УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КЕРУЮЧИХ СИСТЕМИ ТА ТЕХНОЛОГІЙ

Кафедра спеціалізованих комп'ютерних систем

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт та самостійної роботи

з дисципліни

«ПРОЕКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ ТА СИСТЕМ НА ПЛІС»

Харків – 2020

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри спеціалізованих комп’ютерних систем 24 лютого 2020 р., протокол № 9.

Описано основи проектування і верифікації складних цифрових систем та їх імплементації в кристали програмувальної логіки, використання мов опису апаратури (HDL та Verylog) з технологіями структурного і поведінкового опису проектів.

Методичні вказівки призначено для студентів спеціальності 123 «Комп’ютерна інженерія», що вивчають дисципліну «Проектування цифрових пристроїв та систем на ПЛІС» денної та заочної форм навчання.

Укладачі:

проф. М. А. Мірошник,
доц. Л. А. Клименко,
асист. Ю. В. Савін

Рецензент

доц. Н. А. Корольова

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Ці методичні вказівки призначено для проведення лабораторних робіт з використанням програм моделювання Active-HDL і синтезу Synplify Pro, що входять до складу САПР цифрових пристроїв на ПЛІС. Для проведення відповідної лабораторної роботи студент повинен самостійно вивчити теоретичний матеріал із зазначеної теми за лекційним матеріалом та літературою, наведеною у методичних вказівках до цієї роботи

Перед виконанням лабораторної роботи студенти повинні скласти допуск з теоретичного матеріалу у співбесіді з викладачем. Студенти, які успішно склали допуск, переходять до роботи з програмами моделювання ActiveHDL та синтезу Synplify Pro. При роботі з програмами студенти ознайомлюються із завданням, яке вони будуть виконувати в ході проведення роботи, та методом його вирішення з даними, що використовуються як приклад. Кожен студент отримує індивідуальне завдання. Варіанти завдань надає викладач чи студенти самостійно обирають з таблиці, наведеної в методичних вказівках.

Після отримання результатів виконаного завдання студент оформлює індивідуальний електронний звіт з виконаної роботи, в разі потреби роздруковує його та складає викладачу у формі співбесіди за темою проведеної роботи й отримує відповідну оцінку.

За вказаною технологією лабораторні роботи з використанням програм моделювання Active-HDL і синтезу Synplify Pro студенти можуть виконувати в лабораторіях кафедри СКС чи індивідуально на власному персональному комп'ютері за згодою з викладачем. Звіти про виконання лабораторних робіт студенти здають викладачеві незалежно від місця їх виконання у зазначений час.

ЛАБОРАТОРНА РОБОТА 1. Вивчення синтезу простих операторів паралельного призначення сигналів

Мета роботи

Навчитися працювати в середовищі САПР Synplify Pro 7.0 на прикладі синтезу моделі повного суматора. Вивчити прості оператори паралельного призначення сигналів та методи їх синтезу.

1.1 Методичні вказівки з підготовки до виконання лабораторної роботи

Як приклад розглянемо VHDL-модель класичного однорозрядного суматора з переносом, наведену в лістингу 1.1. Відзначимо, що в моделі не задані затримки в операторах призначення сигналів. Ця інформація є надлишковою, оскільки результатом синтезу є структурна схема, що складається з найпростіших бібліотечних елементів системи синтезу, затримки яких і визначають точні часові параметри синтезованої схеми. Тому системи синтезу, в загальному випадку, ігнорують інформацію про затримки, описану в тексті VHDL-моделі. Для аналізу реальної поведінки проекту після синтезу в системах моделювання використовуються затримки найпростіших елементів з бібліотеки системи синтезу [1, 2].

Лістинг 1.1 – Однорозрядний суматор

```
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
port   (in1, in2, c_in: in std_ulogic;
sum, c_out: out std_ulogic);
end entity full_adder;
architecture dataflow of full_adder is
signal s1, s2, s3: std_ulogic;
begin
sum <= in1 xor in2 xor c_in;
```

```
c_out <= (in1 and c_in) or (in2 and c_in) or (in1 and in2);
end architecture dataflow;
```

Прості оператори паралельного призначення сигналів (ППС) використовуються для опису обчислення значення одного сигналу на підставі значень інших сигналів. Коли змінюється значення одного з вхідних сигналів, то змінюється і значення вихідного сигналу. Така поведінка відповідає комбінаційній логічній схемі, і тому система синтезу перетворить оператори ППС в елементи, що реалізують булеві функції.

Обробка VHDL-моделі в системі синтезу Synplify Pro здійснюється у два етапи. На першому етапі система переглядає VHDL-код і ставить йому у відповідність схему, що складається з бібліотечних примітивів. На другому етапі здійснюється перетворення отриманої схеми в схему, що складається з елементів обраного кристала (мікросхеми) програмованої логіки (ПЛІС – програмованих логічних інтегральних схем). На рисунку 1.1 наведений результат першого етапу синтезу. Результатом його є так звана RTL-схема (Register Transmition Level) [3, 4].

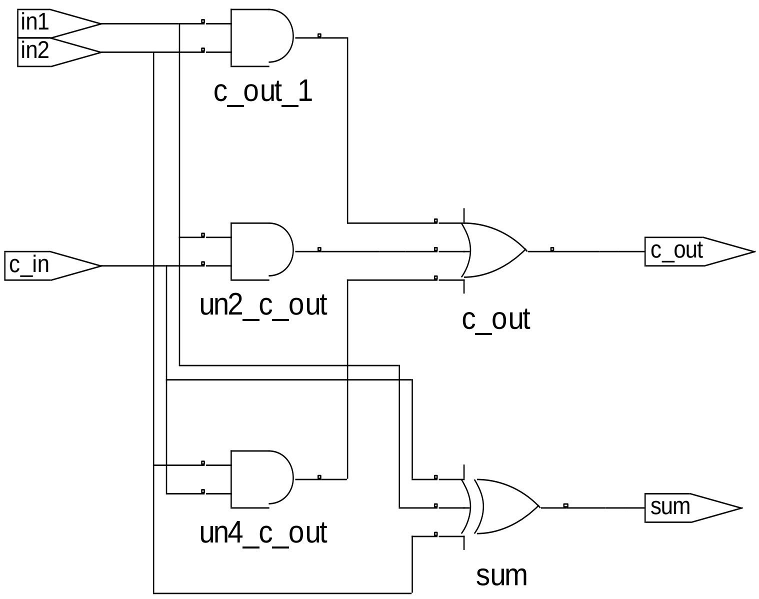


Рисунок 1.1 – Схема однорозрядного суматора після логічного виводу

У системі передбачена можливість виводу синтезованої схеми в графічному вигляді в режимі RTL-view. Зображена схема являє собою комбінаційну схему з входами даних in1 і in2, входом переносу c_in, виходом суми sum і виходом переносу c_out. Суть синтезу полягає в тому, що кожен оператор VHDL-моделі заміняється на один чи декілька бібліотечних елементів, з'єднаних між собою. Видно, що в першому операторі порядок операцій не визначений, а в другому – чітко визначений завдяки розміщенню дужок. Розміщення дужок в операторах ППС залежать від логічних функцій і порядку їх запису в операторі призначення сигналу. Якщо в операторі наявні більше двох операндів, пов'язаних між собою логічними операціями, то дужки можна не ставити тільки у випадку, якщо підряд ідуть кілька однакових операцій, а саме: AND, OR, XOR чи XNOR. Якщо ж в операторі використовуються різні логічні операції, то їх необхідно розділяти за допомогою дужок. Утім, якщо дужки в операторі призначення розставити некоректно, то система синтезу, виконуючи аналіз опису, видасть відповідне повідомлення про помилку [4].

На рисунку 1.2 наведений результат після виконання другого етапу синтезу.

Як елементна база обрана мікросхема XC4010E. Треба зазначити, що вибір типу мікросхеми здійснюється ще до першого етапу синтезу, бо вже тоді здійснюється підключення відповідних бібліотек. У цій схемі як примітиви використовуються LUT (Look-up-table – таблиця перекодування), що являють собою апаратні таблиці істинності у вигляді статичних оперативних запам’ятовувальних пристроїв (ОЗП). Елементи FMAP відповідають 4-вхідним LUT у FPGA цього сімейства. У схемі використовується 2 елементи FMAP, оскільки схема містить 2 виходи. Кожний з елементів FMAP реалізує комбінаційну функцію відповідного йому виходу схеми. У кристалі FPGA LUT містить таблиці істинності цих функцій. Вони подані на схемі у вигляді 16-ткових векторів поруч з назвою елемента FMAP (наприклад, FMAP3) після підкреслення. У таблиці 1.1 наведено приклад правильного прочитання цих векторів (тобто у зворотному порядку, починаючи з останнього рядка) [4].

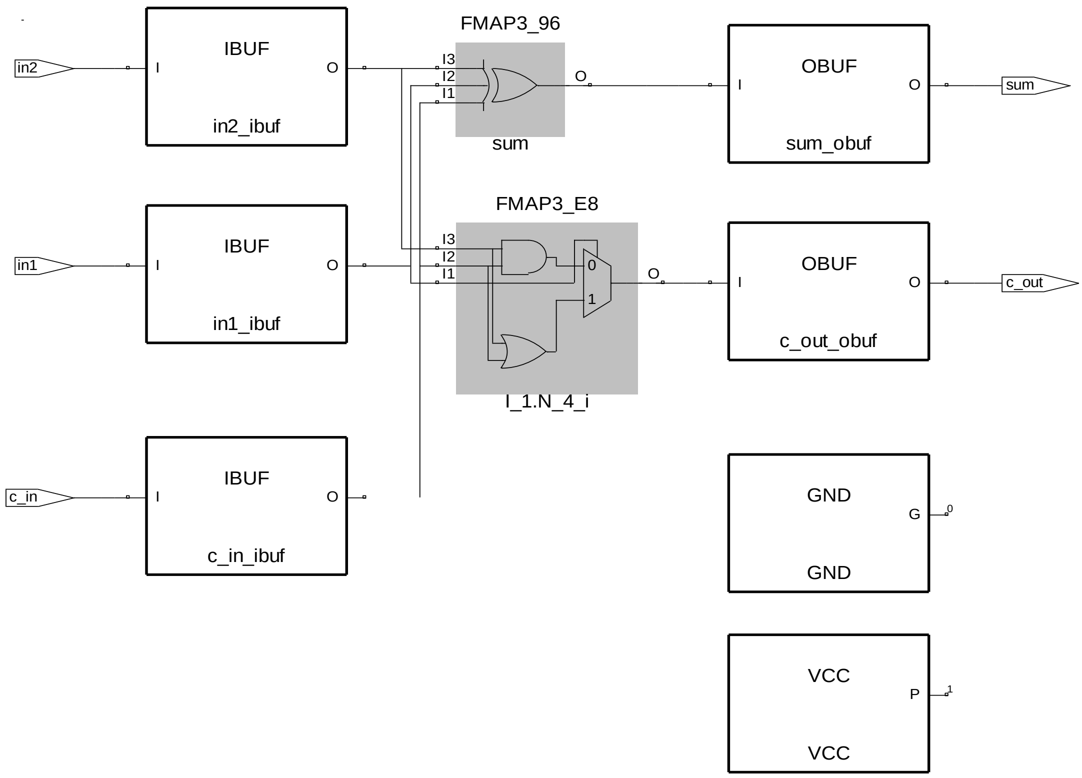


Рисунок 1.2 – Схема синтезованого суматора
на рівні технологічних примітивів FPGA Xilinx 4010E

Таблиця 1.1 – Таблиці істинності виходів суматора

Входи			Виходи			
in1	in2	c_in	sum		c_out	
Комбінації			Вміст LUT			
0	0	0	0	6	0	8
0	0	1	1		0	
0	1	0	1		0	
0	1	1	0		1	
1	0	0	1	9	0	E
1	0	1	0		1	
1	1	0	0		1	
1	1	1	1		1	

1.2 Порядок виконання роботи

Крок 1. Створіть каталог, у якому буде міститися новий проект.

Крок 2. Запустіть систему синтезу Synplify. Виберіть пункт меню File–>New. У вікні, що відкрилося, виберіть пункт "VHDL file", у рядку File Name вкажіть ім'я файла adder, а в File Location – шлях до місця, де міститиметься проект, і натисніть OK (рисунок 1.3).

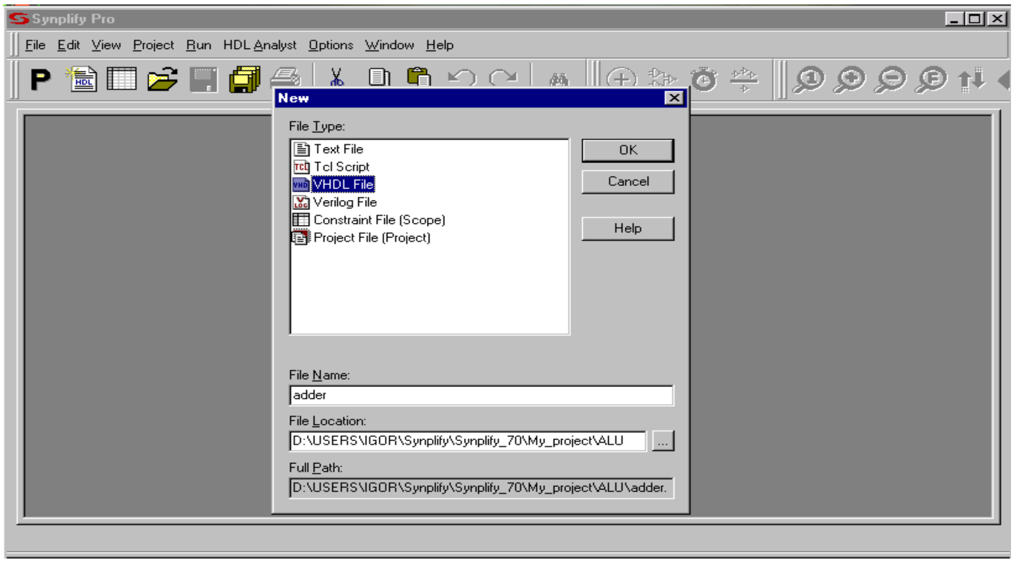


Рисунок 1.3 – Створення файла alu.vhd

Крок 3. У вікні, що відкрилося, [<adder>] опишіть VHDL-модель повного суматора (лістинг 1.1) .

Крок 4. Тепер необхідно створити проект для файла adder.vhd. Виберіть у меню пункт File–>Build Project. З'явиться вікно проекту adder.prj (рисунок 1.4).

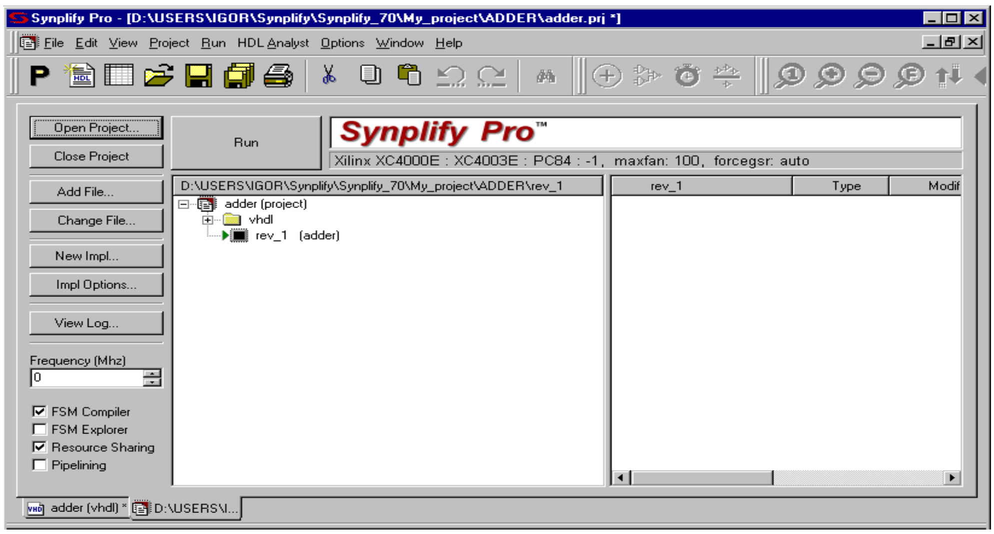


Рисунок 1.4 – Створення проекту в системі синтезу

Крок 5. Щоб вибрати тип ПЛІС, на якій буде синтезований проект, виберіть пункт меню Impl Options–>Set Device Options. У вікні, що з'явилося, є кілька випадних списків. У них треба встановити одне з таких значень за завданням викладача: Xilinx XC4000E, Altera APEX II, Quick Logic Quick PCI. Після цього вікно трохи зміниться, як показано на рисунку 1.5.

Крок 6. Виберіть пункт Write Mapped VHDL Netlist, який приведе до генерації VHDL-коду(файл adder.vhm), що описує схему, яка складається з елементів обраного кристала, з переліком фактичних затримок, як на рисунку 1.6. Інші настроювання можна залишити без змін.

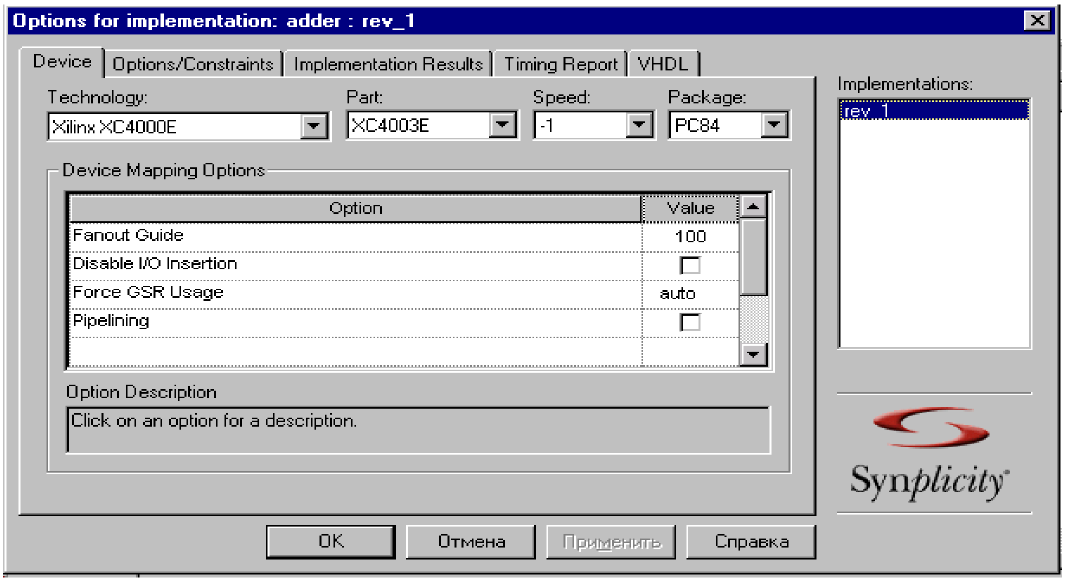


Рисунок 1.5 – Вибір установок системи синтезу

Крок 7. Запустіть синтез проекту, натиснувши кнопку Run. Після закінчення з'явиться напис Done!

Крок 8. Перегляньте результати логічного виводу (синтезу), вибравши один з пунктів, зазначених стрілками на рисунку 1.7. Відкриється вікно з графічним зображенням RTL-view, що є результатом компіляції VHDL-опису в бібліотечні примітиви системи синтезу.

Крок 9. Перегляньте остаточний результат синтезу, вибравши один з пунктів, зазначених стрілками на рисунку 1.7. У вікні, що відкрилося, буде графічне зображення синтезованого суматора з примітивів обраного типу мікросхеми.

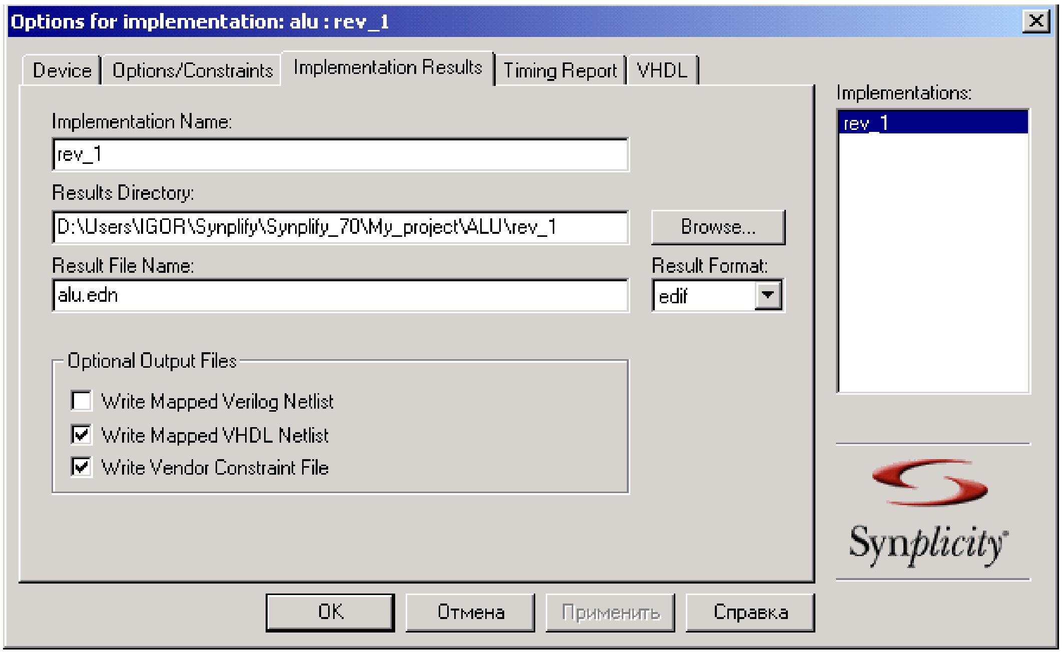


Рисунок 1.6 – Вікно завдання генерації файла adder.vhm

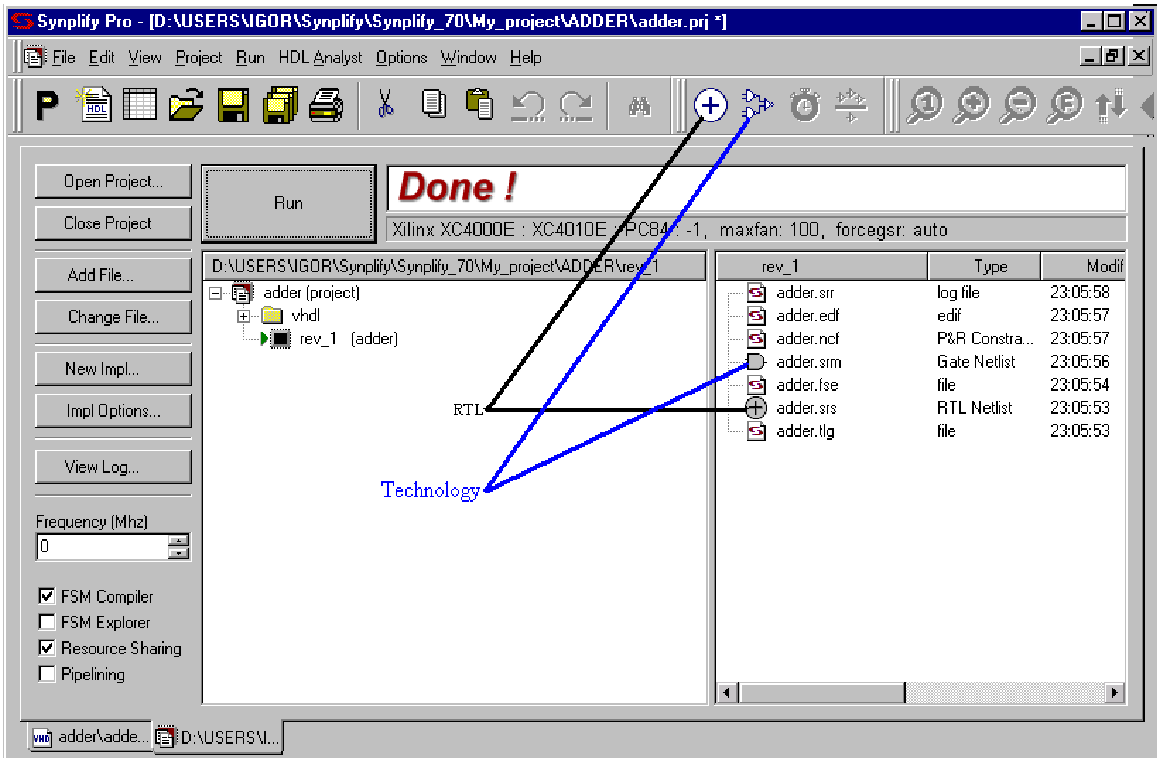
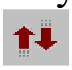


Рисунок 1.7 – Основне вікно проекту

Крок 10. Виберіть пункт меню View–>Push/Pop Hierarchy чи натисніть кнопку . Курсор миші набуде вигляду двосторонньої перекресленої стрілки. Якщо навести його на один з елементів FMAP, він набуде вигляду стрілки, спрямованої вниз. Натисніть один раз, і ви побачите еквівалентну схему для цього елемента FMAP. Щоб повернутися до попередньої схеми, досить натиснути кнопку миші один раз на будь-якій вільній ділянці схеми (коли стрілка спрямована вгору).

Крок 11. Виберіть у пункті меню Window вікно проекту adder.prj. Збережіть проект (File–>Save). Перегляньте файл звіту, що відкривається з натисканням кнопки View Log.

1.3 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити:

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- розроблені Verilog-моделі;
- код testbench для першого варіанта;
- результати тестування всіх пристроїв;
- порівняння результатів синтезу для різних типів кристалів (з додаванням графічних результатів роботи системи синтезу Synplify);
- результати автоматизованого синтезу проекту у вигляді графічного відображення схем тригерів;
- висновки до роботи.

Контрольні запитання

1 Які існують моделі цифрових систем?

2 Які існують технології проектування цифрових систем?

3 Які існують мови опису апаратури?

4 Що таке VHDL і Verilog?

5 Які існують етапи проектування цифрових пристроїв?

6 Як відбувається проектування з FPGA?

ЛАБОРАТОРНА РОБОТА 2. Вивчення синтезу операторів умовного і вибіркового призначення сигналів

Мета роботи

Освоїти методику використання операторів умовного і вибіркового призначення сигналів на прикладі побудови однорозрядного арифметико-логічного пристрою (АЛП).

2.1 Методичні вказівки з підготовки до виконання лабораторної роботи

У ході виконання лабораторної роботи необхідно синтезувати однорозрядний АЛП, використовуючи під час опису його VHDL-моделі оператори умовного чи вибіркового призначення сигналів. Функціональна схема пристрою наведена на рисунку 2.1. Майте на увазі, що сигнал OPCODE подано двома розрядами, але в мультиплексора задіяні лише три входи [1].

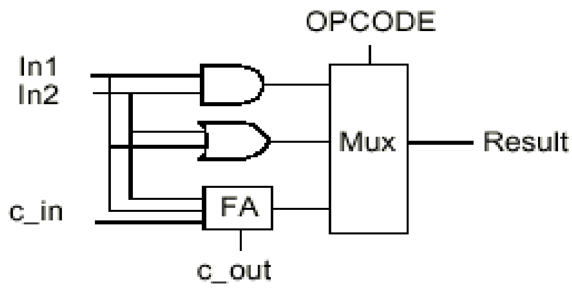


Рисунок 2.1 – Повний однорозрядний арифметико-логічний пристрій (АЛП) - суматор (FA – full adder) на три операції

2.1.1 Оператор умовного призначення сигналу

Модель, подана у лістингу 2.1, ілюструє використання оператора умовного призначення сигналу для опису мультиплексора 4-в-1 [4].

Лістинг 2.1 – VHDL-модель мультіплексора 4-в-1 для однорозрядних сигналів, яка може бути синтезована

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
port (in0, in1, in2, in3: in std_logic;
     s0, s1: in std_logic;
     z: out std_logic);
end entity mux4;
architecture behavioral of mux4 is
begin
z <=  in0 when s0 = '0' and s1 = '0' else
      in1 when s0 = '0' and s1 = '1' else
      in2 when s0 = '1' and s1 = '0' else
      in3 when s0 = '1' and s1 = '1' else
      '0';
end architecture behavioral;
```

У цій моделі значення вихідного сигналу Z визначено для всіх можливих комбінацій вхідних сигналів. Очевидно, що значення вихідного сигналу залежить тільки від значень цих вхідних сигналів. Тому система синтезу перетворить цей VHDL-код у комбінаційну схему.

Оператори умовного призначення сигналів, як правило, описують 2 типи структур: мультиплексори (комутатори) та шифратори/дешифратори. У мультиплексорах вихідний сигнал одержує значення одного з вхідних сигналів залежно від значень керуючих сигналів. Перетворювачі кодів (тобто шифратори чи дешифратори) генерують вихідне значення сигналу на підставі вхідних сигналів. Варто помітити, що мультиплексори синтезуються тільки у випадку, якщо умовні вирази в усіх альтернативах оператора (варіантах значень умовного виразу) є взаємовиключними. Інакше в схемі елементи будуть розташовуватися послідовно, відповідно до черговості умов. Порядок розташування альтернатив при цьому буде впливати на синтезовану схему. Оскільки умовні вирази в цьому операторі призначення сигналів розташовуються послідовно, то в кожній наступній альтернативі пріоритет буде нижчим, ніж у попередній. Тому синтезована з таких описів логіка називається *пріоритетною*.

При порівняно невеликій кількості вхідних сигналів складно помітити вплив пріоритетної логіки на синтезовану схему. Але не можна забувати про згадані властивості цього оператора, тому що у великих схемах фрагменти з пріоритетною логікою можуть призвести до появи довгих шляхів проходження сигналу, і відповідно збільшення затримок.

У схемній реалізації оператора умовного призначення сигналу вибір варіанта буде залежати від виконання чи невиконання відповідної послідовності булевих умов [5].

Ще одна важлива деталь: якщо в процесі синтезу зустрічаються операції порівняння, у яких беруть участь значення "не має значення" наприклад, “00” **when** datain=“-1” **and** valid=‘1’ **else**, то результатом завжди буде НЕПРАВДА (FALSE). Іншими словами, такі альтернативи на схемі будуть відсутні (хоча під час моделювання такі умови будуть перевірені).

2.1.2 Оператори вибіркового призначення сигналів

В операторах вибіркового призначення сигналів відсутній пріоритетний порядок аналізу альтернатив з умовними виразами. Розглянемо опис шифратора 4-в-2, поданий у лістингу 2.2.

Лістинг 2.2 – VHDL-модель шифратора 4-в-2

```
library IEEE;
use IEEE.std_logic_1164.all;
entity encoder is
port (datain: in std_logic_vector(3 downto 0);
     result: out std_logic_vector(1 downto 0));
end entity encoder;
architecture behavioral of encoder is
begin
with datain select
result <="00"  when"0001" ,
        "01"  when"0010" ,
        "10"  when"0100" ,
        "11"  when"1000" ,
        "XX"  when others;
end architecture behavioral;
```

Під час опису шифратора звичайно передбачається, що в кожен момент часу на одному з входів значення сигналу буде дорівнювати '1', а вихідний вектор являє собою дворозрядний двійковий номер цього входу. На відміну від цього, наприклад, у пріоритетних шифраторах у кожен момент часу відразу кілька входів можуть мати значення '1', і першим буде перевірений вхід з найвищим пріоритетом (наприклад, найстарший розряд вхідного вектора) [4, 5].

Вираз "XX" **when others** говорить про те, що при будь-якому вхідному векторі, вихідному сигналі завжди виконується призначення, тобто буде синтезована комбінаційна логіка без елементів пам'яті.

2.2 Порядок виконання роботи

Крок 1. Опишіть VHDL-модель однорозрядного АЛП. Не вказуйте ніяких затримок виконання операторів. Опишіть вихід мультиплексора, використовуючи оператори умовного чи вибіркового призначення сигналу. Якщо використовувати оператор вибіркового призначення сигналу, то в ньому обов'язково треба ставити вираз **when others** для врахування того факту, що мультиплексор має всього три входи замість чотирьох.

Крок 2. Згідно із завданням викладача виберіть тип мікросхеми для фізичної реалізації АЛП і визначте настроювання системи синтезу за швидкодією та займаною площею на кристалі. Синтезуйте проект *alu.vhd*.

Крок 3. Промоделюйте схему, яка описана у файлі alu.vhd, у середовищі САПР Active-HDL, для цього запустіть його за допомогою дій, указаних на рисунку 2.2. Потім у наступному вікні оберіть пункт Add existing resource file, далі пропишіть місцезнаходження файла alu.vhd. і вже у вікні (рисунок 2.3) вкажіть назву вашої моделі.

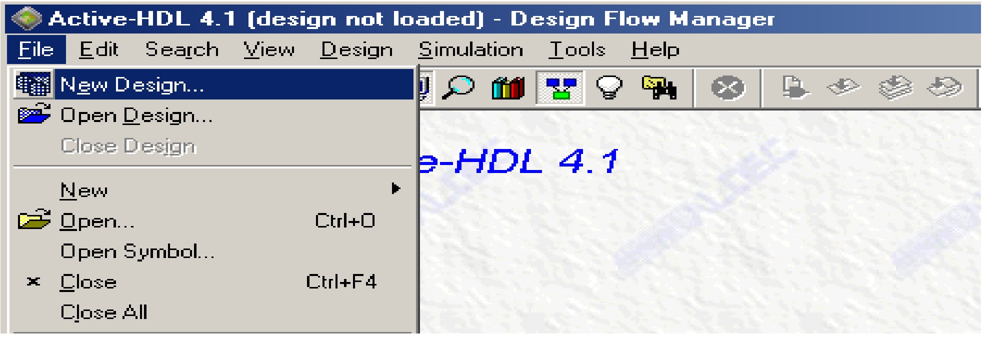


Рисунок 2.2 – Підключення файла alu.vhd

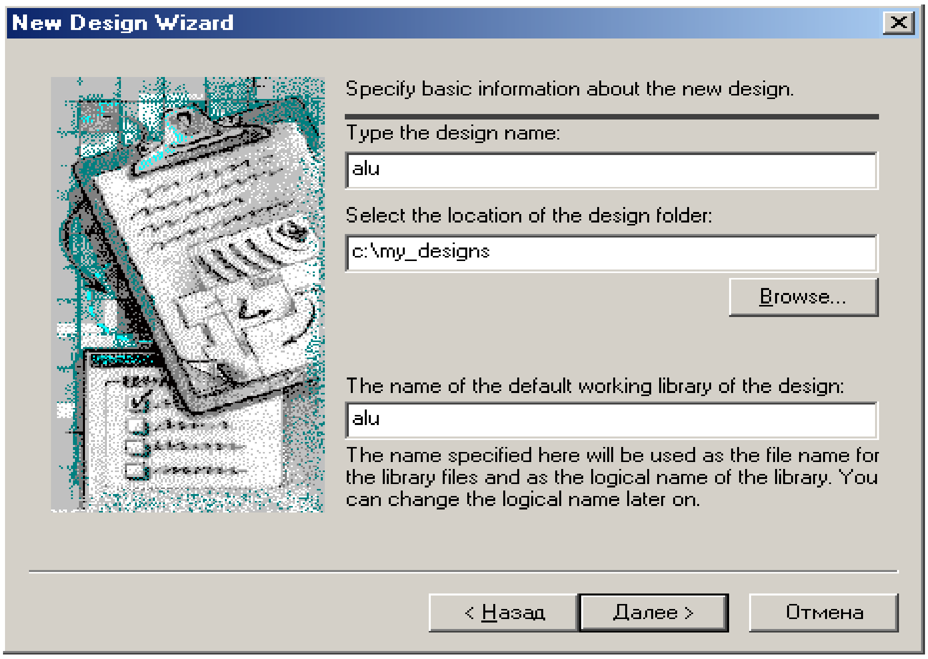


Рисунок 2.3 – Опис моделі у середовищі Active-HDL

У підсумку ви побачите вікно Design Browser, де виконайте процедури, зазначені на рисунках 2.4 та 2.5. Потім виділіть усі сигнали, додайте їх у Waveform (рисунок 2.6) і промоделюйте обрану схему.

Задайте часові діаграми (значення у часі) для вхідних сигналів і спостерігайте:

1) вхідні сигнали;
2) внутрішні сигнали;
3) вихід Result.

Крок 4. Визначте затримку від вхідних сигналів до вихідного.

Крок 5. Промоделюйте файл проекту alu.vhd і порівняйте з попередніми результатами моделювання. Чи є які-небудь розходження в часових параметрах? Чим ви можете пояснити такі розходження в часових діаграмах?

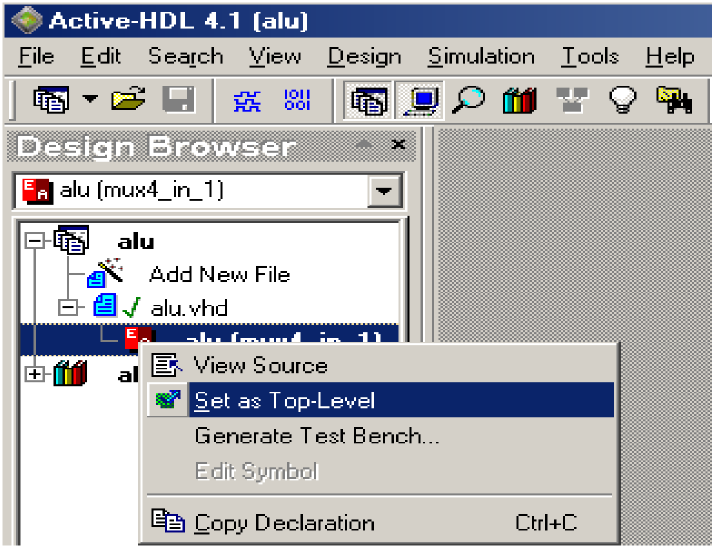


Рисунок 2.4 – Вікно Design Browser

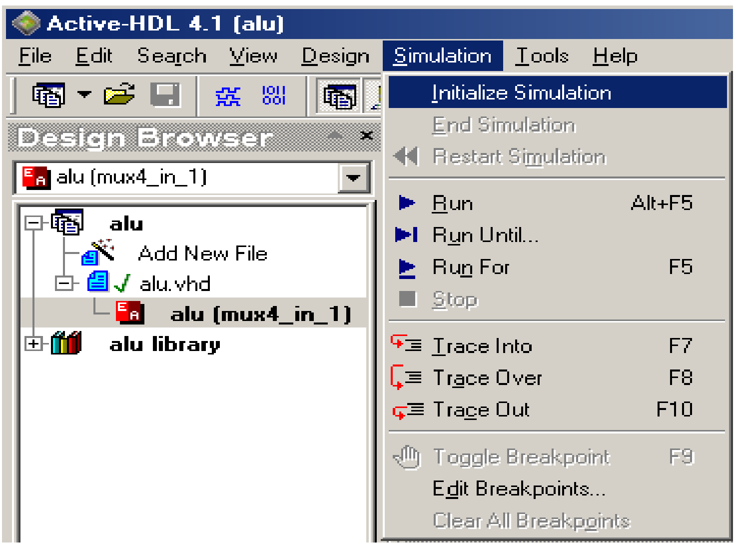


Рисунок 2.5 – Ініціалізація моделювання

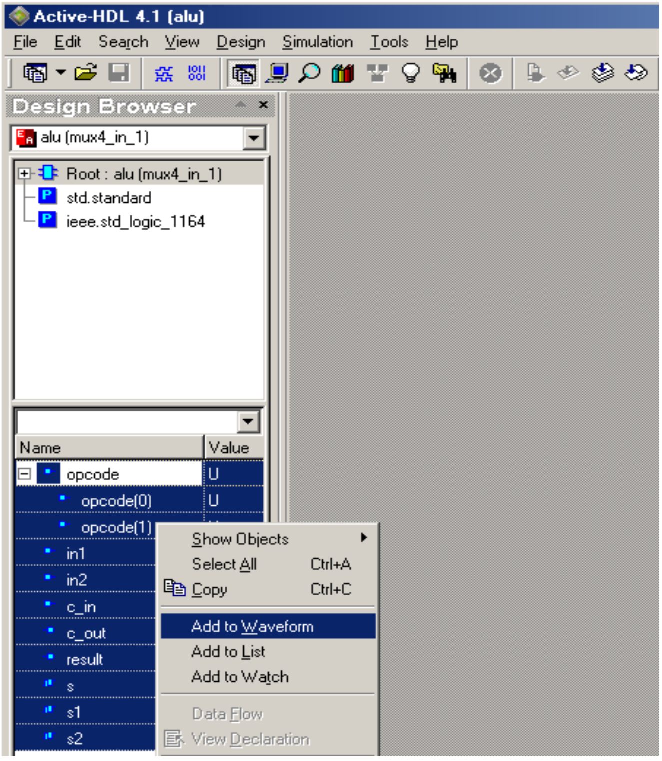


Рисунок 2.6 – Створення вікна Waveform Editor

2.3 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити:

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- розроблені Verilog-моделі АЛП з обраним типом оператора призначення;
- код testbench для першого варіанта;
- результати синтезу та тестування всіх пристроїв;
- порівняння результатів синтезу для різних типів кристалів (з додаванням графічних результатів роботи системи синтезу Synplify);
- результати моделювання (Waveform) для двох варіантів моделювання;
- висновки до роботи.

Контрольні запитання

1 Пояснити модель булевого програмування.

2 Пояснити принцип оптимальності в задачі динамічного програмування.

3 Пояснити зв'язок задачі оптимального управління з варіаційним численням.

4 Пояснити зв'язок задач управління із задачами диференціальних ігор.

5 Що таке оптимальна стратегія у диференціальній грі?

ЛАБОРАТОРНА РОБОТА 3. Вивчення принципів синтезу операторів IF-THEN--ELSIF та CASE

Мета роботи

Усвідомити на практиці різницю в синтезі конструкцій з використанням операторів **if-then-elsif** і **case.**

3.1 Методичні вказівки з підготовки до виконання лабораторної роботи

3.1.1 Оператор if-then-elsif

Оператор **if** використовується в процесах для умовного виконання блоку послідовних виразів. У цю структуру може входити необов'язковий компонент **else**. Оператор може також не включати жодної альтернативи **elsif**. У цьому випадку всі булеві значення умовних виразів будуть перевірятися послідовно доти, поки не зустрінеться умова, що виконується. [4]. Розглянемо модель, подану у лістингу 3.1.

Лістинг 3.1 – Приклад VHDL-моделі з умовним оператором призначення у процесі

```
library IEEE;
use IEEE.std_logic_1164.all;
entity inf_latch is
port (sel : in std_logic;
     x, y, z, p: in std_logic;
     w: out std_logic );
```

```
end entity inf_latch;
architecture behavior of inf_latch is
begin
process (x, y, z, p, sel) is
variable s1, s2: std_logic;
begin
  if (sel = '1')then
            s1 := x and z;
            s2 := p xor y;
            w <= s2 and s1;
  end if;
end process;
end architecture behavior;
```

Якщо sel=1, то значення сигналу w буде обчислено відповідно до коду програми. Очевидно, що значення w при sel=0, у цьому випадку буде реалізований асинхронний тригер (рисунок 3.1).

Комбінаційна логіка, що передує елементу пам'яті (асинхронному DC-тригеру), являє собою реалізацію трьох виразів, які становлять тіло гілки **then**, що обчислюють нове значення сигналу w (як синхросигнал використовується сигнал sel). При написанні вкладених конструкцій неявно встановлюється пріоритетний порядок операторів призначення. Отже, при синтезі цей порядок відіб'ється на отриманій схемі.

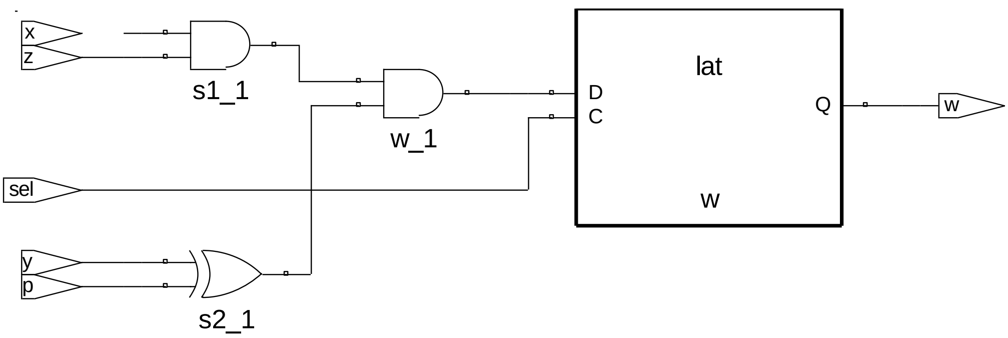


Рисунок 3.1 – Синтез оператора **if-then-elsif**

Ще раз повернемося до синтезу елементів пам'яті. Цього можна уникнути, якщо виконувати одне просте правило за будьякої умови виконання процесу. Іншими словами, всі вихідні сигнали повинні бути визначені в кожній альтернативі **then** і **else**. Можливий також інший варіант – призначати всі виходи до умовного оператора [4, 6].

3.1.2 Оператор case

Оператор **case** еквівалентний набору вкладених конструкцій **if-then-else**. В операторі **case** тільки одна альтернатива виразу може бути істинною. При цьому набір альтернатив повинен покривати всі можливі варіанти значень умови вибору. Альтернатива **when others** полегшує це завдання. Вибір одного з декількох можливих варіантів приводить до синтезу позиційного мультиплексора і логіки, що використовує значення умови вибору для формування керуючих сигналів.

Якщо вихідний сигнал одержує значення в кожній альтернативі цього оператора, то генерується комбінаційна логіка. Інакше буде синтезований тригер, що зберігає попереднє значення виходу. Розглянемо приклад у лістингу 3.2.

Лістинг 3.2 – Приклад використання оператора **case**

```
library IEEE;
use IEEE.std_logic_1164.all;
entity case_ex is
port ( sel : in integer range 0 to 2;
       x, y: in std_logic;
       res : out std_logic );
end entity case_ex;
architecture behavior of case_ex is
 begin
  process (x,y,sel) is
   begin
   case sel is
    when 0 => res <= x and y;
    when 1 => res <= y xor x;
    when others => res <= x nor y;
   end case;
```

end process;
end architecture behavior;

Для кожної альтернативи **case** буде синтезована логіка, що обчислює значення сигналу **res** (рисунок 3.2**)**. Потім генерується логіка, що формує дозвільний сигнал **sel** за допомогою групи елементів ***AND***. Будь-який оператор **case** можна перетворити в умовний оператор з використанням конструкції **if-then-elsif**. Однак використання її приведе до синтезу пріоритетної логіки. У випадку використання **case** такого не відбувається. Отже, його використання має більше переваг [5, 6].

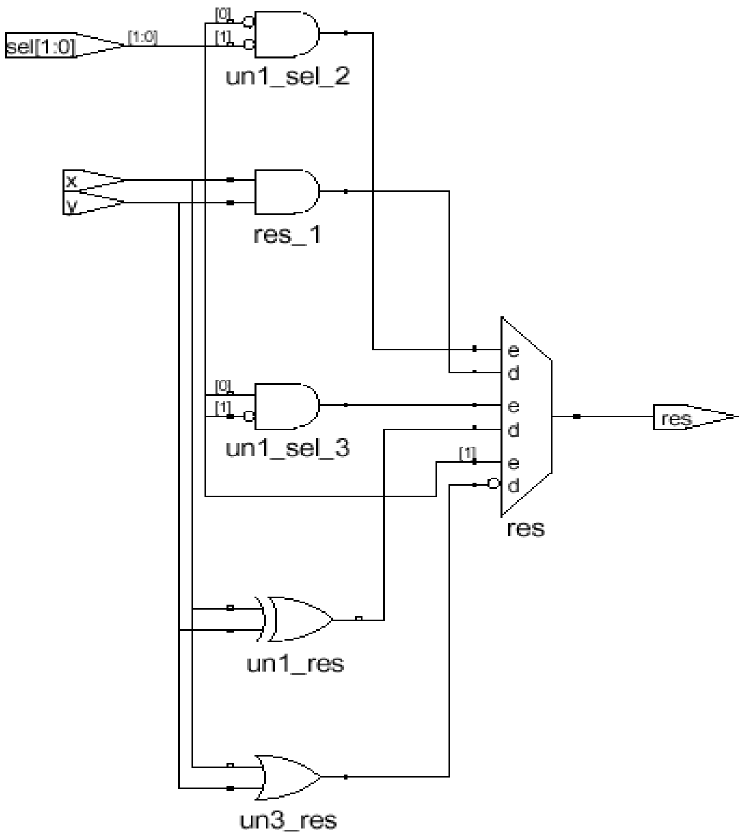


Рисунок 3.2 – Результат синтезу моделі з оператором **case**

3.2 Порядок виконання роботи

Крок 1. Створіть модель пріоритетного перетворювача кодів 4-в-2. Модель складається з одного процесу, містить 4 однорозрядних входи, 2-розрядний вихід і дозвільний сигнал.

Пріоритетність обрана в такий спосіб: 3, 2, 0, 1. Для дозвільного сигналу використовуйте оператор **if**.

Крок 2. Синтезуйте пристрій.

Крок 3. Розгляньте схемну реалізацію (RTL–view) обох моделей:

а) у чому відмінності використання вкладених конструкцій **if-then-elsif** від оператора **case**;

б) у чому причина появи розходжень при апаратній реалізації логіки? Як можна зв'язати ці відмінності з процесом синтезу;

в) якій мовній конструкції в моделі варто віддати перевагу.

3.3 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити;

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- розроблені Verilog-моделі з конструкціями **if-then-elsif** та **case**;
- код testbench для першого варіанта;
- результати тестування всіх пристроїв;
- результати автоматизованого синтезу проекту у вигляді графічного відображення схем тригерів і порівняльний аналіз;
- висновки до роботи.

Контрольні запитання

1 Що таке задача оптимізації та індивідуальна задача оптимізації?

2 Як оцінюють складність алгоритмів розв’язання оптимізаційних задач?

3 Як формалізуються задачі про найкоротший шлях?

4 Як формалізуються задачі про незалежну множину?

5 Як формалізуються задачі про фарбування графів?

6 Як формалізуються задачі про покриття?

7 Як формалізується задача комівояжера?

ЛАБОРАТОРНА РОБОТА 4. Синтез схем з елементами пам'яті

Мета роботи

Освоїти синтез синхронних послідовних схем за допомогою операторів **wait** і **if-then-end if** та усвідомити розходження в їх апаратній реалізації.

4.1 Методичні вказівки з підготовки до виконання лабораторної роботи

Оскільки при виконанні цієї лабораторної роботи потрібно використовувати оператор **wait**, то нагадаємо деякі моменти, пов'язані з його застосуванням.

Включення оператора **wait** у VHDL-модель мотивується необхідністю побудови моделі схеми, у якій події відбуваються у визначені моменти часу, обумовлені фронтом синхроімпульсу. Це відбивається таким типом оператора **wait** [7]:

wait until умова,

де умова – це булевий вираз, який правильний, якщо виникає фронт синхроімпульсу, наприклад:

wait until (clk**’event** and clk = ‘1’);
wait until (rising_edge (clk);
wait until (falling_edge (clk).

У результаті оператор **wait** синтезується в синхронну логіку. Деякі системи синтезу накладають обмеження на типи виразів, що можуть містити оператор **wait**.

В одному процесі дозволено використання тільки одного оператора **wait until** <умова>, і він має бути першим у процесі. Отже, якщо в операторі використовується умова спрацювання по фронту, то тригер буде синтезуватися для кожного сигналу процесу. Це може не відповідати завданню, що поставлене, якщо потрібно синтезувати елементи пам'яті тільки для деяких сигналів. У таких випадках краще використовувати оператор **if-then-end if** з умовою спрацьовування по фронту. Це приведе до синтезу тригерів для сигналів, що містяться в альтернативі **then** оператора **if**, і не вплине на інші сигнали процесу. Отже, якщо необхідно синтезувати блоки процесу в комбінаційну логіку, використання операторів **if-then-elsif** чи **if-then-end if** забезпечує необхідну гнучкість [7].

4.2 Порядок виконання роботи

Необхідно синтезувати схему, зображену на рисунку 4.1, яка реалізує такі обчислення:

Aout = In1 **and** In2 **xor** In3;
MyState = In1 **or** In2 **and** MyState;

Елементи пам'яті в синтезованій схемі керуються прямим фронтом синхросигналу. Обидва сигнали Aout і MyState є вихідними сигналами, і для сигналу MyState синтезується елемент пам'яті.

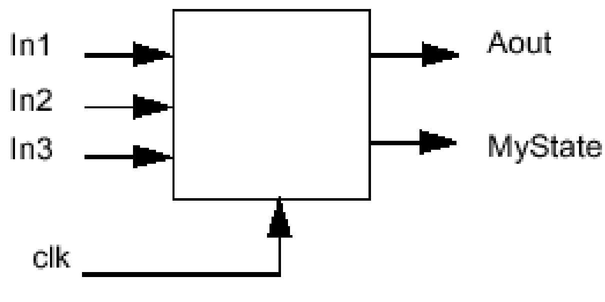


Рисунок 4.1 – Графічний опис інтерфейсу синтезованої схеми

Крок 1. Побудуйте VHDL-модель для заданих обчислень з такими характеристиками:

а) усі сигнали є однорозрядними і належать типу **std_logic;**

б) використовується один процес з оператором **wait**, що чутливий до переднього фронту синхроімпульсу.

Крок 2. Виконайте синтез проекту і розгляньте синтезовану схему.

Крок 3. Виконайте моделювання проекту. Встановіть максимальний період сигналу **clk**.

Крок 4. Виконайте модифікацію проекту, замінивши оператор **wait** на конструкцію з оператором **if-then-end if.**

Крок 5. Виконайте повторний синтез проекту і розгляньте отриману схему. Чи є розходження з реалізацією першої моделі? Яка кількість запам'ятовувальних елементів?

Крок 6. Виконайте моделювання проекту. Встановіть максимальний період сигналу **clk.**

Крок 7. Порівняйте поводження сигналу **Aout** у синтезованих схемах.

4.3 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити;

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- коментарі;
- розроблені Verilog-моделі пристрою обробки інформації;
- код testbench для першого варіанта;
- результати синтезу;
- результати моделювання (Waveform) для двох варіантів моделювання;
- висновки до роботи.

Контрольні запитання

1 Як пов’язана структура алгоритму з архітектурою паралельною обчислювальною системою?

2 Які основні завдання вирішуються при створенні паралельної обчислювальної системи?

3 Що таке асимптотика і як її використовують для визначення складності алгоритмів?

4 Як класифікуються паралельні обчислювальні системи?

5 Які типи абстрактних машин виділяються у паралельних обчислювальних системах?

ЛАБОРАТОРНА РОБОТА 5. Синтез кінцевих автоматів

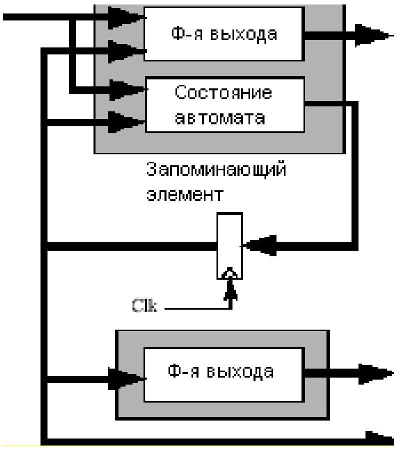
Мета роботи

Вивчити способи опису VHDL-моделей кінцевих автоматів на основі підмножини мови VHDL, яка синтезується.

5.1 Методичні вказівки з підготовки до виконання лабораторної роботи

Загальна модель автомата подана на рисунку 5.1.

Входи



Вихід автомата Мілі

Вихід автомата Мура

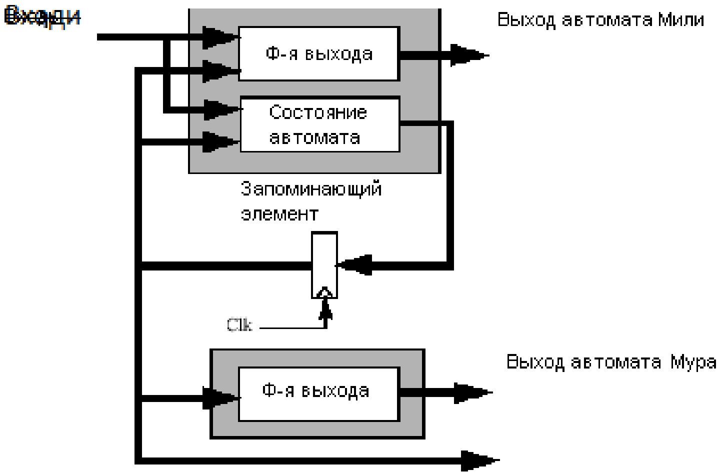


Рисунок 5.1 – Модель кінцевого автомата Мілі та Мура

Цей рисунок зображує модель автоматів Мілі та Мура. Значення виходів першого залежать від вхідних сигналів та поточного стану, а другого—тільки від поточного стану. Така модель достатньо загальна та придатна для відображення всіх кінцевих автоматів, які цікаві для вивчення. Одна із задач, що виникають при синтезі кінцевих автоматів, це кодування станів. Існують три основні методи кодування, які подані в таблиці 5.1.

Таблиця 5.1 – Типи кодування станів автомата

Стан	Послідовний код	Код Грея	Унітарне кодування
0	000	000	00000001
1	001	001	00000010
2	010	011	00000100
3	011	010	00001000
4	100	110	00010000
5	101	111	00100000
6	110	101	01000000
7	111	100	10000000

У цій таблиці перша колонка – це номер стану, останні – варіанти кодування різноманітними методами. Послідовне кодування та сусіднє кодування (код Грея) є найбільш оптимальними за кількістю тригерів. Однак це майже не відбивається на швидкості роботи та площині схеми на кристалі ПЛІС. Унітарне кодування використовує один тригер на кожний стан. Такий тип кодування дає найбільшу кількість тригерів, однак, перевага цього методу в тому, що його реалізація простіша, та при цьому кодування якогось певного стану відносно легко обчислити.

Мова VHDL дає змогу розробникам використовувати свої типи кодування станів. Наприклад, можна ввести новий тип даних:

type statetype **is** (state0, state1, state2, state3);

При цьому сигнали тепер можна описати як:

signal state, next_state : statetype;

Розробник повинен мати можливість присвоїти кожному стану унікальний код, наприклад, стан state0 має код 0001, стан state1 – код 1000 і т. д. Це можна використати за умови, що відома структура задавання автомата та вплив кодування на реалізацію. В іншому випадку необхідно відмовитись від явного задавання кодів станів. Система синтезу самостійно вибере спосіб кодування. Вона також оптимізує кодування станів, критерієм якого є швидкість та площа.

Розглянемо приклад реалізації кінцевого автомата, описаного графом переходів на рисунку 5.2. VHDL-модель цього автомата наведена в лістингу 5.1. Вона побудована на основі однопроцесного шаблона автомата Мілі. Результати синтезу наведені на рисунку 5.3.

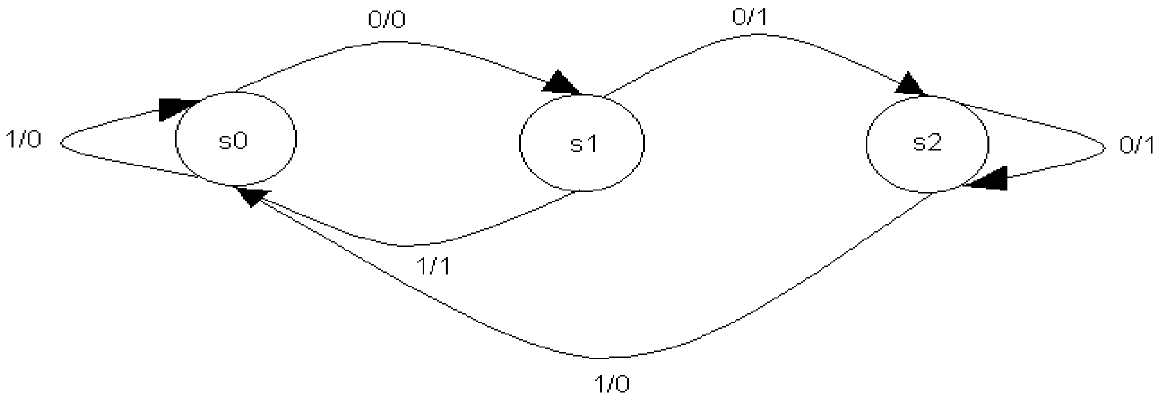


Рисунок 5.2 – Граф-схема автомата

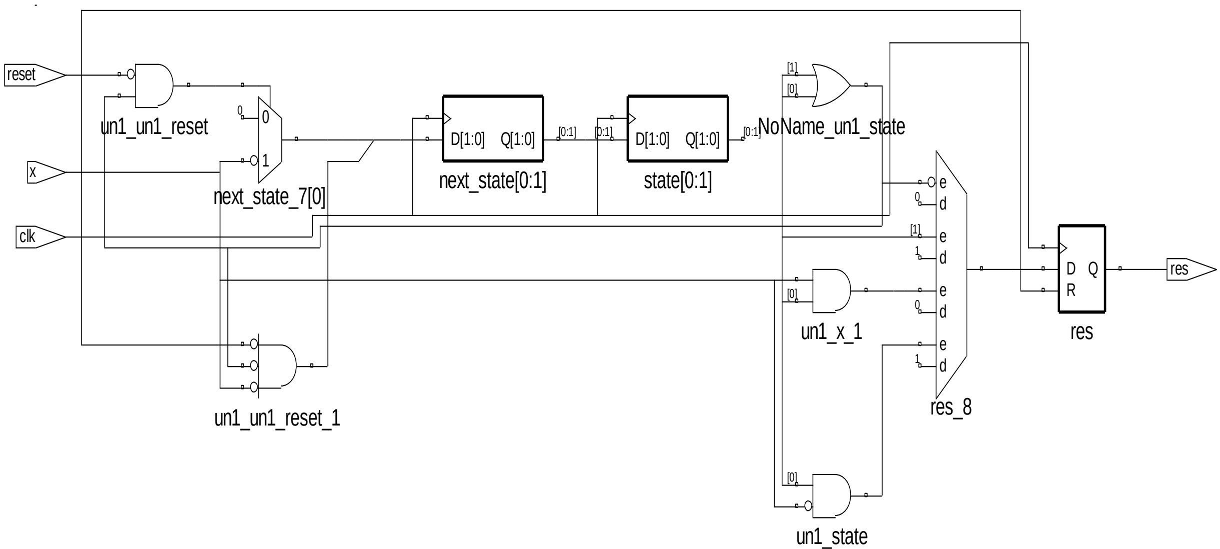


Рисунок 5.3 – Синтезований кінцевий автомат

Лістинг 5.1 – VHDL-модель керуючого автомата у вигляді однопроцесного шаблона

```
library IEEE;
use IEEE.std_logic_1164.all;
entity state_machine is
port(reset, clk, x : in std_logic;
    res : out std_logic);
end entity state_machine;
architecture behavioral of state_machine is
type statetype is (state0, state1, state2);
signal state, next_state : statetype ;
begin
process is
begin
wait until (rising_edge(clk));
state <= next_state;
if reset = '1' then
res <= '0';
next_state<=statetype'left;
else
case state is
        when state0 =>
                if x = '1' then
                        next_state <= state0;
                        res <= '0';
                else next_state <= state1;
                        res <= '0';
                end if;
        when state1 =>
                if x = '1' then
                        next_state <= state0;
                        res <= '1';
                else next_state <= state2;
                        res <= '1';
                end if;
        when state2 =>
                if x = '1' then
```

```
                    next_state <= state0;
                    res <= '0';
               else next_state <= state2;
                    res <= '1';
          end if;   end case;   end if;      end process;
end architecture behavioral;
```

Тригери синтезуються для сигналу **state** та для вихідного сигналу res, бо вони обидва призначаються в тіла процесу, який керується оператором w**ait until**. Для того, щоб уникнути синтезу вихідного тригера для сигналу res, слід винести його з тіла процесу, який керується оператором **wait until**. Змінимо модель автомата, який розглядається, та запишемо його у вигляді двопроцесного шаблона (лістинг 5.2). Результати синтезу наведені на рисунку 5.4.

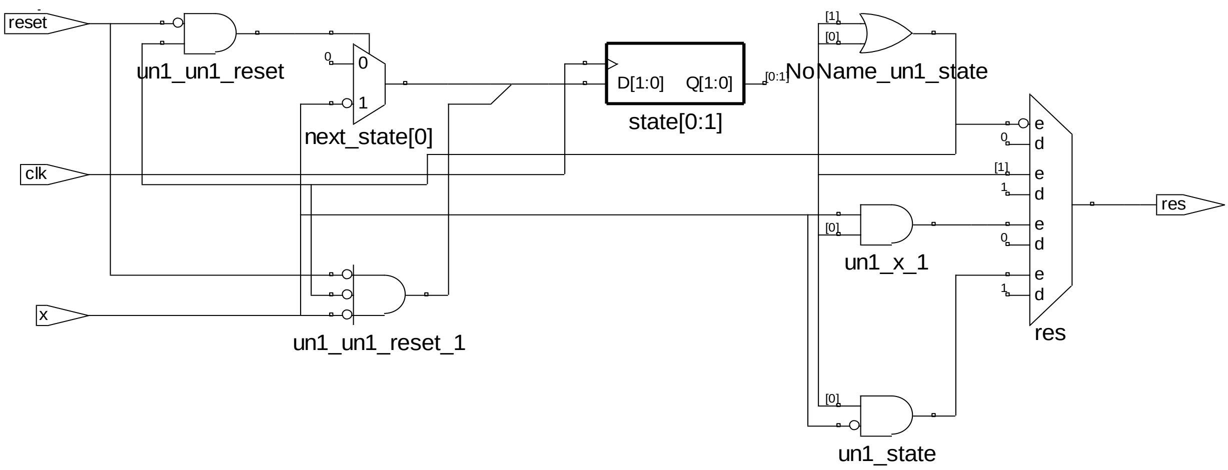


Рисунок 5.4 – Кінцевий автомат, синтезований за двопроцесною моделлю

Лістинг 5.2 – Двопроцесний шаблон керуючого автомата

```
library IEEE;
use IEEE.std_logic_1164.all;
entity state_machine is
port(reset, clk, x : in std_logic;
     res : out std_logic);
end entity state_machine;
architecture behavioral of state_machine is
```

```
type statetype is (state0, state1,state2);
signal state, next_state : statetype ;
begin
process (state, next_state,reset,x)is
begin
 if reset = '1' then
  res <= '0';
  next_state<=statetype'left;
 else
case state is
        when state0 =>
                if x = '1' then
                        next_state <= state0;
                        res <= '0';
                else next_state <= state1;
                        res <= '0';
                end if;
        when state1 =>
                if x = '1' then
                        next_state <= state0;
                        res <= '1';
                else next_state <= state2;
                         res <= '1';
                end if;
       when state2 =>
                if x = '1' then
                         next_state <= state0;
                         res <= '0';
                else next_state <= state2;
                         res <= '1';
            end if;
        end case;
 end if;
end process;
process is
begin
 wait until (rising_edge(clk));
  state <= next_state;
```

end process;
end architecture behavioral;

Як бачимо з рисунка, в цій моделі автомата тригери синтезуються тільки для сигналу *state*. Отже, використовуючи двопроцесну модель, можна запобігти синтезу зайвих тригерів у схемі. З цього можна зробити висновок: двопроцесний шаблон моделі автомата дає більш економну схемну реалізацію, ніж однопроцесний, бо він точніше відображає модель кінцевого автомата, яка подана на рисунку 5.2.

Ще одна задача: як запобігти встановленню елементів пам`яті в будь-який невизначений стан під час початкового запуску автомата. Існує дві можливості: одна – це у явному вигляді включати до проекту сигнал початкового скидання, як показано в усіх прикладах цієї роботи. Друга – використати спеціальні механізми, запропоновані виробниками САПР, які дають змогу упевнитися, що елементи пам`яті встановлюються у визначений стан за включенням.

5.2 Порядок виконання роботи

Необхідно виконати синтез схеми для заданого керуючого автомата. Цей автомат пристосовано для розпізнавання групи з 3 біт у вхідній двійковій послідовності. Поява зазначеної послідовності викликає встановлення виходу автомата в 1 і значення виходу залишається рівним 1 доти, доки не з'явиться сигнал скидання *reset*. Для представлення автомата використовуємо модель Мілі.

Крок 1. Опишіть VHDL-модель автомата, що розпізнає 3-бітову послідовність. Вид послідовності вибирається відповідно до вашого варіанта з таблиці 5.2. Граф-схема автомата наведена на рисунку 5.5.

Таблиця 5.2 – Варіанти видів розпізнаваної послідовності бітів

Варіант	Вид послідовності
1	000
2	001
3	010
4	011
5	100
6	101
7	110
8	111

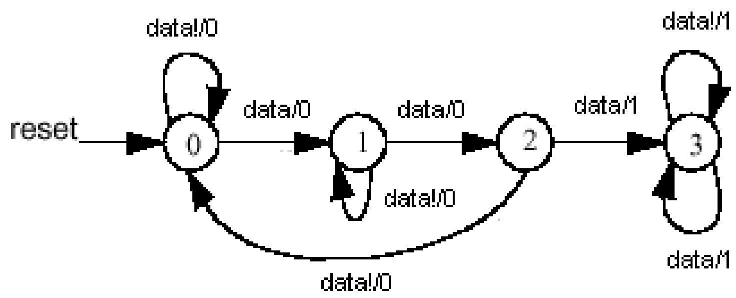


Рисунок 5.5 – Граф-схема моделі автомата

Символи Data і data! вказують на наявність чи на відсутність на вході автомата чергового біта послідовності, що розпізнається, відповідно. Для представлення станів використовуйте тип, що перелічується, statetype. Модель автомата слід подати у вигляді трипроцесного шаблона.

Процес 1. Процес, що обчислює наступний стан на основі поточного і значення вхідного сигналу. За імпульсом reset=1 автомат скидається в 0.

Процес 2. Вихідний процес, що визначає однорозрядний вихідний сигнал на основі поточного стану та значення одного розряду вхідної послідовності.

Процес 3. Процес, що оновлює стан автомата по передньому фронту синхросигналу.

Крок 2. Синтезуйте модель для заданого викладачем типу FPGA.

Крок 3. Розгляньте отриману схему. Спробуйте виділити компоненти відповідно до моделі на рисунку 5.2.

Крок 4. Промоделюйте проект. Задайте вхідні сигнали для моделювання в такий спосіб:

- синхросигнал матиме період 20 нс;
- початковий сигнал скидання буде мати тривалість 10 нс;
- згенеруйте псевдовипадкову однобітову послідовність, де існує набір, який відповідає вашому варіанту з таблиці 5.2.

Перевірте правильність виконання переходів автоматом. Яку максимальну частоту синхросигналу забезпечує обраний тип FPGA?

Крок 5. Змініть проект таким чином, щоб він містив тільки один процес.

Крок 6. Синтезуйте проект знову і проаналізуйте отриману схему. Які з'явилися розходження ?

5.3 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити:

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- коментарі;
- трипроцесну VHDL-модель автомата;
- результати синтезу;
- результати моделювання (Waveform) ;
- однопроцесну VHDL-модель автомата та результати синтезу і моделювання;
- порівняльний аналіз;
- висновки до роботи;

Контрольні запитання

1 Як описуються структурні, поведінкові та DataFlow моделі в Verilog?

2 Як описуються затримки в Verolog?

3. Для чого використовується директива 'timescale?

4.Як працює блок initial?

5 Для чого використовується блок always?

ЛАБОРАТОРНА РОБОТА 6. Розробка поведінкової та Data-Flow моделей

Мета роботи

Отримати навички з побудови VHDL-опису моделей поведінкової та Data-Flow.

6.1 Методичні вказівки з підготовки до виконання лабораторної роботи

Згідно з варіантом завдання записати Verilog-модель пристрою рівня передачі даних.

Завдання 1. Згідно з варіантом завдання записати Verilog-модель пристрою рівня передачі даних.

```
// Приклад моделі рівня передачі даних (data flow) завдання

module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
// декларація портів
output out;
input i0, i1, i2, i3;
input s1, s0;
assign out =
(~s1&~s0&i0)|(~s1&s0&i1)|(s1&~s0&i2)|(s1&s0&i3);
endmodule
```

Для верифікації отриманої моделі побудувати testbench, який перебирав би всі можливі вхідні набори.

```
// Приклад тестового модуля
module stimulus_simple;
  // Декларація внутрішніх змінних
  reg I0, I1, I2, I3, S1, S0;
  wire OUT;
    // Реалізація копії модуля
  mux4_to_1 MOD1(OUT, I0, I1, I2, I3, S1, S0);
    // Формування тестових послідовностей
  initial
    begin
```

```
        I0 = 1'b0; I1 = 1'b0; I2 = 1'b0; I3 = 1'b0;
        S1 = 1'b0; S0 = 1'b0;
        #10 I0 = 1'b1;     // I0=1 в 10 ns
        . . .
        #10 $finish;     // Моделювання закінчиться в 30 ns
      end
endmodule.
```

Завдання 2. Згідно з варіантом розробити поведінкову модель лічильника.

```
// Приклад моделі лічильника
module counter(COUNTER, CLK, RESET);
  parameter SIZE = 'd4;
  output [SIZE-1:0] COUNTER;
  input CLK, RESET;
  reg [SIZE-1:0] COUNTER;
always @(posedge CLK or posedge RESET) begin
  if (RESET) COUNTER = 'd0;
  else COUNTER = COUNTER + 'd1;
end endmodule
```

Завдання 3. Побудувати модель зсувного регістра

```
// Приклад Verilog-моделі 4-бітного без знакового лічильника з синхронним скиданням
module counter (input C, R, input [3:0] D, output reg [3:0] Q);
always @(posedge C)
        if (R) Q = 4'b0;
        else Q = Q + 1'b1;
endmodule
```

6.2 Опис лабораторної установки

У лабораторній роботі використовуються пакет Xilinx ISE Project Navigator і плата Xilinx Spartan-3E Starter Kit. Для моделювання і тестування розроблених пристроїв використовується пакет Active-HDL. Методична література і додаткове програмне забезпечення розміщені в http://kart.edu.ua/documentu-zvo-ua

Інструкція з використання програмних засобів міститься в тому ж каталозі.

6.3 Порядок виконання роботи

6.3.1 Ознайомтеся з літературою до лабораторної роботи. Перед виконанням роботи дайте відповіді на контрольні запитання.

6.3.2 Індивідуально виконайте запропонований варіант завдання.

6.3.3. Згідно з варіантом завдання (завдання а і б) запишіть Verilog-модель пристрою рівня передачі даних.

6.3.4 Напишіть VHDL-код поведінкової та Data-Flow моделей, обраних відповідно до варіанта з таблиці. Виконайте для нього пункти 5.3.5 – 5.3.10.

6.3.5 Створіть проект у середовищі Project Navigator, додайте в проект VHDL- опис тригера.

6.3.6 Виконайте синтез проекту.

6.3.7 Виконайте імплементацію проекту, створивши попередньо UCF файл.

6.3.8 Виконайте компіляцію і моделювання.

6.4 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити:

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- розроблені Verilog-моделі;
- код testbench для першого варіанта;
- результати тестування сіх пристроїв;
- результати автоматизованого синтезу проекту у вигляді графічного відображення схем тригерів;
- висновки до роботи.

Контрольні запитання

1 Як описуються структурні, поведінкові та DataFlow моделі в Verilog?

2 Як описуються затримки в Verolog?
3 Для чого використовується директива 'timescale?
4 Як працює блок initial?
5 Для чого використовується блок always?

ЛАБОРАТОРНА РОБОТА 7. Синтез мікропрограмного автомата Мура в системі автоматизації проектування Xilinx ISE

Мета роботи

Ознайомитися з автоматизованим синтезом керуючих автоматів на прикладі автомата Мура і порівняти його з канонічним структурним синтезом мікропрограмних автоматів.

7.1 Методичні вказівки з підготовки до виконання лабораторної роботи

Плата FPGA XC3S500E-5fg320 використовується як макет, який ілюструє процеси, що відбуваються в схемі. Для подачі вхідних впливів використовуються кнопки і перемикачі, для спостереження вихідних сигналів – світлодіоди. Для генерації синхросигналу використовується вбудований генератор з частотою 50 МГц (періодом 20 нс) (рисунок 7.1). Також плата дає можливість підключати зовнішні генератори частоти. Для цього на платі є два рознімання (рисунок 7.1) [8, 9].

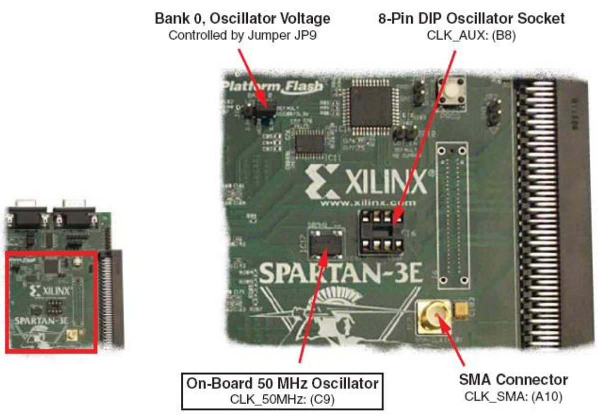


Рисунок 7.1 – Вбудований генератор синхроімпульсів і рознімання для підключення зовнішніх генераторів

Якщо при реалізації автомата інформаційні сигнали подавати з перемикачів, а синхросигнал отримувати з вбудованого генератора, спостерігати покрокову роботу автомата буде неможливо через високу швидкодію схеми, що визначається частотою синхросигналу [9, 10].

Якщо синхросигнал подавати з перемикача або кнопки, ми теж не зможемо спостерігати покрокову роботу автомата, оскільки кнопки і перемикач дають на вхід схеми не чистий сигнал, а сигнал з брязкотом (точніше сигнал, що викликається брязкотом механічних контактів), тобто з багаторазовою зміною сигналу протягом короткого часу (рисунок 7.2). Ця багаторазова зміна сигналу утворює безліч фронтів, за якими автомат зможе переключитися неодноразово.

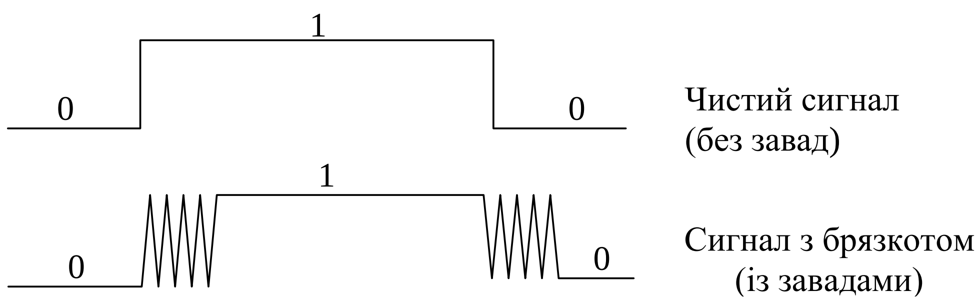


Рисунок 7.2 – Брязкіт механічних перемикачів

Для виходу з положення необхідно використовувати стробувальний дозвільний сигнал синхронізації. Його потрібно подавати з перемикача або кнопки, а синхросигнал підключати з генератора. Оскільки період синхросигналу значно менший від часу натискання кнопки або перемикача, то за одне натискання (подачу дозвільного сигналу синхронізації) або перемикання автомат може перейти в t + 1 стан багато разів, і ми не зможемо спостерігати його покрокову роботу. Тому потрібна схема, що формує на виході одиничний імпульс протягом одного синхротакту для вирішення синхронізації (внутрішній стрибок). Ця схема повинна також ураховувати, що стрибок зовнішнього сигналу, що подається з перемикача або кнопки, теж має брязкіт. Таким чином, необхідна протибрязкітлива схема для формування одиничного імпульсу (внутрішнього стрибка), що подається на автомат після натискання кнопки або перемикача (зовнішнього стрибка). Нижче на рисунку 7.3 подана тимчасова діаграма, що ілюструє цю ідею. Тут AD – протибрязкітлива схема, En – зовнішній стрибок, EnFSM – внутрішній стрибок [10, 11].

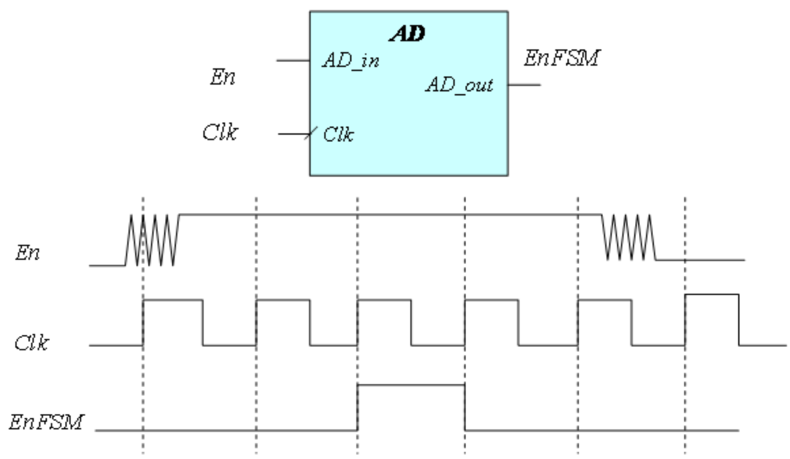


Рисунок 7.3 – Формування внутрішнього стрибка

Структура протибрязкітливої схеми і часові діаграми, що пояснюють її роботу, подані на рисунку 7.4. Часові діаграми наведені з урахуванням затримок спрацьовування елементів.

Період синхроімпульсу повинен бути більшим від часу брязкоту. Якщо передній фронт синхроімпульсу Clk надійде в момент брязкоту, значення 0 або 1 буде збережено в тригері в момент t1. При наявності 0 одиниця буде отримана при наступному активному фронті синхроімпульсу (t2). Тому сигнал S1 є беззавадовою і синхронізованою версією сигналу AD_in. Проте можливість помилки існує, якщо зміна сигналу AD_in буде близька до активного фронту синхроімпульсу, що призведе до порушення часу установки і зберігання.

У такому випадку вихід тригера може почати генерувати або видати неправильний результат. Хоча така ситуація буває рідко, краще виключити її, додавши ще один тригер S2, який зберігає затримане на один такт значення вхідного сигналу і дає змогу виявити його зміну. Сигнал AD_out протягом одного синхротакту дорівнює 1, якщо на вході AD_in відбулася зміна значення з 0 на 1. VHDL-модель протибрязкітливої схеми подана в лістингу 7.1 [11, 12]. На рисунку 7.5 наведено результат моделювання протибрязкітливої схеми.

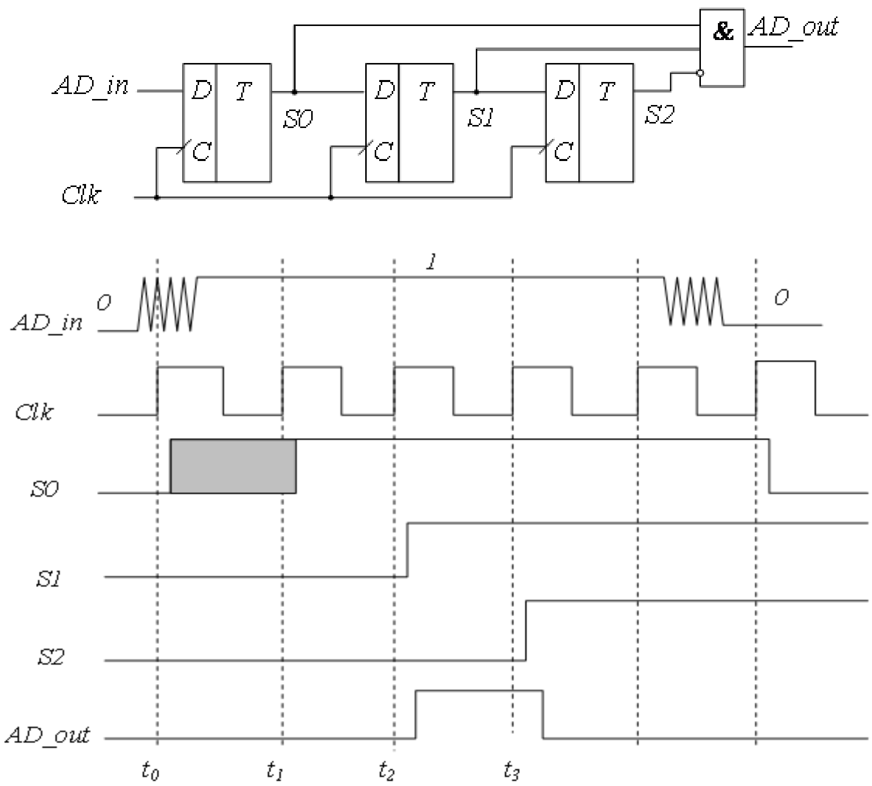


Рисунок 7.4 – Протибрязкітлива схема і її часові діаграми з урахуванням затримок

Лістинг 7.1 – VHDL-модель протибрязкітливої схеми

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AD is
   Port (AD_in, clk: in  STD_LOGIC;
                        AD_out: out STD_LOGIC);
end AD;
architecture AD of AD is
signal S0, S1, S2: STD_LOGIC;
begin
    PR1: process (clk)
    begin
        if clk = '1' and clk'event then
                S0 <= AD_in;
                S1 <= S0;
                S2 <= S1;
    end if;
```

end process;

AD_out <=S0 and S1 and not S2; -- формування керуючого імпульсу;

end AD;

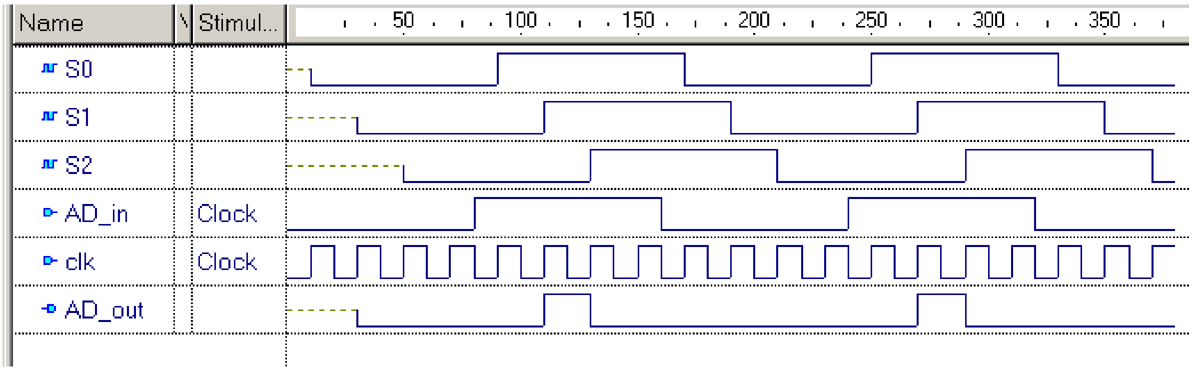


Рисунок 7.5 – Результат моделювання протибрязкітливої схеми

Наявності протибрязкітливої схеми не буде достатньо для моделювання роботи автомата на макеті. Це пов'язано з тим, що період синхроімпульсу набагато коротший від брязкоту перемикача. Тому необхідно збільшити період синхроімпульсу (зменшити частоту синхроімпульсу) до величини, яка відповідає тривалості брязкоту.

Перемикачі на платі Spartan-3E Starter Kit мають тривалість брязкоту t_d=2 мс=$2 \cdot 10^{-3}$ с, а частота синхроімпульсу, що генерується вбудованим у плату генератором, становить h_{clk}=50 Мгц=$50 \cdot 10^{6}$ Гц. Період синхроімпульсу визначається як $1/h_{clk} = 1/(50 \cdot 10^{6}) = 0{,}02 \cdot 10^{-6}$ с=$20 \cdot 10^{-9}$ с= 20 нс. Щоб визначити, у скільки разів необхідно збільшити період clk, необхідно розділити t_d / t_{clk}=$(2 \cdot 10^{-3})/(20 \cdot 10^{-9})=10^{5}$, тобто в 100000 разів або більше. Для реалізації збільшення періоду (зменшення частоти) *clk* необхідно скористатися дільником частоти, виконаним на лічильнику (рисунок 7.6).

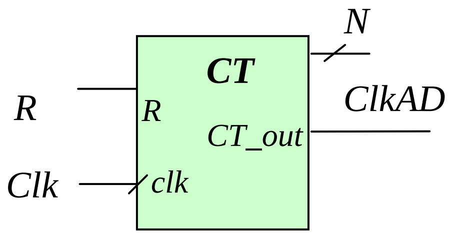


Рисунок 7.6 – Дільник частоти на лічильнику

Визначимо розрядність лічильника N, якщо коефіцієнт перерахунку лічильника повинен бути більшим ніж 100000. 2^{17}=131072, 2^{16}=65536, отже, N=17. У такому разі період буде дорівнювати $t_{clk}\approx 2,62$ мс, що більше t_d=2 мс.

VHDL-модель дільника частоти подана в лістингу 7.2.

Лістинг 7.2 – VHDL-модель дільника частоти

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity CT is
   Port (R, clk: in  STD_LOGIC;
    CT_out: out STD_LOGIC);
end CT;
architecture CT of CT is
constant N: INTEGER := 17;
signal COUNT_INT: STD_LOGIC_VECTOR(N-1 downto 0);
begin
process (CLK, R)
begin
    if R = '1' then
        COUNT_INT <= (others => '0');
    elsif CLK'event and CLK='1' then
        COUNT_INT <= COUNT_INT + 1;
     end if;
end process;
CT_out<='1' when COUNT_INT(N-1)= '1' else
        '0';
end CT;
```

Замість CT_out<='1' when COUNT_INT(N-1)= '1' else '0'; можна було написати CT_out<= COUNT_INT(N-1).

На рисунку 7.7 наведені результати моделювання дільника частоти для N = 4 замість N = 17 для наочності.

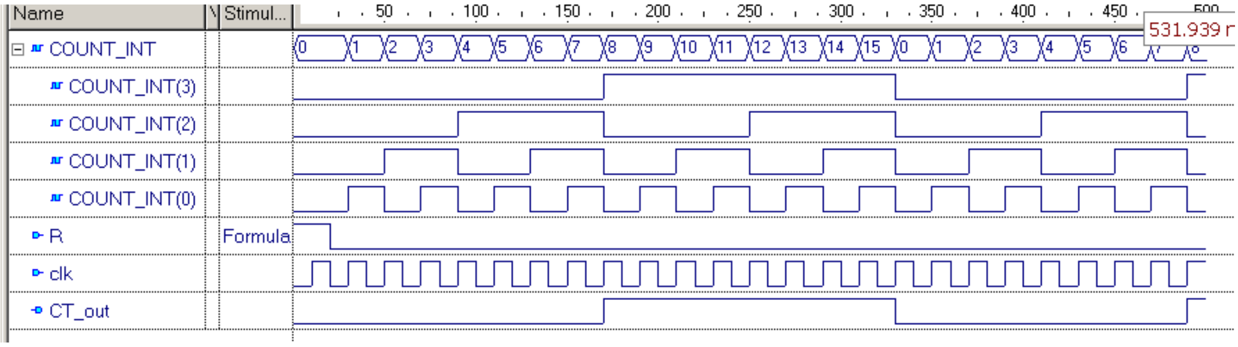


Рисунок 7.7 – Часові діаграми для дільника частоти
з перерахунковим коефіцієнтом 16

Для того, щоб скористатися протибрязкітливою схемою і дільником частоти при створенні проекту автомата, необхідно передбачити в блоці, що описує власне автомат, ще один керуючий вхід En. Розглянемо як приклад автомат Мура (рисунок 7.8).

VHDL-модель автомата Мура подана в лістингу 7.3. Керуючий вхід En описується в блоці формування послідовних частин. Синхронізується автомат заднім фронтом.

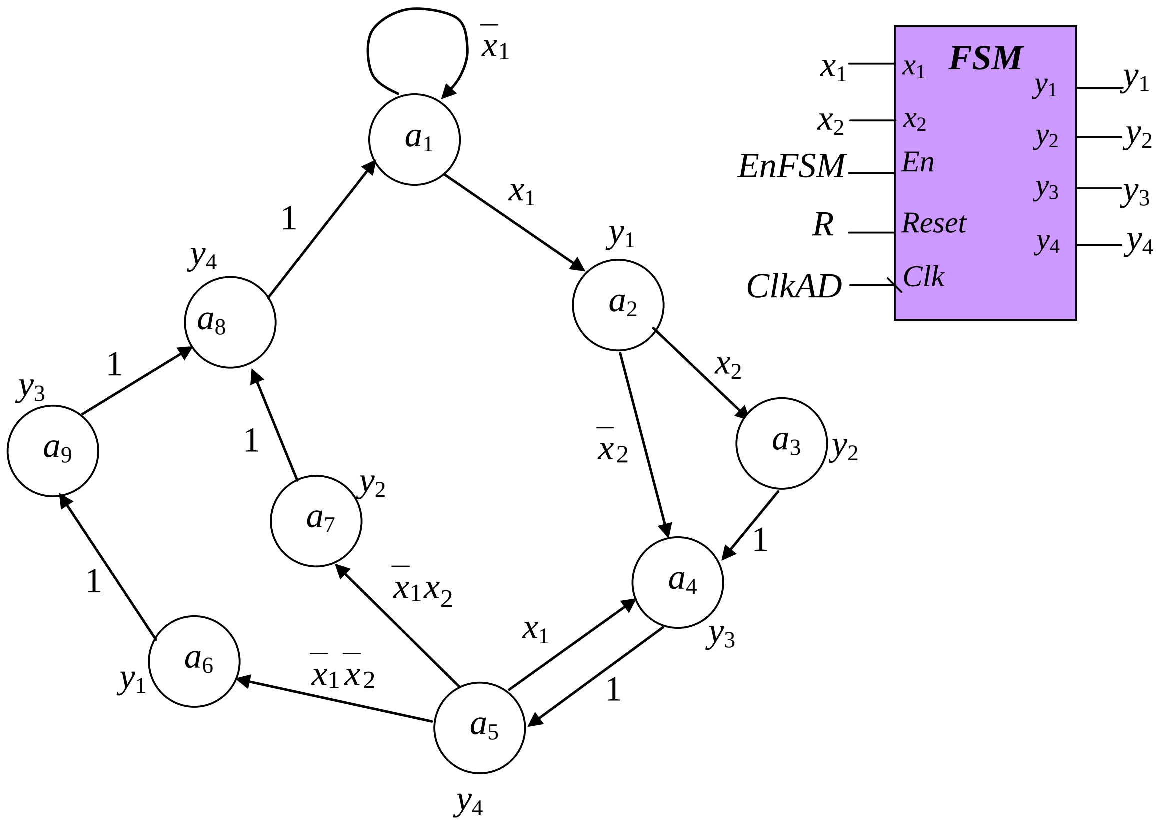


Рисунок 7.8 – Граф переходів автомата Мура

Лістинг 7.3 – VHDL-модель автомата Мура

```
library IEEE;
use IEEE.std_logic_1164.all;
-- Опис інтерфейсу пристрою
entity FSM is
    port ( Clk: in STD_LOGIC;
        Reset, En: in STD_LOGIC;
        x1, x2: in STD_LOGIC;
        y1, y2, y3, y4: out STD_LOGIC);
end;
-- Опис архітектури пристрою architecture FSM of FSM is
    type State_type is (a1, a2, a3, a4, a5, a6, a7, a8, a9);
    signal State, NextState: State_type;
begin
- Блок для формування послідовних частин
Sreg0_CurrentState: process (Clk, reset)
begin
```

```
    if Reset='1' then
       State <= a1;
    elsif Clk'event and Clk = '0' then
       if En='1'   then
       State <= NextState;
       end if;
    end if;
end process;
- Блок для формування комбінаційної частини
- Опис переходів станів за умови
Sreg0_NextState: process (State, x1, x2)
begin
       case State is
       when a1=> if x1='1' then     NextState <= a2;
                            else       NextState <= a1;
                 end if;
       when a2=> if x2='1' then     NextState <= a3;
                     else NextState <= a4;
              end if;
       when a3=> NextState <= a4;
       when a4=> NextState <= a5;
       when a5=> if x1='1' then     NextState <= a4;
              elsif x2='1' then NextState <= a7;
                     else  NextState <= a6;
                     end if;
     when a6=> NextState <= a9;
     when a7=> NextState <= a8;
    when a8=> NextState <= a1;
    when a9=> NextState <= a8;
       when others => NextState <= a1;
    end case;
end process;
- Опис вихідних сигналів
y1<='1' when State=a2 or State=a6 else      '0';
y2<='1' when State=a3 or State=a7 else      '0';
y3<='1' when State=a4 or State=a9 else      '0';
y4<='1' when State=a5 or State=a8 else      '0';
    end;
```

На рисунку 7.9 наведено часові діаграми роботи автомата Мура

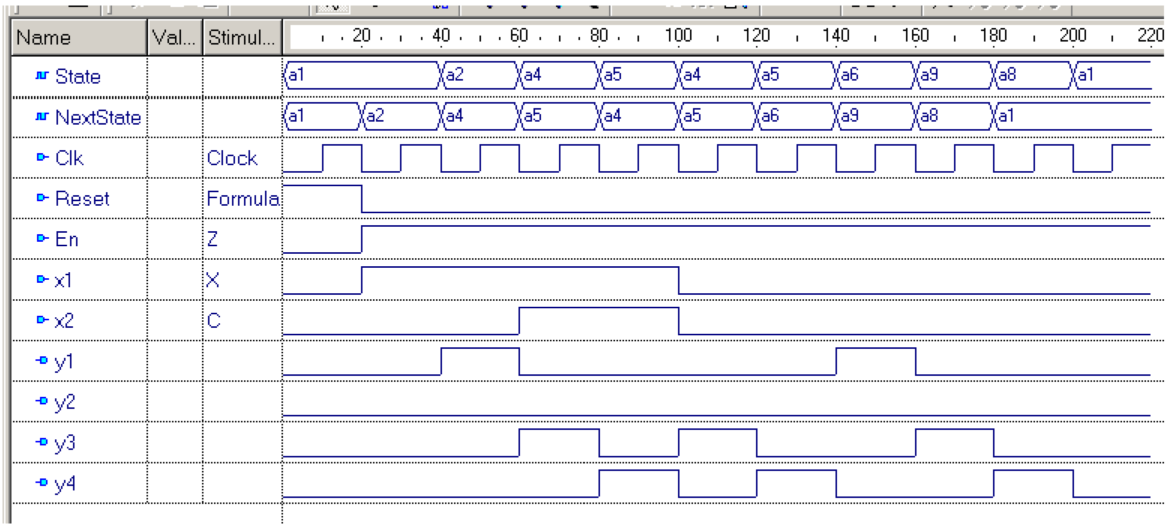


Рисунок 7.9 – Часові діаграми роботи автомата Мура

Цілком проект автомата Мура матиме вигляд як на рисунку 7.10.

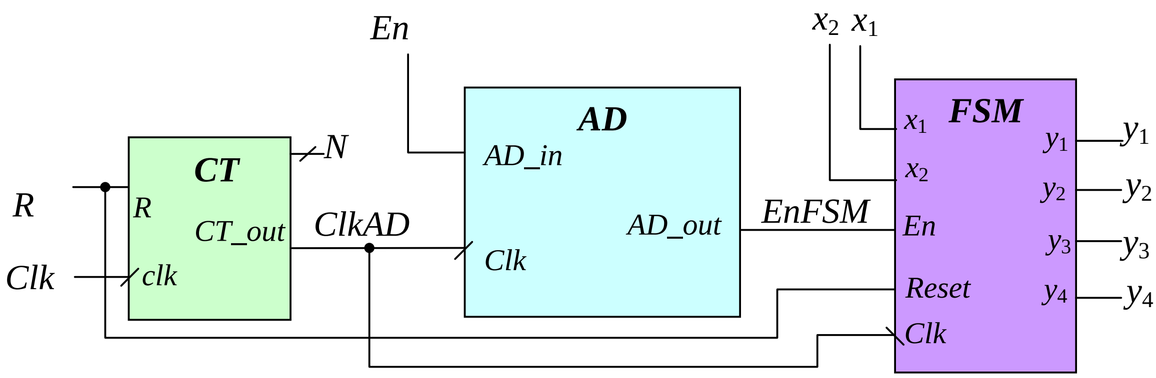


Рисунок 7.10 – Структурна схема для автомата Мура

Збільшена схема для автомата Мура подана рисунку 7.11.

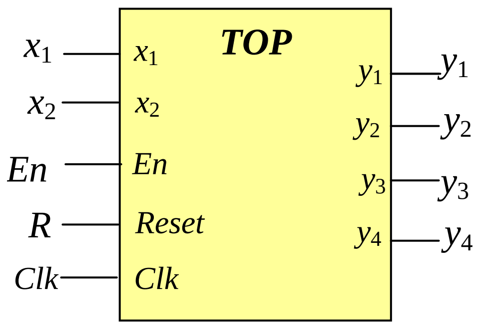


Рисунок 7.11 –Збільшена схема для автомата Мура

VHDL-модель збільшеної схеми автомата Мура подана в лістингу 7.4.

Лістинг 7.4 – VHDL-модель збільшеної схеми автомата Мура

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity top is
     port(
         R : in STD_LOGIC;
         CLK : in STD_LOGIC;
         En : in STD_LOGIC;
         x1 : in STD_LOGIC;
         x2 : in STD_LOGIC;
         y1 : out STD_LOGIC;
         y2 : out STD_LOGIC;
         y3 : out STD_LOGIC;
         y4 : out STD_LOGIC
         );
end top;
architecture top of top is
component CT
    Port (R, clk: in  STD_LOGIC;
     CT_out: out STD_LOGIC);
end component;
component AD
    Port (AD_in, clk: in  STD_LOGIC;
     AD_out: out STD_LOGIC);
```

```
end component;
component FSM
    port ( Clk: in STD_LOGIC;
        Reset, En: in STD_LOGIC;
        x1, x2: in STD_LOGIC;
        y1, y2, y3, y4: out STD_LOGIC);
end component;
signal EnFSM: STD_LOGIC;
signal ClkAD: STD_LOGIC;
begin
U1:CT
    port map (R=>R, clk=>CLK, CT_out=>ClkAD );
U2:AD
    port map (AD_in=>En, clk=>ClkAD, AD_out=>EnFSM );
U3:FSM
    port map (Clk=>ClkAD, Reset=>R, En=>EnFSM, x1=>x1,
        x2=>x2, y1=>y1, y2=>y2, y3=>y3, y4=>y4);
end top;
```

Стани автомата були закодовані системою наступним чином:

```
Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <U3/State/FSM> on signal <State[1:4]> with gray encoding.
-------------------
 State | Encoding
-------------------
 a1    | 0000
 a2    | 0001
 a3    | 0011
 a4    | 0010
 a5    | 0110
 a6    | 0101
 a7    | 0111
 a8    | 1100
 a9    | 0100
-------------------
```

7.2 Опис лабораторної установки

У лабораторній роботі використовуються пакет Xilinx ISE Project Navigator і плата Xilinx Spartan-3E Starter Kit. Методична література і додаткове програмне забезпечення розміщено в

http://kart.edu.ua/documentu-zvo-ua

Інструкція з використання програмних засобів міститься в тому ж каталозі.

7.3 Порядок виконання роботи

7.3.1 Ознайомтеся з літературою до лабораторної роботи. Перед виконанням роботи дайте відповіді на контрольні запитання.

7.3.2 Індивідуально виконайте запропонований варіант завдання.

7.3.3 Виконайте лабораторну роботу відповідно до таких пунктів:

1) нарисуйте граф переходів автомата Мура відповідно до варіанта завдання з керуючими й інформаційними сигналами. Для виконання цієї лабораторної роботи скористайтеся графом переходів станів автомата Мура з розрахункового завдання з канонічного синтезу мікропрограмних автоматів. До вершин графа додайте керуючі сигнали, а до дуг - умови переходів;

2) модифікуйте відповідно до вашого варіанта VHDL-модель керуючого автомата в уже готовому проекті, що додається до роботи;

3) промоделюйте автомат (модуль FSM) у системі Active-HDL для верифікації моделі FSM. Потім у системі Active-HDL відкомпілюйте весь проект (4 файли) – Compile all.

7.3.4 Створіть порожній проект у середовищі Project Navigator, у нього додайте за допомогою Project| Add copy of Sourse копію проекту (VHDL-файли всього проекту) з каталога src.

7.3.5 Виконайте синтез проекту.

7.3.6 Виконайте імплементацію проекту, використовуючи запропонований у методичних вказівках UCF файл.

Можна спробувати написати свій UCF файл. Використовуйте для цього перемикачі та кнопки, зазначені в таблиці 7.1 і на рисунках 7.12 і 7.13, і світлодіоди, зазначені в таблиці 7.2 і на рисунку 7.14.

Таблиця 7.1 - Перелік використовуваних перемикачів і кнопок для задавання вхідних впливів

Перемикачі	SW3	SW2	SW1	SW0	
FPGA Pin	N17	H18	L14	L13	
Кнопки	BTN_WEST	BTN_NORTH	BTN_EAST	BTN_SOUTH	ROT_CENTER
FPGA Pin	D18	V4	H13	K17	V16

Таблиця 7.2 – Перелік використовуваних світлодіодів для спостереження вихідних реакцій

Світлодіоди	LED7	LED6	LED5	LED4
FPGA Pin	F9	E9	D11	C11
Світлодіоди	LED3	LED2	LED1	LED0
FPGA Pin	F11	E11	E12	F12

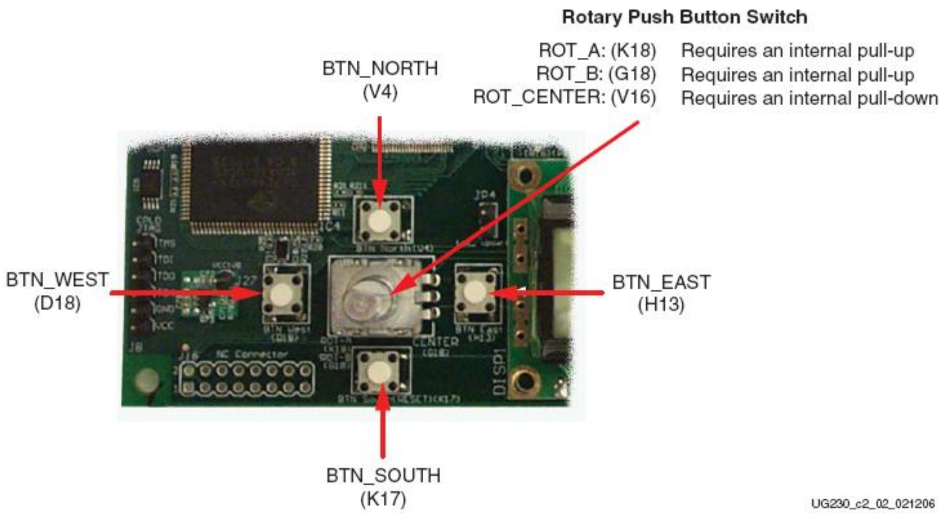


Рисунок 7.12 – Кнопки для задавання вхідних впливів

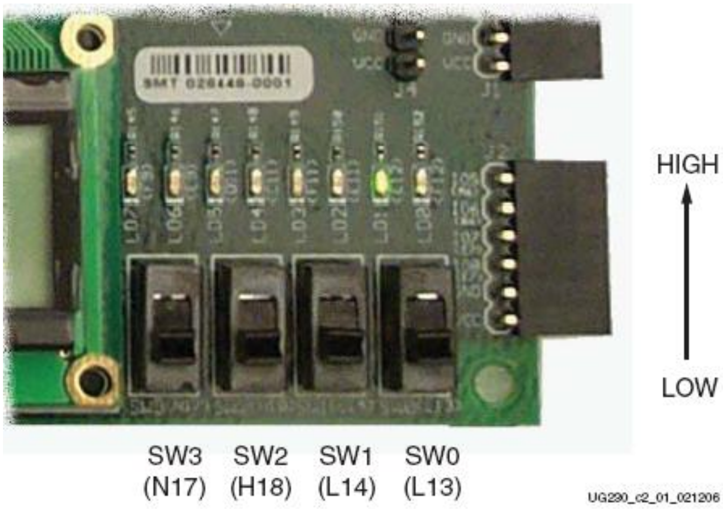


Рисунок 7.13 – Перемикачі вхідних впливів

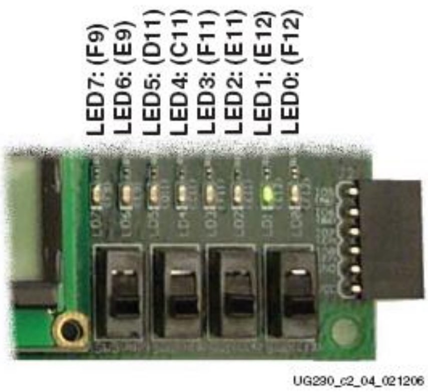


Рисунок 7.14 – Індикатори вихідних значень

Для того, щоб ненатиснута кнопка відповідала «0», а натиснута – «1», додається |IOSTANDARD = LVTTL | PULLDOWN після кожного призначення сигналу виведення мікросхеми. UCF файла. Наприклад: NET "a" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN;

У лістингу 7.5 наведено варіант UCF файла для проекту, що розглядається.

При імплементації ігноруйте попередження системи:
WARNING:Route:455 – CLK Net:U1/COUNT_INT<16> may have excessive skew because...

Лістинг 7.5 - Варіант UCF файла для автомата Мура

NET "Clk" LOC = "C9";
NET "R" LOC = "N17";
NET "En" LOC = "H18";
NET "x1" LOC = "L14";
NET "x2" LOC = "L13";
NET "y1" LOC = "F12";
NET "y2" LOC = "E12";
NET "y3" LOC = "E11";
NET "y4" LOC = "F11";

7.3.7 Виконайте безпосереднє програмування FPGA на платі **Xilinx Spartan-3E Starter Kit** і перевірте правильність роботи проекту.

Для цього виконуйте нижченаведені кроки (рисунок 7.15).

Крок 1. Один раз на початку перевірки встановіть перемикач R вгору (1), а потім вниз (0). Автомат перейде в початковий стан.

Крок 2. Потім встановіть необхідні значення перемикачів x_1, x_2.

Крок 3. Потім встановіть перемикач En вгору (1) і подивіться, у який стан перейде автомат (за світлодіодами, на яких відображаються сигнали). Встановіть перемикач En вниз (0). переходьте до кроку 2.

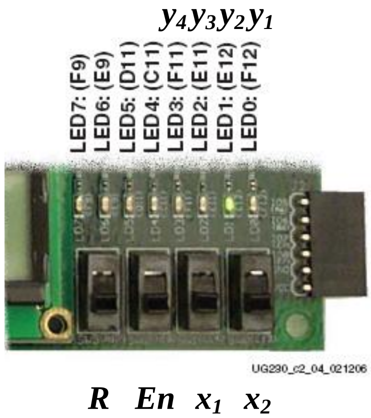


Рисунок 7.15 – Призначення сигналів на перемикачі і світлодіоди

7.3.8 Оформіть звіт про виконану лабораторну роботу, використовуючи шаблон.

7.4 Зміст звіту

Звіт оформляється кожним студентом індивідуально і повинен містити;

- титульний аркуш з номером і назвою роботи;
- мету роботи;
- граф переходів автомата;
- умовне графічне зображення досліджуваних елементів;
- VHDL-опис елементів;
- структурну схему всього проекту;
- часові діаграми перевірки правильності роботи елементів;
- лістинги VHDL-описів частин проекту;
- листинг UCF файла;
- результати автоматизованого синтезу проекту у вигляді графічного відображення схем елементів;
- таблицю кодування станів автомата із САПР;
- висновки до роботи.

Контрольні запитання

1 Що таке брязкіт механічних перемикачів?

2 Який період брязкоту перемикачів, установлених на платі Spartan-3E Starter Kit?

3 Яка частота генератора синхроімпульсів, установлених на платі Spartan-3E Starter Kit?

4 Як організовується макет для спостереження покрокової роботи автомата?

5 Що таке лічильник?

6 Для чого потрібна протибрязкітлива схема?

7 Що таке дільник частоти і для чого він використовується?

СПИСОК ЛІТЕРАТУРИ

1 Мірошник М. А. Проектування діагностичної інфраструктури обчислювальних систем і пристроїв на ПЛІС : монографія. Харків : ХУПС, 2012. 188 с.

2 Мірошник М. А. Комп’ютерні технології автоматизованого проектування : навч. посіб. Харків : ХНУРЕ, 2007. 300 с.

3 САПР пристроїв і систем автоматики. Основи систем автоматизації проектування : конспект лекцій для студ. спец. 123 «СКС» / упоряд. М. А. Мірошник. Харків : УкрДУЗТ, 2013. 102 с.

4 Привой С. Л. САПР цифровых устройств. Проектирование цифровых схем с помощью САПР WeBPACK ISE : учеб.-метод. пособ. по курсу «САПР цифровых устройств» для студ. спец. «Защита информации в телекоммуникациях» и «Телекоммуникационные системы» дневной формы обучения. Минск : БГУИР, 2006. 56 с.

5 Лістровий С. В., Мірошник М. А. Інформаційно-управляючі системи та організація паралельних обчислювань : навч. посіб. Харків : Діса плюс, 2015. 324 с.

6 Лістровий С. В., Мірошник М. А. Теорія автоматичного управління, штучний інтелект і автоматизація процесу прийняття рішення : навч. посіб. Харків : УкрДУЗТ, 2018. 144 с.

7 Хаханов В. І., Литвинова Є. І., Гузь О. А. Проектування і тестування цифрових систем на кристалах. Харків : ХНУРЕ, 2009. 484 с.

8 Семенець В. В., Хаханов В. І. Проектування цифрових схем з використанням мови VHDL : навч. посіб. Харків : ХНУРЕ, 2003. 492 с.

9 Хаханов В. І., Хаханова І. В. Verilog & SystemVerilog. Харків : Нове слово, 2010. 528 с.

10 Леонов С. Ю., Загарій Г. І. Автоматизоване проектування складних систем у комп'ютерній схемотехніці : навч. посіб. Харків : ПП видавництво «Нове слово», 2012. 287 с.

11 Боровик В. М., Гамаюн В. П. Автоматизоване робоче місце проектування інформаційних систем і баз даних: навч. посіб. Київ : Вид-во Нац. авіац. ун-ту «НАУ-друк», 2010. 128 с.

12 Хацук В. А. Проектирование в системе Xilinx ISE. URL : http://www.scan-west.com/xilinx ise.pdf.

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт та самостійної роботи

з дисципліни

*«ПРОЕКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ
ТА СИСТЕМ НА ПЛІС»*

Відповідальний за випуск Мірошник М. А.

Редактор Еткало О. О.

Підписано до друку 07.07.20 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 3,5. Тираж 5. Замовлення №

Видавець та виготовлювач Український державний університет
залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб’єкта видавничої справи ДК № 6100 від 21.03.2018 р.