О.І. Гороховський

«МОДЕЛЮВАННЯ В СЕРЕДОВИЩІ Active-HDL»

Міністерство освіти і науки України
Вінницький національний технічний університет

О.І. Гороховський

«МОДЕЛЮВАННЯ В СЕРЕДОВИЩІ Active-HDL»

Затверджено Вченою радою Вінницького національного технічного університету як лабораторний практикум для студентів бакалаврського напряму 6.091500 – “Комп’ютерна інженерія”. Протокол № 11 від “30” червня 2005 р.

Вінниця ВНТУ 2006

УДК 658.512.22
Г 77

Р е ц е н з е н т и:

В.М. Лисогор, доктор технічних наук, професор
В.А. Лужецький, доктор технічних наук, професор
Д.Т. Обідник, кандидат технічних наук, доцент

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

Гороховський О.І.
Г 77 **«МОДЕЛЮВАННЯ В СЕРЕДОВИЩІ Active-HDL»**. Лабораторний практикум. – Вінниця: ВНТУ, 2006.- 89 с.

В практикумі розглянуто основи використання пакета „Active-HDL 3.6” для автоматизації проектування та моделювання цифрових пристроїв. Посібник розроблений відповідно до плану кафедри та програми дисципліни "Основи автоматизованого проектування засобів ОТ”.

УДК 658.512.22

ЗМІСТ

ВСТУП

На сьогодні при проектуванні будь-якого цифрового пристрою головна увага приділяється підходу до проектування цифрових пристроїв, який базується на застосуванні програмованих логічних інтегральних схем (ПЛІС), як найсучаснішої елементної бази електронних пристроїв. Суть проектування ПЛІС полягає в розробці спеціальної програми (у вигляді двійкового потоку "bit-stream"), при завантаженні якої в інтегральну мікросхему (ІМС) остання виконує функції спроектованого електронного пристрою.

Найсучасніший підхід до проектування ІМС грунтується на застосуванні спеціалізованих мов описання обладнання (HDL – Hardware Description Language). Ці мови, крім звичних для мов Паскаль та Сі конструкцій, таких як розгалуження, цикли, підпрограми тощо, мають також спеціалізовані засоби:

- робота з портами;
- забезпечення паралельності виконання процесів та ін.

Найпоширенішими серед HDL стали ABEL, Verilog та VHDL.

ABEL (*Advanced Boolean Equation Language* – розширена мова логічних рівнянь) є промисловим стандартом, що розроблений Data I/O Corp. для програмованих логічних пристроїв. ABEL може застосовуватися для описання поведінки систем (за допомогою С-подібних операторів) в різних формах: на основі логічних рівнянь, таблиць істинності, діаграм стану.

Порівняно з ABEL мови VHDL та Verilog є складнішими, але придатнішими для описання великих систем. Вони практично рівні за своїми технічними можливостями. В Європі та США ширше застосовується VHDL, в країнах Азії – Verilog.

VHDL базується на мові високого рівня Ada. З цієї мови розробниками VHDL було запозичено синтаксис та основні структури.

Відповідно до сучасних вимог системи автоматизованого проектування програмованих логічних ІМС повинні забезпечувати: реалізацію однієї або більше HDL-мов з можливістю введення, редагування та відлагодження вихідного тексту програм; реалізацію засобів графічного введення проектної схеми, наприклад, за допомогою редактора скінченних автоматів, та засобів компіляції графічного подання в HDL-код; реалізацію засобів моделювання поведінки описаного об'єкта; реалізацію засобів синтезу бітового потоку з підтримкою широкого класу серій ІМС; реалізацію засобів моделювання об'єкта на рівні вентилів; реалізацію засобів програмування ІМС.

В наш час розроблено ряд програмних продуктів, що реалізують ту чи іншу частину загальних вимог до САПР ПЛІС. Застосування цих продуктів вимагає в кожному конкретному випадку розв'язання проблем сумісності між пакетами, тому перевагу слід надавати пакетам, що

реалізують процес проектування ПЛІС якомога повніше. Одним з таких пакетів є Active-HDL фірми Aldec Inc.

Лабораторний практикум призначено для виконання курсу лабораторних робіт з дисципліни “Основи автоматизованого проектування засобів ОТ”. Матеріал практикуму поділено на три розділи:

- Інформація про основні мовні конструкції для описання модельованих схем (розділи 1, 2 і 3);
- Принципи роботи в робочому середовищі Active-HDL (розділи 4, 5 і 6);
- Методичні рекомендації для виконання циклу лабораторних робіт (розділ 7).

Робочою навчальною програмою дисципліни “Основи автоматизованого проектування засобів ОТ” передбачено виконання таких робіт:

1. Опис інтерфейсу та архітектури найпростіших об’єктів (сутностей) – цифрових пристроїв.
2. Дослідження цифрових компонентів стандартних бібліотек середовища Active-HDL.
3. Побудова принципової цифрової схеми та її моделювання в середовищі Active-HDL.
4. Розробка і моделювання паралельного та зсувного регістрів.
5. Проектування та дослідження запам’ятовувальних пристроїв в середовищі Active-HDL.
6. Моделювання транспортної та інерційної затримок часу в середовищі Active-HDL.

1 АЛФАВІТ МОВИ VHDL

1.1 Правила формування ідентифікаторів

Ідентифікатори в VHDL поділяються на прості й розширені. Прості ідентифікатори формуються за такими правилами:

1. Ідентифікатори можуть складатися з латинських букв, цифри або знаку підкреслення “_”.
2. Ідентифікатори повинні починатися з букви.
3. Ідентифікатори не повинні закінчуватися підкресленням.
4. Ідентифікатори не можуть включати в себе два знаки підкреслення підряд.
5. Як ідентифікатори не можуть застосовуватися зарезервовані слова стандартного VHDL, повний список яких наведено нижче:

abs	**downto**	**Library**	**shared**	**sra**
access	**else**	**Linkage**	**sla**	**srl**
after	**elseif**	**Literal**	**sll**	**subtype**
alias	**end**	**Loop**	**postponed**	**then**
all	**entity**	**Map**	**procedure**	**to**
and	**exit**	**Mod**	**process**	**transport**
architecture	**file**	**Nand**	**pure**	**type**
array	**for**	**New**	**range**	**unaffected**
assert	**function**	**Next**	**record**	**units**
attribute	**generate**	**Nor**	**register**	**until**
begin	**generic**	**Not**	**reject**	**use**
block	**group**	**Null**	**rem**	**variable**
body	**guarded**	**Of**	**report**	**wait**
buffer	**if**	**On**	**return**	**when**
bus	**impure**	**Open**	**rol**	**while**
case	**in**	**Or**	**ror**	**with**
component	**inertial**	**Others**	**select**	**xnor**
configuratio	**inout**	**Out**	**severity**	**xor**
n	**is**	**Package**	**signal**	
constant	**label**	**Port**		
disconnect				

Крім того, мова VHDL дає можливість застосування розширених ідентифікаторів. Такі ідентифікатори можуть складатися з будь-яких символів, обмежених з двох сторін символами “\” (слеш), наприклад, \Ідентифікатор 1&2\.

Приклади правильно створених ідентифікаторів:

Decoder_1 FFT Sig_N Not_Ack \signal\ \C:\\Cads\ \Сигнал @#

Приклади неправильних ідентифікаторів:

_Decoder_1 2FFT Sig_#N Not-Ack Сигнал_1

1.2 Спеціальні символи

Крім ідентифікаторів, мова VHDL включає в себе спеціальні символи:

*& ‘ ( ) * + , - . / : ; < = > |*

та пари спеціальних символів:

*** := /= >= <= <>**

1.3 Числа в VHDL

В VHDL застосовуються цілі та дійсні числа. Приклад цілих чисел:

23 0 146 -10

Приклад дійсних чисел:

23.1 12.0 3.11459 3.234E09 8.6E+21 34.0E-08

Необхідно звернути увагу на те, що в дійсних числах обов’язково повинен стояти хоча б один знак після коми, навіть якщо це нуль.

При необхідності числа можна задавати в інших системах числення (від 2 до 16), наприклад:

2#11011# 2#0011# 16#10FA# 10#1024#E+00

Цифри, більші 9 задаються за допомогою латинських літер A, B, C, D, E, F.

Для більшої наочності запису довгих чисел всередині їх можна вставляти знак підкреслення “_”. На інтерпретацію числа комп’ютером цей знак не впливає, однак спрощує сприйняття цього числа оператором, наприклад:

1_050_406 еквівалентно 1050406;

2#1010_1001_1011_1110# еквівалентно 2#1010100110111110#.

1.4 Символи

Символи в VHDL – аналог типу char в мові Pascal. При записуванні вони поміщаються в одинарні лапки, наприклад:

‘a’ ‘b’ ‘;’ ‘>’.

1.5 Рядки

На відміну від Pascal, в VHDL рядки поміщаються не в одинарні, а в парні лапки, наприклад:

“Рядок 1”, “Вивчаємо VHDL”.

Якщо є необхідність задавати довгі рядки, які не поміщаються в один екранний рядок, то можна записувати їх в кілька екранних рядків, ставлячи на початку знак конкатенації “&”, наприклад, запис:

“Довгий, довгий, довгий …… рядок”

еквівалентний запису

“Довгий, довгий”

&”, довгий ……”

&”…… рядок”.

2 ЗАСОБИ VHDL ДЛЯ МОДЕЛЮВАННЯ РЕАЛЬНИХ ОБ'ЄКТІВ

2.1 Описання об'єктів в VHDL (Entity)

VHDL є мовою описання реальних об'єктів та процесів, що протікають в цих об'єктах. Для такого описання VHDL містить ряд спеціалізованих конструкцій.

Описання об'єкта в VHDL починається з визначення зв'язків між об'єктом та зовнішнім середовищем. Повний перелік таких зв'язків називається ***інтерфейсом***. Синтаксис описання об'єктів:

entity *identifier* **is**
port (*port_interface_list*);
entity_declarative_item1;
entity_declarative_item2;
…
entity_declarative_item;
end entity *identifier*;

В даному випадку *identifier* – ім'я об'єкта, *port_interface_list* – інтерфейс об'єкта, *entity_declarative_items* – необов'язковий список оголошень, що може містити описання констант, типів, сигналів (буде розглянуто нижче);

port_interface_list містить повний перелік всіх портів (вхідних та вихідних сигналів) об'єкта з означенням їх типу та напряму (вхідний, вихідний, двонаправлений). Синтаксис опису портів:

(*identifier1*: **mode** *type* **: =** *expression1*;
identifier2: **mode** *type* **: =** *expression2*;
…);

Тут *identifier1, identifier2* – імена портів; **mode** – параметр, що вказує на напрям порту і може набувати одного з трьох значень: **in** – вхідний, **out** – вихідний та **inout** – двонаправлений; **type** – тип порту; *expression* – початкове значення порту. Для прикладу розглянемо описання інтерфейса чотирьохрозрядного суматора з двома вхідними та одним вихідним портами:

entity *adder* **is**
port(*a*: **in** *Std_Logic_Vector* (3 **downto** 0);
b: **in** *Std_Logic_Vector* (3 **downto** 0);
c: **out** *Std_Logic_Vector* (3 **downto** 0) := "0000");
end entity *adder*;

В одному рядку можна оголосити кілька однакових за типом та напрямом портів. Так, попередній приклад можна переписати у вигляді:

entity *adder* **is**
port(*a, b*: **in** *Std_Logic_Vector* (3 **downto** 0);
c: **out** *Std_Logic_Vector* (3 **downto** 0) := "0000");
end entity *adder*;

2.2 Архітектура

Після описання інтерфейсу об'єкта необхідно описати його архітектуру (його роботу). Архітектурою називається внутрішня структура об'єкта, що визначає його поведінку. Саме архітектура визначає яким чином вхідні порти об'єкта поєднуються з вихідними. Синтаксис архітектури:

architecture *identifier* **of** *entity_name* **is**
Block_Declarative_Item
begin
Concurrent_Statements
end architecture *identifier*;

Тут *identifier* – ім'я архітектури; *entity_name* – ім'я об'єкта, поведінку якого описує дана архітектура; *Block_Declarative_Item* – список елементів програми, що будуть доступні всім елементам в межах архітектури. В цьому розділі можна об'являти константи, типи, компоненти та сигнали.

Поведінка об'єкта описується всередині архітектури за допомогою конкурентних операторів (*Concurrent_Statements*). В послідовних мовах програмування, таких як Паскаль та Сі, поняття конкурентних операторів відсутнє. Ці оператори призначаються для реалізації паралельних процесів і виконуються одночасно. До конкурентних операторів відносяться процеси (**process**), компоненти та деякі інші.

2.3 Сигнали

Сигнали служать для зв'язків між різними об'єктами (**entity**) та процесами, що протікають в цих об'єктах. Перед застосуванням сигнал необхідно попередньо оголосити. Синтаксис оголошення сигналу має вигляд:

signal *Ідентифікатор*: *Тип_Сигналу* := *Початкове_Значення*;

Тут *Ідентифікатор* – ім'я сигналу, яке сформоване за правилами формування ідентифікаторів; *Тип_Сигналу* – може задаватися будь-яким типом чи підтипом; *Початкове_Значення* – значення сигналу в момент часу 0 с. Початкове значення може не задаватися явно, в такому випадку сигнал набуває крайнього лівого значення підтипу *Тип_Сигналу*.

Кілька однотипних сигналів з однаковим початковим значенням можуть оголошуватися одним оператором **signal**. У цьому випадку їх ідентифікатори перераховуються через кому, наприклад:

signal *S1, S2, S3*: *Std_Logic* := '1';

Оператори оголошення сигналів можуть розміщуватися в розділі оголошень архітектури, тоді даний сигнал буде доступний всім процесам поточної архітектури. Крім того, сигнал можна оголосити в розділі оголошень **entity**, у цьому випадку сигнал буде доступний всім архітектурам даного об'єкту.

2.4 Оператор напряму сигналу, способи реалізації затримки в VHDL

2.4.1 Оператор напряму

Для зміни значення сигналу використовується оператор напряму, його синтаксис має вигляд:

Target_Signal <= *Source* **after** *delay*;

Тут *Target_Signal* – ідентифікатор сигналу, значення якого має змінитися; *Source* – джерело нового значення (може застосовуватися змінна, константа, вираз або сигнал); *delay* – часова затримка.

Мова VHDL є спеціалізованою мовою для описання та моделювання пристроїв, реальні сигнали в яких не можуть змінювати свого значення миттєво. Для врахування цієї особливості в VHDL застосовується механізм *транзакцій.*

Транзакцією називається відкладена дія призначення сигналу. Кожна транзакція містить ім'я сигналу, що буде змінюватись, нове значення сигналу та значення модельного часу, в якому буде проведено зміну значення сигналу.

Отже коли при моделюванні об'єкта зустрічається оператор напряму сигналу, то його значення не змінюється. Замість цього формується трансакція, в яку заноситься ім'я сигналу (*Target_Signal*), нове значення (*Source*) та значення модельного часу зміни сигналу, що формується як сума поточного значення модельного часу та часової затримки *delay* (у випадку, коли *delay* не задано явно, як затримка використовується крок модельного часу). Сформована транзакція додається до списку транзакцій, який перевіряється системою кожного разу при зростанні модельного часу. Якщо ж настає момент зміни сигналу, то йому присвоюється нове значення. Зміна значення сигналу називається *подією* на сигналі. Такий алгоритм призводить до того, що сигнал не змінює свого значення миттєво, і мінімальна часова затримка дорівнює кроку модельного часу.

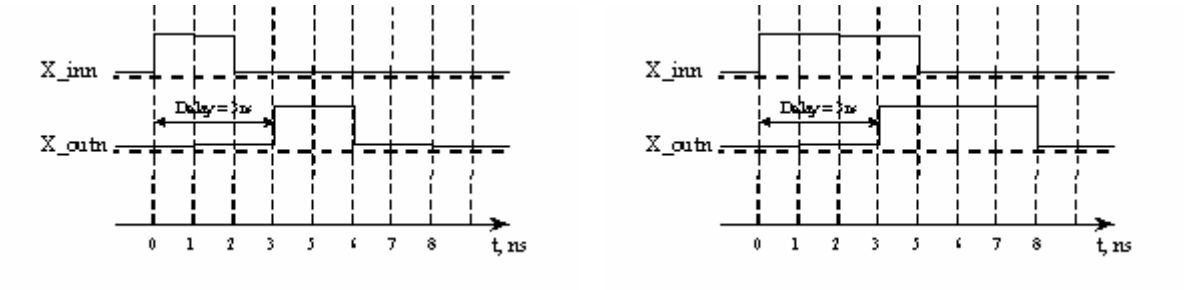
Для точнішого різноманітних умов передачі сигналів в реальних об'єктах в VHDL введено два типи затримки: транспортна та інерційна.

2.4.2 Транспортна затримка сигналу

Транспортна затримка сигналу (ТЗ) в загальному випадку моделює просту затримку в передачі вхідного сигналу на чітко визначений час. В загальному вигляді в VHDL транспортна затримка може бути описана, як

X_out <= **transport** X_in **after** time_value;

Приклад (рис. 2.1) показує, як впливає наявність ТЗ на вихідний сигнал при різній тривалості вхідного сигналу.



а) б)

Рисунок 2.1 – Транспортна затримка

а) Тривалість вхідного сигналу *X_in* менше значення затримки 3 ns

X_out = **transport** *X_in* **after** 3 ns;

б) Тривалість вхідного сигналу *X_in* більше або дорівнює значенню затримки 3 ns

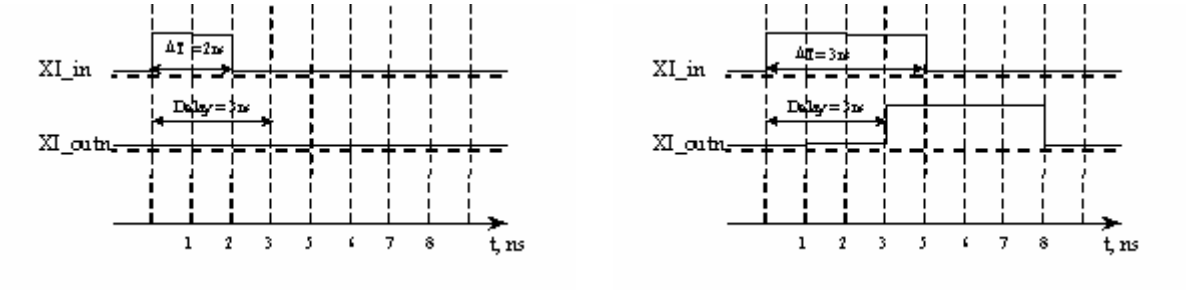
X_out = **transport** *X_in* **after** 3 ns;

2.4.3 Інерційна затримка

Інерційна затримка дозволяє моделювати роботу таких елементів схеми, які не пропускають коротких імпульсів. Тобто, якщо який-небудь з елементів схеми має інерційну затримку T (як, наприклад, логічні елементи ТАК, АБО, НІ), то вхідний сигнал буде затримуватися на час T, однак імпульси, тривалість яких менше T, передаватися взагалі не будуть. Загальна форма запису сигналу з інерційною затримкою має вигляд:

XI_out <= **inertial** *XI_in* **after** *time_value*;

Наступний приклад (рис. 2.2) демонструє роботу пристрою з інерційною затримкою при різній тривалості вхідного сигналу:



а) б)

Рисунок 2.2 – Інерційна затримка

а) Тривалість вхідного сигналу менше значення затримки

XI_out <= **inertial** XI_in **after** 3 ns;

б) Тривалість вхідного сигналу більше або дорівнює значенню затримки

XI_out <= **inertial** *XI_in* **after** 3 ns;

Відзначимо, що часто у реальних пристроїв мінімальна тривалість імпульсів, які повинні пропускатися, менша значення інерційної затримки.

У цьому випадку форма запису даного типу затримки зміниться таким чином:

XI_out <= **reject** *impuls_length* **inertial** *XI_inertial* **after** *time_value*;

Приклад:

XI_out <= **reject** 1 ns **inertial** *XI_in* **after** 3 ns;

2.5 Процес

Одним з ключових елементів мови VHDL є процес (**process**). Процес є конкурентним оператором, тому поміщається в розділ *Concurrent_Statements* архітектури. Синтаксис процесу має вигляд:

Мітка_процесу: **process** (*Список_Чутливості*) **is**
Розділ_Оголошень;
begin
Послідовні_Оператори;
end process *Мітка_процесу*;

В наведеному прикладі *Мітка_Процесу* – сформований за правилами ідентифікатор процесу; *Розділ_Оголошень* – розділ, в якому можна оголошувати локальні (доступні лише в поточному процесі) константи, змінні, підтипи, а також підпрограми (процедури та функції); *Список_Чутливості* – список сигналів, виникнення події на яких (зміна сигналу) викличе запуск процесу. *Списку_чутливості* може не бути, але в цьому випадку обов'язкова наявність у процесі хоча б одного оператора **wait** (розглядаються нижче); *Послідовні_Оператори* – оператори, що виконуються (на відміну від конкурентних) послідовно один за одним. До послідовних операторів відносяться: умовний оператор, оператор вибору, циклічні оператори, оператор присвоювання, оператор пересилання значення сигналу та інші. Для аналогії зазначимо, що всі оператори таких мов як Паскаль та Сі є послідовними.

Розглянемо роботу процесів на прикладі RS – тригера:

```
entity RSTrigger is
port( S, R: in Std_Logic;
D, Inv_D: out Std_Logic;
end entity RSTrigger;
architecture RSTrigger of RSTrigger is
 begin
   Flip_Flop: process (R,S) is
      begin
      if S = '1' and R = '0' then
      D<='1';
      Inv_D<='0';
      elseif S = '0' and R = '1' then
      D<='0';
      Inv_D<='1';
      else
```

```
        D<='Z';
       Inv_D<='Z';
       end if;
    end process Flip_Flop;
end architecture RSTrigger;
```

Робота цього процесу протікає таким чином: процес очікує появи події (зміни сигналу) на сигналах *R* чи *S.* Після цього перевіряються значення вхідних сигналів, якщо *R* ='0', *S* = '1', то на вихідний сигнал *D* подається '1', а на інверсний вихідний сигнал подається '0', якщо *R* ='1', *S* = '0', то на вихідний та на інверсний вихідний сигнали відповідно подаються '0' та '1'. У всіх інших випадках вихідний сигнал RS-тригера невизначений, тому на його виходи подається сигнал 'Z' (високий імпеданс).

2.6 Оператор Wait

Аналіз основних класів електронних пристроїв показує, що вони, як правило, мають безперервний цикл роботи. Тобто, після ініціалізації вони виконують визначені для них операції, а потім переходять в режим очікування відповідної події запуску. Іншими словами, пристрої припиняють свою роботу після завершення поточних задач і поновлюють її знову, як тільки виникає необхідна подія. Дана властивість систем описується в VHDL за допомогою службового слова **wait,** яке визначає не тільки момент розриву послідовності операцій, але і умови, що необхідні для їх поновлення. Аналогічні функції виконує *Список_чутливості* процесу, тому один і той самий процес не може містити одночасно *Список_ чутливості* і оператор **wait.** При цьому процес, в який входить вираз, що містить **wait,** має вигляд:

```
process is
begin
sequential_statement ;
wait_statement;
sequential_statement;
wait_statement;
...
end process;
```

Даний процес виконує послідовність дій sequential_statement доти, доки не зустріне оператор **wait** (*wait_statement*), який припиняє виконання процесу до виникнення події, що зазначена в параметрах оператора **wait**.

Є три основних і один додатковий типи **wait**:

1. **wait on *sensitivity_list*** – очікування здійснюється доти, доки не зміниться який-небудь з сигналів в *Списку_чутливості* (sensitivity_list).

Приклад:

wait on CLK;

wait on Ain, Bout;

2. **wait for *time_expression*** – очікування здійснюється поки не мине час, що вказаний в time_expression.

Приклад:

wait for 5 ns;

wait for CLKperiod/2;

3. **wait until *boolean_expression*** – очікування виконання умови, що вказана в boolean_expression.

Приклад:

wait until *CLK*='1';

wait until *IntData*>15;

4. додатковий тип можна визначити як *комбінований тип* запису припинення процесу, який може складатися із двох або трьох різних форм оператора wait.

Приклад:

wait on A **until** CLK='1';

wait until CLK='1' for 10 ns;

Окрім наведених вище типів оператора **wait**, треба відзначити, що **wait** може використовуватись і без параметрів, наприклад:

process is
begin
list_of_statements;
wait;
end process;

У цьому випадку процес почне свою роботу відразу після початку моделювання і зупиниться, коли будуть виконані всі операції з *list_of_statements*. Такий оператор **wait** без параметрів корисний при створенні випробувальних стендів і використовується для формування задавальних впливів.

2.7 Моделювання в VHDL

Розглянемо детально процедуру моделювання об’єктів в VHDL. Моделювання поведінки об’єктів в ЕОМ передбачає розрахунок значень сигналів об’єкта (вхідних, вихідних та внутрішніх) через рівні (дискретні) проміжки часу. Такі проміжки часу називаються кроком або циклом моделювання. Зрозуміло, що адекватність результатів моделювання реальним процесам обернено пропорційна величині циклу моделювання.

Нижче наведено основні етапи процедури моделювання об’єкта в VHDL.

1. Ініціалізація моделювання:
   - поточне значення *модельного часу* встановлюється в 0;
   - всім сигналам об’єкта присвоюються початкові значення.

2. Запуск на виконання всіх процесів.

Процеси виконуються до тих пір, доки не дійдуть до свого кінця (**end process**) або не зустрінуть оператор **wait**. Лише після зупинки всіх процесів система може перейти до наступного етапу.

3. Модельний час збільшується на цикл Моделювання.
4. Перевіряється список транзакцій, якщо є транзакції, що настроєні на поточний *модельний час*, то відповідні сигнали набувають нових значень.
5. Перевіряються всі оператори **wait**, що задіяні в даний момент, якщо є оператори, для яких виконуються умови запуску, то запускаються відповідні процеси.
6. Якщо *модельний час* не досягає значення Кінець моделювання, то система повертається до п. 3.

2.8 Компонування складних об'єктів із структурних складових

При описанні складних об'єктів часто виникає необхідність в їх декомпозиції (розбиттю на структурні складові). Крім того, часто буває корисною можливість повторного застосування типових елементів об'єкта (наприклад, таких як лічильники, суматори, мультиплексори тощо). Мова VHDL надає програмісту можливості ефективно розв'язувати вказані задачі за допомогою механізму компонентів.

Один з можливих шляхів компонування складного об'єкта з уже розроблених елементів – застосування оператора включення. Цей оператор є конкурентним, отже розміщується в тілі архітектури (після слова **begin**). Синтаксис оператора включення має вигляд:

Instance_Name: **entity** *work.Entity_Name*
port map(*S1=>A, S2=>B, S3=>C*);

Тут *Instance_Name* – ім'я включення, *work* – ім'я бібліотеки (ім'я *work* вказує на поточний проект); *Entity_Name* – ім'я об'єкта, що включається. Після ключових слів **port map** в дужках необхідно вказати, які саме сигнали поточного об'єкта підключаються до входів та виходів об'єкта, що включається (в розглянутому прикладі *A,B,C*– сигнали складного об'єкта; *S1,S2,S3* – сигнали компонента).

Розглянемо приклад. Необхідно спроектувати арифметико-логічний пристрій (АЛП), при чому, уже розроблено пристрої, що реалізують арифметичні операції. Описання інтерфейсів цих пристроїв наведено нижче:

```
entity adder is -- Суматор
port( a, b: in Std_Logic_Vector (7 downto 0);
c: out Std_Logic_Vector (7 downto 0) := "00000000");
end entity adder;

entity mul is -- Помножувач
port( a, b: in Std_Logic_Vector (7 downto 0);
```

```
 c: out Std_Logic_Vector (7 downto 0) := "00000000");
end entity mul;

entity divider is -- Поділювач
port( a, b: in Std_Logic_Vector (7 downto 0);
c: out Std_Logic_Vector (7 downto 0) := "00000000");
end entity divider;
```

Програма, що реалізує арифметико-логічний пристрій:

```
entity ALU is
port( X, Y: in Std_Logic_Vector (7 downto 0);
operation: in Std_Logic_Vector (1 downto 0);
en: in Std_Logic;
Z: out Std_Logic_Vector (7 downto 0) := "00000000");
end entity ALU;

architecture ALU of ALU is
signal S1, S2, S3: Std_Logic_Vector (7 downto 0);
begin
   Op1: entity work.Adder
   port map(a=>X, b=>Y, c=>S1);
   Op2: entity work.Mul
   port map(a=>X, b=>Y, c=>S2);
   Op2: entity work.Divider
   port map(a=>X, b=>Y, c=>S3);
   process (En) is
      begin
      case Op is
      when "00" => Z <= S1;
      when "01" => Z <= S2;
      when "10" => Z <= S3;
      when others => Z <= "ZZZZZZZZ";
      end case;
      end process;
end architecture ALU;
```

На момент компіляції наведеної програми необхідно, щоб програми, які описують окремі елементи, уже були скомпільовані в окремих файлах і, якщо це необхідно, були б підключені відповідні бібліотеки.

2.9 Константи Generic

При компонуванні складних об'єктів часто виникає необхідність у зміні параметрів компонентів. Можливість такої зміни *без перекомпілювання* компонента надає застосування констант **generic**. Ці

константи розміщуються в розділі описання об'єкта **entity**, їх синтаксис наведено нижче:

```
entity Entity_Name is
generic (a,b,c: integer range 0 to 255; d: STD_LOGIC);
port( port_list );
end Entity_Name;
```

Як видно з наведеного прикладу, константам, описаним у розділі **generic,** не присвоюється конкретне значення на етапі компіляції об'єкта, однак вирази, що містять ці константи, будуть локально-статично визначеними. Це дає можливість застосовувати константи **generic** для задання діапазонів індексів масивів, підтипів тощо. Конкретне значення цим константам присвоюється при включенні об'єкта як компонента до складу складного об'єкта, наприклад:

```
entity Entity_Name is
generic (a,b,c: integer range 0 to 255; d: STD_LOGIC);
port( port_list_1 );
end Entity_Name;

entity Main_Entity is
port( port_list_2 );
end Main_Entity;

architecture Main_Entity of Main_Entity is
begin
   INST0: entity work.Entity_Name
   generic map(a=> 10, b=> 125, c=> 0, d => '1');
   port map( port_association_list );
   INST1: entity work.Entity_Name
   generic map(a=> 2, b=> 3, c=> 4, d => '0');
   port map( port_association_list );
   …
end Main_Entity;
```

Розглянемо застосування **generic** на прикладі реалізації універсального паралельного регістра. Програма, що реалізує 8-бітовий паралельний регістр має вигляд:

```
entity Reg is
port(
X : in STD_LOGIC_VECTOR(7 downto 0);
CLK: in STD_LOGIC;
WE : in STD_LOGIC;
Y: out STD_LOGIC_VECTOR(7 downto 0)
);
end Reg;
```

```
 architecture REG of Reg is
begin
   process (CLK) is
   variable U: STD_LOGIC_VECTOR (7 downto 0);
      begin
      if (WE = '1' ) then
      U := X;
      end if;
      Y<=U;
      end process;
end REG;
```

Щоб реалізувати універсальний регістр, необхідно дати користувачу можливість змінювати розрядність регістра без створення нових **entity** та без перекомпіляції регістра. Здійснимо це за допомогою включення необхідної константи:

```
entity Reg is
Generic (N: integer range 1 to 63);
port(
X : in STD_LOGIC_VECTOR(N downto 0);
CLK: in STD_LOGIC;
WE : in STD_LOGIC;
Y: out STD_LOGIC_VECTOR(N downto 0)
);
end Reg;

architecture REG of Reg is
   begin
      process (CLK) is
      variable U: STD_LOGIC_VECTOR (N downto 0);
         begin
         if (WE = '1' ) then
         U := X;
         end if;
         Y<=U;
         end process;
end REG;
```

Тепер для включення 8-бітового регістра треба записати такий оператор включення компонента:

```
REG8: entity work.Reg
generic map(N=>7);
port map(X=>S1,CLK=>S2,WE=>S3,Y=>S4);
```

Для 16-бітового:

```
REG16: entity work.Reg
generic map(N=>15);
```

port map(*X=>S1,CLK=>S2,WE=>S3,Y=>S4*);

2.10 Паралельна робота процесів

Головною особливістю процесів є їх здатність працювати паралельно. Тобто, якщо всередині архітектури міститься два або більше процесів, *Списки_чутливості* яких містять один і той самий сигнал, то при зміні значення цього сигналу всі процеси запускаються одночасно. При цьому дії всередині кожного з них виконуються послідовно і незалежно від інших паралельних процесів.

Для наочності розглянемо приклад, зображений на рис.2.3. Якщо змінити значення сигналу A (змінені сигнали будемо позначати індексом (*)), два процеси P1 і P2 одночасно почнуть свою роботу. Внаслідок їх виконання зміняться значення сигналів D і E, які, в свою чергу, входять в *Списки_чутливості* процесів P3 і P1. В результаті роботи процесу P1 змінюється сигнал D, що призводить до нової ініціалізації процесу P3. Після виконання процесу P3 жодному з сигналів в розглянутих *Списках_чутливості* не присвоюється нове значення, тобто, всі процеси припиняються в очікуванні чергової зміни одного з сигналів із *Списків_чутливості*.

Тут слід відзначити важливий момент: якщо внаслідок виконання процесу P1 сигналу D присвоюється таке саме значення, яке у нього було до ініціалізації процесу P1, то в цьому випадку процес P3, в *Списку_чутливості* якого знаходиться сигнал D, не поновлюється.

2.11 Перекриття сигналів

Коли мова йде про паралельну роботу процесів, звичайно виникає питання про те, що станеться з сигналом, якщо внаслідок роботи двох або більше процесів йому будуть одночасно присвоєні різні значення. Іншими словами, стоїть задача визначення сигналу, який має два і більше джерел. Для цього в VHDL існує так званий метод перекриття сигналів, принцип якого полягає в “змішуванні” сигналів різного рівня відповідно до функції перекриття.

Функція перекриття задається при описанні підтипу і визначає закон формування результувального значення сигналу для всіх можливих комбінацій сигналів джерел. Для моделювання реальних сигналів у цифрових пристроях в стандартному VHDL створено тип *std_logic*. Сигнал такого типу може набувати одного з дев’яти рівнів, а саме:

‘U’ – невизначений;
‘X’ – сильний невідомий (або ‘0’, або ‘1’);
‘0’ – сильний нуль;
‘1’ – сильна одиниця;
‘Z’ – високий імпеданс;
‘W’ – слабкий невідомий;
‘L’ – слабкий нуль;

‘H’ – слабка одиниця;

‘-’ – байдужий стан.

Типи Std_Logic та Std_Logic_Vector (масив елементів типу Std_Logic) описано у пакеті Std_Logic_1164 бібліотеки IEEE, що обумовлює необхідність звернення до нього перед початком описання об’єкта, що буде використовувати такі типи. Синтаксис звернення має вигляд:

Library *IEEE*;

Use *IEEE.Std_Logic_1164*.**all**;

Принцип перекриття таких сигналів подано на рис. 2.4.

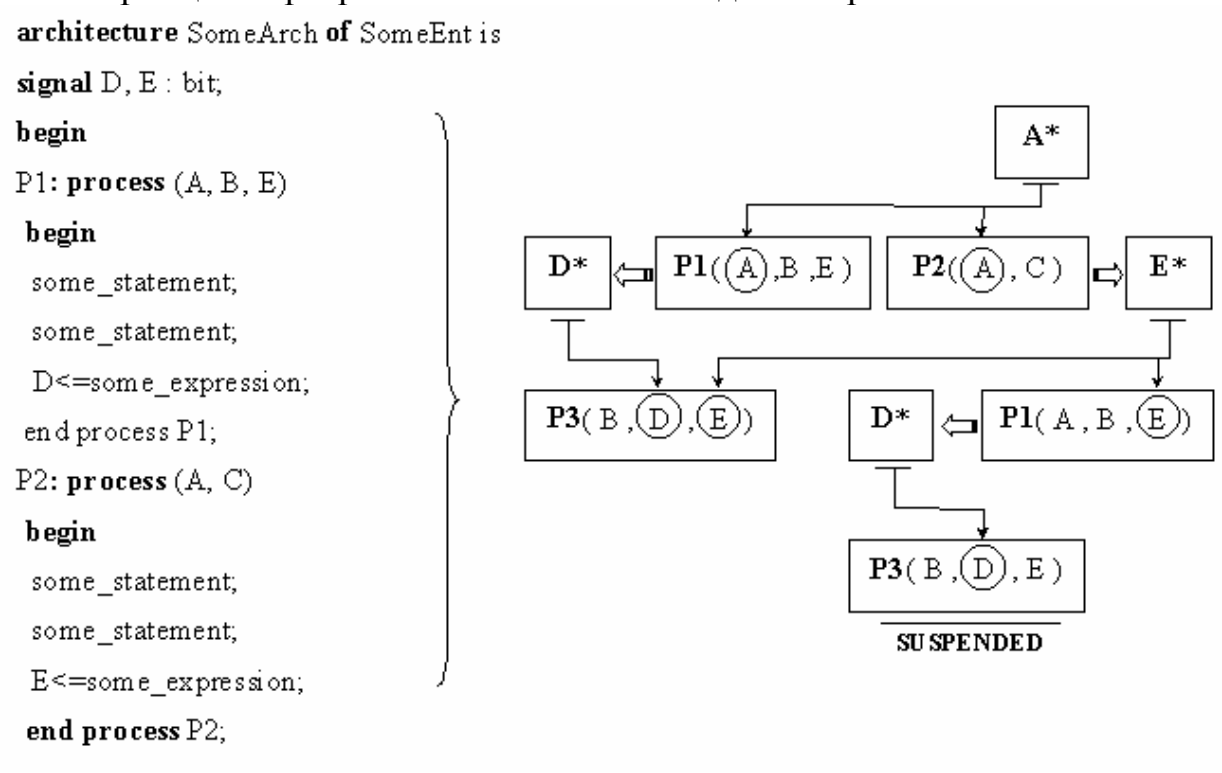


Рис. 2.3 – Паралельна робота процесів

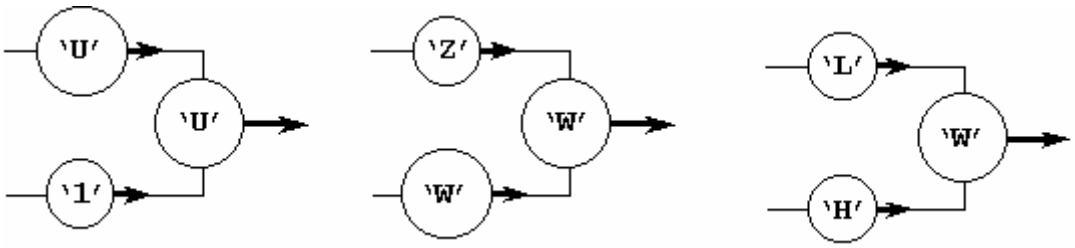


Рис. 2.4 – Перекриття сигналів типу *Std_Logic*

Повний перелік усіх можливих комбінацій подано у таблиці 2.1.

Таблиця 2.1 - Перекриття сигналів типу *Std_Logic*

	U	**X**	**0**	**1**	**Z**	**W**	**L**	**H**	**-**
U	‘U’	‘U’	‘U’	‘U’	‘U’	‘U’	‘U’	‘U’	‘U’
X	‘U’	‘X’	‘X’	‘X’	‘X’	‘X’	‘X’	‘X’	‘X’
0	‘U’	‘X’	‘0’	‘X’	‘0’	‘0’	‘0’	‘0’	‘X’
1	‘U’	‘X’	‘X’	‘1’	‘1’	‘1’	‘1’	‘1’	‘X’
Z	‘U’	‘X’	‘0’	‘1’	‘Z’	‘W’	‘L’	‘H’	‘X’
W	‘U’	‘X’	‘0’	‘1’	‘W’	‘W’	‘W’	‘W’	‘X’
L	‘U’	‘X’	‘0’	‘1’	‘L’	‘W’	‘L’	‘W’	‘X’
H	‘U’	‘X’	‘0’	‘1’	‘H’	‘W’	‘W’	‘H’	‘X’
-	‘U’	‘X’	‘X’	‘X’	‘X’	‘X’	‘X’	‘X’	‘X’

Треба зазначити, що серед стандартних типів операція перекриття у VHDL визначена тільки для типів *Std_Logic* та *Std_Logic_Vector*. Якщо це необхідно, VHDL дозволяє користувачеві сформувати свою функцію перекриття для будь–яких створених ним типів сигналів.

2.12 Декларування компонентів

Якщо сутності та архітектури компонентів, з яких складається елемент, відкомпільовані і записані до бібліотеки, то декларувати ці компоненти в архітектурі нема необхідності, так як робоча бібліотека включена в проект (всі файли проекту за замовчуванням).

Якщо ж цього не зроблено, то є можливість дати опис компонентів у архітектурі елементу, де вони використовуються. Перед цим необхідно вказати в файлі назву бібліотеки, в якій розміщується компонент. Це робиться за допомогою ключового слова **library** за зразком: **Library** <Назва_бібліотеки>. Декларація компонента дуже подібна до декларації сутності:

component *<ім'я компонента>* **is**
generic *(<список узагальнень>);*
port *(<список портів>);*
end component *<ім'я компонента>;*

Нижче наведено приклад декларування двох компонентів: двовходового диз'юнктора *Logic_OR* та елемента постійної пам'яті *ROM*:

component *logic_or* **is**
generic *(delai: time:=1ns)*
port *(x1,x2:* **in** *bitl; y:* **out** *bitl);*
end component *logic_or;*
component *rom*
generic *(data_bits, addr_bits: positive);*
port *(en:* **in** *bit; addr:* **in** *bit_vector(addr_bits-1* **downto** *0);*
data: out bit_vector (data_bits -1 downto 0));
end component rom*;*

Зазначимо, що часова затримка у диз'юнкторі та розрядність шини адреси і даних визначається тут константами у списку узагальнень.

2.13 Використання компонентів

Щоб використати задекларовані компоненти у архітектурі користуються такою конструкцією:

<мітка> : <ім'я_компоненту>
generic map *<список констант>;*
port map *<список сигналів>;*

Наприклад, фрагмент архітектури, що використовує описані вище компоненти, може виглядати так:

signal *en1, en2, rom_sel : bit, a: bit_vector(31* **downto** *0),*
d: bit_vector(15 **downto** *0);*
enable_data: logic_or
port map *(en1, en2, rom_sel);*
parametr_rom: rom
generic map *(data_bits =>16, addr_bits =>32);*
port map*(en =>rom_sel, data =>d(15* **downto** *0), addr =>a(31* **downto** *0));*

З цього опису випливає, що внутрішні сигнали *en1, en2* типу *bit* підключені до входів диз'юнктора *logic_or*, вихід якого *rom_sel* з'єднаний з входом *en* елементу пам'яті *rom*.

При застосуванні диз'юнктора *logic_or* оператор **generic map** опущено. Отже, у цьому елементі затримка сигналу *delai* залишається такою, якою вона була вказана у декларації компонента, тобто 1 нс. Оператор **generic map** задає розрядність шини адреси *addr_bits* і даних.

3 СТРУКТУРА МОВИ VHDL

3.1 Умовний оператор

Умовний оператор дозволяє реалізувати розгалужені обчислювальні процеси. Для умовних операторів в мові VHDL можливий будь-який ступінь вкладеності . Синтаксис умовного оператора має вигляд:

if *boolean_expression_1* **then**
послідовність_операторів_1
else if *boolean_expression_2* **then**
послідовність_операторів_2
else
послідовність_операторів_3
end if;

Виконується умовний оператор таким чином: якщо результатом обчислення *boolean_expression_1* є "**Істина**", то виконується *послідовність_операторів_1*, інакше обчислюється *boolean_expression_2*. Якщо результат обчислення "**Істина**", то виконується *послідовність_операторів_2* , інакше виконується *послідовність_операторів_3*.

3.2 Оператор вибору

Оператор вибору дозволяє вибрати одне з кількох можливих продовжень програми. Формат запису оператора вибору в VHDL наведено нижче:

case *expression* **is**
when *choices_1* **=>** послідовність_операторів_1;
when *choices_2* **=>** послідовність_операторів_2;
when others => послідовність_операторів_3
end case;

Тут *expression* – змінна (вираз) порядкового типу;

choices_1, *choices_2* – локально-статично визначені вирази того самого типу, що й *expression*.

Локально-статично визначеним називається вираз, значення якого може бути обчисленим на етапі компіляції.

Оператор вибору працює таким чином: якщо *choices_1* містить значення змінної *expression,* то виконується послідовність_операторів_1, якщо *choices_2* містить значення змінної *expression,* то виконується послідовність_операторів_2. У всіх інших випадках виконується послідовність_операторів_3.

Приклад:

variable *k***:** *integer* **:=** 3;
variable *a***:** *integer***;**

…

```
case k is
when 1 => a := 5;
when 3 => a := 6;
when 2 => a := 7;
when others a := 8;
end case;
```

В розглянутому прикладі в ролі *expression* виступає змінна *k*, так як *k* = 3, то змінна *a* набуває значення 6.

У випадку, коли альтернативи *choices* не покривають всього діапазону можливих значень *expression,* наявність альтернативи **when others** є обов'язковою.

Альтернативи *choices* можуть задаватися не тільки одиночними значеннями, а й діапазонами, наприклад:

```
when 'a' to 'c' => …
when 0 to 15 => …
```

Як альтернативи складається з декількох виразів, то вони записуються через знак '|' (об'єднуються за операцією "**or**"), наприклад:

```
when 1|2|5|8 => …
when 1 to 7|18 to 25 => …
```

В якості альтернативи можна використовувати підтипи, наприклад:

```
subtype New_Integer is integer range 0 to 10;
…
case k is
when New_Integer => …
when New_Integer|11 =>…
end case;
```

3.3 Порожній оператор

Для запису порожнього оператора в мові VHDL використовується ключове слово **null**. Цей оператор може застосовуватися для відлагоджувальних цілей, наприклад:

```
if a < b then
null;
end if;
```

3.4 Цикли

Цикли застосовуються для програмування фрагментів, що повторюються. Базовим для всіх циклічних конструкцій в VHDL є нескінченний цикл.

3.4.1 Нескінченний цикл

Формат запису нескінченного циклу має вигляд:

Loop_Label: **loop**

послідовність_операторів

end loop *Loop_Label*;

Тут *Loop_Label* – мітка циклу (може бути пропущена). Розглянемо фрагмент програми:

Unlimited: **loop**

a:=*a*+1;

end loop *Unlimited*;

Даний приклад буде нескінченно збільшувати змінну *a.*

3.4.2 Оператор Exit

Хоч нескінченні цикли і зустрічаються в практичній роботі, однак набагато частіше виникає необхідність в циклі, який закінчується при виконанні якоїсь умови. Для реалізації таких циклів в VHDL використовується оператор **exit**. Формат його запису:

exit *loop_label* **when** *boolean_expression*;

Його найпростіша форма:

exit;

За цією командою переривається виконання поточного циклу і управління передається на оператор, що є наступним після **end loop**.

В реальних програмах часто виникає необхідність в застосуванні фрагмента,

if *boolean_expression* **then**

exit;

end if;

який перериває виконання поточного циклу при досягненні *boolean_expression* значення **TRUE**. Є спрощена форма запису такого фрагмента:

exit when *boolean_expression*;

Застосовуючи мітки, можна реалізувати вкладені цикли, наприклад:

outer: **loop**

…

inner: **loop**

…

exit *outer* **when** *Умова_1;* -- Вихід з зовнішнього циклу за міткою

…

exit when *Умова_2;* **--** Вихід з поточного

… -- (внутрішнього) циклу

end loop *inner;*

end loop *outer;*

3.4.3 Оператор Next

За допомогою оператора **Next** можна перейти одразу до початку наступної ітерації циклу (тобто на оператор **loop**). Синтаксис оператора **Next** такий самий, як і оператора **Exit:**

next when boolean_expression;

3.4.5 Цикл “Поки” (While)

За допомогою оператора **exit** можна реалізувати будь-який цикл (“Поки”, “До”, з параметром), однак, для спрощення запису та полегшення розуміння програм в мові VHDL введено спеціальні конструкції для основних типів циклічних операторів. Цикл “Поки” записується таким чином:

loop_label: **while** *boolean_expression* **loop**
послідовність_операторів
end loop;

Працює цей цикл так: поки *boolean_expression* має значення **TRUE**, виконується тіло циклу (послідовність_операторів).

Всередині циклу **while** можна використовувати будь-які оператори, включаючи **exit** та **next**.

3.4.6 Цикл з параметром (for)

Крім циклу “Поки” в мові VHDL є також спеціальна конструкція для реалізації циклу з параметром. Його синтаксис:

loop_label: **for** *parameter* **in** *diapason* **loop**
послідовність_операторів
end loop;

Тут *diapason* – діапазон порядкового типу, наприклад :

0 **to** 10
10 **downto** 0.

Діапазон також може задаватися підтипом:

subtype *New_Integer* **is** *integer* **range** 0 **to** 10;
…
loop_with_subtype: **for** *i* **in** *New_Integer* **loop**
…
end loop *loop_with_subtype***;**

Суттєва відмінність мови VHDL від таких мов, як Паскаль та Сі, полягає в тому, що параметр тут не є змінною. Параметр не потрібно попередньо об’являти, всередині циклу параметр може розглядатися як константа (тобто його можна використовувати в виразах, але не можна присвоювати параметру значення), після закінчення циклу параметр зникає, наприклад:

Erorr_Parametr_Demo: **process is**
*variable i,j***:** *integer*;
begin

i:=*loop_par*; --Error (Невідомий ідентифікатор *loop_par*)
for *loop_par* **in** 1 **to** 10 **loop**
j := *loop_par*;
loop_par:=5; --Error (Параметр циклу не можна
--використовувати з оператором присвоювання)
end loop;
j:= *loop_par*; -- Error (Невідомий ідентифікатор *loop_par*)
end *Erorr_Parametr_Demo*;

Рядки, в яких наведено неправильне застосування параметра в даному прикладі, помічено коментарем Error.

3.5 Змінні

В попередньому розділі розглядалося таке поняття, як “сигнал”. Очевидно, що обмеження, які накладаються на використання сигналів (відсутність можливості декларування сигналу і зберігання проміжних даних всередині процесу; нове значення сигналу присвоюється тільки після закінчення роботи процесу), обумовлюють серйозні труднощі щодо їх практичного використання. Таким чином, виникає необхідність в об'єкті, який би компенсував всі відзначені “недоліки” сигналів. У VHDL таким об'єктом є “змінна”.

Змінні можуть бути оголошені тільки всередині процесу, в якому вони використовуються, і є локальними об'єктами цього процесу. Оголошення змінних здійснюється таким чином:

Variable *list_of_var_names*: *type_name* [:= initial_value];

де list_of_var_names – список змінних;

type_name – тип змінних; initial_value – початкове значення.

Вирази, які взяті в квадратні дужки […] є необов'язковими.

Основною функцією змінних є зберігання проміжної інформації всередині процесів і підпрограм (процедур і функцій). Покажемо наочно відмінності, які виникають внаслідок виконання одних і тих самих операцій над сигналами (Приклад а) і над змінними (Приклад б):

Приклад а)

```
Architecture var of EntA is
Signal InS, Sum: integer :=0;
begin
process (InS)
variable var1, var3: integer :=0;
variable var2 : integer :=1;
begin
var1:=var2+1;
var2:=var1-2;
var3:=var1+var2;
Sum<=va1+var2+var3;
End process;
```

Приклад б)

```
architecture var of EntA is
signal InS, Sum: integer :=0;
signal var1, var3: integer :=0;
signal var2 : integer :=1;
begin
process (InS)
begin
var1 <= var2+1;
var2 <= var1-2;
var3 <= var1+var2;
Sum <= var1+var2+var3;
end process;
```

end var;
Результат виконання процесу:
Sum = **4**

end var;
Результат виконання процесу:
Sum = **1**

3.6 Константи

Крім сигналів і змінних, VHDL може оперувати джерелом статичної інформації в рамках окремо взятої архітектури – константою. Оголошення константи виконується в розділі оголошень архітектури, діє тільки в рамках цієї архітектури і має вигляд:

constant *constant_names*: *type_name*: = constant_value;

де *constant_names* – найменування константи;
type_name – тип константи;
constant_value – значення константи.

Зазвичай константи використовуються для:

- призначення розміру складних об'єктів (масивів, шин і т.п.);
- призначення числа ітерацій циклів;
- визначення параметрів часу (тимчасових установок, затримок, моментів перемикання і таке інше):

constant *delay1*: *time:* = 10 ns.

Примітка: Якщо константа оголошена не в архітектурі, а в *процесі*, то вона діє тільки в рамках цього *процесу*.

3.7 Типи в VHDL, перетворення типів

VHDL містить ряд стандартних типів, які визначені в пакетах Standard і Std_Logic_1164. Деякі з них наведено в таблиці 3.1.

Таблиця 3.1 – Стандартні типи

BIT	‘0’ або ‘1’
BOOLEAN	FALSE або TRUE
INTEGER	Цілі числа в діапазоні від $-(2^{31}-1)$ до $+(2^{31}-1)$
REAL	Числа з плаваючою комою в діапазоні від -1.0E38 до +1.0E38.
CHARACTER	Символьний тип
TIME	Цілочисельний тип з розмірністю fs, ps, ns, us, ms, sec, min, hr.

Крім стандартних типів VHDL дозволяє користувачеві визначати типи об'єктів самостійно. Найзагальнішими є, так звані, порядкові типи, в яких визначається повний список значень, що належать даному типу. Так, наприклад, запис:

type *user_type* **is (**V1, V2, V3, V4, V5, V6, V7);
signal *Sig1*: *user_type*: = V1;

визначає деякий сигнал *Sig1*, що може приймати одне з значень, вказаних в дужках *V1*, *V2*, *V3*, *V4*, *V5*, *V6*, *V7*, і набуває початкового значення *V1*.

Якщо початкове значення не вказується явно, то змінна чи сигнал набуває крайнього лівого значення типу. Наприклад, запис

variable *A*: *integer*;

дасть початкове значення змінної *A* = -2 147 483 647.

Відзначимо, що VHDL суворо типізована мова. Таким чином, ні сигнали, ні змінні різних типів звичайно не можуть використовуватися в одному виразі. При цьому, в даній мові опису обладнання не передбачені засоби автоматичного перетворення типів. Так, наприклад, вираз *A*<=*B* **or** *C* правомірний тільки у випадку, коли *A, B* і *C* одного типу або аналогічних типів. Операції над сигналами (змінними) аналогічних типів вимагають їх обов'язкового узгодження. Таким чином, узгодження може виконуватися тільки для сумісних типів і відповідно до таких правил:

Цілі (*integer*) і дійсні (*real*) типи є аналогічними типами. При перетворенні дійсних чисел в цілі результат округляється до найближчого цілого числа.

Два масиви розглядаються як аналогічні, якщо вони:

- мають однакову розмірність;
- елементи обох масивів мають однаковий тип;
- індекси обох масивів мають однаковий тип або їхні типи аналогічні.

Всі інші типи не є аналогічними.

Приведемо приклад узгодження типів:

```
Variable A, B: integer;
...
A:=integer(25.17*real(B));
```

3.8 Масиви

В VHDL масив інтерпретується як тип, значення якого складається з ряду елементів одного підтипу. Кожен елемент масиву відрізняється своїм індексом або рядом індексів (в багатовимірних масивах). Індекс масиву повинен мати порядковий тип і знаходитись в області допустимих значень. Таким чином, перш ніж застосовувати масив в VHDL, необхідно оголосити тип масиву (а), а потім оголосити масив як об'єкт (б):

(а) **type** *array_type_name* **is array** index_range **of** *element_type*;

(б) **variable** *array_name*: *array_type_name* [:= initial_values];

Приклад одновимірного масиву:

```
type A is array (5 downto 0) of bit;
variable B : A := “0 1 1 0 0 1 ”;
```

Для багатовимірного масиву порядок і структура оголошення масиву залишаються такі ж самі:

Приклад:

```
type A is array (1 to 3, 1 to 2) of integer;
variable B : A := ((0 , 1),(5 , 2),(7 , 8));
```

У наведених прикладах замість службового слова **variable** може стояти **signal** або **constant**.

При оголошенні типу масиву дозволяється не визначати його розмірність (а). Такі типи масивів називаються необмеженими. У цьому випадку розмірність вказується при оголошенні масиву, як об'єкта (б):

а) **type** *arr_type* **is array (***index_type* r**ange <>**) **of** elements_type;

б) **variable** *arr_name*: *arr_type* (index_range)[:=initial_val];,

де *index_type* – тип індексів масиву (*integer*, *natural* etc.).

Багатовимірні необмежені масиви і їх типи оголошуються аналогічно:

type *multy* **is array** (*integer* **range** <>, *bit* **range** <>,. .., *boolean* **range** <>) **of** elements_type.

4 РОБОЧЕ СЕРЕДОВИЩЕ Active-HDL

Робоче середовище *Active-HDL* складається з головного вікна програми, в якому відкриваються вікна його компонентів (рис.4.1). Нижче в цьому розділі ми коротко розглянемо призначення основних компонентів, а в наступних розділах - роботу з ними.

4.1 Вікно перегляду проектів - Design Browser

Проект, розроблений у Active-HDL, складається з набору пов'язаних між собою файлів (файлів опису, бібліотечних файлів, файлів результатів моделювання проекту, різних допоміжних файлів тощо). Вікно **Design Browser** (рис. 4.2) є зручним засобом маніпулювання з файлами проекту. У цьому вікні можна додавати чи вилучати файли, отримати інформацію про ієрархічну підпорядкованість об'єктів, а за виглядом піктограми файла можна дізнатись чи компілювався файл після внесення до нього змін і чи була ця компіляція успішною.

Вікно **Design Browser** відкривається і закривається за допомогою одноіменної кнопки на панелі інструментів чи такої ж опції у меню **View**. Воно має три панелі (рис. 4.2).

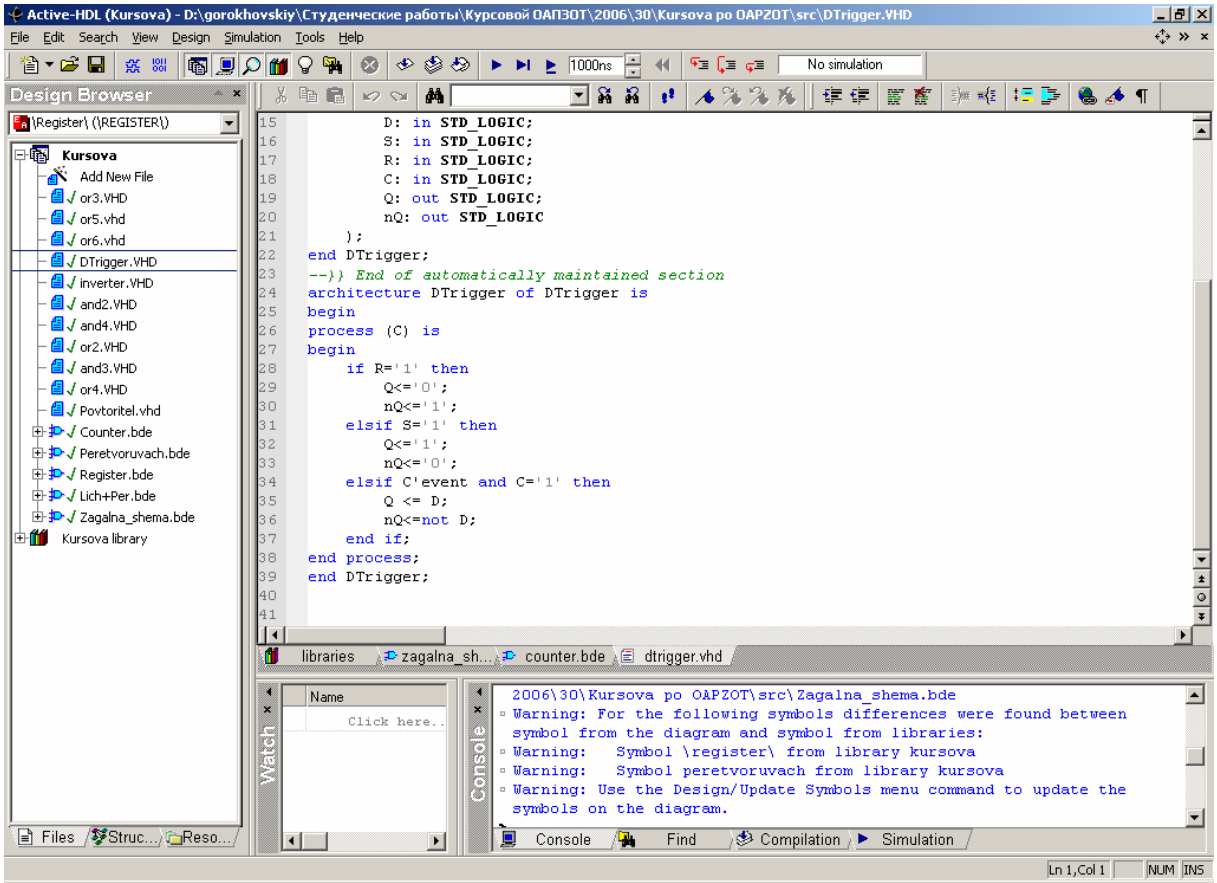
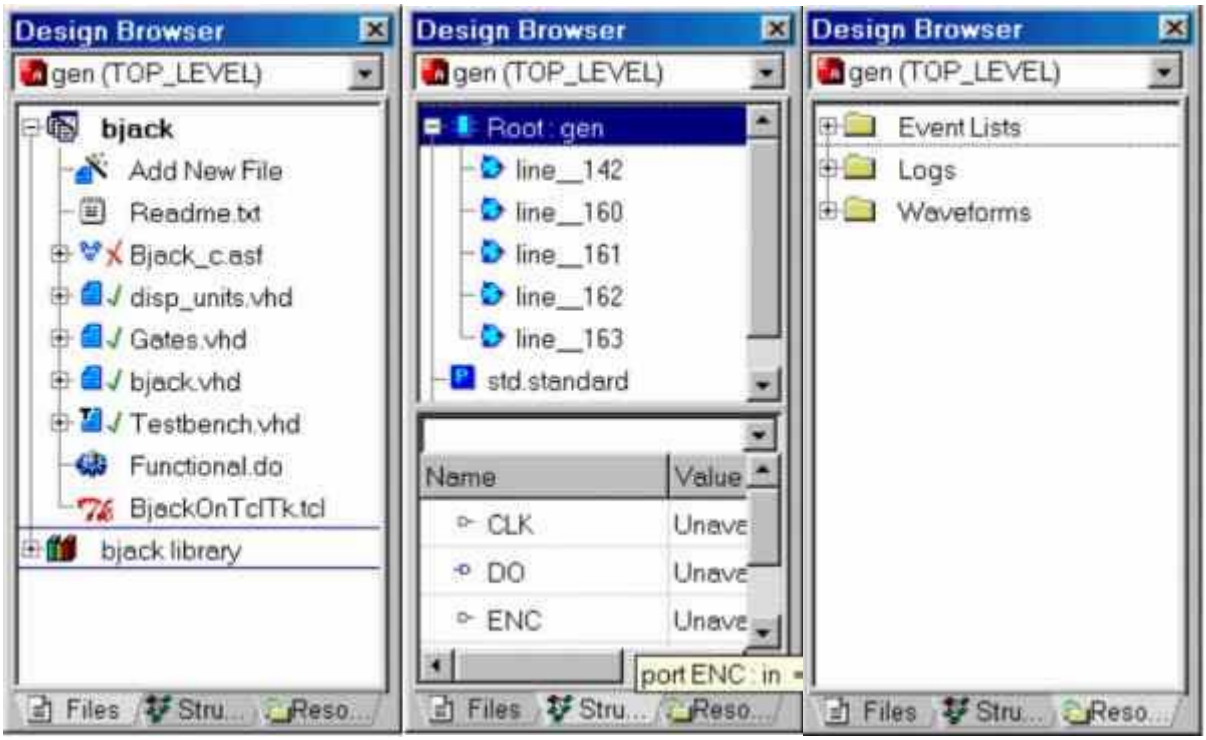


Рисунок 4.1 – Робоче середовище Active-HDL



а) б) в)

Рисунок 4.2 - Вікна **Design Browser**

- · ***Панель файлів*** – *Files* (рис. 4.2,а) показує перелік файлів проекту і зміст робочих бібліотек, а також дозволяє вибрати модуль верхнього рівня зі списку модулів проекту. Іконки файлів несуть інформацію про тип файла: - vhdl-код, - граф автомата, - блок-схема, - файл макрокоманд тощо, а також результати компіляції файла: - успішно скомпільований, - той, що мав помилки компіляції, - попередження компіляції, - той що не компілювався після внесення до нього змін. Під іконкою файла розташовані іконки, що несуть інформацію про його зміст: - опис сутності й архітектури, - опис конфігурації тощо.
- · ***Панель структури*** – *Structure* (рис. 4.2,б) у своїй верхній частині відображає ієрархічну структуру проекту, що складається з модулів . Там же відображаються процеси , описані у модулях. У нижній частині панелі відображається список сигналів і змінних, присутніх у модулі чи процесі, виділеному у верхній частині панелі.
- · ***Панель ресурсів*** – *Resources* (рис. 4.2,в) містить папки з результатами моделювання: *List* - файли списків, створених під час моделювання; *Logs* - файли звітів із вікон *Console, Simulation, Find і Compile; Waveforms* - файли часових діаграм. До них можна долучати й інші папки, наприклад, файли документації.

4.2 Провідник проекту - Design Explorer

Провідник проекту викликається за допомогою опції **Open Design** із меню **File**. Це засіб ефективного управління проектом, що дозволяє не

турбуватися про фізичне розташування файлів проекту. Провідник проекту має функції, що забезпечують створення і вилучення папок, створення нових і копіювання будь-яких проектів у нові папки. Зазвичай, усі проекти знаходяться у папці **Projects**, створеній при інсталяції. Кожний проект поданий у вікні Провідника окремою іконкою (рис. 4.3). Провідник дозволяє групувати їх в окремі папки. Ці папки не мають жодного відношення до папок файлів, у яких записані проекти. У лівій частині вікна відображається ієрархічна структура папок, у правій - вміст поточної папки. Рядок стану показує назву і шлях до файла опису проекту.

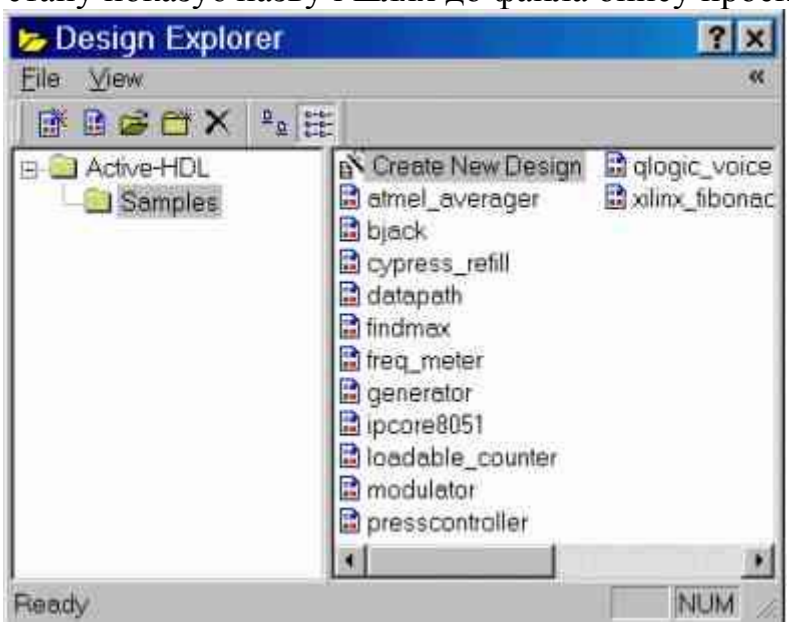


Рис. 4.3 – Вікно **Design Explorer**

4.3 HDL-редактор - HDL-editor

HDL-редактор – це текстовий редактор для вводу і редагування VHDL-файлів. У процесі редагування він розпізнає і автоматично виділяє відповідними кольорами ключові слова, коментарі, ідентифікатори та інші елементи мови VHDL. Це полегшує сприйняття тексту програми та є засобом додаткового синтаксичного контролю.

Роботу у HDL-редакторі полегшують Мовний помічник і Майстер нового файла, про які йдеться нижче.

4.4 Мовний помічник – Language Assistant

Мовний помічник дозволяє прискорити розробку коду на VHDL за рахунок використання шаблонів – готових фрагментів коду, що містять базові конструкції мови VHDL та описи основних функціональних блоків: мультиплексорів, тригерів, лічильників тощо. Мовний помічник містить також шаблон користувача і навчальну програму з декількома простими шаблонами.

4.5 Редактор автоматів – State Diagram Editor

Редактор автоматів – це спеціалізований графічний редактор, призначений для побудови графів скінченних автоматів. Граф автомата транслюється програмою Active-HDL у VHDL-код.

4.6 Редактор блок-схем – Block diagram editor

Редактор блок схем - графічний редактор, що дозволяє описати схему у вигляді елементів (блоків) і зв'язків між ними. Кожний елемент, у свою чергу, може бути описаний у вигляді VHDL-коду, графа автомата чи блок-схеми нижчого ієрархічного рівня. В кінці весь опис транслюється програмою Active-HDL у VHDL-код.

Щоб відкрити вікно HDL-редактора, Редактора автоматів чи Редактора блок-схем для редагування існуючого файла, достатньо двічі клацнути мишкою на піктограмі файла у вікні **Design Browser**.

Для створення і редагування нового файла у тому самому вікні можна скористатися піктограмою **Add New File** або викликати **Майстер нового файла**.

4.7 Майстер нового файла – New Source File Wizard

Цей **Майстер** полегшує створення нового файла у HDL-редакторі, **Редакторі автоматів** чи **Редакторі блок-схем**. При його використанні, початковий етап створення файла зводиться до вводу відповідей на запити **Майстра** про ім'я файла, назву сутності (об’єкту), а також про імена і типи портів. Для виклику **Майстра** можна скористатися опцією **New** з контекстного меню, що викликається натисканням правої кнопки мишки на вільному полі вікна **Design Browser**.

4.8 Редактор часових діаграм – Waveform Editor

Редактор часових діаграм (рис. 4.4) – це графічний редактор, призначений для перегляду і редагування часових діаграм. Зокрема, у цьому редакторі можна створювати форми сигналів-стимулів, що подаються на входи схем при моделюванні, а також переглядати результати моделювання.

При редагуванні часових діаграм можна застосовувати переміщення, копіювання, склеювання тощо. Редактор часових діаграм дозволяє також порівняти між собою два сигнали і виділити кольором їх відмінності.

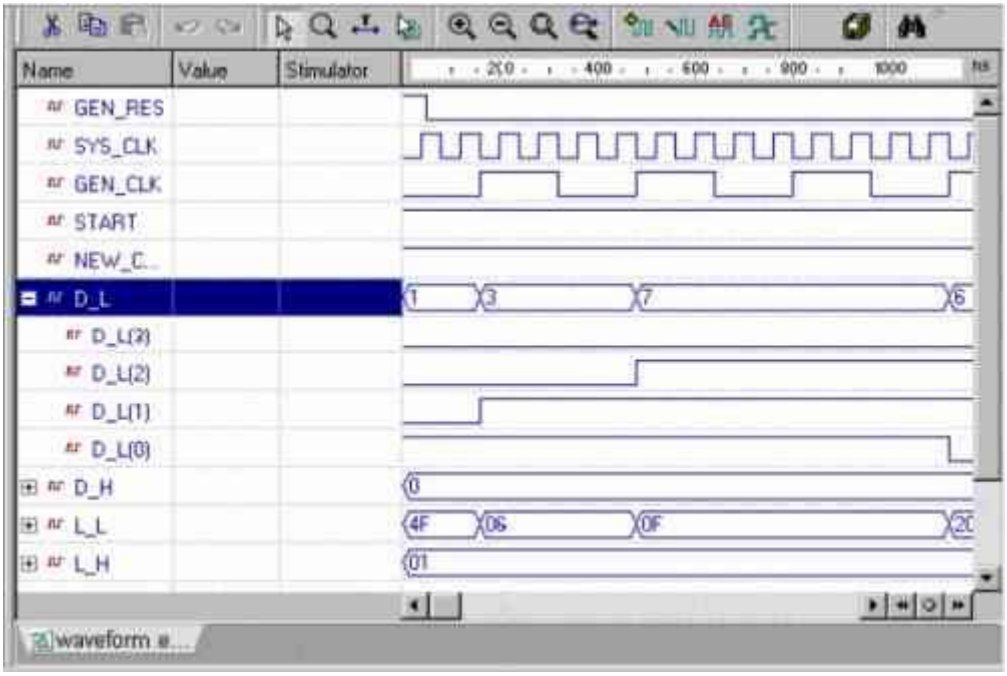


Рисунок 4.4 – Вікно редактора часових діаграм

4.9 Вікна List, Watch, Process

У вікні **List** результати моделювання відображаються у вигляді таблиці, кожний рядок якої описує одну зміну значень сигналів і містить інформацію про час, коли відбулася ця зміна і про значення сигналів, які встановились у результаті.

У вікні **Watch** відображаються поточні значення вибраних користувачем сигналів, час їх встановлення, а також значення, які вони мали до цього моменту.

У вікні **Process** відображається інформація про зв'язок між процесами, та сигналами, які у них беруть участь.

4.10 Вікно Console

Консоль – це вікно для виводу текстових повідомлень програмного середовища Active-HDL, наприклад, повідомлень про виконання і результати компіляції. Сюди ж можна вводити з клавіатури команди управління цим середовищем. Вікно консолі показане у правому нижньому кутку (див. рис. 4.1).

5 СТВОРЕННЯ ПРОЕКТУ

Проектом в програмі Active-HDL називають набір пов'язаних між собою файлів, що описують будову і функціонування схеми, а також забезпечують моделювання її роботи тощо. У кожний момент часу в програмі може бути відкрито не більше одного проекту.

5.1 Використання Майстра нового проекту

Створення проекту: відразу після запуску Active-HDL на екран виводиться вікно діалогу (рис. 5.1), що пропонує відкрити один з існуючих проектів чи створити новий. Вже після запуску програми відкрити інший проект чи створити новий можна за допомогою опцій **Open Design** та **New Design** у меню **File**. Для створення проекту використовується описаний нижче Майстер.

Рисунок 5.1 – Вікно запуску Active-HDL

Майстер нового проекту – це програма, що допомагає створити шаблони файлів проекту та папки, де вони зберігатимуться.

В першому вікні програми (рис. 5.2) треба ввести ім'я створюваного проекту, папку, де він зберігатиметься, а також ім'я робочої бібліотеки. У другому – обирають спосіб створення файлів:

- **Create new Source files now** – створити нові файли зараз;
- **Add existing resources files** – додати існуючі файли проектів;
- **Import design from Active-CAD** – імпортувати проект із Active-CAD;
- **Create an empty design** – створити пусті файли проекту.

Якщо вибрати **Create new Source files now**, то в третьому вікні (рис. 5.2) в колонку **Entity Name** треба ввести імена файлів, що створюватимуться, а у колонці **Source Type** вибрати типи цих файлів:

- **VHDL Code** - програма мовою VHDL;
- **State Diagram** - граф автомата;
- **Block Diagram** - блок-схема;
- **Verilog Source Code** - програма мовою Verilog.

Кнопки **New** та **Delete** служать для додання і вилучення рядків у списку файлів, а кнопка **Ports** відкриває вікно (рис. 5.3), що дозволяє ввести інформацію про порти створюваних файлів.

Кнопки **New** та **Delete** служать для додання і вилучення рядків у списку портів. Кнопка **Type** дозволяє задати тип порту (**bit, std_logic** тощо), у поля **Name** та **Array Index** вводять ім'я та розрядність порту, а на панелі **Port direction** вибирають його режим: **in**, **out**, **input** чи **buffer**.

Рисунок 5.2 – Перше вікно **Design Wizard**

Рисунок 5.3 – Створення шаблонів файлів

В міру введення інформації про порти у лівій частині вікна формуються їхні графічні зображення (рис.5.4).

У останньому вікні **Майстра нового проекту** виводиться список підготовлених ним шаблонів файлів. При потребі можна повернутись до виконання попередніх етапів роботи з **Майстром** або завершити створення нового проекту, натиснувши кнопку **Готово**, або відмовитись від створення проекту і завершити роботу з **Майстром**, натиснувши кнопку **Отмена**.

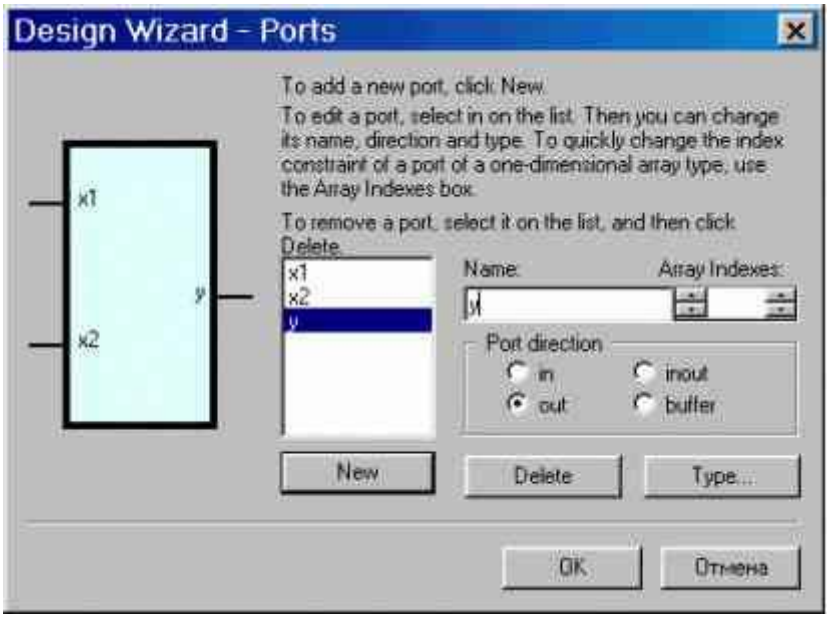


Рисунок 5.4 – Визначення портів

На завершення зазначимо, що створені за допомогою даного **Майстра** файли поки що містять лише декларації сутностей (опис інтерфейсів об'єктів), портів і пусте тіло опису архітектури. Для подальшої роботи з проектом і внесення змін і доповнень у сформовані **Майстром** файли застосовуються відповідні редактори, а саме, текстовий HDL-редактор та графічні – **Редактор автоматів** і **Редактор блок-схем**.

5.2 Використання Майстра нового файла – New Source File Wizard

Якщо під час створення проекту у вікні **Майстра нового проекту** вказані не всі потрібні вам файли, то у подальшій роботі з проектом додати до нього нові файли можна, двічі клацнувши мишкою на команді **Add New File** у вікні **Design Browser**. В результаті активізується вікно діалогу (рис. 5.5), за допомогою якого ви можете створити новий файл чи додати до проекту існуючий (**Add Existing File**), вказати ім'я нового файла у полі **Name** та обрати його тип:

- **VHDL Code** – програма мовою VHDL,
- **State Diagram** – граф автомата,
- **Block Diagram** – блок-схема,
- **Verilog Source Code** – програма мовою Verilog.

Крім того, тут обирається спосіб створення нового файла: чи це буде пустий файл (**Empty File**), що від початку (з нуля) створюється у відповідному редакторі, чи початковий етап створення файла буде виконуватись за допомогою **Майстра нового файла** (**Wizards**). Останній варіант дозволяє зекономити час і уникнути зайвих помилок, тому користуватись ним рекомендується не лише початківцям, а й досвідченим програмістам.

Майстер нового файла може викликатися також вибором із меню File опції New, після якої відривається меню, що містить, зокрема, і згадані вище типи файлів проекту (**VHDL Code**, **State Diagram**, **Block Diagram**, **Verilog Source Code**).

У першому вікні **Майстра нового файла** треба вказати, чи потрібно додавати створюваний файл до поточного проекту (**Add generated file to the design**).

Якщо створюється графічний файл блок-схеми чи графа автомата, то у наступному вікні вибирають, у який саме текстовий файл (мовою VHDL чи мовою Verilog) він компілюватиметься.

У наступному вікні (рис. 5.6) треба ввести ім'я створюваного файла, а також імена описуваних у ньому сутності й архітектури. Якщо імена сутності та архітектури не вказати, то вони співпадатимуть з ім'ям файла.

Останнім вікном **Майстра нового файла** є описане у попередньому розділі вікно (див. рис. 5.4) для введення інформації про порти.

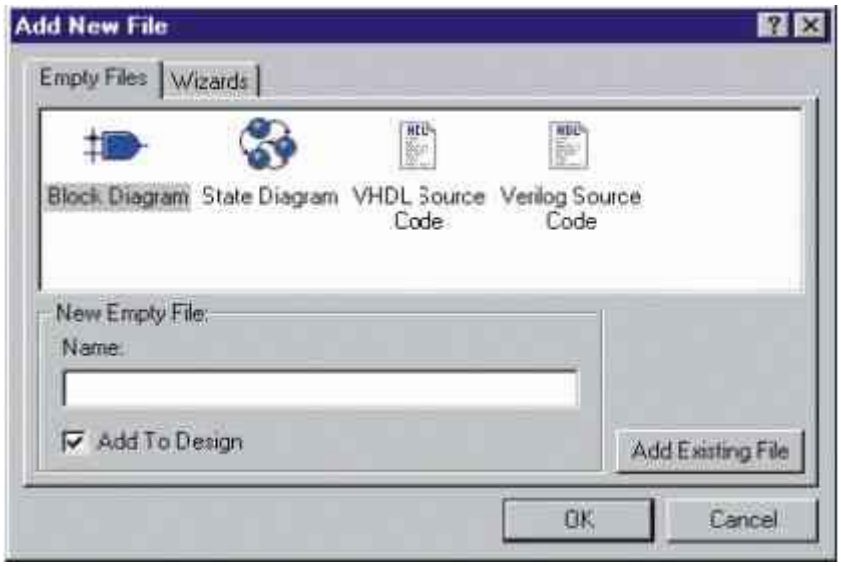


Рисунок 5.5 – Додання нового файла

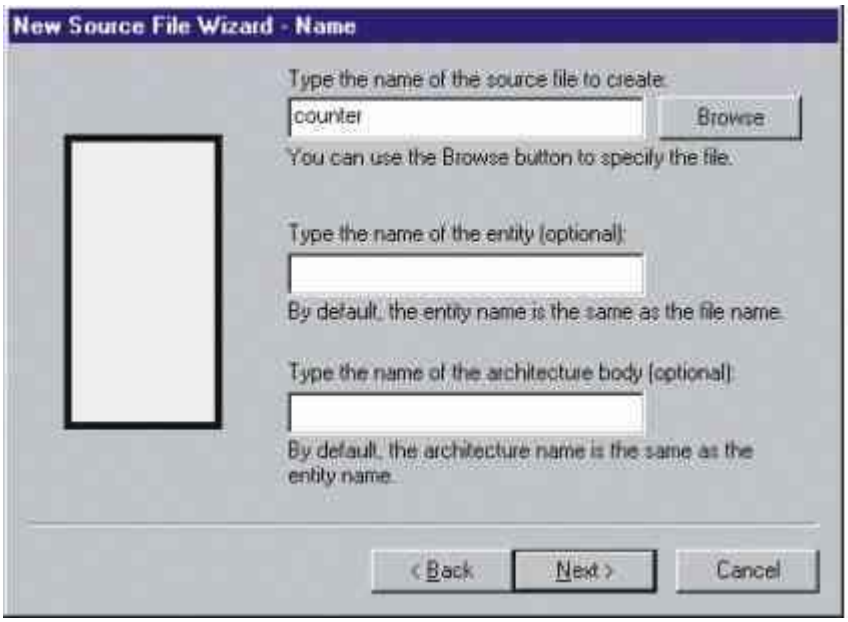


Рисунок 5.6 – Введення імен файла, сутності та архітектури

5.3 Робота з HDL-редактором

HDL-редактор (рис. 5.7) – це інструмент для введення VHDL-коду, файлів випробувального стенда та інших текстових файлів. Щоб перейти до редагування існуючого файла, треба клацнути мишкою на його закладці у вікні редактора або двічі клацнути на імені файла у вікні **Design Browser**, або скористатися опцією **Open** у контекстному меню. Створення нового файла коду описане у попередньому розділі.

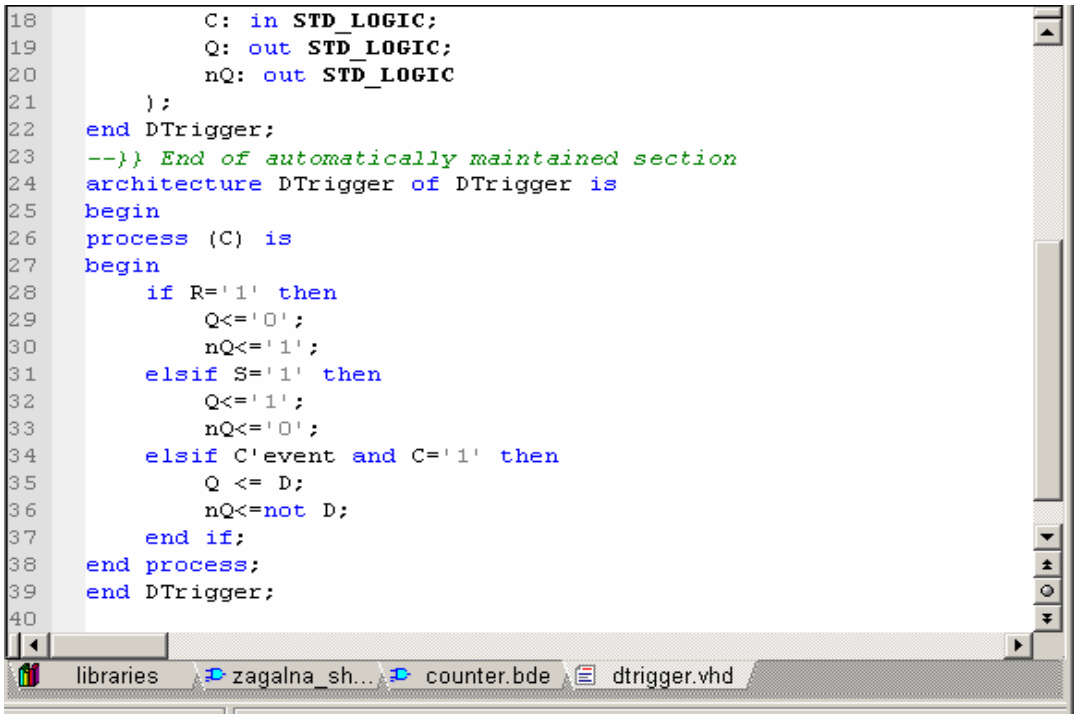


Рисунок 5.7 – HDL-редактор

Як уже відзначалось, HDL-редактор розпізнає і автоматично виділяє відповідними кольорами ключові слова, коментарі, ідентифікатори та інші елементи мови VHDL чи інших мов, з якими він може працювати. Це полегшує сприйняття тексту програми і є засобом додаткового синтаксичного контролю. Для вибору кольорів і шрифтів служить опція Preferences контекстного меню редактора.

В лівій частині вікна HDL-редактора знаходяться такі функціональні кнопки:

 - робить відступ виділеного фрагмента тексту (коду);

 - видаляє відступ в виділеному тексті;

 - перетворює обраний текст на коментар;

 - знімає ознаки коментарів з обраного тексту;

 - генерує для виділеного тексту символ структури (прямокутник зі значком "+" або "-"), натиснувши на який можна згортати до одного рядка та знову розгортати даний фрагмент тексту;

 - видаляє символи структур з усіх груп;

 - аналізує будову програми, знаходить у ній специфічні групи операторів, виділяє їх різними кольорами фону та генерує для них символи структури;

 - автоматично встановлює відступи відповідно до вкладеності операторів;

 - додає посилання (у формі коментарів);

 - розміщує у тексті програми іменовану закладку, перехід до якої можна буде виконати за допомогою кнопки;

 - управляє відображенням символів кінця рядка;

 - відповідно: вирізає та копіює в буфер обміну виділений фрагмент тексту, і вставляє текст із буфера;

 - відповідно: відміняє попередню операцію редагування та відновлює відмінену операцію;

 - відповідно: активізує вікно діалогу для пошуку і заміни тексту за введеним зразком, дозволяє вибрати один із раніше введених зразків для пошуку, виконує перехід до наступного і попереднього екземпляра даного зразка;

 - відповідно: встановлює або знімає неіменовану закладку, що відображається у вигляді прапорця, виконує перехід до наступної чи попередньої закладки, відміняє всі неіменовані закладки.

HDL-редактор дозволяє прискорити і полегшити написання програм, використовуючи автоматичне введення закінчень ключових слів. При наборі ключового слова можна вводити тільки його початкові літери доти, доки справа від курсора не з'явиться закінчення. Якщо ви погоджуєтесь з таким закінченням, натисніть клавішу переміщення курсора вправо, якщо ні – продовжуйте набирати текст. Подібним чином можна автоматично вводити не лише закінчення ключових слів, а й цілі шаблони мовних конструкцій. Для цього після появи справа від курсора закінчення ключового слова, треба натиснути комбінацію клавіш **Ctrl+Enter**.

Наприклад, для вводу оператора case достатньо ввести лише дві перші літери "*ca*", після чого, натиснувши **Ctrl+Enter**, ми одержимо у файлі шаблон для написання коду.

```
case <expression> is
    when <choices> =>
            <statements>
    when <choices> =>
            <statements>
    when others =>
            <statements>
end case;
```

Ще одним засобом, що суттєво полегшує роботу з HDL-редактором, є **Мовний помічник**, застосування якого описано нижче.

5.3.1 Використання Мовного помічника – Language Assistant

Як вже відзначалось, **Мовний помічник** дозволяє прискорити розробку програм на VHDL за рахунок використання шаблонів - готових

фрагментів коду, що містять базові конструкції мови VHDL та описи основних функціональних блоків: мультиплексорів, тригерів, лічильників тощо. **Мовний помічник** містить також шаблон користувача і навчальну програму з декількома простими проектами.

Для активізації вікна **Мовного помічника** (рис. 5.8) можна скористатися кнопкою на панелі інструментів. Ліва панель вікна **Language Assistant** показує ієрархічний список файлів шаблонів, згрупованих у папки, права - відображає вміст відміченого на лівій панелі файла. За допомогою контекстного меню лівої панелі можна долучати, перейменовувати і видаляти файли шаблонів та вставляти їх у файл, що редагується HDL-редактором.

Для цього можна методом **drag & drop** перетягти ім'я шаблона у вікно HDL-редактора з вікна **Мовного помічника,** або скористатися кнопкою на його інструментальній панелі чи командою **Use** з контекстного меню.

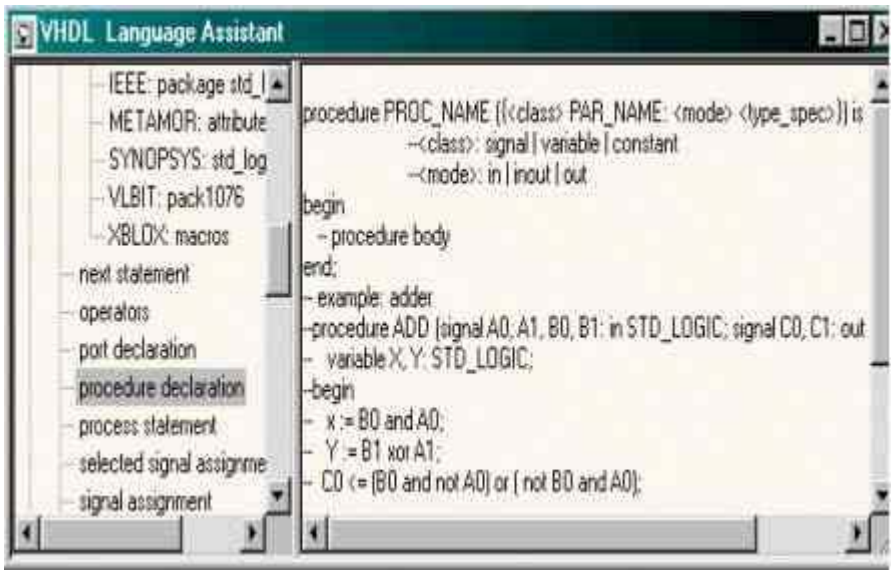


Рисунок 5.8 – Вікно Мовного помічника

5.4 Робота з Редактором автоматів – State Diagram Editor

Будь-яка цифрова схема з детермінованою поведінкою може розглядатися як скінченний автомат (автомат зі скінченним числом внутрішніх станів). Отже, зв'язок між сигналами на вхідних і вихідних портах такої схеми може бути описаний графом автомата. Ця форма опису є особливо зручною і наочною для схем, що реалізують алгоритми управління. Її застосування дозволяє суттєво підвищити швидкість створення і верифікації проектів.

Active-HDL має в своєму складі графічний **Редактор автоматів**, що дозволяє легко створювати і редагувати графи, компілювати цю форму опису в опис у вигляді файлів мовою VHDL чи Verilog, реалізувати на графі ефекти анімації, висвітлюючи поточні стани автомата під час моделювання тощо.

Файли графів автоматів мають імена з розширенням *.asf. Щоб перейти до редагування існуючого файла, треба клацнути мишкою на його закладці у вікні редактора або двічі клацнути на імені файла у вікні **Design Browser** чи скористатися опцією **Open** у контекстному меню. Створення нового файла графа автомата описане у розділі 5.1.

Вікно Редактора автоматів (рис. 5.9) складається з *зони декларацій* (у верхній частині вікна) і *робочої зони* (обведеної контуром). Перша містить інформацію про ім'я сутності і порти, друга призначена власне для побудови графа.

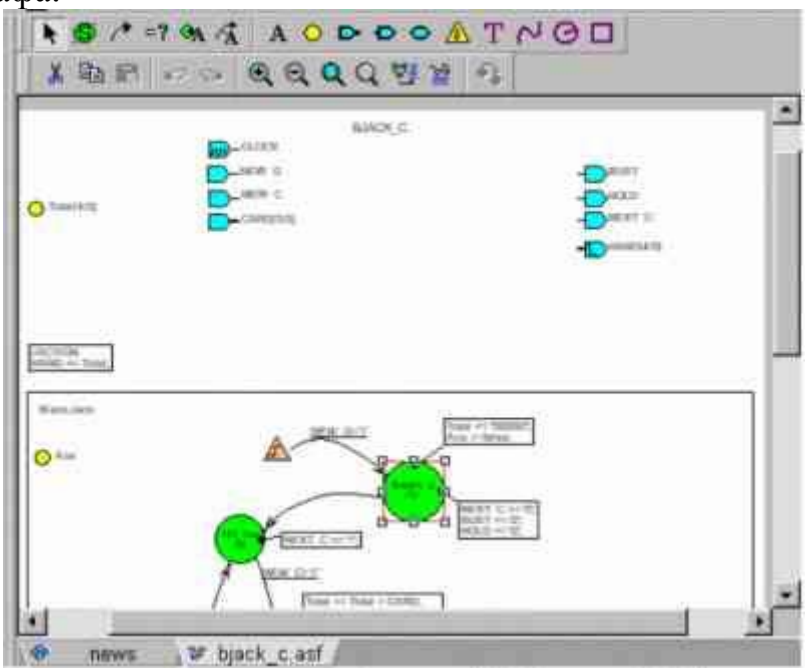


Рисунок 5.9 – Вікно Редактора автоматів

Панель інструментів **Редактора автоматів** містить такі функціональні кнопки:

 - вибір та редагування;

 - додання нової вершини (стану);

 - додання нового ребра (переходу);

 - додання умови переходу;

 - дії автомата під час перебування у стані і під час переходу;

 - додання дії "Автомат-діаграма";

 - додання сигналу;

 - додання портів: вхідного, вихідного, двонаправленого;

 - додання початкового стану;

 - вставка текстового рядка (коментаря), рисування кривої лінії, кола, прямокутника;

 - вирізання і копіювання в буфер обміну та вставка з буфера;

 - відміна дії та повернення до попередньої відміненої дії;

 - збільшення і зменшення усієї сторінки, показ усієї сторінки, збільшення виділеного фрагмента;

 - генерація VHDL-коду поточного графа автомата та перегляд згенерованого VHDL-коду;

 - трасування до наступного стану.

5.4.1 Розміщення і редагування портів та глобальних сигналів

Файл, створений **Майстром нового проекту** чи **Майстром нового файла**, як правило, уже містить імена сутності і архітектури, а також зображення портів, що були введені у діалоговому вікні, показаному на рисунках 5.4 та 5.6. Якщо ж був створений пустий файл, то усі названі реквізити треба вводити вручну.

- Для розміщення порту треба натиснути на панелі інструментів іконку з зображенням потрібного режиму  (вхід, вихід чи двонаправлений порт), потім вказати мишкою положення порту на полі декларативної частини.
- Переміщення порту, як і будь-яких інших елементів, виконується мишкою методом **drag** & **drop**.
- Для вилучення порту його спочатку треба виділити, клацнувши на ньому мишкою (порт буде обрамлений червоною рамкою), а потім натиснути клавішу **Delete**.
- Для зміни режиму, розрядності, імені, типу та інших реквізитів порту треба клацнути на ньому правою клавішею мишки, вибрати у контекстному меню опцію **Properties**, ввести корективи у відповідні поля вікна діалогу і натиснути кнопку **ОК**. Зокрема вхідний порт, сигнал якого задає відлік дискретного часу, у якому функціонує автомат, повинен мати відмітку у полі **Clock**.
- Крім того, ім'я порту, як і будь-яку іншу текстову інформацію, можна коригувати безпосередньо на графі. Для цього його треба спочатку виділити, клацнувши мишкою (у результаті навколо нього з'явиться червона рамка). Потім на ньому треба клацнути мишкою вдруге. В результаті відкриється вікно редагування (чорна об'ємна рамка з курсором). Для завершення редагування слід клацнути мишкою за межами рамки.

Розміщення та редагування глобальних сигналів виконується аналогічно розміщенню та редагуванню портів, але з використанням кнопки  .

Серед параметрів сигналів, що задаються у вікні діалогу опції **Properties** контекстного меню, так само, як і серед параметрів усіх портів, крім вхідних, є інформація про спосіб формування сигналу: **Registered** чи **Combinatorial**. Перший означає, що даний сигнал формується на виході

регістра, отже моменти зміни його значень “прив'язані” до міток дискретного часу, в якому функціонує автомат. Другий варіант означає формування сигналу на виході комбінаційної схеми, отже, його значення можуть змінюватись внаслідок зміни вхідних сигналів у будь-який момент часу.

5.4.2 Розміщення і редагування вершин станів

Вершинам графа відповідають стани автомата.

- Щоб розмістити вершину **B** на робочій області вікна **Редактора автомата** треба натиснути на панелі інструментів кнопку **State**  (при цьому курсор набуває форми круга) і клацнути мишкою в тому місці робочої області вікна, де потрібно розташувати вершину.
- Для зміни розмірів та форми вершини потрібно клацнути на ній мишкою. В результаті навколо вершини з'являться маркери, переміщуючи які можна надати вершині бажаної форми та розмірів.
- Колір вершини можна обрати на закладці **Graphics** з опції **Properties** контекстного меню, що викликається натисканням правої клавіші мишки на зображенні вершини.
- Створюваним вершинам автоматично надаються імена *S1*, *S2*, ... Для зміни імені можна скористатися вищезгаданою опцією Properties контекстного меню (закладка **General**). Крім того, ім'я можна коригувати так, як коригується будь-яка текстова інформація на графі: спочатку відмітити її, а потім, вдруге клацнувши на ній мишкою, відкрити вікно редагування.
- З кожним станом автомата можуть бути пов'язані його дії щодо надання значень сигналам і змінним. Ці дії можуть виконуватися при входженні автомата у цей стан (**Entry**), його перебуванні в даному стані (**State**) і виході з нього (**Exit**). На графі дії, що виконуються при входженні, перебуванні і виході зі стану відображаються у прямокутниках, сполучених з точками, розташованими відповідно у верхній, правій та нижній частині вершини (рис. 5.10). Для вводу і редагування цих дій можна скористатися закладкою Actions з опції Properties у контекстному меню вершини (рис. 5.11). Крім того, дії можуть редагуватися як текстова інформація безпосередньо на графі.
- Для зміни положення вершини у вікні редактора треба навести курсор на вершину і, утримуючи ліву кнопку мишки, перетягнути вершину на нове місце розташування.
- Для вилучення вершини треба її виділити, клацнувши мишкою, і натиснути клавішу Delete.

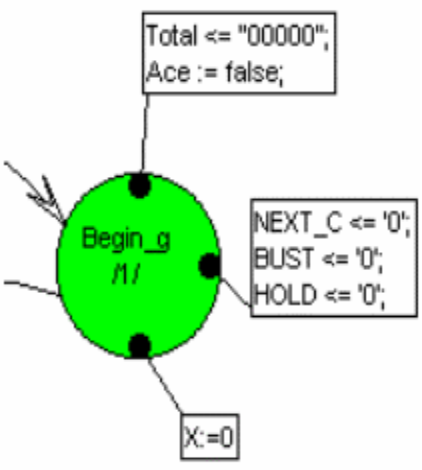


Рисунок 5.10 – Відображення дії на графі

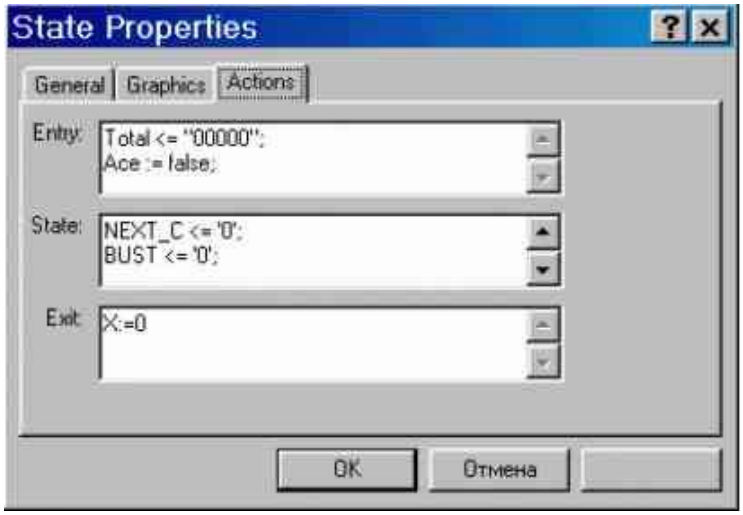


Рисунок 5.11 – Зміна дій стану

5.4.3 Розміщення і редагування ребер переходів

Ребрам, що сполучають вершини графа, відповідають переходи автомата від одного стану до іншого.

Для побудови ребра натисніть на панелі інструментів кнопку Transition  . Коли курсор набуде форми хрестика, клацніть мишкою на вершині, з якої виходить лінія ребра, потім (при бажанні) на одній чи декількох точках перегину цієї лінії, і нарешті – на вершині, в яку це ребро входить.

Щоб скоригувати форму ребра, його треба виділити, клацнувши мишкою, а потім перетягти на нове місце точки перегину лінії чи маркери навколо них.

Вже після побудови ребра, вершину, з якої виходить, чи у яку входить це ребро, можна замінити іншою. Для цього потрібно виділити ребро і перетягти маркер точки входу чи точки виходу ребра від однієї вершини до іншої. Можна також тимчасово залишити дане ребро неприєднаним до жодної з вершин. Таке ребро матиме вигляд, показаний на рис. 5.12. Після приєднання ребра до вершини хрестик на його кінці зникне.

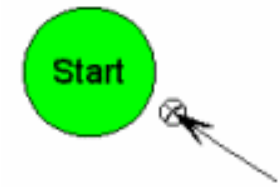


Рисунок 5.12 – Ребро, не приєднане до жодної з вершин

Для вилучення ребра його треба виділити і натиснути кнопку **Delete**.

Крім переходів між вершинами, на графі автомата також повинен бути показаний перехід автомата у початковий стан, що виконується при його ініціалізації. Для цього застосовується позначення  , що відіграє ту саму роль, що і вершина "початок" на схемі алгоритму. У контекстному

меню цього позначення можна вибрати синхронний чи асинхронний характер переходу автомата у початковий стан.

5.4.4 Розміщення і редагування умов переходів

Як відомо, автомат виконує той чи інший перехід, залежно від значень вхідних сигналів. Отже, якщо з вершини виходить більше одного ребра, то кожному з них повинна бути поставлена у відповідність умова, при виконанні якої відбуватиметься перехід, що описується саме даним ребром. Умови, що описують переходи з однієї вершини автомата, повинні бути взаємовиключними, а одну з них можна позначити @ELSE. Остання вважається виконаною тоді, коли не виконуються всі інші умови переходу із даної вершини.

Для введення умови переходу треба натиснути на панелі інструментів кнопку Condition =? (курсор при цьому змінить свою форму) і відмітити ним потрібне ребро. В результаті відкриється вікно редагування, у яке слід ввести умову переходу (рис. 5.13), дотримуючись правил синтаксису мови VHDL, і натиснути клавішу Enter або клацнути мишкою за межами вікна.

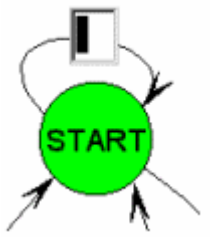


Рисунок 5.13 – Введення умови переходу

Умова переходу відображається на графі підкресленою. Цим її вигляд відрізняється від вигляду вихідних сигналів, що відображаються у рамках.

Щоб змінити розташування умови на графі, її треба виділити, клацнувши мишкою (при цьому з'являється лінія, що показує до якого саме ребра відноситься ця умова), а потім перетягти мишкою на бажане місце.

Якщо треба відредагувати сам зміст умови переходу, то, виділивши її, треба вдруге клацнути на ній мишкою. В результаті відкриється вікно для редагування умови переходу. Іншим способом редагування умови переходу є використання закладки HDL опції **Properties** у контекстному меню ребра.

Щоб вилучити умову переходу, треба виділити її і натиснути клавішу **Delete**.

5.4.5 Розміщення і редагування дій переходів

Дії автомата щодо надання значень сигналам і змінним можуть бути пов'язані не лише з входом, перебуванням чи виходом автомата із певного стану, як це розглядалось у розділі 5.4.2, а й з його переходом із одного стану в інший.

Для введення дій переходу треба натиснути на панелі інструментів кнопку **Transition Action**  (курсор при цьому змінить свою форму) і відмітити ним потрібне ребро переходу на графі. Порядок введення і редагування дій переходів, аналогічний описаному вище порядку введення і редагування умов переходів.

5.4.6 Розміщення на графі коментарів та інших елементів

На графі автомата можна розмістити також коментарі у вигляді текстів та графіки, наприклад, для визначення внутрішніх сигналів. Для цього використовуються кнопки  , що розміщені на панелі редактора автоматів.

5.4.7 Компіляція графа автомата у програму на VHDL

На основі файла графа автомата *.asf, створеного у Редакторі автоматів, може бути згенерований файл, *.vhd, що описує даний автомат мовою VHDL.

Перш ніж приступати до генерації VHDL-файла, слід записати на диск останні зміни, внесені у файл графа автомата, вибравши у меню **File** опцію **Save** або одноіменну кнопку на панелі інструментів.

Для генерації VHDL-файла треба натиснути кнопку  . Повідомлення про результати генерації та виявлені помилки виводиться у вікні **Console**. Якщо генерація була успішною, переглянути і, при потребі, скоригувати створений файл можна, натиснувши кнопку  .

5.5 Робота з редактором блок-схем

Редактор блок-схем дозволяє описувати проект як схему з'єднання елементів (рис. 5.13). Елементи, у свою чергу, можуть описуватися або безпосередньо VHDL-кодом, або графом автомата, або блок-схемою нижчого ієрархічного рівня. У кінцевому результаті увесь опис транслюється у VHDL-код.

Файли, створені у **Редакторі блок-схем**, мають імена з розширенням *.bde. Подібно до інших графічних редакторів схемотехнічного призначення, **Редактор блок-схем** дозволяє розміщувати, обертати й упорядковувати символи, проводи, шини тощо. Інструментальна панель з основними командними кнопками розташована у верхній частині вікна редактора і може викликатися також з контекстного меню. Далі ми наведемо короткий опис призначення кнопок, а потім розглянемо їх застосування для побудови схем.

 - переключає редактор в режим редагування з інших режимів;

 - застосовується для розміщення на схемі елементів, архітектура яких вже описана у відкомпільованих файлах проекту;

 - створює і розміщує на схемі порожній елемент, архітектура якого буде описана пізніше;

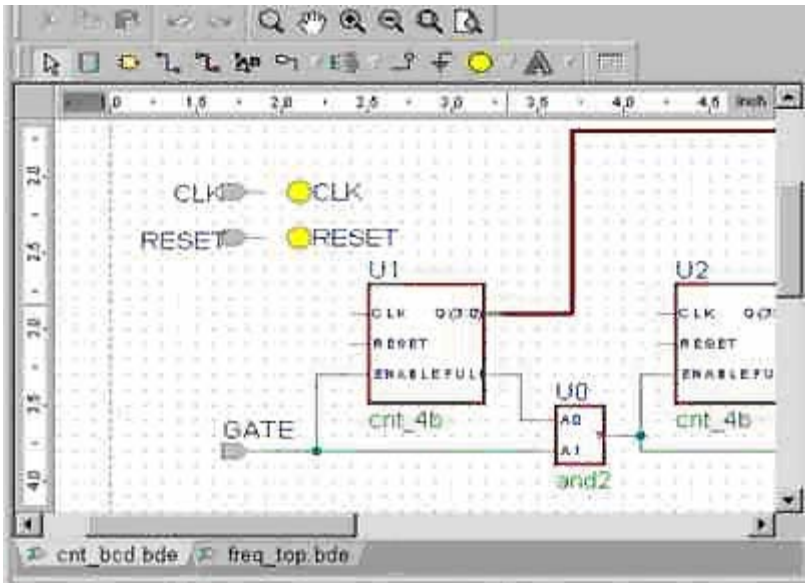


Рисунок 5.13 – Вікно редактора блок-схем

 - застосовуються, відповідно, для креслення проводів, шин та надання їм імен (ці кнопки дозволяють встановлювати зв'язки між компонентами);

 - дозволяють створювати порти, тобто точки, якими завершуються провідники вводу, виводу, двонаправлені і буферні відповідно;

 - виконують ту саму функцію для шин;

 - позначають на схемі провідники з напругами живлення Vcc та землі;

 - застосовуються для оголошення глобальних сигналів і шин;

 - дозволяють розмістити на схемі додаткові елементи типу коментарів у вигляді тексту чи малюнків;

 - використовуються для вводу фрагментів коду на VHDL.

5.5.1 Розміщення і редагування портів

Якщо файл створюється за допомогою **Майстра нового проекту** (див. розділ 4.7), то імена і типи портів вводяться у вікні діалогу, показаному на рис. 5.4, ще до початку роботи з редактором блок-схем. В такому випадку, після відкриття вікна редактора, зображення портів уже будуть наявні у файлі.

Якщо ж ви створили пустий файл, то зображення портів провідників і шин можна нанести за допомогою кнопок  . Цими ж кнопками можна скористатися для розміщення на блок-схемі нових портів.

Зображення портів можна розмістити також під час креслення ліній зв'язку, як це буде описано у розділі 5.5.3.

Для корегування імен, розрядності та режимів існуючих портів треба клацнути правою клавішею мишки на позначенні порту та вибрати у

контекстному меню опцію **Properties**. Крім того, ім'я та розрядність порту можна редагувати, двічі клацнувши мишкою на зображенні імені. Завершивши редагування, слід натиснути клавішу **Enter**.

5.5.2 Розміщення і редагування елементів

Розробка проектів у Active-HDL може виконуватись як знизу-вверх, так і зверху-вниз.

При проектуванні знизу-вверх, для нанесення на схему графічного позначення елементу, архітектура якого вже описана і відкомпільована, треба:

- клацнути мишкою на кнопці  , що відкриває вікно **Symbols Toolbox**;
- вибрати потрібний елемент у розташованому там списку відкомпільованих файлів проекту;
- методом **drag&drop** перетягти графічне зображення обраного елемента із вікна **Symbols Toolbox** у вікно редактора блок-схем.

При проектуванні зверху-вниз, щоб нанести на схему елемент, архітектура якого буде описана пізніше , треба:

- натиснути кнопку  (після цього курсор змінить свою форму на );
- намалювати прямокутник за розмірами майбутнього елемента;
- натиснути кнопку  , щоб повернутися до звичайного режиму редагування (курсор набуде своєї попередньої форми);
- клацнути правою клавішею мишки у прямокутнику і вибрати із контекстного меню опцію **Properties**;
- у полях **Fub name** та **Interface name** вказати, відповідно, функціональне та позиційне позначення елемента на схемі, замінивши позначення, автоматично надані елементові при його створенні;
- користуючись кнопками  , нарисувати виводи елемента - прості провідники та шини;
- повернутися до звичайного режиму редагування, натиснувши кнопку  ;
- клацнути правою клавішею мишки на зображенні елемента вибрати в контекстному меню команду Edit;
- для кожного з виводів елемента, користуючись опцією Properties із контекстного меню, замінити режими та імена виводів, а також розрядність шин, що були їм автоматично надані при створенні, на справжні режими, імена та розрядність.

Для того, щоб виконати описання архітектури створеного у такий спосіб позначення елемента, треба клацнути правою клавішею мишки всередині цього позначення і вибрати в контекстному меню опцію **Push**.

Після цього у вікні, подібному до показаного на рис. 5.5, треба вибрати тип опису: блок-схема, граф автомата, програма мовою VHDL чи програма мовою Verilog. Натиснувши кнопку **ОК**, ви перейдете до створення цього опису у відповідному редакторі. При цьому у створюваному файлі вже будуть наявні порти, імена й режими яких відповідають зовнішнім виводам створеного вами елемента **Fub**.

Після створення файла опису архітектури елемента команда **Push** забезпечуватиме безпосередній перехід до редагування цього файла, пропускаючи вікно, показане на рис. 5.5.

5.5.2.1 Symbols Toolbox

Symbols Toolbox – це зручний інструмент, який використовується для перегляду та розміщення символів на схемі. Для того, щоб відкрити вікно **Symbols Toolbox**, необхідно клацнути мишкою на кнопці  . Він дозволяє переглядати список компонентів, які є в вибраних бібліотеках. За замовчуванням **Symbols Toolbox** показує символи компонентів робочої бібліотеки, яка створюється разом з проектом, і містить всі створені в цьому проекті компонети. На додаток до робочої бібліотеки програміст може вибрати інші доступні бібліотеки.

Для вибору інших доступних бібліотек необхідно клацнути правою кнопкою мишки в області вікна Symbols Toolbox, і в контекстному меню вибрати **Choose Libraries**, у відповідь з’явиться діалогове вікно **Libraries**, в якому слід відмітити галочками бібліотеки, які необхідно включити до проекту. Після цього всі компоненти включених бібліотек стають доступними в редакторі блок-схем для розміщення та редагування.

5.5.3 Встановлення зв'язків

Щоб підготовити Редактор блок-схем до креслення ліній зв'язку, треба натиснути кнопку  . При цьому відповідним чином зміниться форма курсора.

Для проведення лінії зв'язку між будь-якими двома точками на схемі треба натиснути клавішу мишки у початковій і відпустити у кінцевій точці лінії. Звичайно редактор сам обирає раціональну траєкторію проведення лінії так, щоб вона обминала елементи та написи на схемах. Проте, якщо ви бажаєте самі задати цю траєкторію, то положення точок її перегину можна фіксувати клавішею "пропуск" під час креслення лінії.

Якщо лінія закінчується портом, то для його розміщення на схемі можна натискати клавішу Tab доти, доки зображення режиму порту (вхід, вихід чи двонаправлений порт) не набуде бажаного вигляду.

Лінії зв'язку, як правило, проводяться, “прив'язуючись” до точок координатної сітки. Для проведення ліній поза сіткою треба натиснути клавішу **Alt**.

Так само, але з використанням кнопки  , будуються лінії групового зв'язку (шини). Лінії зв'язку можна проводити також і в звичайному режимі

редагування (з курсором ). Для цього достатньо натиснути клавішу мишки на одному з контактів і відпустити її на іншому. Крім того, цей режим дозволяє методом **drag&drop** редагувати схему, зокрема, переміщувати схемні елементи, відрізки і точки згинання ліній зв'язку, копіювати або вилучати елементи та лінії.

Лінії зв'язку утворюють на схемі мережі з'єднань. Кожну мережу можна розуміти як окремий провідник, що з'єднує між собою дві чи більше точок на схемі. Іншими словами, якщо між двома точками є контакт, то вони належать до однієї мережі, якщо ні - до різних.

При створенні мереж з'єднань Active-HDL автоматично надає їм імена за замовчуванням: NET1, NET2 і т.д. Проте ці імена на схемі не відображаються і такі мережі називають неіменованими.

Мережі можна надати довільне ім'я та розмістити його в одному чи декількох місцях схеми, користуючись кнопкою  або опцією **Property** у контекстному меню лінії зв'язку. Таку мережу називають іменованою. Іменованими є також мережі, підключені до портів. Їм автоматично надаються імена цих портів.

Ім'я мережі може бути скоригованим безпосередньо на схемі чи у вікні діалогу **Property**. Можна також взагалі вилучити ім'я, виділивши його і натиснувши клавішу **Delete**. Тоді мережа знову стане неіменованою.

У будь-якому випадку одна мережа від іншої відрізняється іменем. Тому для з'єднання точок на схемі не обов'язково сполучати їх лінією зв'язку. Достатньо показати, що вони підключені до однієї мережі, надавши однакових імен лініям зв'язку, що підведені до цих точок. Такі точки називають *з'єднаними за іменем*.

Це саме правило застосовується і для імен глобальних сигналів  та шин глобальних сигналів  . Усі порти і непідключені виводи елементів, які матимуть імена, що співпадають з іменами глобальних сигналів, вважатимуться з'єднаними між собою.

Якщо скомпілювати блок-схему у програму на VHDL, то мережам з'єднань схеми відповідатимуть у програмі сигнали з такими самими іменами. При цьому одинарним лініям відповідатимуть сигнали скалярного типу, а шинам – сигнали типу одномірних масивів. Якщо підключити мережу до контакту, то в програмі їх сигналам буде надано значень констант '0' чи '1' відповідно.

5.5.4 Включення операторів VHDL до блок-схеми

Як випливає з розглянутого вище, є пряма відповідність між графічними зображеннями на блок-схемі і їх описом у програмі на VHDL. Проте далеко не всі елементи мови VHDL мають відповідні графічні позначення на схемах. Через це у редакторі блок-схем передбачена можливість включення до графічного файла блок-схеми певних фрагментів описів, виконаних мовою VHDL. Під час компіляції графічного файла

блок-схеми в програму на VHDL ці описи будуть перенесені у відповідні місця програми.

Для включення операторів VHDL до графічного файла використовуються такі кнопки панелі інструментів або опції меню Diagram/VHDL Statements:

 - дозволяє додати оператори у заголовок проекту;

 - дозволяє додати оператори до опису сутності;

 - дозволяє додати оператори у декларацію архітектури;

 - дозволяє додати оператори у тіло архітектури;

 - дозволяє ввести узагальнення (generics).

5.5.5 Компіляція схемного файла у програму на VHDL

На основі схемного файла *.bde, створеного у Редакторі блок-схем, може бути згенерований файл, *.vhd, що описує дану схему мовою VHDL.

Перш ніж приступати до генерації VHDL-файла, слід записати на диск останні зміни, внесені у схемний файл, вибравши в меню **File** опцію **Save** або одноіменну кнопку на панелі інструментів.

Для генерації VHDL-файла треба натиснути кнопку  . Повідомлення про результати генерації та виявлені помилки виводиться у вікні **Console**. Якщо генерація була успішною, переглянути і, при потребі, скоригувати створений файл можна, натиснувши кнопку  .

5.6 Компіляція проекту

Програма моделювання проектів, що входить до складу Active-HDL, не може моделювати безпосередньо вихідний текст, написаний мовою VHDL. Тому проект, який ви бажаєте моделювати, потрібно спочатку відкомпілювати в бібліотеку проекту. Тільки дані, збережені в бібліотеці, можуть читатися програмою моделювання.

Перед компіляцією треба записати на диск останні зміни, внесені до файлів проекту, та вибрати у вікні Design Browser сутність верхнього рівня ієрархії (**Top-level Entity**). Щоб виконати компіляцію проекту, треба вибрати в контекстному меню опцію **Compile all** або натиснути одноіменну кнопку на панелі інструментів.

Повідомлення про результати компіляції та виявлені помилки виводиться у вікні **Console**, відповідні позначення про результат компіляції кожного файла виводяться біля його іконки у вікні **Design Browser**. Крім того, у вхідних файлах проекту відповідним чином виділяються оператори VHDL та елементи графів і блок-схем, що містять помилки.

Після виправлення помилок та успішної компіляції всього проекту можна переходити до його моделювання.

6 МОДЕЛЮВАННЯ ПРОЕКТУ

Моделювання є дуже важливим етапом розробки проекту. Це етап аналогічний тестуванню програм. Його мета – вивчити функціонування проекту, виявити та виправити можливі помилки ще до того, як проект буде реалізовано у вигляді мікросхеми. Для цього Active-HDL має у своєму складі різноманітні засоби, що дозволяють програмно імітувати подачу сигналів (стимулів) на входи проекту і відтворювати та аналізувати сигнали на його виходах. Такими засобами є:

- VHDL-файли випробовувального стенда, створені вручну або з використанням **Майстра випробовувального стенда** чи зовнішніх редакторів;
- команди моделювання, що вводяться у вікно консолі;
- файли макрокоманд моделювання;
- файли векторів перевірки, імпортовані з Active-CAD;
- моделювання, засноване на формах сигналу, відредагованих користувачем.

Всі ці засоби можуть використовуватись в одному проекті паралельно. Користувач обирає метод, що найкраще підходить для потреб проекту й має найкращий баланс між часом, необхідним для проведення моделювання, і складністю перевірки проекту.

6.1 Робота з Редактором часових діаграм – Waveform Editor

Редактор часових діаграм є основним засобом для створення стимулювальних сигналів і перегляду результатів моделювання. Файли часових діаграм мають імена з розширенням *.awf і зберігаються у папці **Waveforms**. Щоб відкрити існуючий файл часової діаграми, можна клацнути мишкою на закладці файла у вікні редакторів або двічі клацнути на імені файла у вікні **Design Browser**, або вибрати у контекстному меню команду **Open**. Для створення нового файла часової діаграми можна скористатися кнопкою  (**New Waveform**).

Вигляд вікна Редактора часових діаграм показано на рис. 4.4. В його лівій частині у колонках **Name**, **Value**, **Stimulator** відображаються, відповідно, імена сигналів, їх значення у момент, що відповідає положенню курсора, а також стимулятори, які застосовуються для формування вхідних сигналів. Натискаючи на кнопки з заголовками цих трьох колонок, можна розташовувати сигнали за алфавітним порядком їх імен чи стимуляторів, у порядку, зростання значень або у порядку, протилежному вказаним. Якщо сигнали не впорядковані за жодною з цих ознак, то їх розташування можна обирати довільно, перетягуючи сигнали за допомогою мишки.

Масштаб зображення регулюється кнопками  , які дозволяють, відповідно, збільшити масштаб вдвічі; зменшити масштаб

вдвічі; вивести всю діаграму на один екран; задати у вікні діалогу часові межі зображення на екрані.

Кнопка  включає режим **Zoom**, у якому можна, виділивши мишкою частину зображення, показати його на весь екран. Щоб збільшити масштаб вдвічі, тут достатньо просто клацнувши мишкою по діаграмі, а для зменшення - клацнути мишкою при натиснутій клавіші **Ctrl**. Крім того, в будь-якому режимі зручно збільшувати і зменшувати масштаб за допомогою клавіш "+" і "–" на додатковій клавіатурі.

Векторні сигнали справа від імені мають кнопку, що дозволяє переключатись між згорнутим (єдина діаграма для всього сигналу) і розгорнутим (окрема діаграма для кожного біту) способом їх відображення. Опція **Properties** з контекстного меню сигналу дозволяє також обрати його колір, систему числення, в якій вказується значення векторного сигналу тощо.

6.1.1 Додання та вилучення сигналів

Щоб розмістити сигнал на часовій діаграмі, треба відкрити закладку **Structure** у вікні **Design Browser**, виділити в структурі проекту елемент, сигнали якого ви хотіли б відобразити на часовій діаграмі та методом **drag & drop** перетягти потрібні сигнали з нижньої частини закладки Structure у ліву частину вікна Редактора часових діаграм.

Інший спосіб розміщення сигналу на часовій діаграмі – це використання опції **Add Signals** з контекстного меню лівої частини вікна Редактора часових діаграм. Користуючись ним, треба виділити потрібні сигнали і натиснути кнопку **Add**.

Сигнали можна також додавати за допомогою команди **Wave**, що вводиться з клавіатури у вікно **Console**.

Щоб вилучити сигнал зі списку сигналів часової діаграми треба, встановивши на ньому курсор, натиснути клавішу **Delete** чи одноіменну опцію у контекстному меню.

6.1.2 Призначення стимуляторів

Будь-якому сигналові, наявному на часовій діаграмі, можна надати бажаних значень, скориставшись вбудованими у програму стимуляторами. Для цього треба встановити курсор на потрібному сигналі і вибрати опцію **Stimulators** у контекстному меню або одноіменну кнопку на панелі інструментів. Вікно, що відкривається при цьому (рис. 6.1), дає можливість вибору описаних нижче стимуляторів.

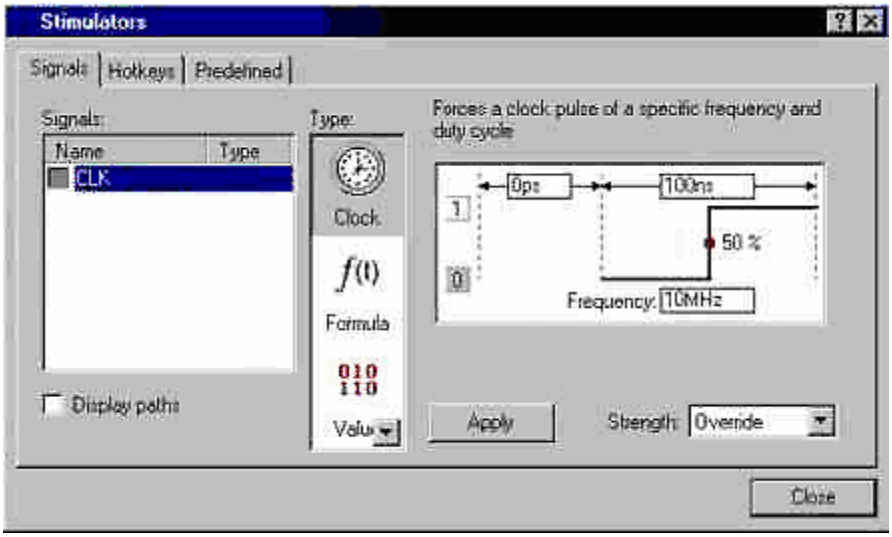


Рисунок 6.1 – Вікно вибору стимулятора

Value – дозволяє вибрати значення, що буде надане виділеному сигналу чи групі сигналів, об'єднаних у шину;

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;

Hotkey – дозволяє призначити сигналові гарячу клавішу, натискання якої приводитиме 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;

Clock – дозволяє задати періодичний сигнал з визначеними параметрами (частотою, періодом тощо), користуючись зручним графічним редактором, показаним на рис. 6.1

Counter – дозволяє моделювати сигнали на виходах лічильника, з заданими значеннями періоду, напрямку лічби і кодування даних у лічильнику: двійковий код, код Грея та ін.;

Custom – дозволяє використати стимул користувача, створений у Редакторі часових діаграм, як це описано в розділі 4.8;

Predefined – дозволяє вибрати наперед визначений стимулювальний сигнал з заданого переліку та поповнити цей перелік власними сигналами, що часто використовуються.

На практиці трапляються випадки, коли значення сигналу визначається не лише самим стимулятором, а й іншими драйверами. Тоді виникає питання: як стимулятор повинен взаємодіяти з цими драйверами і як розв'язувати протиріччя, якщо стимулятор намагається встановити одне значення сигналу, а драйвери – інше? Відповідь на це питання визначається типом стимулятора, що вибирається у полі **Strength**, розташованому у правому нижньому кутку вікна вибору стимулу (рис. 6.1):

- ***Override*** – стимулятор примусово встановлює значення сигналу, що залишається незмінним до наступної команди FORCE або UNFORCE;
- ***Deposit*** – стимулятор встановлює значення сигналу, що залишається незмінним тільки до наступної транзакції драйвера або команд FORCE, UNFORCE;
- ***Drive*** – приєднує віртуальний драйвер до сигналу і цей драйвер форсує його значення до наступної команди FORCE або UNFORCE. Цей тип стимулятора використовується для миттєвих значень сигналів типу std_logic.

6.1.3 Переміщення по часових діаграмах та порівняння діаграм

Переміщення по часовій діаграмі може виконуватись за допомогою смуг прокрутки, а також кнопок:

-  , що переміщують курсор, відповідно, до наступної чи попередньої зміни значення виділеного сигналу;
-  , що дозволяє знайти потрібне значення виділеного сигналу;
-  , що дозволяє перейти до моменту часу, який задається у вікні діалогу;
-  , що дозволяють перейти, відповідно, до наступної чи попередньої закладки, котрі відображаються на часовій осі блакитними трикутниками, які можна встановити або зняти за допомогою кнопки  або просто клацнувши мишкою на часовій осі. Кнопка  відміняє усі закладки. Крім того, за допомогою кнопки  , розташованої на горизонтальній лінійці прокрутки, можна отримати доступ до описаних вище переходів, а також кнопок:
-  , яка забезпечує перехід до бажаного коментаря, що був введений за допомогою кнопки  ;

-  , яка забезпечує перехід до моменту, коли виявлена різниця між сигналами поточної часової діаграми та діаграми, записаної у іншому файлі.

Для порівняння діаграм слід натиснути кнопку  і ввести ім'я файла іншої діаграми. Приклад відображення результату порівняння діаграм показаний на рис. 6.2.

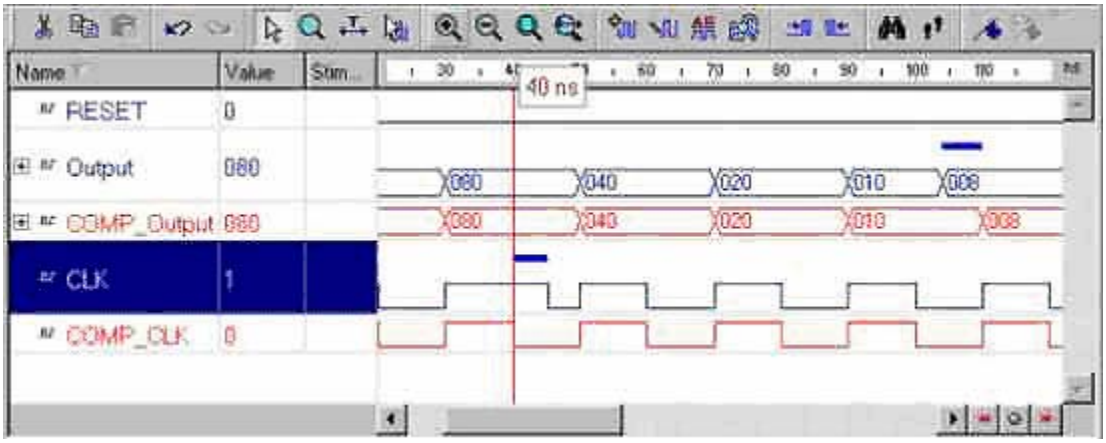


Рисунок 6.2 – Порівняння часових діаграм.

Якщо сигнал даної діаграми не має відмінностей від одноіменного сигналу на іншій діаграмі, то у вікні результату він відображається один раз. На рис. 6.2 таким є сигнал RESET.

Якщо ж ці сигнали відрізняються, то вони обидва відображаються на діаграмі поруч, причому, сигнал з іншого файла виділяється червоним кольором, а до його імені додається префікс COMP_. Інтервали часу, у які значення сигналів не співпадають, відмічаються вгорі жирними блакитними відрізками.

Щоб вилучити з діаграми результати порівняння можна описаним вище способом порівняти її саму з собою.

6.1.4 Виміри у часових діаграмах

Активний курсор відображається на діаграмі вертикальним червоним відрізком. Його часова координата відображається в прямокутнику біля верхнього краю відрізка (на рис. 6.2 це *40 ns*). Значення сигналів, що відповідають поточному положенню активного курсора, відображаються на діаграмі у колонці **Value**. Якщо курсор проходить точно через фронт сигналу, то у таблиці вказується його значення справа від курсора.

Щоб розмістити на діаграмі ще один курсор, треба клацнути у потрібному місці правою клавішею мишки і вибрати у контекстному меню опцію **Add Cursor**. Якщо на діаграмі є декілька курсорів, то їх координати відображаються у прямокутниках на нижніх кінцях відрізків (рис. 6.3). Крім того, стрілками відображаються проміжки часу від даного курсора до його найближчих сусідів. Активний курсор відображається червоним кольором, решта – синім. Щоб зробити курсор активним, на ньому треба клацнути мишкою.

Для переміщення активного курсора можна скористатися будь-яким способом, описаним у попередньому розділі, або просто клацнути мишкою на діаграмі.

Для видалення курсора треба клацнути правою клавішею мишки на прямокутнику з його координатою, а потім натиснути у контекстному меню опцію **Remove**.

Ще одним засобом вимірювання часових інтервалів є застосування режиму вимірювань, що включається кнопкою . У цьому режимі можна вимірювати часові інтервали тільки між фронтами імпульсів. Курсор мишки, що підводиться до фронту, набуває зеленого кольору. У цей момент треба натиснути ліву клавішу мишки і відпустити її біля іншого фронту. У такий спосіб на рис. 6.3 виміряно, наприклад, часовий інтервал *40 ns* між встановленням у сигналах RESET і Output значень 0 та 040.

Щоб видалити стрілку з виміром, треба перейти у звичайний режим, який включається кнопкою і натиснути кнопку **Delete**.

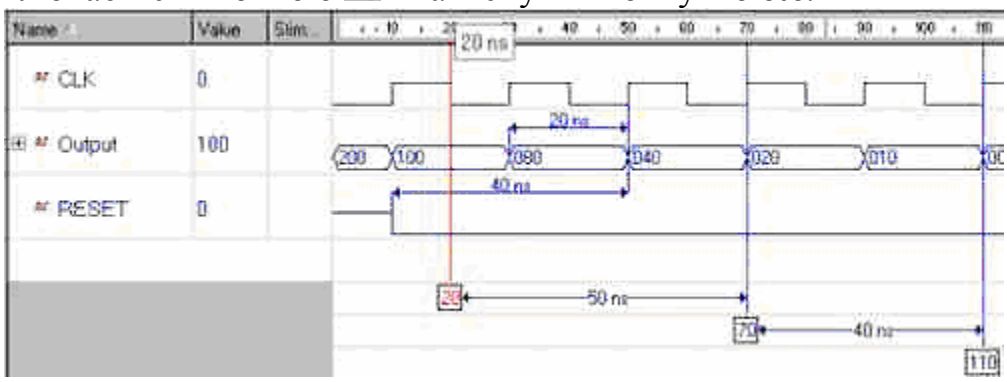


Рисунок 6.3 – Вимірювання часових інтервалів

6.1.5 Редагування часових діаграм

Часові діаграми можна редагувати та зберігати у файлах для подальшого використання. Щоб приступити до редагування діаграми, слід припинити моделювання командою **End Simulation** з меню **Simulation** та натиснути кнопку , що переводить редактор у режим редагування. При цьому стає доступними виконання описаних нижче операцій.

Зміна моменту переключення (фронту) сигналу виконується аналогічно регулюванню ширини колонок у таблицях Word чи Excel. Для цього треба підвести курсор до зображення фронту сигналу (вертикального відрізку на часовій діаграмі) так, щоб курсор набув вигляду двонаправленої стрілки, потім натиснути ліву клавішу мишки і відпустити її там, де має проходити фронт.

Аналогічним чином, підвівши курсор до горизонтального відрізку діаграми, можна підняти його вгору або опустити вниз, замінивши тим самим значення сигналу “**0**” на “**1**” чи навпаки.

Для створення нового імпульсу треба натиснути клавішу **Shift** (при цьому курсор набуде форми олівця) та клацнути мишкою по часовій діаграмі. В результаті на цьому місці буде створено імпульс невеликої

ширини. Потім описаним вище способом можна відредагувати ширину та положення імпульсу на часовій осі.

Щоб виділити фрагмент часової діаграми, треба натиснути клавішу **Ctrl** (у результаті курсор набуде форми хрестика), натиснути ліву клавішу мишки на початку і відпустити її у кінці фрагмента, що виділяється. Фрагмент може включати як одну, так і декілька сусідніх часових діаграм.

Виділений фрагмент можна:

- видалити натисканням клавіші **Delete** (при цьому за межами виділеного фрагмента значення сигналів не зміняться, а у його межах сигнали зберігатимуть ті значення, які вони мали на початку виділеного фрагмента);
- встановити у потрібне значення (для простих сигналів достатньо просто натиснути відповідну клавішу: **0**, **1**, **U**, **X** тощо, а для векторних сигналів краще скористатися опцією **Fill**... у контекстному меню);
- змінити масштаб часу, вибравши у контекстному меню команду **Stretch**... і вказавши у вікні **Scale** процент "розтягування" даного фрагмента у часі (дані поза межами фрагмента при цьому не змінюються);
- скопіювати у буфер обміну натисканням кнопки або комбінації клавіш **Ctrl+C**;
- вирізати натисканням кнопки або комбінації клавіш **Ctrl+X** (у результаті фрагмент буде скопійовано у буфер обміну, а з часової діаграми - видалено).

За допомогою кнопки або комбінації клавіш **Ctrl+V** фрагмент діаграми, що міститься у буфері обміну, може вставлятися у відмічені курсором місця інших часових діаграм, якщо дані у них належать до того самого типу. Тобто у діаграму, що відображає, наприклад, векторний сигнал, можна вставити лише векторний сигнал такої ж розмірності. При цьому, на відміну від вставки текстових даних, наявні у діаграмі дані не зсуваються далі вправо, а заміняються даними, що вставляються з буфера.

Записування результатів редагування часової діаграми у файл виконується за допомогою команд **Save** та **Save as**... з меню **File**.

Крім цього, дані з часової діаграми можуть бути експортовані у файли інших форматів за допомогою команди **Export Waveforms**... з меню **Waveform**. У вікні діалогу, що після цього відкривається, треба вказати ім'я файла, а також вибрати його тип та групу сигналів, що експортуватимуться.

Тип файла може бути або VHDL Process (*.VHS), що містить опис сигналів мовою VHDL у вигляді процесу, який їх формує, або Waves vector (*.VEC), що містить опис сигналів у вигляді таблиці, де кожна зміна значень сигналів подана одним рядком, у якому вказані значення сигналів і час, що минув від попередньої зміни та від початку моделювання.

При виборі групи сигналів, що експортуватиметься, є такі альтернативи:

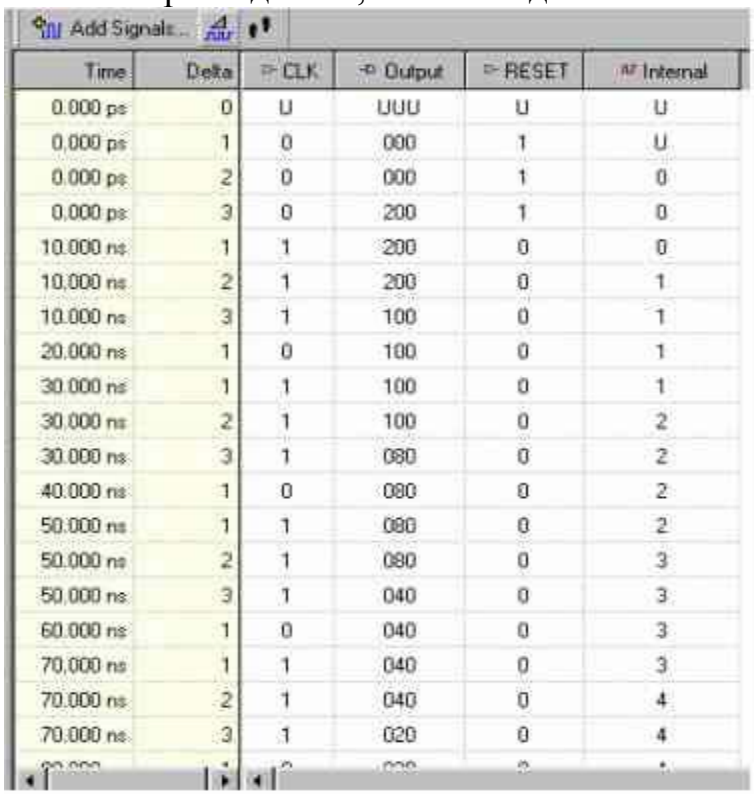
- **All input ports** – усі вхідні порти;
- **All ports** – усі порти;
- **All signals** – усі сигнали;
- **Selected signals** – вибрані сигнали.

6.2 Інші вікна перегляду результатів моделювання

Крім **Редактора часових діаграм**, Active-HDL має інші засоби відображення результатів моделювання проекту.

6.2.1 Вікно List

Вікно **List**, як і **Редактор часових діаграм**, містить інформацію про зміну у часі значень сигналів проекту, (рис. 6.4). Але, на відміну від **Редактора часових діаграм**, у вікні **List** ця інформація подана у вигляді таблиці і може тільки переглядатися, внесення до неї змін неможливе.



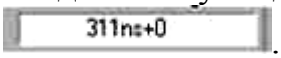
Time	Delta	CLK	Output	RESET	Internal
0.000 ps	0	U	UUU	U	U
0.000 ps	1	0	000	1	U
0.000 ps	2	0	000	1	0
0.000 ps	3	0	200	1	0
10.000 ns	1	1	200	0	0
10.000 ns	2	1	200	0	1
10.000 ns	3	1	100	0	1
20.000 ns	1	0	100	0	1
30.000 ns	1	1	100	0	1
30.000 ns	2	1	100	0	2
30.000 ns	3	1	080	0	2
40.000 ns	1	0	080	0	2
50.000 ns	1	1	080	0	2
50.000 ns	2	1	080	0	3
50.000 ns	3	1	040	0	3
60.000 ns	1	0	040	0	3
70.000 ns	1	1	040	0	3
70.000 ns	2	1	040	0	4
70.000 ns	3	1	020	0	4

Рисунок 6.4 – Вікно List

Вікно **List** відкривається за допомогою кнопки **New List**  . Сигнали, які треба відобразити в цьому вікні, вибирають так само, як це описано у розділі 4.8 для **Редактора часових діаграм**.

Дані у вікні **List** можуть видаватися в повному і скороченому варіанті, переключення між якими виконується за допомогою кнопки  .

Повний варіант дозволяє простежити послідовності зміни значень сигналів, кожна з яких відображається у таблиці окремим рядком навіть тоді, коли часовий інтервал між ними дорівнює **Delta** (так при

моделюванні позначають нескінченно малий відрізок часу між подіями). Отже, у таблиці може бути декілька рядків, що мають однакові значення у полі **Time** і відрізняються лише значеннями у полі **Delta** (рис. 6.4). До речі, у вікні **Simulation status**, час, що минув від початку моделювання, також відображається у форматі **Time + Delta**: 

.

Скорочений варіант відображення інформації у вікні **List** передбачає для кожного моменту часу виведення лише одного рядка з остаточними значеннями сигналів, що встановилось у даний момент. Іншими словами, у скороченому варіанті кожний момент часу поданий у таблиці лише рядком з максимальним значенням **Delta**.

Дані вікна **List** можна зберегти у текстовому файлі з розширенням *.lst , виконавши команду **Save as** ... з меню **File**.

6.2.2 Вікно Watch

Вікно **Watch** (рис. 6.5) має вигляд таблиці, в якій виводяться: імена змінних і сигналів (**Name**), їх тип (**Type**), поточні значення (**Value**), значення, які вони мали до цього (**Last Value**), і час, коли саме поточні значення сигналу були встановлені (**Last Event Time**). Якщо час зміни співпадає з поточним часом (тобто значення змінилося щойно), то ці значення виділяються у таблиці червоним кольором.

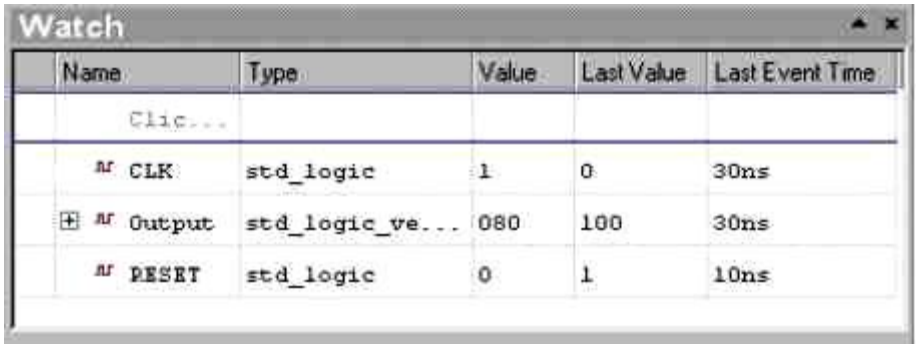


Рисунок 6.5 – Вікно **Watch**

Вікно **Watch** відкривається за допомогою кнопки **View Watch Window** або опції **Watch** у меню **View**. Потім треба відкрити закладку **Structure** у вікні **Design Browser**, виділити у структурі проекту елемент, сигнали якого ви хотіли б бачити у вікні **Watch** та методом **drag & drop** перетягти потрібні сигнали з нижньої частини закладки **Structure** у вікно **Watch**. Тобто виконується та сама процедура, що і для **Редактора часових діаграм** та вікна **List**.

6.2.3 Вікно Process

Вікно **Process** (рис. 6.6) відкривається за допомогою опції **Processes** у меню **View** і відображає стан процесів під час моделювання. Отже програма моделювання перед цим має бути активізованою. Якщо це не так, то її слід попередньо активізувати командою **Initialize Simulation** з меню **Simulation**.

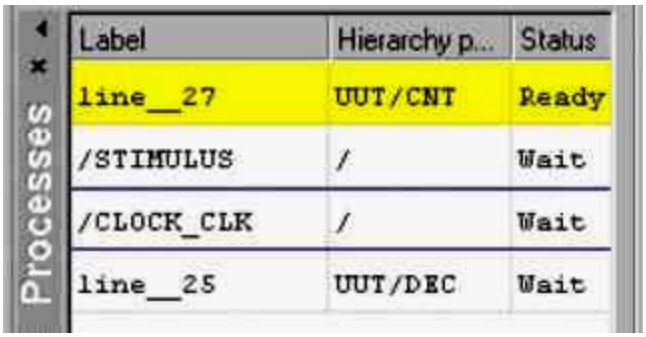


Рис. 6.6 – Вікно Process

У колонці **Label** відображається мітка, якою у програмі помічено процес, або номер рядка для процесів без мітки. Колонка **Hierarchy path** вказує на місце процесу в ієрархії проекту, а колонка **Status** містить інформацію про його стан. Як відомо, будь-який процес може знаходитись або в стані очікування (**Wait**), або у активному стані (**Ready**), тобто у стані, готовому для виконання в поточному циклі моделювання. Активні процеси завжди відображаються у верхній частині вікна, а процеси у стані очікування можуть взагалі не відображатись, якщо в контекстному меню даного вікна вибрати опцію **Show active**.

6.2.4 Вікно стеку викликів – Call Stack

Вікно стека викликів **Call Stack** дозволяє спостерігати за викликами підпрограм під час моделювання. У верхній частині вікна відображається ім'я поточної підпрограми (процесу), а у самій таблиці – імена, типи і поточні значення формальних параметрів, а також локальних змінних, констант і файлів, оголошених у даній підпрограмі.

6.2.5 Вікно потоку даних – Data Flow

Вікно потоку даних дозволяє наочно, в графічному вигляді, відображати зв'язок між сигналами і процесами проекту, що моделюється. Воно викликається за допомогою опції **Data Flow** в меню **View** і має дві форми відображення інформації.

Відображення відносно процесу (рис. 6.7) – це рисунок, на якому даний процес зображено прямокутником. Лінії, що підведені до нього зліва, відповідають вхідним сигналам цього процесу, а лінії справа – його вихідним сигналам.

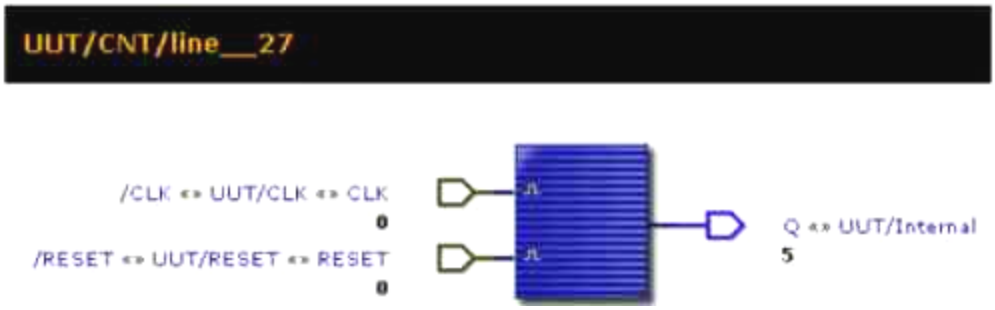


Рисунок 6.7 – Вікно потоку даних. Відображення відносно процесу

Відображення відносно сигналу (рис. 6.8) має вигляд вертикального відрізка, яким позначається даний сигнал. Лінії, що підведені до нього зліва, показують, які процеси беруть участь у формуванні даного сигналу, а лінії справа - які процеси використовують даний сигнал як вхідний.

Клацнувши мишкою на назві процесу, переходять до відображення відносно даного процесу, а клацнувши на назві сигналу - до відображення відносно даного сигналу.

В обох формах відображення поряд з іменами сигналів вказуються їх поточні значення.

Імена процесів і сигналів подаються з префіксами, що вказують на їх місце у ієрархії модулів проекту, наприклад: **/CLK** – сигнал **CLK** з кореневого модуля проекту, **UUT/CNT/line_27** – процес, що описаний починаючи з 27-го рядка у модулі **CNT**, який входить до складу модуля **UUT**. У відображеннях відносно процесу префікси сигналів опускаються, якщо вони співпадають з префіксами даного процесу. Наприклад, на рис. 6.7 сигнали, що вказані без префіксів: **CLK**, **RESET**, **Q**, фактично є сигналами, які мають такий самий префікс, як і процес, а саме, **UUT/CNT/**.

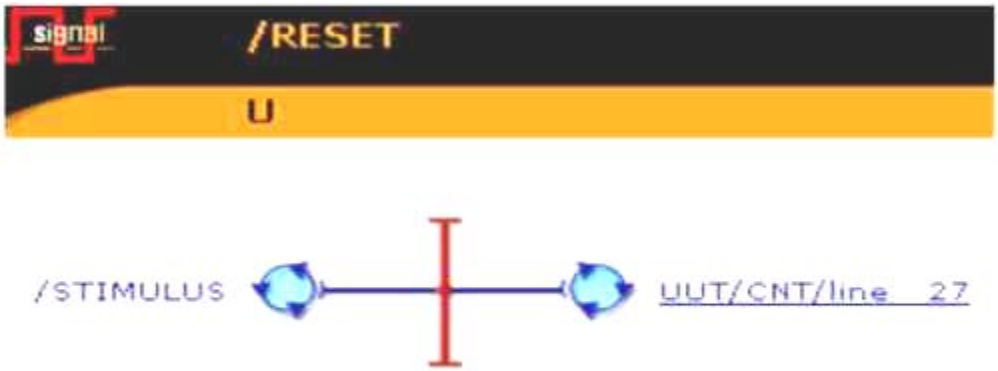


Рисунок 6.8 – Вікно потоку даних, відображення відносно сигналу

Імена сигналів є локальними іменами модуля. Через це той самий сигнал може по-різному позначатись у різних модулях проекту. Це добре видно на відображеннях відносно процесу. Наприклад, з рис. 6.7 випливає, що сигнал **Q** у поточному модулі **UUT/CNT** є те саме, що і сигнал **Internal** з модуля **UUT**.

Опція **View Source** з контекстного меню, що викликається при натисканні правої клавіші мишки на імені сигналу чи процесу, дозволяє швидко переходити від вікна потоку даних до опису цих сигналів та процесів у програмах. Подібні переходи до програм і до відображень потоку даних можливі і з вікна перегляду проектів **Design Browser**.

У вікні потоку даних запам’ятовуються відображення, що переглядались. Це дає змогу, користуючись кнопками, повертатись відповідно до попереднього чи наступного відображення, подібно до перегляду web-сторінок у Internet.

6.3 Моделювання методом ручного стимулювання

Щоб виконати моделювання методом ручного стимулювання входів треба:

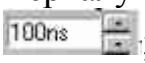
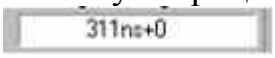
- проініціалізувати моделювальний пристрій Active-HDL за допомогою опції **Initialize Simulation** із меню **Simulation**;
- відкрити вікно часової діаграми та, згідно з рекомендаціями, описаними у розділі 4.8, додати до нього сигнали, які ви бажаєте відобразити на діаграмі й призначити стимулятори вхідним сигналам;
- виконати один або декілька циклів моделювання проекту, користуючись описаними нижче кнопками панелі інструментів чи відповідними опціями меню **Simulation**. При цьому результати моделювання відображатимуться у вікні **Редактора часових діаграм**.
- завершити моделювання, вибравши опцію **End Simulation** в меню **Simulation**.

Active-HDL дозволяє виконувати покрокове моделювання (трасування) проекту з зупинкою перед виконанням кожного оператора програми. Оператор, що підлягатиме виконанню, при цьому виділяється у тексті програми жовтим кольором.

Якщо у програмі є звернення до підпрограм, то кнопка

-  **Trace into** – виконує моделювання, зупиняючись перед виконанням кожного оператора як у головній програмі, так і у підпрограмах, що нею викликаються;
-  **Trace over** – виконує усю підпрограму за один крок;
-  **Trace out** – виконує за один крок стільки операторів, скільки потрібно для завершення виконання поточної підпрограми.

Крім того, Active-HDL дозволяє виконувати моделювання протягом заданого часу:

-  **Run For -** виконує моделювання протягом інтервалу часу, встановленого у розташованому справа від неї вікні  ;
-  **Run until -** дозволяє ввести момент часу, коли моделювання буде зупинено. Мається на увазі час, що відраховується від самого початку процесу моделювання і відображається у вікні  Simulation status, розташованому справа від кнопок Trace;
-  **Run -** забезпечує неперервний процес моделювання, що зупиняється лише тоді, коли стимулювальні сигнали вичерпані або коли досягнуто контрольної точки.

Контрольні точки – це такі точки в програмі, де її виконання тимчасово зупиняється. Вони вводяться і знімаються за допомогою клавіші

F9 або опції **Toggle Breakpoint** у меню **Simulation** і відображаються в тексті програми значком  .

Зупинку в контрольній точці можна зробити як безумовною, так і умовною. В останньому випадку треба виконати команду **Edit Breakpoint** з меню **Simulation** і у панелі **Signal Breakpoint** ввести значення сигналів, що визначатимуть умови зупинки в даній контрольній точці.

Для поновлення виконання програми, після зупинки у контрольній точці, можна скористатися вказаними вище кнопками **Trace** чи **Run**.

Для зняття усіх встановлених у програмі контрольних точок можна скористатися опцією **Clear All Breakpoints** з меню **Simulation**.

6.4 Моделювання на Випробувальному стенді – Test Bench

Розглянуте вище ручне стимулювання є найпростішим і швидким методом моделювання проектів. Воно може застосовуватись до будь-якого сигналу чи порту в ієрархії проекту. Це зручний засіб відлагодження процесів низького рівня. Проте його недоліками є те, що стимулятори не можуть застосовуватись у складніших процедурах моделювання, наприклад таких, що включають читання файлів тощо. Крім того, стимулятори є частиною самого Active-HDL, а не розроблюваного проекту, тобто – в інші середовища розробки проектів вони не переносяться.

Вище вже відзначалось, що моделювання проектів – це процес, аналогічний тестуванню програм. За цієї аналогії ручному стимулюванню відповідає таке тестування, де підготовку наборів вхідних даних для тестів та аналіз результатів їх виконання забезпечує сам програміст.

Відомо, що досконалішою формою тестування є розробка спеціальних програмних модулів, які генерують вхідні дані для програм, що підлягають тестуванню, та аналізують результати, які ті виробляють. Ці програмні модулі є важливою складовою частиною проекту. Вони використовуються щоразу після внесення до проекту змін і дозволяють швидко і в повному обсязі випробовувати нові версії програм.

При розробці проектів на VHDL роль вказаних службових модулів виконують файли, що називаються ***Файлами випробувального стенда***. Зазначимо, що у стандарті мови VHDL є ряд функцій і мовних конструкцій, розроблених спеціально для моделювання. Дані для моделювання можна читати з текстового файла, створювати окремі процеси, що управляють вхідними портами і т.п.

Типовий спосіб побудови випробувального стенда **Test bench** (рис. 6.9) полягає в створенні додаткових VHDL-файлів, які трактують проект, що моделюється, як компонент **UUT** (**Unit Under Test** – Піддослідний пристрій) і включають також програму **Генератор стимулів** (**Stimulus Generator**), що виробляє тестові вектори, а також програму **Верифікатор** (**Verifier**), що аналізує результати моделювання.

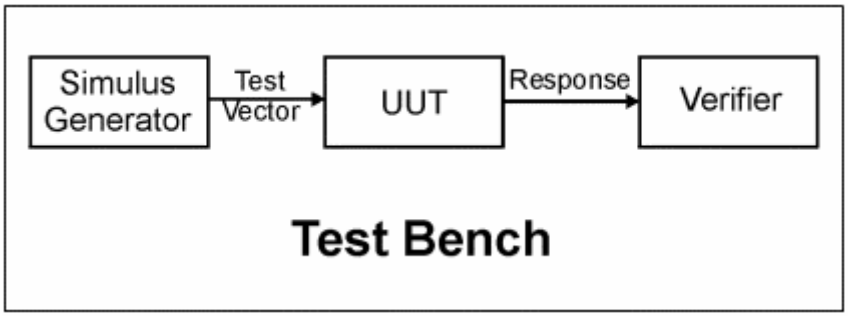


Рисунок 6.9 – Модель випробувального стенда

Така методологія забезпечує найповнішу перевірку проекту при мінімальній взаємодії з користувачем.

Розрізняють три конфігурації випробувальних стендів.

- У автономній конфігурації (*off-line configuration*) тестові вектори і еталони верифікатора записані у файлі, отже, моделювання проекту зводиться до подачі на вхід UUT сигналів, що зчитуються з файла і порівняння вихідних сигналів UUT з даними у файлі.
- У неавтономній конфігурації (*on-line configuration*) замість еталонного файла застосовують еталонний випробовуваний пристрій UUTe, на вхід якого подають ті самі тестові вектори, що і на вхід основного UUT. При цьому функція верифікатора зводиться до порівняння сигналів на виходах основного і еталонного випробовуваних пристроїв.
- Якщо у двох попередніх конфігураціях послідовність тестових векторів, що подається на вхід UUT, відома наперед, то у адаптивній конфігурації наступні тестові вектори вибираються з урахуванням результатів попереднього тестування та стану самого досліджуваного пристрою.

6.4.1 Створення і використання випробувальних стендів

З огляду на стандартну структуру випробувального стенда **Test bench**, роботу з його створення можна автоматизувати за допомогою **Майстра випробувального стенда Test Bench Wizard**, що викликається командою **Generate Test Bench** з меню **Tools** або з контекстного меню іконки **Entity/Architecture** досліджуваного пристрою UUT, розташованої на панелі **Files** вікна **Design Browser**.

- У першому вікні **Майстра** можна уточнити імена сутності та архітектури досліджуваного пристрою UUT, а також вибрати який саме тип випробувального стенда ви бажаєте побудувати: **Single Process** – одиночний процес чи **WAVES based** – заснований на часовій діаграмі. Перший забезпечує тільки подачу тестових сигналів на вхід, другий, крім того, порівнює вихідні сигнали зі зразками, що зберігаються у файлі, та записує інформацію про виявлені невідповідності у *log*-файл.

- У другому вікні вказують, звідки взяти тестові вектори для портів досліджуваного пристрою UUT. Якщо ви не створюєте їх вручну, а запозичуєте з існуючих файлів, то треба поставити прапорець у полі **Test vectors from file** і або безпосередньо ввести ім'я файла у поле **Select a test vector file**, або вибрати цей файл зі списку, скориставшись кнопкою **Browse**. Для **WAVES based** випробувальних стендів ним може бути або файл часової діаграми, або векторний файл *.vec. Для стендів **Single Process** це повинен бути файл *.vhs, що містить опис сигналів на VHDL у вигляді процесу. Якщо файл тестових векторів вибраний коректно, то у вікні **Signals found in file** повинні бути наявними усі сигнали, що є у вікні **UUT entity ports**.
- У третьому вікні **Майстра** вказують імена сутності та архітектури випробувального стенда, а також ім'я файла та папки, де вони знаходитимуться.
- Останнє, четверте вікно відображає повний список файлів, що будуть створені **Майстром**. Відмітивши прапорець, що вказує на потребу генерації файла конфігурації, та натиснувши кнопку **Готово**, ви сформуєте набір файлів випробувального стенда, що знаходитиметься у окремій папці, доступ до якої відбувається через панель **Files** у вікні **Design Browser**.

VHDL-файли випробувального стенда позначаються іконкою . Крім них, у папці є файли інших типів, зокрема, файл макрокоманд, помічений іконкою . Усе, що треба зробити для того, щоб провести моделювання - це запустити даний файл на виконання, клацнувши на ньому правою клавішею мишки і вибравши у контекстному меню команду **Execute**. В результаті у вікно **Console** будуть виведені повідомлення про виконання записаних у цьому файлі макрокоманд, які забезпечують компіляцію файлів як самого проекту, так і випробувального стенда; створення нової часової діаграми; внесення до неї потрібних сигналів і виконання моделювання. Після появи повідомлення про завершення моделювання: **Simulation has finished**... слід натиснути кнопку **ОК** і перейти до перегляду часової діаграми з результатами.

Випробувальні стенди **Single Process** просто відображають на часовій діаграмі сигнали досліджуваного пристрою.

На часових діаграмах **WAVES based** випробувальних стендів відображаються не тільки вихідні сигнали, фактично сформовані досліджуваним пристроєм UUT, а й еталонні вихідні сигнали, запозичені з часових діаграм тестових векторів. На діаграмі до їх імен додаються префікси ACTUAL_ та EXPECT_, відповідно, а до імен вхідних сигналів додається префікс STIM_. Крім того, ці часові діаграми містять деякі службові сигнали, зокрема сигнал ERR_STATUS, що дорівнює “1” там, де фактичні значення вихідних сигналів UUT не збігаються з еталонними.

У текстовому вигляді інформацію про невідповідність значень фактичних та еталонних сигналів можна знайти у файлі < ім'я_UUT>_report.log, доступ до якого можна одержати через папку **Logs** панелі Resources у вікні **Design browser**.

6.5 Макрокоманди моделювання

Active-HDL дозволяє управляти моделюванням шляхом вводу макрокоманд у вікно **Console**. Макрокоманди можуть застосовуватись для надання значень сигналам, покрокового виконання моделювання тощо.

Наведемо приклади деяких макрокоманд:

Wave	- створює порожній файл часової діаграми,
Wave RESET	- розміщує сигнал RESET у файлі часових діаграм,
Force RESET 1	- надає сигналові RESET значення “1”,
Force LOAD 1 0 ns, 0 10 ns	- установлює сигнал LOAD у “1” при 0 нс і в “0” при 10 нс,
Run 20 us	- виконує моделювання протягом 20 мкс.

Щоб зекономити час на введення з консолі великої кількості макрокоманд, їх можна записати у файл, а потім виконати його. До цього файла можуть бути включені не тільки макрокоманди моделювання, а й інші команди управління робочим середовищем Active-HDL, а також команди Script Basic, що включені в Active-HDL з метою автоматизації.

Для створення файла макрокоманд треба вибрати у меню **File** опції **New** і **Macro**, а щоб виконати його, потрібно або вибрати команду **Execute** у контекстному меню цього файла, або вибрати у меню **Tools** команду **Execute m**.

7 ЛАБОРАТОРНІ РОБОТИ

7.1 Лабораторна робота №1

Вивчення інтегрованого середовища автоматизованого проектування Active-HDL фірми Aldec Inc. Опис інтерфейсу та архітектури найпростіших об’єктів (сутностей) – цифрових пристроїв.

Мета роботи: ознайомитись з принципами автоматизованого проектування ПЛІС за допомогою Active-HDL. Вивчити структуру VHDL-проекту. Навчитися працювати з засобами управління проектом (Майстром проекту, Вікном перегляду проекту тощо). Навчитися описувати інтерфейси та архітектури найпростіших цифрових пристроїв.

Порядок виконання роботи

1. Вивчити склад та призначення робочого інтегрованого Active-

HDL та його основних компонентів (розділ 4 “Робоче середовище Active-HDL”).

2. Вивчити порядок застосування та функціональні можливості Майстра Нового Проекту (***New Design Wisard)***, розглянути поняття проекту та його складових (розділ 5 “Створення проекту”).
3. Створити за допомогою Майстра Нового Проекту (***New Design Wisard)*** новий проект.
4. Створити та описати на VHDL інтерфейс та архітектуру об’єкта найпростішого цифрового пристрою заданого відповідно до варіанта завдання на лабораторну роботу. Для опису архітектури об’єкту найпростішого цифрового пристрою необхідно ознайомитися з “Основними теоретичними відомостями про мову VHDL” в розділах 1-3 та прикладами опису архітектур об’єктів в Мовному помічникові (розділ 5.3.1 “Використання Мовного помічника - **Language Assistant**”).
5. Виконати компіляцію створеного проекту (розділ 5.6 “Компіляція проекту”).
6. Промоделювати створений цифровий пристрій (розділ 6 “Моделювання проекту ”).
7. Підготувати до захисту звіт.

Зміст звіту

До звіту необхідно включити:

а) назву та мету виконання лабораторної роботи;

б) опис засобів Active-HDL для управління проектами;

в) опис структури VHDL-проекту;

д) склад та структуру проекту, сформованого в результаті роботи;

е) об’єкт проекту, сформований в результаті роботи, та його інтерфейс (лістинг файла в якому описаний інтерфейс об’єкта та його архітектура);

є) часові діаграми створеної моделі цифрового пристрою;

ж) висновки.

Контрольні запитання

1. Опишіть склад та структуру проекту в Active-HDL.
2. Перерахуйте можливості Майстра Нового Проекту (***New Design Wisard)***.
3. Які засоби надаються Active-HDL для управління проектом?
4. Що таке провідник проекту та яке його призначення?
5. Перерахуйте основні компоненти інтегрованого середовища Active-HDL.
6. Що таке інтерфейс? Що таке архітектура об’єкта?
7. Яким чином описуються об’єкти в VHDL?
8. Що таке сигнал в VHDL та які його основні властивості?
9. Опишіть роботу оператора направлення сигналів та порядок

формування транзакцій сигналів.

10. Як встановлюються затримки сигналів? Які ви знаєте види затримок сигналів в VHDL? Дайте їх характеристику.
11. Що таке конкурентні оператори, та чим вони відрізняються від послідовних?
12. Дайте визначення процесу VHDL. Що таке список чутливості процесу, перерахуйте основні особливості роботи процесів?
13. Яке призначення оператора Wait? Опишіть його роботу.
14. Назвіть основні структури мови VHDL.
15. Опишіть роботу оператора вибора. Для чого він використовується?
16. Назвіть відомі вам VHDL-типи даних та розкажіть про правила перетворення типів.
17. Які Ви знаєте різновиди циклів VHDL?
18. Для чого використовуються константи Generic? Наведіть приклад.
19. Опишіть процес компонування складних об’єктів із структурних складових.
20. Що таке перекриття сигналів?
21. Чим відрізняються сигнали та змінні? Наведіть приклади.
22. Що таке компіляція проекту та навіщо вона використовується?
23. Опишіть послідовність моделювання проекту.

Завдання на лабораторну роботу №1

Номер варіанта	Назва пристрою	Перелік портів		
		Назва порту	Напрям порту	Тип порту
1.	Динамічний Д-тригер	R (інверсний)	in	STD_LOGIC
		D	in	STD_LOGIC
		C (перед. фронт)	in	STD_LOGIC
		S (інверсний)	in	STD_LOGIC
		Q1	out	STD_LOGIC
		Q2 (інверсний)	out	STD_LOGIC
2.	Трирозрядний дешифратор	X	in	STD_LOGIC_VECTOR (2 downto 0)
		F	out	STD_LOGIC_VECTOR (7 downto 0)
3.	JK - тригер	S (інверсний)	in	STD_LOGIC
		J	in	STD_LOGIC
		K	in	STD_LOGIC
		C (задн. фронт)	in	STD_LOGIC
		R (інверсний)	in	STD_LOGIC
		Q1	out	STD_LOGIC
		Q2 (інверсний)	out	STD_LOGIC
4.	Трирозрядний шифратор	X	in	STD_LOGIC_VECTOR (2 downto 0)
		F	out	STD_LOGIC_VECTOR (7 downto 0)
5.	Чотирирозрядний мультиплексор	X	in	STD_LOGIC_VECTOR (15 downto 0)
		F	out	STD_LOGIC
6.	Чотирирозрядний демультиплексор	X	in	STD_LOGIC
		Y	in	STD_LOGIC_VECTOR (1 downto 0)
		F	out	STD_LOGIC
7.	Дворозрядний суматор	P0	in	STD_LOGIC
		A	in	STD_LOGIC_VECTOR (1 downto 0)
		B	in	STD_LOGIC_VECTOR (1 downto 0)

Номер варіанта	Назва пристрою	Перелік портів		
		Назва порту	Напрям порту	Тип порту
		S	out	STD_LOGIC_VECTOR (1 downto 0)
		P	out	STD_LOGIC
8.	RS - тригер	R	in	STD_LOGIC
		S	in	STD_LOGIC
		Q1	out	STD_LOGIC
		Q2 (інверсний)	out	STD_LOGIC
9.	Чотирозрядний двійковий лічильник	R	in	STD_LOGIC
		C (задн. фронт)	in	STD_LOGIC
		Q	out	STD_LOGIC_VECTOR (4 downto 0)
10.	Двійково-десятковий лічильник	R	in	STD_LOGIC
		C (перед. фронт)	in	STD_LOGIC
		Q	out	STD_LOGIC_VECTOR (4 downto 0)

7.2 Лабораторна робота №2

Дослідження цифрових компонентів стандартних бібліотек середовища Active-HDL

Мета роботи: навчитися працювати з бібліотеками в середовищі Active-HDL та дослідити роботу типових цифрових компонентів стандартної бібліотеки середовища Active-HDL lat_vhd

Порядок виконання роботи

1. Вивчити основні відомості про роботу з редактором блок-схем (розділ 5.5 “Робота з редактором блок-схем”).
2. Створити новий пустий проект в Active-HDL (розділ 5. “Створення проекту”).
3. Створити в проекті новий файл редактора блок-схем (**Block diagram**) з назвою **Container** з використання Майстра нового файла (розділ 5.2 “Використання Майстра нового файла – **New Source File Wizard**”) або за командою **File→New→Block Diagram**.
4. Активізувати вікно **Symbols Toolbox** та додати до проекту бібліотеку lat_vhd (розділ “5.5.2.1 Symbols Toolbox”).
5. Знайти в бібліотеці lat_vhd заданий за назвою, згідно з варіантом завдання на лабораторну роботу, компонент та нанести його на схему (розділ 5.5.2 “Розміщення і редагування елементів”).
6. Нанести на схему такі самі вхідні та вихідні порти, як і в досліджуваного компонента <імена портів сформувати з імен портів досліджуваного компонента з префіксом my_> (розділ “5.5.1 Розміщення і редагування портів”).
7. Скомпілювати створений об’єкт (меню **Design\Compile** або клавіша <**F11**>) (розділ 5.5.5 “Компіляція схемного файла у програму на VHDL”).
8. Уважно дослідити VHDL-код створеної схеми. Особливо звернути увагу, яким чином в VHDL-файл включено вибрані бібліотеки та яким чином створено карту портів.
9. Змоделювати створений об’єкт (розділ 6 “Моделювання проекту”).
10. Підготувати до захисту звіт.

Зміст звіту

До звіту необхідно включити:

а) назву та мету виконання лабораторної роботи;

б) опис компонента Active-HDL Symbol Toolbox ;

в) зображення створеної блок-схеми;

д) лістинг HDL-файла створеної блок-схеми *Container.vhd*;

е) часові діаграми створеного об’єкта;

ж) висновки.

Контрольні запитання

1. Опишіть інструмент AHDL Symbol Toolbox. Яке його призначення?
2. Яким чином можна додати бібліотеку в AHDL-проект?
3. Яка синтаксична конструкція використовується при доданні бібліотек в VHDL-код?
4. Яким чином виконується декларування компонентів AHDL-бібліотек?
5. Опишіть процес використання компонентів VHDL-бібліотек. Які при цьому використовуються конструкції?
6. Яким чином можна переглянути опис компонентів, розміщених на блок схемі?

Завдання на лабораторну роботу №2

Номер варіан-ту	Функціональне призначення елемента	Назва елемента в бібліотеці lat_vhd
1	Восьмирозрядний суматор	addf8
2	Восьмирозрядний компаратор	cmp8
3	Чотирирозрядний дешифратор	dec4
4	Чотирирозрядний дешифратор – мультиплексор	dmux4
5	Чотирихрозрадний двохкаскадний дешифратор – мультиплексор	dmux24
6	Чотирирозрядний мультиплексор	mux4
7	Дворозрядний чотирьохкаскадний мультиплексор	mux24
8	Чотирирозрядний D-тригер з позитивним фронтом спрацювання	fd14
9	JK-тригер з позитивним фронтом спрацювання	fjk11
10	T-тригер з позитивним фронтом спрацювання	tf11

Примітка: звернути особливу увагу на те, що в даних моделях електронних компонентів, як, наприклад, T – тригер, наявні окремі входи для початкової ініціалізації (установлення). В моделі T-тригера – це вхід ***CD*** (Варіант 9). Поки правильно не буде проведена початкова ініціалізація, то модель не буде працювати. В T-тригері – це подача логічної одиниці на вхід ***CD***, при цьому тригер скидається в нуль при приході фронту. Аналогічна ситуація і для JK-тригера (**Варіант 10**). Але в цьому випадку встановлювальний вхід відсутній. Для початкової ініціалізації (скидання в нуль) потрібно використовувати комбінацію на входах: *J0=1, K0=0*. У всіх інших випадках, якщо виникають проблеми з ініціалізацією, слід вивчати VHDL-код опису компонента. Переглянути VHDL-код компонента можна за командою **Push**.

7.3 Лабораторна робота №3

Побудова принципової цифрової схеми та її моделювання в середовищі Active-HDL

Мета роботи: навчитися будувати принципові цифрові схеми з використанням примітивів стандартної бібліотеки середовища Active-HDL lat_vhd та моделювати розроблені схеми

Порядок виконання роботи

1. Відповідно до власного завдання на лабораторну роботу виконати мінімізацію заданої логічної функції.
2. Розглянути основні відомості про роботу з редактором блок-схем (розділ 5.5 “Робота з редактором блок-схем”).
3. Створити новий пустий проект Active-HDL (розділ 5 “Створення проекту”).
4. Створити в проекті новий файл редактора блок-схем (Block diagram) з назвою Diagram з Використання Майстра нового файла (розділ 5.2 “Використання Майстра нового файла – New Source File Wizard”) або за командою File→New→Block Diagram.
5. Активізувати вікно Symbols Toolbox та додати до проекту бібліотеку lat_vhd (розділ “5.5.2.1 Symbols Toolbox”).
6. Знайти в бібліотеці lat_vhd необхідні примітиви логічних елементів (І, АБО, НЕ та ін.), які необхідні для реалізації заданої логічної функції, нанести їх на схему та відповідним чином з’єднати (розділ 5.5.2 “Розміщення і редагування елементів”).
7. Нанести на схему вхідні та вихідні порти, якщо цього не було зроблено в Майстрові нового файла (зазначимо, що в даному випадку вхідними портами будуть аргументи логічної функції, що реалізується: X1, X2, X3, X4, а вихідними – значення функції: Y) (розділ “5.5.1 Розміщення і редагування портів”).
8. Скомпілювати створений об’єкт (меню Design\Compile або клавіша <F11>) (розділ 5.5.5 “Компіляція схемного файла у програму на VHDL”).
9. Промоделювати створений об’єкт (розділ 6 “Моделювання проекту”) та перевірити правильність роботи схеми.
10. Підготувати до захисту звіт.

Зміст звіту

До звіту необхідно включити:

а) назву та мету виконання лабораторної роботи;

б) мінімізацію заданої логічної функції;

в) зображення створеної блок-схеми;

д) лістинг та аналіз HDL-файла створеної блок-схеми Diagram.vhd;

е) часові діаграми створеного об’єкта та їх аналіз;

ж) висновки.

Контрольні запитання

1. Що таке логічна функція? Які є форми подання логічних функцій?
2. Що таке мінімальна форма логічної функції? Яким чином виконується мінімізація логічних функцій?
3. Яким чином виконується підключення бібліотек в середовищі AHDL? Які мовні конструкції VHDL використовуються для підключення бібліотек?
4. Які бібліотеки доступні в AHDL за замовчуванням?
5. Що розуміють під компонентами в VHDL? Яким чином виконується опис, декларування та використання компонентів?
6. Що таке стимулятори та яке їх призначення? Перерахуйте стандартні стимулятори AHDL вікна **Stimulators** та опишіть кожен з них.

Завдання на лабораторну роботу №3

Для визначення власного варіанта логічної функції слід виконати такі дії: номер свого варіанта перевести в двійкову систему числення і записати його шість молодших розрядів в вигляді слова <a6 a5 a4 a3 a2 a1>, відповідні значення a_i поставити в таблицю і отримати таблицю істинності заданої функції.

Наприклад, якщо номер варіанта 5(000101), то a6=0, a5=0, a4=0, a3=1, a2=0, a1=1.

X4	X3	X2	X1	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	a1
0	0	1	1	1
0	1	0	0	a2
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	a3
1	0	0	1	a4
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	a5
1	1	1	1	a6

7.4 Лабораторна робота №4

Розробка і моделювання паралельного та зсувного регістрів

Мета роботи: засвоїти принципи роботи паралельного та зсувного регістрів. Навчитися описувати синхронізовані процеси та використовувати циклічні оператори при описанні поведінки об'єктів за допомогою VHDL

Постановка задачі

Необхідно описати за допомогою VHDL роботу паралельного та зсувного регістрів, промоделювати їх та одержати часові діаграми.

Паралельний восьмирозрядний регістр має восьмирозрядний вхід *DATA_IN*(7 **downto** 0) типу *STD_LOGIC_VECTOR* для передачі даних, вхід синхронізації *CLK*, вхід дозволу на запис *WE* та вхід дозволу зчитування *RE* (типу *STD_LOGIC*). Крім того, регістр має восьмирозрядний вихід *DATA_OUT* (7 **downto** 0) для виводу даних (типу *STD_LOGIC_VECTOR*).

Робота регістра має здійснюватися таким чином:

У стані збереження байта на виході регістра постійно утримується високий імпеданс ("ZZZZZZZZ"). Це дасть змогу організувати роботу кількох регістрів на одну шину, так як, згідно таблиці перекриття сигналів (розділ 5.11 "Перекриття сигналів" теоретичної частини), сигнал високого імпедансу має найнижчий пріоритет.

Якщо WE = '1' і RE = '0', то здійснюється запис інформації до регістра.

Якщо WE = '0' і RE = '1', то на вихід регістра подається значення байта, що зберігається в цьому регістрі.

Всі інші комбінації WE та RE розглядаються як стан збереження байта.

Робота регістра має бути синхронізована за сигналом CLK.

Зсувний восьмирозрядний регістр має один вхід *DATA_IN* типу *STD_LOGIC* для вводу інформації, вхід синхронізації *CLK*, вхід дозволу на запис *WE* та вхід дозволу зчитування *RE* (типу *STD_LOGIC*). Крім того, регістр має восьмирозрядний вихід *DATA_OUT* (7 **downto** 0) для паралельного виводу даних (типу *STD_LOGIC_VECTOR*).

Робота регістра має здійснюватися таким чином:

У стані збереження байта на виході регістра постійно утримується високий імпеданс ("ZZZZZZZZ").

Якщо WE = '1' і RE = '0', то здійснюється запис інформації до регістра, при цьому сигнал DATA_IN надходить в DATA_OUT(0), значення DATA_OUT(0) зміщується в DATA_OUT(1) і т. д.

Якщо WE = '0' і RE = '1', то на вихід регістра подається значення байта, що зберігається в цьому регістрі.

Всі інші комбінації WE та RE розглядаються як стан збереження байта.

Робота регістра має бути синхронізована за сигналом CLK.

Рекомендації щодо написання програми:

Для збереження проміжної інформації доцільно застосовувати змінні типу *STD_LOGIC_VECTOR* (7 **downto** 0).

Для забезпечення синхронізації в список чутливості процесів доцільно поміщати сигнал *CLK*.

Зсув у зсувному регістрі слід реалізовувати за допомогою циклу з параметром (**for**).

Ефективнішим є зсув від старшого (7-го) біта до молодшого (0-го).

Порядок виконання роботи

1. Вивчити розділи 2.9 “Константи Generic”, та 3.4 “Цикли”, 2.5 “Процес”.
2. Розробити VHDL-модель паралельного восьмирозрядного регістра.
3. Промоделювати роботу паралельного регістра, розробленого в попередньому пункті, в режимах запису інформації, збереження байта та зчитування інформації.
4. Проаналізувати на основі одержаних часових діаграм відповідність роботи паралельного регістра заданому алгоритму.
5. Розробити VHDL-модель зсувного восьмирозрядного регістра.
6. Промоделювати роботу зсувного регістра, розробленого в попередньому пункті, в режимах запису інформації, збереження байта та зчитування інформації.
7. Проаналізувати на основі одержаних часових діаграм відповідність роботи зсувного регістра заданому алгоритму.

Зміст звіту

До звіту необхідно включити:

а) назву та мету виконання лабораторної роботи;

б) опис циклічних конструкцій, що використовуються в VHDL;

в) тексти VHDL-програм, що описують інтерфейс та архітектуру паралельного та зсувного регістрів;

д) часові діаграми роботи регістрів;

е) висновки.

Контрольні запитання

1. Опишіть роботу паралельного регістра.
2. Опишіть роботу зсувного регістра.
3. Перерахуйте типи циклічних операторів у VHDL.
4. Яким чином в VHDL реалізується цикл з параметром?
5. Яким чином в VHDL реалізується цикл “Поки”?
6. Яким чином в VHDL реалізуються вкладені цикли?

7.5 Лабораторна робота №5

Проектування та дослідження запам'ятовувальних пристроїв в середовищі Active-HDL

Мета роботи: вивчення основних методів побудови запам'ятовувальних пристроїв (ЗП), придбання навиків їх практичного використання і логічне моделювання оперативного запам'ятовувального пристрою (ОЗП) та постійного запам'ятовувального пристрою (ПЗП) в середовищі Active-HDL

Постановка задачі

Необхідно описати за допомогою VHDL роботу постійного запам'ятовувального пристрою (ПЗП) та оперативного запам'ятовуючого пристрою (ОЗП), промоделювати їх роботу та одержати відповідні часові діаграми.

Оперативний запам'ятовувальний пристрій (ОЗП) має восьмирозрядний вхід *DATA_IN*(7 **downto** 0) типу *STD_LOGIC_VECTOR* для передачі вхідних даних, восьмирозрядний вхід *ADDRESS*(7 **downto** 0) типу *STD_LOGIC_VECTOR* для визначення адреси комірки з якою будуть виконуватись операції (тим самим визначається розмір ОЗП $8\times256=2048$ Байтів), вхід синхронізації *CLK*, вхід дозволу на запис *WE* та вхід дозволу зчитування *RE* (типу *STD_LOGIC*), восьмирозрядний вихід *DATA_OUT* (7 **downto** 0) для виводу даних (типу *STD_LOGIC_VECTOR*).

Робота ОЗП має здійснюватися таким чином:

У стані збереження байта на виході ОЗП постійно утримується високий імпеданс ("*ZZZZZZZZ*").

Якщо WE = '0' і RE = '1', то на вихід ОЗП подається значення байта, що знаходиться за адресою ADDRESS.

Якщо WE = '1' і RE = '0', то в ОЗП записується байт DATA_IN за адресою ADDRESS.

Всі інші комбінації WE та RE розглядаються як стан збереження байта.

Робота ОЗП має бути синхронізована за сигналом CLK.

Постійний запам'ятовувальний пристрій (ПЗП) має васьмирозрядний вхід *ADDRESS*(7 **downto** 0) типу *STD_LOGIC_VECTOR* для визначення адреси комірки, з якої будуть вибиратися дані, вхід синхронізації *CLK*, вхід дозволу зчитування *RE* (типу *STD_LOGIC*), восьмирозрядний вихід *DATA_OUT* (7 **downto** 0) для виводу даних (типу *STD_LOGIC_VECTOR*). Таким чином визначається об'єм ПЗП $8\times256 = 2048$ Байтів = 2 Кбайти.

Робота ПЗП має здійснюватися таким чином:

У стані збереження байта на виході ПЗП постійно утримується високий імпеданс ("*ZZZZZZZZ*").

Якщо *RE* = '1', то на вихід ПЗП подається значення байта, що знаходиться за адресою *ADDRESS*.

Якщо *RE* = ’0’, то це розглядаються як стан збереження байта.
Робота ОЗП має бути синхронізована за сигналом *CLK*.

Порядок виконання роботи.

1. Ознайомитись з принципом роботи ПЗП і ОЗП та теоретичним матеріалом, наведеним у розділі 3.8 “Масиви”.
2. За допомогою Active-HDL описати роботу ПЗП та ОЗП:
   2.1. Створити новий проект.
   2.2. Додати бібліотеку *ieee.std_logic_unsigned*.
   2.3. Описати інтерфейси ОЗП та ПЗП (для цього створити два окремих VHDL-файли).
   2.4. Визначити тип масиву даних, які будуть зберігатись в ОЗП (ПЗП).
   2.5. Сформувати масив даних, які зберігаються в ПЗП (для ОЗП цього робити непотрібно).
   2.6. Описати процес вибірки/запису даних, які зберігаються в ОЗП (ПЗП).
3. Скомпілювати та промоделювати процес вибірки/запису даних для ОЗП (ПЗП). Результати моделювання представити у табличній формі за допомогою “List Viewer”, та у вигляді часових діаграм.
4. Виконати звіт.

Зміст звіту

У звіт необхідно включити:

а) назву та мету лабораторної роботи;

б) основні теоретичні відомості про ПЗП та ОЗП;

в) навести основні теоретичні відомості про використання масивів в Active-HDL;

г) навести VHDL-код для описання ПЗП та ПЗП;

д) навести результати моделювання у табличній формі та у вигляді часових діаграм;

е) зробити висновки.

Контрольні запитання

1. Що таке ПЗП і які його основні функції?
2. Що таке ОЗП, яке його призначення та основні функції?
3. Як описуються та використовуються масиви в Active-VHDL?
4. Яким чином можна задавати масиви із змінною розмірністю?
5. Роль бібліотеки *ieee.std_logic_unsigned* в програмі описання ПЗП?

7.6 Лабораторна робота №6

Моделювання транспортної та інерційної затримок часу в середовищі Active-HDL

Мета роботи: навчитися описувати транспортну та інерційну затримки часу за допомогою Active-HDL, дослідити роботу обох видів затримок часу та з'ясувати принципову різницю між ними

Порядок виконання роботи

1. Відповідно до власного завдання на лабораторну роботу (завдання на дану лабораторну роботу взяти з лабораторної роботи №3) виконати мінімізацію заданої логічної функції та привести її до базису І-АБО-НЕ.
2. Створити новий проект (розділ 5 "Створення проекту"), в якому створити два об'єкти (схеми) з назвами *inertial_diagram* (для схеми з інерційними затримками) та *transport_diagram* (для схеми з транспортними затримками). Описати інтерфейси створених об'єктів, які для кожного з них будуть мати однаковий вигляд: *: **X1, X2, X3, X4 -** вихідні порти (**in**) – аргументи функції: **Y –** вихідний порт (**out**)– значення функції*).
3. Описати архітектуру визначених в попередньому розділі об'єктів за допомогою VHDL-коду (користуючись лише логічними операторами VHDL). Врахувати, що кожен логічний елемент І має затримку (інерційну/транспортну) – **6ns**, кожен елемент АБО має затримку (інерційну/транспортну) – **5ns** та кожен елемент АБО має затримку (інерційну/транспортну) – **2ns**.
4. Скомпілювати створені об'єкти (меню ***Design\Compile*** або клавіша <**F11**>) (розділ 5.6 "Компіляція проекту").
5. Промоделювати роботу схеми для випадків: а) коли тривалість вхідного сигналу більша за інерційну затримку логічних елементів побудованої схеми; б) коли тривалість вхідного сигналу менша за інерційну затримку логічних елементів побудованої схеми (розділ 6 "Моделювання проекту").
6. Пояснити результати моделювання по отриманих часових діаграмах.
7. Створити новий об'єкт *regect_diagram*, описати його інтерфейс (який визначений в п. 2 даної лабораторної роботи) та його архітектуру, щоб через побудовану схему пропускалися лише імпульси, тривалість яких складає 7 ns і більше.
8. Скомпілювати створений об'єкт (меню ***Design\Compile***, або клавіша <**F11**>) (розділ 5.6 "Компіляція проекту") та промоделювати роботу схеми для випадків: а) коли тривалість вхідного сигналу більша за 7ns; б) коли тривалість вхідного сигналу менша за 7ns (розділ 6 "Моделювання проекту").

9. Пояснити результати моделювання.

10. Підготувати до захисту звіт.

Зміст звіту

До звіту необхідно включити:

а) назву та мету виконаної лабораторної роботи;

б) визначення транспортної та інерційної затримок, метод їх моделювання за допомогою Active-HDL;

в) принципову схему з'єднання логічних елементів, яку треба моделювати;

г) опис всіх схем (об'єктів) в VHDL-коді;

д) результати моделювання для різної тривалості вхідних сигналів (п.5) та їх порівняння;

е) результати моделювання для різної тривалості вхідних сигналів (п.8) та їх порівняння;

є) зробити висновки.

Контрольні запитання

1. Що таке інерційна затримка, який її фізичний зміст?
2. Що таке транспортна затримка, який її фізичний зміст?
3. Як описуються інерційна і транспортна затримки в VHDL?
4. Яким чином виконується фільтрація імпульсів заданої тривалості засобами VHDL?
5. Які логічні оператори наявні в VHDL?

Навчальне видання

Олександр Іванович Гороховський

« МОДЕЛЮВАННЯ В СЕРЕДОВИЩІ Active-HDL »

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Оригінал – макет підготовлено автором

Редактор Т.О. Старічек

Науково-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку Гарнітура Times New Roman
Формат 29,7×42¼ **Папір офсетний**

Друк різографічний Ум. друк. арк.
Тираж прим.
Зам №

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ