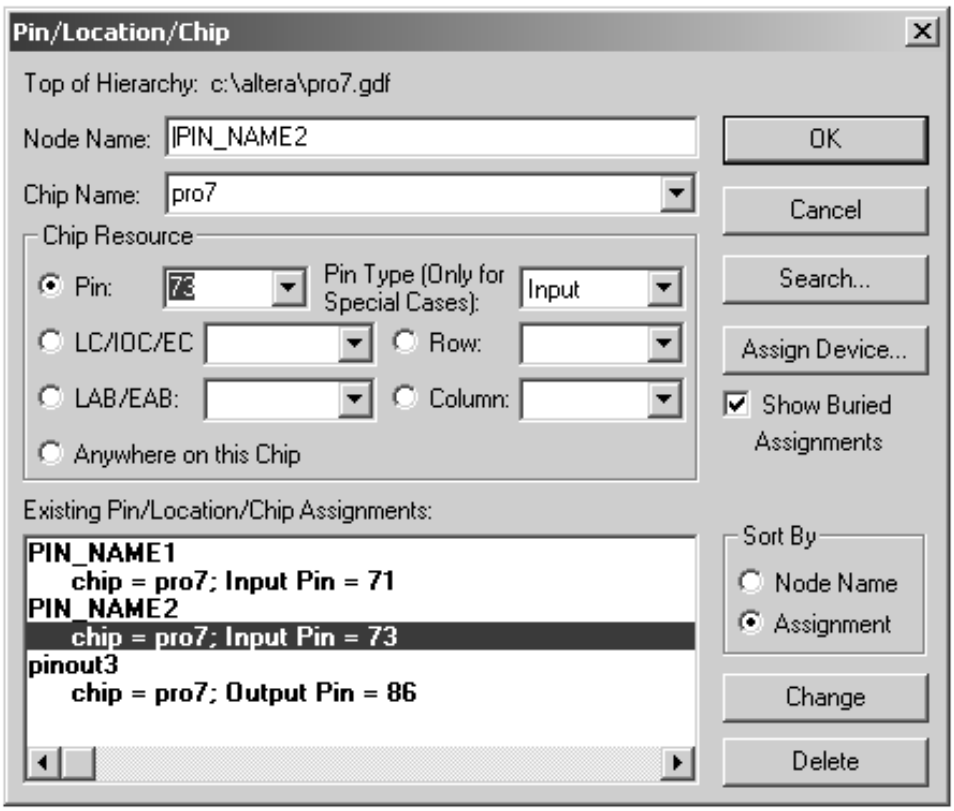


Рисунок 1.8 – Діалогове вікно відповідності виходів схеми виводів ПЛІС

Результуючий вигляд створеної схеми проекту наведений на рисунку 1.9.

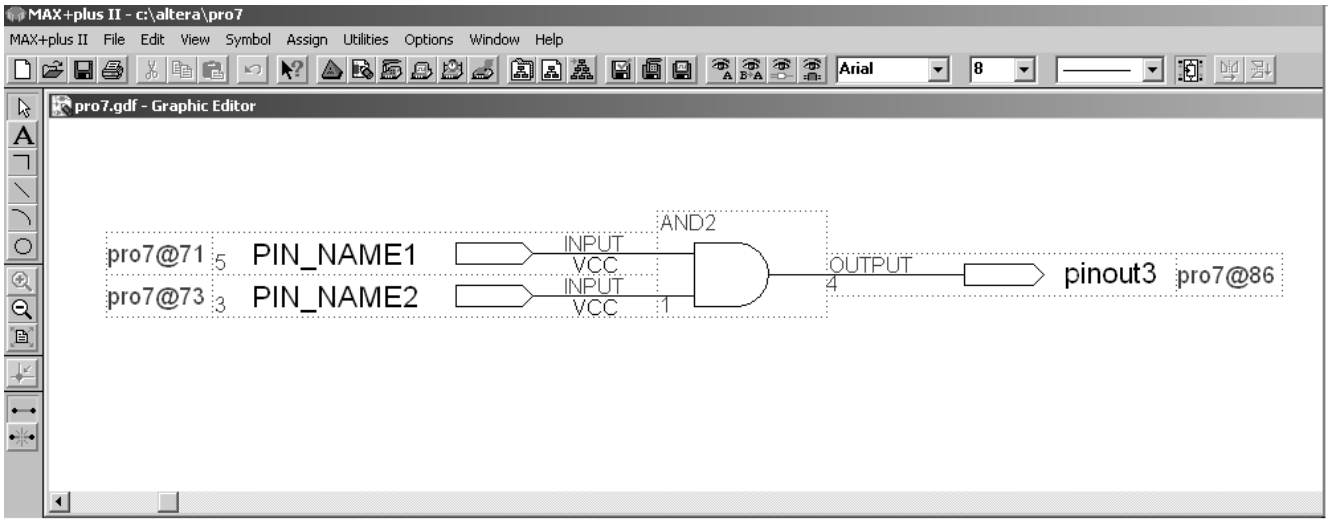


Рисунок 1.9 – Вікно створеного проекту для синтезу логічного елемента «И» за його графічним описом

Для виконання подальших дій під час роботи із проектом будуть потрібні піктограми графічного меню, відображені на панелі інструментів (див. рис. 1.10)

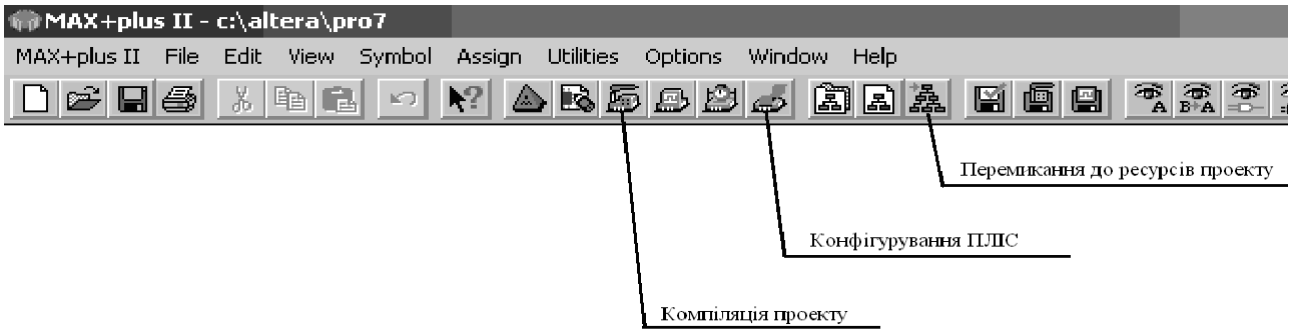


Рисунок 1.10 – Основні піктограми графічного меню MAX+plusII в ході компіляції й синтезу проекту

Наступний етап – компіляція й створення символу проекту для включення його у файл проекту верхнього рівня. Перед компіляцією можна виконати перевірку коректності введеного проекту. Перевірка здійснюється через підменю **Project** меню **File** головного меню робочого вікна вибором пункту: **Save & Check** або натисканням лівої кнопки мишки на піктограмі відповідного інструмента основної панелі інструментів.

Виклик діалогового вікна компіляції здійснюється за допомогою відповідної піктограми основної панелі інструментів (див. рис. 1.11).

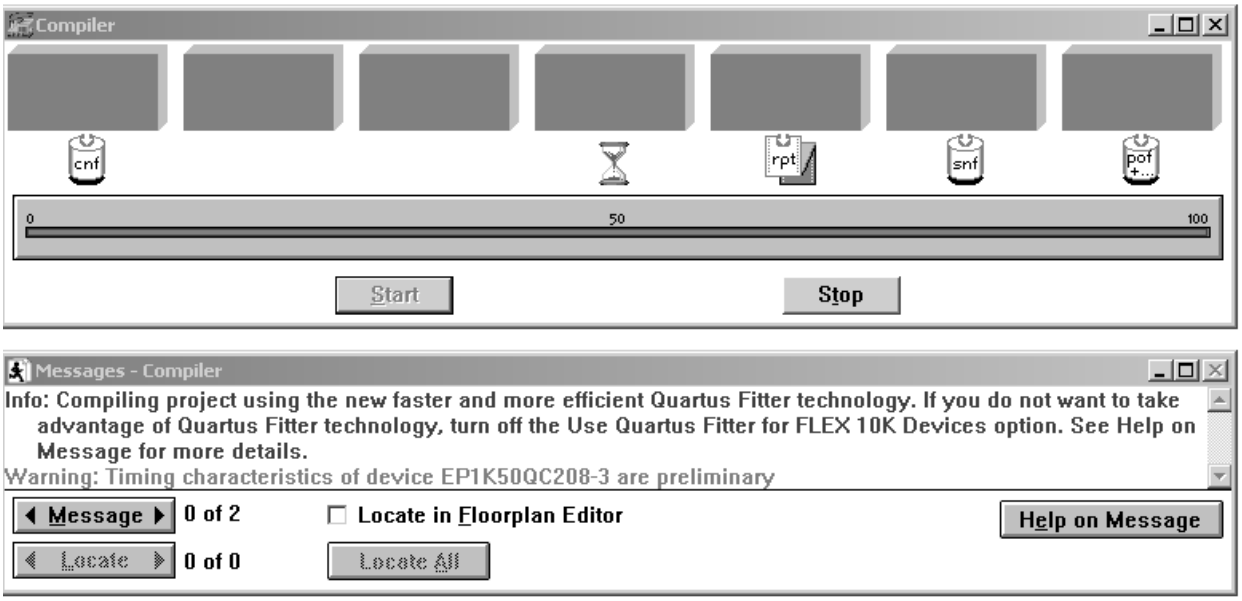


Рисунок 1.11 – Вікно компілятора

Після натискання на кнопку Start відбувається запуск процесу компіляції проекту, при успішному завершенні якого з'являється повідомлення, аналогічне зображеному на рис. 1.12.

Рисунок 1.12 – Вікно повідомлення про успішний процес компіляції

У випадку одержання повідомлень про помилки необхідно перевірити правильність з'єднань схемних елементів і їхніх позначень. Після успішної компіляції проекту необхідно натиснути на піктограму конфігурування

ПЛІС і вибрати пункт Configure (див. рис. 1.13). Перед цим необхідно впевнитися в підключенні інтерфейсного кабелю програматора й шнура живлення до лабораторного макета МЛ-2 і увімкнути клавішу електроживлення.

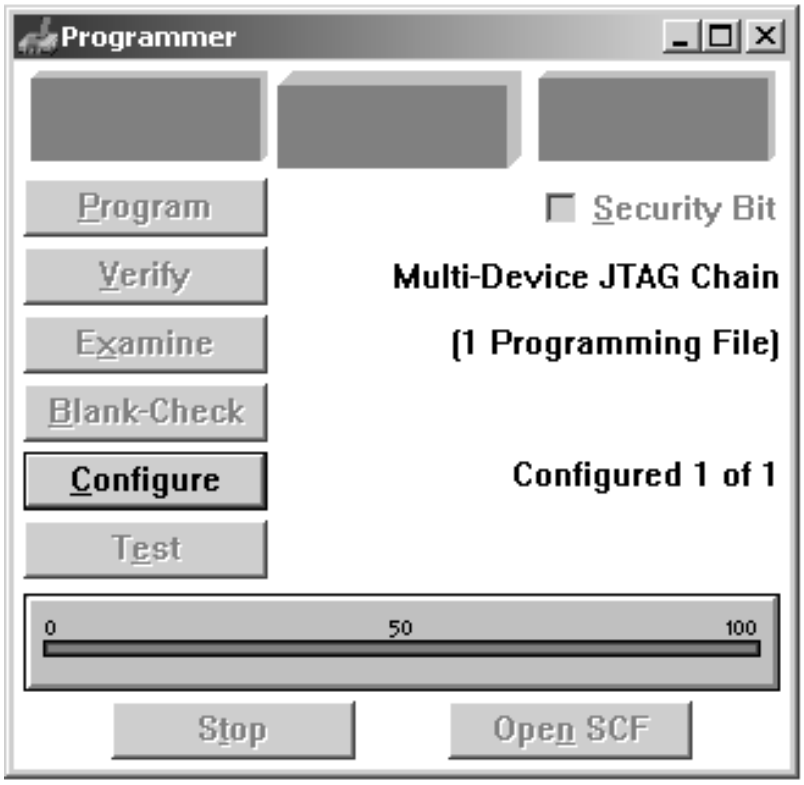


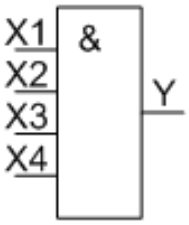
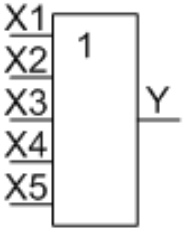
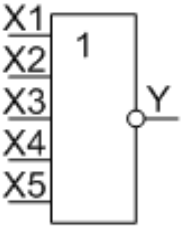
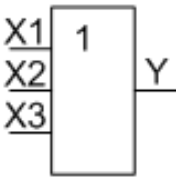
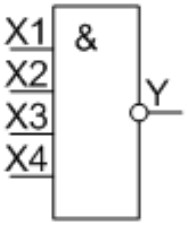
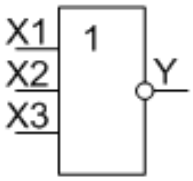
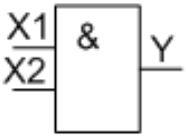
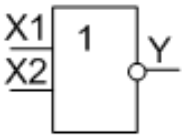
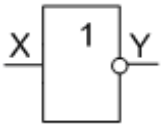
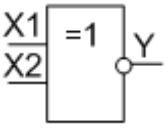
Рисунок 1.13 – Вікно конфігурування ПЛІС

Далі проводиться етап тестування працездатності синтезованої схеми шляхом натискання на відповідні вхідні кнопки F1 F2 лабораторні макети МЛ-2 і перевірки світіння світлодіода під час виконання логічних операцій. Згідно із принципами підключення кнопок і світлодіодів до виводів ПЛІС (див. рис. 4.3, 4.4) за відсутності натискань на кнопки на входах логічного елемента «І» перебуватимуть рівні логічної «одиниці» і на виході - відповідно з таблицею істинності рівень логічної «одиниці» (світлодіод не горітиме), під час натискання хоча б на одну із кнопок (F1, F2) на відповідних входах логічного елемента «І» перебуватимуть рівні логічного «нуля», а на виході логічний «нуль» (світлодіод горить).

Результати роботи необхідно занести у звіт згідно з вимогою до оформлення звіту з лабораторної роботи у підрозділі 1.5.

Варіанти індивідуальних завдань наведені в табл. 1.1.

Таблиця 1.1 – Варіанти індивідуальних завдань

Номер варіанта	Умовне графічне позначення	Номер варіанта	Умовне графічне позначення
1		6	
2		7	
3		8	
4		9	
5		10	

1.5 Зміст звіту

У звіті необхідно навести характеристики лабораторної обчислювальної системи. Вікно графічного інтерфейсу середовища MAX+PLUS II з повністю підготовленою для синтезу пристрою схемою. Експериментально знято таблицю істинності для розробленого пристрою. Стислі висновки по роботі, у яких необхідно вказати особливості синтезу цифрових пристроїв на ПЛІС у середовищі MAX+PLUS II.

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть принципи розробки цифрових пристроїв на ПЛІС.
2. Поясніть основні переваги використання ПЛІС під час проектування цифрових пристроїв.
3. Які типи ПЛІС є найбільш перспективними в цей час?
4. Які дії відбуваються на етапі компіляції проекту?
5. Як виконується програмування ПЛІС?
6. Поясніть основні особливості архітектури ПЛІС на основі технології FPGA.

Лабораторна робота 2

ВИВЧЕННЯ СЕРЕДОВИЩА АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ MAX+PLUS II І РЕАЛІЗАЦІЯ НАЙПРОСТІШИХ ЛОГІЧНИХ ЕЛЕМЕНТІВ НА ПЛІС ЗА ГРАФІЧНИМ ОПИСОМ

2.1 Мета роботи

Вивчення можливостей реалізації найпростіших логічних елементів на ПЛІС за їхнім алгоритмічним описом мовою VHDL.

2.2 Опис лабораторної установки

Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3. Зовнішній вигляд передньої й інтерфейсної панелей наведені на рис. 4.1 і 4.2, нумерація виводів ПЛІС із позначенням підключених модулів наведені в таблиці 4.1.

2.3 Вказівки до організації самостійної роботи

Під час підготовки до лабораторної роботи необхідно проробити теоретичний матеріал за літературою [1–8] і за конспектами лекцій, ознайомитися зі структурою й принципами функціонування ПЛІС типу FPGA, можливостями алгоритмічного опису цифрових пристроїв мовою проектування апаратури VHDL у середовищі MAX+PLUS II.

2.4 Порядок виконання роботи

Етапи виконання роботи розглянемо на прикладі синтезу двовходового логічного елемента «І», розглянутого в лабораторній роботі №1. Більшість основних діалогових вікон розглянуто в лабораторній роботі №1 і не вимагають додаткових коментарів.

Запустіть пакет MAX+PLUS II, відкриється головне вікно MAX+PLUS II.

Відкрийте текстовий документ: після вибору пункту меню **File\New** у діалоговому вікні необхідно вибрати пункт **Text Editor file** (див. рис. 2.1).

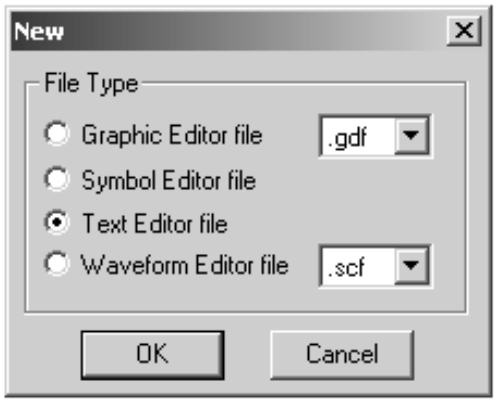


Рисунок 2.1 – Діалогове вікно вибору типу створюваного файла

Збережіть порожню сторінку, що відкрилася, текстового документа (пункт **меню File/Save As**) як файл із вихідним текстом програми мовою VHDL – вибравши розширення ***.vhd** у відповідному діалоговому вікні (див. рис. 2.2) і вказавши ім'я файла, наприклад, pro_i, при цьому ім'я об'єкта моделювання, що вказується після **ENTITY**, має збігатися з ім'ям файла.

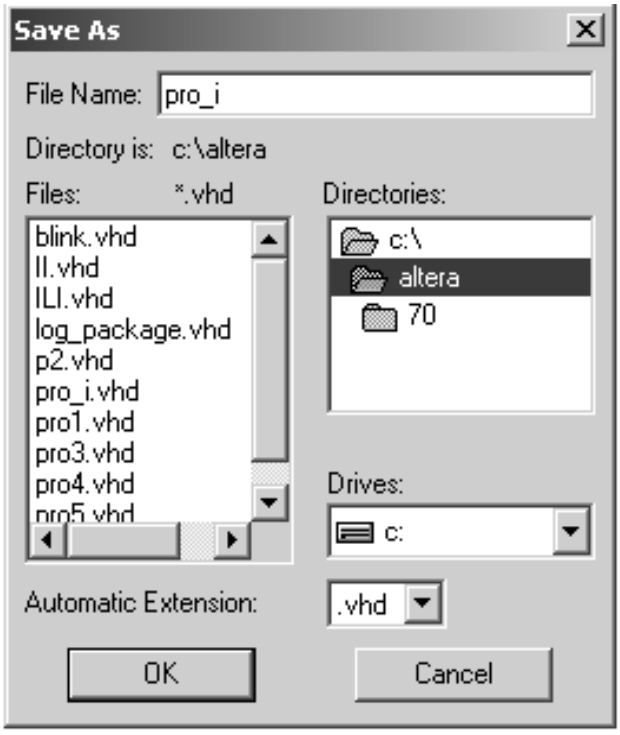


Рисунок 2.2 – Діалогове вікно збереження файла

Введіть алгоритмічний опис заданого схемного елемента мовою VHDL.

Програма (опис об'єкта моделювання) мовою VHDL складається з таких структурних частин:

- підключення бібліотек і пакетів, наприклад:

LIBRARY ieee;

USE ieee.std_logic_1164.all;

де **ieee –** ім'я бібліотеки, **std_logic_1164 –** ім'я пакета, ключове слово **all** вказує на доступність усіх компонентів бібліотеки;

- опису інтерфейсу об'єкта моделювання **ENTITY** із зазначенням зовнішніх виводів, їхніх режимів і типів даних, наприклад, для двовходового логічного елемента «І» (позначеного в програмі як II) опис інтерфейсної частини виглядає в такий спосіб:

ENTITY II IS

PORT(X1,X2: IN std_logic; Y1:OUT std_logic);

END II;

- опису архітектури **ARCHITECTURE** об'єкта моделювання – принципів його роботи на структурному, поведінковому й потоковому рівнях. Приклад потокового опису архітектури для вищезазначеного компонента «І» II наведений нижче

ARCHITECTURE BEHII OF II IS

BEGIN

Y1<=X1 AND X2;

END BEHII.

Після написання коду програми необхідно зберегти даний текстовий файл, створити проект для даного VHDL-опису (пункти меню **File/Project/Set Project for Current File**), вибрати тип ПЛІС шляхом виклику з пункту меню **Assign** підпункту **Device**, у якому необхідно ввести позначення типу використовуваної в лабораторному макеті ПЛІС (**Device Family** – ACEX1K;

EP1K50QC 208-3, див. рис. 1.7). Після цього в пункті меню **Assign** (підпункт **Pin/Location/Chip**) необхідно привласнити інтерфейсним сигналам номери виводів ПЛІС. Вхідні лінії X1 і X2 відповідатимуть кнопкам, наприклад, F1 - 71, F2 - 73, а вихід схеми – Y1 світлодіоду D0 - 86. Повний текст програми VHDL-Опису логічного елемента «І» і вікно призначення інтерфейсних сигналів наведені на рис. 2.3.

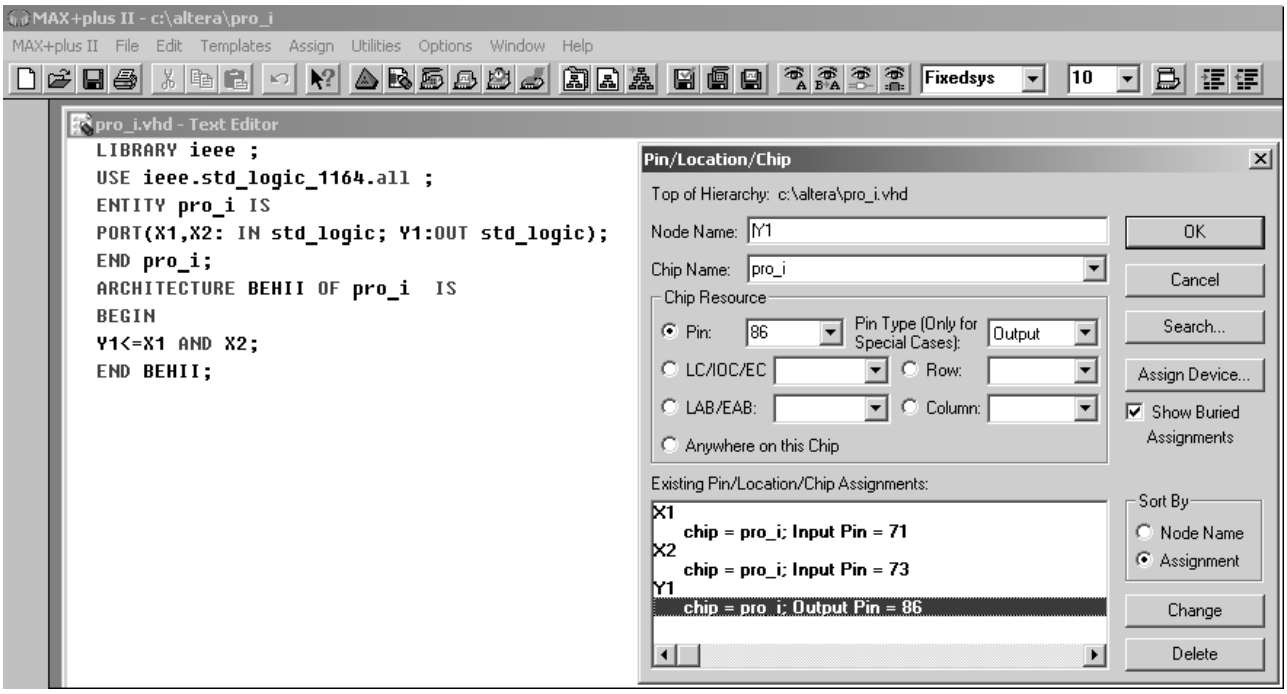


Рисунок 2.3 – Повний текст програми VHDL-Опису логічного елемента «І» і вікно призначення інтерфейсних сигналів

Далі необхідно виконати компіляцію проекту за допомогою натискання на відповідну піктограму основної панелі інструментів (див. рис. 1.11) і натискання на кнопку **Start**. У випадку появи повідомлень про помилки необхідно ознайомитися з їхнім поясненням унизу робочого вікна й номерами рядків, у яких вони були виявлені.

Після успішної компіляції проекту необхідно натиснути на піктограму конфігурування ПЛІС і вибрати пункт **Configure** (див. рис. 1.13). Перед цим необхідно впевнитися в підключенні інтерфейсного кабелю програматора й шнура електроживлення до лабораторного макета МЛ-2 і ввімкнути клавішу електроживлення.

Далі проводиться етап тестування працездатності синтезованої схеми шляхом натискання на відповідні вхідні кнопки F1, F2 лабораторні макети МЛ-2 і перевірки світіння світлодіода під час виконання логічних операцій.

Згідно із принципами підключення кнопок і світлодіодів до виводів ПЛІС (див. рис. 4.3, 4.4) за відсутності натискань на кнопки на входах логічного елемента «І» перебуватимуть рівні логічної «одиниці» і на виході – відповідно з таблицею істинності рівень логічної «одиниці» (світлодіод не горітиме), під час натискання хоча б на одну із кнопок (F1, F2) на відповідних входах логічного елемента «І» перебуватимуть рівні логічного «нуля», а на виході логічний «нуль» (світлодіод горить).

Результати роботи необхідно занести у звіт згідно з вимогою до оформлення звіту з лабораторної роботи у підрозділі 2.5.

Варіанти індивідуальних завдань наведені в таблиці 1.1 (ЛР 1).

2.5 Зміст звіту

У звіті необхідно вказати характеристики лабораторної обчислювальної системи. Лістинг програми мовою VHDL є описом схеми, що відповідає варіанту індивідуального завдання. Вікно графічного інтерфейсу середовища MAX+PLUS II з повністю підготовленою для синтезу пристрою програмою. Експериментально зняту таблицю істинності для розробленого пристрою. Стислі висновки по роботі, у яких необхідно вказати особливості VHDL – описів і синтезу цифрових пристроїв на ПЛІС у середовищі MAX+PLUS II.

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть наявність у мові VHDL загально алгоритмічних і спеціалізованих операторів.
2. Поясніть необхідність принципу паралельного виконання команд у мовах проектування апаратури.
3. З яких частин складається VHDL – опис пристрою?
4. Для чого в мові VHDL використовуються різні рівні опису архітектури?
5. Поясніть призначення інтерфейсної і архітектурної частин VHDL – опису.
6. Поясніть необхідність створення бібліотек компонентів.

Лабораторна робота 3
РЕАЛІЗАЦІЯ КОМБІНАЦІЙНИХ СХЕМ НА ПЛІС ЗА ЇХНІМ ГРАФІЧНИМ ОПИСОМ

3.1 Мета роботи

Вивчення графічного інтерфейсу середовища автоматизованого проектування MAX+PLUS II і реалізації комбінаційних схем на ПЛІС із використанням графічного опису.

3.2 Опис лабораторної установки

Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3. Зовнішній вигляд передньої й інтерфейсної панелей наведені на рис. 4.1 і 4.2, нумерація виводів ПЛІС із позначенням підключених модулів наведені в таблиці 4.1.

3.3 Вказівки до організації самостійної роботи

Під час підготовки до лабораторної роботи необхідно проробити теоретичний матеріал за літературою [1–8] і за конспектами лекцій, ознайомитися зі структурою й принципами функціонування ПЛІС типу FPGA, можливостями графічного опису цифрових пристроїв у середовищі MAX+PLUS II.

3.4 Порядок виконання роботи

Запустіть пакет MAX+PLUS II, відкриється головне вікно MAX+PLUS II.

Розглянемо основні етапи синтезу цифрових схем на ПЛІС за їхнім графічним описанням на прикладі комбінаційної схеми, зображеної на рис. 3.1. Для цього після запуску системи MAX+PLUS II необхідно створити новий файл (**File/New**). У діалоговому вікні, що відкрилося, New вибираємо пункт **Graphic Editor file** і натискаємо кнопку ОК, при цьому автоматично відкривається вікно графічного редактора.

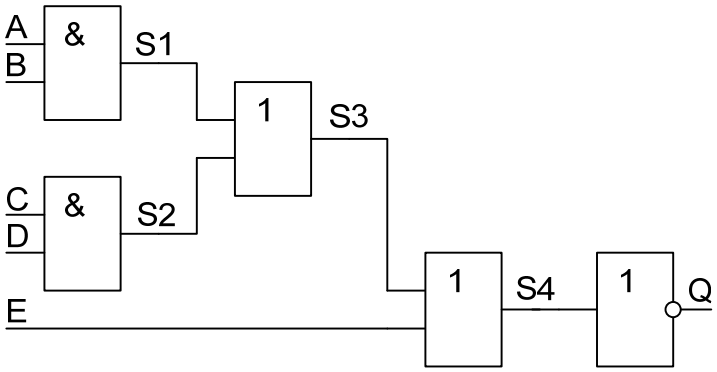


Рисунок 3.1 – Приклад комбінаційної схеми

Ім'я файла проекту обов'язково прив’язується до ім'я проекту. Прив'язка ім'я файла до ім'я проекту здійснюється через підменю **Project** меню **File** головного меню робочого вікна вибором пункту: **Set Project to Current File**.

Подвійним натисканням лівої клавіші мишки на порожньому робочому полі графічного редактора викликається підменю **Enter Symbol** вибору робочої бібліотеки й, безпосередньо, схемотехнічних вузлів, до складу яких входить великий набір основних логічних елементів і модулів введення й виведення (input, output, bidir).

Вибір елементів схеми здійснюється послідовно, шляхом вказівки необхідної бібліотеки у вікні **Symbol Libraries** (використовувана бібліотека prim) діалогового вікна **Enter Symbol**, і вказівки відповідного елемента.

Так само можна переміщати або вилучити обраний елемент. Відповідно до схеми, зображеної на рис. 3.1, виконуємо введення двох елементів логічне «і» (and2), двох елементів логічне «або» (or2), інвертора (not), 5-ти модулів введення (input), що відповідають вхідним сигналам A, B, C, D, E і одного модуля виведення (output), що відповідає вихідному сигналу Q. Далі необхідно з'єднати відповідні елементи лініями зв'язку, піктограма вибору яких перебуває в графічному меню ліворуч від робочого вікна графічного редактора. Проведення сигнальних ліній здійснюється сполученням курсору мишки з відображенням елемента, при цьому курсор автоматично перетворюється в інструмент рисування ортогональних ліній (перехрестя), і натисканням лівої кнопки мишки (при втриманні якої проводиться лінія). За один прийом можна провести два ортогональних відрізки лінії.

Далі необхідно ввести позначення для зовнішніх виводів схеми (A,B,C,D,E,Q), вибравши режим редагування текстової інформації схемних компонентів одиночним натисканням лівої клавіші мишки по відповідному підпису.

Потім виконується вибір типу ПЛІС шляхом виклику з пункту меню **Assign**/**Device**, у якому необхідно ввести позначення типу використовуваної в лабораторному макеті ПЛІС (Device Family – ACEX1K; EP1K50QC 208-3. Після цього в пункті меню **Assign**/**Pin/Location/Chip** необхідно привласнити інтерфейсним сигналам номери виводів ПЛІС. Вхідні лінії відповідатимуть

кнопкам, наприклад, A (F1-71), B (F2-73), 3 (F3-74), D (F4-75), E (F5-79), а вихід схеми - світлодіоду Q (D0 - 86).

Вікно графічного редактора з повністю набраною принциповою схемою й інтерфейс призначення виводів схеми наведені на рис. 3.2 і 3.3 відповідно.

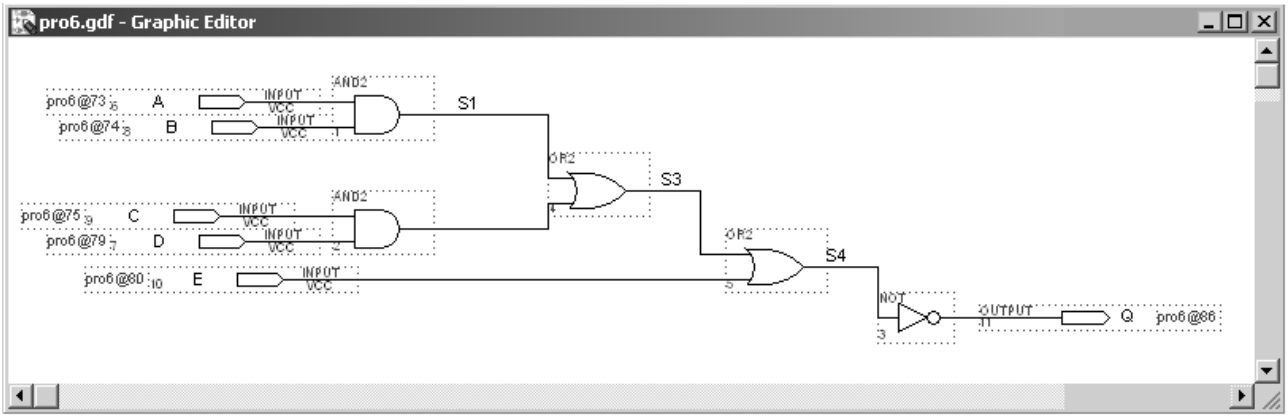


Рисунок 3.2 – Вікно графічного редактора з повністю набраною принциповою схемою

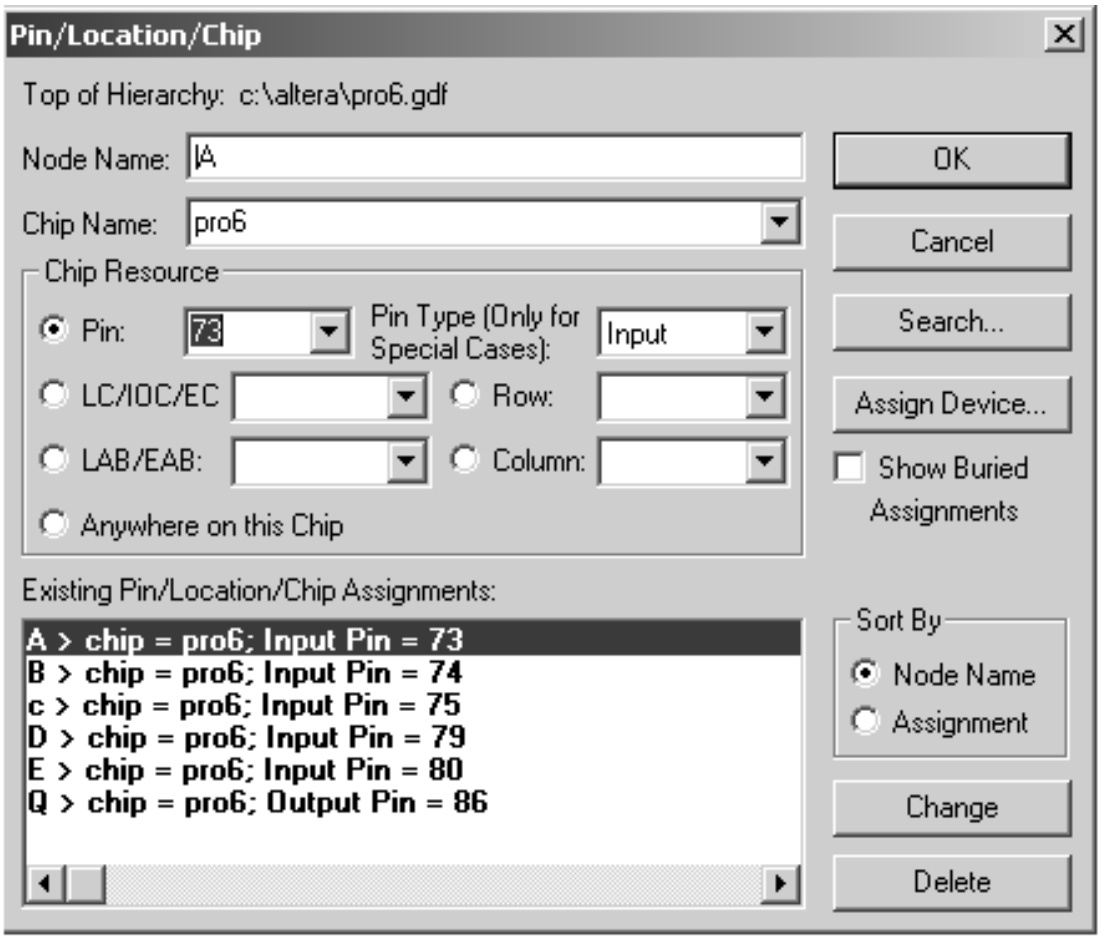


Рисунок 3.3 – Діалогове вікно відповідності інтерфейсу схеми виводам ПЛІС

Наступним етапом є те, що необхідно виконати компіляцію створеного проекту. Перед компіляцією можна виконати перевірку коректності введеного проекту. Перевірка здійснюється через меню **File/Project** меню **File** головного меню робочого вікна вибором пункту: **Save & Check** або клацанням лівої кнопки мишки на піктограмі відповідного інструмента основної панелі інструментів.

Виклик меню компіляції здійснюється за допомогою відповідної піктограми основної панелі інструментів. Під час натискання на кнопку **Start** відбувається запуск процесу компіляції проекту.

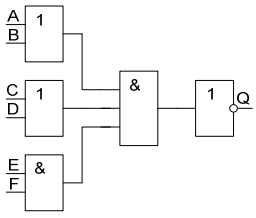
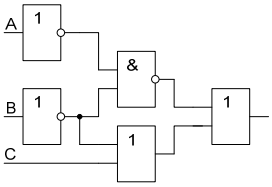
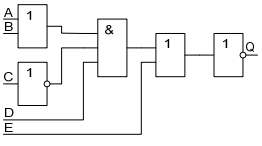
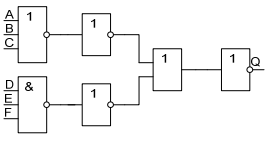
Після успішної компіляції проекту необхідно натиснути на піктограму конфігурування ПЛІС і вибрати пункт Configure. Перед цим необхідно впевнитися в підключенні інтерфейсного кабелю програматора й шнура електроживлення до лабораторного макета МЛ-2 і ввімкнути клавішу електроживлення.

Далі проводиться етап тестування працездатності синтезованої схеми шляхом натискання на відповідні вхідні кнопки (F1, F2, F3, F4, F5) лабораторного макета МЛ-2 і перевірки світіння світлодіода під час виконання логічних операцій. При цьому необхідно пам'ятати про принципи підключення кнопок і світлодіодів до виводів ПЛІС.

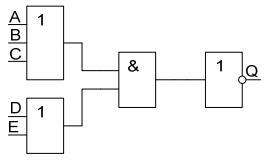
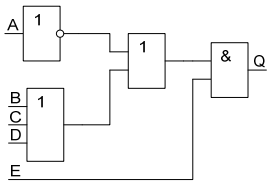
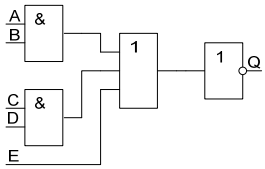
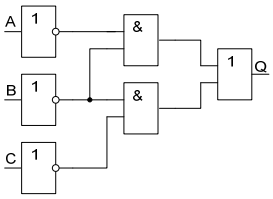
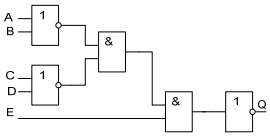
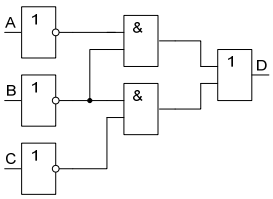
Результати роботи необхідно занести у звіт згідно з вимогою до оформлення звіту з лабораторної роботи в підрозділі 3.5.

Варіанти індивідуальних завдань наведені в таблиці 3.1.

Таблиця 3.1 – Варіанти індивідуальних завдань

Номер варіанта	Умовне графічне позначення	Номер варіанта	Умовне графічне позначення
1	2	3	4
1		6	
2		7	

Продовження табл. 3.1

1	2	3	4
3		8	
4		9	
5		10	

3.5 Зміст звіту

У звіті необхідно вказати: характеристики лабораторної обчислювальної системи, лістинг програми мовою VHDL з описом схеми, що відповідає варіанту індивідуального завдання, вікно графічного інтерфейсу середовища MAX+PLUS II з повністю підготовленою для синтезу пристрою програмою, експериментально зняту таблицю істинності для розробленого пристрою, стислі висновки по роботі, у яких необхідно зазначити особливості графічного опису й синтезу цифрових пристроїв на ПЛІС у середовищі MAX+PLUS II.

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть сутність архітектури ПЛІС на основі ПЛМ/ПМЛ.
2. Поясніть сутність архітектури ПЛІС на основі БМК.
3. Поясніть сутність архітектури ПЛІС на основі CPLD.
4. Яким параметром характеризується логічна складність ПЛІС?
5. Поясніть поняття трасувальної здатності ПЛІС.
6. У чому полягає сутність системного підходу під час проектування цифрових пристроїв?

Лабораторна робота 4
РЕАЛІЗАЦІЯ КОМБІНАЦІЙНИХ СХЕМ НА ПЛІС ЗА ЇХНІМ АЛГОРИТМІЧНИМ ОПИСОМ

4.1 Мета роботи

Вивчення можливостей реалізації комбінаційних схем на ПЛІС за їхнім алгоритмічним структурним описом мовою VHDL.

4.2 Опис лабораторної установки

Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3. Зовнішній вигляд передньої й інтерфейсної панелей наведені на рис. 4.1 і 4.2, нумерація виводів ПЛІС із позначенням підключених модулів наведені в таблиці 4.1.

4.3 Вказівки до організації самостійної роботи

Під час підготовки до лабораторної роботи необхідно проробити теоретичний матеріал за літературою [1–8] і за конспектами лекцій, ознайомитися зі структурою й принципами функціонування ПЛІС типу FPGA і принципами алгоритмічного структурного опису цифрових пристроїв мовою проектування апаратури VHDL, а також з особливостями середовища MAX+PLUS II.

4.4 Порядок виконання роботи

Етапи виконання роботи розглянемо на прикладі синтезу на ПЛІС комбінаційної схеми, зображеної на рисунку 4.1, за її алгоритмічним VHDL-описом на структурному рівні. Більшість основних діалогових вікон розглянуто в лабораторній роботі № 2 і не вимагають додаткових коментарів. Детальніший опис схеми мовою VHDL наведений у підрозділі 2.2.

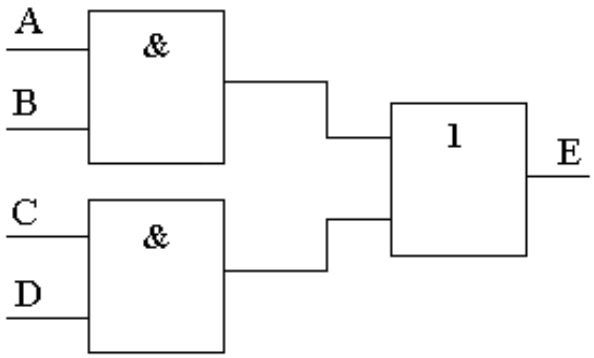


Рисунок 4.1 – Приклад комбінаційної схеми

Запустіть пакет MAX+PLUS II, відкриється головне вікно MAX+PLUS II.

Відкрийте текстовий документ: після вибору пункту меню **File\New** у діалоговому вікні необхідно вибрати пункт **Text Editor file**.

Збережіть порожню сторінку, що відкрилася, текстового документа (пункт **меню File/Save** As) як файл із вихідним текстом програми мовою VHDL – вибравши розширення ***.vhd** у відповідному діалоговому вікні (див. рис. 2.2) і вказавши ім'я файла, наприклад, pro4 (ім'я об'єкта моделювання, що вказується після ENTITY, має збігатися з ім'ям файла).

Введіть алгоритмічний опис заданого схемного елемента мовою VHDL з використанням позначень на рис. 4.2.

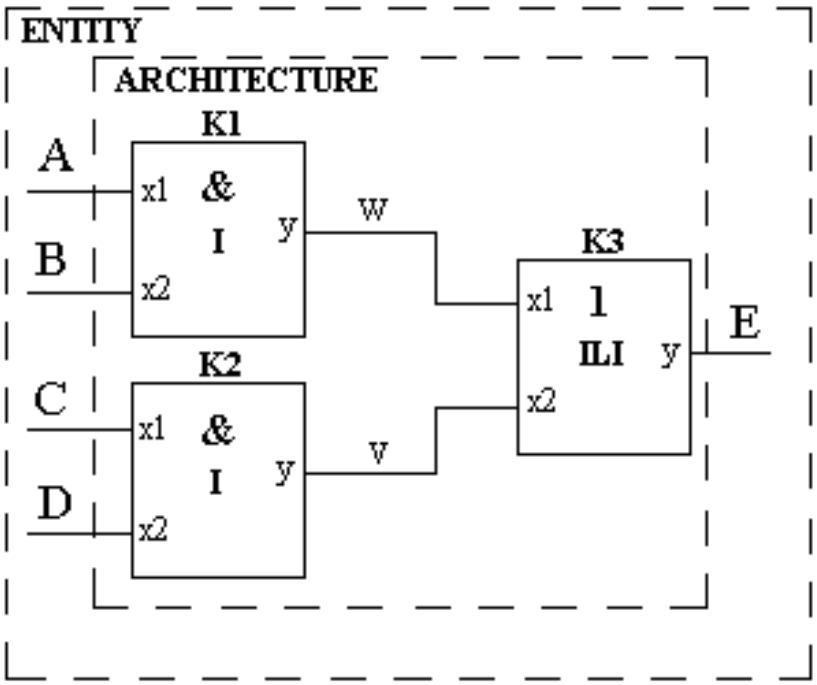


Рисунок 4.2 – Приклад комбінаційної схеми з описом компонентів і внутрішніх сигналів

Повний текст розглянутої програми наводиться нижче.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY pro4 IS
PORT(A,B,C,D: IN std_logic; E:OUT std_logic);
END pro4;
ARCHITECTURE STRUCT OF pro4 IS
COMPONENT I
PORT(X1,X2:IN std_logic;
Y:OUT std_logic);
END COMPONENT;
COMPONENT ILI
PORT(X1,X2:IN std_logic;
Y:OUT std_logic);
END COMPONENT;
SIGNAL W,V:std_logic;
BEGIN
M1: I PORT MAP(A,B,W);
M2: I PORT MAP(C,D,V);
M3: ILI PORT MAP(W,V,E);
END STRUCT;
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
```

```
ENTITY I IS

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

END I;

ARCHITECTURE BEHII OF I IS

BEGIN

Y<=X1 AND X2;

END BEHI;

LIBRARY ieee ;

USE ieee.std_logic_1164.all ;

ENTITY ILI IS

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

END ILI;

ARCHITECTURE BEHILI OF ILI IS

BEGIN

Y<=X1 OR X2;

END BEHILI;
```

Після написання коду програми необхідно зберегти даний текстовий файл як **pro4.vhd**, створити проект для даного VHDL-опису (пункти меню **File/Project/Set Project for Current File**), вибрати тип ПЛІС шляхом виклику з пункту меню **Assign** підпункту **Device**, у якому необхідно ввести позначення типу використовуваної в лабораторному макеті ПЛІС (**Device Family** – ACEX1K; EP1K50QC 208-3, див. рис. 1.7). Після цього в пункті меню **Assign** (підпункт **Pin/Location/Chip**) необхідно привласнити інтерфейсним сигналам номери виводів ПЛІС. Вхідні лінії A,B,C,D відповідатимуть кнопкам, а вихід E - світлодіоду. Вікно призначення інтерфейсних сигналів наведено на рис. 4.3.

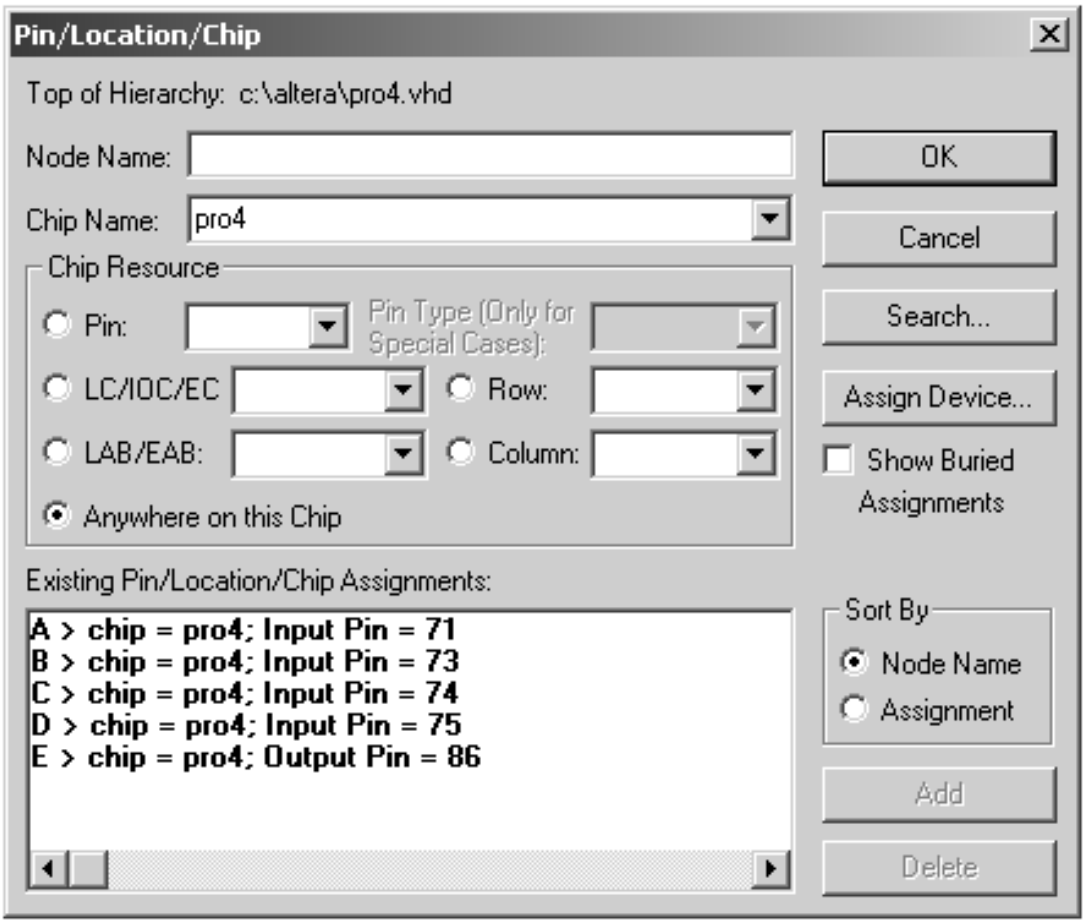


Рисунок 4.3 – Вікно призначення інтерфейсних сигналів схеми

Далі необхідно виконати компіляцію проекту за допомогою натискання на відповідну піктограму основної панелі інструментів (див. рис. 1.11) і натискання на кнопку **Start**. У випадку появи повідомлень про помилки необхідно ознайомитися з їхнім поясненням унизу робочого вікна й номерами рядків, у яких вони були виявлені.

Після успішної компіляції проекту необхідно натиснути на піктограму конфігурування ПЛІС і вибрати пункт **Configure** (див. рис. 1.13). Перед цим необхідно впевнитися у підключенні інтерфейсного кабелю програматора й шнура електроживлення до лабораторного макета МЛ-2 і ввімкнути клавішу електроживлення.

Далі проводиться етап тестування працездатності синтезованої схеми шляхом натискання на відповідні вхідні кнопки лабораторного макета МЛ-2 і перевірки світіння світлодіода під час виконання логічних операцій. При цьому необхідно пам'ятати принципи підключення кнопок і світлодіодов до виводів ПЛІС (див. рис. 4.3, 4.4).

Результати роботи необхідно занести у звіт згідно з вимогою до оформлення звіту з лабораторної роботи у підрозділі 4.5.

Варіанти індивідуальних завдань наведені в таблиці 3.1 (див. ЛР № 3).

4.5 Зміст звіту

У звіті необхідно вказати характеристики лабораторної обчислювальної системи. Лістинг програми мовою VHDL є описом схеми, що відповідає варіанту індивідуального завдання. Вікно графічного інтерфейсу середовища MAX+PLUS II з повністю підготовленою для синтезу пристрою програмою. Експериментально зняту таблицю істинності для розробленого пристрою. Стислі висновки по роботі, у яких необхідно вказати особливості VHDL - описів і синтезу цифрових пристроїв на ПЛІС у середовищі MAX+PLUS II.

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть принцип структурного опису схем мовою VHDL.
2. Для яких цілей використовуються різні типи описів архітектури?
3. Поясніть наявність паралельної й послідовної частин VHDL-Програми.
4. Поясніть на прикладах види позиційної й ключової відповідності сигналів в операторі конкретизації компонента port map.
5. Як відбувається виконання паралельних операторів у мові VHDL?
6. Які оператори входять у синтезовану підмножину операторів VHDL?

Лабораторна робота 5
РЕАЛІЗАЦІЯ СХЕМ З ПАМ'ЯТТЮ НА ПЛІС ЗА ЇХНІМ ПОВЕДІНКОВИМ VHDL-ОПИСОМ

5.1 Мета роботи

Вивчення можливостей середовища автоматизованого проектування MAX+PLUS II і реалізації на ПЛІС схем з пам'яттю на ПЛІС за їхнім алгоритмічним поведінковим VHDL-описом.

5.2 Опис лабораторної установки

Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3.

5.3 Вказівки до організації самостійної роботи

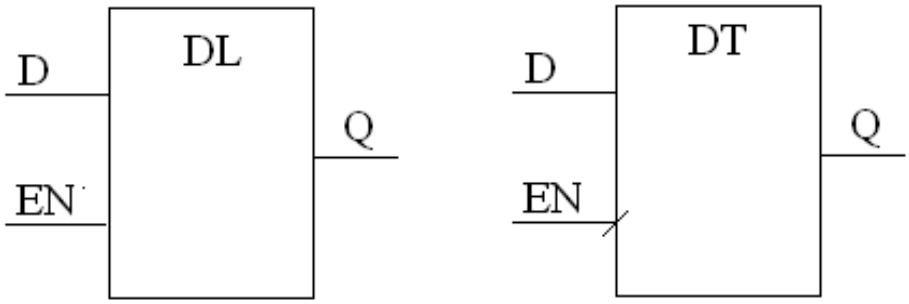
Під час підготовки до лабораторної роботи необхідно проробити теоретичний матеріал за літературою [1–8] і за конспектами лекцій, ознайомитися зі структурою й принципами функціонування ПЕВМ типу IBM PC/AT, системою команд для мікроконтролера фірми Altera.

5.4 Порядок виконання роботи

Етапи виконання роботи розглянемо на прикладі синтезу на ПЛІС елементів з пам'яттю (див. рис. 5.1):DL- D-засувки (D-latch), що спрацьовує за рівнем сигналу дозволу EN, і D-Тригера (D-trigger), що спрацьовує за переднім фронтом сигналу дозволу EN.

Розглянемо докладно синтез D-засувки (див. рис. 5.1,а) за її поведінковим VHDL-Описом.

Запустіть пакет MAX+PLUS II, відкриється головне вікно MAX+PLUS II. Більшість основних діалогових вікон розглянуто в лабораторній роботі № 2 і не вимагають додаткових коментарів.



а) б)

Рисунок 5.1 – Приклад елементів з пам'яттю: а) D-D- засувка, б) D - тригер

Відкрийте текстовий документ: після вибору пункту меню **File\New** у діалоговому вікні необхідно вибрати пункт **Text Editor file**.

Збережіть порожню сторінку, що відкрилася, текстового документа (пункт **меню File/Save** As) як файл із вихідним текстом програми мовою VHDL – вибравши розширення ***.vhd** у відповідному діалоговому вікні (див. рис. 2.2) і вказавши ім'я файла, наприклад, DL. Введіть алгоритмічний опис заданого схемного елемента мовою VHDL (ім'я об'єкта моделювання, що вказується після ENTITY, має збігатися з ім'ям файла).

Опис DL - D-засувки проводиться на поведінковому рівні, за якого необхідно описати алгоритм роботи синтезованого пристрою за допомогою послідовних VHDL-Операторів, поміщених у тіло оператора PROCESS. Першим етапом необхідно підключити використовувані бібліотеки:

```
library ieee;
use ieee.std_logic_1164.all;
```

Інтерфейсними сигналами **PORT** об'єкта моделювання DL будуть D і EN, оголошені як вхідні, і Q – вихідний сигнал. Всі сигнали мають тип **std_logic:**

```
entity DL is
port(D,EN: in std_logic; Q:out std_logic);
end entity DL;
```

Поведінкова архітектура **a1** розроблювального пристрою містить оператор процесу **PROCESS** із зазначеним у дужках списком чутливості (EN, D) вхідних сигналів, зміна яких призводить до запуску програми, розташованої в тілі оператора **PROCESS.** Паралельний оператор **PROCESS** виконує функцію операторних дужок для послідовної частини VHDL-програми, що у цьому випадку складається з оператора умовного призначення сигналу if, за допомогою якого реалізується алгоритм роботи D-засувки: сигнал із входу D проходить на вихід Q тільки при позитивному рівні сигналу на вході дозволу EN, при низькому рівні сигналу EN на виході пристрою зберігається попередня величина рівня сигналу:

```
process (EN, D) begin
if  (EN='1') then Q<=D;
end if;
end process;
```

Повний текст поведінкового VHDL-Опису D-засувки наводиться нижче.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity DL is
port(D,EN: in std_logic; Q:out std_logic);
end entity DL;
architecture a1 of DL is
BEGIN
process (EN, D) begin
if  (EN='1') then Q<=D;
end if;
end process;
END architecture a1;
```

Після написання коду програми необхідно зберегти даний текстовий файл як **DL.vhd**, створити проект для даного VHDL-опису (пункти меню **File/Project/Set Project for Current File**), вибрати тип ПЛІС шляхом виклику з пункту меню **Assign** підпункту **Device**, у якому необхідно ввести позначення типу використовуваної в лабораторному макеті ПЛІС (**Device Family** – ACEX1K; EP1K50QC 208-3, див. рис. 1.7). Після цього в пункті меню **Assign** (підпункт **Pin/Location/Chip**) необхідно привласнити інтерфейсним сигналам номери виводів ПЛІС. Вхідні сигнали D,EN відповідатимуть кнопкам, а вихід Q – світлодіоду. Вікно призначення інтерфейсних сигналів наведено на рис. 5.2.

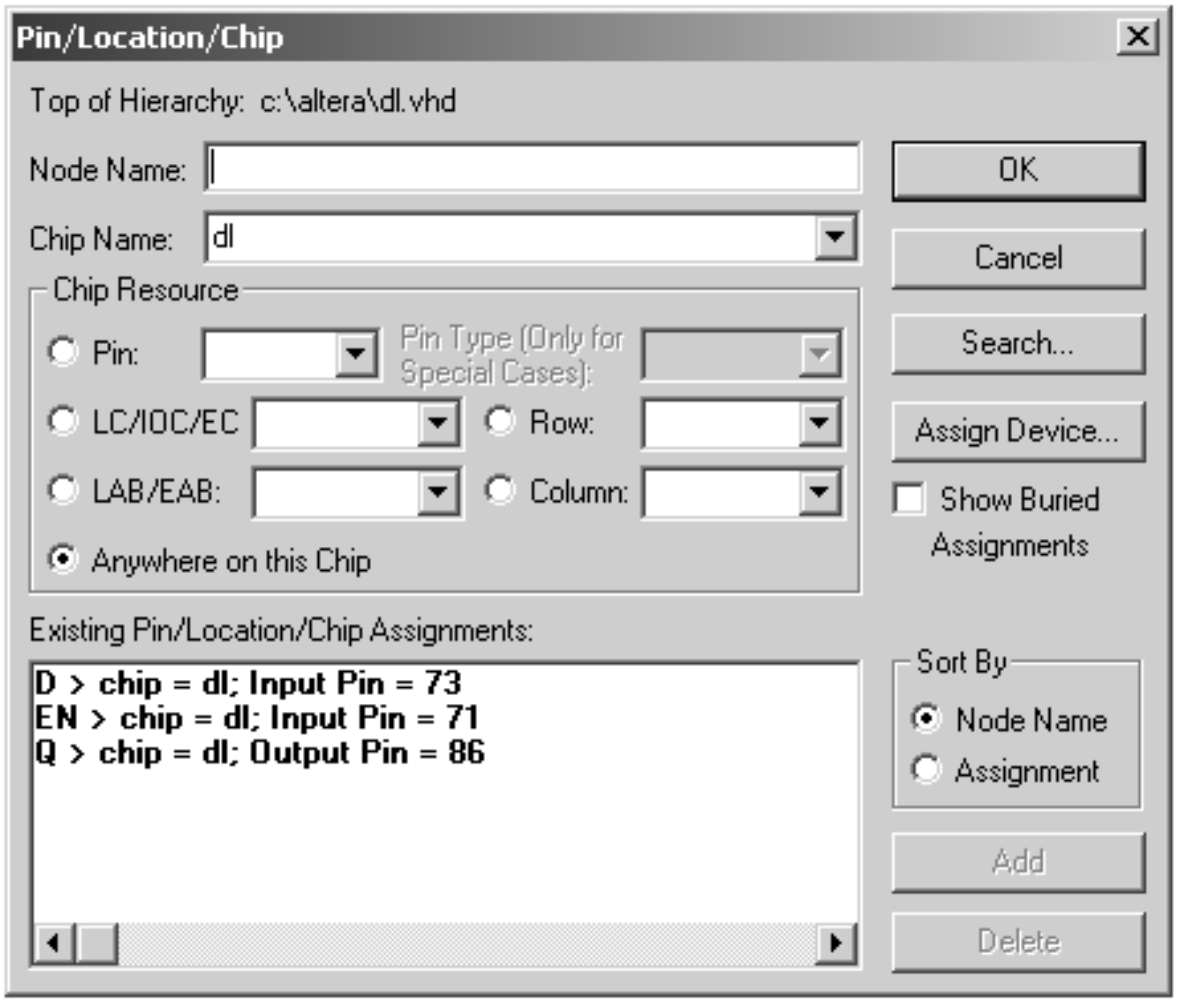


Рисунок 5.2 – Вікно призначення інтерфейсних сигналів пристрою

Далі необхідно виконати компіляцію проекту за допомогою натискання на відповідну піктограму основної панелі інструментів (див. рис. 1.11) і натискання на кнопку **Start**. У випадку появи повідомлень про помилки необхідно ознайомитися з їхнім поясненням унизу робочого вікна й номерами рядків, у яких вони були виявлені.

Після успішної компіляції проекту необхідно натиснути на піктограму конфігурування ПЛІС і вибрати пункт **Configure** (див. рис. 1.13). Перед цим необхідно впевнитися у підключенні інтерфейсного кабелю програматора й шнура електроживлення до лабораторного макета МЛ-2 і ввімкнути клавішу електроживлення.

Далі проводиться етап тестування працездатності синтезованої схеми шляхом натискання на відповідні вхідні кнопки лабораторного макета МЛ-2 і перевірки світіння світлодіода під час виконання відповідних дій. При цьому необхідно пам'ятати принципи підключення кнопок і світлодіодів до виводів ПЛІС (див. рис. 3, 4).

В ході опису пристроїв, що спрацьовують не за рівнем, а за фронтом керуючого сигналу (наприклад D-Тригер, зображений на рис. 5.1,б) умова переднього фронту на VHDL записується за допомогою атрибута сигналу Event. Наприклад, для керуючого сигналу EN умова переднього фронту:

```
if (EN'event and EN='1') then...
```

Повний текст програми поведінкового VHDL-опису D-тригера наводиться нижче.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity DL is
port(D,EN: in std_logic; Q:out std_logic);
end entity DL;
architecture a1 of DL is
BEGIN
process (EN, D) begin
if  (EN'event and EN='1') then Q<=D;
end if;
end process;
END architecture a1;
```

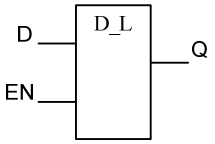
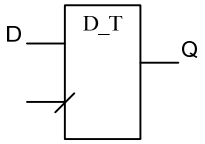
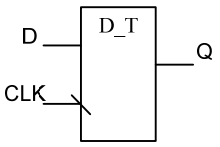
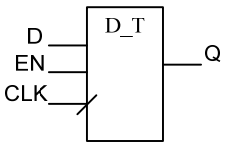
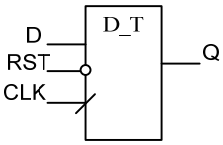
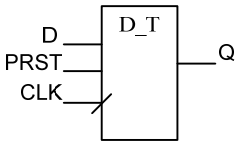
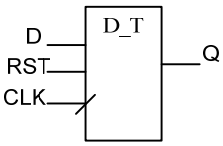
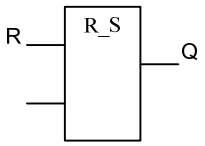
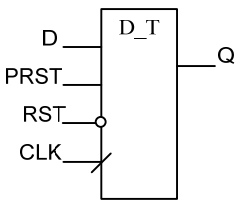
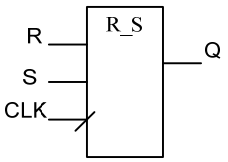
Результати роботи необхідно занести у звіт згідно з вимогою до оформлення звіту з лабораторної роботи в підрозділі 5.5.

Варіанти індивідуальних завдань наведені в таблиці 5.1.

5.5 Зміст звіту

У звіті необхідно вказати характеристики лабораторної обчислювальної системи. Лістинг програми мовою VHDL є описом схеми, що відповідає варіанту індивідуального завдання. Вікно графічного інтерфейсу середовища MAX+PLUS II з повністю підготовленою для синтезу пристрою програмою. Експериментально зняту таблицю істинності для розробленого пристрою. Стислі висновки по роботі, у яких необхідно вказати особливості VHDL - описів і синтезу цифрових пристроїв на ПЛІС у середовищі MAX+PLUS II.

Таблиця 5.1 – Варіанти індивідуальних завдань

Номер варіанта	Умовне графічне позначення	Номер варіанта	Умовне графічне позначення
1		6	
2		7	
3		8	
4		9	
5		10	

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть особливості VHDL-описів тригерів і засувок.
2. Поясніть принцип поведінкового опису архітектури в мові VHDL.
3. Поясніть на прикладах типи затримок у мові VHDL.
4. Поясніть можливість утворення паразитних засувок у ході синтезу комбінаційних пристроїв.
5. Поясніть основні типи даних і класи об'єктів у мові VHDL.
6. Що означають атрибути сигналів у мові VHDL?

Лабораторна робота 6

РЕАЛІЗАЦІЯ ПЕРЕТВОРЮВАЧІВ КОДУ, МУЛЬТИПЛЕКСОРІВ І ДЕМУЛЬТИПЛЕКСОРІВ НА ПЛІС ЗА ЇХНІМ ПОВЕДІНКОВИМ VHDL-ОПИСОМ

6.1 Мета роботи

Вивчення можливостей середовища автоматизованого проектування MAX+PLUS II і реалізації на ПЛІС перетворювачів коду, мультиплексорів і демультиплексорів за їхнім алгоритмічним поведінковим VHDL-описом.

6.2 Опис лабораторної установки

Лабораторна робота виконується в індивідуальному порядку. На робочому місці кожного студента встановлений ПК типу IBM PC/AT з інстальованим програмним забезпеченням MAX+PLUS II і лабораторний макет МЛ-2, що включає в себе ПЛІС фірми Altera EP1K50QC 208-3.

6.3 Вказівки до організації самостійної роботи

Під час підготовки до лабораторної роботи необхідно проробити теоретичний матеріал за літературою [1–8] і за конспектами лекцій, ознайомитися зі структурою й принципами функціонування ПЕВМ типу IBM PC/AT, системою команд для мікроконтролера фірми Altera.

6.4 Порядок виконання роботи

Етапи виконання роботи розглянемо на прикладі синтезу на ПЛІС мультиплексора із двома адресними входами (див. рис. 6.1) за його поведінковим описом.

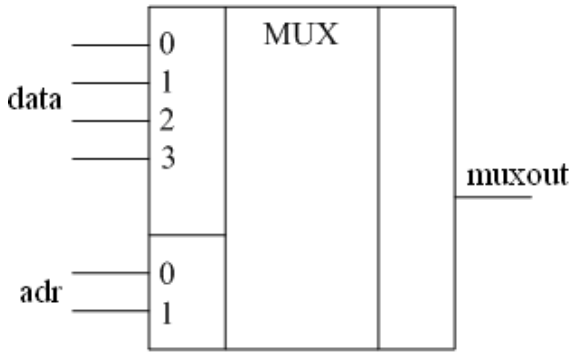


Рисунок 6.1 – Мультиплексор із двома адресними входами

Запустіть пакет MAX+PLUS II, відкриється головне вікно MAX+PLUS II. Більшість основних діалогових вікон розглянуто в лабораторній роботі № 2 і не вимагають додаткових коментарів.

Відкрийте текстовий документ: після вибору пункту меню **File\New** у діалоговому вікні необхідно вибрати пункт **Text Editor file**.

Збережіть порожню сторінку, що відкрилася, текстового документа (пункт **меню File/Save As**) як файл із вихідним текстом програми мовою VHDL – вибравши розширення ***.vhd** у відповідному діалоговому вікні (див. рис. 2.2) і вказавши ім'я файла, наприклад, pro3. Введіть алгоритмічний опис заданого схемного елемента мовою VHDL (ім'я об'єкта моделювання, що вказується після ENTITY, має збігатися з ім'ям файла).

Опис пристрою pro3 - мультиплексора із двома адресними входами проводиться на поведінковому рівні, при якому описується алгоритм роботи синтезованого пристрою за допомогою послідовних VHDL-операторів, поміщених у тіло оператора PROCESS. Першим етапом необхідно підключити використовувані бібліотеки:

```
library ieee;
use ieee.std_logic_1164.all;
```

Інтерфейсними сигналами **PORT** об'єкта моделювання pro3 будуть вхідні векторні сигнали data (чотирирозрядний) і adr (дворозрядний), а також вихідний сигнал **muxout.** Всі сигнали мають тип **std_logic:**

```
entity pro3 is
port(data: in std_logic_vector(3 downto 0);
adr:in std_logic_vector(1 downto 0);
muxout:out std_logic);
end entity pro3;
```

Поведінкова архітектура **a1** розроблювального пристрою містить оператор процесу **PROCESS** із зазначеним у дужках списком чутливості (**adr,data**) вхідних сигналів, зміна яких призводить до запуску програми, розташованої в тілі оператора **PROCESS.** Паралельний оператор **PROCESS** виконує функцію операторних дужок для послідовної частини VHDL-програми, що у цьому випадку складається з оператора вибіркового призначення сигналу **case**, за допомогою якого реалізується алгоритм роботи мультиплексора: на вихід передається сигнал із вхідної лінії, номер якої заданий на адресних входах.

```
process (adr,data) begin
case adr is
WHEN "00" => muxout <= data(0);
when "01" => muxout <= data(1);
when "10" => muxout <= data(2);
when "11" => muxout <= data(3);
```

```
when others => muxout <='0' ;
end case;
end process;
```

Повний текст поведінкового VHDL-опису мультиплексора із двома адресними входами наводиться нижче.

```
library ieee;
use ieee.std_logic_1164.all;
entity pro3 is
port(data: in std_logic_vector(3 downto 0);
adr:in std_logic_vector(1 downto 0);
muxout:out std_logic);
end entity pro3;
architecture a1 of pro3 is
begin
process (adr,data) begin
case adr is
WHEN "00" => muxout <= data(0);
when "01" => muxout <= data(1);
when "10" => muxout <= data(2);
when "11" => muxout <= data(3);
when others => muxout <='0' ;
end case;
end process;
END architecture a1;
```

Після написання коду програми необхідно зберегти даний текстовий файл як **pro3.vhd**, створити проект для даного VHDL-Опису (пункти меню **File/Project/Set Project for Current File**), вибрати тип ПЛІС шляхом виклику з пункту меню **Assign** підпункту **Device**, у якому необхідно ввести позначення типу використовуваної в лабораторному макеті ПЛІС (**Device Family** – ACEX1K; EP1K50QC 208-3, див. рис. 1.7). Після цього в пункті меню **Assign** (підпункт **Pin/Location/Chip**) необхідно привласнити інтерфейсним сигналам номери виводів ПЛІС. Вхідні сигнали data і adr відповідатимуть кнопкам, а вихід muxout – світлодіоду. Вікно призначення інтерфейсних сигналів наведено на рис. 6.2.

Далі необхідно виконати компіляцію проекту за допомогою натискання на відповідну піктограму основної панелі інструментів (див. рис. 1.11) і натискання на кнопку **Start**. У випадку появи повідомлень про помилки необхідно ознайомитися з їхнім поясненням унизу робочого вікна й номерами рядків, у яких вони були виявлені.

Після успішної компіляції проекту необхідно натиснути на піктограму конфігурування ПЛІС і вибрати пункт **Configure** (див. рис. 1.13). Перед цим необхідно впевнитися в підключенні інтерфейсного кабелю програматора й

шнура електроживлення до лабораторного макета МЛ-2 і ввімкнути клавішу електроживлення.

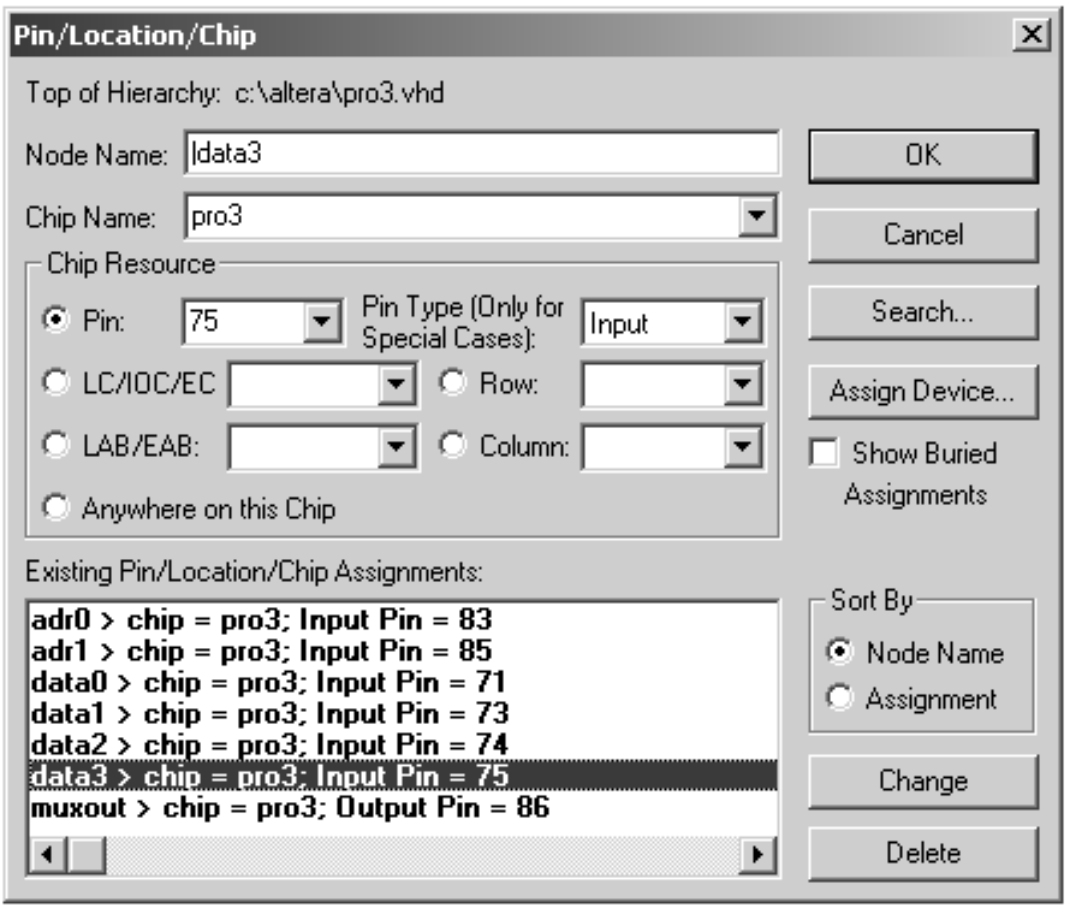


Рисунок 6.2 – Вікно призначення інтерфейсних сигналів пристрою

Далі проводиться етап тестування працездатності синтезованої схеми шляхом натискання на відповідні вхідні кнопки лабораторного макета МЛ-2 і перевірки світіння світлодіода під час виконання пристроєм відповідних операцій. При цьому необхідно пам'ятати принципи підключення кнопок і світлодіодів до виводів ПЛІС (див. рис. 4.3, 4.4).

Результати роботи необхідно занести у звіт згідно з вимогою до оформлення звіту по лабораторній роботі у підрозділі 6.5.

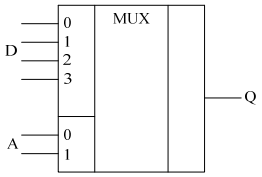
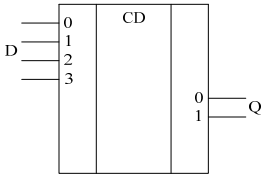
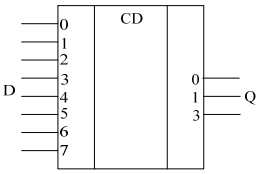
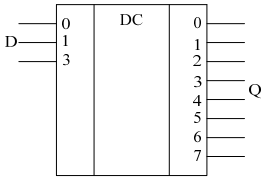
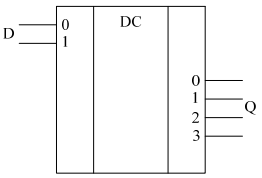
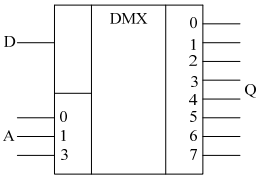
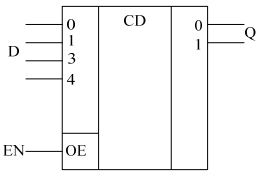
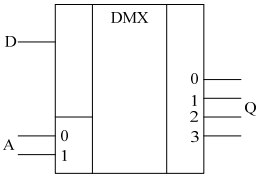
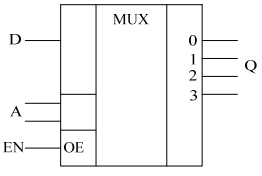
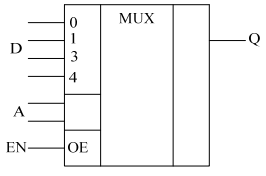
Варіанти індивідуальних завдань наведені в таблиці 6.1.

6.5 Зміст звіту

У звіті необхідно вказати характеристики лабораторної обчислювальної системи. Лістинг програми мовою VHDL з описом схеми, що відповідає варіанту індивідуального завдання. Вікно графічного інтерфейсу середовища

MAX+PLUS II з повністю підготовленою для синтезу пристрою програмою. Експериментально зняту таблицю істинності для розробленого пристрою. Стислі висновки по роботі, у яких необхідно вказати особливості VHDL – описів і синтезу цифрових пристроїв на ПЛІС у середовищі MAX+PLUS II.

Таблиця 6.1 – Варіанти індивідуальних завдань

Номер варіанта	Умовне графічне позначення	Номер варіанта	Умовне графічне позначення
1		6	
2		7	
3		8	
4		9	
5		10	

КОНТРОЛЬНІ ЗАПИТАННЯ ТА ЗАВДАННЯ

1. Поясніть відмінності операторів паралельного і послідовного призначення сигналу.

2. Поясніть на прикладах необхідність зазначення оператора wait у тілі оператора process.

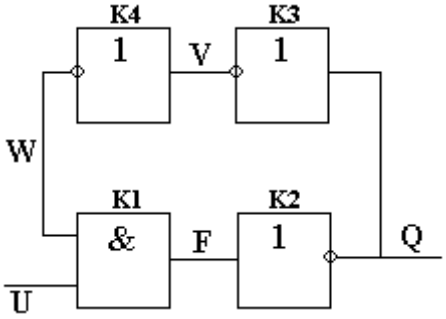
3. Поясніть синтаксис та особливості оператора process мовою VHDL.

4. Що таке віртуальний процесорний елемент?

5. Поясніть принципи роботи послідовних і паралельних операторів умовного й вибіркового призначення сигналу.

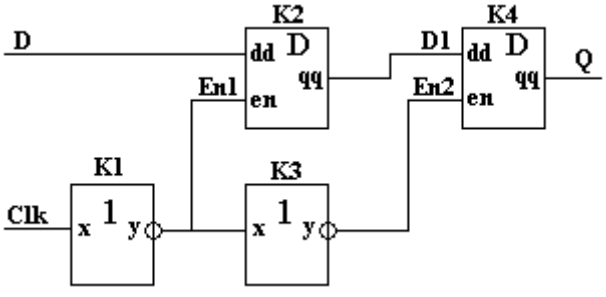
6. Поясніть необхідність зазначення списку чутливості в операторі process.

Генератор




```
entity gener is
Port (U:in bit;Q:inout bit);
end gener;
architecture arh1 of gener is
signal F,V,W: bit;
begin
Q<=not F after 1 ns;
F<=U and W after 1 ns;
V<=not Q after 10 ns;
W<=not V after 10 ns;
end architecture;
```

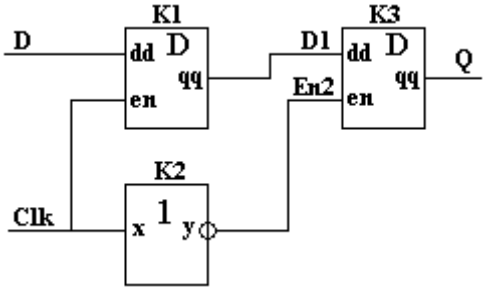
Тригер, що тактується переднім фронтом




```
entity shtrig is
port (d,clk: in bit; q: out bit);
end shtrig;
```

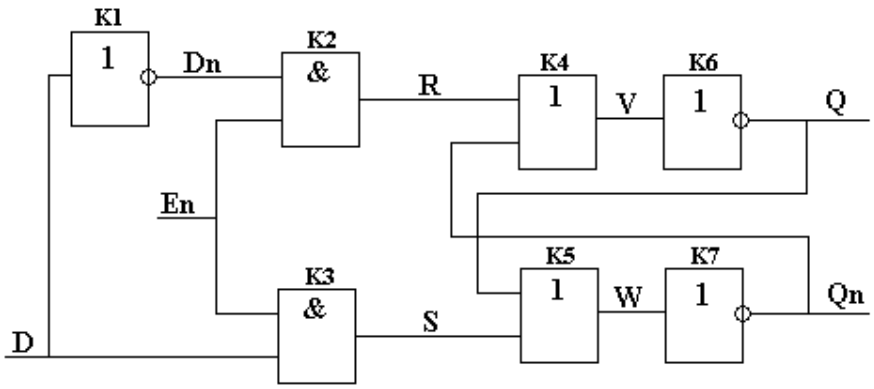
```
architecture art of shtrig is
component dz
port (dd, en: in bit; qq:out bit);
end component;
component ne
port (x: in bit; y:out bit);
end component;
signal d1,en1,en2 : bit;
begin
K1: ne port map(x=>clk,y=>en1);
K2: dz port map(dd=>d,en=>en1,qq=>d1);
K3: ne port map(x=>en1,y=>en2);
K4: dz port map(dd=>d1,en=>en2,qq=>q);
end art;
entity ne is
port (x: in bit; y:out bit);
end ne;
architecture arh1 of ne is begin
y<= not x after 1 ns;
end arh1;
entity dz is
port (dd, en: in bit; qq:out bit);
end dz;
architecture arh2 of dz is
begin
process (en)
begin
if en = '1' then qq<=dd after 1 ns;
end if;
end process;
end arh2;
```

Тригер, що тактується заднім фронтом



```
entity shtrig is
port (d,clk: in bit; q:out bit);
end shtrig;
architecture art of shtrig is
component dz
port (dd,en: in bit; qq:out bit);
end component;
component ne
port (x: in bit; y:out bit);
end component;
signal d1, en2 : bit;
begin
K1: dz port map(dd=>d,en=>clk,qq=>d1);
K2: ne port map(x=>clk,Y=>en2);
K3: dz port map(d=>d1,en=>en2,qq=>q);
end art;
entity ne is
port (x: in bit; y:out bit);
end ne;
architecture arh1 of ne is begin
y<= not x after 1 ns;
end arh1;
entity dz is
port (dd,en: in bit; qq:out bit);
end dz;
architecture arh2 of dz is
begin
process (en)
begin
if en = '1' then qq<=dd after 2 ns;
end if;
end process;
end arh2;
```

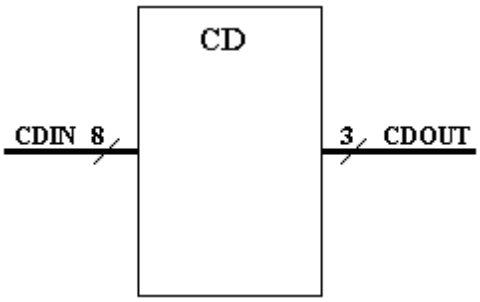
D-тригер, що тактується за рівнем сигналу En.




```
entity trg is
port(d,en:in bit;
q,qn:inout bit);
end trg;
architecture arh1 of trg 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;
component ne
port(x:in bit; y:out bit);
end component;
signal dn,r,s,v,w: bit;
begin
K1:ne port map(d,dn);
K2:i port map(dn,en,r);
K3:i port map(d,en,s);
K4:ili port map(r,qn,v);
K5:ili port map(s,q,w);
K6:ne port map(v,q);
K7:ne port map(w,qn);
end architecture;
entity i is
port(x1,x2:in bit; y: out bit);
end i;
architecture arh1 of i is begin
y<= (x1 and x2) after 10 ns;
end arh1;
entity ili is
port(x1,x2:in bit; y: out bit);
end ili;
```

```
architecture arh1 of ili is begin
y<= (x1 or x2) after 10 ns;
end arh1;
entity ne is
port(x:in bit; y: out bit);
end ne;
architecture arh1 of ne is begin
y<=not x after 100 ns;
end arh1;
```

Шифратор




```
library ieee;
use ieee.std_logic_1164.all;
entity cd is
port(CDIN: in std_logic_vector(7 downto 0);
CDOUT: out std_logic_vector(2 downto 0));
end cd;
architecture arhcd1 of cd is
begin
process (CDIN) begin
case CDIN is
when "00000001" => CDOUT<="000" after 100 ns;
when "00000010" => CDOUT<="001" after 100 ns;
when "00000100" => CDOUT<="010" after 100 ns;
when "00001000" => CDOUT<="011" after 100 ns;
when "00010000" => CDOUT<="100" after 100 ns;
when "00100000" => CDOUT<="101" after 100 ns;
when "01000000" => CDOUT<="110" after 100 ns;
when "10000000" => CDOUT<="111" after 100 ns;
end case;
end process;
end architecture;
```

Опис шифратора за допомогою функцій перетворювання

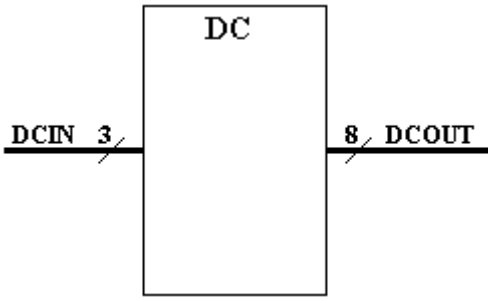
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity cd is
port(CDIN: in std_logic_vector (7 downto 0);
CDOUT: out std_logic_vector (2 downto 0));
end cd;
architecture arhcd2 of cd is
function  fcd (signal X: std_logic_vector (7 downto 0) ) return std_logic_vector
(2 downto 0)  is
begin
for i in 0 to 7 loop
if X(i)='1' then return conv_std_logic_vector (i,3);
end if;
end loop;
return "000";
end fc;
begin
CDOUT<=fcd (CDIN);
end architecture;
```

У зворотному порядку від старшого до молодшого – пріоритетний шифратор.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity cd is
port(CDIN: in std_logic_vector (7 downto 0);
CDOUT: out std_logic_vector (2 downto 0));
end cd;
architecture arhcd2 of cd is
function  fcd (signal X: std_logic_vector (7 downto 0) ) return std_logic_vector
(2 downto 0)  is
begin
for i in 7 downto 0 loop
if X(i)='1' then return conv_std_logic_vector (i,3);
end if;
end loop;
return "000";
end fc;
```

```
begin
CDOUT<=fcd (CDIN);
end architecture;
```



Приклад опису дешифратора

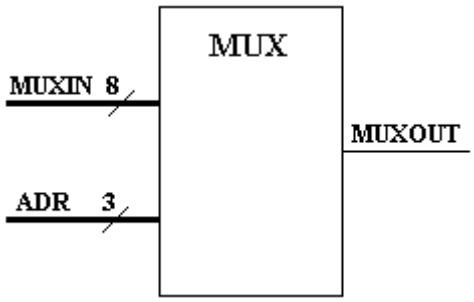
```
library ieee;
use ieee.std_logic_1164.all;
entity dc is
port(DCIN: in std_logic_vector(2 downto 0);
DCOUT: out std_logic_vector(7 downto 0));
end dc;
architecture arhdc1 of dc is
begin
process (DCIN) begin
case DCIN is
when "000" => DCOUT<="00000001" after 100 ns;
when "001" => DCOUT<="00000010" after 100 ns;
when "010" => DCOUT<="00000100" after 100 ns;
when "011" => DCOUT<="00001000" after 100 ns;
when "100" => DCOUT<="00010000" after 100 ns;
when "101" => DCOUT<="00100000" after 100 ns;
when "110" => DCOUT<="01000000" after 100 ns;
when "111" => DCOUT<="10000000" after 100 ns;
end case;
end process;
end architecture;
```

Приклад опису дешифратора за допомогою функцій перетворення

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity dc is
```

```
port(DCIN: in std_logic_vector(2 downto 0);
DCOUT: out std_logic_vector(7 downto 0));
end dc;
architecture arhdc1 of dc is
begin
process (DCIN)
variable V:integer range 0 to 7;
variable VSP: std_logic_vector(7 downto 0);
begin
V:=conv_integer (DCIN);
For I:=0 to 7 loop
VSP(I):='0';
end loop
VSP(V)<= '1';
DOUT<=VSP;
end loop;
end process;
end architecture;
```

Приклад опису мультиплексора



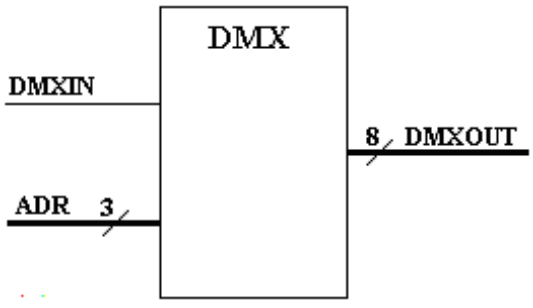

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port(MUXIN: in std_logic_vector(7 downto 0);
ADR: in std_logic_vector(2 downto 0);
MUXOUT:out std_logic);
end mux;
architecture arhmux1 of mux is
begin
process (MUXIN, ADR) begin
case ADR is
when "000" => MUXOUT<= MUXIN(0) after 100 ns;
when "001" => MUXOUT<= MUXIN(1) after 100 ns;
```

```
when "010" => MUXOUT<= MUXIN(2) after 100 ns;
when "011" => MUXOUT<= MUXIN(3) after 100 ns;
when "100" => MUXOUT<= MUXIN(4) after 100 ns;
when "101" => MUXOUT<= MUXIN(5) after 100 ns;
when "110" => MUXOUT<= MUXIN(6) after 100 ns;
when "111" => MUXOUT<= MUXIN(7) after 100 ns;
end case;
end process;
end architecture;
```

Другий варіант мультиплексора з використанням функції перетворення

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity mux is
port(MUXIN: in std_logic_vector(7 downto 0);
ADR: in std_logic_vector(2 downto 0);
MUXOUT:out std_logic);
end mux;
architecture arhmux2 of mux is
signal S1:integer range 0 to 7;
begin
S1<=conv_integer (ADR);
MUXOUT<= MUXIN(S1) after 100 ns;
end architecture;
```

Приклад опису демультиплексора




```
library ieee;
use ieee.std_logic_1164.all;
entity dmx is
```

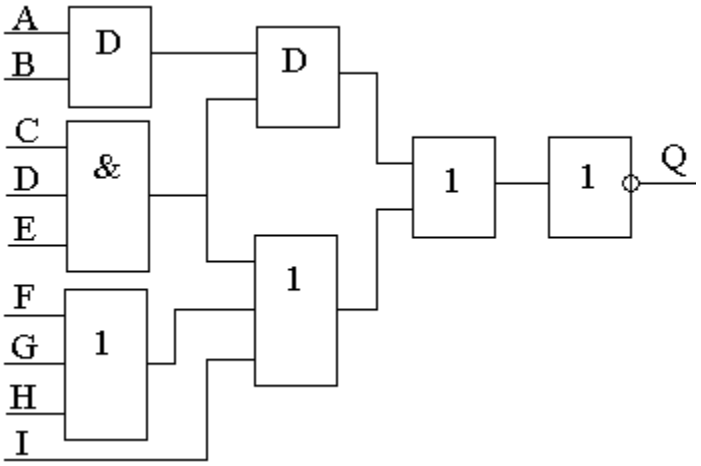
Завдання № 17

1. Структура програми мовою VHDL.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

D<= not F;
C<= B and D;
A<= F and D;
B<= not (D and C and E and F);

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



Завдання № 18

1. Основні оператори паралельного VHDL.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

A<= not F;
C<= B or D;
D<= F and E;
B<= D and A and F;

```
port (DMXIN: in std_logic;
ADR: in std_logic_vector(2 downto 0);
DMXOUT:out std_logic_vector(7 downto 0));
end dmx;
architecture arhdmx1 of dmx is
begin
process (DMXIN, ADR) begin
case ADR is
when "000" => DMXOUT(0)<= DMXIN after 100 ns;
when "001" => DMXOUT(1)<= DMXIN after 100 ns;
when "010" => DMXOUT(2)<= DMXIN after 100 ns;
when "011" => DMXOUT(3)<= DMXIN after 100 ns;
when "100" => DMXOUT(4)<= DMXIN after 100 ns;
when "101" => DMXOUT(5)<= DMXIN after 100 ns;
when "110" => DMXOUT(6)<= DMXIN after 100 ns;
when "111" => DMXOUT(7)<= DMXIN after 100 ns;
end case;
end process;
end arhdmx1;
```

Другий варіант демультиплексора з використанням функції перетворення типів conv_integer () з пакета ieee.std_logic_unsigned.all.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity dmx is
port (DMXIN: in std_logic;
ADR: in std_logic_vector(2 downto 0);
DMXOUT:out std_logic_vector(7 downto 0));
end dmx;
architecture arhdmx2 of dmx is
signal S1:integer range 0 to 7;
begin
S1<=conv_integer (ADR);
MUXOUT(S1)<= MUXIN after 100 ns;
End  architecture;
```

МОДУЛЬНІ КОНТРОЛЬНІ ЗАВДАННЯ З ДИСЦИПЛІНИ

Модуль 1

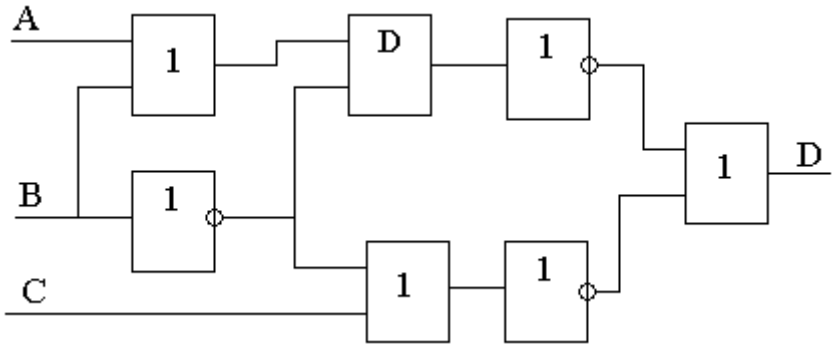
Завдання № 1

1. Пояснити принципи проектування цифрових пристроїв під час використання структур типу ПЛМ.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

```
A<= not B;
C<= F and B;
B<= C and D;
E<= D or B;
```

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



Завдання № 2

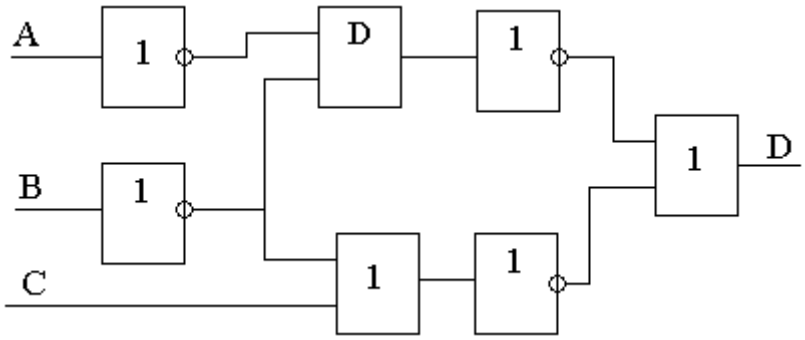
1. Пояснити особливості архітектур ПЛМ та ПМЛ.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

```
A<= not C;
```

C<= F and D;
B<= C and D;
E<= C and B;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:

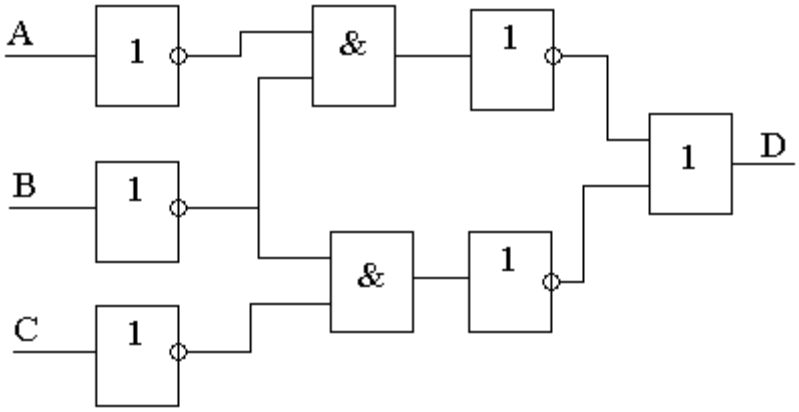


Завдання № 3

1. Назвати сучасні тенденції розвитку структур ПЛМ та ПМЛ.
2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

A<= not B;
C<= B and D;
B<= C and D;
E<= C and B;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



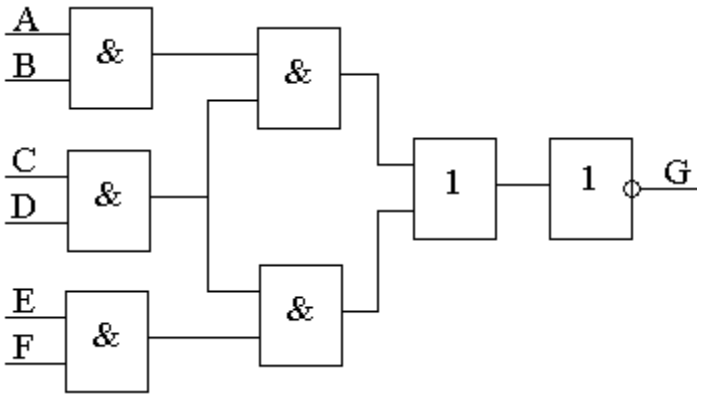
Завдання № 4

1. Пояснити принципи проектування цифрових пристроїв на БМК.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

D<= not B;
C<= B or D or E;
A<= F and D;
B<= C and E;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



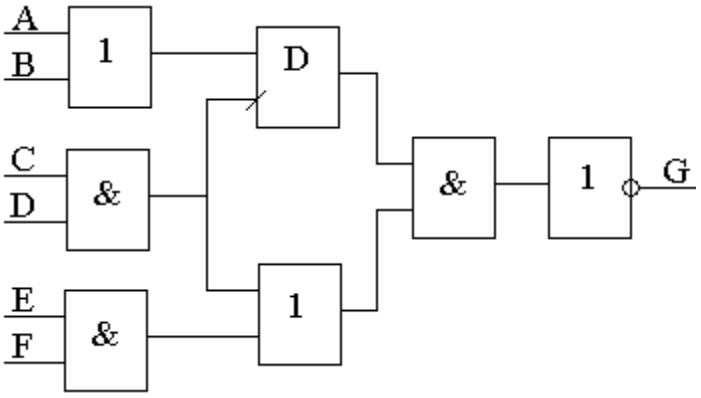
Завдання № 5

1. Як визначається логічна складність пристроїв на БМК ?

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



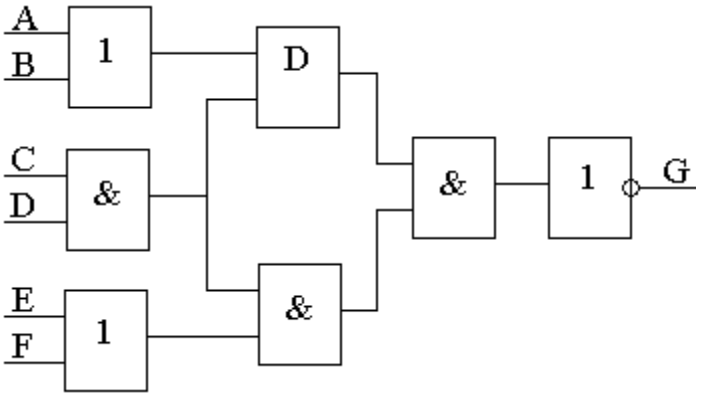
Завдання № 6

1. Пояснити характеристики канальних та безканальних структур БМК.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

D<= C or B;
E<= B and D;
A<= F and D;
B<= C and B;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



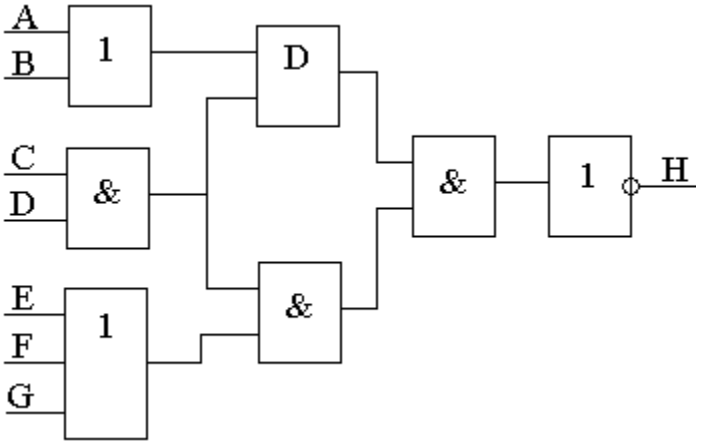
Завдання № 7

1. Пояснити принципи внутрішньої організації структур FPGA.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

```
D<= not F;
C<= not B;
A<= F and D and C;
B<= C or E;
```

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



Завдання № 8

1. Пояснити структуру конфігураційно-логічного блоку КЛБ у FPGA.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

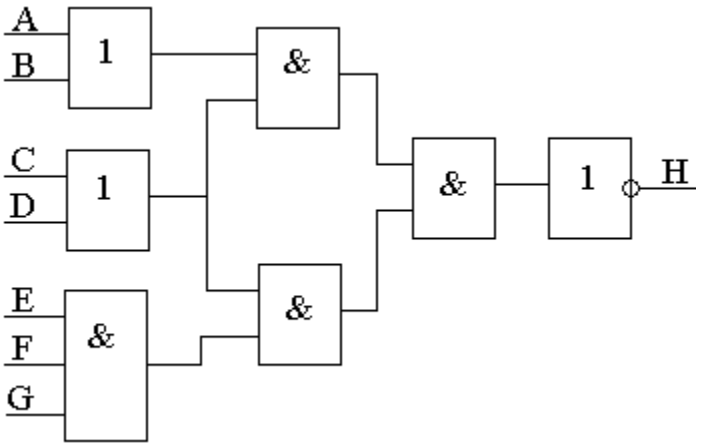
D<= not F;

C<= B or D;

A<= F and D;

B<= C or A;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



Завдання № 9

1. Яку функцію виконує логічна таблиця LUT у структурах FPGA ?

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

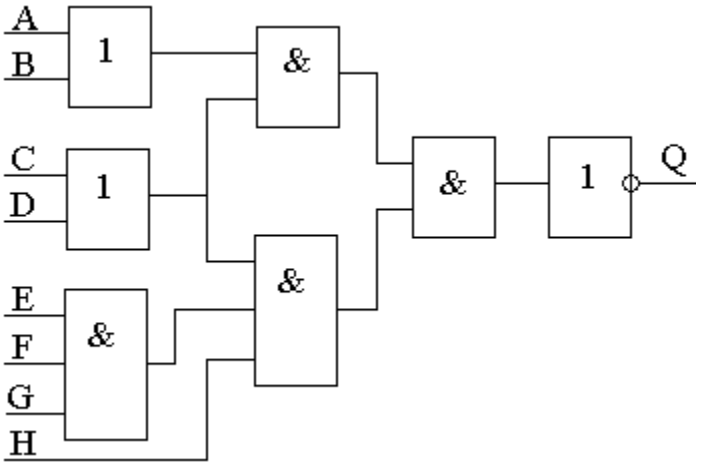
D<= E or F;

C<= B and D;

A<= (F and D) or E;

B<= D and E;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



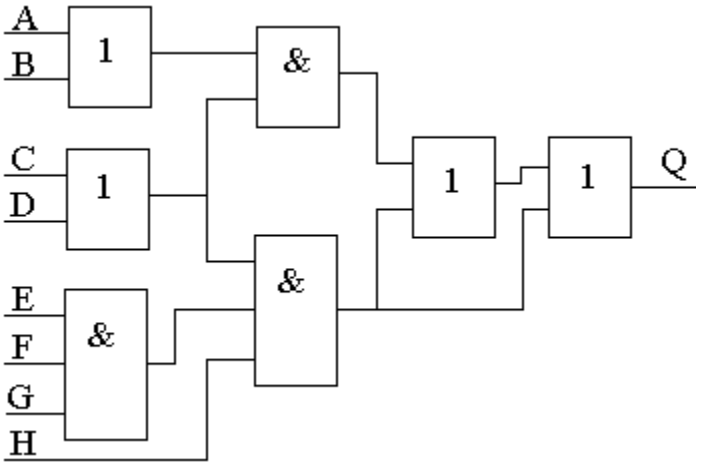
Завдання № 10

1. Пояснити принцип завдання конфігурації у структурах FPGA.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

```
D<= not F;
C<= B or D;
A<= F and D;
B<= D and C and E and F;
```

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



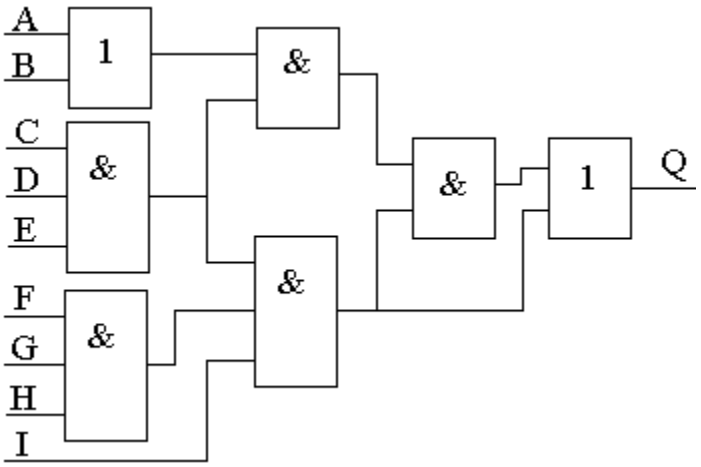
Завдання № 11

1. Основні концепції розвитку архітектур ПЛІС.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

D<=A and F;
C<= B or D;
A<= F and D;
B<= C and E and F;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



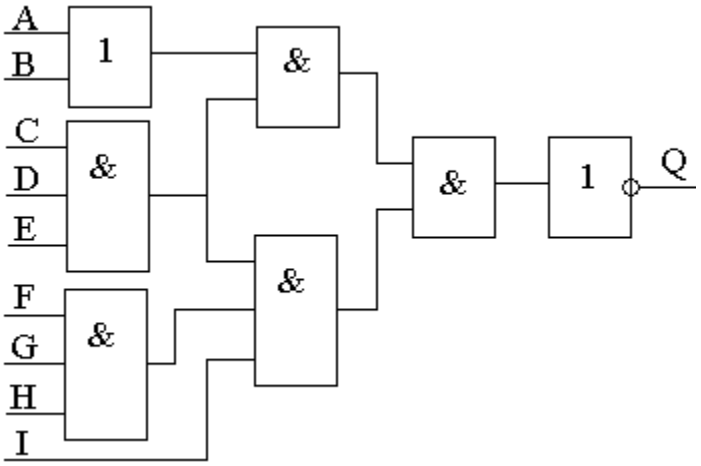
Завдання № 12

1. Структурний підхід щодо проектування ПЛІС.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



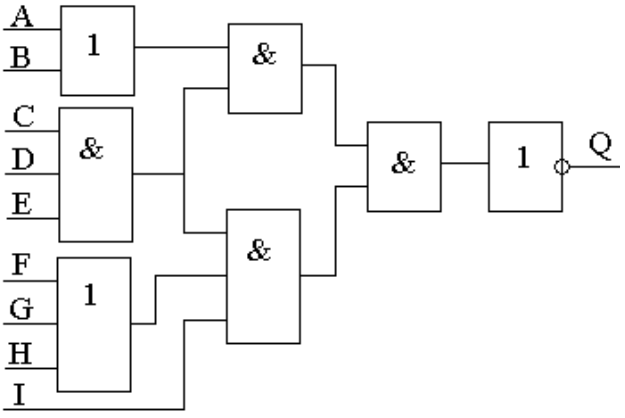
Завдання № 13

1. Принципи проектування «систем на кристалі».

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

D<= A and F;
C<= B or D;
A<= F or D;
B<= D and C and E and F;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



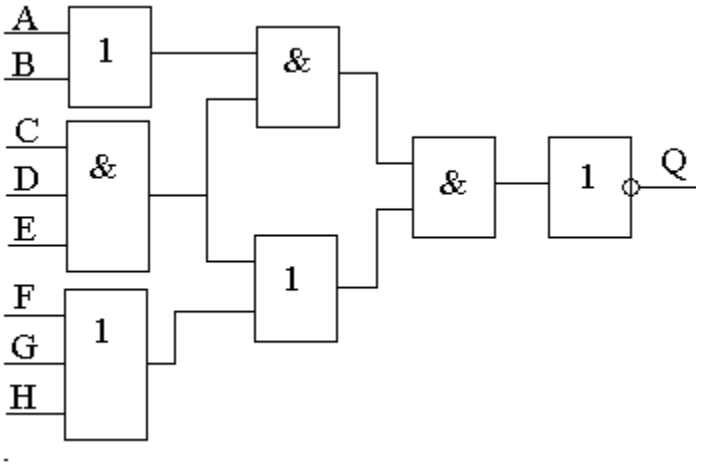
Завдання № 14

1. У чому полягають особливості проектування цифрових пристроїв на ПЛІС ?

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

D<= not B;
C<= B or D;
A<= F and D;
B<= D or C or E or F;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



Завдання № 15

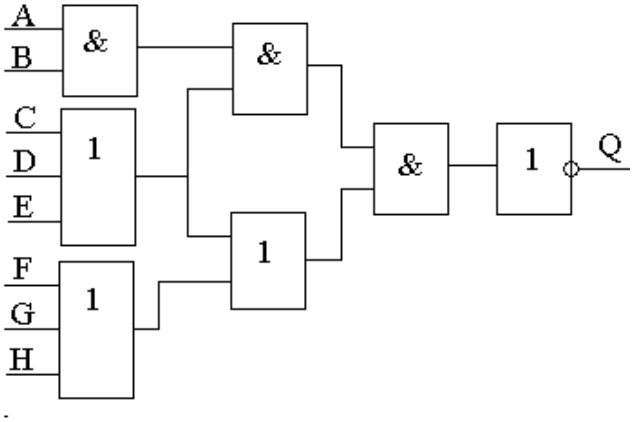
1. Навести основні переваги та недоліки способів схемного та текстового подавань цифрових пристроїв.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

B<= D and C;

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



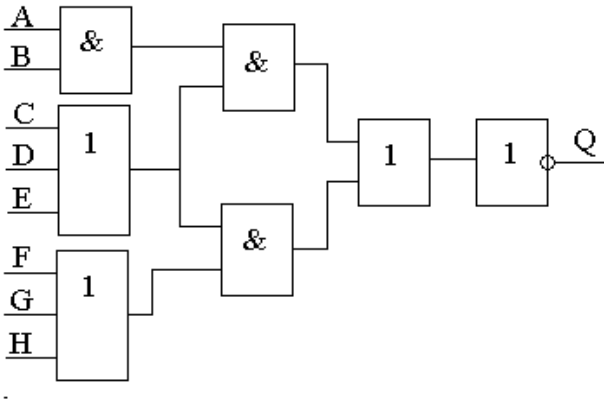
Завдання № 16

1. Сформулювати основні особливості мов опису апаратури.

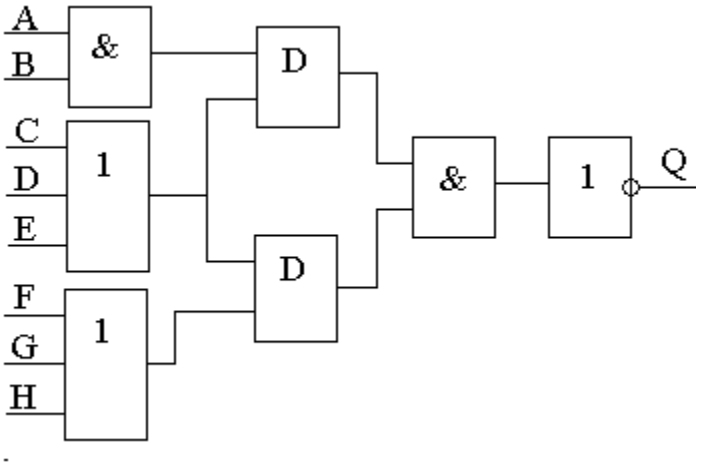
2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



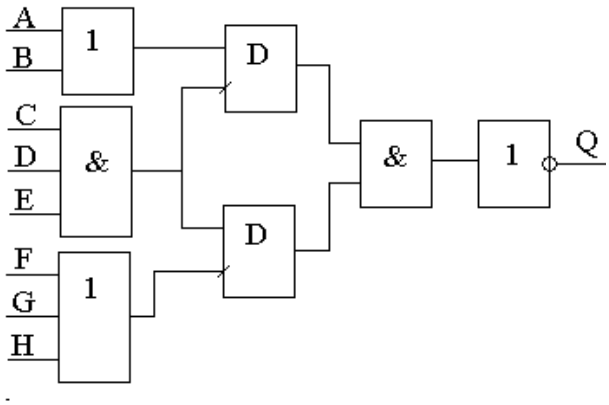
Завдання № 19

1. Основні оператори послідовного VHDL.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



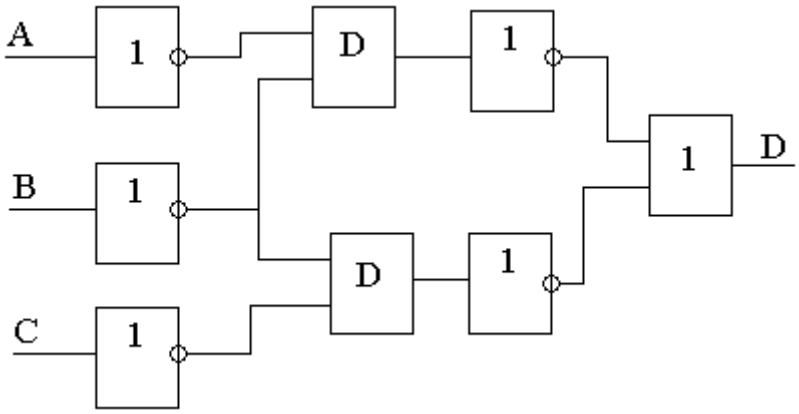
Завдання № 20

1. Основні види VHDL – описів щодо цифрових пристроїв.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



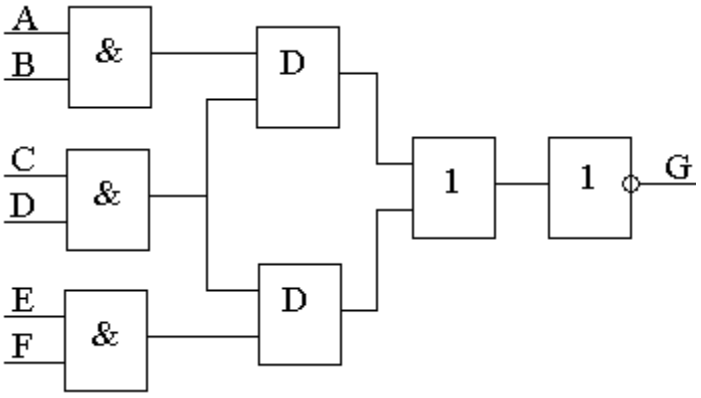
Завдання № 21

1. Поняття сигналу у мові VHDL.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



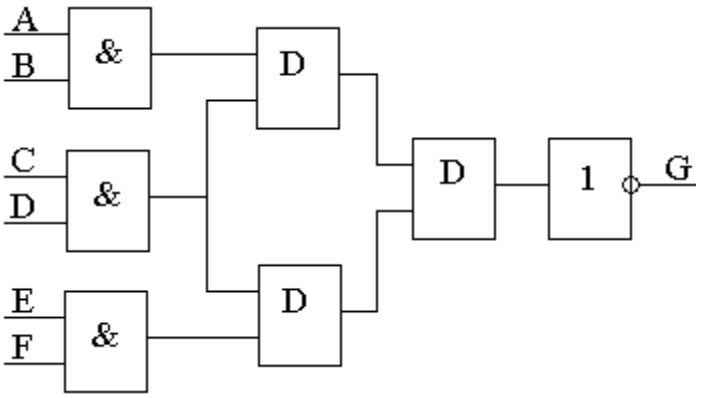
Завдання № 22

1. Структурний підхід щодо проектування ПЛІС.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

```
D<=not (A and F);
C<= B or D;
A<= F and D;
B<= not (D or C or E or F);
```

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



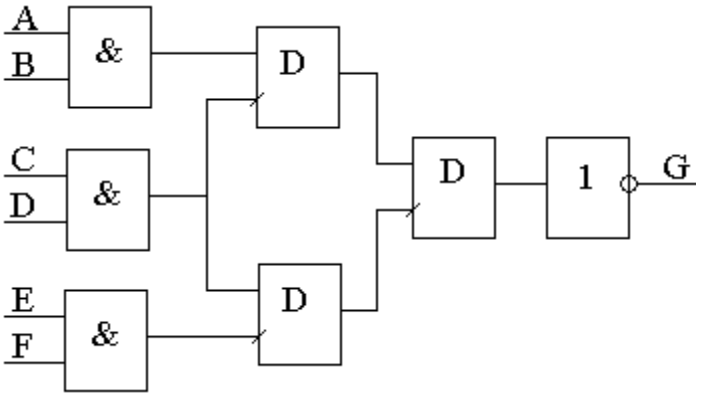
Завдання № 23

1. Як визначається логічна складність пристроїв на БМК ?

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



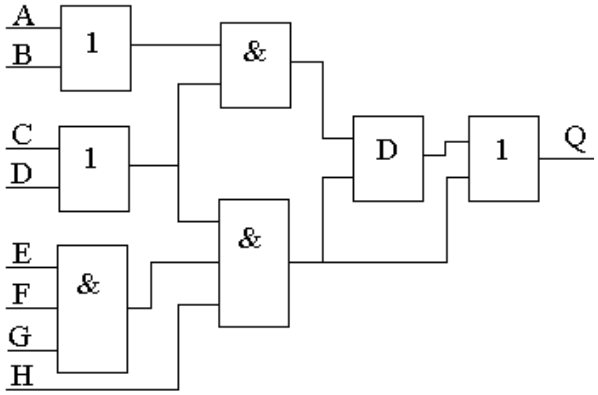
Завдання № 24

1. Пояснити принципи внутрішньої організації структур FPGA.

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:



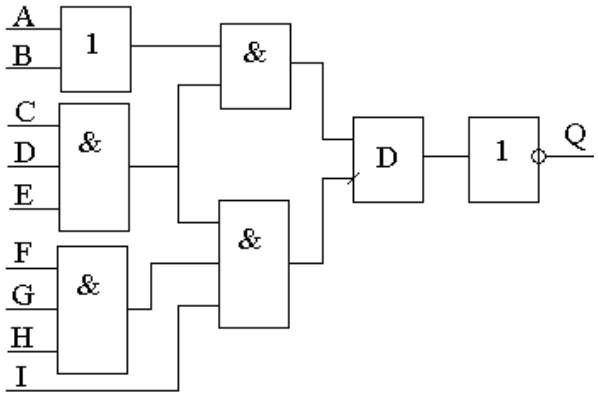
Завдання № 25

1. Яку функцію виконує логічна таблиця LUT у структурах FPGA ?

2. Виконати синтез схеми за описом фрагмента із паралельних операторів VHDL:

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

3. Провести структурний опис схеми на VHDL із описом архітектурних тіл компонентів:

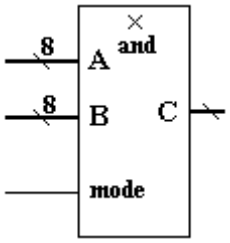


МОДУЛЬНІ КОНТРОЛЬНІ ЗАВДАННЯ
Модуль 2

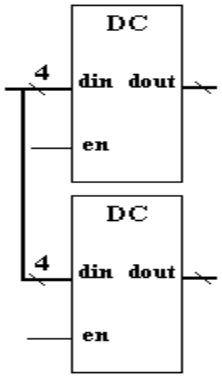
Завдання № 1

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу AFH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



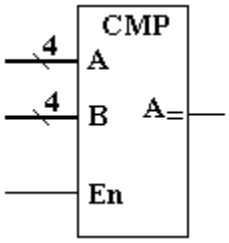
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



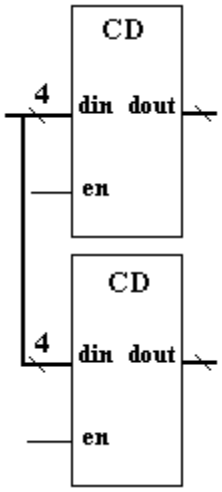
Завдання № 2

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу ACH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



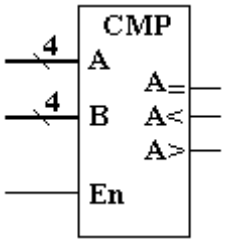
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



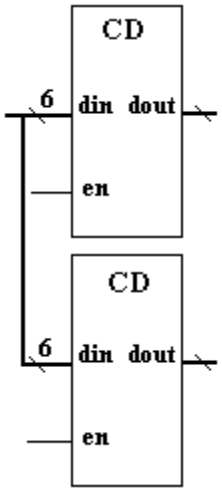
Завдання № 3

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 7FH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



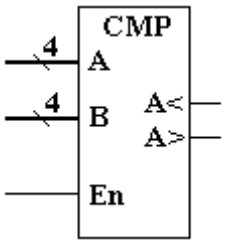
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



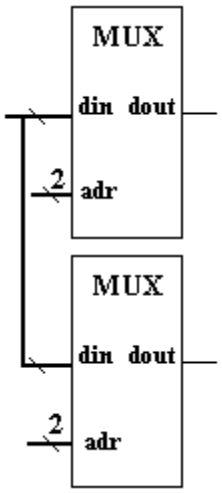
Завдання № 4

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 58H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



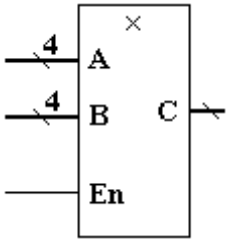
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



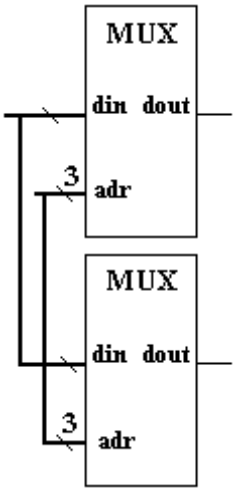
Завдання **№** 5

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 37H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми помножувача на поведінковому рівні.



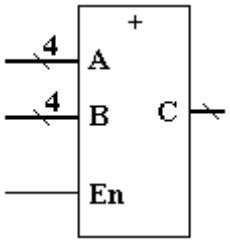
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



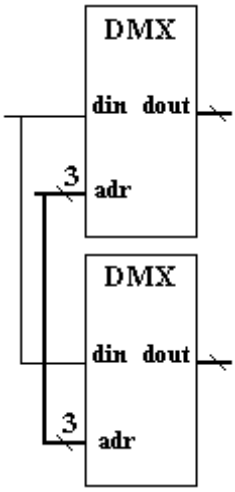
Завдання **№** 6

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 75H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



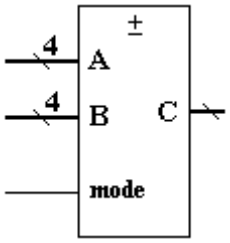
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



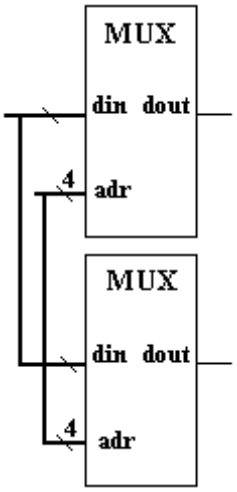
Завдання **№ 7**

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 77H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2**.** Виконати VHDL-опис схеми суматора на поведінковому рівні.



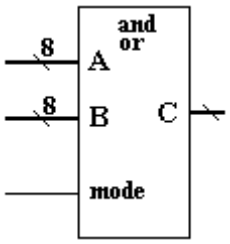
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



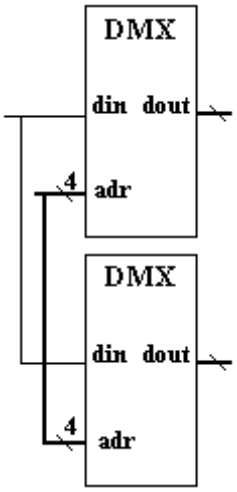
Завдання № 8

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу AAH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



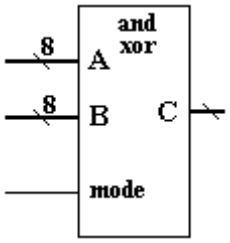
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



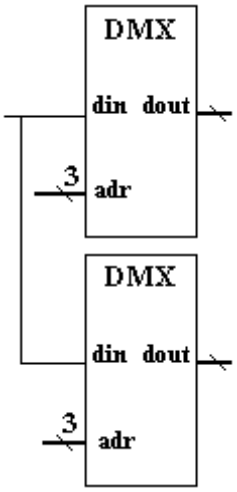
Завдання **№** 9

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу ВЗН та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



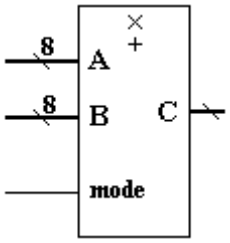
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



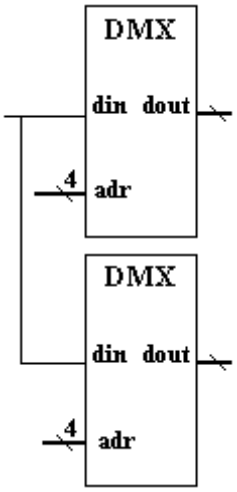
Завдання **№** 10

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу E4H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



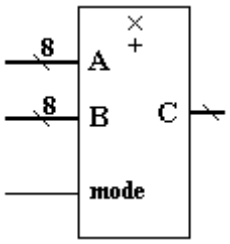
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



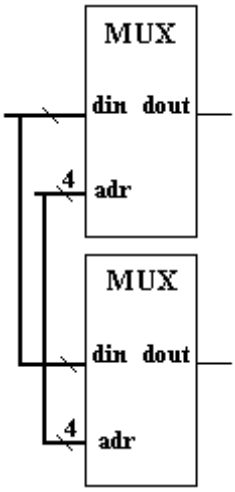
Завдання **№** 11

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу E4H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



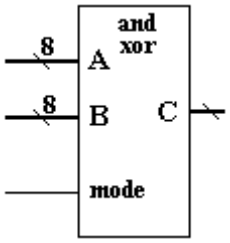
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



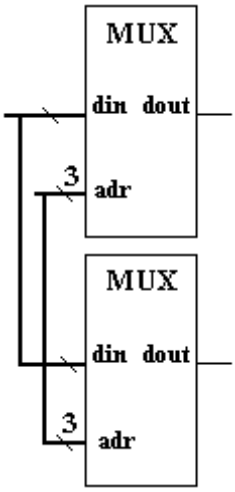
Завдання **№** 12

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу B3H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



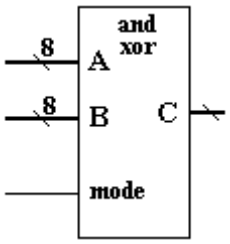
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



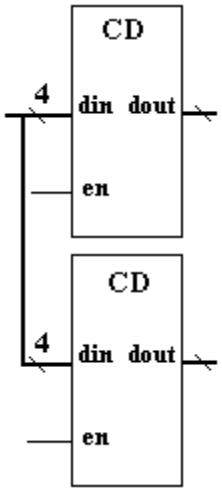
Завдання № 13

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу В3Н та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



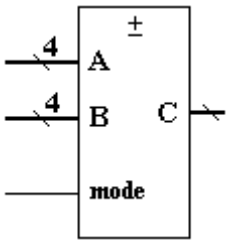
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



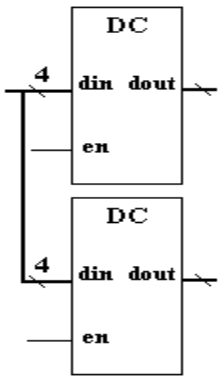
Завдання № 14

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 77H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми суматора на поведінковому рівні.



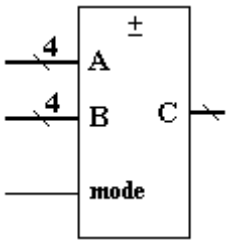
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



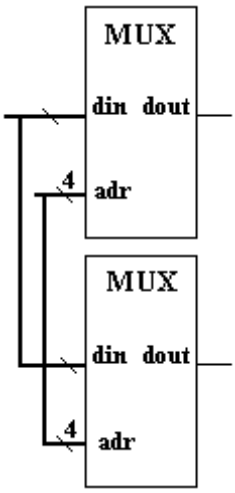
Завдання № 15

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 77H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми суматора на поведінковому рівні.



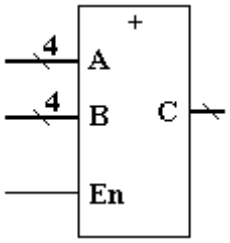
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



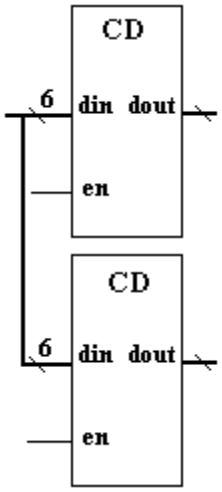
Завдання № 16

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 75H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми суматора на поведінковому рівні.



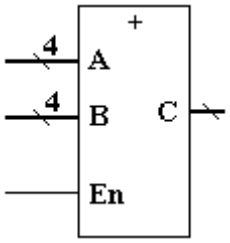
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



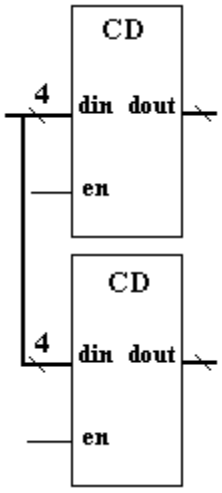
Завдання № 17

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу ВЗН та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми суматора на поведінковому рівні.



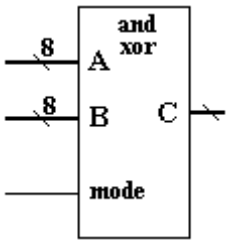
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



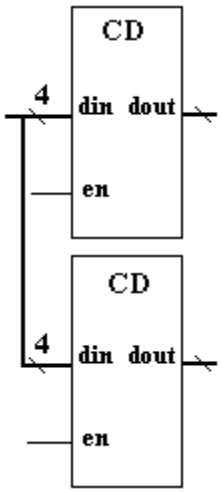
Завдання № 18

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу ВЗН та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



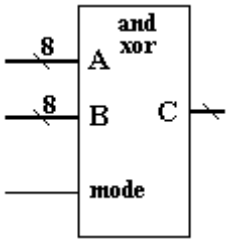
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



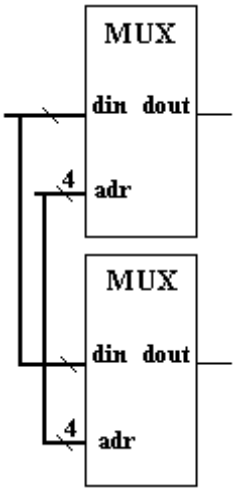
Завдання № 19

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу B3H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



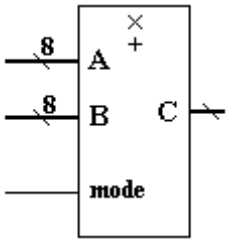
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



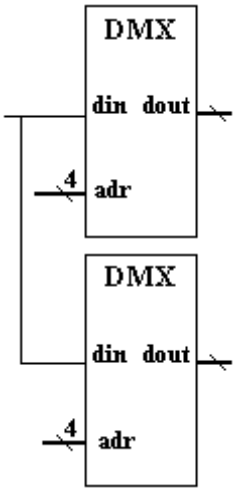
Завдання № 20

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу B5H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



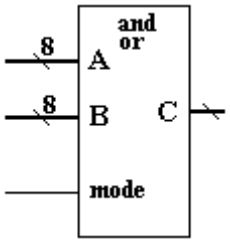
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



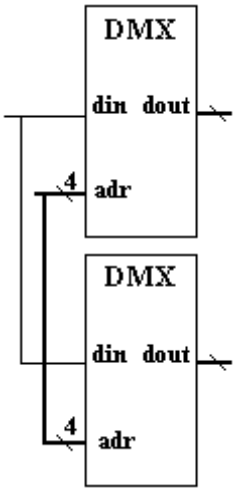
Завдання № 21

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу C7H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми пристрою для виконання логічних операцій на поведінковому рівні.



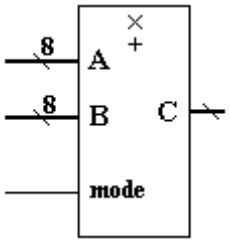
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



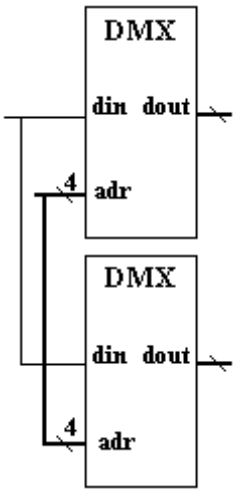
Завдання № 22

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу C7H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



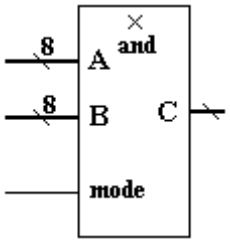
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



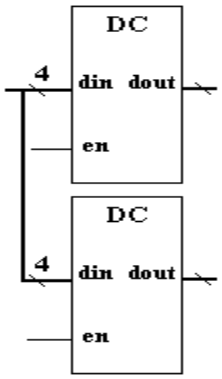
Завдання № 23

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу C3H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



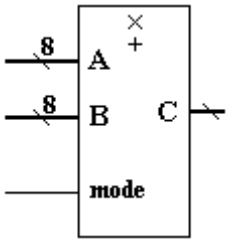
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



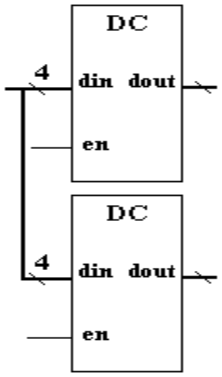
Завдання № 24

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу B7H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми АЛУ на поведінковому рівні.



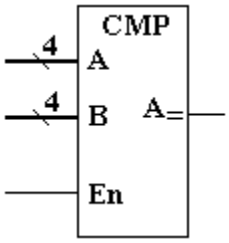
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



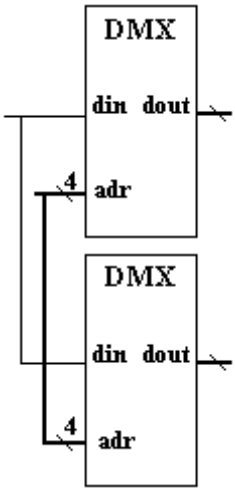
Завдання № 25

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу ADH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



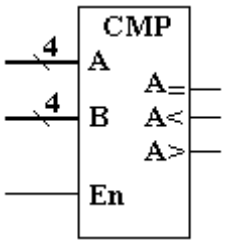
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



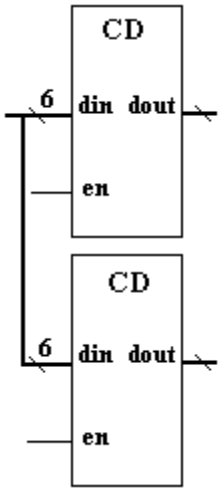
Завдання № 26

1. Виконати VHDL-опис схеми восьмирозрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу CFH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



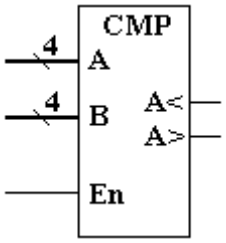
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



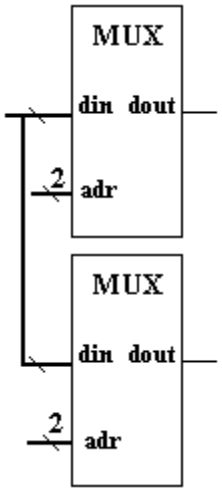
Завдання № 27

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу F8H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



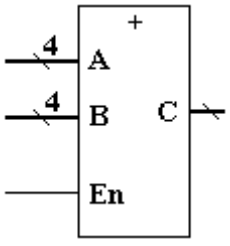
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



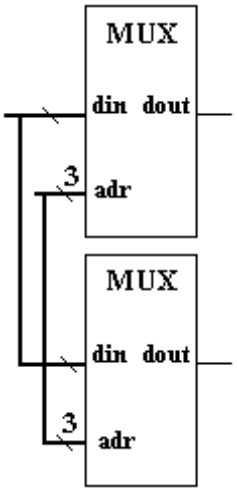
Завдання № 28

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 79H та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

Завдання № 2. Виконати VHDL-опис схеми компаратора на поведінковому рівні.



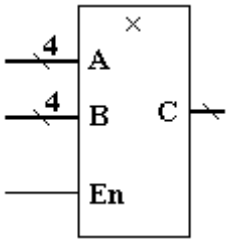
3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



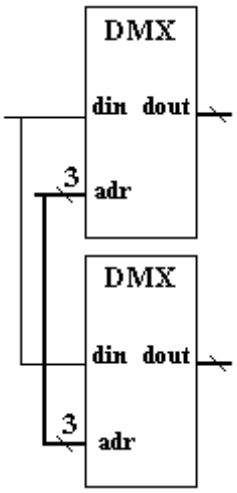
Завдання № 29

1. Виконати VHDL-опис схеми 16-розрядного інтерфейсного пристрою спряження ПЕОМ із зовнішнім пристроєм, який реагує на адресу 7FH та сигнали вводу RD і виводу WR даних; та відобразити структурну схему селектора адреси.

2. Виконати VHDL-опис схеми помножувача на поведінковому рівні.



3. Виконати VHDL-опис схеми на структурному рівні із описом архітектурних тіл компонентів.



ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Семенець В.В., Хаханова І.В., Хаханов В.І. Проектування цифрових систем з використанням мови VHDL. – Харків: ХНУРЕ, 2003. – 492 с.

2. Семенець В.В., Хаханов В.І. Проектування цифрових систем з Використання мови VHDL. – Харків: ХНУРЕ, 2002. – 192 с.

3. Бібіло П.М. Основи мови VHDL. – М.: СОЛОН-Р, 2003. – 294 с.

4. Бібіло П.М. Синтез логічних схем з використанням мови VHDL. – М.: СОЛОН-Р, 2002. – 384 с.

5. Сергієнко А.М. VHDL для проектування обчислювальних пристроїв. – К.: ПП "Корнійчук", ТОВ "ТИД ДС", 2003. – 208 с.

6. Е. А. Суворова, Ю. Є. Шейнин Проектування цифрових систем на VHDL BHV. - Санкт - Петербург, 2003. – 603 с.

7. Стешенко В.Б. Плис фірми ALTERA: елементна база, системи проектування і мови опису апаратури. – М .: Видавничий дім "Додека-XXI", 2002. – 576 с.

8. Угрюмов Є.П. Цифрова схемотехніка. – СПб .: БХВ-Петербург, 2002. – 528 с.

9. Бондаренко М.Ф., Семенец В.В., Белоус Н.В., Куцевич И.В., Белоус И.А. Оценивание тестовых заданий разных типов и определение их уровня сложности //Искусственный интеллект. – 2009. – №4. – С. 322 – 329.

10. Семенець В. Впровадження технологій дистанційного навчання у навчальний процес / В. В. Семенець, В. Каук, О. Аврунін // Вища школа. – 2009. – № 5. – С. 40 – 51.

11. Бондаренко М.Ф., Семенец В.В., Белоус Н.В., Куцевич И.В., Белоус И.А. Технология оценивания тестов в зависимости от типа и уровня сложности тестовых заданий на основе интегрированной модели // International Book Series "Information Science and Computing". – Sofia: Human Aspects of Artificial Intelligence. – 2009. – No:12. – С. 55 – 62.

12. Технические аспекты разработки виртуальных лабораторных работ по техническим дисциплинам / О.Г. Аврунин, О.Я.Крук, Т.В. Носова, В.В. Семенец // Открытое образование. – 2008. – №3(68). – С. 11 – 17.

13. Аврунин О.Г. Возможности разработки виртуальных лабораторных работ для изучения микропроцессорных систем/ О.Г.Аврунин, А.И.Бых, В.В. Семенец // Технічна електродинаміка, 2009, тем. випуск «Силова електроніка та енергоефективність». – Т. 1. – С. 109 – 112

14. Аврунін О.Г. Развитие современного инженерного мышления у студентов технических вузов / Матер. Міжвуз. наук.-практич. семінару “Філософія в сучасному світі”, 20-21.11.2014 р., Харків. – 2014. – С. 95-96

15. Современная лабораторная база для изучения микропроцессорных систем / О.Г. Аврунин, Т.В. Носова // Материалы 6-й Международной конференции «Стратегия качества в промышленности и обра-зовании» Днепропетровск-Варна. – 2010. – Т.2. – С.444 – 445.

16. Avrunin, O.G., Development of up-to-date laboratory base for microprocessor systems investigation / O.G., Avrunin, S.N., Sakalo, V.V. Semenetc

// KpbiMuKo 2009 CriMiCo – 2009 19th International Crimean Conference Microwave and Telecommunication Technology, Conference Proceedings. – 2009. – P. 301 – 302.

17. Методичні вказівки до лабораторних робіт з дисциплін «Автоматизація проектування електронних пристроїв», «Застосування ПЛІС під час проектування біомедичної апаратури» для студентів денної та заочної форм навчання спеціальностей: 7.090804 Фізична та біомедична електроніка, 7.091002 Біотехнічні та медичні апарати і системи / Упоряд. О.Г. Аврунін, О.Я. Крук, Т.В. Носова , В.В. Семенець. – Харків: ХНУРЕ, 2010. – 52 с.