МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

МЕТОДИЧНІ ВКАЗІВКИ І ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ З КУРСУ «МОВИ ОПИСУ АПАРАТУРИ»

для студентів 3-го курсу інженерно-технічного факультету,
напряму підготовки „Комп'ютерна інженерія"
Частина 2

Ужгород – 2018 року

Методичні вказівки і завдання до лабораторних робіт з курсу „Мови опису апаратури” для студентів 3-го курсу інженерно-технічного факультету, напряму підготовки „Комп’ютерна інженерія”. Частина 2.

Укладачі: Король І.Ю., канд. фіз.-мат. наук, доцент, завідувач кафедри комп’ютерних систем та мереж;
Тютюнникова Г.С., старший викладач кафедри комп’ютерних систем та мереж.

Рецензент: Ваврук Є.Я. – канд. техн. наук, доцент кафедри комп’ютерних систем та мереж, інженерно-технічного факультету, УжНУ.

Відповідальний за випуск – Туряниця І. І., канд. фіз.-мат. наук, професор, декан інженерно-технічного факультету.

Дані методичні вказівки розглянуто та схвалено на засіданні кафедри комп’ютерних систем та мереж, протокол №4 від “21 листопада 2018 року”.

ВСТУП

Метою викладення дисципліни «Мови опису апаратури» є ознайомлення з маршрутом проектування ПЛІС (Програмовані Логічні Інтегральні Схеми), способами опису проекту, загальним користувальницьким інтерфейсом системи MAX+plus II (Multiple Array Matrix Programmable Logic User System).

Вивчення дисципліни «Мови опису апаратури» дає студентам необхідну теоретичну і практичну підготовку для того, щоб вміти розробляти і аналізувати алгоритми перетворення дискретної інформації складних процесів, складати структурні схеми комбінаційних логічних схем та автоматів з пам’яттю, ефективно розв’язувати практичні задачі з використанням ЕОМ.

Дані методичні вказівки ознайомлять студентів з маршрутом проектування ПЛІС (Програмовані Логічні Інтегральні Схеми), способами вхідного опису проекту, загальним користувальницьким інтерфейсом системи MAX+plus II (Multiple Array Matrix Programmable Logic User System).

Виконання студентами лабораторних робіт з курсу «Мови опису апаратури» дозволяє закріпити теоретичні знання, надає можливість набути практичних навичок роботи з основними редакторами системи MAX+PLUS II та оволодіти методиками проектування і дослідження вузлів та пристроїв комп’ютера.

Кожній лабораторній роботі має передувати самостійна підготовка студентів, у процесі якої вони вивчають теоретичні відомості, що стосуються виконуваної роботи. При оформленні студент повинен сформулювати мету і порядок виконання лабораторної роботи і відповісти на контрольні питання викладача.

Перед початком наступного заняття в лабораторії студент зобов’язаний подати викладачеві повністю оформлений звіт з попередньої лабораторної роботи. Звіт повинен містити всі необхідні схеми, формули, таблиці, діаграми, графіки, одержані у процесі експериментального дослідження схем, а також підсумкові висновки.

Лабораторна робота № 6

Тема: Типові вузли цифрової схемотехніки. Побудова шифраторів та дешифраторів

Мета роботи: Набуття практичних навичок проектування шифраторів та дешифраторів з використанням пакету MAX+PLUS II та за допомогою мови опису апаратури AHDL.

ТИПОВІ ВУЗЛИ ЦИФРОВОЇ СХЕМОТЕХНІКИ

До типових вузлів належать логічні схеми, які найчастіше використовуються в цифрових ЕОМ. Серед них існують як комбінаційні так і послідовнісні схеми з пам'яттю.

До типових комбінаційних вузлів цифрової схемотехніки відносяться шифратори, дешифратори, мультиплексори і демультиплексори.

1. Дешифратори

Дешифратор (декодер) – це типова логічна комбінаційна схема з n інформаційними входами і 2^n виходами. Тобто це схема призначена для реалізації конституент одиниці.

Розрізняють *повні* і *неповні* дешифратори. Повні дешифратори реалізують 2^n конституент, де n – це число інформаційних входів. Неповні дешифратори реалізують менше ніж 2^n конституент. Функціонування повного дешифратора описується системою логічних виразів вигляду:

$$F_0 = \overline{X}_{n-1}\overline{X}_{n-2}\ldots\overline{X}_1\overline{X}_0;$$
$$F_1 = \overline{X}_{n-1}\overline{X}_{n-2}\ldots\overline{X}_1 X_0;$$
$$\ldots$$
$$F_{2^n-1} = X_{n-1}X_{n-2}\ldots X_1 X_0,$$

де $X_{n-1}X_{n-2}\ldots X_0$ – вхідні двійкові змінні; F_0, F_1, ..., F_{2^n-1} – вихідні логічні функції, що являють собою мінтерми. Якщо дешифратор неповний, то число виходів m менше ніж 2^n.

Таблицю істинності функцій повного дешифратора для $n=3$ наведено в табл.1.1.

Таблиця 1.1

X_2	X_1	X_0	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Індекс функції F_i визначає номер обраного виходу і відповідає десятковому еквіваленту вхідного коду. Вихід, на якому з'являється керуючий сигнал, називають ***активним***. Якщо значення сигналу на активному виході відображається логічною 1 (H), то на решті пасивних виходів встановлюється логічний 0 (L). Двійковий код, який завжди містить тільки одну одиницю, решта – нулі, називається ***унітарним***. Тому дешифратори є перетворювачами вхідного позиційного коду в унітарний вихідний.

Зауважимо, що крім позначень H і L для логічних входів і виходів можуть вживатись позначення X (байдужий – 0 чи 1) і Z , що відповідає буферу виходу зі Z станом.

У дешифраторах в інтегральному виконанні стан активного виходу часто відображається значення логічний 0 (L), а на інших пасивних виходах установлюється логічна 1 (H).

Функціонування повного дешифратора з інверсними виходами представляється логічними формулами вигляду:

$$G_0 = X_{n-1} \vee X_{n-2} \vee \ldots \vee X_1 \vee X_0;$$

$$G_1 = X_{n-1} \vee X_{n-2} \vee \ldots \vee X_1 \vee \overline{X}_0;$$

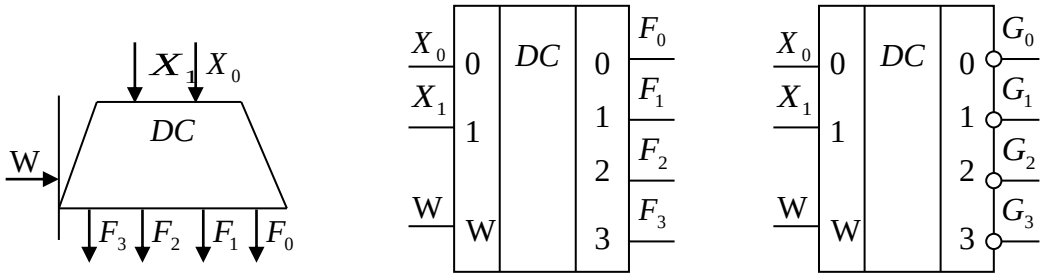
...

$$G_{2^n-1} = \overline{X}_{n-1} \vee \overline{X}_{n-2} \vee \ldots \vee \overline{X}_1 \vee \overline{X}_0,$$

де G_0, G_1, ..., G_{2^n-1} – вихідні логічні функції, що являють собою макстерми.

Умовні графічні позначення дешифраторів на електричних схемах показані на рис.1.1.

Логічна функція дешифратора позначається буквами DC (decoder). Мітки лівого додаткового поля в умовному позначенні відображають десяткові ваги вхідних комбінацій двійкових змінних, а мітки правого додаткового поля відповідають десятковим еквівалентам вихідних комбінацій двійкових змінних. У схему дешифраторів вбудовується один або два стробуючі (дозволяючі) входи, наприклад, *W* (рис. 1 б). За допомогою цього сигналу на вході визначають момент спрацювання дешифратора; крім того, вхід *W* використовується для нарощування розрядності вхідного коду.



а) *б)* *в)*

Рисунок 1.1 — Умовні графічні позначення дешифратора:
а — на функціональних схемах; б, в — на принципових схемах

Розглянемо деякі методи побудови дешифраторів з використанням пакету MAX+plus II та мови програмування апаратури AHDL.

1.1. Побудова дешифраторів з використанням пакету MAX+plus II

Приклад 1. Побудувати повний дешифратор при $n=3$ користуючись пакетом MAX+plus II та дослідити його роботу.

На рис. 1.2 наведено схему повного дешифратора та результати моделювання його роботи.

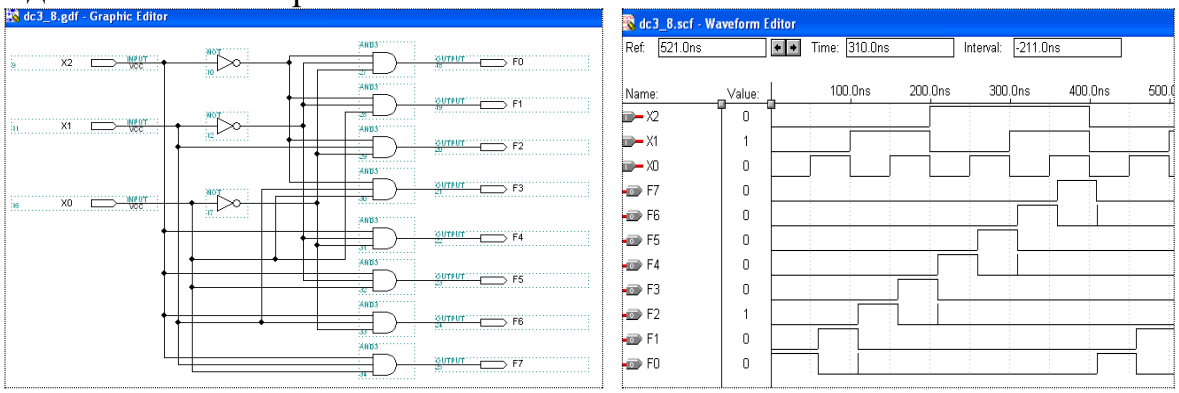


Рисунок 1.2 — Схема повного дешифратора та результати моделювання його роботи

1.2. Побудова дешифраторів з використанням мови опису апаратури AHDL

Приклад 2. Побудувати повний дешифратор при $n=3$ з використанням мови опису апаратури AHDL та дослідити його роботу.

Побудову шифраторів з використанням мови опису апаратури AHDL можна виконати декількома способами.

1^0. Реалізація повного дешифратора при $n=3$ з використанням оператора CASE мови AHDL (текстовий файл dc3_8_t). На рис. 1.3 наведено результати моделювання побудованого дешифратора.

```
SUBDESIGN dc3_8_t
(
  code[2..0]      : INPUT ;
  Out[7..0]       : OUTPUT;
)
BEGIN
  CASE code[] IS
          WHEN 0 => Out[]=B"00000001";
          WHEN 1 => Out[]=B"00000010";
          WHEN 2 => Out[]=B"00000100";
          WHEN 3 => Out[]=B"00001000";
          WHEN 4 => Out[]=B"00010000";
          WHEN 5 => Out[]=B"00100000";
          WHEN 6 => Out[]=B"01000000";
          WHEN 7 => Out[]=B"10000000";
  END CASE;
END;
```

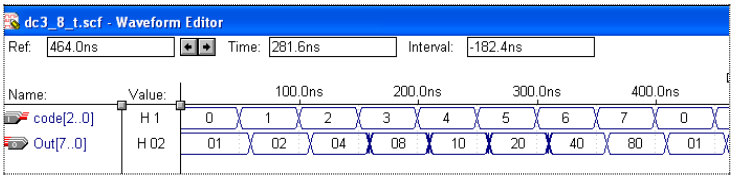


Рисунок 1.3 — Результати моделювання побудованого дешифратора

2^0. Реалізація повного дешифратора при $n=3$ з використанням таблиці істинності TABLE мови AHDL (текстовий файл dc3_8_tr_tb). На рис. 1.4 наведено результати моделювання роботи побудованого дешифратора.

```
SUBDESIGN dc3_8_tr_tb
        (i[2..0] : INPUT ;
        Y[7..0]: OUTPUT;)
BEGIN
        TABLE
                i[]=>Y[];
                 0 => 1;
                 1 => 2;
                 2 => 4;
                 3 => 8;
                 4 => 16;
                 5 => 32;
                 6 => 64;
                 7 => 128;
        END TABLE;
END;
```

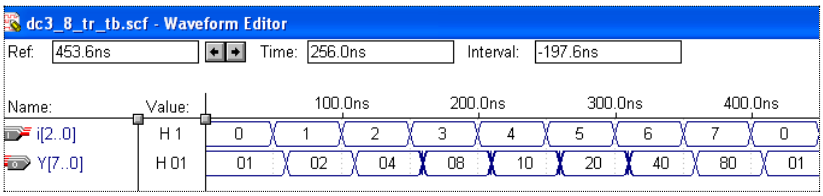


Рисунок 1.4 — Результати моделювання повного дешифратора при $n=3$ з використанням таблиці істинності TABLE мови AHDL

Приклад 3. Користуючись пакетом MAX+plus II побудувати дешифратор на два входи і чотири виходи, на одному з яких формується низький потенціал (логічний 0), на інших – високий (логічна 1). Нижче наведено таблицю істинності такого дешифратора (табл.1.2).

Таблиця 1.2

X_1	X_0	F_0	F_1	F_2	F_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Функції виходу для такого дешифратора мають вигляд:

$$Y0 = \overline{\overline{X}_1 \cdot \overline{X}_0}\,,\ Y1 = \overline{\overline{X}_1 \cdot X_0}\,,\ Y2 = \overline{X_1 \cdot \overline{X}_0}\,,\ Y3 = \overline{X_1 \cdot X_0}\,.$$

Схема та результати такого дешифратора наведено на рис. 1.5

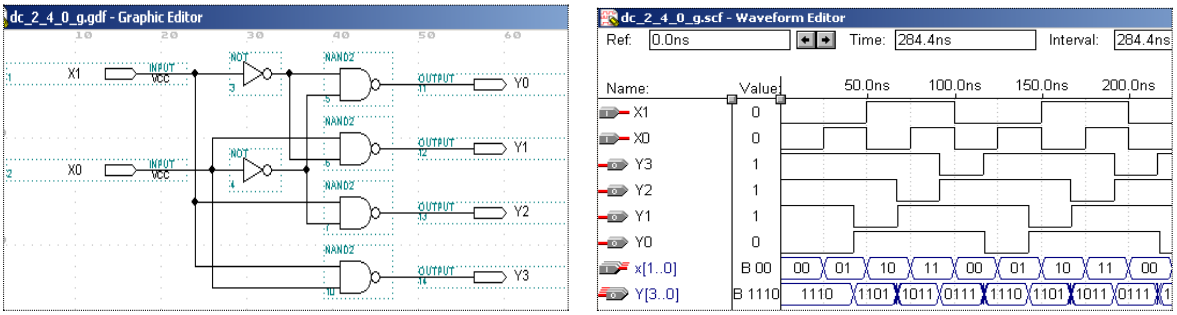


Рисунок 1.5 — Схема повного дешифратора на два входи і чотири виходи та результати моделювання його роботи

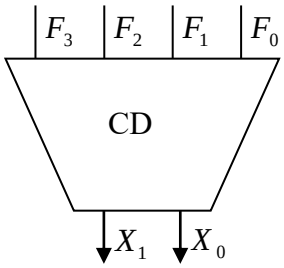
2. Шифратори

Шифратор – це типова комбінаційна схема, призначена для перетворення просторового унітарного коду (коду 1 із m) в двійковий код з природним порядком ваг.

Унітарний код, як уже відмічалось, має в своєму записі одну одиницю. З урахуванням цього можна вважати, що шифратор виконує функцію зворотну функції дешифратора, хоча в загальному випадку вихідний код може відрізнятися від коду з природним порядком ваг.

Повний двійковий шифратор має $m = 2^n$ входів і n виходів. Умовні графічні зображення шифратора при $n = 2$ на схемах наведені на рис. 2.1.

а)



б)

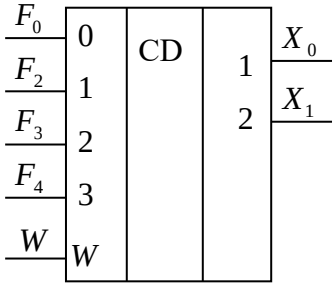


Рисунок 2.1 — Умовні графічні позначення шифратора:
а — на функціональних схемах; б, в — на принципових схемах.

Таблиця істинності шифратора для трирозрядного вхідного коду з природним порядком ваг наведено у наступній таблиці (табл.2.1).

Таблиця 2.1

X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	F_2	F_1	F_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

2.1. Побудова шифраторів з використанням пакету MAX+plus II

Приклад 4. Побудувати шифратор, заданий наведеною таблицею істинності та дослідити його роботу.

Функціональну схему шифратора, реалізовану на чотиривходових елементах АБО наведено на рис.2.2. Шляхом редагування символу одержати умовне графічне зображення шифратора, наведене на рис. 2.2. Результати моделювання роботи шифратора наведено на рис. 2.3.

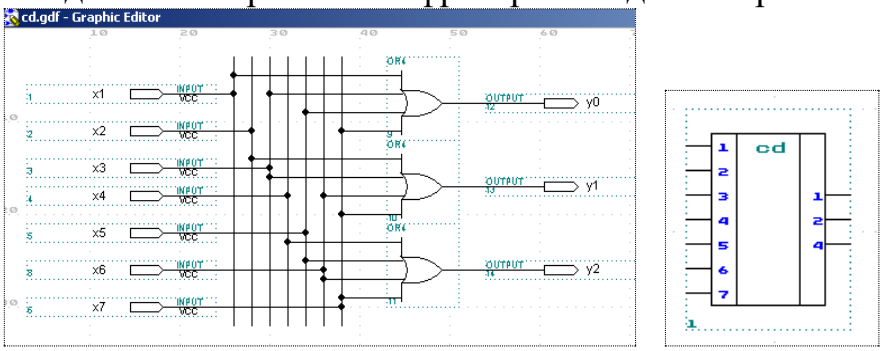


Рисунок 2.2 — Функціональна схема шифратора та його графічне зображення

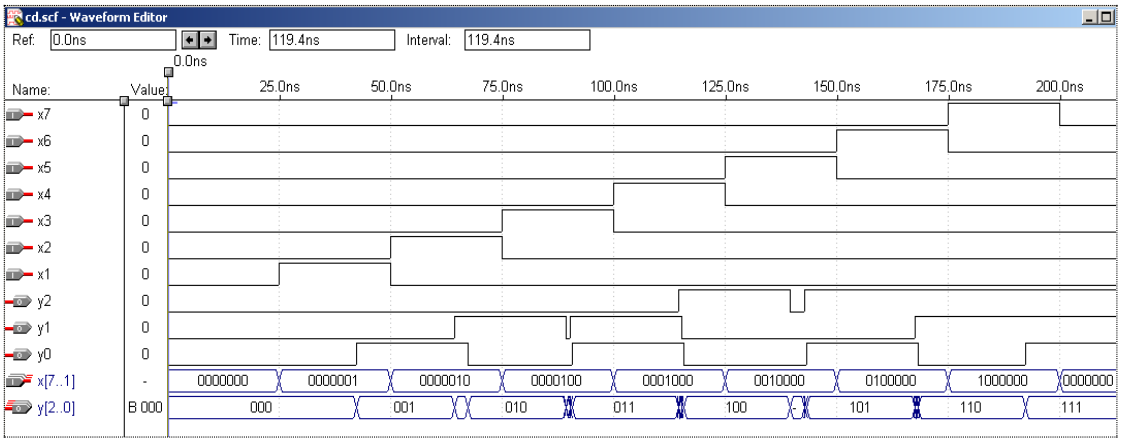


Рисунок 2.3 — Результати моделювання роботи шифратора

2.2. Побудова шифраторів з використанням мови опису апаратури AHDL

Приклад 5. Мовою AHDL з використанням таблиці істинності TABLE реалізувати шифратор з 8 в 3. Код програми (CD_8_3) наведено нижче.

```
SUBDESIGN CD_8_3
(cod_in[7..0]   : INPUT;
cod_out[2..0]  : OUTPUT;)
BEGIN
TABLE   cod_in[]      =>  cod_out[];
        B"00000001"   =>  0;
        B"00000010"   =>  1;
        B"00000100"   =>  2;
        B"00001000"   =>  3;
        B"00010000"   =>  4;
        B"00100000"   =>  5;
        B"01000000"   =>  6;
        B"10000000"   =>  7;
END TABLE;
END;
```

Результати моделювання роботи розглянутого шифратора наведено на рис. 2.4.

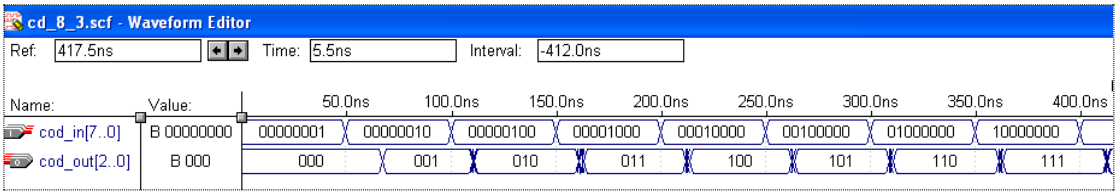


Рисунок 2.4 — Результати моделювання роботи шифратора з 8 в 3

Для введення даних з клавіатури зазвичай застосовується неповний шифратор 10→4. У такому шифраторі кожній десятковій цифрі відповідає свій двійковий код. Ознака помилки якщо одночасно натиснути більше однієї клавіші або не натиснуто жодної.

Приклад 6. Реалізувати проект CD_10_4 для введення даних з клавіатури. Результати моделювання наведено на рис. 2.5

```
        SUBDESIGN CD_10_4
(In[9..0]                          :INPUT;
 Binry_cod[4..1], EROR_:OUTPUT;)
BEGIN
        CASE In[] IS
        WHEN H"001" => Binry_cod[]=B"0000";
        WHEN H"002" => Binry_cod[]=B"0001";
        WHEN H"004" => Binry_cod[]=B"0010";
        WHEN H"008" => Binry_cod[]=B"0011";
        WHEN H"010" => Binry_cod[]=B"0100";
```

```
        WHEN H"020" => Binry_cod[]=B"0101";
        WHEN H"040" => Binry_cod[]=B"0110";
        WHEN H"080" => Binry_cod[]=B"0111";
        WHEN H"100" => Binry_cod[]=B"1000";
        WHEN H"200" => Binry_cod[]=B"1001";
        WHEN OTHERS => EROR_=VCC;
        END CASE;
END;
```

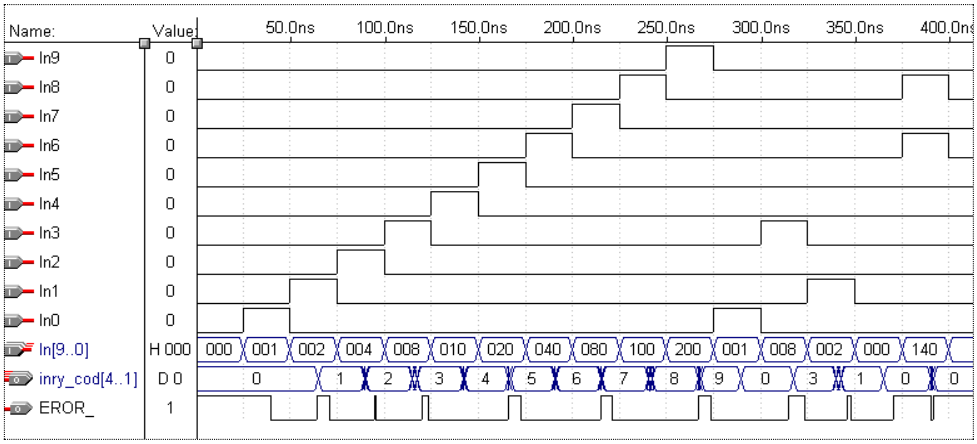


Рисунок 2.5 — Результати моделювання роботи неповного шифратора 10→4.

2.3. Пріоритетний шифратор клавіатури

Одне із основних застосувань шифратора – введення даних з клавіатури, наприклад десяткових цифр. Натискання клавіші з десятковою цифрою 0 , 1, ..., 9 мають приводити до передачі в цифровий пристрій двійково-десяткового коду цієї цифри. Для цього використовують неповний шифратор “з 10 в 4 ”.

D_i	A_3	A_2	A_1	A_0
D_0	0	0	0	0
D_1	0	0	0	1
D_2	0	0	1	0
D_3	0	0	1	1
D_4	0	1	0	0
D_5	0	1	0	1
D_6	0	1	1	0
D_7	0	1	1	1
D_8	1	0	0	0
D_9	1	0	0	1

Шифратори, які при одночасному натискуванні декількох клавіш виробляють код тільки старшої цифри, називаються ***пріоритетними***.

Логіка роботи пріоритетного шифратора на десять входів наведена в наступній таблиці, де $D_0, D_1, \ldots, D_9$ вхідні сигнали з десяткової клавіатури (наприклад, мікрокалькулятора); $A_3 A_2 A_1 A_0$ – вихідні сигнали, які задають двійково-десятковий код (двійкову тетраду) десяткової цифри. Оскільки є можливість натиснення зразу декількох клавіш, то це може привести до збудження декількох вхідних шин, яке в свою чергу приведе до видачі коду, який у загальному випадку, не буде співпадати з кодом жодної із натиснутих шин. Наприклад, якщо $D_5 = D_6 = 1$, то одержимо $A_3 A_2 A_1 A_0 = 0111$, що відповідає сигналу $D_7 = 1$. Усунути вказаний недолік можна шляхом застосування пріоритетних шифраторів, які при декількох збуджених вхідних шинах формують код однієї із них, наприклад, старшої.

У цьому випадку

$$A_3 = D_9 \vee D_8,$$

$$A_2 = \left(\overline{D_9 \vee D_8}\right) \wedge \left(D_7 \vee D_6 \vee D_5 \vee D_4\right),$$

$$A_1 = \left(\overline{D_9 \vee D_8}\right) \wedge \left(\left(D_7 \vee D_6\right) \vee \left(\overline{D_5 \vee D_4}\right) \wedge \left(D_3 \vee D_2\right)\right),$$

$$A_0 = D_9 \vee \overline{D}_8 \wedge \left(D_7 \vee \overline{D}_6 \wedge \left(D_5 \vee \overline{D}_4 \wedge \left(D_3 \vee \overline{D}_2 \wedge D_1\right)\right)\right).$$

Завдання для самостійної роботи

Приклад 1. Користуючись пакетом **MAX+plus II** реалізувати проект пріоритетного шифратора.

На рис.2.6 наведено схему пріоритетного шифратора, реалізовану за допомогою пакету MAX+plus II, а на рис. 2.7 результати моделювання

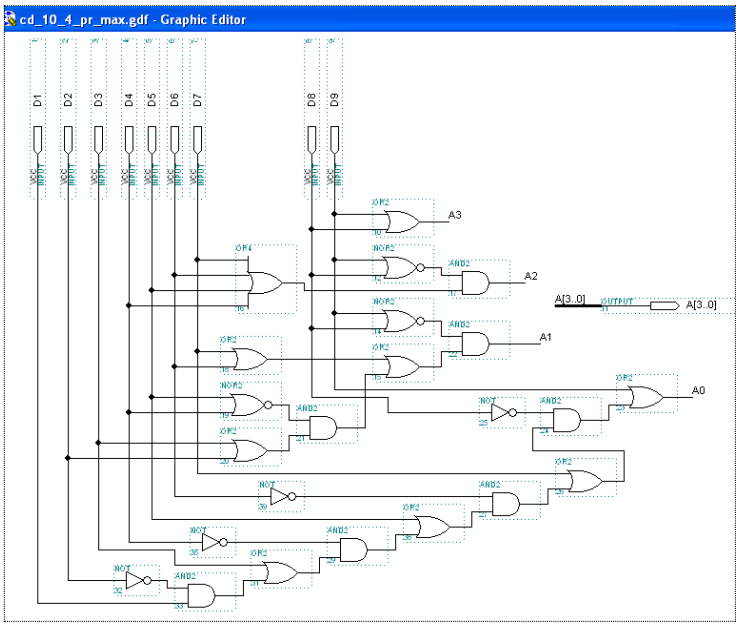


Рисунок 2.6 — Схема пріоритетного шифратора, реалізовану за допомогою MAX+plusII

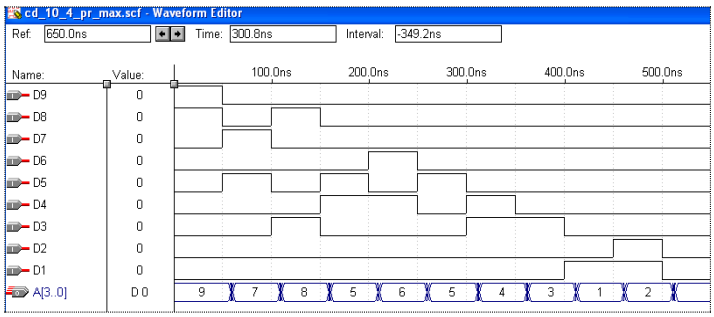


Рисунок 2.7 — Результати моделювання пріоритетного шифратора

Приклад 2. Побудова пріоритетного шифратора з використанням мови AHDL.

Розглянемо декілька варіантів побудови пріоритетного шифратора з використанням мови AHDL.

Нижче наведений текстовий файл CD_10_4_pr реалізовує пріоритетний шифратор, який еквівалентний шифратору, реалізованому в пакеті MAX+plus II. На рис. 2.8 наведено результати моделювання такого шифратора.

```
SUBDESIGN CD_10_4_pr
(D[9..1]        : INPUT;
        A[3..0]: OUTPUT;)
BEGIN
A3=D9#D8;
A2=!(D9#D8)&(D7#D6#D5#D4);
A1=!(D9#D8)&((D7#D6)#!(D5#D4)&(D3#D2));
A0=D9#!D8&(D7#!D6&(D5#!D4&(D3#!D2&D1)));
END;
```

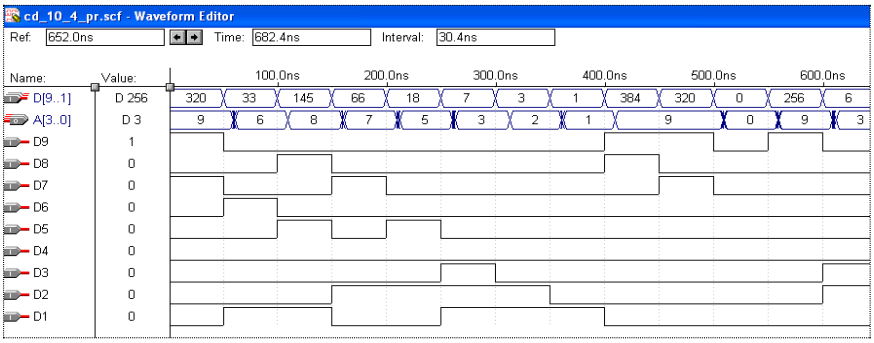


Рисунок 2.8 — Результати моделювання пріоритетного шифратора з використанням мови AHDL

Нижче наведений файл, що дає можливість вводити і цифру 0. Результати моделювання наведено на рис.2.9.

```
SUBDESIGN CD_10_4_pr
(D[9..0]        : INPUT;
        A[3..0]: OUTPUT;)
BEGIN
IF D0!=0 THEN
A3=D9#D8;
A2=!(D9#D8)&(D7#D6#D5#D4);
A1=!(D9#D8)&((D7#D6)#!(D5#D4)&(D3#D2));
A0=D9#!D8&(D7#!D6&(D5#!D4&(D3#!D2&D1)));
ELSE
A[3..0]=0;
END IF;
END;
```

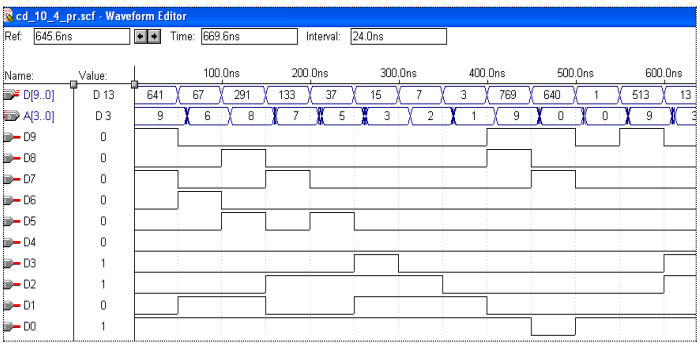


Рисунок 2.9 — Результати моделювання пріоритетного шифратора

Лабораторна робота № 7

Тема: Типові вузли цифрової схемотехніки. Побудова мультиплексорів та демультиплексорів

Мета роботи: Набуття практичних навичок проектування мультиплексорів та демультиплексорів з використанням пакету MAX+PLUS II та за допомогою мови опису апаратури AHDL.

1. Мультиплексори

Цифровий мультиплексор або ***селектор даних*** – комбінаційна логічна схема, яка являє собою керований перемикач, який приймає декілька інформаційних сигналів, вибирає один із них і передає його на вихід (рис. 1.1).

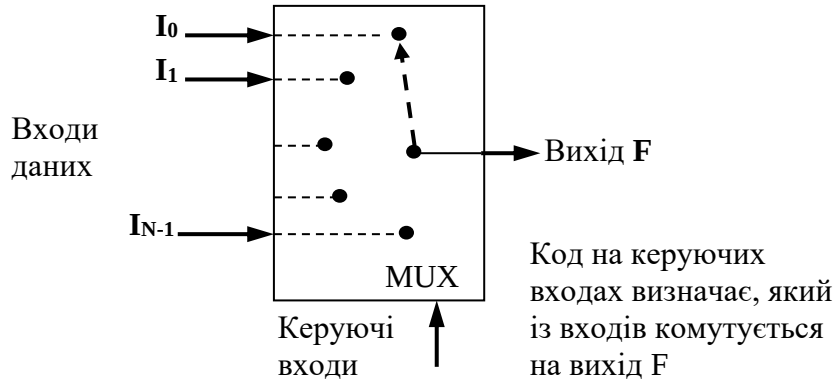


Рисунок 1.1 — Функціональна схема мультиплексора

Номер під'єднаного входу дорівнює числу (адресі), яке визначається комбінацією логічних значень на входах керування. Крім інформаційних входів і входів керування, схема мультиплексора може містити вхід дозволу, при подачі на який активного рівня, мультиплексор переходить в пасивний стан, при якому сигнал на виході зберігає постійне значення незалежно від значення інформаційних і керуючих сигналів. Число інформаційних входів у мультиплексора, як правило, є 2, 4, 8 або 16.

Мультиплексори застосовуються, наприклад, у мікропроцесорах для видачі на одні і ті ж виводи адреси і даних, що дає можливість істотно скоротити загальну кількість виводів мікросхеми; у мікропроцесорних системах керування мультиплексори встановлюються на віддалених об'єктах для можливості передачі інформації по одній лінії від декількох встановлених на них датчиків.

Мультиплексор з чотирма входами даних.

Умовне графічне позначення мультиплексорів показане на рис. 1.2.

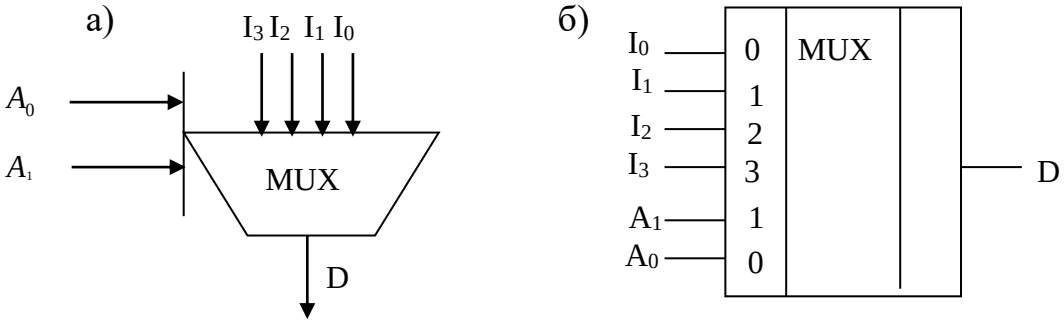


Рисунок 1.2 —Умовне позначення мультиплексорів:
а) на функціональних схемах; б) – на принципових схемах

Таблиця 1.1

A_0	A_1	F_0	F_1	F_2	F_3	D
0	0	1	0	0	0	F_0I_0
0	1	0	1	0	0	F_1I_1
1	0	0	0	1	0	F_2I_2
1	1	0	0	0	1	F_3I_3

Логіка роботи чотиривходового мультиплексора наведена в табл. 1, де A_0, A_1 – адресний код; F_0, F_1, F_2, F_3 – виходи внутрішнього дешифратора; I_0, I_1, I_2, I_3 – вхідна інформація; D – загальний інформаційний вихід.

На основі табл.1.1 вираз для вихідної функції D можна подати з використанням виходів $F_0 - F_3$ внутрішнього дешифратора у вигляді

$$D = F_0 I_0 \vee F_1 I_1 \vee F_2 I_2 \vee F_3 I_3, \qquad (1.1)$$

або з мінтермами адресного коду

$$D = \overline{A}_1 \overline{A}_0 I_0 \vee \overline{A}_1 A_0 I_1 \vee A_1 \overline{A}_0 I_2 \vee A_1 A_0 I_3. \qquad (1.2)$$

1.1. Побудова мультиплексорів з використанням пакету MAX+PLUS II

Приклад 1. Побудувати мультиплексори з використанням пакету MAX+PLUS II.

Схеми мультиплексорів та результати моделювання, які відповідають рівнянням (1.1), (1.2) показані на рис.1.3 — 1.4.

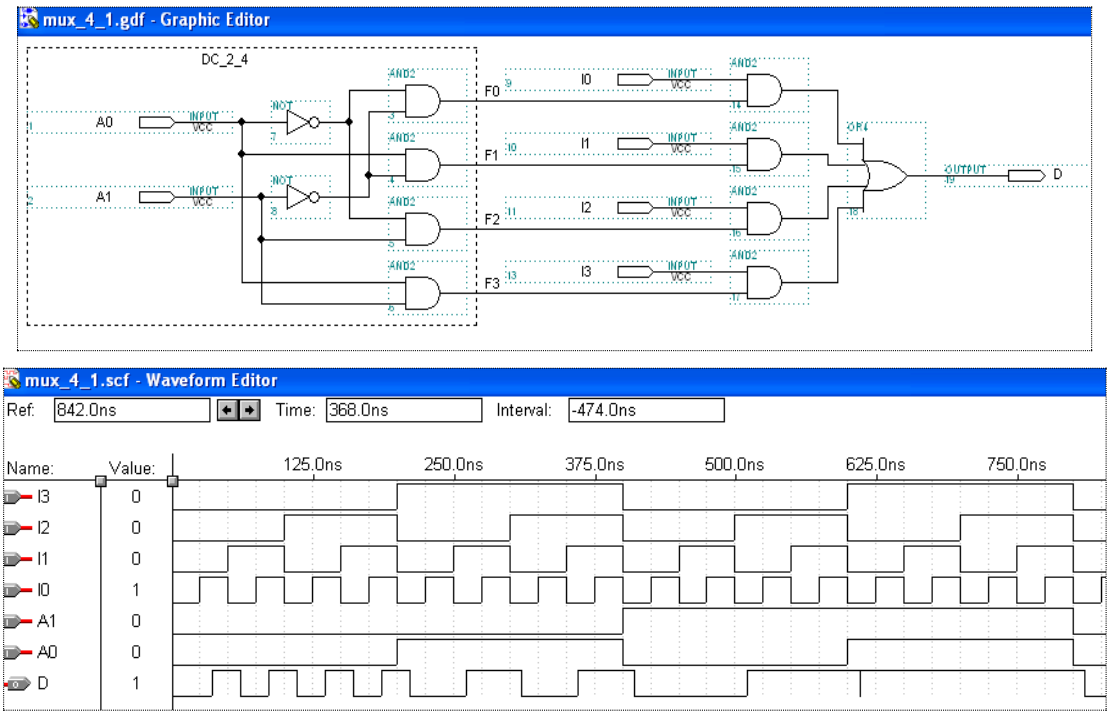


Рисунок 1.3 — Схема мультиплексора та результати моделювання, які відповідають рівнянням (1.1)

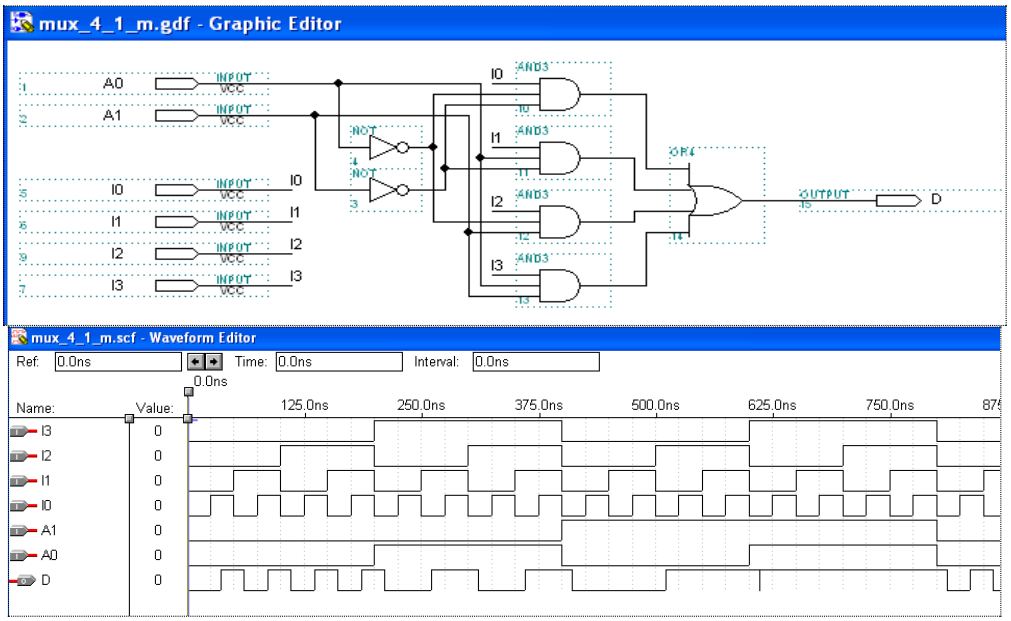


Рисунок 1.4 — Схема мультиплексора та результати моделювання, які відповідають рівнянням (1.2)

На рис. 1.5 наведено схему і символ внутрішнього дешифратора.

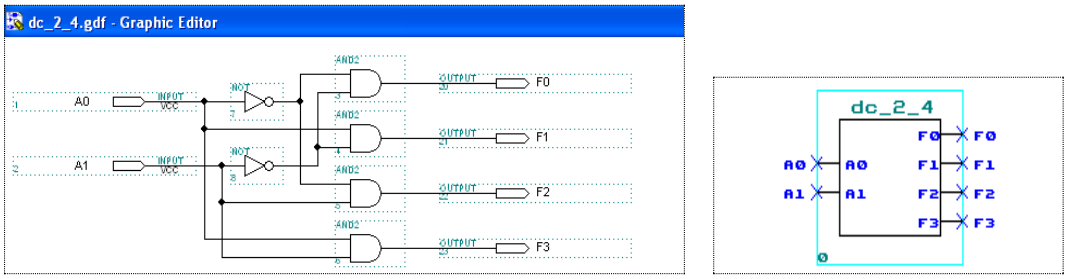


Рисунок 1.5 — Схема і символ внутрішнього дешифратора

1.2. Побудова мультиплексорів з використання мови опису апаратури AHDL

Приклад 2. Нижче наведено текстовий файл mux_4_1_t, який реалізовує мультиплексор 4→1 з використанням внутрішнього дешифратора. Результат моделювання даного дешифратора наведено на рис. 1.6.

```
SUBDESIGN mux_4_1_t
(       A[1..0],  I[3..0]: INPUT ;
        D: OUTPUT;)
VARIABLE
        F[3..0]:NODE;
BEGIN
TABLE
        A[]=>F[];
        B"00"=>B"0001";
        B"01"=>B"0010";
        B"10"=>B"0100";
        B"11"=>B"1000";
END TABLE;
D=F0&I0#F1&I1#F2&I2#F3&I3;
END;
```

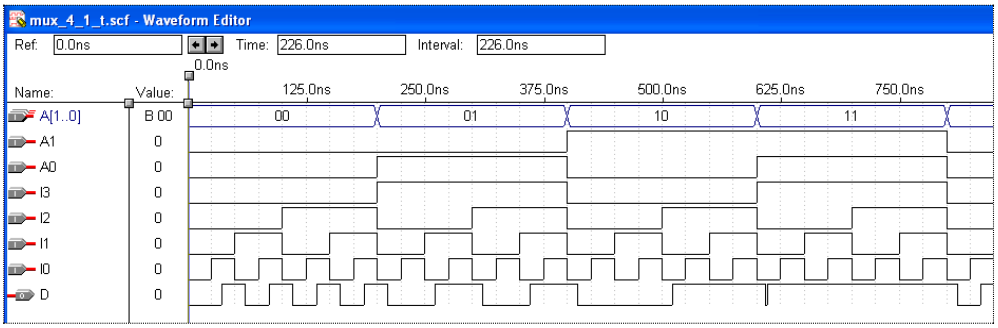


Рисунок 1.6 — Результати моделювання мультиплексор 4→1 з використанням внутрішнього дешифратора

Приклад 3. Нижче наведений текстовий файл mux_4_1_tm, який реалізовує мультиплексор 4→1 з використанням адресних мінтермів. Результат моделювання даного мультиплексора наведено на рис.1.7.

```
SUBDESIGN mux_4_1_tm
(A[1..0],  I[3..0]: INPUT ;
      D: OUTPUT;)
BEGIN
D=!A1&!A0&I0#!A1&A0&I1#A1&!A0&I2#A1&A0&I3;
END;
```

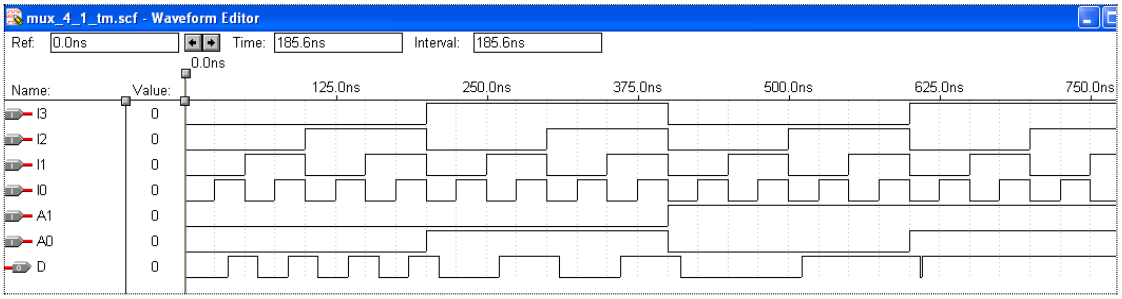


Рисунок 1.7 — Результати моделювання мультиплексора 4→1 з використанням адресних мінтермів

Приклад 4. а) Текстовий файл mux_8_1_tm реалізовує мультиплексор 8→1 у шинному виконанні

```
SUBDESIGN mux_8_1_tm
(A[2..0],  I[7..0]: INPUT ;
      D: OUTPUT;)
BEGIN
D=!A2&!A1&!A0&I0#!A2&!A1&A0&I1#!A2&A1&!A0&I2#!A2&A1&A0&I3#
      A2&!A1&!A0&I4#A2&!A1&A0&I5#A2&A1&!A0&I6#A2&A1&A0&I7;
END;
```

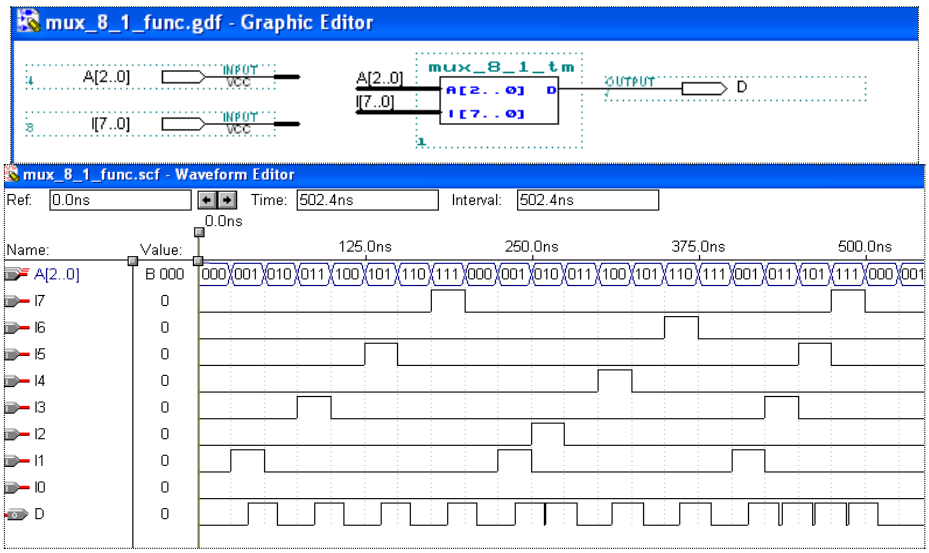


Рисунок 1.8 — Результати моделювання мультиплексора 8→1 у шинному виконанні

Приклад 4. б) Текстовий файл mux_8_1 реалізовує мультиплексор $8 \rightarrow 1$ у провідниковому виконанні. Реалізація даного мультиплексора з використанням символу та результат його моделювання наведено на рис.1.9.

```
SUBDESIGN mux_8_1
(A2,A1,A0, I7,I6,I5,I4,I3,I2,I1,I0: INPUT ;
      D: OUTPUT;)
BEGIN
D=!A2&!A1&!A0&I0#!A2&!A1&A0&I1#!A2&A1&!A0&I2#!A2&A1&A0&I3#
        A2&!A1&!A0&I4#A2&!A1&A0&I5#A2&A1&!A0&I6#A2&A1&A0&I7;
END;
```

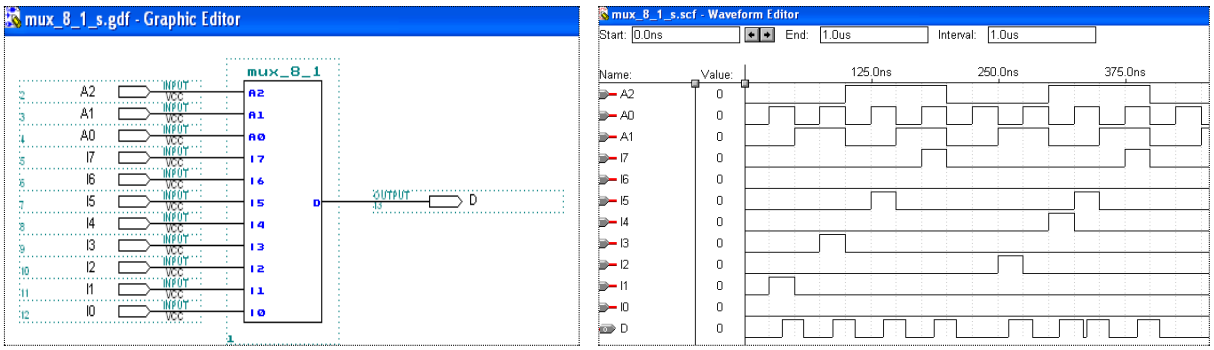


Рисунок 1.9 — Мультиплексор з використанням символу та результати моделювання

Приклад 5. Текстовий файл mux_8_1_en реалізовує мультиплексор $8 \rightarrow 1$ з входом дозволу роботи En. Результат моделювання даного мультиплексора наведено на рис. 1.10.

При $En = 0$ на виході мультиплексора буде формуватися сигнал логічного нуля незалежно від стану інших виходів.

Якщо $En = 1$, то сигнал на виході мультиплексора адресним кодом Adr[] і рівнем сигналу на відповідному інформаційному вході.

```
SUBDESIGN mux_8_1_en
(Inf[7..0], Adr[2..0], En: INPUT;
      D: OUTPUT;)
BEGIN
      IF En THEN
        CASE Adr[] IS
        WHEN 0 => D=Inf[0];
        WHEN 1 => D=Inf[1];
        WHEN 2 => D=Inf[2];
        WHEN 3 => D=Inf[3];
        WHEN 4 => D=Inf[4];
        WHEN 5 => D=Inf[5];
        WHEN 6 => D=Inf[6];
        WHEN 7 => D=Inf[7];
        END CASE;
      END IF;
END;
```

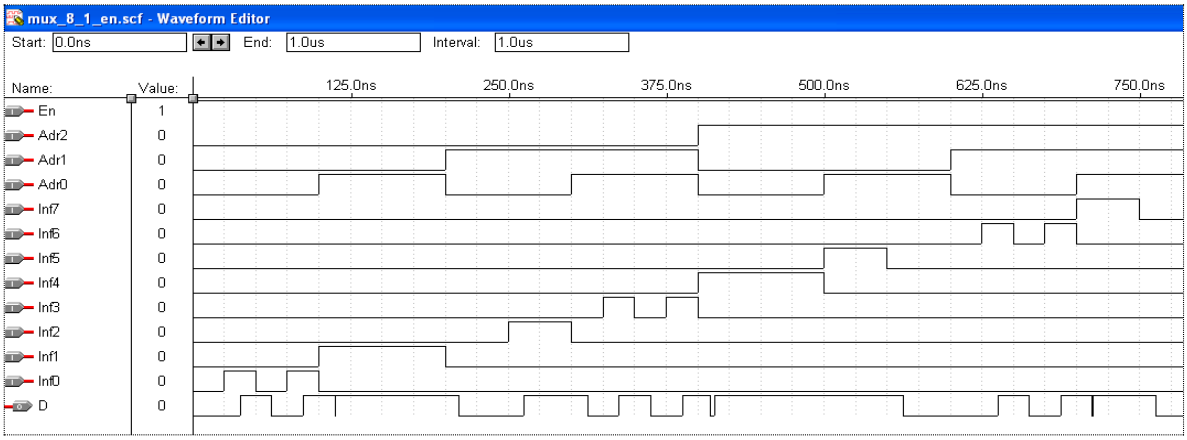


Рисунок 1.10 — Результати моделювання мультиплексора $8 \rightarrow 1$ з входом дозволу En

2. Демультиплексори

Демультиплексором називається функціональний вузол комп'ютера, який призначено для комутації (перемикання) сигналу з одного інформаційного входу D на один з n інформаційних виходів. Номер виходу, на який в кожний такт машинного часу передається значення вхідного сигналу, визначається адресним кодом $A_0, A_1, \ldots, A_m$. Адресні входи m та інформаційні виходи n пов'язані співвідношенням $n = 2^m$ або $m = \log_2 n$.

Демультиплексор виконує функцію, обернену функції мультиплексора.

В умовних графічних позначеннях функція демультиплексора позначається буквами *DMX* . Крім цього часто використовують позначення $1 \rightarrow n$. На рис.1.11 наведено умовні графічні позначення демультиплексорів: а) – на функціональних схемах; б) – на принципових схемах.

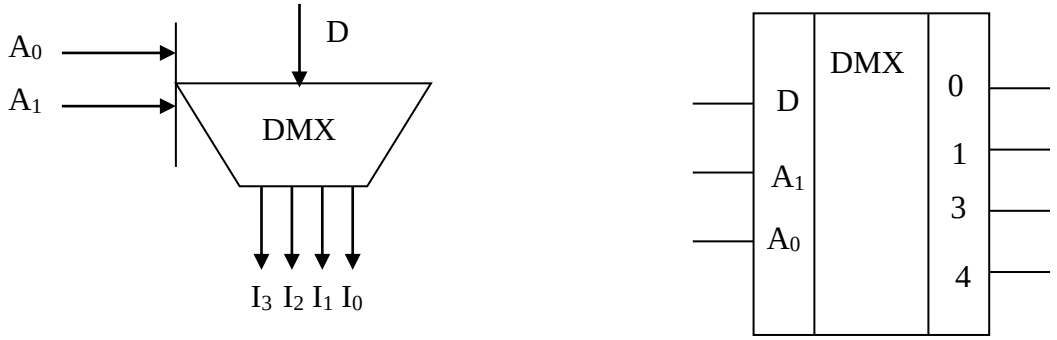


Рисунок 2.1 — Умовні графічні позначення демультиплексорів: а) – на функціональних схемах; б) – на принципових схемах.

Демультиплексори використовуються для наступних операцій:

- комутації як окремих ліній, так і багаторозрядних шин;
- перетворення послідовного коду в паралельний;
- реалізації логічних функцій та ін.

Логіка роботи двоадресного мультиплексора на мові мікрооперацій на ведена в табл.2.1, де A_0, A_1 – адресний код; F_0, F_1, F_2, F_3 – виходи внутрішнього дешифратора адреси; I_0, I_1, I_2, I_3 – вихідна інформація; D – загальний інформаційний вхід.

Таблиця 2.1

A_1	A_0	F_0	F_1	F_2	F_3	I_0	I_1	I_2	I_3
0	0	1	0	0	0	F_0D	–	–	–
0	1	0	1	0	0	–	F_1D	–	–
1	0	0	0	1	0	–	–	F_2D	–
1	1	0	0	0	1	–	–	–	F_3D

За даними табл.2.1 записуємо систему рівнянь для інформаційних виходів:

$$I_0 = F_0D = \overline{A}_1\overline{A}_0D,\ \ I_1 = F_1D = \overline{A}_1A_0D,\ \ I_2 = F_2D = A_1\overline{A}_0D,\ \ I_3 = F_3D = A_1A_0D. \qquad (2.1)$$

2.1. Побудова мультиплексорів з використанням пакету MAX+PLUS II

Приклад 6. На основі рівнянь (2.1) з використанням пакету MAX+PLUS побудовані схеми демультиплексорів із внутрішнім дешифратором (рис.2.2) та з поєднанням адресних вхідних змінних на тривходових елементах *I* (рис.2.4). Результати моделювання даних демультиплексорів наведено, відповідно, на рис.2.3 та рис.2.5.

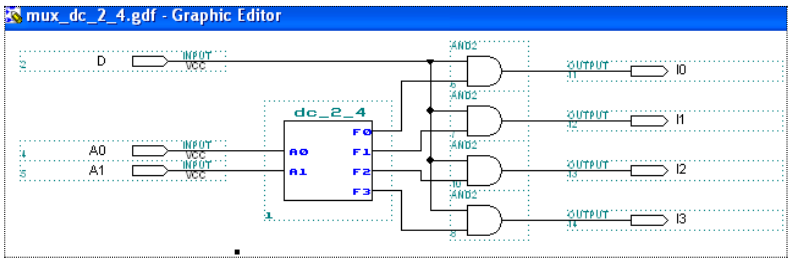


Рисунок 2.2 — Мультиплексор з із внутрішнім дешифратором

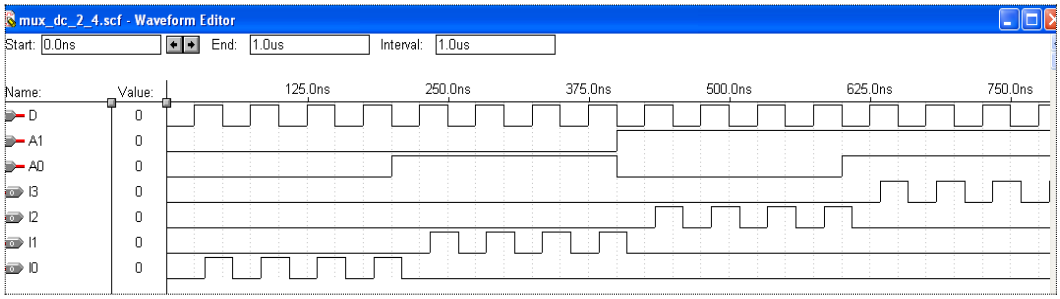


Рисунок 2.3 — Результати моделювання мультиплексора із внутрішнім дешифратором

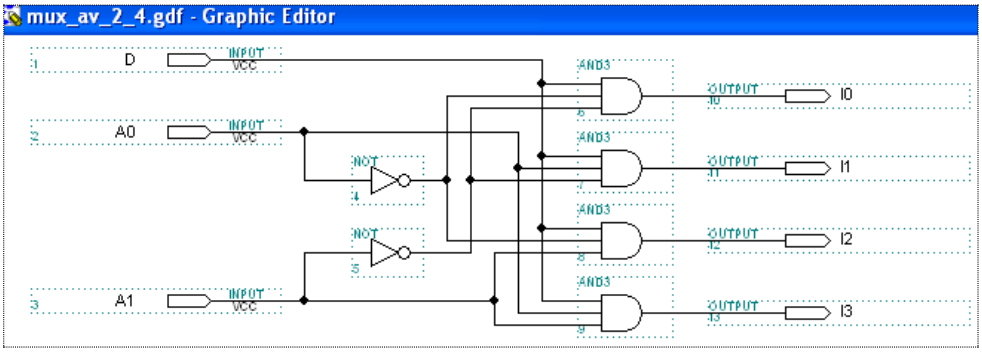


Рисунок 2.4 — Мультиплексор з поєднанням адресних вхідних змінних на тривходових елементах *I*

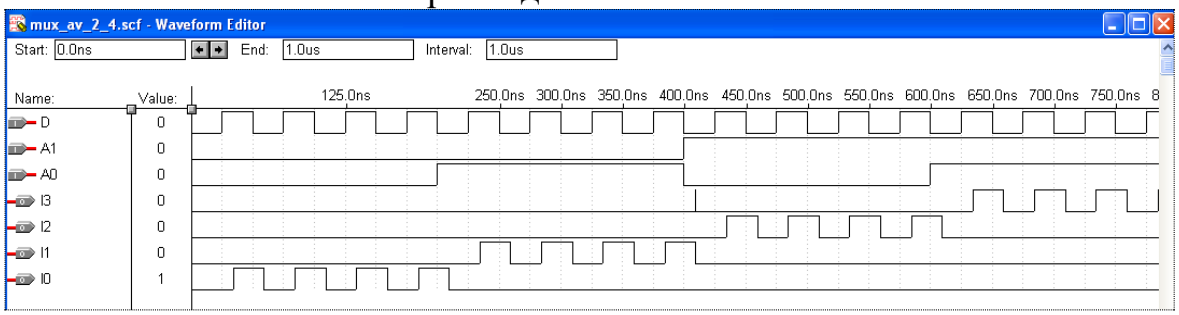


Рисунок 2.5 — Результати моделювання мультиплексора з поєднанням адресних вхідних змінних на тривходових елементах *I*

2.2. Побудова демультиплексорів з використання мови опису апаратури AHDL

Приклад 7. Нижче наведено текстовий файл DMX_1_4_T, який реалізовує демультиплексор $1 \rightarrow 4$ з використанням внутрішнього дешифратора. Результат моделювання даного дешифратора наведено на рис.2.6.

```
SUBDESIGN DMX_1_4_T
(D, A[1..0]     : INPUT;
        I[3..0]  : OUTPUT;)
VARIABLE
         F[3..0]:NODE;
BEGIN
        TABLE
                A[]=>F[];
                B"00"=>B"0001";
                B"01"=>B"0010";
                B"10"=>B"0100";
                B"11"=>B"1000";
        END TABLE;
        FOR k IN 0 TO 3 GENERATE
                I[k]=F[k]&D;
                END GENERATE;
END;
```

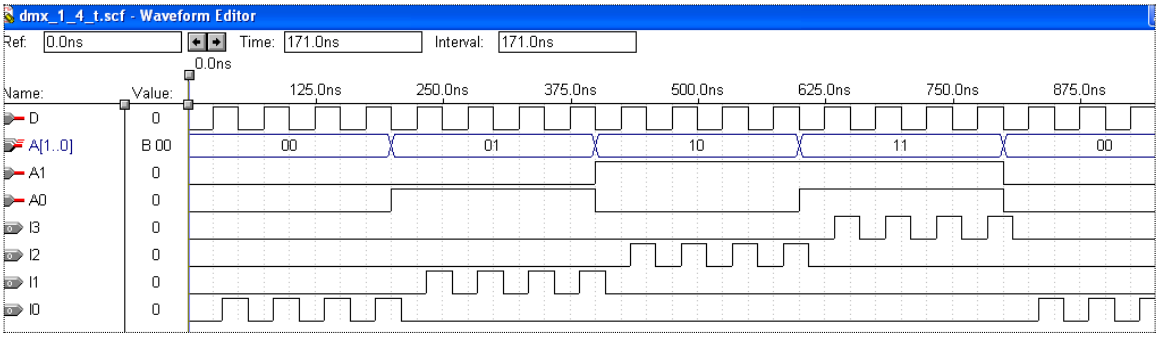


Рисунок 2.6 — Результати моделювання демультиплексора $1 \rightarrow 4$ з використанням внутрішнього дешифратора

Приклад 8. Нижче наведений текстовий файл DMX_1_4_T1, який реалізовує мультиплексор $1 \rightarrow 4$ з використанням адресних мінтермів. Результат моделювання даного мультиплексора наведено на рис.2.7.

```
SUBDESIGN DMX_1_4_T1
(D, A[1..0]  : INPUT;
      I[3..0]: OUTPUT;)
BEGIN
      I0=!A1&!A0&D;
      I1=!A1&A0&D;
      I2=A1&!A0&D;
      I3=A1&A0&D;
END;
```

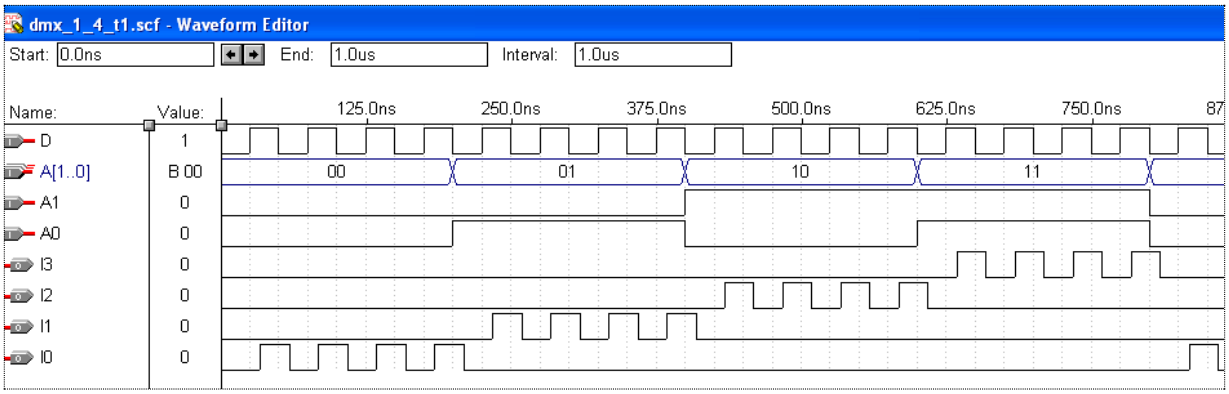


Рисунок 2.7 — Результати моделювання демультиплексора $1 \rightarrow 4$ з використанням адресних мінтермів

2.3. Мультиплексор із внутрішнім дешифратором, реалізованим мовою AHDL

Приклад 9. Проект внутрішнього дешифратора 2→4 мовою AHDL

```
SUBDESIGN dc_2_4_tp
(A1,A0:INPUT;
F3,F2,F1,F0:OUTPUT;)
BEGIN
F0=!A1&!A0;
F1=!A1&A0;
F2=A1&!A0;
F3=A1&A0;
END;
```

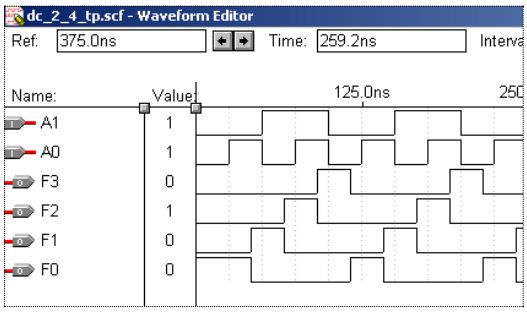


Рисунок 2.8 — Результати моделювання внутрішнього дешифратора 2→4

Приклад 10. Проект внутрішнього дешифратора 2→4 на основі символу dc_2_4_tp та результати моделювання наведено на рис.2.9.

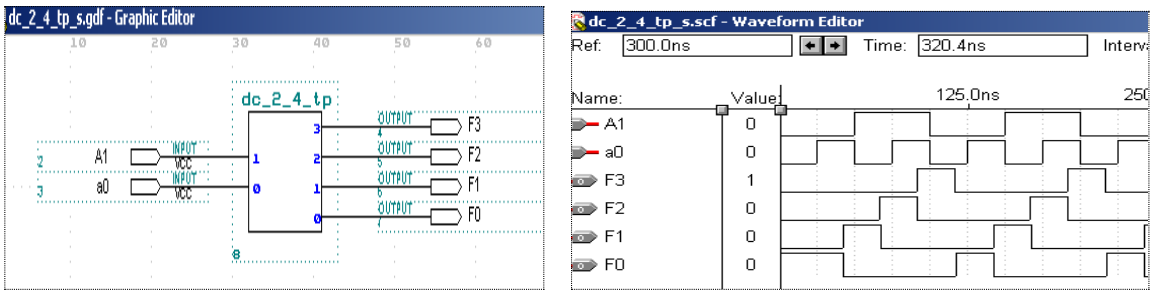


Рисунок 2.9 — Проект внутрішнього дешифратора 2→4 на основі символу dc_2_4_tp та результати моделювання

Приклад 11. Проект мультиплексора 4→1 з використанням внутрішнього дешифратора наведено на рис. 2.10, а результати моделювання представлено на рис. 2.11.

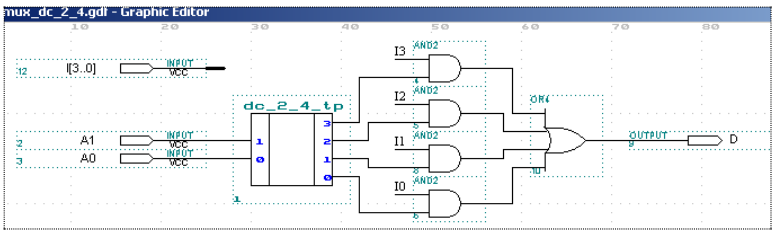


Рисунок 2.10 — Проект мультиплексора 4→1 з використанням внутрішнього дешифратора

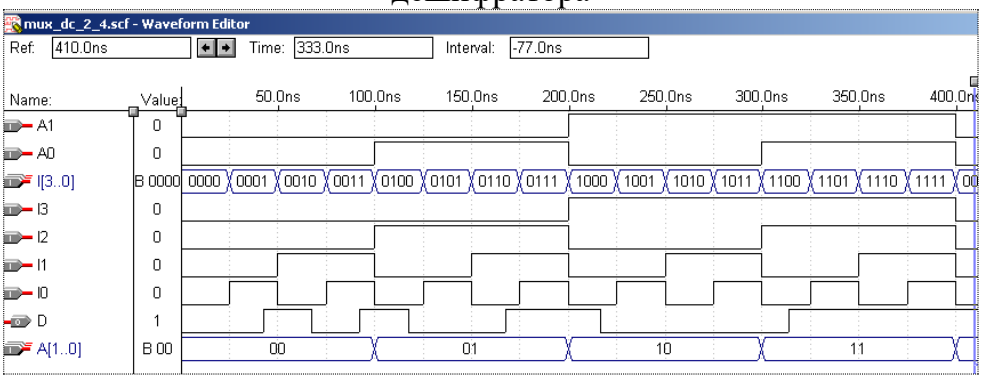


Рисунок 2.11 — Результати моделювання мультиплексора 4→1 з використанням внутрішнього дешифратора

Приклад 12. Проект мультиплексора 4→1 з використанням символу dc_2_4_tp (внутрішнього дешифратора) і символу (i_f_t) текстового проекту I_F_t наведено на рис. 2.12

```
SUBDESIGN I_F_t
(I3,I2,I1,I0:INPUT;
F3,F2,F1,F0:INPUT;
D:OUTPUT;)
BEGIN
D=I3&F3#I2&F2#I1&F1#I0&F0;
END;
```

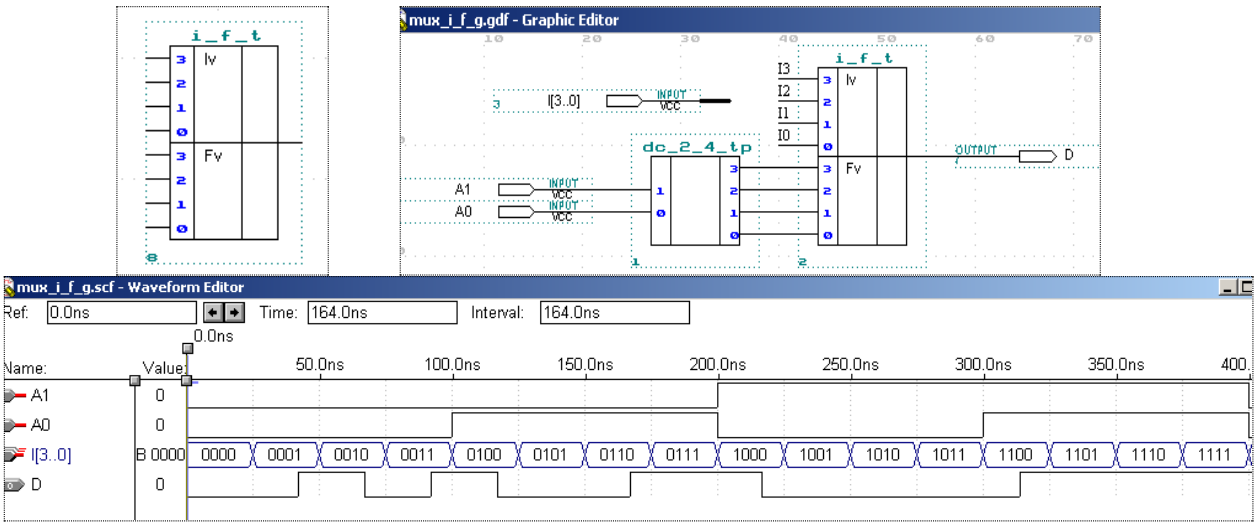


Рисунок 2.12 — Проект мультиплексора 4→1 з використанням символу dc_2_4_tp (внутрішнього дешифратора) і символу (i_f_t) текстового проекту I_F_t

Завдання для самостійної роботи

Приклад 1. Нижче наведений файл dmx_par_k є прикладом параметризованого опису шинного демультиплексора, що має $N+1$інформаційний вхід Inf[N..0], трирозрядний одноадресний вхід Adr[2..0], вхід дозволу En і вісім N-розрядних виходів (D[7..0][N..0]). Опис параметризованого шинного мультиплексора виконано з використанням оператора CASE [1].

У наведеному описі використана двовимірна група D[7..0][N..0], у якій параметризований один діапазон зміни індексів (розрядність шини, що комутується). Результат моделювання даного мультиплексора наведено на рис. 2.13.

```
PARAMETERS  (N=7);

ASSERT  (N != 1)
REPORT "Value of parameter N = %" N
SEVERITY INFO;

ASSERT  (N > 0)
REPORT "Value of parameter N must be greate then  %" N
SEVERITY ERROR;
SUBDESIGN dmx_par_k
(Inf[N..0], Adr[2..0], En: INPUT;
        D[7..0][N..0]: OUTPUT;)
BEGIN
        IF En THEN
                CASE Adr[] IS
                WHEN 0 => D[0][N..0]=Inf[N..0];
                WHEN 1 => D[1][N..0]=Inf[N..0];
                WHEN 2 => D[2][N..0]=Inf[N..0];
                WHEN 3 => D[3][N..0]=Inf[N..0];
                WHEN 4 => D[4][N..0]=Inf[N..0];
                WHEN 5 => D[5][N..0]=Inf[N..0];
                WHEN 6 => D[6][N..0]=Inf[N..0];
                WHEN 7 => D[7][N..0]=Inf[N..0];
                END CASE;
                END IF;
         END;
```

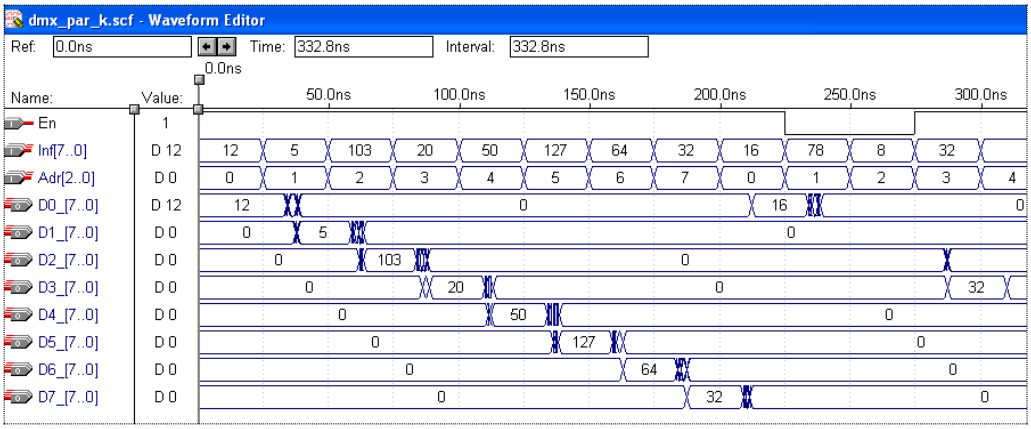


Рисунок 2.13 — Результати моделювання параметризованого шинного мультиплексора з використанням оператора CASE

Приклад 2. У нижче наведеному файлі виконано опис параметризованого шинного мультиплексора з використанням оператора FOR k IN 0 TO 7 GENERATE. Результати моделювання наведено на рис.2.14.

```
PARAMETERS  (N=7);

ASSERT  (N != 1)
REPORT "Value of parameter N = %" N
SEVERITY INFO;

ASSERT  (N > 0)
REPORT "Value of parameter N must be greate then  %" N
SEVERITY ERROR;

SUBDESIGN dmx_par_
(       Inf[N..0], Adr[2..0], En: INPUT;
        D[7..0][N..0]: OUTPUT;)
BEGIN
        IF En THEN
        FOR k IN 0 TO 7 GENERATE
                CASE Adr[] IS
                WHEN k => D[k][N..0]=Inf[N..0];
                END CASE;
                END GENERATE;
        END IF;
         END;
```

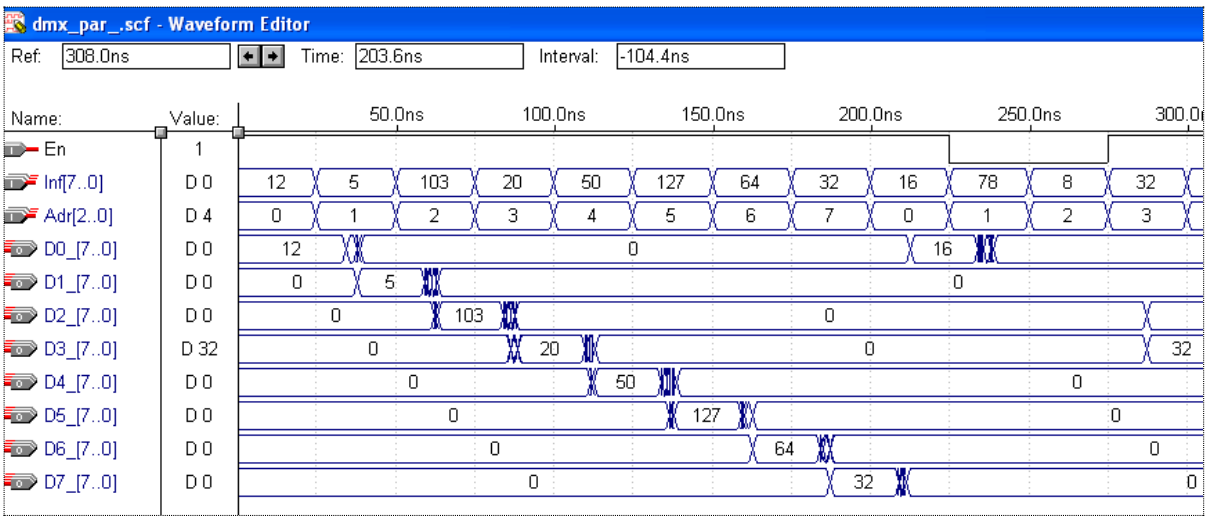


Рисунок 2.13 — Результати моделювання параметризованого шинного мультиплексора з використанням оператора FOR k IN 0 TO 7 GENERATE.

Звернемо увагу на те, що два описи параметризованого демультиплексора з використанням оператора циклу FOR k IN 0 TO 7 GENERATE (файл dmx_par_k) і оператора CASE (файл dmx_par_) дають однаковий результат моделювання. Однак, використання оператора циклу дало можливість спростити код програми.

Лабораторна робота № 8

Тема: Проектування цифрових пристроїв з пам'яттю

Мета роботи: Набуття практичних навичок проектування цифрових пристроїв з пам'яттю з використанням пакету MAX+PLUS II та за допомогою мови опису апаратури AHDL.

1. Опис тригерів

Розглянемо опис синхронних D-тригерів зі входом дозволу DFFE та без нього DFF. На рис. 1 наведено бібліотечне зображення обох варіантів такого тригера.

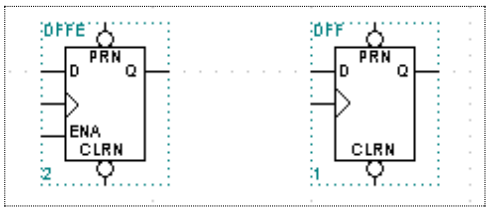


Рисунок 1.1 — Бібліотечне зображення обох варіантів D-тригера

Призначення виводів наступне:

D — інформаційний вхід;

CLK — тактовий вхід (на рисунку від показаний трикутником). З приходом додатного (високого) фронту, сигнал з інформаційного входу передається на вихід тригера;

Q — вихід тригера;

ENA — вхід дозволу. За замовчуванням базове значення цього входу — одиничне, тому, якщо сигнал на цей вхід не подається, то робота тригера дозволена;

PRN — вхід асинхронної установки тригера. Керується низьким рівнем сигналу.

CLRN — вхід асинхронного скидання тригера. Керується низьким рівнем сигналу.

Входи асинхронної установки мають більш високий пріоритет, ніж вхід сигналу дозволу.

Наведені імена виводів є вбудованими і використовуються в текстових описах тригерних схем. При текстовому опису тригера в розділі VARIABLE повинна бути оголошена тригерна змінна. Через цю змінну здійснюється доступ до вбудованих імен тригерів. Наведемо приклад опису тригерної змінної:

```
SUBDESIGN d_triger
(clk, load, d: INPUT;
            q: OUTPUT;)
VAIABLE
            ff: DFFE;
BEGIN
```

```
        ff.clk=clk;
        ff.ena=load;
          ff.d=d;
           q   =ff.q;
END;
```

Тригерна змінна ff тут оголошена за допомогою ключового слова DFFE. Тепер з цією змінною зв'язується D-тригер з описаними вище виводами. Доступ до виводів виконується через знак присвоєння, який прирівнює змінні зовнішніх портів із внутрішніми виводами тригера, які записуються через крапку, яка з'єднує ім'я тригера і його внутрішній вивід.

Порівняємо роботу D-тригера, описаного мовою AHDL та бібліотечного тригера DFFE.

Нижче наведена текстовий опис D-тригера (файл D_triger_t) та його символ (рис. 2)

```
SUBDESIGN D_triger_t
(
clk, prn, clrn, enable, d: INPUT;
        Q: OUTPUT;
)
VARIABLE
    ff: DFFE;
BEGIN
        ff.clk=clk;
        ff.prn=prn;
        ff.clrn=clrn;
        ff.ena=enable;
        ff.d=d;
        Q=ff.q;
END;
```

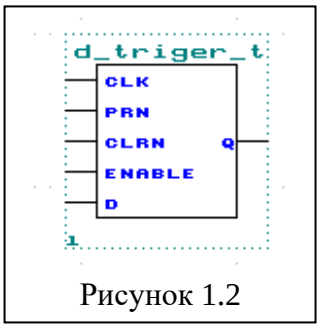


Рисунок 1.2

Для порівняння роботи розгядуваних компонентів створено графічний файл D_triger_g (рис. 1.3) та проведено симуляцію його роботи (рис. 1.4).

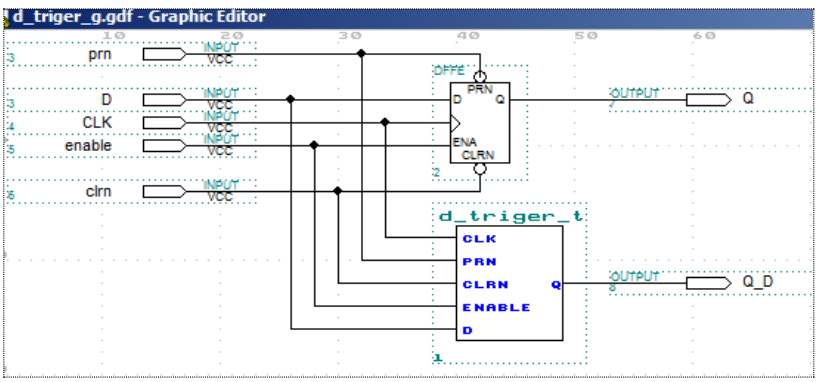


Рисунок 1.3 — Графічний файл D_triger_g

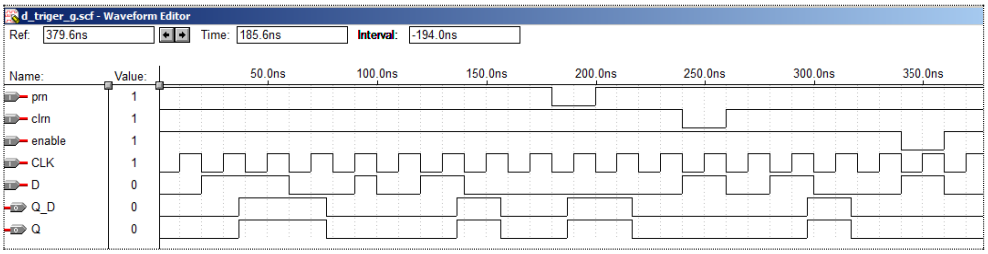


Рисунок 1.4 — Результати моделювання графічного файлу D_triger_g

За результатами моделювання можна прослідкувати роботу побудованого D-тригер (D_triger_t) і бібліотечного тригера DFFE. Результати роботи розглядуваних тригерів повністю співпадають.

Крім D-тригера, є вбудовані описи T, JK та SR тригерів. Ці тригери оголошуються наступними ключовими словами (*примітивами*): TFFE, JKFFE, RSFFE (TFF, JKFF, RSFF). У цих тригерах замість входу D використовуються інші імена. Це відповідно: T; J,K; S, R.. Крім цих тригерів є ще функція тригера-засувки з іменем примітива LATCH.

Зображення даних тригерів наведено на рис.1.5.

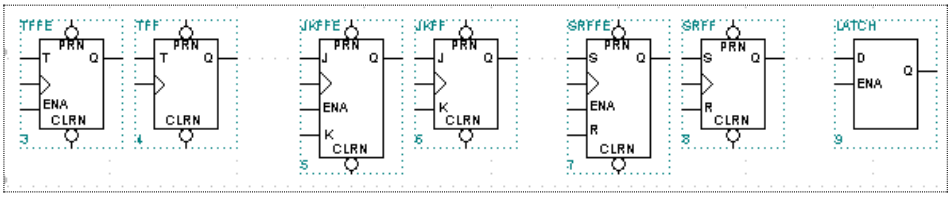


Рисунок 1.5 — Зображення тригерів

Індивідуальне завдання 1. Скласти проект, за допомогою якого порівняти роботу одного із бібліотечних тригерів, наведених на рис.1.5 і відповідним текстовим описом. Перевірку роботи перевірити шляхом моделювання.

2. Послідовнісна логіка в AHDL

Логічна схема називається послідовнісною, якщо виходи в заданий момент часу є функцією входів не тільки в даний момент, але й в усі попередні моменти часу. Таким чином, у послідовнісну схему повинні входити деякі елементи пам'яті (тригери). У мові AHDL послідовнісна логіка реалізована цифровими автоматами з пам'яттю (state machines), регістрами й тригерами. При цьому засобу опису цифрових автоматів займають особливе місце. Крім того, до послідовнісних логічних схем відносяться різні лічильники й контролери.

2.1. Опис тригерних схем

Тригерна (послідовнісна) схема — схема з елементом пам’яті (тригером), який синхронізується фронтом (синхронний тригер) або рівнем тактового сигналу (тригер-засувка).

2.1.1. Регістри

Регістр — пристрій призначений для прийому, зберігання, передачі та перетворення форм подання даних (паралельна, послідовна).

За способами прийому і передачі даних розрізняють:

Паралельний регістр — регістр з паралельним завантаженням і видачею даних;

Зсувний регістр — регістр з послідовним завантаженням і видачею даних;

Зсувний регістр з паралельним завантаженням даних;

Зсувний регіт з паралельним виводом даних.

Паралельний регістр

З метою кращої уяви структури паралельного регістра паралельно розглянемо структурну схему чотирирозрядного паралельного регістра (рис.2.1), виконану засобами системи проектування MAX+plus II.

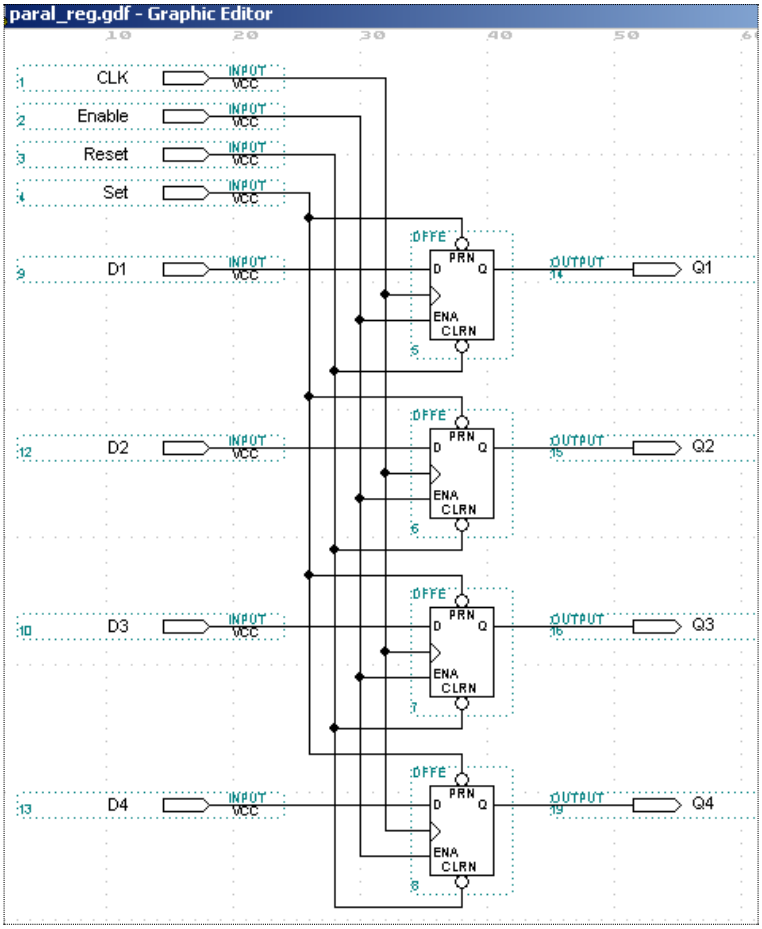


Рисунок 2.1 — Структурна схема чотирирозрядного паралельного регістра

У наведеного регістра $D[4..1]$, $Q[4..1]$, відповідно, чотирирозрядний вхід і вихід даних. Інші виводи описані вище.

Опис відповідного регістра, виконаного мовою AHDL, наведено в проекті RG_4

```
SUBDESIGN RG_4
(
D[4..1],CLK, Set,Rese:INPUT=GND;
Enable:INPUT=VCC;
Q[4..1]:OUTPUT;
)
VARIABLE
  FF[4..1]:DFFE;
BEGIN
            FF[].clk      =CLK;
            FF[].(clrn, prn)=(Reset, Set);
            FF[].ena      =Enable;
            FF[].d        =D[];
             Q[]          =FF[].q;
END;
```

Як видно з файлу, регістр оголошений у секції VARIABLE як D-Тригер з дозволом (DFFE). У першому булевому рівнянні в логічній секції відбувається під'єднання входу тактового сигналу CLK (початковий стан GND) до портів тактового сигналу clk тригерів ff[4..1].

У другому рівнянні входи скидання (clrn) і установки (prn) з'єднуються із входами Reset і Set, початковий стан яких GND. У третьому рівнянні вхід дозволу ena з'єднується із входом Enable, початковий стан якого VCC. У четвертім рівнянні входи даних проекту з'єднуються з портами даних тригерів ff[7..0]. У п'ятому рівнянні виходи проекту з'єднуються з виходами тригерів. Всі п'ять рівняння оцінюються одночасно. Зауважимо, що кілкабітну тригерну змінна ff[] називають *регістровою змінною*.

Результати моделювання схемного проекту та проекту, описаного мовою AHDL наведено на рис. 2.2 та рис.2.3 відповідно.

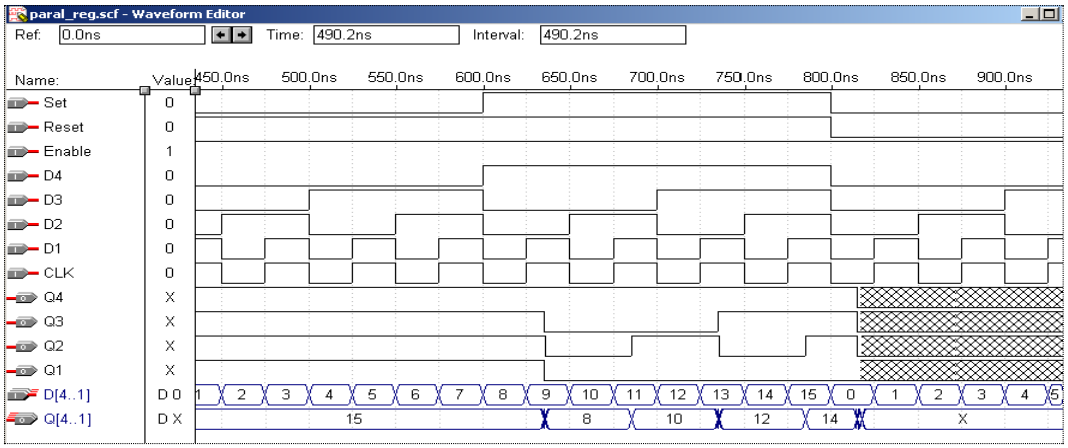


Рисунок 2.2 — Результати моделювання схемного проекту

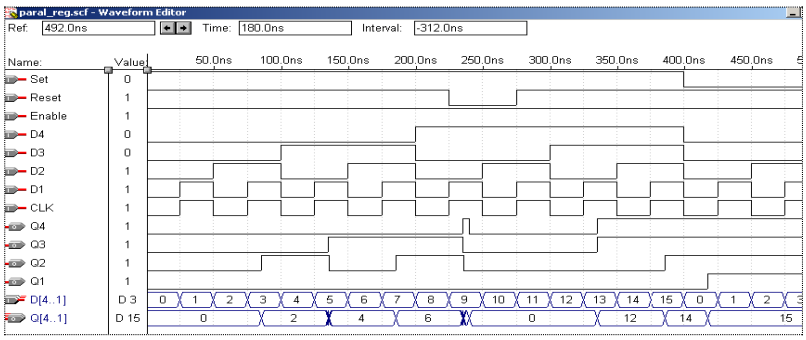


Рисунок 2.3 — Результати моделювання проекту, описаного мовою AHDL

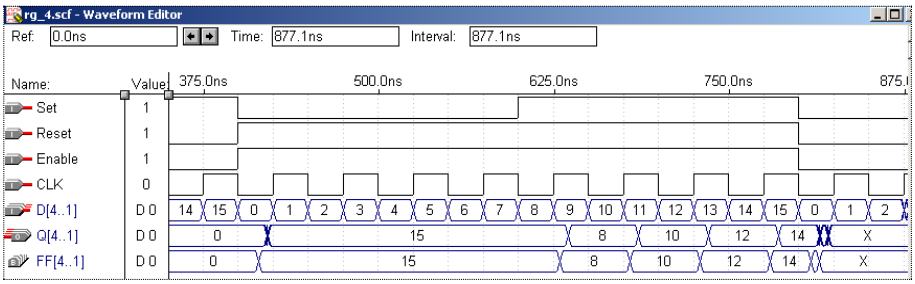


Рисунок 2.4 — Результати моделювання проекту

Звернемо увагу на те, що у випадку коли на входи Set і Reset подані одночасно нулі, то регістри не працюють. На рис.2.2 це відображено сіточкою і символом “x”, а на рис.2.4 символом “x”.

Аналогічним чином можна реалізувати регістри паралельного завантаження за допомогою примітиву DFFE, але без підключення входів PRN і CLRN (рис. 2.5 та рис. 2.6)

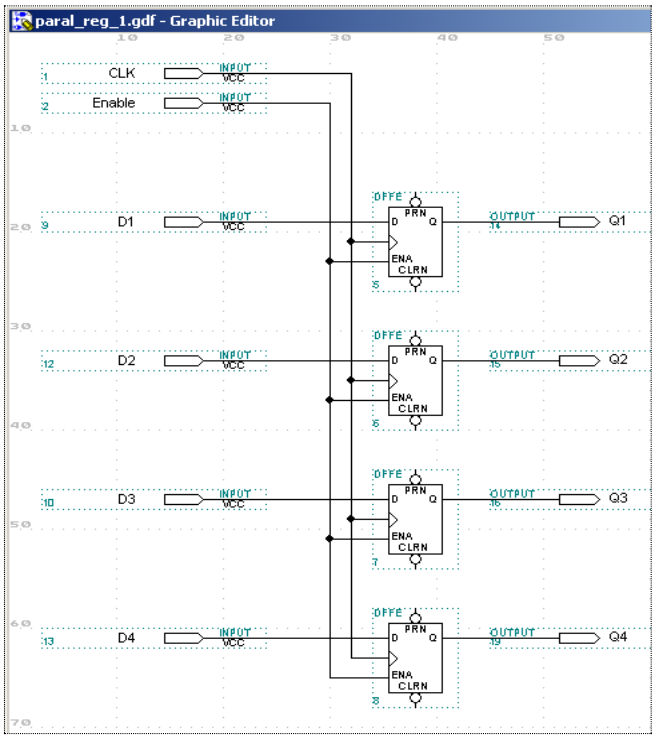


Рисунок 2.5 — Реалізація регістрів паралельного завантаження за допомогою примітиву DFFE без підключення входів PRN і CLRN

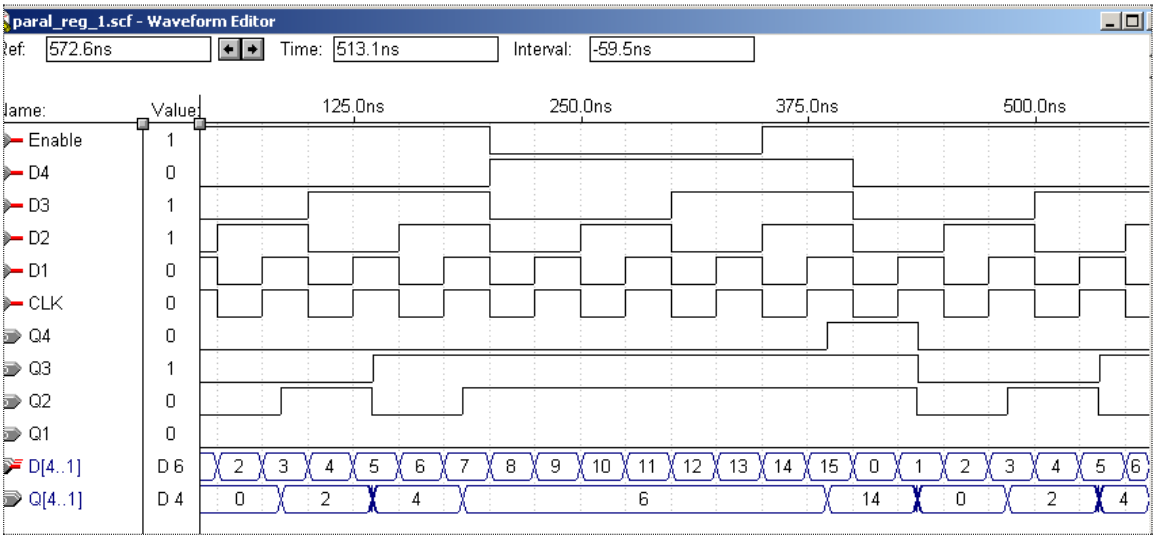


Рисунок 2.6 — Результати моделювання регістрів паралельного завантаження за допомогою примітиву DFFE без підключення входів PRN і CLRN

Нижче наведено проект аналогічний регістру паралельного завантаження, описаний мовою опису апаратури AHDL. Результати моделювання роботи такого регістра наведено на рис. 2.7.

```
SUBDESIGN RG_4_1
(
D[4..1], CLK: INPUT;
Enable: INPUT=VCC;
Q[4..1]: OUTPUT;
)
VARIABLE
  FF[4..1]:DFFE;
BEGIN
            FF[].clk      =CLK;
            FF[].Ena     =Enable;
            FF[].d        =D[];
             Q[]          =FF[].q;
END;
```

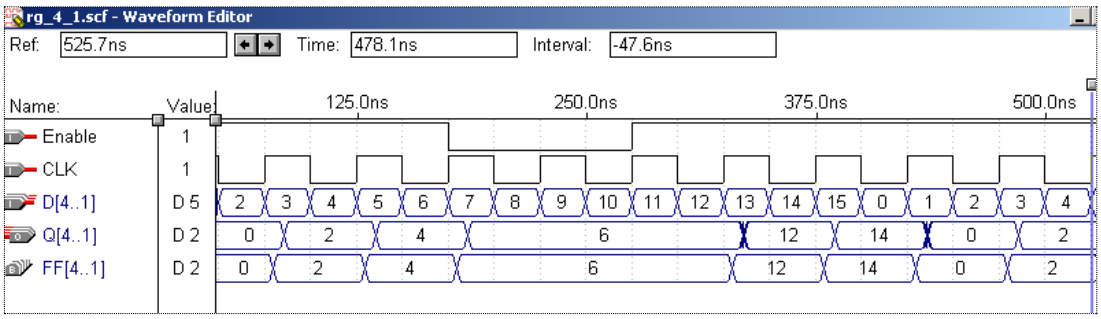


Рисунок 2.7 — Результати моделювання регістру паралельного завантаження, описаного мовою опису апаратури AHDL

Індивідуальне завдання 2. Скласти проект регістра з паралельним завантаженням, для чого використати один із бібліотечних тригерів, наведених на рис.1.5 і відповідним текстовим описом та порівняти їх роботу шляхом моделювання.

Тригери з регістровими виходами

У попередніх прикладах тригерну змінну, наприклад, ff[4..1]: DFFE, яка описана в розділі VARIABLE, як уже вказувалось, називають ***регістровою змінною***.

Разом з тим, можна оголосити регістрові виходи, якщо оголосити вихідні порти підпроекту як D-тригери в розділі VARIABLE.

Наведений нижче файл reg_out.tdf забезпечує ті ж функції, що й попередній (RG_4_1.tdf), але має регістрові виходи.

```
SUBDESIGN reg_out
(
 clk, load, D[4..1]: INPUT;
 Q[4..1]: OUTPUT;
)
VARIABLE
    q[4..1]: DFFE;
BEGIN
    q[].clk = clk;
    q[].ena = load;
    q[] = d[];
END;
```

Значення з D входів направляється в регістр. Виходи з регістра не до тих пір, поки не появиться додатній фронт сигналу CLK. Звернення до виходів Q тригерів відбувається без використання порту Q. Результати моделювання роботи такого тригера наведено на рис. 2.8.

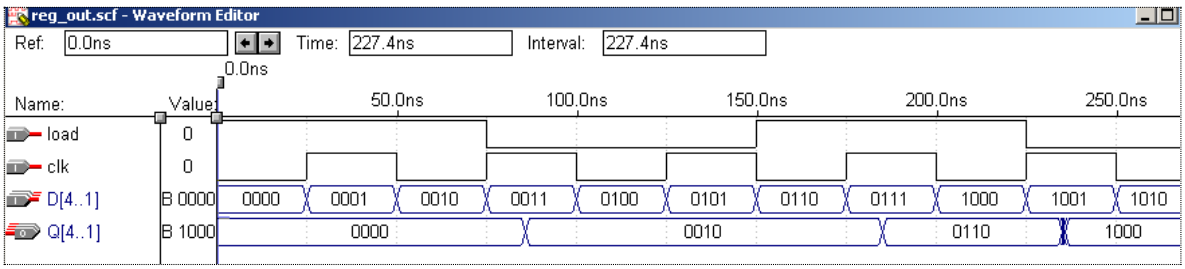


Рисунок 2.8 — Результати моделювання Q тригера без використання порту Q

Параметризований N-розрядний паралельний регістр

Нижче наведено параметризований опис N-розрядний паралельний регістр зі входами асинхронної установки (Set) асинхронного скидання (Reset) і дозволу завантаження даних (Enable). Моделювання роботи такого регістра наведено на рис. 2.9.

```
PARAMETERS (N=8);
SUBDESIGN rg _par
( Din[N..1], Set, Reset: INPUT=GND;
CLK:  INPUT;
Enable: INPUT=VCC;
Q[N..1]: OUTPUT;)
VARIABLE
    FF[N..1]: DFFE;
BEGIN
    FF[].clk = clk;
    FF[].ena = Enable;
    FF[].d=Din[];
    FF[].prn=Set;
    FF[].clrn=Reset;
    Q[] =FF[].q;
END;
```

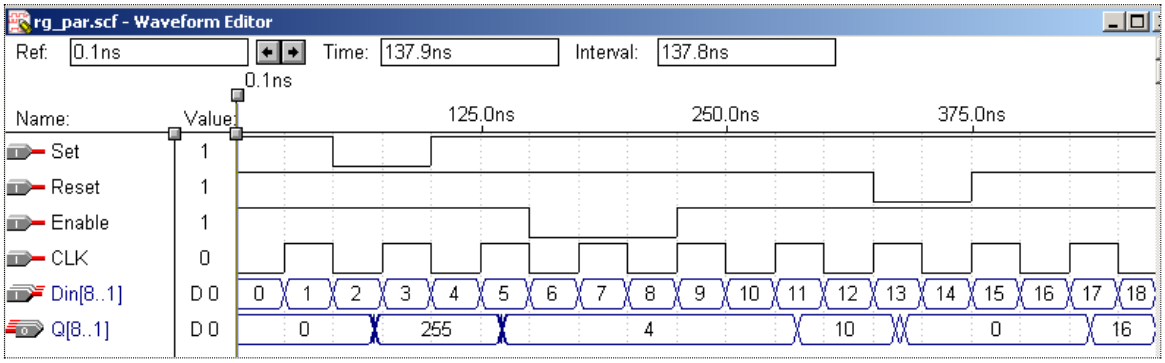


Рисунок 2.9 — Результати моделювання параметризованого N-розрядного паралельного регістру

Індивідуальне завдання 3. Скласти проект параметризованого N-розрядного паралельного регістр з регістровими виходами, для чого використати один із бібліотечних тригерів, наведених на рис.5 і відповідним текстовим описом та порівняти їх роботу шляхом моделювання.

Зсувний регістр

Нижче наведено параметризований опис N-розрядного зсувного регістра зі входами асинхронної установки (Set) асинхронного скидання (Reset) і дозволу завантаження даних (Enable). Моделювання роботи такого регістра наведено на рис. 2.10.

```
PARAMETERS (N=8);
ASSERT (N>0)
        REPORT "Value of WIDTH parameter must be greater then %" N
Severity Error;
SUBDESIGN rg_sh
( Din, Set, Reset: INPUT=GND;
CLK:  INPUT;
Enable: INPUT=VCC;
Q: OUTPUT;)
VARIABLE
    FF[N..1]: DFFE;
BEGIN
    FF[].clk = clk;
    FF[].ena = Enable;
    FF[].prn=Set;
    FF[].clrn=Reset;
        FF[].d=(FF[N-1..1].q,Din);
    Q =FF[N].q;
END;
```

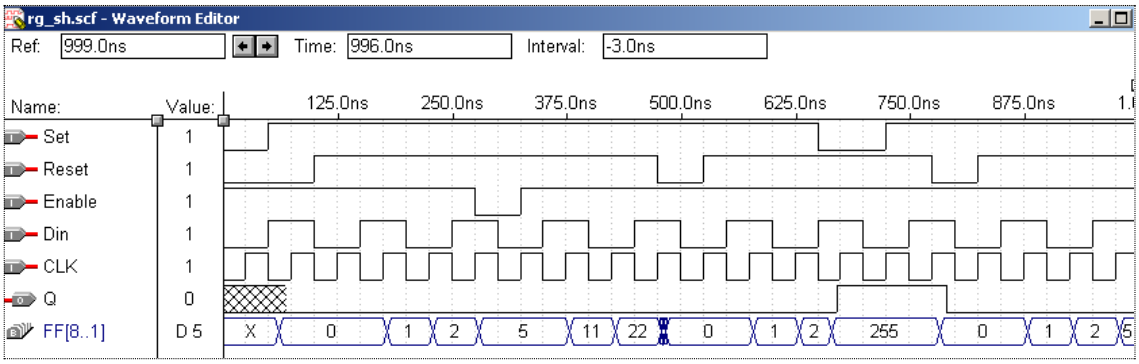


Рисунок 2.10 — Результати моделювання параметризованого N-розрядного зсувного регістра

Таблиця 2.1 — Таблиця станів розрядів регістра зсуву

7	6	5	4	3	2	1	0	←	Розряди регістра
0	0	0	0	0	0	0	0	0	Установка в 0
0	0	0	0	0	0	0	1	1	Зсув і уведеня 1
0	0	0	0	0	0	1	0	2	Зсув
0	0	0	0	0	1	0	1	5	Зсув і уведеня 1
0	0	0	0	0	1	0	1	5	Заборона зсуву
0	0	0	0	1	0	1	1	11	Зсув і уведеня 1
0	0	0	1	0	1	1	0	22	Зсув
0	0	0	0	0	0	0	0	0	Установка в 0
0	0	0	0	0	0	0	1	1	Зсув і уведеня 1
0	0	0	0	0	0	1	0	2	Зсув
1	1	1	1	1	1	1	1	255	Установка в 1

Застосування регістра зсуву даних. Нехай нам потрібно перемножити числа:

$$X = x_7x_6x_5x_4x_3x_2x_1x_0,\ Y = y_7y_6y_5y_4y_3y_2y_1y_0$$

Потрібно знайти добуток $Z = X \cdot Y$ шляхом розбиття чисел X і Y на два доданки по чотири розряди кожний:

$$X = x_7x_6x_5x_4 \cdot 10^4 + x_3x_2x_1x_0 = X_s \cdot 10^4 + X_m,$$

$$Y = y_7y_6y_5y_4 \cdot 10^4 + y_3y_2y_1y_0 = Y_s \cdot 10^4 + Y_m.$$

Тоді можна записати: $Z = X_s \cdot Y_s \cdot 10^8 + X_s \cdot Y_m \cdot 10^4 + X_m \cdot Y_s \cdot 10^4 + X_m \cdot Y_m$.

Таким чином, для одержання добутку нам потрібно перемножувач 4x4 розряди, 16-ти розрядний суматор та регістр зсуву для передачі даних:

без зсуву — $\mathrm{Re}\,z[15..8] = 0$, $\mathrm{Re}\,z[7..0] = X_m \cdot Y_m$;

зсув даних вліво на чотири розряди: $\mathrm{Re}\,z[15..12] = 0$, $\mathrm{Re}\,z[11..4] = X_s \cdot Y_m$, або $\mathrm{Re}\,z[11..4] = X_m \cdot Y_s$, $\mathrm{Re}\,z[3..0] = 0$;

зсув даних вліво на вісім розрядів: $\mathrm{Re}\,z[15..8] = X_s \cdot Y_s$, $\mathrm{Re}\,z[7..0] = 0$.

Оскілки при множенні маємо три випадки зсуву на нуль, чотири і вісім розрядів, то нам потрібно змінну, яка може приймати не менше як три різні значення, наприклад, змінна *cnt*[1..0] та одну змінну, яка може приймати результат множення складових, наприклад, змінна *In*[7..0]. Такий регістр, мовою опису апаратури AHDL, має вигляд:

```
SUBDESIGN shift_8r
(in[7..0]:INPUT;
cnt[1..0]:INPUT;
result[15..0]:OUTPUT;)
BEGIN
CASE cnt[] IS
        WHEN 1 => result[15..12]=0;
                  result[11..4]=in[];
                  result[3..0]=0;
```

```
        WHEN 2 => result[15..8]=in[];
                  result[7..0]=0;
        WHEN others => result[15..8]=0;
                  result[7..0]=in[];
END CASE;
END;
```

Результати моделювання наведено на рис. 2.11.

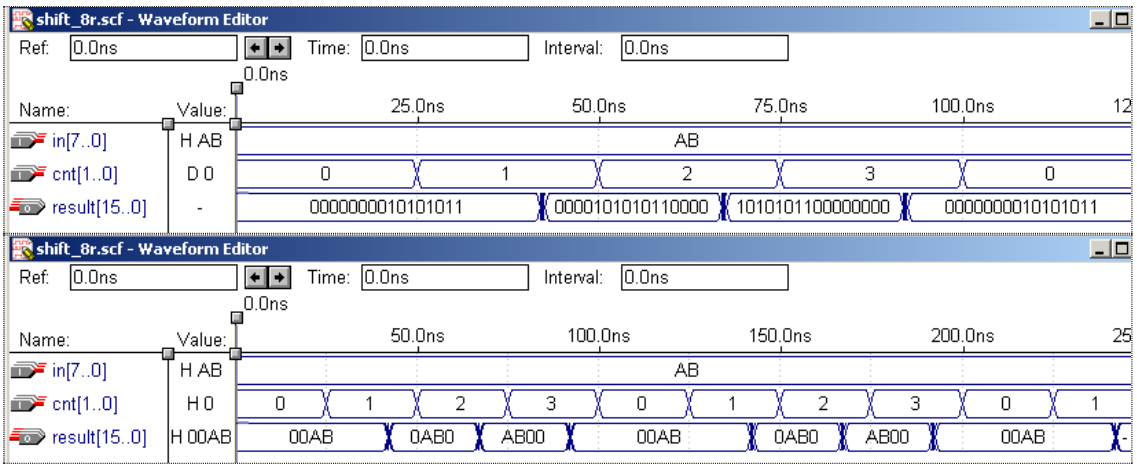


Рисунок 2.11 — Результати моделювання зсувного регістру, описаного мовою опису апаратури AHDL

Індивідуальне завдання 4. Для перевірки правильності роботи проекту shift_8r використати комбінацію із двох шістнадцяткових цифр набору: A,B,C,D,E,F (крім AB) у двійковій і шістнадцятковій формі.

Лабораторна робота № 9

Тема: Опис лічильників мовою AHDL

Мета: Набути знання та навички в проектуванні лічильників з використанням програмного комплексу MAX+PLUS II та мови опису апаратури AHDL.

Лічильник – послідовнісний пристрій, призначений для реєстрації числа імпульсів, що надійшли на його тактовий вхід і реєстрації їх числа в якому-небудь двійковому коді. Мовою AHDL лічильник може бути описаний:

- на поведінковому рівні. При цьому використовуються високорівневі конструкції мови, а також арифметичні оператори додавання та віднімання;
- у вигляді кінцевого автомата;
- на вентильному рівні.

При будь-якому описі, для побудови лічильника потрібний елемент пам'яті – тригерний елемент.

Тригер – це елемент, який може зберігати поданий на нього логічний рівень після зняття вхідного сигналу.

Тригером типу **D** називається синхронний запам'ятовуючий елемент з двома стійкими станами і одним інформаційним **D**-входом.

Закон функціонування **D –** тригера описується логічним рівнянням:

$$Q_{t+1} = C_t D_t ,$$

де C_t - тактовий сигнал, а D_t - вхідний сигнал в момент часу t.

DFF - синхронний **D**-тригер (рис. 1) має наступні виводи:

D – синхронний вхід даних;

Clk (позначається трикутником) **–** вхід тактового сигналу;

PRN – асинхронна установка тригера (в логічну 1);

CLRN – асинхронне скидання тригера (в логічний 0);

Q – вихід.

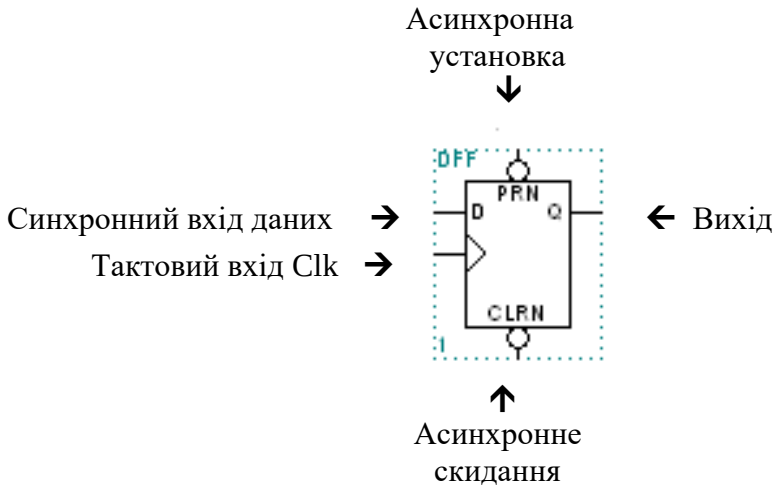


Рисунок 1.1 — **DFF** - синхронний **D**-тригер

1. ПОВЕДІНКОВИЙ ОПИС

1.1. Двійковий підсумовуючий лічильник

Опис на поведінковому рівні - це найпростіший і найбільш загальний спосіб задання алгоритму роботи лічильника. Проілюструємо його на прикладах.

Розглянемо параметризований опис двійкового WIDTH-розрядного підсумовуючого лічильника.

Зауваження. В подальшому замість слова WIDTH (ширина) можна вживати латинську букву N.

Виводи лічильника:

- **Clk** — тактовий вхід;
- **Set** — вхід асинхронної установки лічильника в одиницю;
- **Reset** — вхід асинхронного скидання лічильника в нуль;
- **Dout[WIDTH..1]** — WIDTH-розрядний вихід лічильника.

Поведінковий опис підсумовуючого (increment - збільшувати) лічильника має вигляд.

```
PARAMETERS (WIDTH=7);
        ASSERT (WIDTH>0)
                REPORT "Значення параметра WIDNH повинне бути більше %
" WIDTH
                SEVERITY ERROR;
SUBDESIGN COUNT_1
(
CLK, RESET, SET : INPUT = VCC;
Q[WIDTH..0]       :       OUTPUT;
)
VARIABLE
        RR[WIDTH..0] : DFF;
BEGIN
        RR[].(CLK, CLRN, PRN) = (CLK, RESET,SET);
        RR[].D = RR[].Q+1;
        Q[] = RR[].Q;
END;
```

У програмі для перевірки значення параметра WIDTH використаний оператор ASSERT (WIDTH>0), що має рівень строгості **ERROR**. Тому при невиконанні заданої умови компілятор видаватиме повідомлення про помилку.

При описі змінних використано визначення їхніх значень за замовчуванням. Якщо при подальшому використанні значення змінних CLK, RESET, SET не задані, то їм автоматично присвоюється значення логічної одиниці.

Відповідно до наведеного опису, компілятор пакета синтезує WIDTH-розрядний накопичуючий суматор, на інформаційний вхід якого надходить константна одиниця. Структурна схема, що відповідає даному пристрою, показана на рис. 1.2.

З появою на вході Clk фронту тактового сигналу, до поточного значення накопичувального суматора додається одиниця. Підсумовування одиниці із числом 2^{WIDTH}-1 (у всіх WIDTH розрядах накопичувального суматора логічні одиниці) приведе до переповнення суматора, що приводить до появи у всіх його розрядах сигналу логічного нуля.

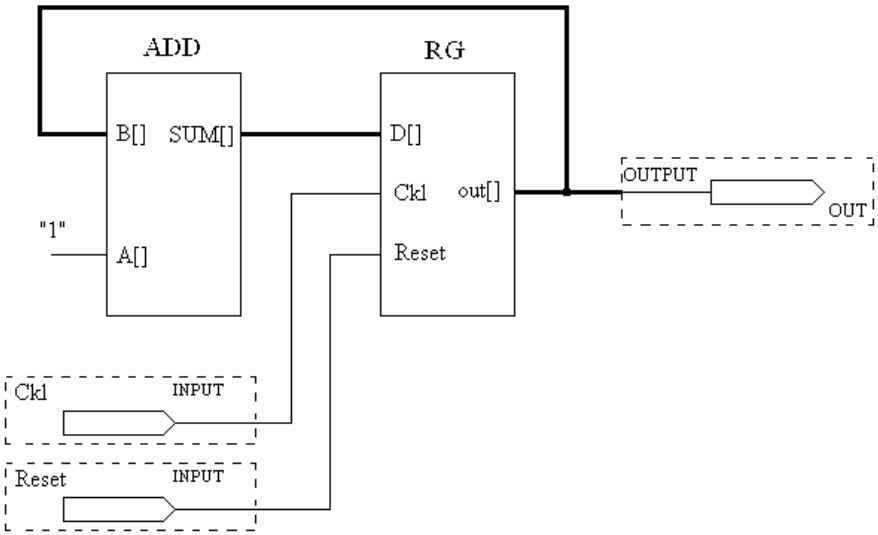


Рисунок 1.2 — Структурна схема, що відповідає поведінковому опису підсумовуючого лічильника

Результати моделювання лічильника COUNT_1, при WIDTH=7, наведені на рис. 1.3. На наведених часових діаграмах вихідний сигнал лічильника показаний двічі: перший раз у вигляді числа Q[7..1], представленого в десятковій системі числення; другий раз – у вигляді значення регістрової змінної RR[7..1], що не є обов'язковим.

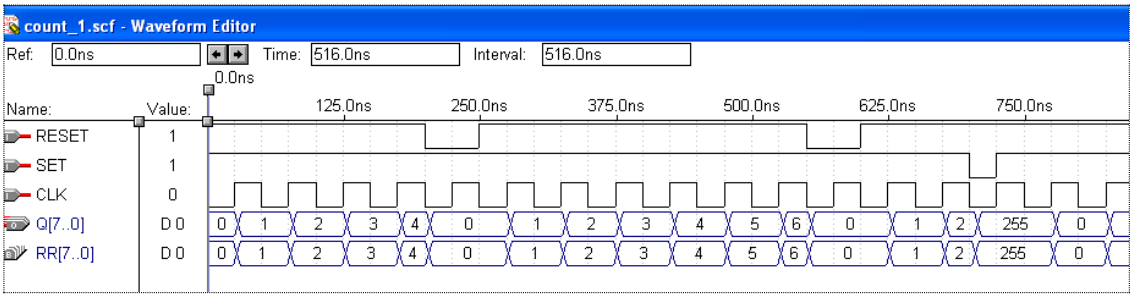


Рисунок 1.3 — Результати моделювання лічильника COUNT_1, при WIDTH=7

1.2. Двійковий віднімаючий лічильник

За аналогією можна описати й параметризований WIDTH-розрядний віднімаючий (decrement - зменшення) лічильник. Цей лічильник має ті ж самі виводи, що і попередній.

```
PARAMETERS (WIDTH=7);
        ASSERT (WIDTH>0)
                REPORT "Значення параметра WIDNH повинне бути більше % " WIDTH
                SEVERITY ERROR;
SUBDESIGN COUNT_2
(
CLK, RESET, SET             :       INPUT = VCC;
Q[WIDTH..0]                 :       OUTPUT;
)
VARIABLE
        RR[WIDTH..0]        : DFF;
BEGIN
        RR[].(CLK, CLRN, PRN) = (CLK, RESET, SET);
RR[].D = RR[].Q-1;
        Q[] = RR[].Q;
END;
```

Результати моделювання лічильника, що віднімає, наведені на рис. 1.4

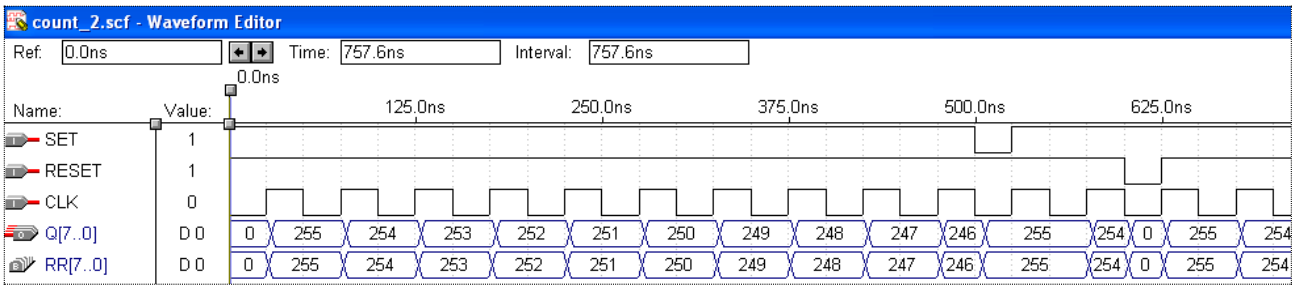


Рисунок 1.4 — Результати моделювання лічильника, що віднімає

1.3. Двійковий реверсний лічильник

Розглянемо опис параметризированого реверсного лічильника з додатково введеним входом дозволу роботи. Якщо вхідній змінній IN/DEC присвоєне значення одиниці – лічильник виконує операцію інкременту (підсумовуючий лічильник), у протилежному випадку (IN/DEC == 0) - лічильник виконує мікрооперацію декременту (віднімаючий лічильник).

```
PARAMETERS (WIDTH=7);
        ASSERT (WIDTH>0)
        REPORT "Значення параметра WIDNH повинно бути більше % " WIDTH
                SEVERITY ERROR;
```

```
SUBDESIGN COUNT_3
(
CLK, RESET, SET                 :       INPUT = VCC;
ENABLE, IN/DEC                  :       INPUT = VCC;
Q[WIDTH..0]                     :       OUTPUT;
)
VARIABLE
RR[WIDTH..0]        : DFFE;
BEGIN
        RR[].(CLK, CLRN, PRN, ENA) = (CLK, RESET, SET, ENABLE);
        IF IN/DEC==1 THEN
                        RR[].D = RR[].Q+1;
        ELSE
                        RR[].D = RR[].Q-1;
        END IF;
        Q[] = RR[].Q;
END;
```

Наведена програма фактично описує пристрій аналогічний, наведеному на рис.1.2. Відмінність полягає в заміні схеми додавання універсальним суматором, що залежно від величини керуючого сигналу виконує або операцію додавання (ADD), або віднімання (SUB).

Структурна схема такого пристрою наведена на рис.1.5. Відмітимо, що для реалізації можливості керування роботою пристрою, його регістр додатково обладнаний входом дозволу роботи (Enable), що вимагає використання тригера **DFFE**.

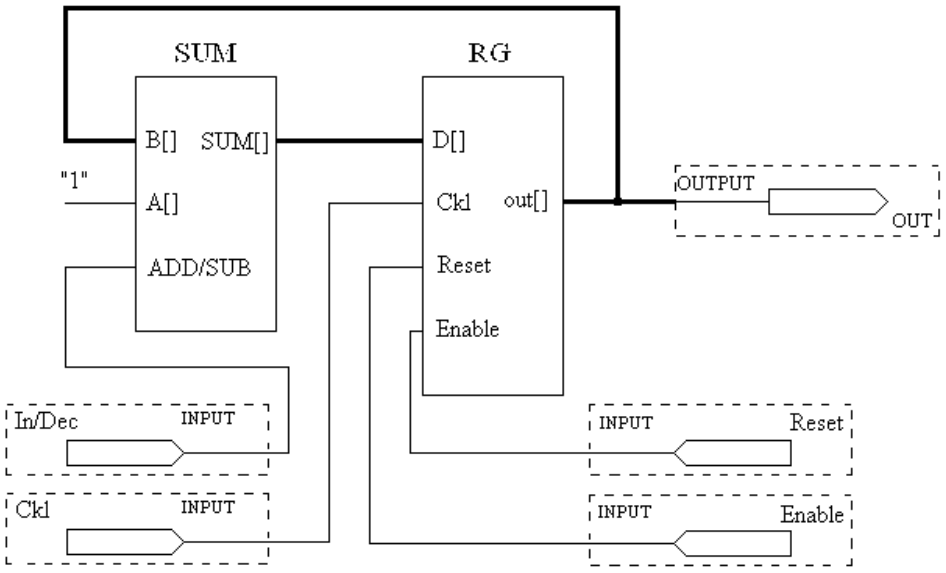


Рисунок 1.5 — Структурна схема параметризированого реверсного лічильника з додатково введеним входом дозволу роботи

Часові діаграми, що пояснюють роботу цього лічильника, наведено на рис.1.6. Із них слідує, що сигнали RESET і SET мають більш високий пріоритет, ніж сигнал ENABLE. Подача активного логічного рівня (сигнал

нуля) на вхід ENABLE приводить до зупинки роботи лічильника, і він переходить у режим зберігання раніше записаної інформації. Цікаво відзначити, що, оскільки сигнал CLK є, як правило, глобальним, а сигнал ENABLE формується логічним елементом макроосередку (логічного блоку) ПЛІС, то при збігу фронтів зміни сигналів CLK і ENABLE (див. рис.6) тригер встигає перемкнутися до появи на його інформаційному вході сигналу ENABLE.

Аналогічне зауваження справедливо й для сигналу IN/DEC (див. рис.1.6).

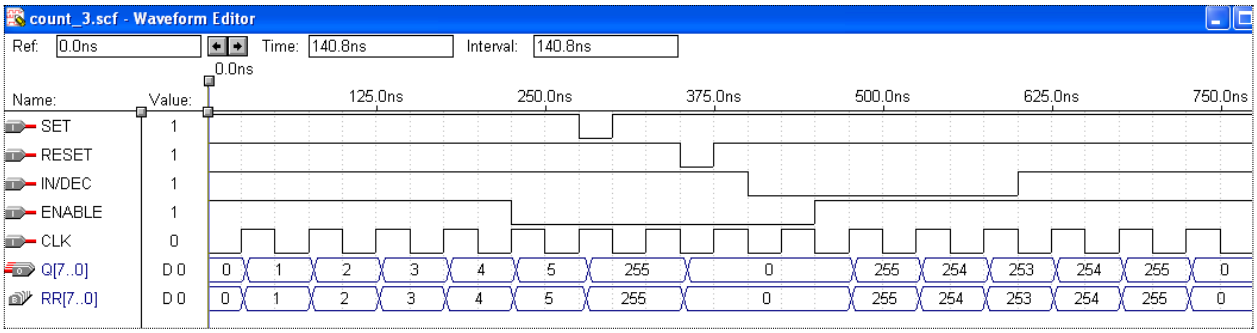


Рисунок 1.6 — Результати моделювання параметризированого реверсного лічильника

1.4. Реверсний лічильник з паралельним завантаженням

Розглянемо опис параметризированого, реверсного лічильника, що використовує режим паралельного завантаження інформації. Для цього в програму необхідно ввести вхід для задання керуючого сигналу, відповідального за режим паралельного завантаження (LOAD) і входи для подачі записуваної інформації (DATA[WIDTH..0]). Очевидно, що розрядність слова даних вхідної інформації повинна збігатися з розрядністю лічильника, тобто бути параметризированою.

```
PARAMETERS (WIDTH=7);
        ASSERT (WIDTH>0)
                REPORT "Значення параметра WIDNH повинно бути більше % " WIDTH
                SEVERITY ERROR;
SUBDESIGN COUNT_4
(
CLK, RESET, SET             :       INPUT = VCC;
ENABLE, IN/DEC              :       INPUT = VCC;
LOAD, DATA[WIDTH..0]        :       INPUT;
Q[WIDTH..0]                 :       OUTPUT;
)
VARIABLE
        RR[WIDTH..0]        : DFFE;
BEGIN
        RR[].(CLK, CLRN, PRN, ENA) = (CLK, RESET, SET, ENABLE);
        IF LOAD THEN
```

```
        -- Load=1 Режим синхронного паралельного завантаження
                RR[].D = DATA[];

        ELSE                                    -- Load=0 Режим рахунку
                IF IN/DEC==1 THEN                       -- Визначення напрямку рахунку
                        RR[].D = RR[].Q+1;  -- Виконання операції інкремента
                ELSE
                        RR[].D = RR[].Q-1;  -- Виконання операції декремента
                END IF;
        END IF;
        Q[] = RR[].Q;                           -- Присвоєння вихідного сигналу
END;
```

В наведеному тексті програми наведено додаткові коментарі, що пояснюють виконувані програмою дії.

На рис.1.7 показані результати роботи розглянутої програми.

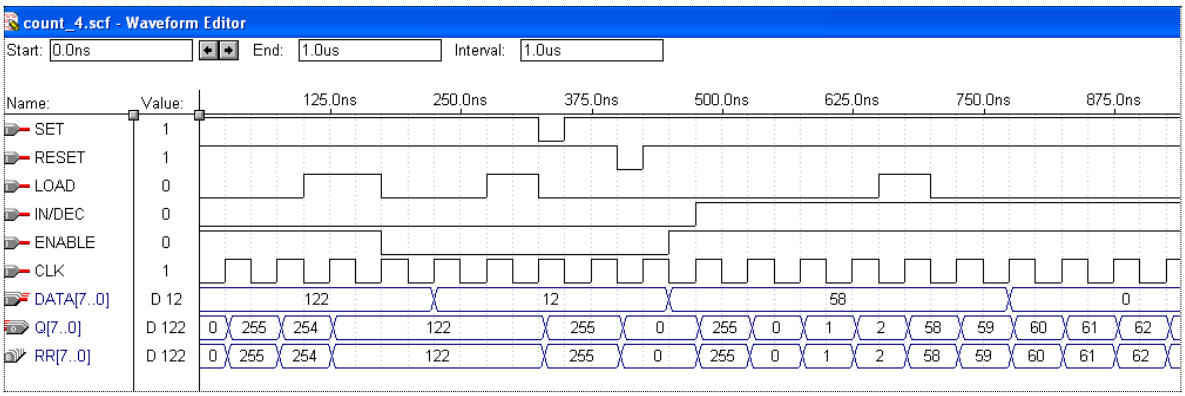


Рисунок 1.7 — Результати моделювання **р**еверсного лічильника з паралельним завантаженням

З наведених часових діаграм випливає, що пріоритет керуючого сигналу LOAD нижчий, ніж сигналу ENABLE. Заслуговує на увагу момент формування фронту сигналу ENABLE. Фронт глобального тактового сигналу надходить на вхід тригерів раніше, ніж вихідний сигнал логічного елемента знімає активний рівень сигналу ENABLE. У результаті перемикання тригера не відбувається й на виході лічильника зберігається нульовий код.

1.5. Двійково-десятковий лічильник

Розглянемо чотирирозрядний двійково-десятковий реверсний лічильник з паралельним завантаженням, у якого:

Clk − тактовий вхід;

nReset − вхід асинхронного скидання лічильника в нуль;

Load − вхід дозволу завантаження;

Dir_CNT − вхід керування напрямком рахунку;

Din[4..1] − чотирирозрядний вхід даних;

Dout[4..1] − чотирирозрядний вихід даних

Нижче наведений текст програми, що описує роботу такого лічильника.

```
SUBDESIGN CNT_4
(
Clk, nReset, Dir_CNT, Din[4..1], Load: INPUT;
Dout[4..1]: OUTPUT;
)
VARIABLE
Count[4..1]: DFF;
BEGIN
Count[].(Clk)=Global(Clk);
Count[].(Clrn)=Global(nReset);
IF Load
        THEN
        IF Din[]>B"1001"
          THEN Count[].D=B"1001";
          ELSE Count[].D=Din[];
        END IF;
END IF;
IF (!Load & Dir_CNT)
 THEN
        IF(Count[].Q>=B"1001")
        THEN Count[].D=B"0000";
           ELSE Count[].D=Count[].Q+1;
  END IF;
END IF;
IF (!Load & !Dir_CNT)
        THEN
         IF(Count[].Q==B"0000")
          THEN Count[].D=B"1001";
          ELSE Count [].D=Count[].Q-1;
        END IF;
END IF;
Dout[]=Count[].Q;
END;
```

Часові діаграми, що пояснюють роботу лічильника, наведені на рис. 1.8.

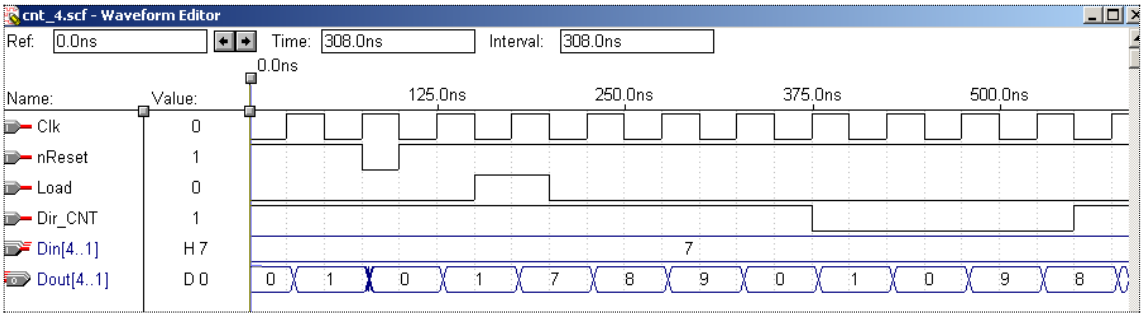


Рисунок 1.8 — Часові діаграми, що характеризують роботу лічильника

Лабораторна робота №10

Тема: Проектування модулів множення з використанням FUNCTION lpm_mult

Мета роботи: Набуття практичних навичок проектування цифрових пристроїв множення мовою опису апаратури AHDL з використанням компоненти **lpm_mult** пакету MAX+PLUS II.

Теоретичні відомості

При описі модуля множення можна скористатися його параметризованою заготовкою, прототип якої заходиться у файлі **LPM_MULT.INC** бібліотеки **\maxplus2\max2lib\mega_lpm**. Для того щоб скористатися цією заготовкою (як і будь-якою іншою бібліотечною заготовкою), потрібно в текстовому редакторі звернутись до її опису шляхом натискання File/Open/All files і вибору із списку компоненту **LPM_MULT.INC**, яка має вигляд.

```
FUNCTION lpm_mult
(
        dataa[(LPM_WIDTHA - 1)..0],
        datab[(LPM_WIDTHB - 1)..0],
        sum[(LPM_WIDTHS - 1)..0],
        aclr,
        clock,
        clken
)
WITH
(
        LPM_WIDTHA,
        LPM_WIDTHB,
        LPM_WIDTHP,
        LPM_WIDTHS,
        LPM_REPRESENTATION,
        LATENCY,
        LPM_PIPELINE,
        INPUT_A_IS_CONSTANT,
        INPUT_B_IS_CONSTANT,
        USE_EAB,
        MAXIMIZE_SPEED
)
RETURNS
(
        result[(LPM_WIDTHP - 1)..0]
);
```

У наведеному описі для вхідних (**INPUTS**) портів виписати імена тих, які є обов'язковими (**Required**) для виконання. Це порт **dataa[]**, який має розмірність **LPM_WIDTHA** і порт **datab[]**, який має розмірність, оголошену змінною з іменем **LPM_WIDTHB**. Для вихідних портів (**OUTPUTS**) виписуємо єдине ім'я **result[]** й ім'я змінної **LPM_WIDTHP**, яка оголошує розмірність результату. У наведеному вище описі вони виділені жирним.

Далі слід звернутися до опису розмірності вхідних і вхідни**x** портів (**PARAMETERS**). Відмітимо, що саме у можливості змінювати величину цих параметрів і полягає параметризація модулів. Із описаного видно, що обов'язковими є оголошення LPM_WIDTHA, LPM_WIDTHB, LPM_WIDTHP, LPM_WIDTHS.

Тобто, крім розмірності множників і результату, потрібно задати розмірність **LPM_WIDTHS** внутрішньої змінної **sum**, хоча вона і не використовується в проекті.

Тепер можна звернутися до функції (примітиву) перемножувача, чотирьма способами: з використанням FUNCTION lpm_mult; з використанням звертання до функції LPM_MULT безпосередньо в розділі логіки; з оголошенням змінної в розділі VARIABLE, якій присвоюється примітив LPM_MULT і звертанням до графічним символу примітиву LPM_MULT.

1. Модуль множення з використанням FUNCTION lpm_mult

```
FUNCTION lpm_mult
(
        dataa[(LPM_WIDTHA - 1)..0],
        datab[(LPM_WIDTHB - 1)..0],
        sum[(LPM_WIDTHS - 1)..0],
)
WITH
(       LPM_WIDTHA,
        LPM_WIDTHB,
        LPM_WIDTHP,
        LPM_WIDTHS,
        RETURNS
(       result[(LPM_WIDTHP - 1)..0]  );
PARAMETERS (N=4);
SUBDESIGN multnxn_func
( A[N..1], B[N..1]:INPUT;
Result[N*2..1]:OUTPUT; )
VARIABLE
MULT:LPM_MULT with (LPM_WIDTHA=N, LPM_WIDTHB=N, LPM_WIDTHP=N*2,
LPM_WIDTHS=N*2);
BEGIN
mult.dataA[]=A[];
mult.dataB[]=B[];
Result[]=mult.result[];
END;
```

2. Звертання до функції LPM_MULT безпосередньо в розділі логіки

```
INCLUDE "LPM_MULT.INC";
SUBDESIGN multNxN_1
(
a[N..1], b[N..1]:INPUT;
result[2*N..1]:OUTPUT;
)
BEGIN
result[]=LPM_MULT(.dataa[]=a[],.datab[]=b[]) with
                (LPM_WIDTHA=N, LPM_WIDTHB=N, LPM_WIDT Hp=2*N,
LPM_WIDTHS=2*N);
END;
```

3. Оголошення змінної в розділі VARIABLE, якій присвоюється примітив LPM_MULT

```
INCLUDE "LPM_MULT.INC";
SUBDESIGN multNxN_2
(
a[N..1], b[N..1]:INPUT;
result[2*N..1]:OUTPUT;
)
VARIABLE
mult:LPM_MULT with (LPM_WIDTHA=N, LPM_WIDTHB=N, LPM_WIDTHP=2*N,
LPM_WIDTHS=2*N);
BEGIN
mult.dataa[]=a[];
mult.datab[]=b[];
result[]=mult.result[];
END;
```

Результати моделювання з використанням **FUNCTION lpm_mult** наведено на рис.1.1.

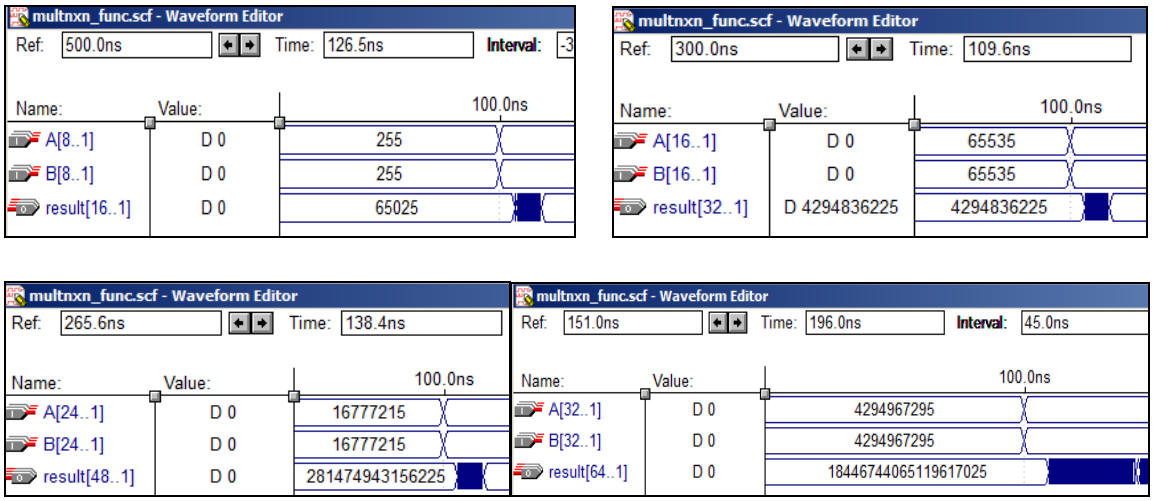


Рисунок 3.1 — Результати моделювання з використанням **FUNCTION lpm_mult**

4. Графічний спосіб з використанням графічного символу примітиву LPM_MULT

Для створення пристрою множення відкриваємо графічний редактор. Після цього enter_symbol, натискуємо meg_lpm і вибираємо lpm_mult. В результаті таких дій появиться символ (рис. 4.1), який налаштовуємо на перемноження двійкових чисел відповідної розрядності, задаючи обов'язковим портам імена і відповідні значення (рис. 4.2).

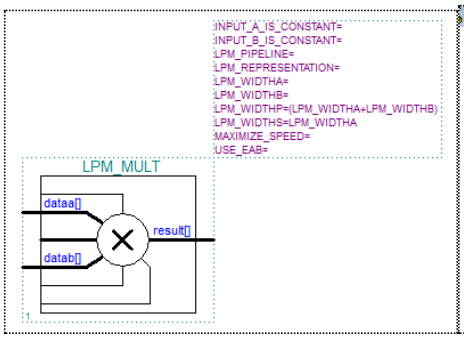


Рисунок 4.1

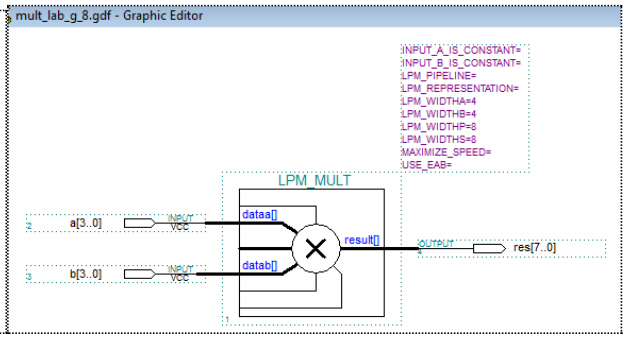


Рисунок 4.2

Для задання відповідних параметрів потрібно два рази клацнути лівою кнопкою миші на відповідному параметрі, в результаті чого появиться випадаюче вікно Edit Ports/ Parameters (рис. 4.3).

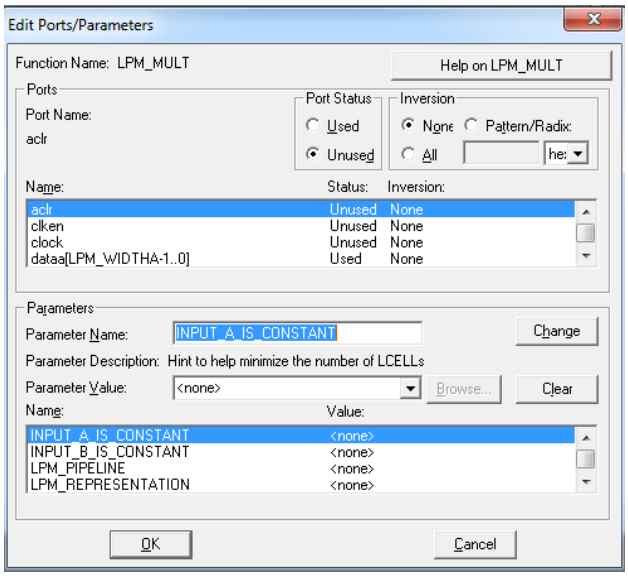


Рисунок 4.3

Далі, у нижній частині вікна знаходимо відповідний параметр (наприклад LPM_WIDTHA=), активізуємо його і на місце <none> набираємо число 4 (рис.4.4). Аналогічно задаємо інші три параметри (4,8,8), які показані внизу випадаючого вікна.

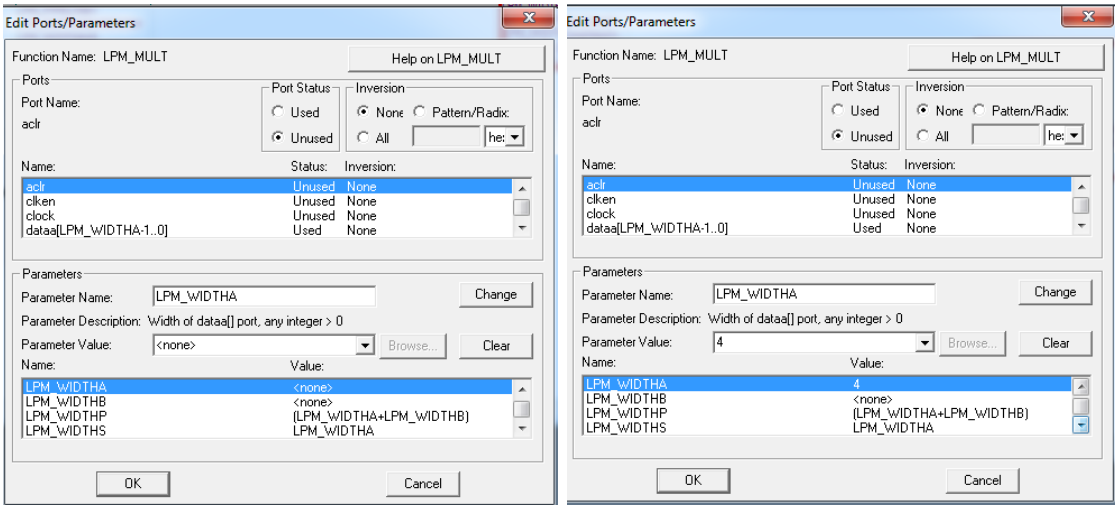


Рисунок 4.4

Результати моделювання побудованого пристрою множення наведено на рис. 4.5.

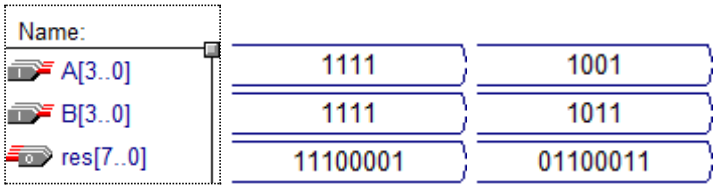


Рисунок 4.5 — Результати моделювання пристрою множення

Завдання. Використовуючи наведені програми розробити параметризовані модулі для $N = 8, 16, 24, 32$. Виконати відповідну компіляцію і симуляцію.

Лабораторна робота № 11

Тема: Опис скінченних автоматів мовою AHDL в програмному комплексі Max+plus II

Мета роботи: навчитися проектувати цифрові автомати мовою AHDL в програмному комплексі MAX+PLUS II.

Теоретичні відомості

Узагальнену схему автомата наведено на рис.1.1, де

x[N..1] — вхідні сигнали;

y[k..1] — вихідні сигнали;

q[K..1] — розряди пам'яті, які визначають стан автомата;

D[n..1] — дані для запису в пам'ять;

RG — пам'ять автомата (набір тригерів);

КЦ1 — комбінаційна схема, яка забезпечує формування даних для запису в пам'ять;

КЦ2 — комбінаційна схема, яка породжує вихідні сигнали.

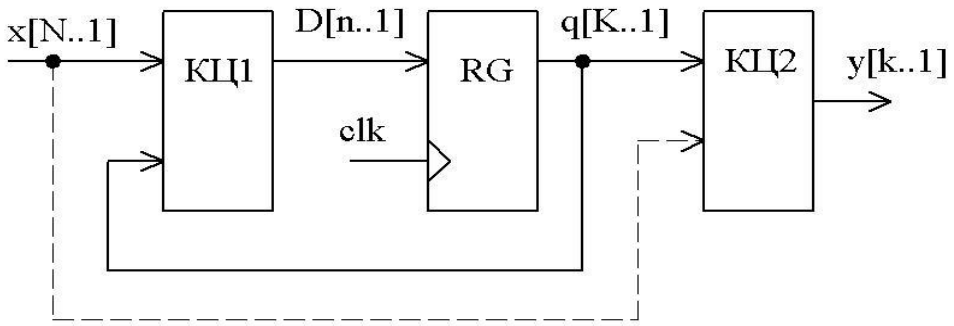


Рисунок 1.1 — Узагальнена схема автомата

Синхронний скінчений автомат (ССА) - автомат, пам'ять якого реалізована на синхронних тригерах. ССА переходить зі стану в стан тільки по активному перепаду (фронту або спаду) тактового сигналу синхронізуючи тригери в його блоці пам'яті.

Залежно від способу формування вихідних сигналів виділяють два класи скінчених автоматів:

•Автомат Мура - автомат, вихідні сигнали в якому залежать тільки від поточнного стану автомата.

•Автомат Милі - автомат, вихідні сигнали в якому залежать від поточного стану автомата й від поточних вхідних сигналів (пунктирна лінія на рис. 1).

Наступний приклад демонструє вигляд загального опису скінченого автомата:

VARIABLE

ss : MACHINE OF BITS (q1, q2, q3) with STATES (s1 = B"000", s2 = B"010", s3 = B"111");

Ім'я скінченого автомата в даному прикладі **ss**. Після ключових слів **MACHINE OF BITS** у дужках перераховуються імена виходів тригерів даного автомата: **q1, q2** і **q3**. Кожний тригер визначає біт стану автомата. Далі оголошуються імена станів автомата, і числове значення, що присвоюється цьому стану. Станами даного скінченого автомата є **s1, s2** і **s3**, кожному з яких присвоєно числове значення подане бітами відповідно **B"000", B"010", B"111"**.

Необов'язковим є вказівка ключового слова **OF BITS** з перерахуванням імен бітів станів. Необов'язково також присвоювати числові значення іменам станів у розділі **with STATES**. У найпростішому випадку опис буде таким:

VARIABLE

ss: MACHINE with STATES (s1,s2,s3);

У цьому випадку компілятор, оптимізуючи комбінаційні схеми КЦ1 і КЦ2, самостійно вибере як розрядність блоку пам'яті, так і коди станів автомата.

Число тригерів (n_t), використовуваних для реалізації модуля пам'яті автомата, визначається числом станів автомата (N_s) і способом їхнього кодування:

• Двійкове кодування (**Binary coding**). При цьому $n_t =]\log_2 Ns[$, де **] [** — операція округлення до найближчого більшого цілого.

• Кодування за принципом: один стан — один тригер (**One Hot State**). При цьому $n_t = Ns$.

Для НВІС програмувальної логіки сімейств MAX, які містять відносно невелике число макроосередків, у кожнім з яких є багатовходова програмуєма матриця «І», матриця «АБО» і тригер, прийнятні обидва способи кодування.

Для опису автомата спочатку потрібно скласти таблиці переходів та виходів або намалювати діаграму станів, потім описати побудовані таблиці або діаграму мовою AHDL. Компілятор автоматично виконає наступні функції:

1) призначить біти, вибираючи або D або T тригер (DFF або TFF) для кожного біта присвоїть значення станам;

2) застосує техніку логічного синтезу для одержання рівнянь збудження тригерів.

Подібно тригерам, кінцевий автомат має три внутрішніх керуючих входи:

clk — вхід тактового сигналу (активний — фронт);

Reset — вхід асинхронного скидання автомата (активний рівень — логічна одиниця);

ena — вхід дозволу роботи автомата (активний рівень — логічна одиниця).

Обов'язкове використання тільки входу **clk**. Якщо входи **Reset** і/або **ena** не використовуються, то на них автоматично подаються логічні рівні, що не перешкоджають нормальній роботі автомата. Проект **simple1**, наведений нижче, володіє такою ж функціональністю як **D** тригер (**DFF**).

```
SUBDESIGN simple1
(   clk, reset, d : INPUT;
   q  : OUTPUT;)
VARIABLE
   ss: MACHINE WITH STATES (s0, s1);
BEGIN
   ss.clk = clk;
   ss.reset = reset;
   CASE ss IS
      WHEN s0 =>
         q = GND;
         IF d THEN
            ss = s1;
         END IF;
      WHEN s1 =>
         q = VCC;
         IF !d THEN
            ss = s0;
         END IF;
   END CASE;
END;
```

На рис.1.2 наведено схему для порівняння роботи тригера DFF і його реалізацію за допомогою автомата

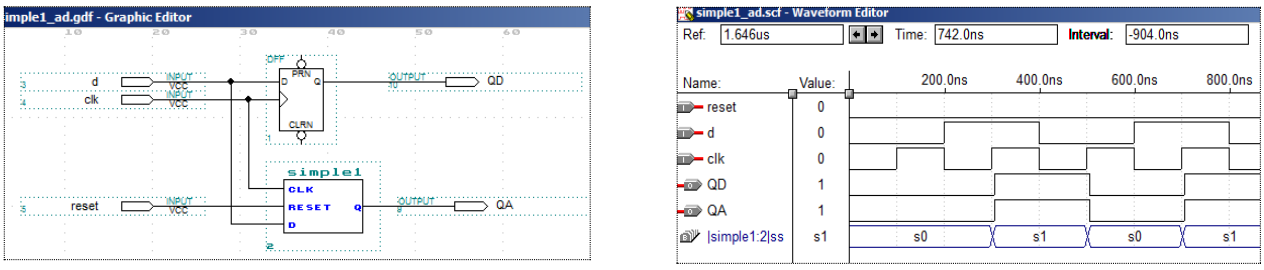


Рисунок 1.2 — Схема для порівняння та результати моделювання

Для прикладу розглянемо абстрактний автомат Мілі заданий таблицями переходів (табл.1.1) і виходів (табл.1.2).

Таблиця 1.1

Стан автомата	Вхідні сигнали	
	x_0	x_1
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_0	s_1

Таблиця 1.2

Стан автомата	Вхідні сигнали	
	x_0	x_1
s_0	y_0	y_2
s_1	y_1	y_3
s_2	y_0	y_1

З навчальною метою проектування розглядуваного автомата виконаємо трьома способами:

1. Графічним способом з використанням середовища MAX+PLUS II;
2. Мовою AHDL з використанням таблиці істинності;
3. Мовою AHDL з використанням оператора CASE.

1. Проектування автоматів графічним способом в середовищі MAX+PLUS II

Побудуємо закодовані таблиці переходів (табл. 1.3) і виходів (табл. 1.4)

Таблиця 1.3

Стан автомата	Вхідні сигнали	
	$x=0$	$x=1$
00	01	00
01	01	10
10	00	01

Таблиця 1.4

Стан автомата	Вхідні сигнали	
	$x=0$	$x=1$
00	00	10
01	01	11
10	00	01

Для побудови функцій збудження D-тригерів скористаємось таблицею 5

Таблиця 1.5

0	$\xrightarrow{D=0}$	0
0	$\xrightarrow{D=1}$	1
1	$\xrightarrow{D=0}$	0
1	$\xrightarrow{D=1}$	1

Нижче наведено таблицю функцій збудження для D-тригера (табл. 1.6) та таблицю функції виходу (табл. 1.7). Звернемо увагу на те, що при використанні D-тригерів таблиця переходів (табл. 1.3) завжди співпадає з таблицею функцій збудження (табл.1.6).

Таблиця 1.6

Стан автомата	Вхідні сигнали	
	$x=0$	$x=1$
0 0	0 1	0 0
0 1	0 1	1 0
1 0	0 0	0 1
q_2 q_1	D_2 D_1	D_2 D_1

Таблиця 1.7

Стан автомата	Вхідні сигнали	
	$x=0$	$x=1$
0 0	00	10
0 1	01	11
1 0	00	01
q_2 q_1	y_2 y_1	y_2 y_1

На основі таблиць 1.6, 1.7 будуємо функції збудження і виходів.

Логічні рівняння функцій збудження D-тригерів:

$$D_1 = \bar{q}_2\bar{q}_1\bar{x} + \bar{q}_2 q_1\bar{x} + q_2\bar{q}_1 x = \bar{q}_2\bar{x} + q_2\bar{q}_1 x, \quad D_2 = \bar{q}_2 q_1 x.$$

Логічні рівняння функцій виходів:

$$y_1 = \bar{q}_2 q_1\bar{x} + \bar{q}_2 q_1 x + q_2\bar{q}_1 x = \bar{q}_2 q_1 + q_2\bar{q}_1 x, \quad y_2 = \bar{q}_2\bar{q}_1 x + q_2\bar{q}_1 x = \bar{q}_1 x.$$

Використовуючи одержані функції будуємо функціональну схему автомата Мілі (рис.1.3) та виконуємо її моделювання. Результати моделювання схемної реалізації автомата (проект mealy_k_g) наведено на рис.1.4. За результатами моделювання можна відслідкувати переходи з одного стану в інший та виходи в залежності від вхідного сигналу та поточного стану автомата.

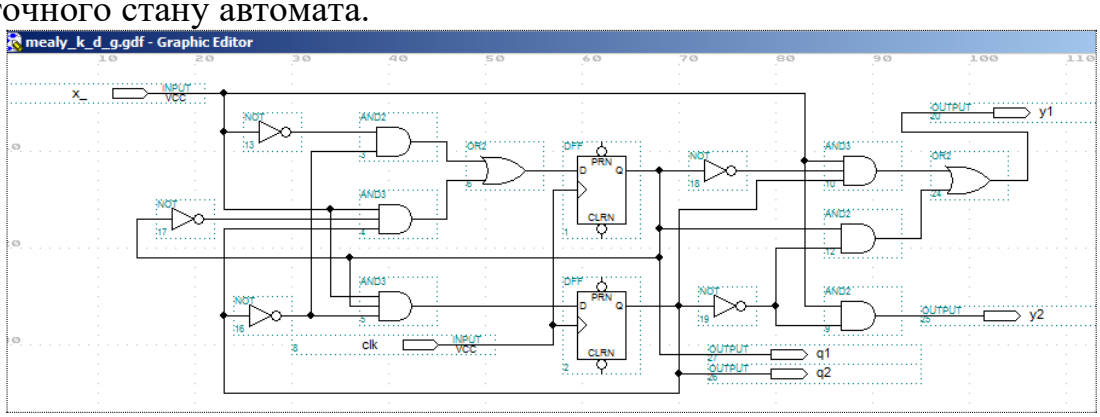


Рисунок 1.3 — Функціональна схема скінченого автомата

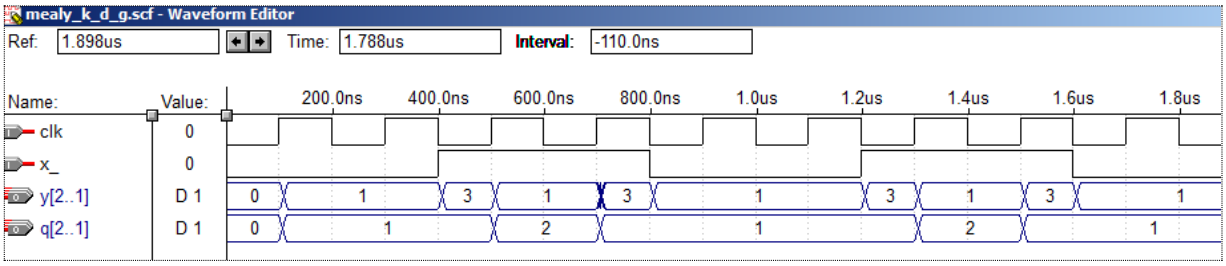


Рисунок 1.4 — Результати моделювання

2. Проектування автомата мовою AHDL з використанням таблиці істинності Table

Текстовий опис автомата Мілі з використанням таблиці істинності має вигляд:

```
SUBDESIGN mealy_k_D
( clk : INPUT;
reset : INPUT;
x_ : INPUT;
y[2..1] : OUTPUT; )
VARIABLE
ss: MACHINE WITH STATES (s0, s1, s2);
BEGIN
ss.clk = clk;   ss.reset = reset;
TABLE
% стан, вхід     виходи стан %
        ss, x_  =>  y2, y1,   ss;
        s0, 0   =>   0,  0,    s1;
        s0, 1   =>   1,  0,    s0;
        s1, 0   =>   0,  1,    s1;
        s1, 1   =>   1,  1,    s2;
        s2, 0   =>   0,  0,    s0;
        s2, 1   =>   0,  1,    s1;
END TABLE;
END;
```

На рис. 2.1 наведено результати моделювання автомата, описаного в проекті mealy_k_D.

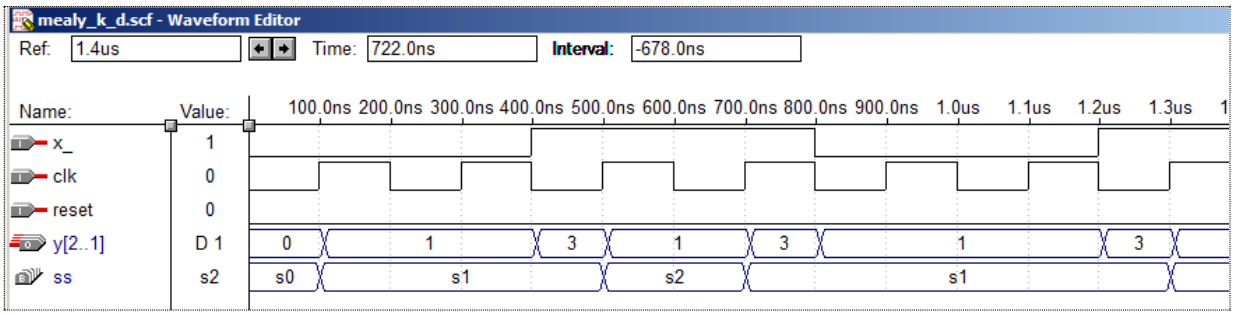


Рисунок 2.1 — Результати моделювання

Для зручнішої перевірки еквівалентності одержаних результатів створимо проект із символів **mealy_k_g** і **mealy_k_d** графічного і текстового опису автомата Мілі (рис.2.2).

Результати моделювання такого проекту наведено на рис. 2.3. Відмітимо повне співпадання, як станів переходу та і функцій виходу розглядуваних автоматів.

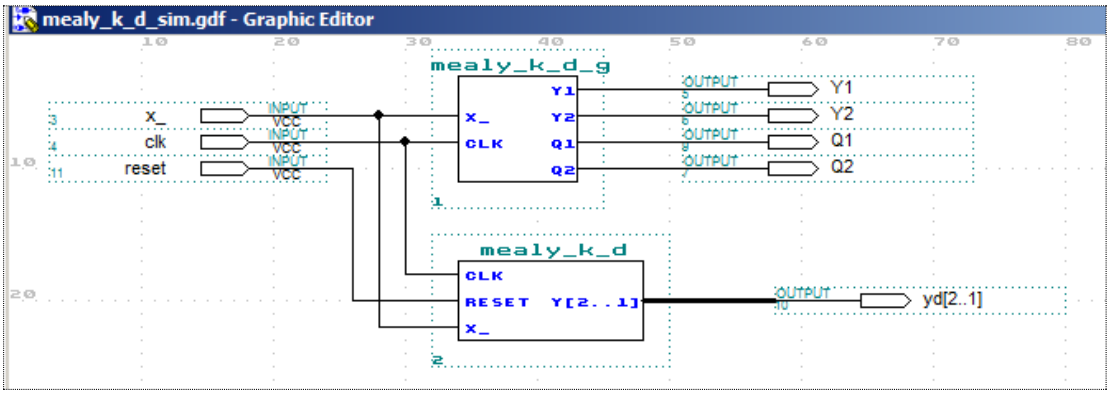


Рисунок 2.2 —. Схема скінчених автоматів на основі символів

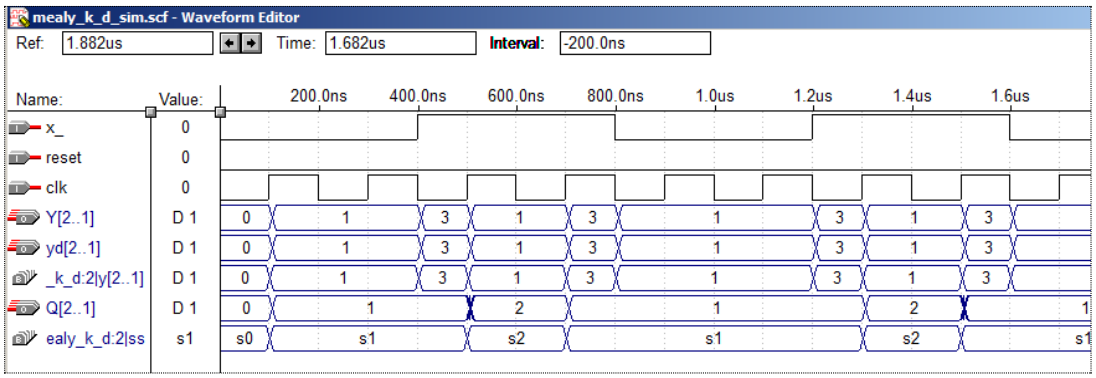


Рисунок 2.3 — Результати моделювання

3. Проектування автомата мовою AHDL з використанням оператора CASE

Текстовий опис автомата Мілі з використанням оператора CASE наведено в проекті mealy_k_case. Результати моделювання даного проекту наведено на рис. 8. Відмітимо повне співпадання, як станів переходу та і функцій виходу, одержаних у попередніх проектах.

```
SUBDESIGN mealy_k_case
(
x_, CLK, reset        :        INPUT;
y[2..1]          :        OUTPUT;
)
VARIABLE
        FSM : MACHINE WITH STATES (s0, s1, s2);
BEGIN
        FSM.(CLK,RESET) = (CLK, reset);
        CASE FSM   IS
                WHEN s0      =>
                        IF x_ == 0
```

```
                    THEN
                        FSM = s1;  y2=GND;  y1=GND;
                    ELSE
                        FSM = s0;  y2=VCC;  y1=GND;
                END IF;
            WHEN s1=>
                IF x_ == 0
                    THEN
                        FSM = s1;  y2=GND;  y1=VCC;
                    ELSE
                        FSM = s2;  y2=VCC;  y1=VCC;
                END IF;
            WHEN s2    =>
                IF x_ == 0
                    THEN
                        FSM = s0;  y2=GND;  y1=GND;
                    ELSE
                        FSM = s1;  y2=GND;  y1=VCC;
                END IF;
        END CASE;
END;
```

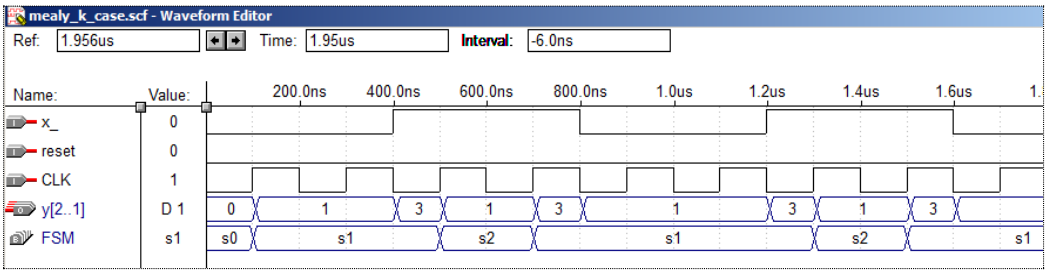


Рисунок 3.1 —. Результати моделювання

4. Порядок виконання роботи

1. Кодуємо таблицю переходів та виходів автомата.
2. Будуємо таблицю збудження автомата.
3. Складаємо рівняння для побудови комбінаційної схеми збудження.
4. Складаємо рівняння для побудови комбінаційної схеми формування вихідних сигналів автомата.
5. Будуємо функціональну схему цифрового автомата.
6. Перевіряємо роботу схеми цифрового автомата.
7. Виконуємо реалізацію мовного опису цифрового автомата з використанням таблиці істинності та досліджуємо його роботу.
8. Виконуємо реалізацію мовного опису цифрового автомата з використанням оператора Case та досліджуємо його роботу.
9. Порівнюємо результати, одержані трьома способами.

5. Зміст звіту

1. Тема і мета роботи.
2. Вихідні дані для виконання роботи.
3. Результати виконання пунктів 1 – 3.
4. Функціональні схеми роботи цифрового автомата з пам’яттю та часові діаграми.
5. Висновки.

6. Індивідуальні завдання для лабораторної роботи

Спроектувати цифровий автомат Мілі згідно заданих таблиць переходів і виходів. Комбінаційну схему збудження і комбінаційну схему формування вихідних сигналів реалізувати з використанням D-тригерів.

Варіант №1.1

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_1	s_1
s_2	s_2	s_2
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_2	y_3
s_2	y_2	y_4
s_3	y_1	y_4

Варіант №1.2

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_1	s_1
s_2	s_2	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_1	y_3
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.3

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_3
s_2	s_1	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_3	y_1
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.4

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_2
s_2	s_3	s_1
s_3	s_1	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_1	y_3
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.5

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_3	s_3
s_2	s_2	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_1	y_2
s_2	y_3	y_4
s_3	y_2	y_2

Варіант №1.6

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_1	s_2
s_2	s_2	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_4	y_3
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.7

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_1	s_3
s_2	s_2	s_1
s_3	s_1	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_1	y_3
s_2	y_2	y_4
s_3	y_2	y_2

Варіант №1.8

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_3	s_3
s_2	s_2	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_4	y_3
s_2	y_2	y_4
s_3	y_1	y_1

Варіант №1.9

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_2
s_2	s_1	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_1	y_3
s_2	y_4	y_4
s_3	y_1	y_2

Варіант №1.10

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_1
s_2	s_3	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_2	y_3
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.11

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_2
s_2	s_2	s_1
s_3	s_1	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_3	y_3
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.12

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_1	s_1
s_2	s_1	s_1
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_3	y_3
s_2	y_2	y_4
s_3	y_1	y_2

Варіант №1.13

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_1
s_2	s_2	s_3
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_3	y_3
s_2	y_2	y_4
s_3	y_4	y_2

Варіант №1.14

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_2	s_3
s_2	s_2	s_1
s_3	s_1	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_4	y_3
s_2	y_2	y_1
s_3	y_1	y_2

Варіант №1.15

Таблиця переходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	s_3	s_1
s_2	s_2	s_3
s_3	s_3	s_2

Таблиця виходів

Стан автомата	*Вхідні сигнали*	
	x_1	x_2
s_1	y_1	y_3
s_2	y_4	y_2
s_3	y_1	y_2

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Антонов А.П. Мова опису цифрових пристроїв AlteraHDL: Практичний курс. – М.: ІП «Радіософт», 2001. – До книги додається CD – ROM, що містить САПР Max+plus II.
2. Король І.Ю. Мова опису апаратури AHDL: Навчальний посібник для студентів напрямку 6.050102 – «Комп'ютерна інженерія»/ І.Ю. Король–Ужгород: УжНУ, 2008. –104 с.
3. Мальцев П.П. Цифровые интегральные микросхемы: Справочник / П.П. Мальцев. – М.: Радио и связь. 1994. –185 с.
4. Опадчий Ю.Ф. Основы языка AHDL / Ю.Ф. Опадчий // Язык описания аппаратуры AHDL. –2005. № 1. – С. 27- 35.
5. Стешенко В.Б. ПЛІС фірми ALTERA: Проектування пристроїв обробки сигналів. – М.: Додека, 2000.

ЗМІСТ

ВСТУП 3

ЛАБОРАТОРНА РОБОТА №6.

Типові вузли цифрової схемотехніки. Побудова шифраторів та дешифраторів 4

ЛАБОРАТОРНА РОБОТА №7.

Типові вузли цифрової схемотехніки. Побудова мультиплексорів та демультиплексорів. 14

ЛАБОРАТОРНА РОБОТА №8.

Проектування цифрових пристроїв з пам’яттю 29

ЛАБОРАТОРНА РОБОТА №9.

Опис лічильників мовою AHDL 41

ЛАБОРАТОРНА РОБОТА №10.

Проектування модулів множення з використанням FUNCTION lpm_mult 49

ЛАБОРАТОРНА РОБОТА №11.

Опис скінченних автоматів мовою AHDL в програмному комплексі Max+plus II 54

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 66