

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни
"Технології проектування комп'ютерних систем"

для студентів спеціальностей 8.05010201 "Комп'ютерні системи
і мережі", 8.05010203 "Спеціалізовані комп'ютерні
системи" всіх форм навчання

Частина 2

2015

Методичні вказівки до лабораторних робіт з дисципліни “Технології проектування комп’ютерних систем” для студентів спеціальностей 8.05010201 ”Комп’ютерні системи і мережі”, 8.05010203 ”Спеціалізовані комп’ютерні системи” всіх форм навчання. Частина 2. /Укл. А.В. Сирота. - Запоріжжя: ЗНТУ, 2015. – 46 с.

Рекомендовано НМК як методичні вказівки до лабораторних робіт з дисципліни „Технології проектування комп’ютерних систем” усіх форм навчання.

Укладач: А.В. Сирота ст. викладач.

Рецензент: Р.К. Кудерметов, доцент, к.т.н.

Відповідальний за випуск: А.В. Сирота ст. викладач.

Затверджено
на засіданні кафедри КСМ

Протокол № 6
від 02.02.2015

ЗМІСТ

1 СЕРЕДОВИЩЕ САПР ACTIVE–HDL	4
1.1 Загальні відомості про Active-HDL	4
1.2 Створення нового проекту	8
1.3 Створення нового документа (введення тексту програми).....	11
1.4 Компіляція проекту	15
1.5 Моделювання	17
1.6 Порядок виконання роботи	22
1.7 Зміст звіту.....	23
1.8 Контрольні питання	23
2 МОВА ПРОГРАМУВАННЯ VHDL	25
2.1 Теоретична довідка	25
2.1.1 Опис об'єктів мовою VHDL	25
2.1.2 Типи даних і операції	30
2.1.3 Оператори мови VHDL	33
2.2 Порядок виконання роботи.....	36
2.3 Зміст звіту.....	36
2.4 Контрольні питання	36
3 ПРИКЛАД МОДЕЛЮВАННЯ СУМАТОРА.....	38
3.1 Створення проекту	38
3.2 Створення компоненту.....	39
3.3 Створення файлу верхнього рівня	42
ЛІТЕРАТУРА.....	46

Лабораторна Робота № 3

1 СЕРЕДОВИЩЕ САПР ACTIVE–HDL

Мета роботи. Вивчити структуру VHDL–проекту і функціональні можливості системи автоматизованого проектування (САПР) Aldec Active–HDL.

1.1 Загальні відомості про Active-HDL

САПР Aldec Active–HDL є інтерактивним інтегрованим середовищем розробки і функціонально–структурного моделювання проектів цифрових систем (ЦС) на HBIC програмованої логіки (HBIC ПЛ). Її основними компонентами є компілятори мов опису апаратури і засоби функціонального моделювання проектів. Вбудовані додаткові засоби – текстовий редактор з функцією перевірки синтаксису, редактор кінцевих автоматів, бібліотека конструкцій мови, розвинута система допомоги та ін. – дозволяють значно скоротити час проектування і моделювання. Підтримуються мови опису апаратури *VHDL*, *Verilog* та *EDIF* (*Electronic Design Interchange Format*). [9]

Можлива спільна робота Active–HDL і програмних засобів фірм–виробників HBIC ПЛ. Це дозволяє вести крізне проектування цифрової системи від початку до кінця: від тексту програми до готових виробів в базисі FPGA і CPLD.

Користувачу доступні групи засобів розробки VHDL–проекту (див. рис. 1.1):

- засоби введення проекту (*Design Entry Tools*);
- засоби керування проектом (*Control Tools*);
- засоби виведення результатів моделювання (*Simulation Output Viewers*);
- засоби відлагодження проекту (*Debugging Tools*);
- ядро системи та інструментальні засоби.

До **засобів введення проекту** належать функціональні одиниці САПР Active-HDL: текстовий редактор з перевіркою синтаксису (*HDL Editor*, *HDE*), графічний редактор логічних схем (*Block Diagram Editor*, *BDE*), графічний редактор кінцевих автоматів (*State Diagram Editor*, *FSM*), бібліотека конструкцій мови (*Language Assistant*), конвертер HDL–тексту в логічні схеми (*Code2Graphics*).

Засоби керування проектом: менеджер проектів (*Design Explorer*), менеджер файлів, ресурсів і структури проектів (*Design Browser*), менеджер керування напрямом проектування (*Design Flow Manager*), менеджер бібліотек (*Library Manager*), вікно команд і повідомлень (*Console*).

Функціональні одиниці, що є **засобами виведення результатів моделювання:** редактор часових діаграм (*Waveform Editor*), вікно таблиці перемикань (*List Viewer*).

До засобів відлагодження проекту належать: ядро системи моделювання (*Simulator Kernel*), вікно паралельних процесів (*Processes*), вікно сигналів користувача (*Watch*), стек викликів підпрограм (*Call stack*), вікно графічного відображення обміну інформацією усередині проекту (*Dataflow*).

Ядро системи та інструментальні засоби Active-HDL складають: генератор VHDL-проектів (*VHDL Generator*), генератор Verilog-проектів (*Verilog Generator*), генератор документів у форматі EDIF (*EDIF Generator*), компілятор мови VHDL (*VHDL Compiler*), компілятор мови Verilog (*Verilog Compiler*), компілятор EDIF (*EDIF Compiler*).

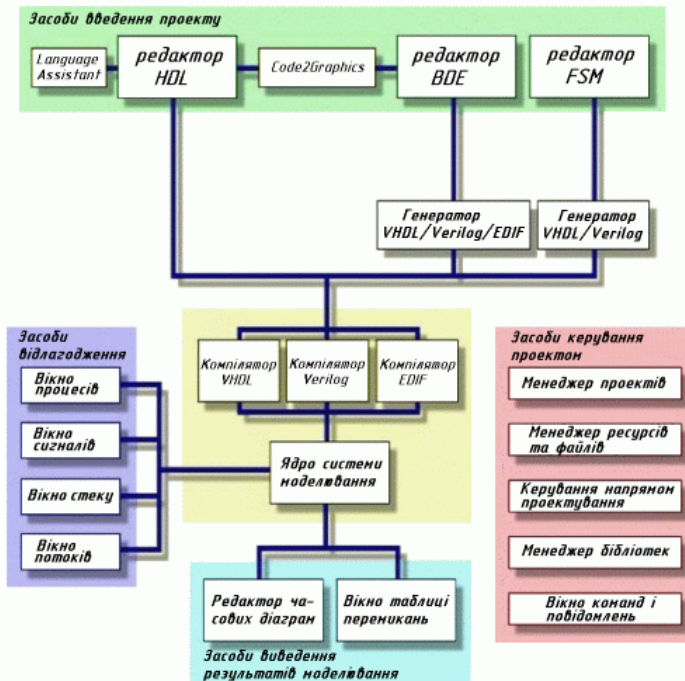


Рисунок 1.1 – Організація САПР Aldec Active-HDL

Часто буває достатньо скласти опис поведінки цифрової системи, алгоритму її роботи. З погляду автоматизації проектування ЦС, якщо не визначений базис її реалізації, такий підхід є більш раціональним.

Будь-який об'єкт може мати декілька варіантів алгоритмічного опису архітектури. Різні за формою (синтаксисом), але єдині за значенням (семантикою) представлення одного і того ж алгоритму можуть задавати однакову

поведінку об'єкту, але мати різну структурну реалізацію.

Можливість за допомогою **мови опису апаратури** (HDL) різними способами задавати один і той же алгоритм дозволяє за допомогою САПР Active–HDL автоматично синтезувати відмінні за складністю (рівнем деталізації) функціонально–структурні описи ЦС. Перевага, яку дає використання САПР Active–HDL, полягає в тому, що користувач, за допомогою конструкцій мови опису апаратури, сам указує рівень деталізації моделі ЦС.

Відповідно до цього складається **VHDL–проект** – формальний опис поведінки ЦС на мові VHDL. Користувач сам вибирає рівень деталізації опису цифрової системи (абстракції) з чотирьох можливих:

- **поведінковий рівень** (опис алгоритму роботи ЦС);
- **рівень регістрових передач** (рівень RTL);
- **функціонально–логічний рівень** (враховуються часові затримки розпосюдження сигналів);
- **рівень логічних елементів** (враховується час спрацювання логічних елементів).

VHDL–проект складається з **логічних файлів** (*Design Files*) опису компонентів ЦС, з'єднаних згідно формально визначеного закону функціонування ЦС. Логічні файли організовуються відповідно до ієрархії опису проєктованої цифрової системи. Один з логічних файлів VHDL–проекту є файлом верхнього рівня ієрархії описів (*Top–level Design File*). Він грає роль організатора, що визначає архітектуру, склад ЦС і алгоритм її поведінки. Логічні файли нижніх рівнів ієрархії (*Low–level Design files*) містять детальний опис блоків і модулів ЦС. Ім'я проєкту завжди співпадає з ім'ям логічного файлу верхнього рівня ієрархії проєкту. До складу VHDL–проекту можуть бути залучені різноманітні допоміжні файли (*Ancillary Files*).

В САПР Active–HDL кожний проєкт зберігається в окремому каталозі, що зветься **каталогом проєкту**. Проєкт і каталог, де він зберігається, мають однакові імена. Кожний каталог проєкту містить файл опису проєкту (розширення .ADF) і підкаталоги *SRC*, *GENERIC* і ін. Підкаталог *SRC* містить текстові файли мовою VHDL (розширення .VHD), файли діаграм автоматів (розширення .ASF) тощо. Підкаталог *GENERIC* містить робочі файли, які використовуються при компіляції і моделюванні.

Бібліотеки Active–HDL діляться на два класи: системні бібліотеки і бібліотеки проєктів.

Системні бібліотеки поставляються разом з САПР Active–HDL і знаходяться в каталозі *VLIB*.

Бібліотеки проєктів автоматично створюються окремо для кожного проєкту і зберігаються в підкаталозі *GENERIC* проєкту.

Кожна бібліотека складається з двох файлів: бібліотечного індексного файлу (розширення .INI) і головного бібліотечного файлу (розширення

.MFG), що містить дані, що використовуються середовищем Active-HDL. В підкаталозі VLIB присутній також бібліотечний файл конфігурації *LIBRARY.CFG*, загальний для всіх бібліотек.

Системні бібліотеки з каталогу VLIB наперед скомпільовані і готові до вживання у VHDL-програмах у вигляді *пакетів* (packages). Деякі вбудовані стандартні пакети (табл. 1.1) містять підпрограми, які використовуються ядром системи функціонального моделювання (Simulator Kernel).

Таблиця 1.1 – Вбудовані стандартні пакети САПР Active-HDL

Пакет	Призначення
STD.STANDARD	Стандартний пакет VHDL загального призначення (стандарт IEEE Std. 1076)
STD.TEXTIO	Стандартний пакет VHDL текстових операцій (стандарт IEEE Std. 1076)
IEEE.STD_LOGIC_1164	Підпрограми багаторівневої логіки (стандарт IEEE Std. 1164)
IEEE.VITAL_TIMING	Примітиви VITAL'95 (стандарт IEEE Std. 1076.4)
IEEE.VITAL_PRIMITIVES	Примітиви VITAL'95 (стандарт IEEE Std. 1076.4)
IEEE.STD_LOGIC_ARITH	Математичні підпрограми (стандарт IEEE Std. 1076.2)
IEEE.STD_LOGIC_SIGNED	Підпрограми IEEE (стандарт IEEE Std. 1076.5)
IEEE.STD_LOGIC_UNSIGNED	Підпрограми IEEE (стандарт IEEE Std. 1076.5)

Під час установки середовища Active-HDL пропонується підключити бібліотеки, що призначені для підтримки розробки проектів ЦС в базисі HBIC ПЛ ведучих виробників (Actel, Altera, Atmel, Cypress, Lattice, Lucent, Quicklogic, Synopsys і Xilinx). САПР Active-HDL включає також бібліотеку вбудованих примітивів оптимізації часу моделювання SIMPRIM.

Разом з середовищем поставляється докладна контекстна довідка та електронні підручники. В табл. 1.2 приведений опис основних файлів **довідкової системи** Active-HDL. Великий обсяг довідкових матеріалів зосереджено в навчальній системі Enhanced VHDL Tutorial (EVITA). Її можна викликати з САПР Active-HDL за допомогою пункту меню *Help* → *Interactive VHDL Tutorial*. В Інтернеті мається електронна сторінка, що містить переклади деяких довідкових матеріалів – www.aldec.com.ua. Електронна адреса фірми Aldec – www.aldec.com.

Таблиця 1.2 – Довідкова система САПР Active–HDL

Короткий опис	Ім'я файлу
Загальні відомості про САПР Active–HDL	RELNOTES
Докладна інформація про склад і призначення САПР Active–HDL і її підсистем	AVHDL
Керівництво по створенню і моделюванню цифрових систем в Active–HDL	TUTAVHDL
Керівництво проектування за допомогою редактора BDE САПР Active–HDL	TUTBDE
Керівництво проектування за допомогою редактора FSM САПР Active–HDL	TUTFSM
Довідник з мови VHDL	TUTVHDL
Опис засобів верифікації VHDL–проектів САПР Active–HDL	TUTTBCH

Розглянемо докладніше процес розробки VHDL–проекту нескладної ЦС в САПР Active–HDL. Наприклад треба виконати функціональне моделювання поведінки демультимплексора DM(3), таблиця дійсності якого наведена в табл. 1.3.

Таблиця 1.3 – Таблиця дійсності демультимплексору DM(3)

Входи				Виходи							
w	x(2)	x(1)	x(0)	y(7)	y(6)	y(5)	y(4)	y(3)	y(2)	y(1)	y(0)
0	X	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

1.2 Створення нового проекту

- 1.2.1 Клацнувши на піктограмі на Робочому Столі Windows, запустить середу Active–HDL 6.1.
- 1.2.2 На панелі інструментів Active–HDL клацніть піктограму *New Workspace* (створити нове робоче середовище).

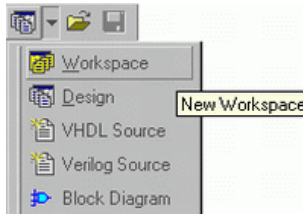


Рисунок 1.2 – Створення робочого середовища

1.2.3 З'явиться вікно **New Workspace**. В полі "Type the workspace name" введіть ім'я робочого середовища. Залиште активною опцію "Add New Design to Workspace" (добавити новий проект до робочого середовища). Клацніть **OK**.

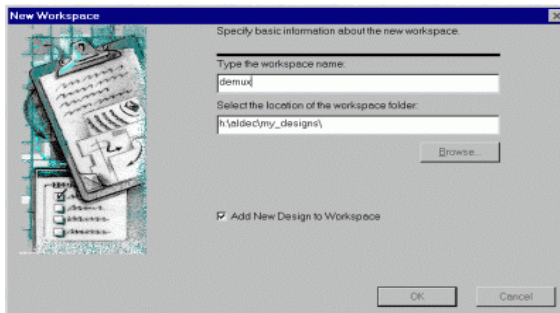


Рисунок 1.3 – Вікно New Workspace

1.2.4 Запускається майстер створення проекту **New Design Wizard** (рис. 1.4).

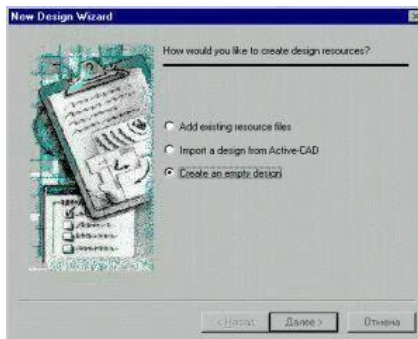


Рисунок 1.4 – New Design Wizard. Вибір режиму

- 1.2.5 Налаштування параметрів проекту виконується за чотири кроки. Виберіть режим *Create an empty design* (створити порожній проект). Клацніть кнопку **Next**.
- 1.2.6 Друге вікно (другий крок) майстра призначено для вибору інструменту синтезу і імплементації. Необхідно також вказати тип НВІС ПЛІ, конфігурацію редактора BDE і мову опису проекту. В полі *Default HDL Language* виберіть **VHDL**.
- 1.2.7 Третій крок – визначення імені проекту, місця на диску для його зберігання і, якщо необхідно, настройка імені бібліотеки проекту. За умовчанням ім'я проекту є ім'ям його бібліотеки (рис. 1.5). В полі *Type the design name* наберіть *Lab1* і клацніть кнопку **Next**.

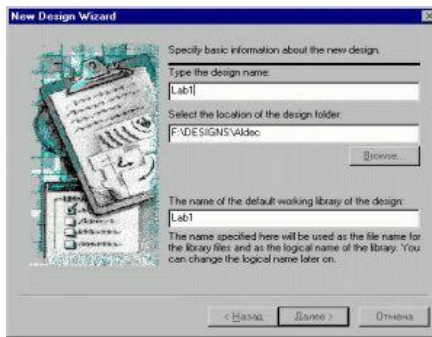


Рисунок 1.5 – New Design Wizard. Вибір імені проекту

- 1.2.8 Знов створеному проекту привласнено ім'я **Lab1**. В кінці роботи майстер створення проекту виводить інформацію про результати роботи. Файлу проекту має ім'я *Lab1.adf*. Клацніть кнопку **Done**.



Рисунок 1.6 – Кнопка і вікно Console. Повідомлення користувача

На диску з'явиться каталог VHDL-проекту *Lab1*. Структура цього каталогу відображається у вікні **Design Browser**. Коли в проект *Lab1* будуть

вноситись зміни, вони торкатимуться змісту файлів і директорій каталогу Lab1. Файл проекту *Lab1.adf* містить загальний опис проекту, компілятора і бібліотеки *Lab1.LIB*. У файл *console.log* будуть заноситися повідомлення про поточний стан проекту. Отримати доступ до повідомлень консолі можна у будь-який час натисненням на кнопку *Console*  панелі інструментів або за допомогою комбінації клавіш <ALT+0> (рис. 1.6).

1.3 Створення нового документа (введення тексту програми)

1.3.1 Натисніть кнопку *Design Flow*  панелі інструментів (комбінація клавіш <Alt+3>). Відкриється вікно управління проектом (рис. 1.7).

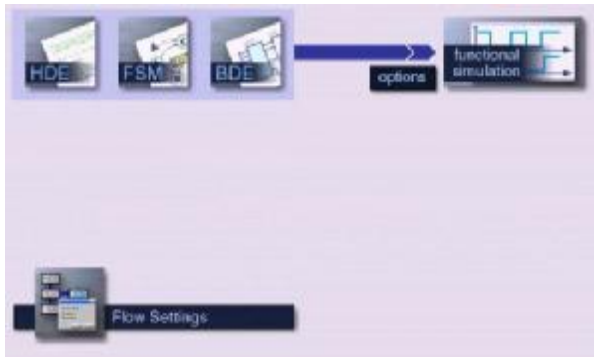


Рисунок 1.7 – Кнопка і вікно Design Flow. Управління проектом

1.3.2 Нам потрібен текстовий редактор. Для цього треба клацнути на піктограмі *HDE*. З'явиться вікно вибору мови опису апаратури (рис. 1.8). Виберіть **VHDL**.



Рисунок 1.8 – HDL Editor. Вибір шаблону опису апаратури

- 1.3.3 З'явиться вікно **New Source File Wizard**. Цей майстер автоматично генерує частину тексту програми на мові VHDL – опис портів в модулі *entity*. На першому кроці пропонується включити створюваний файл в поточний проект (*Add generated file to design*). Клацніть кнопку **Next**.
- 1.3.4 На другому кроці майстер пропонує дати імена VHDL-файлу, сутності і опису архітектури (*architecture*) проекту, що генерується (рис. 1.9). Текстовий файл тут названий *Lab1*, entity – *demux_3*, а опис архітектури має ім'я *beh*. Для продовження клацніть **Next**.

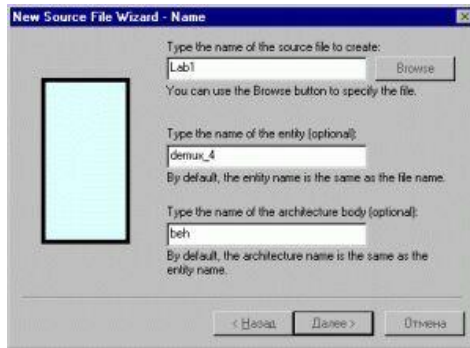


Рисунок 1.9 – New Source File Wizard. Привласнення імен

- 1.3.5 Третій крок найвідповідальніший. Тут задаються порти, їх типи і режими роботи. Демультіплексор DM(3) повинен мати один інформаційний вхід *w*, три входи керування *X* і вісім виходів *Y* (рис. 1.10).

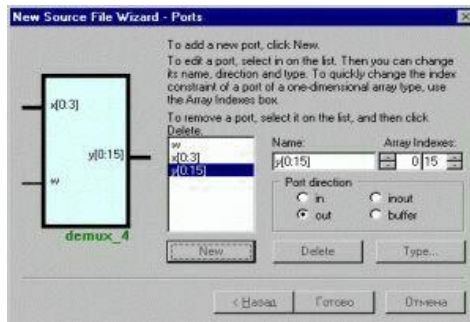


Рисунок 1.10 – New Source File Wizard. Завдання портів

1.3.6 Список портів редагується за допомогою кнопок **New** і **Delete**. Для створення нового порту натисніть кнопку **New**. В полі *Name* введіть його ім'я. Для вибору напрямку порту використовується група параметрів *Port direction*. Для вибору типу порту треба клацнути кнопку **Type** і відкрити вікно **Port Type**. По умовчанням задаються типи *STD_LOGIS* або *STD_LOGIS_VECTOR*. В мові VHDL є чотири режиму роботи портів: **in** – вхідний порт; **out** – вихідний порт; **inout** – двонаправлений порт; **buffer** – порт-буфер. Параметр *Array Indexes* використовується для введенню діапазонів масивів. Опис портів демультимплексору DM(3) має такий вигляд:

```
x: in STD_LOGIS_VECTOR (0 to 2);
w: in STD_LOGIS;
y: out STD_LOGIS_VECTOR (0 to 7);
```

1.3.7 На цьому робота з майстром шаблонів VHDL-програм **New Source File Wizard** закінчується. Клацніть **Done**. Буде створений шаблон VHDL-програми – моделі демультимплексора, в каталозі проекту з'явиться новий файл *Lab1.vhd* і відкриється вікно текстового редактора HDE.

Листинг 1.1 – Текст шаблону програми на мові VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity demux_3 is
    port(
        x : in BIT_VECTOR(0 to 2);
        w : in BIT;
        y : out BIT_VECTOR(0 to 7)
    );
end demux_3;
architecture beh of demux_3 is
begin
    -- enter your statements here --
end beh;
```

Шаблон VHDL-програми складається з трьох основних частин:

- перелік стандартних бібліотек, що використовуються (модуль *library*);
- повністю оформлений опис об'єкту (модуль *entity*);
- порожні операторні дужки опису архітектури (модуль *architecture*).

Модуль **library** визначає імена бібліотек проекту. VHDL-бібліотека може містити пакети, об'єкти, описи архітектури і конфігурації. В даному випадку використовується пакет *std_logic_1164* стандартної бібліотеки *IEEE*.

В модулі **entity** оголошується ім'я сутності (*demux_4*), описується інтерфейс між сутністю і середовищем, в якому вона функціонує. Тут визнача-


```

        then y <= "00000000";
    else
        L2: case x is
            when "000" => y <= "10000000";
            when "100" => y <= "01000000";
            when "010" => y <= "00100000";
            when "110" => y <= "00010000";
            when "001" => y <= "00001000";
            when "101" => y <= "00000100";
            when "011" => y <= "00000010";
            when "111" => y <= "00000001";
        end case L2;
    end if L1;
end process G0;
end beh;

```

Листинг 1.3 – Другий варіант програми

```
-- -- -- -- -- -- -- -- модуль library
library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- -- -- -- -- -- -- -- модуль entity
entity demux_3 is
    port(
        x : in STD_LOGIC_VECTOR(0 to 2);
        w : in STD_LOGIC;
        y : out STD_LOGIC_VECTOR(0 to 7)
    );
end demux_3;

-- -- -- -- -- -- -- -- модуль architecture
architecture beh of demux_3 is
begin
y(7)<= w and      x(2) and      x(1) and      x(0) after 20 ns;
y(6)<= w and      x(2) and      x(1) and not   x(0) after 12 ns;
y(5)<= w and      x(2) and not   x(1) and      x(0) after 24 ns;
y(4)<= w and      x(2) and not   x(1) and not   x(0) after 18 ns;
y(3)<= w and not   x(2) and      x(1) and      x(0) after 15 ns;
y(2)<= w and not   x(2) and      x(1) and not   x(0) after 16 ns;
y(1)<= w and not   x(2) and not   x(1) and      x(0) after 21 ns;
y(0)<= w and not   x(2) and not   x(1) and not   x(0) after 14 ns;
end beh;
```

1.4 Компіляція проекту

1.4.1 Виконаємо компіляцію нашого проекту. Для цього можна натискувати клавішу <F11> або кнопку *Compile*  панелі інструментів. На рис. 1.12 показані кнопки запуску компіляції і результат компіляції нашого VHDL-проекту. Якщо при складанні програми були допущені помилки, повідомлення про них можна побачити у вікні **Console**.

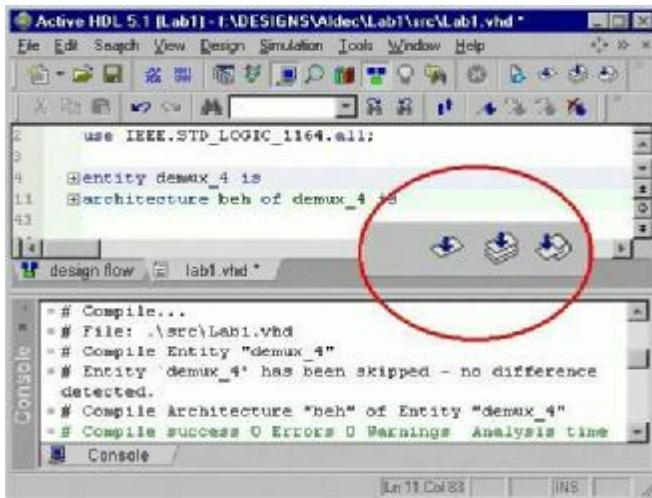


Рисунок 1.12 – Компіляція VHDL-проєкту

Іноді виникають помилки компіляції, що обумовлені неправильним використанням бібліотек мови VHDL. Для виправлення такої помилки можна використати компонент **Library Manager** (рис. 1.13). Цей засіб дозволяє отримати доступ до раніше розроблених пакетів і бібліотек VHDL, у тому числі і до бібліотек інших САПР.



Рисунок 1.13 – Library Manager. Вікна і елементи керування

Для того, щоб підключити до проекту необхідну бібліотеку, треба:

1. На панелі інструментів САПР Active–HDL клацнути кнопку *Library Manager*. З'явиться вікно менеджера бібліотек (рис. 1.13).
2. На панелі інструментів вікна **Library Manager** натискувати кнопку *Attach library* і в діалоговому вікні, що з'явилося, вказати шлях до файлу бібліотеки.

1.5 Моделювання

Для того, щоб перевірити, чи правильно працює синтезований засобами САПР Active–HDL опис поведінки демультимплексора DM(3), виконаємо функціональне моделювання.

Перш, ніж приступити до верифікації VHDL–проекту, необхідно скласти **тестову послідовність** вхідних сигналів w і $x[0:2]$. В найпростішому випадку тестова послідовність – це **вектор**, що містить всі можливі комбінації вхідних сигналів. Для демультимплексора DM(3) це $2^4=16$ векторів $[w, x_0, x_1, x_2]$.

- 1.5.1 Виберіть команду *File ® New ® waveform* або піктограму *New waveform* . З'явиться вікно редактора часових діаграм (рис. 1.14).

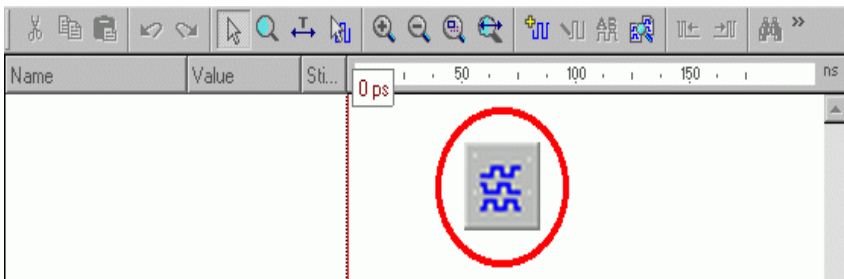


Рисунок 1.14 – Кнопка New waveform

- 1.5.2 Виконайте ініціалізацію моделювання. В головному меню Active–HDL виберіть команду *Simulation ® Initialize Simulation*. Після ініціалізації стане активною сторінка **Structure** у вікні **Design Browser**.
- 1.5.3 Додамо в редактор часових діаграм сигнали, що визначені в моделі *demux_4*. Клацніть **правою** кнопкою в лівій частині вікна редактора. В контекстному меню виберіть команду *Add Signals*. З'явиться вікно **Add Signals** (рис. 1.15).

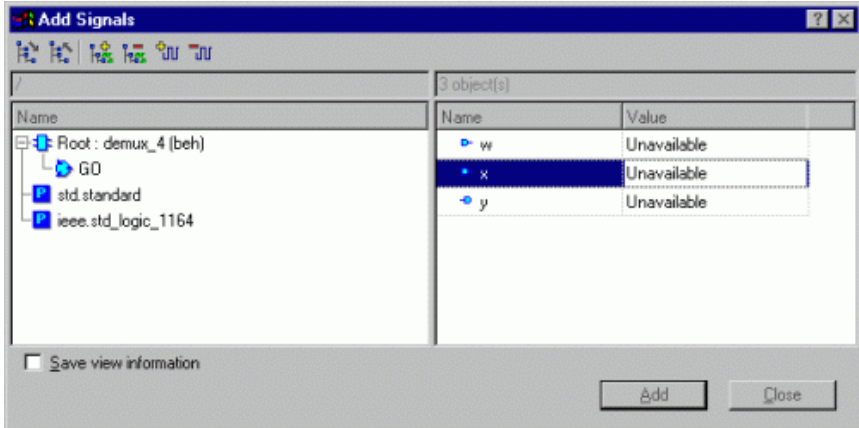


Рисунок 1.15 – Вікно Add Signals

1.5.4 Вибираючи по черзі сигнали **w**, **x**, **y** і клацаючи на кнопці **Add**, додайте ці сигнали у вікно редактора часових діаграм.

Це ж саме можна зробити інакше. Клацніть по імені компонента у верх - ній частині вікна **Design Browser** на вкладці **Structure**, внизу вікна мож - на побачити всі сигнали вибраного модуля. Перетягніть потрібні сигнала - ли в вікно **Waveform**.

1.5.5 Тепер можна задати **стимулятори** моделювання. Щоб створити стиму - лятор треба виділити потрібний сигнал в редакторі часових діаграм, клацнути правою кнопкою і в контекстному меню вибрати команду **Stimulators** (рис. 1.16) .

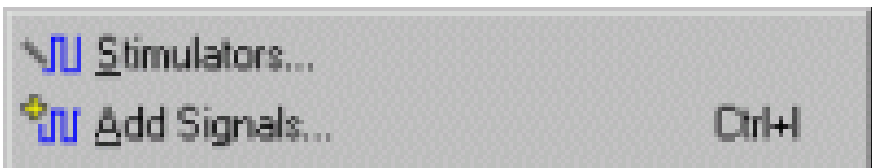


Рисунок 1.16 – Команди Stimulators и Add Signals

Є декілька типів стимуляторів:



Значення (Value). Дозволяє вибрати значення, що буде надане виділеному сигналу чи групі сигналів, об'єднаних у шину.

Формула (Formula). Дозволяє задати часову діаграму за допомогою формули. Наприклад формула 0 0, 1 12 означає, що сигнал має значення '0' у момент часу 0 і змінюється на '1' після 12 пікосекунд.

Гаряча клавіша (Hotkey). Дозволяє призначити сигналові гарячу клавішу, натискання якої призводитиме до зміни значення сигналу. За замовчуванням сигнал переключатиметься між 0 і 1, проте у цей список можна додати і інші допустимі для даного типу сигналу значення. При натисканні гарячої клавіші сигнал набуватиме наступного значення, включеного у список.

Годинник (Clock). Дозволяє задати періодичний сигнал з визначеними параметрами (частотою, періодом тощо), користуючись зручним графічним редактором.

Лічильник (Counter). Дозволяє моделювати сигнали на виходах лічильника як послідовність чисел в заданому напрямку лічби і в заданому базисі (двійковий код, код Грея та ін.).

Наперед визначений (Predefined). Це стимулятори синхросигналів або формульні, які мають власні імена, що дозволяє назначати їх декільком сигналам, не повторюючи кожен раз вводу параметрів.

Стимулятор, що настраюється користувачем (Custom). Дозволяє вибрати наперед визначений стимулюючий сигнал з заданого переліку та поповнити цей перелік власними сигналами, що часто використовуються

1.5.6 Задамо стимулятор для сигналу **w**. Виберіть тип стимулятора **Hotkey** (рис. 1.17). Встановіть курсор в полі *Press new hotkey* і натисніть клавішу <w> клавіатури. Її будемо використовувати для перемикання рівня сигналу **w**.

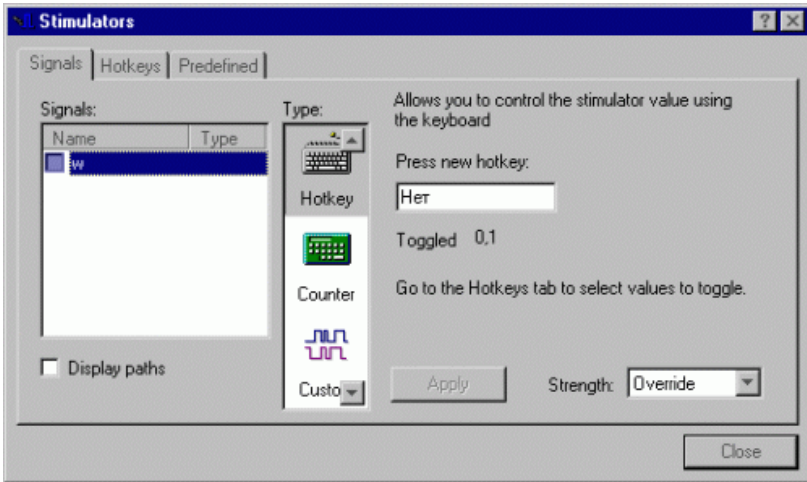


Рисунок 1.17 – Вікно Stimulators. Тип стимулятора – Hotkey

1.5.7 Задамо стимулятор для вектору сигналів **x[0..2]**. Для перебору всіх комбінацій вектора сигналів зручно використовувати стимулятор типу **Counter**. Виділіть вектор **x** і встановіть тип стимулятора **Counter** (рис. 1.18).

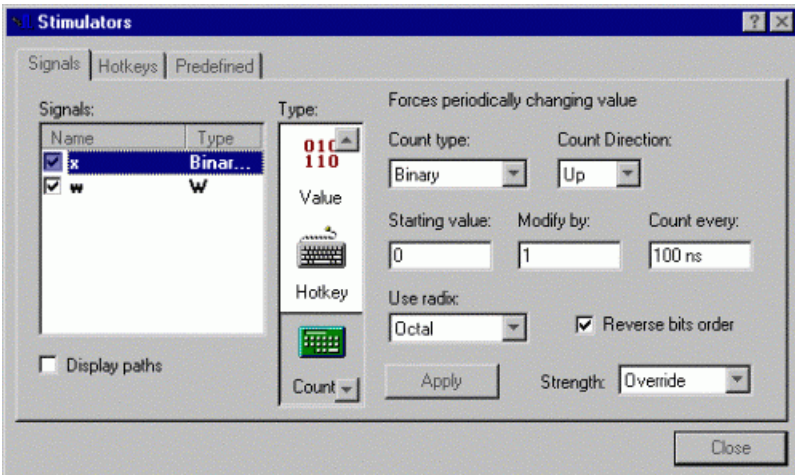


Рисунок 1.18 – Вікно Stimulators. Тип стимулятора – Counter

1.5.8 Встановіть настройки лічильника відповідно до табл. 1.4 (рис. 1.18). Для того, щоб рахунок проводився в зворотному порядку, треба встановити прапорець *Reverse bits order*.

Таблиця 1.4 – Настройки стимулятора для вектора $x[0..2]$

Настройка		Значення
Count type	Тип лічильника	Binary
Count direction	Напрямок рахунку	Up
Starting value	Початкове значення	0
Modify	Величина зміни	1
Use radix	Підстава	Hexadecimal
Count every	Період рахунку	100 ns

- 1.5.9 Перевіримо, як веде себе модель *demux_3*, якщо сигнал **w** має високий логічний рівень. Натисніть клавішу <w> ("гарячу" клавішу, що зв'язана з сигналом **w**). Натискаючи клавішу <F5> вісім разів, виконайте покрокове моделювання роботи демультиплексора для першої половини тестової послідовності (коли сигнал **w**='1').
- 1.5.10 Знову натисніть клавішу <w> і виконайте покрокове моделювання демультиплексора для другої половини тестової послідовності (коли сигнал **w**='0').
- 1.5.11 Вибравши команду головного меню *Simulation* ® *End Simulation*, зупиніть процес моделювання. На рис. 1.19 показаний результат функціонального моделювання правильно працюючого демультиплексора DM(3).

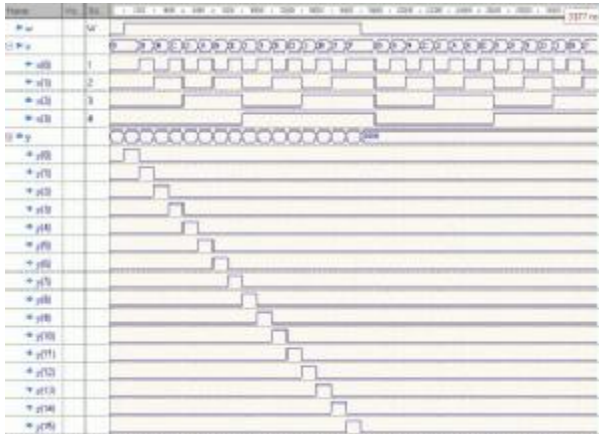


Рисунок 1.19 – Функціональне моделювання демультіплексора DM(3)

1.6 Порядок виконання роботи

1. Проект **Lab1**. Створіть і виконайте моделювання моделі *demux_3*. Для опису поведінки демультіплексора DM(3) використовуйте текст програми, наведений в листингу 1.2. Текст програми та результати моделювання використовуйте для складання звіту.
2. Проект **Lab2**. Створіть новий проект і виконайте моделювання моделі *demux_3*, для опису якої використовується текст програми, наведений в листингу 1.3. Текст програми та результати моделювання використовуйте для складання звіту.
3. Порівняйте результати моделювання і зробіть висновки.

Під час моделювання можна помітити, що сигнали перемикаються неодноразово. Цей ефект називається **гонкою сигналів**. Редактор часових діаграм має необхідні інструменти для докладного вивчення часових діаграм. Наприклад, кнопкою **Zoom in**  можна збільшити масштаб відображення, а натискання кнопки **Measurement Mode** , вмикає режим вимірювання часових інтервалів. На рис. 1.20 показаний приклад вимірювання часу встановлення сигналу.

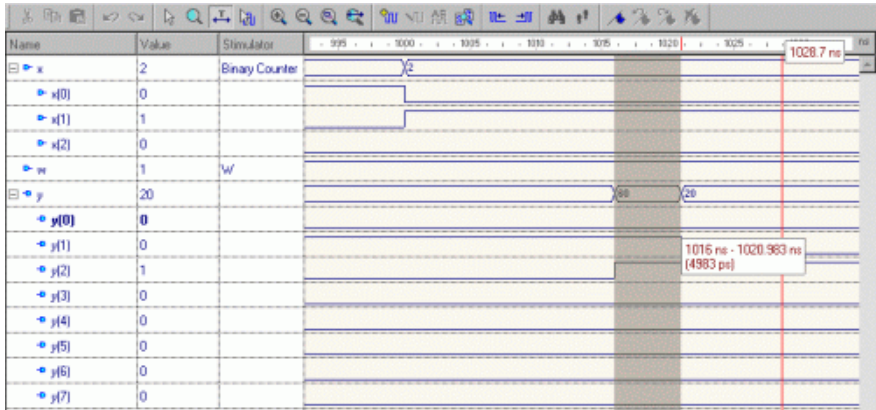


Рисунок 1.20 – Вимірювання часових інтервалів

4. Визначте значення максимальної затримки переключення сигналів вектору $y[0..7]$.

1.7 Зміст звіту

1. Мета роботи.
2. Таблиця дійсності та логічна схема демультимплексора DM(3).
3. Тексти програм.
4. Результати моделювання.
5. Результати вимірювання часових інтервалів.
6. Висновки.

1.8 Контрольні питання

1. Організація VHDL-проекту цифрової системи. Вбудовані стандартні пакети Active-HDL.
2. Мови опису апаратури. Який зв'язок між ними і мовами програмування високого рівня загального користування.
3. Системи автоматизації проектування радіоелектронних пристроїв. Місце САПР Active-HDL серед інших САПР розробки ЦС.
4. Організація САПР Active-HDL. Основні компоненти Active-HDL і їх призначення.

5. Шість рівнів деталізації опису цифрової системи. Ієрархічна організація опису ЦС.
6. Організація VHDL-проекту цифрової системи в САПР Active-HDL. Вбудовані стандартні пакети Active-HDL.
7. Текстовий редактор HDE САПР Active-HDL. Функціональне призначення і основні елементи управління.
8. Використовування бібліотек і пакетів в мові VHDL.
9. З яких частин складається опис об'єкту проектування? Наведіть приклади.
10. Як проводиться компіляція та моделювання проекту?
11. Що таке тестова послідовність? Як задаються тестові набори вхідних сигналів? Наведіть приклади.
12. Типи стимуляторів. Як вони задаються?

Лабораторна Робота № 4

2 МОВА ПРОГРАМУВАННЯ VHDL

Мета роботи. Ознайомитись з мовою VHDL і засобами редагування логічних схем САПР Active-HDL.

2.1 Теоретична довідка

Прагнення до використання єдиної мови на різних ієрархічних рівнях проектування електронних пристроїв, починаючи з вентильного рівня і закінчуючи рівнем опису обчислювальних систем, призвело до появи стандартної мови **VHDL** (Very high speed integrated circuits **H**ardware **D**escription Language).

Мова VHDL базується на мові високого рівня Ада. З цієї мови розроблявачами VHDL були запозичені синтаксис і основні структури. Використано типові структури мов високого рівня: розгалуження, цикли, підпрограми, паралельне виконання процесів і т.д.

2.1.1 Опис об'єктів мовою VHDL

Об'єкти, що описані мовою VHDL, називаються **сутностями** (entity). Об'єктом може бути логічний елемент, функціональний блок, мікросхема, тобто цифрова система (ЦС). Опис об'єкта складається з двох частин (секцій) - декларативної і виконавчої.

Декларативна частина (entity declaration) має наступну форму:

```
entity <ім'я сутності> is
<опис інтерфейсу об'єкта, тобто опис зв'язків сутності
з навколишнім середовищем>;
<опис загальних підпрограм, типів, сигналів і т.і.>;
end <ім'я сутності>;
```

Виконавча частина (architecture body):

```
architecture <ім'я> of <ім'я сутності> is
<декларативна частина - те ж, що й у декларативній частині сутності, але стосовно даного блоку>;
begin <виконавча частина - опис структур і/або дій>;
end <ім'я>;
```

Звичайно в декларативній частині описуються типи компонентів, що входять у сутність, і сигнали, що використовуються, а у виконавчій частині типи конкретизуються, даються імена кожному екземплярові і його виводам.

Опис у виконавчій частині може бути трьох типів:

- **опис потоку даних** (data-flow description), що імітує поведінку об'єкта проекту як послідовність проходження потоку даних і сигналів керування, як правило, на реєстровому рівні.

Опис потоку даних:

```
<мітка>: <ім'я порту> <= <логічний вираз> after <затримка>;
```

- **поведінковий опис**, що імітує поведінку об'єкта проекту як сукупність незалежних, але синхронних за часом паралельних процесів. Послідовні інструкції в середині процесу крок за кроком описують поведінку одного процесу.

Поведінковий опис:

```
<мітка>: process (<список параметрів>)
begin
  <тіло процесу>
end process <мітка>;
```

- **структурний опис** імітує об'єкт проекту як сукупність компонентів структури і зв'язків між ними. Для опису компонентів може бути обраний кожний з перерахованих описів (поведінковий або потоку даних).

Структурний опис:

```
<мітка>: <ім'я - тип компонента> port map (<список сигналів>;
```

Розглянемо як приклад опис структури фрагмента логічної схеми з елементів І-АБО-НІ, що представлений на рис. 2.1, мовою VHDL.

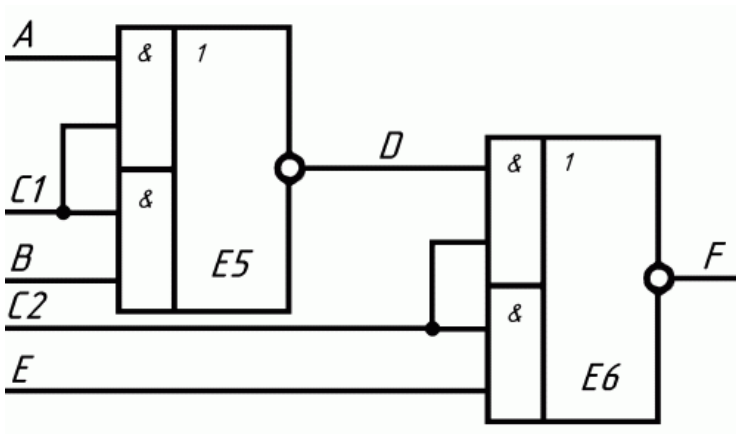


Рисунок 2.1 - Фрагмент логічної схеми

При структурному описі у декларативній частині повинно бути записано:

```
component AND_OR is
port (
    INP1, INP2, INP3, INP4:    in BIT;
    O:                          out BIT);
end component AND_OR;
```

У виконавчій частині:

```
architecture CIRCUIT of LOGIC is
signal D: BIT;
component AND_OR
port (
    INP1, INP2, INP3, INP4:    in BIT;
    O:                          out BIT);
end component AND_OR;
begin
E5: AND_OR
    port map(A, C1, B, C1, D);
E6: AND_OR
    port map(D, C2, E, C2, F);
end CIRCUIT;
```

Опис елемента І-АБО-НІ в декларативній частині включає завдання типу компонента (AND_OR), його входів (INP1 ...INP4), на що вказує символ **in**, його виходу (O), відзначеного символом **out**, і типів сигналів (BIT). Опис

входів і виходів є аналогічним описові формальних параметрів у процедурах алгоритмічних мов.

У виконавчій частині описуються конкретні екземпляри E5 і E6 компонента AND_OR з фактичними параметрами. Тут A, B, C1, C2, D, E, F суть сигнали і, щоб відрізнити їх від звичайних змінних вони повинні бути описані в декларативній частині entity або architecture.

Опис дій, що виконуються над сигналами в елементі AND_OR повинен бути змістом окремого entity.

Опис того ж фрагмента в стилі *data-flow* має вигляд:

```
architecture CIRCUIT of LOGIC is
signal D: BIT;
begin
    F <= not ((D and C2) or (E and C2)) after 5 ns;
    D <= not ((A and C P or (B and C 1)) after 5 ns;
end CIRCUIT;
```

В мові VHDL є три категорії даних, за допомогою яких описуються любі процеси в дискретних системах: константи, змінні і сигнали. Всі вони повинні мати імена: сукупність букв (A-Z), цифр (0-9) та символів підкреслювання '_'. Імена починаються з букви і не можуть закінчуватись символом підкреслювання. Регістр букв не має значення.

Змінна є ім'ям, яке присвоюється вираженням, що змінюються, використовується для зберігання проміжних даних при виконанні послідовних операторів присвоєння в межах процесу.

Сигнал – це інформація, яка передається між модулями проекту і може бути входною чи вихідною. Сигнали передаються по фізичним лініям, які з'єднують елементи схеми. Сигнали завжди пов'язані з часом. Вони повинні бути об'явлені в entity або architecture перед процесом. Присвоєння значення сигналу відображається знаком "<=", а змінній – ":=".

Символ "<=" – еквівалент символу присвоєння, але використовується в описах дій над сигналами на виміну від символу "=", який застосовується до змінних. Відзначимо, що операції зі змінними можна використовувати в поведінкових описах і в підпрограмах, у той час як операції над сигналами можливі в описах і поведінкових, і потоків даних.

Особливості операторів "<=" у мові VHDL полягають у їхньому паралельному виконанні. Це означає, що порядок їхнього запису у виконавчій частині не має значення. Наприклад, якщо маємо запис

```
A <= D or S;
B <= A and S;
```

і відомо, що $D="1"$, $S="0"$ до моменту виконання цих операторів, то після їхнього виконання сигнали будуть мати значення $A="1"$, $B="0"$. Іншими словами, присвоєння нових значень сигналам відбувається в момент $t+\delta$, де t - поточний час, δ - як завгодно мала позитивна величина.

Поведінковий опис для даного приклада оформляється у виді **процесу**:

```
architecture CIRCUIT of LOGIC is
signal D: BIT;
begin
LAB: process (A,B,C1,C2,D,E)
begin
    D <= not ((A and C1) or (B and C1)) after 5 ns;
    F <= not ((D and C2) or (E and C2)) after 5 ns;
end process LAB;
end CIRCUIT;
```

Особливість процесу в мові VHDL полягає в тому, що за його допомогою реалізується моделювання: процес активізується тільки тоді, коли змінюється який-небудь із сигналів, перерахованих у списку його параметрів (у даному випадку це A,B,C1,C2,D,E). Цей список називають **списком чутливості** (sensitivity list).

Один із варіантів повного опису приклада логічної схеми (рис. 2.1):

Листинг 2.1 – Варіант опису логічної схеми мовою VHDL

```
-- Декларативна частина сутності LOGIC
entity LOGIC is
port (
    A,B,E,C1,C2: in BIT;
    F: out BIT);
end LOGIC;

-- Виконавча частина сутності LOGIC
architecture CIRCUIT of LOGIC is
signal D: BIT;

-- Декларація компоненту AND_OR
component AND_OR
port (
    Inp1,Inp2,Inp3,Inp4: in BIT;
    O: out BIT);
end component;
```

```

-- Архітектурне тіло сутності LOGIC
begin
E1:AND_OR
    port map (A,C1,B,C1,D);
E2: AND_OR
    port map (D,C2,E,C2,F);
end CIRCUIT;

-- Декларативна частина сутності AND_OR
entity AND_OR is
port (
    Inp1,Inp2,Inp3,Inp4: in BIT,
    O: out BIT);
end AND_OR;

-- Виконавча частина сутності AND_OR
architecture GATE of AND_OR is

-- Архітектурне тіло сутності AND_OR
begin
    O <= not ((Inp1 and Inp2) or (Inp3 and Inp4));
end GATE;

```

Можна представити вхідні сигнали у вигляді двійкового вектора X. Тоді у вищенаведеному описі були б наступні зміни.

У декларативній частині LOGIC опис зв'язків мав би вид:

```

port (
    X:      in      BIT_VECTOR(1 to 3);
    C1,C2:   in      BIT;
    O:      out     BIT);

```

У виконавчій частині CIRCUIT замість сигналів A, B, E фігурували сигнали X(1), X(2), X(3).

2.1.2 Типи даних і операції

Використовуються **скалярні** і **складені** (компонентні) типи. У свою чергу, скалярні типи підрозділяються на типи нечислові (перелічувані), числові і фізичні.

До **нечислових типів** відносяться:

тип *BOOLEAN* з сукупністю значень "False", "True";

тип *BIT* з сукупністю значень "0", "1";

тип *CHARACTER* – значеннями є літери латинського алфавіту.

До **числових типів** відносяться:

тип *REAL* (дійсні значення);

тип *INTEGER* (цілі числа).

Фізичний тип використовується для позначення одиниць виміру. Прикладом фізичного типу є *TIME* для опису границь діапазону й одиниць вимірювання часу.

Серед **складених типів** виділяють *ARRAY* – масив, *RECORD* – запис, *FILE* – файл. Окремими випадками типу *ARRAY* є типи *STRING* (рядок) і *BIT_VECTOR* (вектор із двійковими елементами).

В пакеті *STANDART*, що поставляється разом з системою, декларуються типи *BOOLEAN*, *BIT*, *BIT_VECTOR*, *INTEGER*, *REAL*, *NATURAL*, *POSITIV*, *CHARACTURE*, *STRING*, а також *TIME* (час).

Ряд важливих визначень типів даних, наприклад, *STD_LOGIC_VECTOR*, *STD_LOGIC*, функцій і т.п. міститься в пакеті *STD_LOGIC_1164* бібліотеки *IEEE*. Для цього на початку програми записуються інструкції

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

Введення нових типів здійснюється так, як це прийнято в алгоритмічних мовах типу Паскаль. Наприклад:

1) декларація фізичного типу:

```
type VOLTAGE is range 1 to 20;
units
```

```
    uv;                                -- microvolts      (базова
ОДИНИЦЯ)--
```

```
    mv = 1000 uv;    -- millivolts--
```

```
    v = 1000 mv;    -- volts--
```

```
end units;
```

2) декларація перелічуваного типу, який задається користувачем:

```
type COLOR is
(WHITE, BLACK, RED, GREEN, BLUE, YELLOW, MAGENTA);
```

3) декларація типу масива:

```
type VEC is array (12 downto 0) of INTEGER;
```

4) декларація типу запису:

type COMP is record

NAME: STRING (3 to 15);

PERF: REAL range 0.1 to 1E9;

FIRM: STRING (2 to 12);

end record;

5) об'ява:

NAME: STRING (3 to 15);

Цікавою рисою мови VHDL є наявність у ній величин, що аналогічні стандартним числовим атрибутам мови GPSS і які також називаються **атрибути**. Приклади часто використовуваних атрибутів:

S'LAST_VALUE попереднє значення змінної S

S'STABLE(T) дорівнює true, якщо сигнал S стабільний протягом останніх T одиниць часу

S'DELAY(T) значення сигналу S у момент часу t-T, де t – даний момент часу

У VHDL можна оформляти описи типів у виді пакетів. **Пакет** - поняття, аналогічне поняттю підпрограми, але яке стосується не виконавчої, а декларативної частини. Структура пакета:

package PPP is

<описи, загальні для декількох частин моделі, в які можуть входити типи, підтипи, константи, сигнали процедури, функції й інші декларації>;

end PPP;

Якщо в загальний опис входять які-небудь функції і/або підпрограми, то додається опис їхніх тіл після заголовка пакету.

Доступ до пакета PPP з боку сутності CCC організується записом

use PPP.ALL

перед декларативною частиною сутності, тобто перед рядком. Це означає, що для сутності CCC стає "видимим" весь уміст пакета PPP.

Щоб визначити, які об'єкти проекту використані для створення цілого проекту, використовується **об'ява конфігурації** (package declaration). Таким чином, при описі ЦС мовою VHDL можна використати п'ять різних типів опису: об'ява об'єкта проекту, архітектурне тіло, об'ява конфігурації, об'ява пакета і тіло пакета.

В арифметичних і логічних виразах і в умовах виконання дій використовуються наступні символи операцій:

not, and, or, nand, nor, xor – для логічних операцій (НІ, І, АБО, І-НІ, АБО-НІ, виключне АБО відповідно);
"=", "/=", "<", "<=", ">", ">=" – для відношень (дорівнює, не дорівнює, менше, менше або дорівнює, більше, більше або дорівнює);
+", "-", "*", ""** – для арифметичних операцій додавання, віднімання, множення, зведення в ступінь відповідно.

2.1.3 Оператори мови VHDL

У VHDL застосовуються оператори присвоєння, умовний, вибору, циклу, очікування, контролю, блоку. Більшість з них за змістом збігається з операторами алгоритмічних мов загального застосування.

Вид умовного оператора:

```
if <умова> then <оператори>
elsif <умова> then <оператори>
else <оператори>
end if;
```

Вид оператора вибору:

```
case <вираз> is
when <альтернатива> => <оператори>
...
when others => <оператори>
end case;
```

Вид оператора циклу:

```
[<мітка циклу>]
while <умова> [for < опис параметрів >] loop
<оператори>
end loop;
```

Вихід з циклу можливий за допомогою оператора

```
exit [<мітка циклу>] [when <умова>];
```

Оператор очікування використовується для припинення процесу на деякий відрізок часу, під час зміни значення сигналу, при виконанні деякої умови:

```
wait on <ім'я сигналу> until <умова> for <час затримки>;
```

Наприклад:

```
wait on A,B until C=0 for 50 ns;
```

Оператор контролю ASSERT потрібен для одержання користувачем повідомлень про помилковій ситуації.

Оператор блоку BLOCK вживають для поділу описів в архітектурних тілах на блоки, кожний з яких має свої внутрішні описи і виконавчу частину.

Як і в алгоритмічних мовах, у VHDL можна використовувати два типи підпрограм:

процедури, які можуть повертати множину значень;

функції, які можуть повертати єдине значення.

Наведемо формат їх виклику та об'яви:

формат виклику:

```
<ім'я функції або процедури> (<фактичні параметри>);
```

формат об'яви:

```
function | procedure <ім'я> (<формальні параметри>)
```

```
return <тип значення, що повертається> is
```

```
begin
```

```
<тіло функції або процедури>
```

```
end [<ім'я >];
```

Наприклад, функція перетворення типу BIT на тип BOOLEAN:

```
function BIT_BOOL (inp_bit: in BIT) return BOOLEAN is
```

```
begin
```

```
  if(inp_bit='1') then return true;
```

```
  else return false;
```

```
  endif;
```

```
end BIT_BOOL;
```

Наведемо варіант програми, в якому використовуються оператори і процедури.

Листинг 2.2 – Використання процедур у програмі

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- декларативна частина
entity example is
    port(
        a : in STD_LOGIC;
        c : out STD_LOGIC_VECTOR (0 to 1)
    );
end example;

-- Виконавча частина
architecture ex_beh of example is

-- Процедура proc1
procedure proc1 (c: out STD_LOGIC_VECTOR (0 to 1)) is
    begin
        c := "00";
        assert a = '1' report "Signal a is one" severity
WARNING;
    end procedure proc1;

-- Процедура proc2
procedure proc2 (c: out STD_LOGIC_VECTOR (0 to 1)) is
variable count : integer := 0;
    begin
        while count < 5 loop
            case count is
                when 0 => c := "11";
                when 1 => c := "11";
                when 2     => c := "11";
                when 3 => c := "11";
                when others => c := "ZZ";
            end case;
            count := count + 1;
        end loop;
        assert a = '0' report "Signal a is zero" severity
WARNING;
    end procedure proc2;

```

```
-- Архітектурне тіло
```

```
begin
```

```
    EXP: process (a)
```

```
    variable value: STD_LOGIC_VECTOR (0 to 1);
```

```
    begin
```

```
        if a = '1' then
```

```
            proc1 (value); -- ВИКЛИК proc1
```

```
            c <= value;
```

```
        else
```

```
            proc2 (value); -- ВИКЛИК proc2
```

```
            c <= value;
```

```
        end if;
```

```
    end process EXP;
```

```
end ex_beh;
```

Зверніть увагу на те, що всередині процедури замість символу призначення сигналу "<=" треба використовувати символ присвоєння значення змінній ":=".

2.2 Порядок виконання роботи

1. Скласти опис схеми (рис. 2.1) на мові VHDL трьома типами опису: структурному, потоковому та поведінковому.
2. Провести компіляцію написаних програм та їх моделювання.
3. Порівняти результати моделювання і зробити висновки.

2.3 Зміст звіту

1. Мета роботи.
2. Рисунок схеми та її опис.
3. Тексти написаних програм.
4. Результати моделювання.
5. Висновки.

2.4 Контрольні питання

1. Історія розвитку мови VHDL. Стандарти мови. Призначення кожного стандарту.

2. Цифрові системи. Зв'язок між описом поведінки і структури ЦС. Наведіть приклад.
3. Організація VHDL-проекту цифрової системи в САПР Active-HDL.
4. Три методи опису ЦС мовою VHDL (потоків даних, поведінковий, структурний). Наведіть приклади.
5. Текстовий редактор HDE САПР Active-HDL. Функціональне призначення і основні елементи управління.
6. Редактор логічних схем BDE САПР Active-HDL. Функціональне призначення і основні елементи управління.
7. Використовування бібліотек і пакетів в мові VHDL.
8. Модуль entity VHDL-проекту САПР Active-HDL. Загальний опис і функціональне призначення декларативної частини VHDL-програми.
9. Модуль architecture VHDL-проекту САПР Active-HDL. Загальний опис і функціональне призначення виконавчої частини VHDL-програми.
10. Базові типи даних VHDL (логічні, числові, фізичні).
11. Багаторівнева логіка (стандарт IEEE Std. 1164).
12. Складені типи даних (масиви, записи тощо).
13. Атрибути мови VHDL.
14. Оператори мови VHDL. Наведіть приклади використання.
15. Послідовні і паралельні інструкції мови VHDL.
16. Використання підпрограм у мові VHDL. Наведіть приклади.

З ПРИКЛАД МОДЕЛЮВАННЯ СУМАТОРА

Суматор $SM(n)$ складається з n повних суматорів. **Повний суматор** має три входи (A – перший доданок, B – другий доданок, C – вхід переносу). На входи A і B повних суматорів надходять два n -розрядних доданка, перенос C передається від попереднього до наступного розряду. Таблиця дійсності повного суматора наведена в таблиці 3.1.

Таблиця 3.1 – Таблиця дійсності повного суматора

Входи			Виходи	
A	B	C	S	P
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Опис мовою VHDL повного суматора, що має входи двох доданків A й B , вхід переносу з боку молодшого сусіднього розряду C та виходи суми S і переносу P у сусідній старший розряд наведений у листингу 3.1.

Складемо на мові VHDL програму імітації поведінки суматора $SM(3)$ (трьох-розрядного суматора). Суматор $SM(3)$ складається з трьох повних суматорів.

3.1 Створення проекту

3.1.1 Запустіть систему проектування і моделювання ЦС Aldec Active-HDL.

3.1.2 Створіть нове робоче середовище. Для цього можна використати команду головного меню *File* ® *New* ® *Workspace*, або відповідну кнопку панелі інструментів (рис. 3.1).

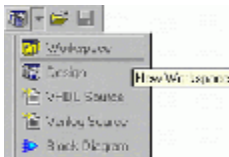


Рисунок 3.1 – Створення робочого середовища

- 3.1.3 З'явиться вікно **New workspace**. Введіть ім'я робочого середовища, наприклад *Summa*. Прапорець "Add new design to workspace" (додати новий проект до створеного робочого середовища) залиште активним. Клацніть кнопку **OK**.
- 3.1.4 Автоматично запуститься майстер **New Design Wizard**. Під час роботи з ним виберіть режим "Create an empty design" (створити пустий проект), введіть ім'я VHDL-проекту. Після натискання кнопки **Done** в робочому середовищі *Summa* буде створено пустий VHDL-проект.

3.2 Створення компоненту

- 3.2.1 Натисніть кнопку *View Design Browser* . З'явиться вікно **Design Browser**, що наведено на рис. 3.2 зліва. Виберіть команду *Add new file* (додати новий файл) у вікні **Design Browser**. З'явиться вікно **Add New File** (на рис. 3.2 праворуч). У вікні **Add New File** виберіть вкладку *Wizards* і клацніть на піктограмі *Block Diagram Wizard*.

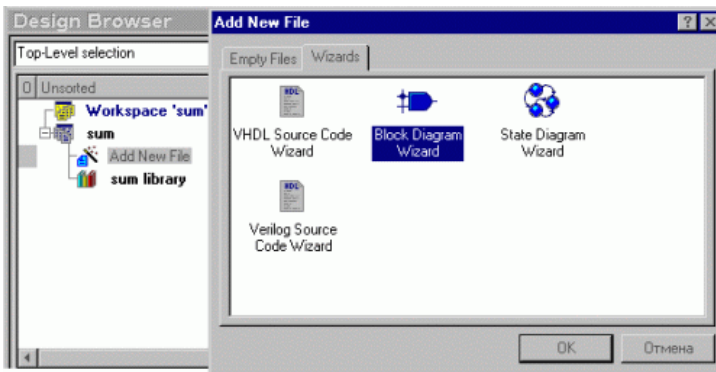


Рисунок 3.2 – Вікна Design Browser і Add New File

- 3.2.2 З'явиться вікно **New Source File Wizard** (майстер нового файлу програми). Під час роботи з майстром **New Source File Wizard** залиште активним прапорець "Add the generated file to the design" (додати файл програми до проекту), виберіть мову опису апаратури VHDL, вкажіть ім'я файлу програми (source file), ім'я сутності (entity), ім'я опису архітектури (architecture body). Виконавши необхідні настройки, для переходу до наступного кроку роботи майстра користуйтеся кнопкою **Next**. Після трьох на-

тискань цієї кнопки з'явиться вікно **New Source File Wizard – Ports** (рис. 3.3).

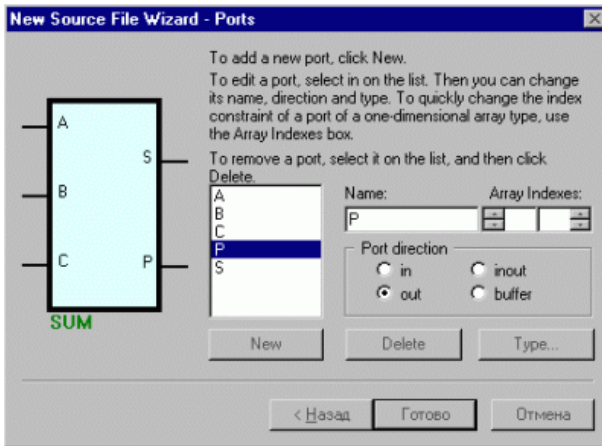


Рисунок 3.3 – Налаштування портів сутності SUM

3.2.3 Додайте до створюваної сутності порти: входи (**in**) **A, B, C** і виходи (**out**) **S, P**. Тип даних, що використовується за замовчуванням – **STD_LOGIC** (багатозначна промислова логіка у відповідності до стандарту IEEE 1164). Зовнішній вигляд вікна **New Source File Wizard – Ports** наведений на рис. 3.3.

3.2.4 Наприкінці роботи з майстром **New Source File Wizard** натисніть кнопку **Done**. Буде автоматично створений файл логічної схеми повного суматора *sum.bde* і з'явиться вікно редактора логічних схем САПР Active-HDL (рис. 3.4).

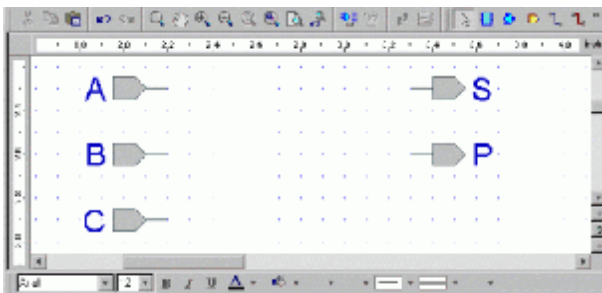


Рисунок 3.4– Редактор логічних схем

3.2.5 Використовуючи інструменти: *Fub* (блок)  і *Wire* (дріт) , добудуйте логічну схему повного суматора (рис. 3.5).

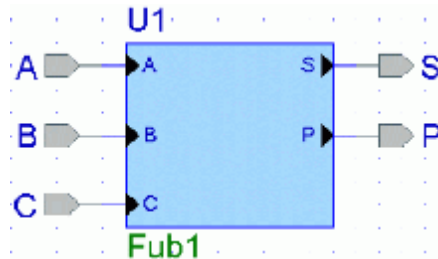


Рисунок 3.5 – Логічна схема повного суматора

3.2.6 Вибравши команду головного меню *File* ® *Save All*, збережіть всі зміни у VHDL-проекті.

3.2.7 Клацніть правою кнопкою миші на блоці *Fub1* (прямокутник блакитного кольору логічної схеми). В контекстному меню виберіть команду *Push*. З'явиться вікно **Create New Implementation** (створити нову виконавчу частину сутності). Виберіть режим *VHDL Source Code*. Буде автоматично створений файл, який містить шаблон VHDL-програми, що відповідає структурі блоку *Fub1*. У нашому випадку це файл *Fub1.vhd*. Файл *Fub1.vhd* автоматично завантажується у текстовий редактор САПР Active-HDL (рис. 3.6).

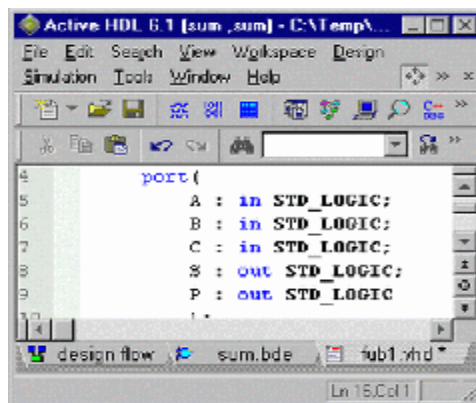


Рисунок 3.6 – Текстовий редактор

3.2.8 Внесіть зміни у виконавчій частині VHDL- програми (листинг 3.1).

Листинг 3.1 – Повний суматор

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- Декларативна частина
entity Fub1 is
    port(
        A,B,C : in STD_LOGIC;
        S,P : out STD_LOGIC
    );
end Fub1;

-- Виконавча частина
architecture Fub1 of Fub1 is
begin
    S <= A xor B xor C;
    P <= (A and B) or (A and C) or (B and C);
end Fub1;

```

3.2.9 Клацнувши кнопку *Compile All* (компілювати все) , додайте створений блок *Fub1* до бібліотеки VHDL-проекту. Тепер цей компонент можна використовувати як складову частину проекту більш складної ЦС.

3.3 Створення файлу верхнього рівня

3.3.1 Створіть логічну схему файлу верхнього рівня проекту. Для цього можна використати команду *Add new file* у вікні **Design Browser**. У контекстному меню виберіть режим *New* ® *Block Diagram*.

3.3.2 З'явиться вікно **New Source File Wizard**. Під час роботи з майстром **New Source File Wizard** залиште активним прапорець "*Add the generated file to the design*" (дати файл програми до проекту), виберіть мову опису апаратури VHDL, вкажіть ім'я файлу програми, ім'я сутності, ім'я опису архітектури. У вікні **New Source File Wizard – Ports** вкажіть порти: входи a(0)-a(2), b(0)-b(2), с і виходи s(0)-s(2), р.

3.3.3 Після натискання кнопки **Done** буде автоматично створений шаблон файлу логічної схеми. Цей файл буде завантажений у редактор логічних схем САПР Active-HDL. Натисніть кнопку *Show Symbols Toolbox* (показати список символів) . З'явиться вікно **Symbols Toolbox**, що містить список вбудованих компонентів (built-in symbols) і компонентів, що належать бібліотеці поточного VHDL-проекту.

3.3.4 Перетягніть компонент *SUM* з вікна **Symbols Toolbox** у вікно редактора логічних схем. За допомогою клавіш <Ctrl+C> і <Ctrl+V> зробіть так, щоб логічна схема файлу верхнього рівня VHDL- проекту містила три компонента SUM (U1, U2 і U3). Наприкінці роботи логічна схема повинна мати вид, як на рис. 3.7.

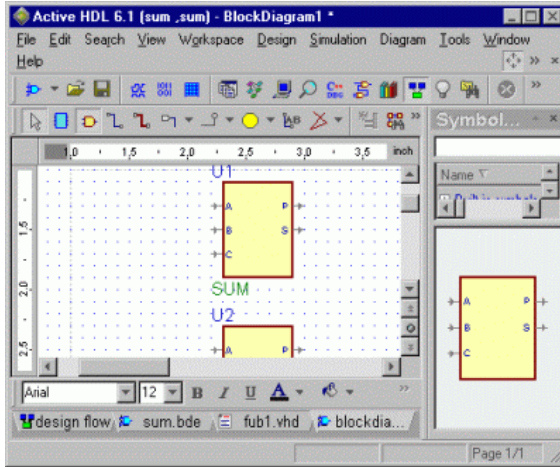


Рисунок 3.7– Додавання символів до логічної схеми

3.3.5 За допомогою інструментів *Wire*  і *Bus* (шина)  добудуйте логічну схему суматора SM(3) (рис. 3.8).

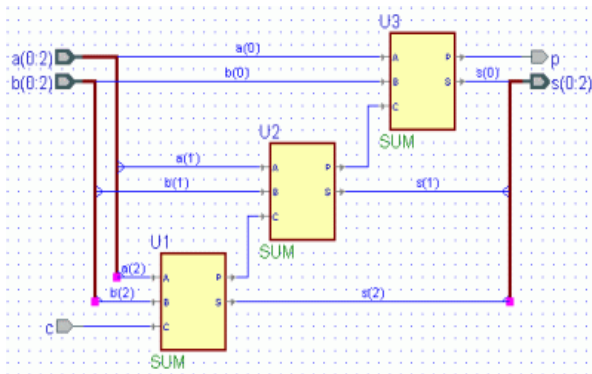


Рисунок 3.8 – Логічна схема суматора SM(3)

- 3.3.6 Вибравши команду головного меню *File ® Save All*, збережіть всі зміни у VHDL-проекті.
- 3.3.7 Створений файл логічної схеми є файл верхнього рівня (top-level file) VHDL-проекту, тому його треба компілювати першим. Змініть порядок компіляції файлів проекту. Для цього у вікні **Design Browser** виділіть піктограму проекту  **sum** і в контекстному меню виберіть команду *Design Compilation Order*. Перетягніть піктограму вашого файлу на першу позицію у списку, що міститься у вікні **Design Compilation Order**.
- 3.3.8 За допомогою кнопки *Compile with file reorder* (компіляція у встановленому порядку)  виконайте заключну частину роботи – компіляцію готового VHDL-проекту, що має ієрархічну організацію. Якщо ніяких помилок під час компіляції не виявлено, будуть автоматично створені файли, що містять структурний опис ЦС.

Листинг 3.2 – Структурний опис суматора SM(3)

```
library IEEE;
use IEEE.std_logic_1164.all;

-- Декларативна частина сутності Sum_3
entity Sum_3 is
    port(
        c : in STD_LOGIC;
        a : in STD_LOGIC_VECTOR(0 to 2);
        b : in STD_LOGIC_VECTOR(0 to 2);
        p : out STD_LOGIC;
        s : out STD_LOGIC_VECTOR(0 to 2)
    );
end Sum_3;

-- Виконавча частина сутності Sum_3
architecture Sum_beh of Sum_3 is

    -- Декларація компоненту SUM
    component SUM
        port (
            A,B,C : in STD_LOGIC;
            S,P : out STD_LOGIC
        );
    end component;

    -- Декларація допоміжних сигналів
    signal NET137 : STD_LOGIC;
    signal NET145 : STD_LOGIC;

begin

    -- Реалізація екземпляра U1
```

```

U1 : SUM
  port map(A => a(2),B => b(2),C => c,P => NET145,S => s(2));

-- Реалізація екземпляра U2
U2 : SUM
  port map(A => a(1),B => b(1),C => NET145,P => NET137,S => s(1));

-- Реалізація екземпляра U3
U3 : SUM
  port map(A => a(0),B => b(0),C => NET137,P => p,S => s(0));
end Sum_beh;

-- Декларативна частина сутності SUM
entity SUM is
  port(
    A,B,C : in STD_LOGIC;
    S,P : out STD_LOGIC
  );
end SUM;

-- Виконавча частина сутності SUM
architecture FUN of SUM is

-- Декларація компоненту Fub1
component Fub1
  port (
    A,B,C : in STD_LOGIC;
    S,P : out STD_LOGIC
  );
end component;

begin

-- Реалізація екземпляра U1
U1 : Fub1
  port map(A => A,B => B,C => C,P => P,S => S);
end FUN;

```

На рис. 3.9 наведений фрагмент часової діаграми, що була отримана під час функціонального моделювання суматора SM(3).

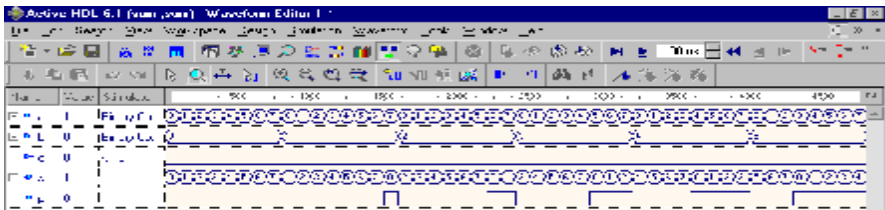


Рисунок 3.9 – Тестування моделі суматора SM(3)

ЛІТЕРАТУРА

1. Проектирование цифровых систем с использованием языка VHDL: Уч. пособие/ В.В. Семенец, И.В. Хаханова, В.И. Хаханов.– Харьков: ХНУРЭ, 2003.– 492 с.
2. Бибилло П.Н. Синтез логических схем с использованием языка VHDL.– М.: СОЛОН-Р, 2002.– 384 с.
3. Кондратенко Ю.П., Сидоренко С.А., Підпригора Д.М. Поведінковий синтез цифрових пристроїв у середовищі Active-HDL: Навчальний посібник.– Миколаїв: МФ НаУКМА, 2002.– 116 с.
4. Стешенко В.Б. ПЛИС фирмы Altera: элементная база, система проектирования и языки описания аппаратуры. – М.: Издательский дом Додэка –XXI, 2002.– 576 с.
5. Шапо Ф.С., Шапо В.Ф. Введение в VHDL – язык проектирования цифровых систем.– Одесса: Астропринт, 2001.– 220 с.
6. Дж. Ф. Уэйкерли Проектирование цифровых устройств. В 2-х томах: Пер. с англ. М.: Постмаркет, 2002.
7. Угрюмов Е.П. Цифровая схемотехника.–СПб.: БХВ–Петербург, 2001.– 528с.(Гл.9)
8. Бибилло П.Н. Основы языка VHDL.– М.: Солон–Р, 2000.–200с.
9. А.К. Тімовський, , О.О.Голдобін Методичні вказівки до лабораторних робіт з дисципліни “Автоматизація проектування комп'ютерних систем”.– ЗДТУ, 2009