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 1.

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 1.

Укладачі: Король І.Ю., канд. фіз.-мат. наук, доцент 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 №1

Тема: Програмувальні логічні матриці (ПЛМ)

Мета роботи: 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) та одержання навичок роботи із основними редакторами системи MAX+PLUS II.

Теоретичні відомості

1.1. Вступні зауваження

У цифрові системи обробки інформації входять процесор, пам'ять, периферійні пристрої й інтерфейсні схеми. Процесор є стандартним пристроєм — він не виготовляється для конкретної системи за спеціальним замовленням, а розв’язує необхідне завдання шляхом послідовного виконання певних команд із властивої йому системи команд.

Пам'ять також реалізується стандартними мікросхемами — її функції залишаються тими самими для різних систем.

Стандартні ВІС/НВІС (Великі Інтегральні Схеми / НадВеликі Інтегральні Схеми) лідирують за рівнем інтеграції, але вартість проектування оптимізованих по щільності ВІС/НВІС обходиться дуже дорого.

Поряд зі стандартними в системі присутні й деякі нестандартні частини, специфічні для даної розробки. Це відноситься до схем керування блоками, забезпечення їхньої взаємодії й ін. Реалізація нестандартної частини системи історично була пов'язана із застосуванням мікросхем малого й середнього рівнів інтеграції. Застосування МІС (Малих Інтегральних Схем) і СІС (Середніх Інтегральних Схем) супроводжується різким ростом числа корпусів ІС (Інтегральних Схем), ускладненням монтажу, зниженням надійності системи і її швидкодії. У той же час замовити для системи спеціалізовані ІС високого рівня інтеграції важкувато, тому що це пов'язане з дуже великими витратами засобів і часу на проектування ВІС/НВІС.

Виникле протиріччя знайшло розв’язання на шляхах розробки ВІС/НВІС із програмувальною й репрограмувальною структурою.

Першими представниками зазначеного напрямку з'явилися програмувальні логічні матриці ПЛМ (PLA, Programmable Logic Array), програмувальна матрична логіка ПМЛ (PAL, Programmable Array Logic) і базові матричні кристали (БМК), які також називають вентильними матрицями (ВМ) (GA, Gate Array).

PLA і PAL в англійській термінології об’єднуються також терміном PLD (Programmable Logic Devices).

Розвиток ВІС/НВІС із програмувальною й репрограмувальною структурою виявилося настільки перспективним напрямком, що привело до створення нових ефективних засобів розробки цифрових систем, таких як CPLD (Complex PLD), FPGA (Field Programmable GA) і SPGA (System Programmable GA).

1.2. Програмувальні логічні матриці (ПЛМ)

Програмувальні логічні матриці з'явилися в середині 70-х років. Основою їх служить послідовність програмувальних матриць елементів І й АБО. У структуру входять також блоки вхідних і вихідних буферних каскадів (Бвх і Бвих).

Вихідні буфери забезпечують необхідну навантажувальну здатність виходів, дозволяють або забороняють вихід ПЛМ на зовнішні шини за допомогою сигналу ОЕ, а іноді виконують і більш складні дії.

Основними параметрами ПЛМ (рис. 1.1) є число входів *m*, число термів *l* і число виходів *n*.

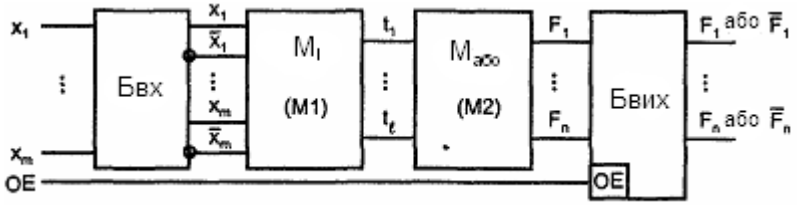


Рисунок 1.1 − Базова структура ПЛМ

Змінні $x_1 \ldots x_m$ подаються через Бвх на входи елементів І (кон’юнкторів), і в матриці M_I $(M1)$ утворяться l термів $(t_1, t_2, \ldots t_l)$. Під термом тут розуміється кон’юнкція, що зв'язує вхідні змінні, представлені в прямій або інверсній формі. Число сформованих термів дорівнює числу кон’юнкторів або, що теж саме, числу виходів матриці M_I.

Терми подаються далі на входи матриці $M_{АБО}$ $(M2)$, тобто на входи диз’юнкторів, що формують вихідні функції. Число диз’юнкторів дорівнює числу створюваних функцій *n*.

Таким чином, ПЛМ реалізує диз'юнктивну нормальну форму (ДНФ) відтворюваних функцій (дворівневу логіку). ПЛМ здатна реалізувати систему *n* логічних функцій від *m* аргументів, що містить не більше *l* термів. Відтворені функції є комбінаціями з будь-якого числа термів, сформованих матрицею M_I. Які саме терми будуть створені і які комбінації цих термів складуть вихідні функції, визначається програмуванням ПЛМ.

1.3. Схемотехніка ПЛМ

Випускаються ПЛМ як на основі біполярної технології, так і на Моп-транзисторах. У матрицях є системи горизонтальних і вертикальних зв'язків, у вузлах перетинання яких при програмуванні створюються або ліквідуються елементи зв'язку.

На рис.1.2, а в спрощеному вигляді (без буферних елементів) показана схемотехніка біполярної ПЛМ із програмуванням перепалювання перемичок, де показано фрагмент для відтворення системи функцій:

$$F_1 = x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_3x_2 \vee \bar{x}_4x_1 = t_1 \vee t_2 \vee t_3;$$

$$F_2 = x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_3\bar{x}_2x_1 \vee x_4x_2x_1 \vee x_4\bar{x}_3x_2 = t_1 \vee t_4 \vee t_5 \vee t_6;$$

$$F_3 = \bar{x}_4x_1 \vee \bar{x}_2x_1 = t_3 \vee t_7,$$

розмірністю 4 (змінні), 7 (кон'юнкторів), 3 (функції). Елементами зв'язків у матриці M_I служать діоди, що з'єднують горизонтальні й вертикальні шини, як показано на рис. 2, б, що зображує ланцюг формування терму t_1. Разом з резистором і джерелом живлення ланцюги формування термів утворюють звичайні діодні схеми M_I. До програмування всі перемички цілі й діоди зв'язку розміщені у всіх вузлах координатної сітки.

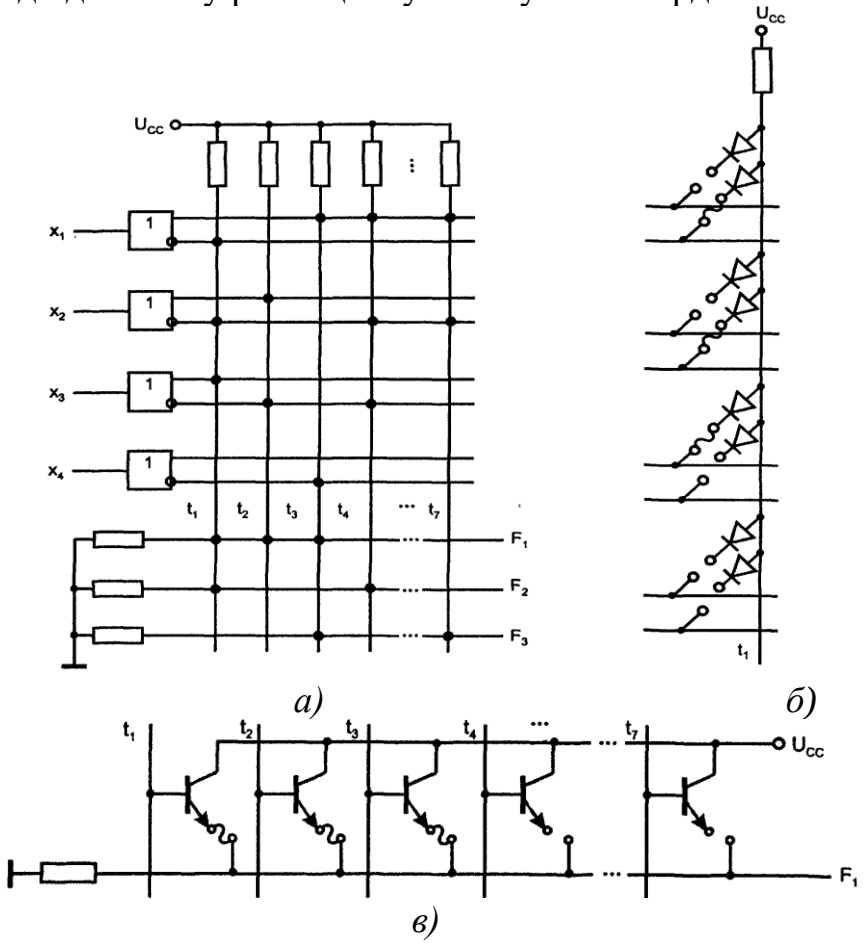


Рисунок 1.2 − Схемотехніка ПЛМ, реалізованої в біполярній технології (а), і елементи зв'язків у матрицях M_I (б) і $M_{АБО}$ (в)

При будь-якій комбінації аргументів на виході буде нуль, тому що на вхід схеми подаються одночасно прямі й інверсні значення аргументів, а $x\&\bar{x}=0$. При програмуванні в схемі залишаються тільки необхідні елементи зв'язку, а непотрібні усуваються перепалюванням перемичок. У цьому випадку на вхід кон'юнктора подані $\bar{x}_1$, $\bar{x}_2$ і x_3. Високий рівень вихідної напруги (логічна одиниця) з'явиться тільки при наявності високих напруг на всіх входах, низька напруга хоча б на одному вході фіксує вихідну напругу низького рівня, оскільки відкривається діод цього входу. Так виконується операція І, у цьому випадку формується терм $x_3\bar{x}_2\bar{x}_1$.

Елементами зв'язку в матриці $M_{АБО}$ служать транзистори (рис. 1.2, *в*), включені за схемою емітерного повторювача відносно ліній термів і утворюючі схему АБО щодо виходу (горизонтальної лінії). На рис.1.2, *в* показано формування функції F_1.

2. Підготовка завдання до розв'язання за допомогою ПЛМ

Маючи на увазі підбор ПЛМ мінімальної складності, варто зменшити, по можливості, число термів у даній системі функцій. Змістом мінімізації функцій буде пошук найкоротших диз'юнктивних форм. Вести пошук мінімальних по числу термів подання задачі треба до рівня, коли число термів стає рівним l — параметру наявних ПЛМ. Подальша мінімізація не потрібна. Якщо розмірність наявних ПЛМ забезпечує розв'язання задачі в її вихідній формі, то мінімізація не потрібна взагалі, оскільки не веде до скорочення устаткування.

2.1. Програмування ПЛМ

Програмування ПЛМ, виконуване користувачем, проводиться за допомогою спеціальних пристроїв (програматорів) і відомості для них про дану ПЛМ повинні мати певну форму. Є програматори, які приймають в якості інформації про ПЛМ таблицю функціонування (істинності), однак зручніше задавати відомості про самі перемички. Символи, використовувані при такому задані відомостей для програмування ПЛМ:

Н — змінна входить у терм у прямому вигляді, тобто потрібно залишити цілою перемичку прямого входу й перепалити перемичку інверсного входу;

L — змінна входить у терм в інверсному вигляді, тобто потрібно зберегти перемичку в інверсного входу й перепалити в прямого;

"–" — змінна не входить у терм і не повинна впливати на нього, тобто потрібно перепалити перемички обох входів.

Залишення перемичок в обох входах змінної як би усуває з матриці відповідну схему І, оскільки в силу рівності $x\bar{x}=0$ вихід цієї схеми завжди нульовий і не впливає на роботу матриці АБО, на вхід якої подається;

А — вказується у вихідному стовпці (стовпці функції) і свідчить про зв'язок даної схеми І з виходом ПЛМ через матрицю АБО. Перемичка повинна бути збережена;

"." — указує на те, що дана схема І не підключається до виходу й повинна мати перепалену перемичку в матриці АБО.

У прийнятій символіці для програмування ПЛМ узятого раніше прикладу відомості будуть задані таблицям: табл. 1.1 та табл. 1.2.

Таблиця 1.1

x_1	x_2	x_3	x_4	F_1	F_2	F_3
L	L	H	—	A	A	.
—	H	L	—	A	.	.
H	—	—	L	A	.	A
H	L	L	—	.	A	.
H	H	—	H	.	A	.
—	H	L	H	.	A	.
H	L	—	—	.	.	A

Таблиця 1.2

t_i		x_4	x_3	x_2	x_1		F_1	F_2	F_3
t_1		–	1	0	0		1	1	•
t_2		–	0	1	–		1	•	•
t_3		0	–	–	1		1	•	1
t_4		–	0	0	1		•	1	•
t_5		1	–	1	1		•	1	•
t_6		1	0	1	–		•	1	•
t_7		0	–	0	1		•	•	1

Через відсутність спеціальних пристроїв (програматорів) для виконання програмування ПЛМ нами розроблено логічні схеми, які дають можливість реалізовувати програмування і репрограмування ПЛМ.

На рис. 1.3 наведено вигляд ПЛМ виконану в системі проектування MAX+PLUS II, яка, до певної міри, дає уявлення про ідею використання ПЛМ для реалізації логічних функцій та їх перепрограмування.

Операція набору відповідних термів за допомогою з'єднань відповідних ліній (відображено крапкою) здійснюється шляхом клацання курсором у потрібному місці з'єднання прямих і натиснення піктограми вставки або вилучення вузла, яка знаходиться на полі графічного редактора. Нижче, для прикладу, реалізовано наведені вище функції F_1, F_2 і F_3. Ще для двох функцій F_4 і F_5, якщо є потреба, зроблено заготовки.

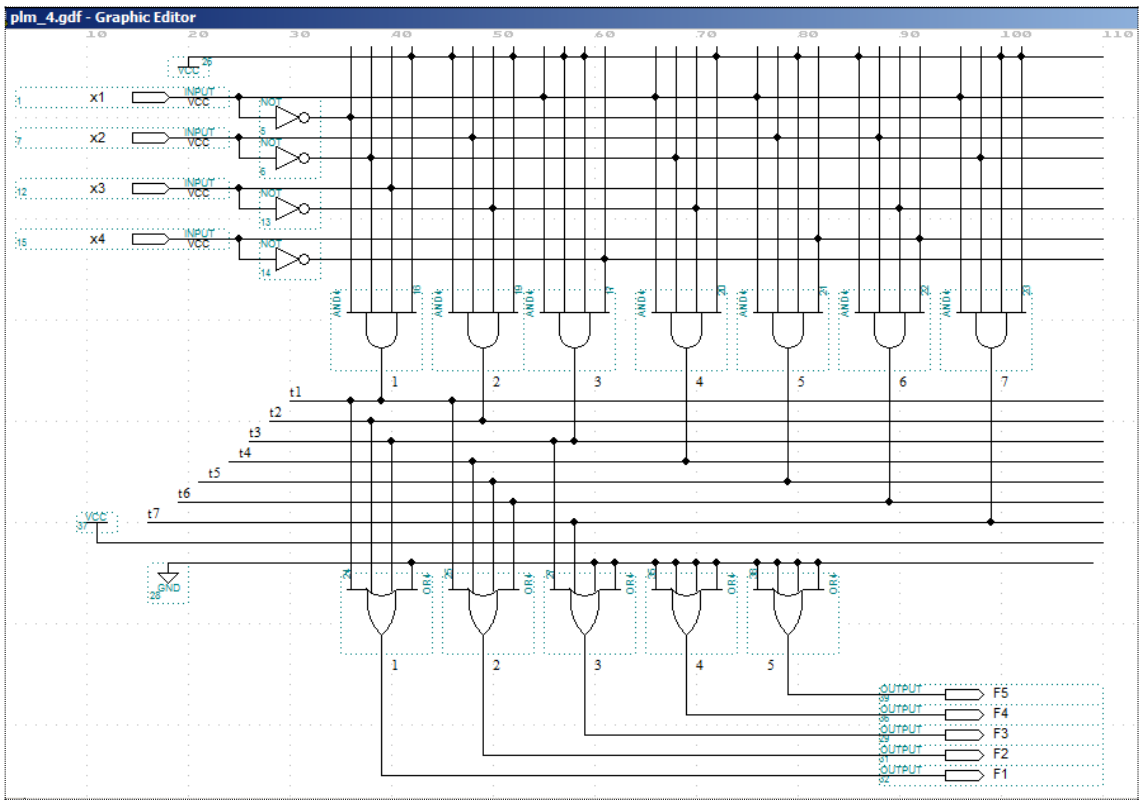


Рисунок 1.3 − Вигляд ПЛМ, реалізованої в системі проектування MAX+PLUS II

Результати моделювання побудованої схеми наведено на рис.1.4.

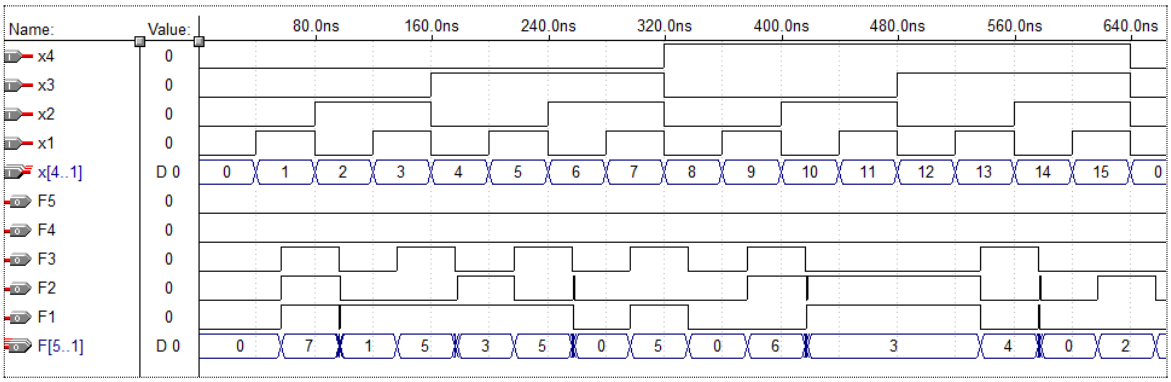


Рисунок 1.4 − Результати моделювання, наведеної ПЛМ

Нижче наведено вирази функцій, поданих у ДДНФ (через мінтерми), що дає можливість легко встановити на яких наборах функції приймають значення 1. Це дає можливість легко перевірити правильність роботи схеми.

$$F_1 = x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_3x_2 \vee \bar{x}_4x_1 = x_4x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_4x_3\bar{x}_2\bar{x}_1 \vee x_4\bar{x}_3x_2x_1 \vee x_4\bar{x}_3x_2\bar{x}_1 \vee$$
$$\vee \bar{x}_4\bar{x}_3x_2x_1 \vee \bar{x}_4\bar{x}_3x_2\bar{x}_1 \vee \bar{x}_4x_3x_2x_1 \vee \bar{x}_4x_3\bar{x}_2x_1 \vee \bar{x}_4\bar{x}_3x_2x_1 \vee \bar{x}_4\bar{x}_3\bar{x}_2x_1 =$$
$$= 12 \vee 4 \vee 11 \vee 10 \vee 3 \vee 2 \vee 7 \vee 5 \vee 3 \vee 1 = 1 \vee 2 \vee 3 \vee 4 \vee 5 \vee 7 \vee 10 \vee 11 \vee 12;$$
$$F_2 = x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_3\bar{x}_2x_1 \vee x_4x_2x_1 \vee x_4\bar{x}_3x_2 = x_4x_3\bar{x}_2\bar{x}_1 \vee \bar{x}_4x_3\bar{x}_2\bar{x}_1 \vee x_4\bar{x}_3\bar{x}_2x_1 \vee \bar{x}_4\bar{x}_3\bar{x}_2x_1 \vee$$
$$\vee x_4x_3x_2x_1 \vee x_4\bar{x}_3x_2x_1 \vee x_4\bar{x}_3x_2x_1 \vee x_4\bar{x}_3x_2\bar{x}_1 = 1 \vee 4 \vee 9 \vee 10 \vee 11 \vee 12 \vee 15;$$
$$F_3 = \bar{x}_4x_1 \vee \bar{x}_2x_1 = \bar{x}_4x_3x_2x_1 \vee \bar{x}_4x_3\bar{x}_2x_1 \vee \bar{x}_4\bar{x}_3x_2x_1 \vee \bar{x}_4\bar{x}_3\bar{x}_2x_1 \vee$$
$$\vee x_4x_3\bar{x}_2x_1 \vee x_4\bar{x}_3\bar{x}_2x_1 \vee \bar{x}_4x_3\bar{x}_2x_1 \vee \bar{x}_4\bar{x}_3\bar{x}_2x_1 = 1 \vee 3 \vee 5 \vee 7 \vee 9 \vee 13.$$

З метою перевірки правильності запрограмування ПЛМ цю ж саму задачу реалізуємо графічним способом в середовищі MAX+PLUS II (проект $F1\_F3\_g$), мовою опису апаратури AHDL з використанням змінних (проект $F1\_F3$) та логічних елементів (проект $F1\_F3\_e$). Нижче наведено схемну та мовну (програмну) реалізацію проектів та результати моделювання їхньої роботи (рис. 1.5, рис. 1.6, рис. 1.7), які підтверджують правильність роботи ПЛМ.

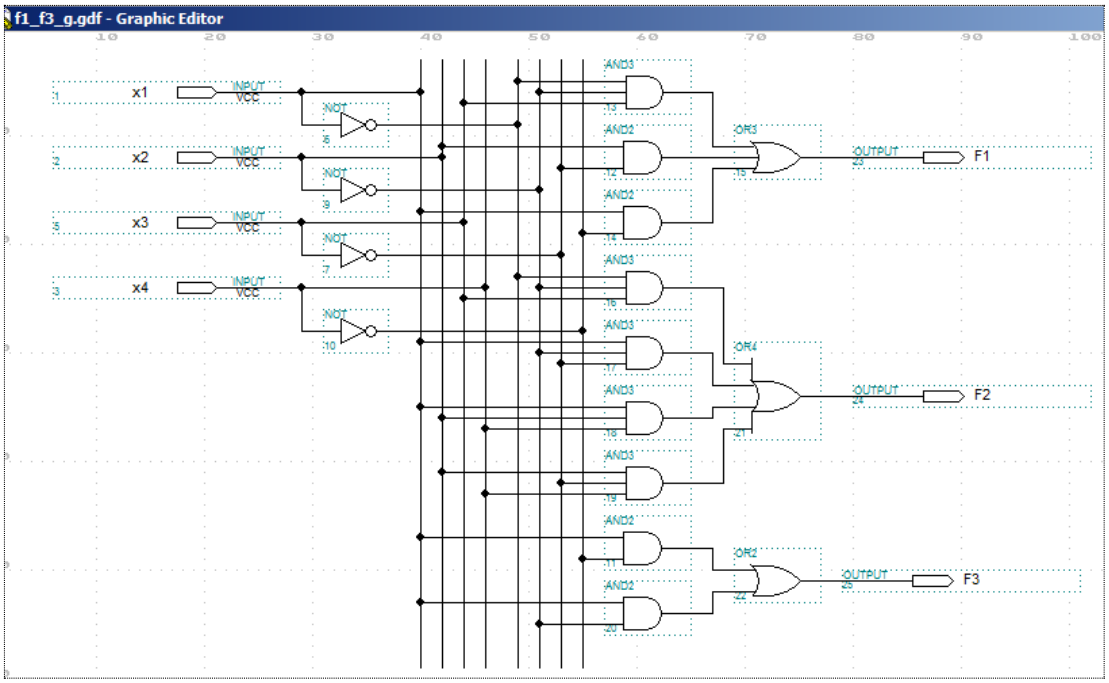


Рисунок 1.5 – Схема реалізації графічним способом

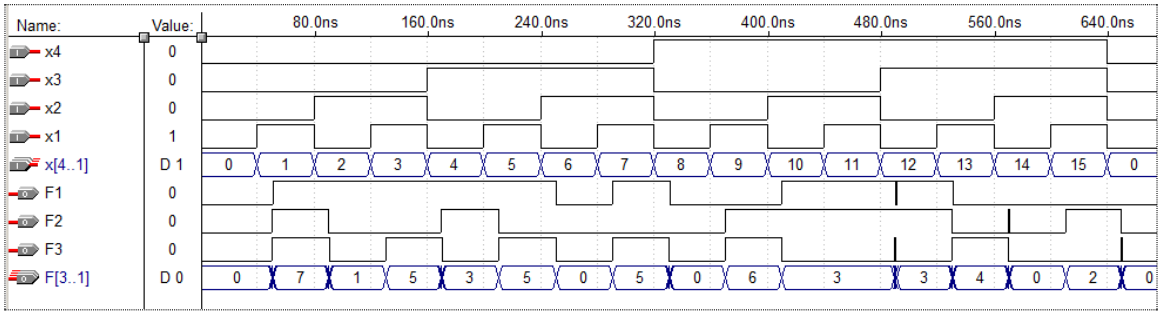


Рисунок 1.6 – Результати моделювання схемного проекту

```
SUBDESIGN F1_F3
(_x[4..1]: INPUT;
  F[3..1]: OUTPUT;)
BEGIN
F1=_x3&!_x2&!_x1#!_x3&_x2#!_x4&_x1;
F2=_x3&!_x2&!_x1#!_x3&!_x2&_x1#_x4&_x2&_x1#_x4&!_x3&_x2;
F3=!_x4&_x1#!_x2&_x1;
END;
```

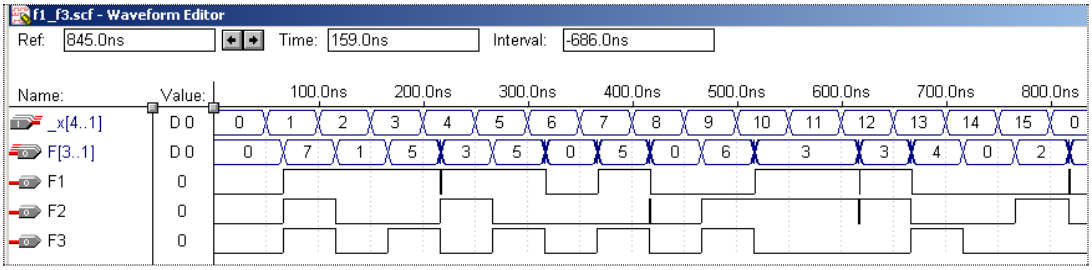


Рисунок 1.7 − Результати моделювання проекту (програми) F1_F3

```
SUBDESIGN F1_F3_e
(_x[4..1]: INPUT;
  F[3..1]: OUTPUT;)
BEGIN
F1=_x3 AND NOT _x2 AND NOT _x1 OR NOT _x3 AND _x2
          OR NOT _x4 AND _x1;
F2=_x3 AND NOT _x2 AND NOT _x1 OR NOT _x3 AND NOT _x2 AND _x1
          OR _x4 AND _x2 AND _x1 OR _x4 AND NOT _x3 AND _x2;
F3=NOT _x4 AND _x1 OR NOT _x2 AND _x1;
END;
```

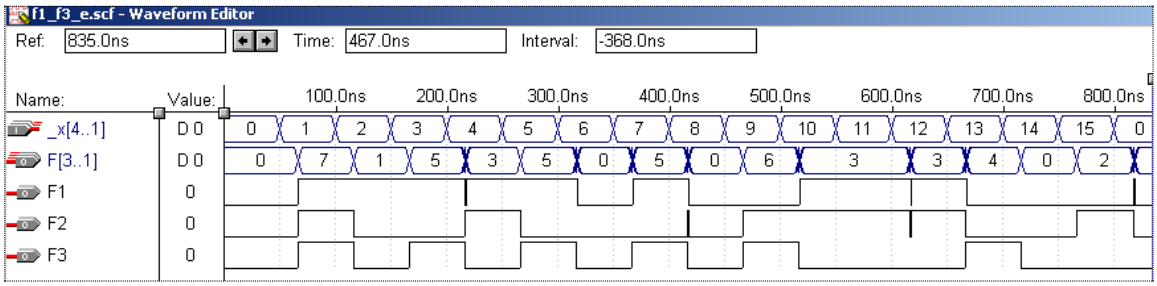


Рисунок 1.8 − Результати моделювання проекту (програми) F1_F3_e

3. Завдання для самостійної роботи

3.1 Розробити ПЛМ для реалізації функцій від двох змінних і запрограмувати шість основних функцій: кон'юнкції (І), диз'юнкції (АБО), суми за модулем 2, штрих Шеффера (І-НЕ), стрілка Пірса (АБО-НЕ) і еквівалентність (заперечення суми за модулем 2).

3.2. Скласти таблицю істинності для п’яти функцій від чотирьох змінних заповнивши її своїми даним, приймаючи до уваги, що голосні буки кодуються символом 0, приголосні букви й м’який знак символом 1.

3.3. Перепрограмувати задану ПЛМ у відповідності зі своїми даними. Правильність перепрограмування перевірити шляхом моделювання.

3.4. Підготувати короткий звіт і захистити виконану роботу.

Приклад заповнення таблиці, вигляд перепрограмувальної ПЛМ та результати моделювання наведено нижче: табл.1.3, рис. 1.9 − 1.10.

Таблиця 1.3 − Приклад функцій для програмування на ПЛМ

№	Змінні				Функції					Букви для кодування				
	x_3	x_2	x_1	x_0	F_1	F_2	F_3	F_4	F_5	b_1	b_2	b_3	b_4	b_5
0	0	0	0	0	1	1	1	0	1	М	Л	Т	Е	Д
1	0	0	0	1	0	0	0	1	1	О	А	Е	Т	Р
2	0	0	1	0	1	1	1	1	0	С	Й	Х	Н	А
3	0	0	1	1	1	0	1	0	1	К	О	Н	А	К
4	0	1	0	0	0	1	0	1	1	А	В	І	П	С
5	0	1	0	1	1	0	1	1	1	Л	И	Ч	Р	М
6	0	1	1	0	1	1	1	0	1	Ь	Ч	Н	Я	К
7	0	1	1	1	1	0	0	1	1	Д	І	И	М	Р
8	1	0	0	0	0	1	1	0	0	Е	Н	Й	О	А
9	1	0	0	1	1	1	1	1	1	Н	Ж	Ф	К	Щ
10	1	0	1	0	0	0	0	1	0	И	Е	А	К	А
11	1	0	1	1	1	1	1	0	1	С	Н	К	І	З
12	1	1	0	0	1	0	0	1	0	М	Е	У	К	А
13	1	1	0	1	0	1	1	0	1	И	Р	Л	А	В
14	1	1	1	0	1	1	1	1	1	К	Н	Ь	Ф	С
15	1	1	1	1	0	0	1	0	0	О	О	Т	Е	І

Схема ПЛМ для перепрограмування

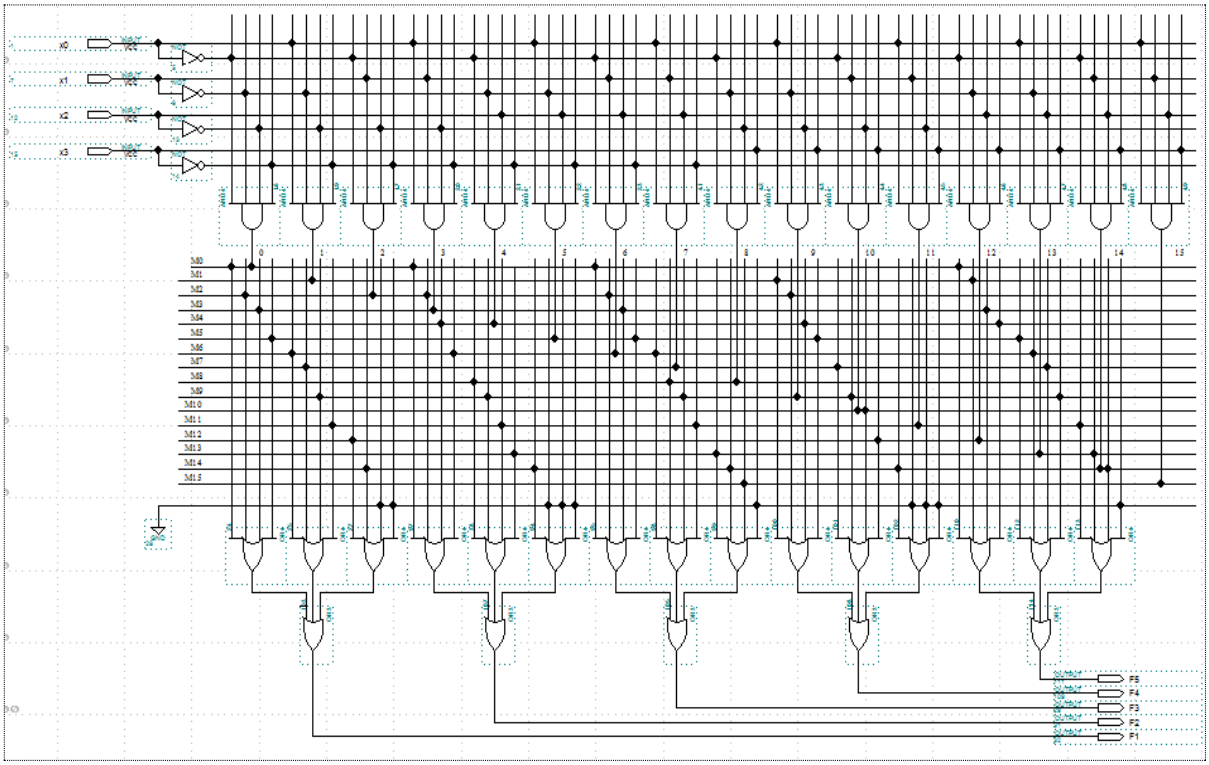


Рисунок 1.9 − Вигляд ПЛМ, реалізованої в системі проектування MAX+PLUS II для реалізації п’яти функцій

Результати моделювання схеми, зображеної на рис.1.9. наведено на рис.1.10.

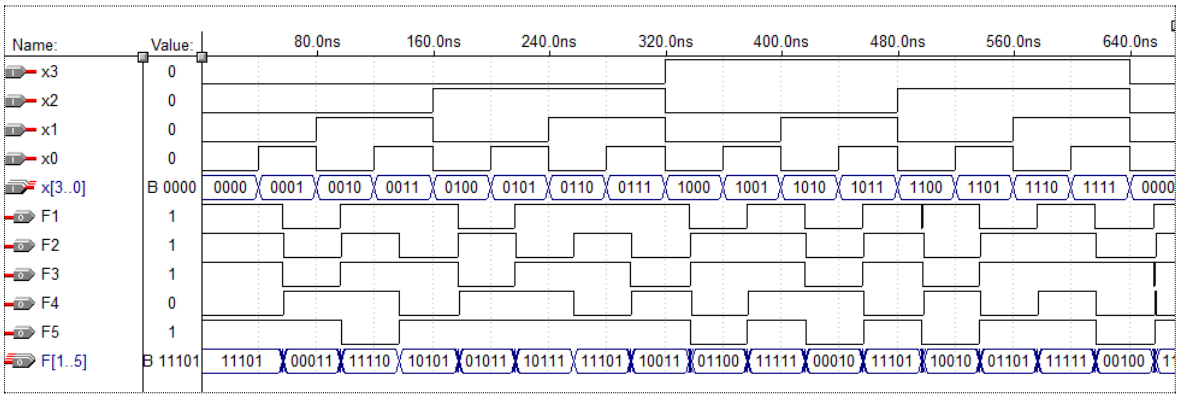


Рисунок 1.10 − Результати моделювання схемного проекту ПЛМ

4. Зміст звіту

4.1 Короткі теоретичні відомості.

4.2 Вихідні дані для виконання роботи.

4.3 Результати виконання роботи (схема, часові діаграми та таблиці істинності).

4.4 Висновки.

5. Контрольні запитання

5.1. Наведіть базову структуру ПЛМ та її основні параметри.

5.2. Наведіть основні компоненти схемотехніки ПЛМ, реалізованої в біполярній технології та на вкажіть способи її перепрограмування.

5.3. Наведіть вигляд таблиць, які використовуються при програмувані ПЛМ.

5.4. На прикладі програмування (перепрограмування) функцій від двох змінних поясніть структуру ПЛМ, реалізованої в системі проектування MAX+PLUS II.

5.5. Якими способами можна перевірити правильність роботи ПЛМ, якщо функції задані логічними формулами або таблицями істинності.

Лабораторна робота № 2

Тема: Проектування 7-сегментного світлодіодного індикатора

Мета роботи: ознайомитись з проектуванням комплексних комбінаційних схем з використанням графічного редактора Graphic Editor, редактора часових діаграм Waveform Editor та редактора зв'язків Floprplan Editor середовища проектування MAX + plus II.

1. Постановка задачі

В даній лабораторній роботі порядок використання основних додатків системи проектування MAX + plus II розглянемо на прикладі розробки пристрою індикації двійкових даних у шістнадцятковій формі на 2-розрядному 7-сегментному світлодіодному індикаторі для гіпотетичної мікропроцесорної системи з 8- розрядною шиною даних і 16- розрядною шиною адреси. Для визначеності будемо вважати, що порт індикації має адресу 10h, а в якості індикатора використовується індикатор типу DA-56 із загальним анодом, тобто сигнали управління сегментами повинні мати активні низькі рівні для підсвічування сегментів, відповідних шістнадцятковому поданню вхідного двійкового коду.

Основними елементами такого пристрою є:

• схема формування сигналу запису в порт, куди входять 16 - розрядний адресний селектор і схема стробування сигналу запису;

• 8- розрядний регістр порту;

• комбінаційні схеми перетворення двійкових кодів тетрад в сигнали управління сегментами індикатора (навантажувальна здатність ПЛІС фірми Альтера дозволяє підключати світлодіоди безпосередньо до виводів мікросхем через навантажувальні опори номіналом 330 ... 390 Ом).

У розроблюваний проект доцільно включити два рівні ієрархії:

• рівень структурних елементів, в який входить схема формування сигналу запису в порт і комбінаційні схеми перетворення двійкових кодів тетрад в сигнали управління сегментами індикатора;

• рівень проекту в цілому.

2. Реалізація проекту

Для початку, використовуючи стандартні засоби Windows, створимо в каталозі \MAX2WORK робочий каталог під ім'ям \MOD_IND. Потім створимо проект схеми формування сигналу запису в порт. Будемо створювати його в графічному редакторі. Для цього запускаємо систему MAX + plus II і створюємо новий файл (крайній лівий значок панелі інструментів). У відкритому діалоговому вікні «New» вибираємо пункт

Graphic Editor file і натискуємо кнопку OK, при цьому автоматично відкривається вікно графічного редактора (рис. 2.1).

Вікно редактора має ряд додаткових пунктів основного меню і панель інструментів редактора, розташовану вертикально з лівого боку вікна. Збережемо
новий файл проекту (через меню File/Save) під ім'ям WRITE_PORT (розширення буде присвоєно автоматично) у створеному каталозі \MOD_IND. Ім’я файлу проекту слід обов'язково прив'язати до імені проекту - це робиться при виборі пункту Set Project to Current File (у підменю Project меню File головного меню робочого вікна. Для створення графічного проекту можна використовувати бібліотеки примітивів (\MAXPLUS \MAX2LIB \ PRIM), макрофункцій (\ MAXPLUS \ MAX2LIB \ MF) і параметризованих мегафункцій (\ MAXPLUS \ MAX2LIB \ MEGA_LPM).

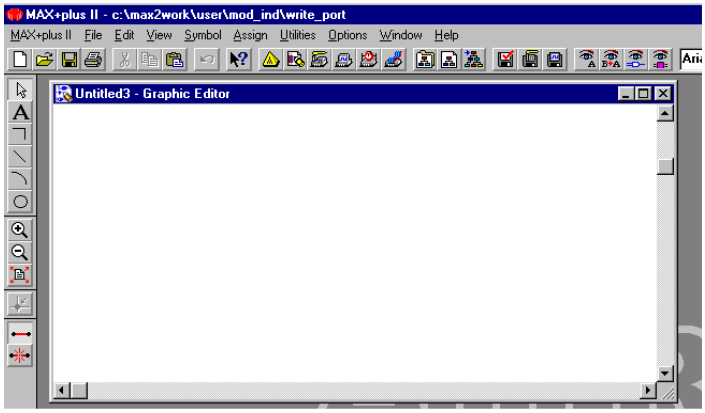


Рисунок 2.1 − Вікно графічного редактора «Graphic Editor»

Примітиви включають великий набір основних логічних елементів, тригерів, елементів входу і виходу (INPUT, OUTPUT, BIDIR), а також допоміжні елементи: GND (логічний нуль), VCC (логічна одиниця).

Макрофункції включають головним чином еквіваленти логічних мікросхем 74- ї серії.

Параметризовані мегафункції дозволяють реалізовувати багатовходові і багаторозрядні елементи цифрової схемотехніки (логіку, регістри, мультиплексори і т.д.), вводячи ряд параметрів в спеціально позначених областях умовних графічних позначень цих елементів. Порядок виконання мегафункцій можна знайти у відповідному розділі електронного довідника системи (Help), а також у документі LPM Quick Reference Guide.

Приступимо безпосередньо до процедури створення проекту, яка здійснюється за кілька кроків.

Крок 1: Створення схеми проекту. Створення загальної схеми проекту почнемо з побудови проекту пристрою формування сигналу запису в порт (рис. 2). Створення схеми цього проекту доцільно починати з розміщення вузлів входів і виходів та присвоєння цим вузлам імен. У нашому випадку необхідно розмістити входи адресних ліній A0... A15, вхід сигналу запису /WR (Символом "/" задається низький активний

рівень) і один вихід сигналу запису в порт WR. Для цієї мети використовуємо бібліотеку примітивів, яка містить примітиви INPUT (для входів) і OUTPUT (для виходів).

Логіка розроблюваного проекту також може бути реалізована з використанням примітивів: двох елементів OR8 (8– входовий елемент «АБО»), одного елемента NOR3 (3– входовий елемент «АБО – НЕ») і одного інвертора NOT. В описаному вище порядку розміщуємо необхідні елементи і виконуємо необхідні з'єднання (рис.2.2).

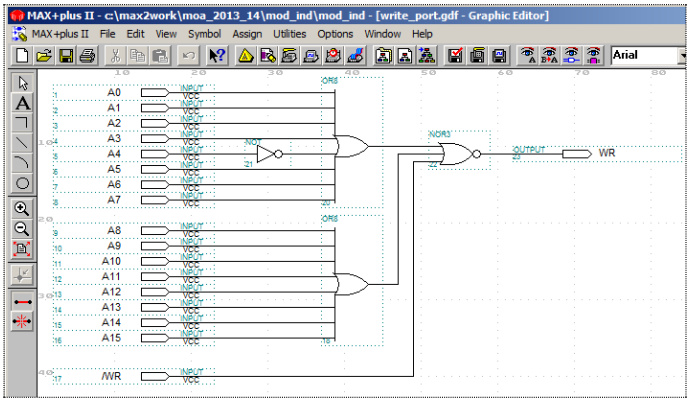


Рисунок 2.2 − Готовий проект пристрою формування сигналу запису в порт

Зауважимо, що запропонований варіант реалізації логіки пристрою – не єдиний і не найкращий. Відмітимо, що реалізувати цей проект можна також з використанням мови AHDL (самий кращий варіант) або мегафункції LPM_OR, що розглянемо в наступних лабораторних роботах.

Крок 2: – компіляція і створення символу проекту. На цьому кроці здійснюється компіляція і створення символу проекту для включення його у файл проекту верхнього рівня. Перед компіляцією можна виконати перевірку коректності введеного проекту. Перевірка здійснюється через підменю Project (меню File головного меню робочого вікна) шляхом вибору пункту Save & Check або клацанням лівої кнопки миші на піктограмі відповідного інструменту основної панелі інструментів.

Компіляція здійснюється також через підменю Project шляхом вибору пункту Save & Compile або знову ж за допомогою відповідного інструменту основної панелі інструментів. Перед компіляцією потрібно вибрати ПЛІС сімейства MAX3000A, а саме ПЛІС типу EPM 3032ALC44 (у корпусі PLCC з 44 виводами).

Створення символу проекту здійснюється через підменю Project, в якому слід вибрати пункт Create Default Symbol – цей пункт стає доступним тільки після закриття вікна компілятора. Створений символ буде поміщений в каталог проекту. Використання створених символів, так само як і елементів інших бібліотек, проводиться через діалогове вікно Enter Symbol.

Крок 3: – Створення проекту і символу комбінаційної схеми перетворення двійкового коду тетради в сигнали управління сегментами індикатора, структура якого, індикація символів при збуджені низьким активним рівнем, наведено на рис. 2.3.

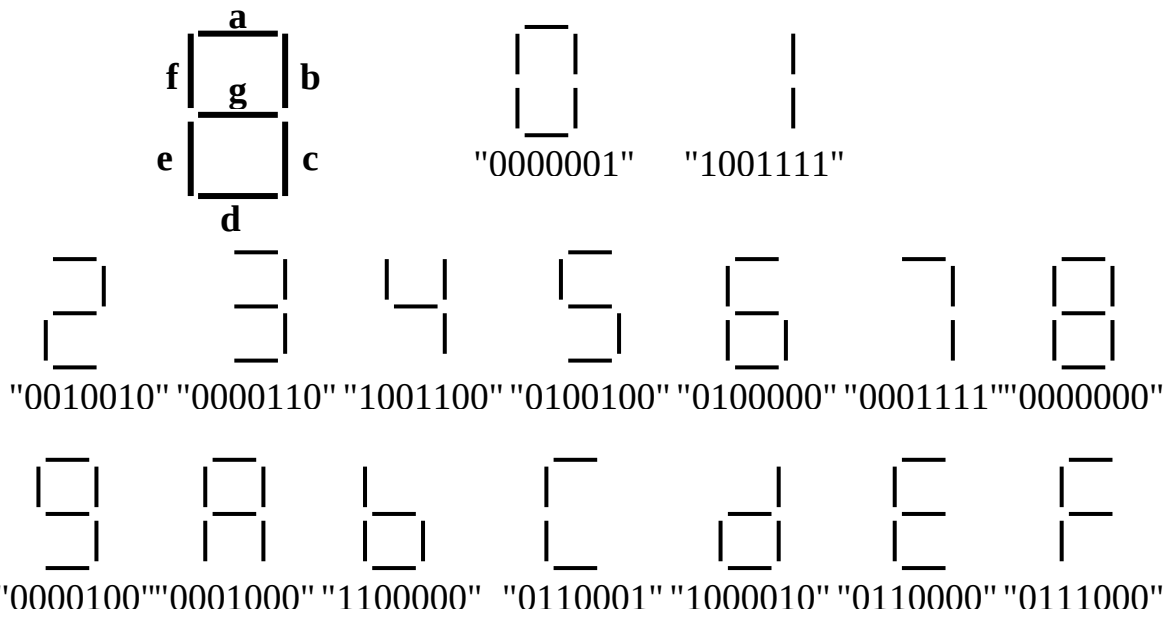


Рисунок 2.3 − Збудження сегментів нулем

В табл. 2.1 наведено стани входів і виходів комбінаційної схеми перетворення двійкових тетрад в семибітний код керування сегментами індикатора. Значення символів шістнадцяткового коду наведено в стовпчику HEX-код.

Таблиця 2.1 − Перелік станів комбінаційної схеми перетворення

HEX-код	Стани входів				Стани входів						
	In3	In2	In1	In0	/a	/b	/c	/d	/e	/f	/g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0
A	1	0	1	0	0	0	0	1	0	0	0
B	1	0	1	1	1	1	0	0	0	0	0
C	1	1	0	0	0	1	1	0	0	0	1
D	1	1	0	1	1	0	0	0	0	1	0
E	1	1	1	0	0	1	1	0	0	0	0
F	1	1	1	1	0	1	1	1	0	0	0

Для створення проекту цього пристрою використаємо редактор часових діаграм (Waveform Editor). Для цього слід натиснути кнопку відкриття нового файлу на панелі інструментів, в діалоговому вікні New відмітити пункт Waveform Editor file, в сусідньому вікні вибрати розширення.WDF (розширення.SCF використовується для моделювання) і натиснути ОК, після чого відкривається вікно редактора.

Вікно редактора має чотири поля, розділених вертикальними лініями. Перше поле зліва (Name) призначено для введення імені вузла, у другому полі (Type) відображається тип введення (INPUT, OUTPUT, BIDIR), в третьому полі (Value) показані стани виводів, що відповідають положенню спеціальної вертикальної візирної лінії, яка при відкритті вікна встановлена в початок горизонтальній осі, розміченій в одиницях часу. Четверте поле призначене для задання необхідних станів виводів, при цьому використовуються інструменти з панелі інструментів редактора, яка розташована вертикально вздовж лівого боку вікна. Активізація панелі інструментів відбувається тільки в тому випадку, якщо виділено один з вузлів. Щоб виділити вузол, необхідно клацнути лівою кнопкою миші на імені вузла, можна також виділити будь–яку ділянку вздовж горизонтальної осі, при цьому межі виділених ділянок прив'язуються до сітки.

Параметри сітки встановлюються таким чином: за допомогою пункту End Time (меню File) задається максимальне значення часового інтервалу із зазначенням одиниць виміру, а за допомогою пункту Grid Size (меню Options) – крок сітки. У верхній частині екрану розташовані вікна для точного відліку інтервалів часу.

Збережемо створений файл в каталозі \ MOD_IND під ім'ям CONV_BIN_HEX і присвоїмо проекту ім'я створеного файлу (через підменю Project меню File головного меню робочого вікна, шляхом вибору пункту Set Project to Current File). Для введення імені вузла можна використовувати пункт Insert Node меню Node.

У нашому випадку (див. табл. 1) необхідно задати 16 можливих станів 4- х входів і станів 7–ми виходів, відповідні кожному стану входів. Для цього вся горизонтальна вісь повинна бути розбита на 16 дискрет. Наприклад, встановивши значення End Time 160 ns, а Grid Size – 10 ns, отримаємо 16 дискрет з кроком сітки 10 ns. Після цього послідовно виділяємо вузли та дискрети і за допомогою панелі інструментів задаємо необхідні стани входів і виходів.

Вигляд уведеного файлу проекту CONV_BIN_HEX наведено на рис.2.4.

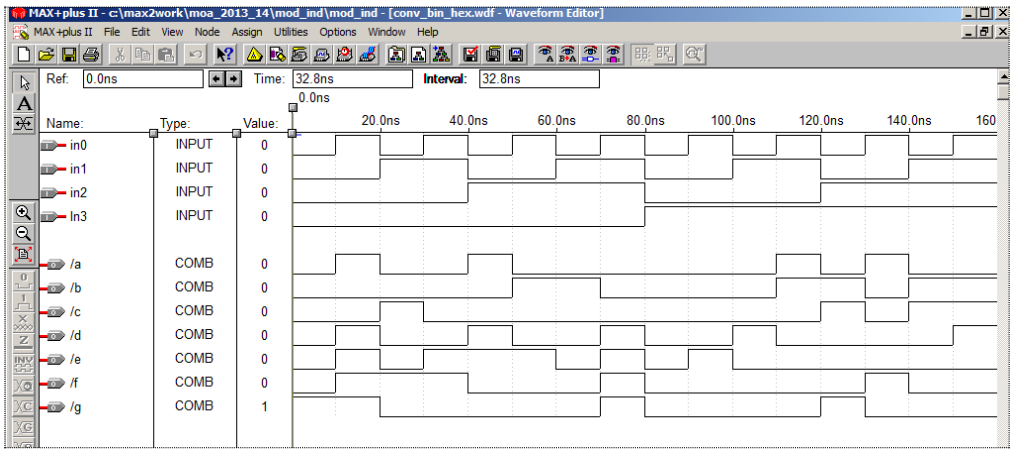


Рисунок 2.4 − Введений файл проекту CONV_BIN_HEX

У стовпчику Type дано характеристики вузлів: INPUT – вхід, COMB – комбінаційний вихід

Далі компілюємо файл і створюємо символ цього пристрою (рис.2.5), як описано вище.

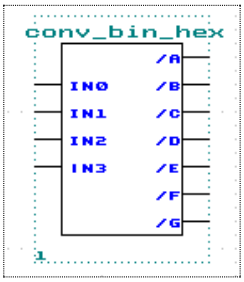


Рисунок 2.5 − Символ проекту CONV_BIN_HEX

На рис.2.6 введений файл CONV_BIN_HEX подано у вигляді групи (шини), що дає можливість краще сприймати вхідні $(in0, in1, in2, in3 \Rightarrow in[3..0])$ й вихідні $(\backslash a, \backslash b, \backslash c, \backslash d, \backslash e, f, \backslash g \Rightarrow ag)$ коди.

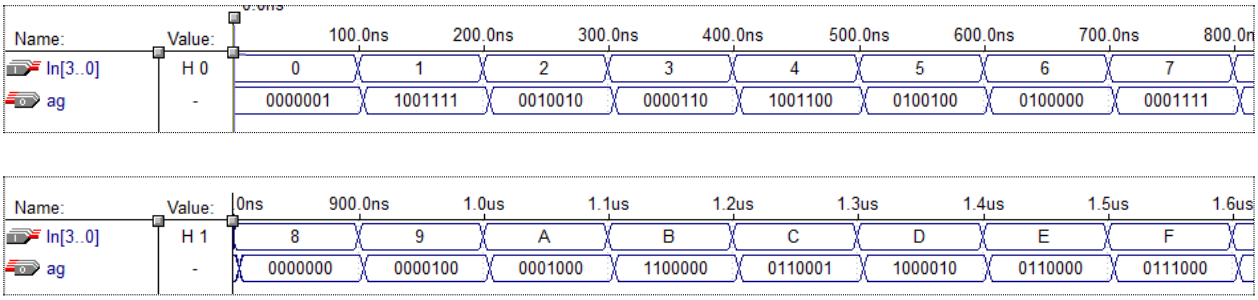


Рисунок 2.6 − Цифрове зображення станів вхідних і вихідних кодів

Для переходу від простого (дротикового) подання до групового (шинного) потрібно виконати наступні дії:

За допомогою лівої кнопки миші виділити змінні відповідної групи й натиснути праву кнопку миші. В результаті появиться випадаюче вікно (рис. 2.7 а), на якому клацнути Enter group, після чого появиться нове випадаюче вікно (рис. 2.7 б), на якому автоматично вказується відповідна шина $In[3..0]$.

У даному випадку, для закінчення дії, достатньо клацнути *OK*. У випадку коли шину (групу) хочемо створити із простих змінних поступаємо як і в попередньому випадку, але після клацання правою кнопкою миші появиться випадаюче вікно без назви групи (рис. 2.7 в). Назву групи задаємо довільним ідентифікатором, наприклад *ag*, і клацаємо *OK*.

Для переходу від групового подання до простого достатньо в стовбчику Name клацнути правою копкою миші на імені групи. В результаті появиться випадаюче вікно (рис.2.7 г), на якому потрібно клацнути на слові Ungroup.

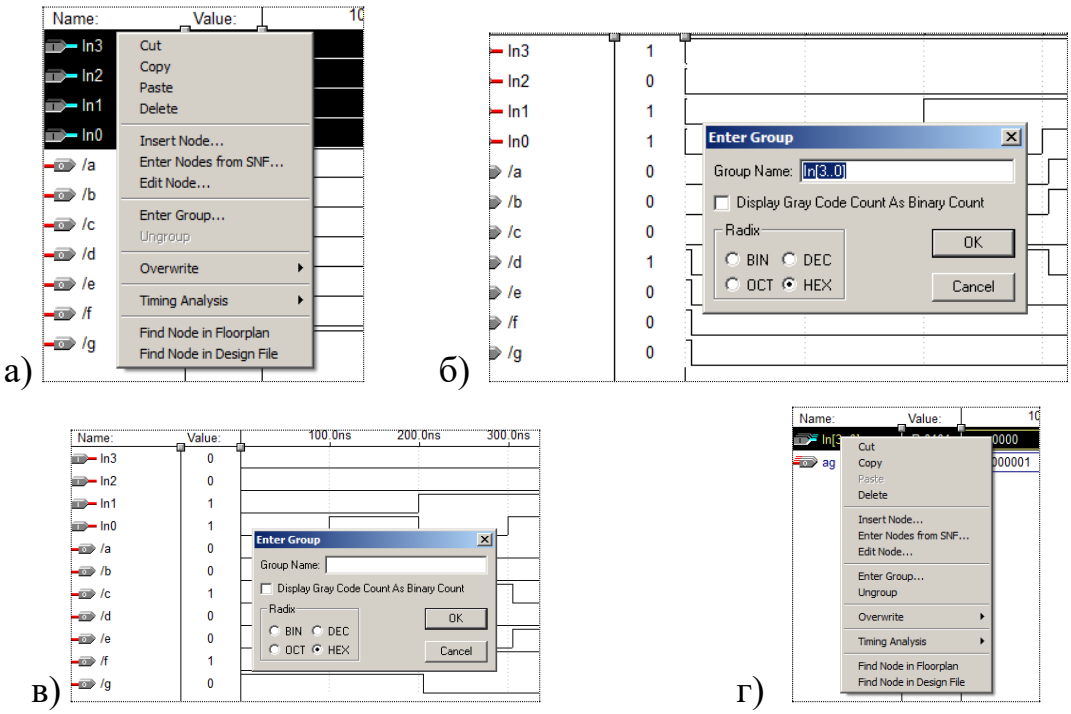


Рисунок 2.7 – Зображення випадаючих вікон для переходу від простого подання даних до групового і навпаки

Після створення символу проекту CONV_BIN_HEX приступаємо до створення (в графічному редакторі) файлу (проекту) в каталозі \ MOD_IND під ім'ям MOD_IND, компілюємо файл проекту всього пристрою в цілому. Як регістр порту використовуємо макрофункцію 74373, а в якості інших елементів – елемент WRITE_PORT і два елементи CONV_BIN_HEX (рис. 2.8).

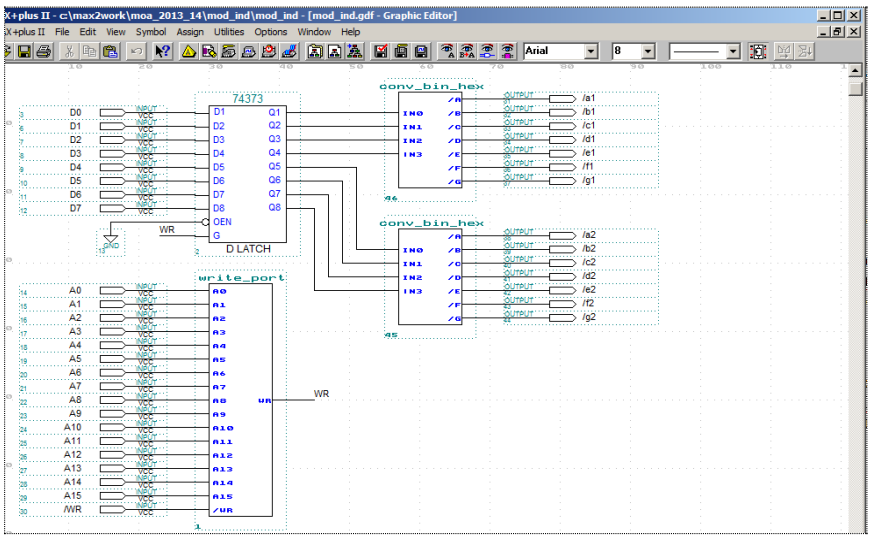


Рисунок 2.8 − Схема проекту пристрою індикації

Порядок відкриття, створення і компіляції файлу проекту верхнього рівня нічим не відрізняється від порядку створення та компіляції файлу WRITE_PORT, який був розглянутий вище.

На рис.2.9 наведено результати моделювання роботи дворозрядного семисегментного індикатора із використання (для зручності аналізу) групового введення/виведення усіх цифрових символів шістнадцяткової та двійковій системах числення.

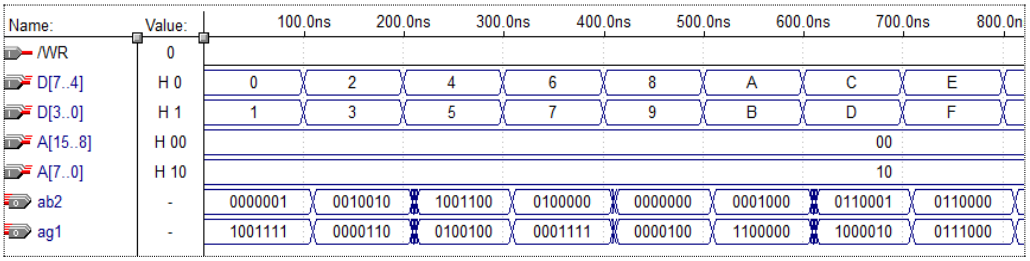


Рисунок 2.9 − Результати моделювання дворозрядного індикатора в цифровому вигляді

3. Призначення типу та виводів ПЛІС

Тип ПЛІС, необхідний для реалізації проекту, може бути вибраний автоматично або призначений вручну. При створенні файлу проекту за замовчуванням встановлений режим автоматичного вибору мінімальної за обсягом ПЛІС, в якій може бути реалізований даний проект. При необхідності тип ПЛІС може бути призначений вручну за допомогою пункту «Device " (меню " Assign» основного меню системи). Після призначення пристрою проект необхідно перекомпілювати.

Виводи ПЛІС також спочатку призначаються автоматично. Після завершення роботи над проектом необхідно закріпити або перепризначити виводи ПЛІС, для того, щоб при можливій подальшій доробці (налагодженні) проекту у складі всього пристрою компілятор не міг змінити їх призначення. Ця операція виконується за допомогою редактора

Floorplan Editor, який запускається або через меню MAX + plus II (в основному меню), або через панель інструментів. Вигляд призначеної ПЛІС сімейства MAX 3000A, а саме ПЛІС типу EPM 3032ALC44 наведено на рис. 3.1.

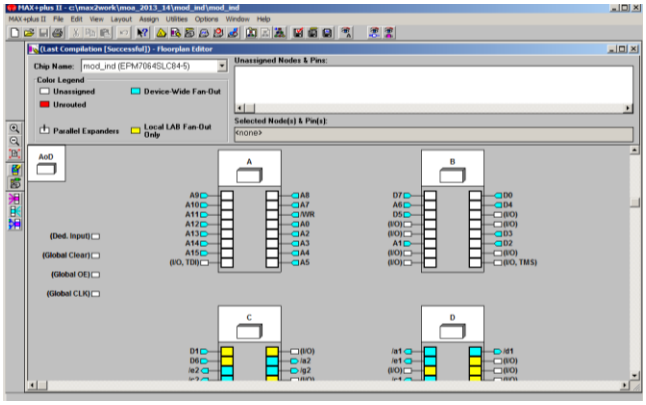


Рисунок 3.1 − Вікно редактора зв'язків Floorplan Editor (при встановленому режимі відображення поточних призначень виводів)

4. Завершення роботи

Для завершення роботи над проектом необхідно провести функціональне моделювання. Для цього необхідно створити вихідний файл з розширенням.SCF, задати тестові (еталонні) стани входів, вибрати перевіряємі виходи і запустити Simulator.

Для створення вихідного файлу необхідно відкрити новий файл, в діалоговому вікні New відмітити пункт Waveform Editor file, встановити в сусідньому полі вікна розширення.SCF і натиснути OK, а потім зберегти його через меню File основного меню. Після цього необхідно пов'язати файл з проектом. Для цього необхідно увійти в меню Node і вибрати пункт Enter Nodes from SNF.

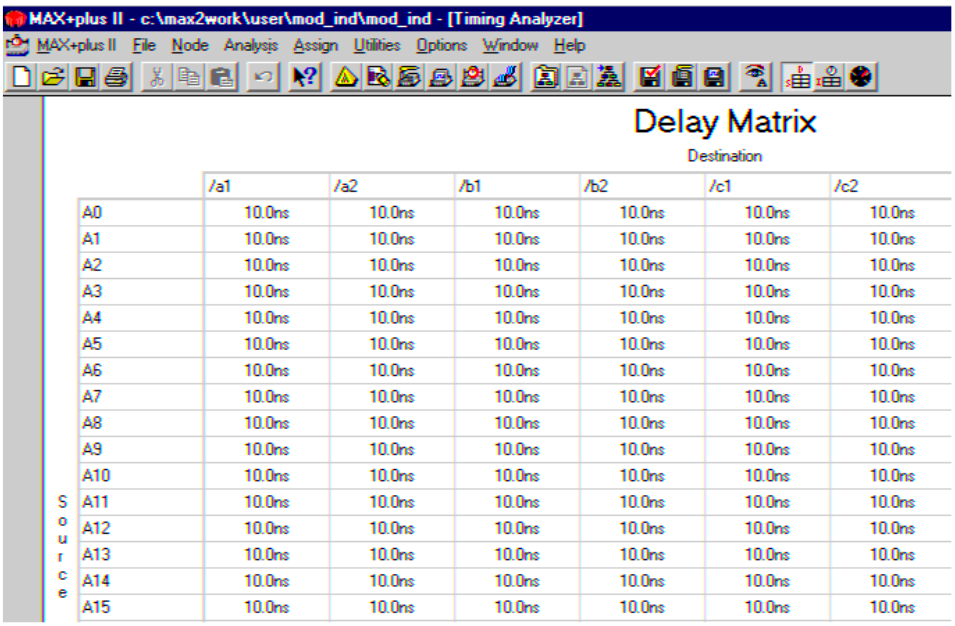
Після натискання кнопки List в лівій панелі діалогового вікна Enter Nodes from SNF з'являється список доступних вузлів (Available Nodes & Groups), виділений синім кольором, який необхідно перенести на праву панель (Selected Nodes & Groups), для чого необхідно натиснути кнопку зі стрілкою, розташовану між панелями. Після натискання кнопки OK у вікні редактора буде створено файл під ім'ям MOD_IND з розширенням.SCF і з'являється готовий шаблон (заготівка) для задання тестових станів входів.

Перед початком введення тестових станів необхідно в порядку, описаному вище, задати тривалість інтервалу моделювання та встановити крок сітки з урахуванням часових параметрів реальних сигналів, які будуть подаватися на вхід ПЛІС в цільовому пристрої.

Самі тестові стани входів вводяться так само, як і при створенні проекту (див. вище).

Simulator може бути запущений або через меню MAX + plus II (в основному меню), або через панель інструментів, при цьому відкривається діалогове вікно з кнопками Start і Open SCF. Для початку моделювання необхідно натиснути кнопку Start, а для перегляду результатів – Open SCF.

Для визначення часових співвідношень використовується додаток Timing Analyzer, який запускається так само, як Simulator. Результати розрахунку часових затримок відображаються у вигляді таблиці Delay Matrix. Один із варіантів вигляду матриці часових затримок наведено на рис.4.1.



	/a1	/a2	/b1	/b2	/c1	/c2
A0	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A1	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A2	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A3	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A4	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A5	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A6	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A7	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A8	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A9	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A10	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A11	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A12	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A13	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A14	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns
A15	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns	10.0ns

Рисунок 4.1 − Фрагмент таблиці «Delay Matrix»

5. Індивідуальні завдання

5.1. Індивідуально реалізувати проект дворозрядного 7-сегментного індикатора з відображенням цифр за допомогою діаграм та цифрового вигляду.

5.2. За допомогою розробленого індикатора відобразити дві цифри згідно варіанту в двох формах: у вигляді діаграм і цифровому вигляді.

5.3. Вивести результат (або фрагмент результату) розрахунку часових затримок для свого варіанту у вигляді таблиці Delay Matrix

5.4. Оформити звіт, який захистити не пізніше як через тиждень після останнього заняття по даній лабораторній роботі.

Варіанти індивідуальних завдань

№ варіанта	Цифри	№ варіанта	Цифри	№ варіанта	Цифри	№ варіанта	Цифри
1	6A	7	A0	13	7A	19	A9
2	5B	8	B1	14	8B	20	B8
3	4C	9	C2	15	9C	21	C7
4	3D	10	D3	16	0D	22	D6
5	2E	11	E4	17	1E	23	E5
6	1F	12	F5	18	2F	24	F4

6. Контрольні запитання

6.1. Наведіть проект пристрою формування сигналу запису в порт.

6.2. Назвіть тип ПЛІС сімейства MAX+PLUS II, яка використовується при проектуванню пристрою формування сигналу запису в порт.

6.3. Яким символом задається низький активний рівень сигналу?

6.4. Наведіть структуру цифрових символів при використанні збудження сегментів нулем.

6.5. Які є способи (за допомогою яких редакторів) реалізації модуля формування цифрових символів? Наведіть приклади відповідних реалізацій.

6.6. Яка макрофункція використовується для реалізації регістру порту? Нарисуйте її символ.

Лабораторна робота № 3

Тема: Опис комбінаційних схем за допомогою мови опису апаратури AHDL

Мета роботи: Набуття практичних навичок проектування комбінаційних схем у програмному комплексі MAX+PLUS II за допомогою мови опису апаратури AHDL.

1. Основні теоретичні відомості

Мова опису апаратури AHDL (ALTERA Hardware Design Language) розроблена фірмою Altera і призначена для опису комбінаційних і послідовнісних логічних пристроїв, групових операцій, цифрових автоматів (state machine) і таблиць істинності з урахуванням архітектурних особливостей ПЛІС фірми Altera. Вона повністю інтегрується із системою автоматизованого проектування ПЛІС MAX+PLUS II. Файли опису апаратури, написані мовою AHDL, мають розширення *.TDF (Text Design File). Для створення TDF-файлу можна використовувати як текстовий редактор системи MAX+PLUS II, так і будь-який інший. Проект, виконаний у вигляді TDF-файлу, компілюється, налагоджується й використовується аналогічно як це робиться графічному редакторі пакету MAX+PLUS II.

Оператори й елементи мови AHDL є досить потужним і універсальним засобом опису алгоритмів функціонування цифрових пристроїв та зручним у використанні. Мова опису апаратури AHDL дає можливість створювати ієрархічні проекти в рамках однієї цієї мови або ж в ієрархічному проекті використовувати як TDF-файли, розроблені мовою AHDL, так і інші типи файлів.

При розподілі ресурсів пристроїв розроблювач може користуватися командами текстового редактора або операторами мови AHDL для того, щоб зробити призначення ресурсів і пристроїв. Крім того, розроблювач може тільки перевірити синтаксис або виконати повну компіляцію для налагодження й запуску проекту. Будь-які помилки автоматично виявляються оброблювачем повідомлень і висвічуються у вікні текстового редактора.

2. Елементи мови AHDL

Зарезервовані ключові слова. Зарезервовані ключові слова використовуються для наступного:

- для позначення початку, кінця й переходів в оголошеннях мови AHDL;
- для позначення визначених констант, наприклад, GND і VCC.

Altera рекомендує всі ключові слова набирати прописними буквами. Список всіх зарезервованих ключових слів мови AHDL наведений у [1].

Булеві вирази. Булеві вирази складаються з операндів, розділених логічними й арифметичними операторами й компараторами і (необов'язково) згруповані за допомогою круглих дужок. Вирази використовуються в булевих рівняннях, а також в інших конструкціях мови, таких як оператори Case та If. Список логічних операторів у двох формах та приклади їх використання наведено в табл.2.1.

Таблиця 2. 1 − Список логічних операторів, використовуваних в мові AHDL

Оператор:	Приклад:	Опис:
! (NOT)	!A (NOT A)	Доповнення (інверсія)
& (AND)	B & C (B AND C)	Логічне І
!& (NAND)	a[3..1] !& b[5..3] a[3..1] NAND b[5..3]	Інверсія логічного І (І-НІ)
# (OR)	A # C (A OR C)	Логічне АБО
!# (NOR)	c[8..5] !# d[7..4] c[8..5] NOR d[7..4]	Інверсія логічного АБО (АБО_НІ)
$ (XOR)	B $ D (B XOR D)	Виключаюче АБО
!$ (XNOR)	x2 !$ x4 (x2 XNOR x4)	Виключаюче АБО-НІ

Застосування логічних операторів до різних типів операндів та їх комбінацій наведено в пункті [1].

Кожний оператор являє собою логічний вентиль із двома входами; виключення становить оператор NOT, що є префіксним інвертором. Для запису логічного оператора можна використовувати його ім'я або символ.

Приклад. Розробити проект, який реалізовує логічну функцію, задану таблицею істинності (табл.2.2) з використанням комплексу MAX+plus II та мовою опису апаратури AHDL.

Таблиця 2.2

A	*B*	*C*	*D*	*F*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Розв'язання. Запишемо функцію, яка реалізовує задану таблицю істинності у вигляді ДНФ

$$Y = \overline{C}\,\overline{D} + AB\overline{D} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}CD + AB\overline{C}\,.$$

За даним логічним рівнянням побудуємо схему з використанням графічного редактора програмного комплексу MAX+plus II. Відповідна схема, результати моделювання схеми та матриця часових затримок, наведені на рис.2.1 – 2.3.

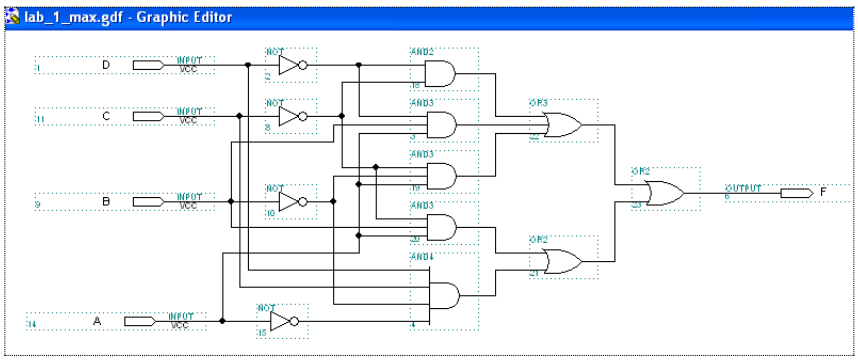


Рисунок 2.1 – Схемна реалізація функції, заданої таблицею 2.2

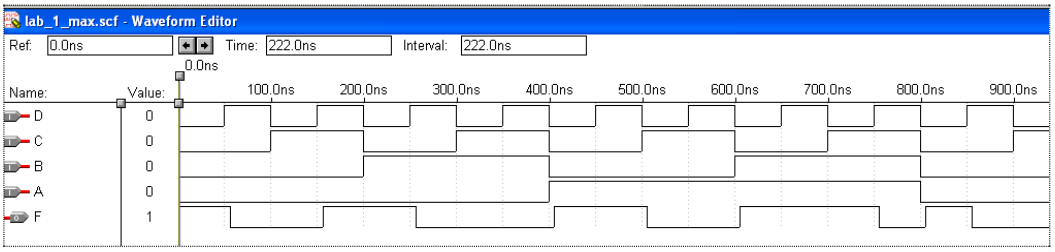


Рисунок 2.2 – Результати моделювання схеми

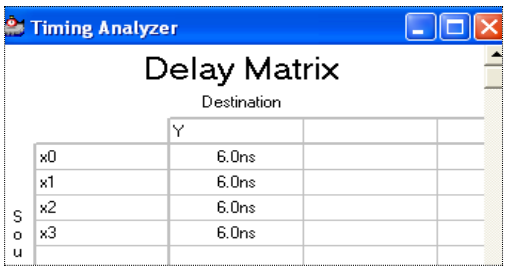


Рисунок 2.3 – Матриця часових затримок

Для розв'язання поставленої задачі за допомогою мови ***AHDL*** використаємо текстовий редактор, який інтегрований в пакет MAX+plus II. Для цього потрібно:

1^0. Завантажити систему проектування MAX+plus II .

2^0. У головному меню вибрати пункт **File**, а у ньому підпункт **New.** У вікні, що з'явилося, вибрати редактор текстових файлів **Text Editor file** і натиснути кнопку **OK.**

Зауважимо, що викликати текстовий редактор можливо і з головного меню, натиснувши **Max+plus II→Text Editor.**

3^0. Викликати перелік шаблонів програм різних комбінаційних логічних схем, натиснувши в головному меню **Templates →AHDL Templates або шляхом клацання в полі текстового редактора** право кнопкою миші з подальшим вибором у спливаючому вікні опції **AHDL Templates**.

4^0. Вибравши розділ SUBDESIGN SECTION (розділ опису інтерфейсу) і натиснувши кнопку OK, дістанемо шаблон вигляду:

```
SUBDESIGN __design_name
(
        __input_name, __input_name          : INPUT = __constant_value;
        __output_name, __output_name : OUTPUT;
        __bidir_name, __bidir_name          : BIDIR;

        __state_machine_name                : MACHINE INPUT;
        __state_machine_name                : MACHINE OUTPUT;
)
```

5⁰. Переробимо даний шаблон для наших потреб та додамо до нього розділ LOGIC SECTION (розділ опису логіки) [1]. У нашому випадку ці два розділи будуть мати вигляд:

```
SUBDESIGN lab_1_ahdl
(
  A,B,C,D     : INPUT ;
  Y           : OUTPUT;
)
BEGIN
Y=!C&!D#A&B&!D#A&!B&!C#!A&!B&C&D#!A&B&!C;
END;
```

6⁰. Далі задаємо ім'я проекту **name** lab_1_ahdl **(File→Project→name)**, до якого прив'язуємо ім'я текстового файлу **(File→Progect→Set Project to Current File)**.

7⁰. Наступні кроки: компіляція, вибір типу ПЛМ, створення файлу симуляції. Симуляція та інші операції виконуються так само, як і в графічному редакторі.

На рис. 2.4 – 2.6 наведено, відповідно: текстовий файл з розширенням .tdf (файл, написаний мовою AHDL) з використанням символів логічних операцій, результати симуляції та матрицю затримок.

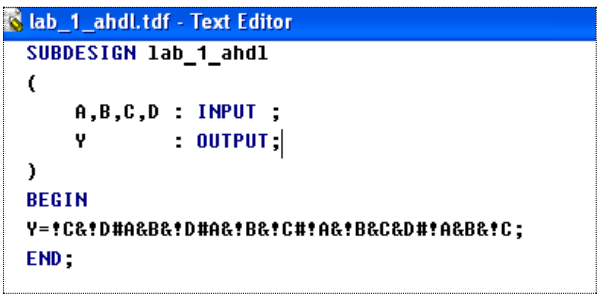


Рисунок 2.4 – Текстовий файл з розширенням .tdf

Звернемо увагу на те, що при використанні символів логічних операторів пропуски між символами є необов’язковими.

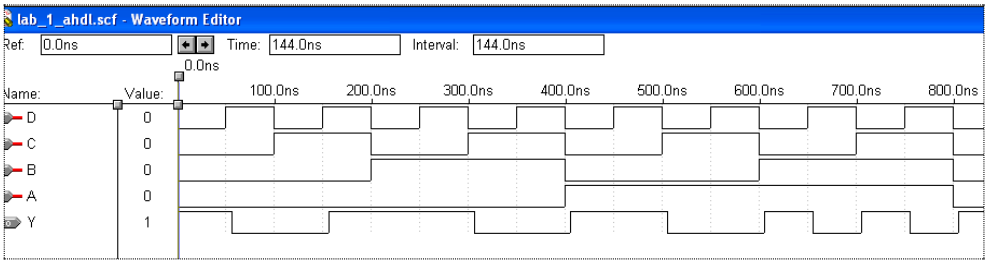


Рисунок 2.5 – Результати моделювання схеми

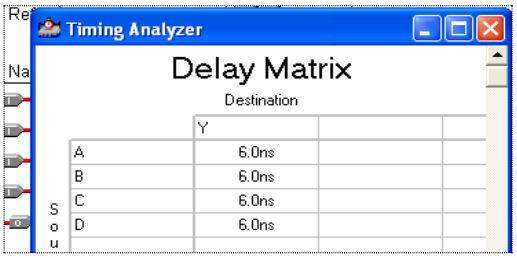
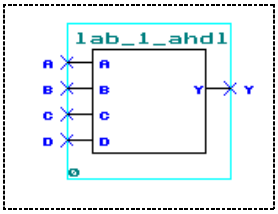


Рисунок 2.6 – Матриця часових затримок

Створення символу в текстовому редакторі здіснюється аналогічно як це робиться в графічному редакторі. А саме, після компіляції проекту, виконавши дії: **file → Create Defaut Symbol,** дістанемо символ



Цю ж саму задачу можна розв’язати і з використанням логічних операторів. Відповідна програма наведена нижче в файлі **lab_1_ahdl_o**

```
SUBDESIGN lab_1_ahdl_o
(
        A,B,C,D         : INPUT ;
        Y               : OUTPUT;
)
BEGIN
Y=not C and not D or A and B and not D or A and not B
and not C or not A and not B and C and D or not A and B and not C;
END;
```

Звернемо увагу на наступне:

а) оператори повинні бути відокремлені один від одного або від змінної хоча би одним пропуском;

б) якщо логічний вираз дуже довгий, то його можна розривати (переносити на наступний рядок) шляхом натискання клавіші ENTER.

Ще один спосіб проектування комбінаційних схем з використанням мови опису апаратури AHDL базується на використанні змінних типу NODE (іменовані лінії зв'язку) [1].

Нижче наведена програма (файл lab_1_ahdl_11), яка реалізовує поставлену задачу з використанням змінних типу NODE. В цій програмі ми використали також змінні типу група [1], які дають можливість задавати вхідні (вихідні) дані у більш компактному і наглядному вигляді. Результати моделювання програми та матрицю затримок наведено на рис. 2.7 – 2.8. Вони повністю співпадають з попередніми результатами.

lab_1_ahdl_11.tdf - Text Editor

```
SUBDESIGN lab_1_ahdl_11
(
    x3,x2,x1,x0 : INPUT ;
        Y       : OUTPUT;
)
VARIABLE
A,B,C,D,E   : NODE;

BEGIN
A=not x1 and not x0;
B=x3 and x2 and not x0;
C=x3 and not x2 and not x1;
D=not x3 and not x2 and x1 and x0;
E=not x3 and x2 and not x1;
Y=A or B or C or D or E;
END;
```

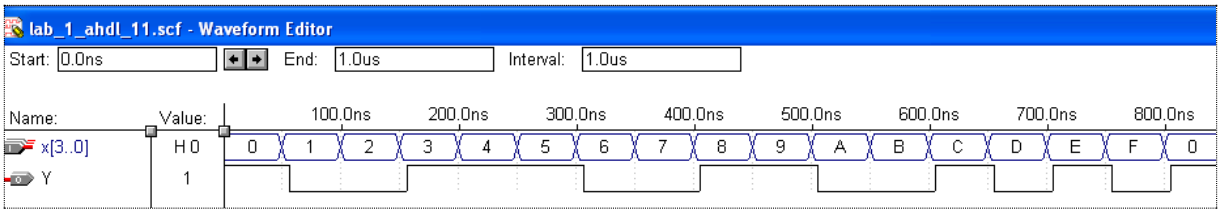


Рисунок 2.7 – Результати моделювання програми

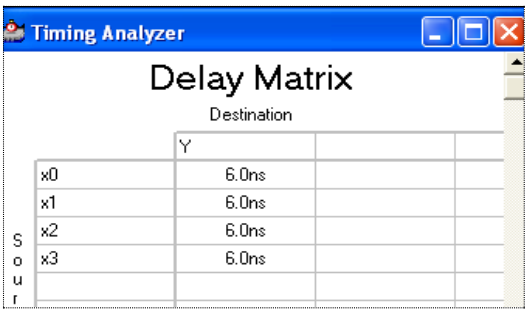


Рисунок 2.8 – Матриця часових затримок

Зауважимо, що якщо ми хочемо побачити проміжні (складові) результати, то це можна зробити задавши одні і ті ж змінні як в розділі опису змінних (тип OUTPUT), так і в розділі опису змінних, використавши змінні типу NODE. Відповідні результати наведено на рис. 2.9 – 2.10.

```
lab_1_ahdl_111.tdf - Text Editor
SUBDESIGN lab_1_ahdl_111
(
    x3,x2,x1,x0 : INPUT ;
    Y,A,B,C,D,E : OUTPUT;
)
VARIABLE
A,B,C,D,E   : NODE;

BEGIN
A=not x1 and not x0;
B=x3 and x2 and not x0;
C=x3 and not x2 and not x1;
D=not x3 and not x2 and x1 and x0;
E=not x3 and x2 and not x1;
Y=A or B or C or D or E;
END;
```

Рисунок 2.9 – Проект, що реалізує поставлену задачу з використанням змінних типу NODE

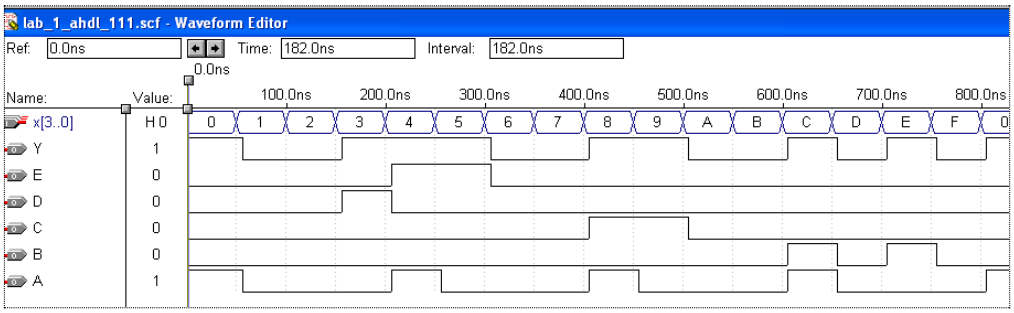


Рисунок 2.10 – Результати моделювання

3. Створення текстового вихідного файлу

Середовище проектування MAX+PLUS II дає можливість створити один або більше текстових вихідних файлів проекту (Text Design Output files (.tdo)), які містять AHDL еквівалент повністю оптимізованої логіки для пристрою, що застосовується в проекті. Файли tdo можна зберігати як TDF файли і повторно використовувати в якості файлів проекту. Крім того компілятор створює також один або більше вихідних файлів призначення і конфігурації (Assignment & Configuration Output Files (.ACO)).

Створений TDO файл можна зберегти як текстовий файл проекту, відредагувати його, визначити його як проект за допомогою команд меню File: Project Name або Project Set Project to Current File і перекомпілювати його. Якщо потрібно зберегти розподіл ресурсів проекту, то потрібно зберегти ACO файл як файл Assignment & Configuration File.

TDO файли полегшують зворотну анотацію і зберігають наявний логічний синтез проекту. Для проекту з декількома пристроями TDO файли дозволяють зафіксувати проект і схему розташування виводів кожного пристрою в проекті.

Для створення TDO файлу проекту потрібно:

У випадаючому вікні MAX + PLUS лівою кнопкою миші клацнути на дадатку Compile, в результаті чого появиться вікно компілятора (рис.3.1). Після цього лівою клавішою клацнути кнопку Processing, в результаті чого появиться випадаюче вікно (рис.3.2), на якому увімкнути опцію Generate AHDL TDO File і клацнути копку Start.

Є й інший варіант. Виконати одну з команд в меню File: Project Save & Compile або Project Save, Compile & Simulate в будь-якому з додатків MAX + PLUS II, а далі увімкнути опцію Generate AHDL TDO File і клацнути копку Start.

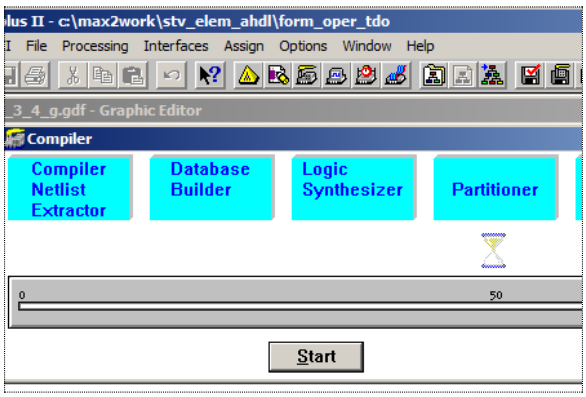


Рисунок 3.1

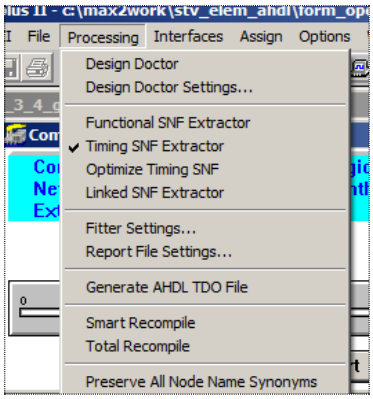


Рисунок 3.2

Для прикладу розглянемо створення текстового вихідного файлу з використанням різних редакторів середовища MAX + PLUS II та різних форм подання логічного опису проекту, логічної функції $F(C,B,A)=F_3(1,2,4,7)=1$, яка на вказаних наборах дорівнює одиниці, а на інших — нулю.

Для розробки того чи іншого проекту спочатку подамо функцію у вигляді таблиці істинності (табл.3.1).

Таблиця 3.1

Входи			Вихід
C	B	A	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

На основі таблиці істинності подамо логічну функціє у вигляді досконалої диз'юнктивної нормальної форми

$$F(C,B,A) = \overline{C}\,\overline{B}A + C\overline{B}A + C\overline{B}\,\overline{A} + CBA. \qquad (3.1)$$

Виконаємо спрощення формули (1):

$$F = \overline{C}\,\overline{B}A + \overline{C}B\overline{A} + C\overline{B}\,\overline{A} + CBA = (\overline{C}\,\overline{B} + CB)\cdot A + (\overline{C}B + C\overline{B})\cdot\overline{A} =$$

$$= A\cdot\overline{\overline{C}\,\overline{B} + CB} + \overline{A}\cdot(\overline{C}B + C\overline{B}) = A\cdot\overline{(C+B)\cdot(\overline{C}+\overline{B})} + \overline{A}\cdot(\overline{C}B + C\overline{B}) =$$

$$= A\cdot\overline{C\overline{C} + B\overline{C} + C\overline{B} + B\overline{B}} + \overline{A}\cdot(\overline{C}B + C\overline{B}) = A\cdot\overline{B\overline{C} + C\overline{B}} + \overline{A}\cdot(\overline{C}B + C\overline{B}) = A\oplus(\overline{C}B + C\overline{B}).$$

На основі одержаного перетворення задану функцію можна подати двома способами:

$$F(C,B,A) = = A\oplus(\overline{C}B + C\overline{B}); \qquad (3.2)$$

$$F(C,B,A) = = A\oplus(\overline{C}B + C\overline{B}) = A\oplus B\oplus C. \qquad (3.3)$$

3.1. Реалізація проекту з використанням графічного редактора

На основі формули (3.1) створюємо проект $f\_3\_4\_g.gdf$ (рис. 13). Результати моделювання наведено на рис. 3.4.

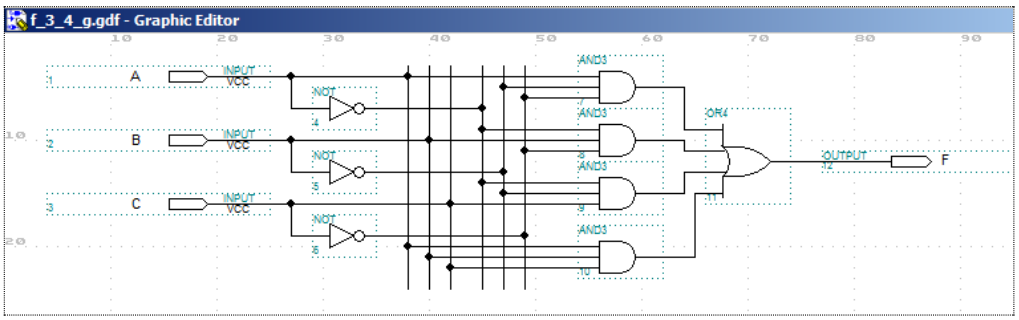


Рисунок 3.3

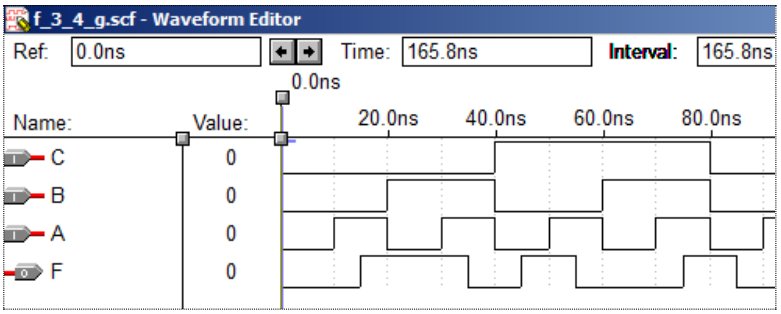


Рисунок 3.4 – Результати моделювання

Створимо відповідний вихідний текстовий файл, виконавши послідовність дій, описаних вище. Для знаходження створеного файлу з розширенням .tdo потрібно натиснути опції Fale → Open і вибрати All files. В результаті появиться випадаюче вікно (рис.3.5), де вибираємо потрібний файл (на рис.3.5 виділений). Вигляд tdo файлу наведено на рис. 3.6. На основі одержаного файлу створюємо проект в текстовому редакторі (рис. 3.7), результати симуляції якого наведено на рис.3.8.

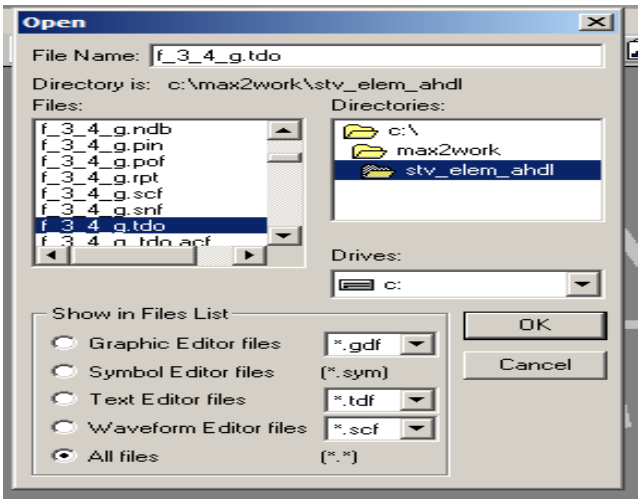


Рисунок 3.5

```
SUBDESIGN 'f_3_4_g'
(
   A              : INPUT;
   B              : INPUT;
   C              : INPUT;
   F              : OUTPUT;
)
VARIABLE
   _EQ001         : NODE;
BEGIN
-- Node name is 'F'
 F      = LCELL( _EQ001 $  A);
  _EQ001 =  B & !C
        # !B &  C;
END;
```

```
SUBDESIGN f_3_4_g_tdo
(
   A              : INPUT;
   B              : INPUT;
   C              : INPUT;
   F              : OUTPUT;
)
VARIABLE
   _EQ001         : NODE;
BEGIN
-- Node name is 'F'
 F      = LCELL( _EQ001 $  A);
  _EQ001 =  B & !C
        # !B &  C;
END;
```

Рисунок 3.6

Рисунок 3.7

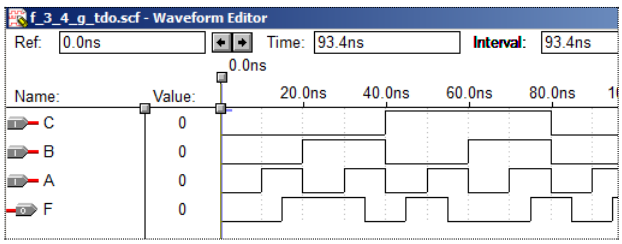


Рисунок 3.8

Зауваження. В одержаному текстовому вихідному файлі (рис. 3.6) примітив LCELL забезпечує найбільше керування. При цьому логічний синтезатор мінімізує усю логіку, яка запускається примітивом LCELL таким чином, щоб її можна звести до однієї комірки.

3.2. Реалізація проекту з використанням текстового редактора

На рис. 3.9 наведено реалізацію проекту **form_oper** з використанням логічних операцій мови AHDL, симуляцію проекту та створений вихідний текстовий файл (TDO). Як бачимо результати симуляції та файл TDO співпадають із відповідними компонентами, одержаними при графічній реалізації. Різниця є лише в порядку розташування змінних в TDO файлах, що немає значення.

```
SUBDESIGN form_oper
(
C,B,A:INPUT;
F:OUTPUT;
)
BEGIN
F=!C&!B&A#!C&B&!A#C&!B&!A#C&B&A;
END;
```

```
SUBDESIGN 'form_oper'
(
   A              : INPUT;
   B              : INPUT;
   C              : INPUT;
   F              : OUTPUT;
)
VARIABLE
   _EQ001         : NODE;
BEGIN
-- Node name is 'F'
 F      = LCELL( _EQ001 $  C);
  _EQ001 = !A &  B
       #  A & !B;
END;
```

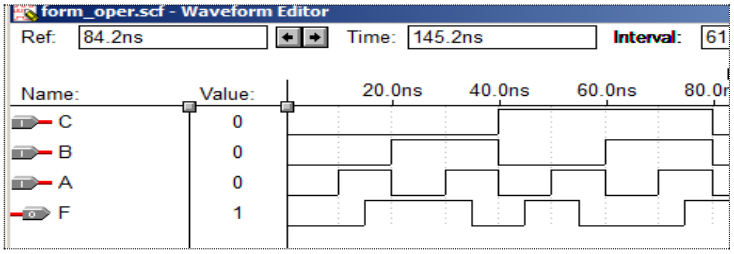


Рисунок 3.9 – Реалізація проекту **form_oper** з використанням логічних операцій мови AHDL

4. Завдання до лабораторної роботи

4.1. Довизначити перемикальну функцію, задану таблицею 4.1 Для цього дві молодші цифри залікової книжки перевести у двійкову систему числення і взяти код у вигляді слова $\alpha_5\alpha_4\alpha_3\alpha_2\alpha_1\alpha_0$. Значення α_i $(i=0-5)$ підставити в табл. 4.1 Наприклад, якщо номер варіанта дорівнює 75 (1001011), то $\alpha_5=0$, $\alpha_4=0$, $\alpha_3=1$, $\alpha_2=0$, $\alpha_1=1$, $\alpha_0=1$.

4.2. Подати одержану функцію у трьох канонічних формах І-АБО-НЕ, І-НЕ/ІНЕ, АБО-НЕ/АБО-НЕ..

4.3. Для одержаних канонічних форм побудувати відповідні комбінаційні схеми користуючись програмним комплексом MAX+plus II та мовою опису апаратури AHDL.

4.4. Виконати моделювання роботи одержаних схем.

4.5. Порівняти результати, отримані різними способами.

4.6. Відповісти на контрольні питання та оформити звіт з виконаної роботи.

Таблиця 4.1

	X_3	X_2	X_1	X_0	Y		X_3	X_2	X_1	X_0	Y
0	0	0	0	0	0	**8**	1	0	0	0	α_2
1	0	0	0	1	1	**9**	1	0	0	1	0
2	0	0	1	0	0	**A**	1	0	1	0	α_3
3	0	0	1	1	α_0	**B**	1	0	1	1	1
4	0	1	0	0	α_1	**C**	1	1	0	0	α_4
5	0	1	0	1	1	**D**	1	1	0	1	0
6	0	1	1	0	0	**E**	1	1	1	0	1
7	0	1	1	1	1	**F**	1	1	1	1	α_5

4.7. Створити текстовий вихідний файл для функції (1) з використанням логічних елементів (операторів) мови AHDL.

4.8. Створити текстовий вихідний файл для функції (1) з використанням логічних операторів мови AHDL.

4.9. Створити текстовий вихідний файл для функції (1) з використанням таблиці істинності мови AHDL.

4.10. Створити текстовий вихідний файл для функції (3) з використанням логічних операторів мови AHDL.

4.11. Створити текстовий вихідний файл для функції (3) з використанням графічного редактора.

5. Контрольні питання

5.1. Назвіть основні елементи мови AHDL, дайте їх коротку характеристику.
5.2. Як задаються логічні оператори в AHDL?
5.3. Назвіть основні розділи мови програмування апаратури AHDL.
5.4. Як задаються змінні типу NODE?
5.5. Як створюється текстовий вихідний файл в середовищі програмного комплексу MAX+plus II.

Лабораторна робота № 4

Тема: Створення та редагування символів і елементів у системі проектування Max+plus II

Мета роботи: набуття практичних навичок створення та редагування символів у програмному комплексі MAX+PLUS II за допомогою мови опису апаратури AHDL; набуття практичних навичок створення замовних елементів за допомогою MEGA WIZARD PLUG-IN MANAGER системи проектування MAX+PLUS II.

1. Створення та редагування символу

Завдання 1. Користуючись мовою опису апаратури AHDL скласти програму та створити символ, який реалізовує логічний елемент LR11 комбінаційного типу. При цьому вимагається, щоб при його використанні в графічному редакторі імена вхідних і вихідних вузлів не виводились. Вигляд такого символу наведено на рис. 1

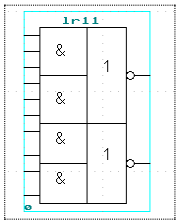


Рисунок 1.1 − Вигляд символу

Виконання завдання 1. Програма, яка реалізовує заданий логічний елемент, написана мовою AHDL з використанням логічних символів має вигляд

```
SUBDESIGN lr11
(
a,b,c,d,e,f,r,s,t,q:INPUT;
y,z:OUTPUT;
)
VARIABLE
        u,v:NODE;
BEGIN
        u=a&b&c;
        v=d&e&f;
        y=!(u#v);
        z=!(r&s#t&q);
END;
```

На рис.1.2 наведено результати моделювання даної програми.

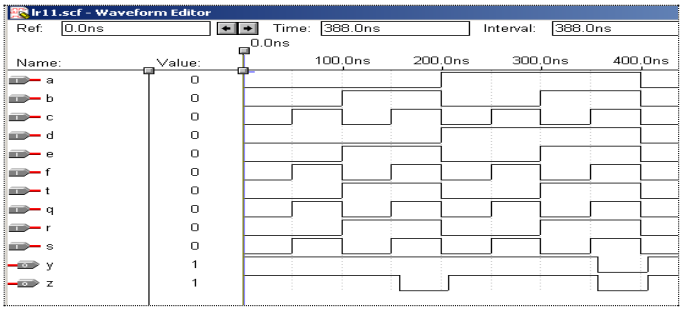


Рисунок 1.2 – Результати моделювання

На рис.1.3 наведено вигляд символу lr11, створеного в автоматичному режимі: File → Create Default Symbol і виведеному на екран за допомогою дій: File → Open → lr11.sym (рис.1.3 а), а також виведеному в поле графічного редактора: Graphic Editor → Enter Symbol → lr11 (рис.1.3 б). Як бачимо, побудований в автоматичному режимі символ не відповідає поставленим вимогам, оскільки залишилися найменування вузлів і не показано, що в елементі використовуються два і три-входові операції типу І/АБО-НЕ.

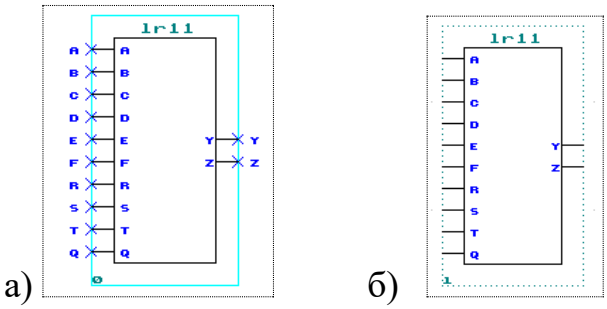


Рисунок 1.3 – Символи

Для одержання потрібного вигляду необхідне редагування символу, яке зводиться до наступного:

1. За допомогою опцій File → Open → lr11.sym викликаємо одержаний символ (рис. 1.3 а).

2. Користуючись інструментами рисування проводимо відповідні лінії розмітки і рисуємо кружечки, переміщаючи, якщо потрібно, вузли і їх імена у відповідні місця.

3. Вставляємо в символ потрібний текст або символи (у даному випадку це & і 1), натискуючи праву клавішу миші у потрібному місці, після чого у випливаючому вікні натискуємо кнопку Enter Text і вводимо відповідний символ.

4. Для того, щоб у графічному редакторі не було видно виклику найменувань вузлів створеного символу потрібно виділити курсором ім’я вузла, яке знаходиться всередині символу (у даному випадку це вузол Y (рис. 1.4), натиснути праву клавішу миші, в результаті чого появиться випливаюче вікно, в якому натиснути **Enter Pinstub**, що викличе появу діалогового вікна **Enter Pinstub (**рис.1.5).

У цьому вікні потрібно у рядку Show Visible Pinstub Name in Graphic Editor вилучити символ галочку. Так робимо з кожним символом, який не хочемо показувати в графічному редакторі.

Звернемо увагу на те, що коли вузли символу не іменовані, то можна використовувати довільні змінні, на відміну від іменованих вузлів, коли потрібно дотримуватись відповідних іменувань.

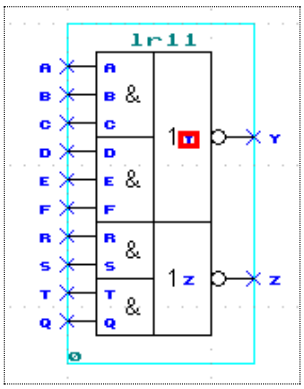


Рисунок 1.4

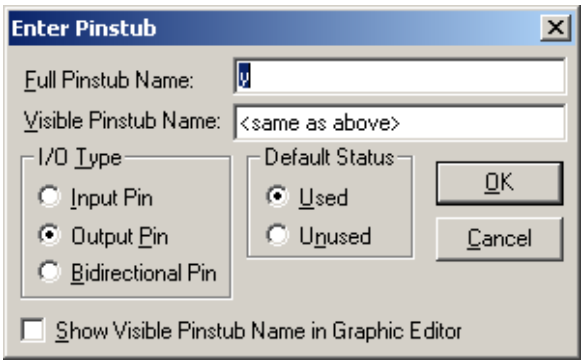


Рисунок 1.5

Після такого редагування символ (у графічному редакторі) буде мати потрібний вигляд (рис. 1.6).

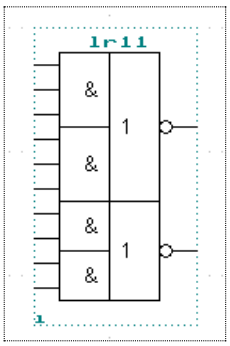


Рисунок 1.6

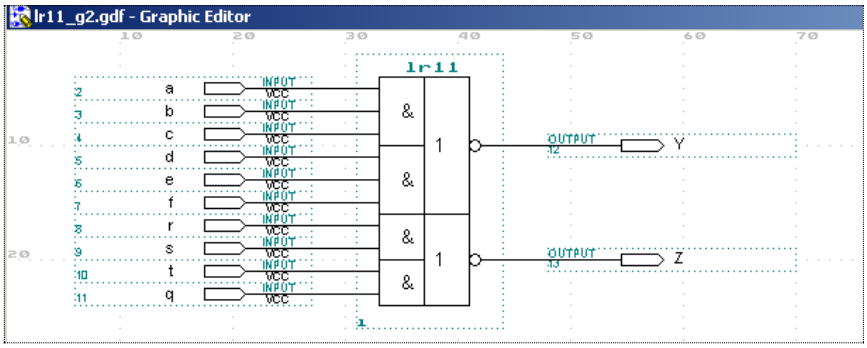


Рисунок 1.7

На рис.1.7 наведено схему з використанням відредагованого символу, а на рис.1.8 – результати моделювання роботи схеми.

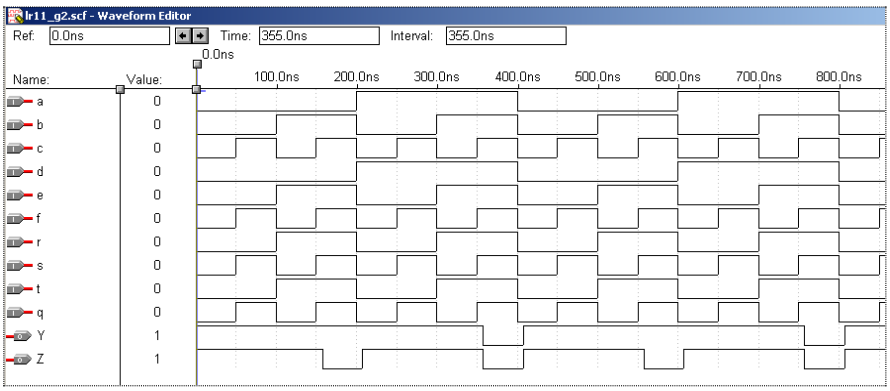


Рисунок 1.8 – Результати моделювання

На рис. 1.9 наведено схему, де замість простих вузлів використовуються шинні. Результати моделювання (рис. 1.10) співпадають з результатами, одержаними вище.

Звернемо увагу на те, що на виходах і входах шин або провідників не обов'язково вказувати імена. А можна їх і вказати, але від цього нічого не міняється.

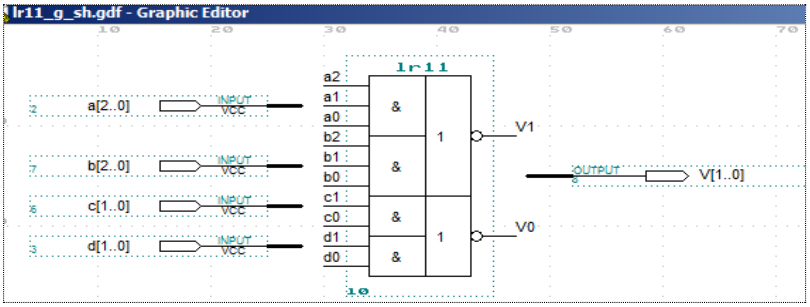


Рисунок 1.9

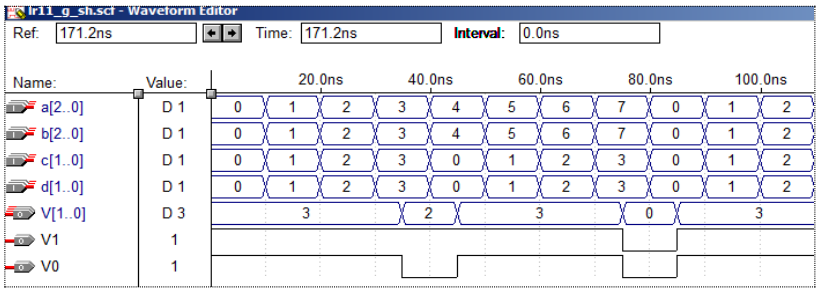


Рисунок 1.10 – Результати моделювання

На рис. 11-13 наведено, відповідно схему елемента *LR11* виконану засобами системи *MAX+plus II,* результати моделювання схеми та символ елемента. Як бачимо результати моделювання та вигляд символу повністю співпадають з результатами (рис. 1.12 − 1.13), одержаними засобами мови *AHDL*. Тому і в цьому випадку можна проводити редагування символу для досягнення поставленої задачі.

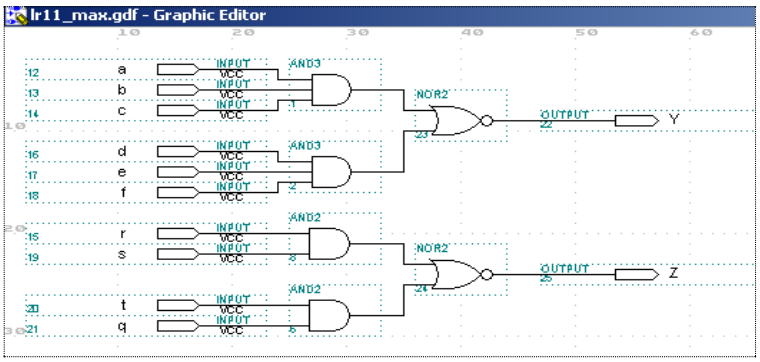


Рисунок 1.11 – Схема елемента *LR11*, виконана засобами системи *MAX+plus II*

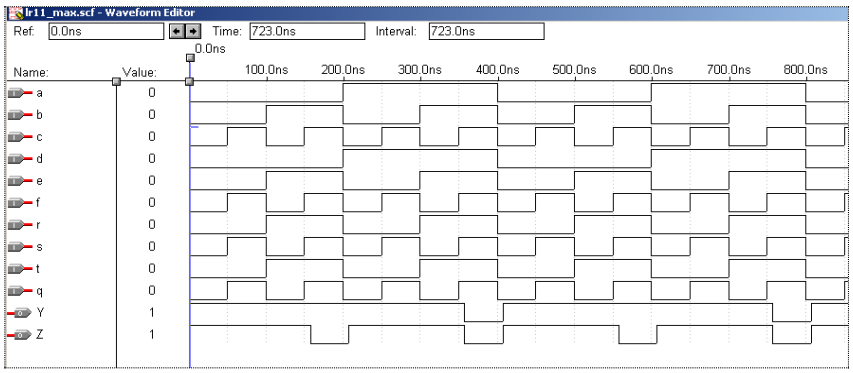


Рисунок 1.12 – Результати моделювання

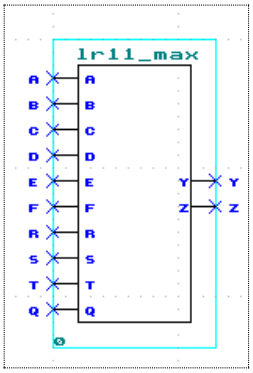


Рисунок 1.13

2. Створення елементів за допомогою MEGA WIZARD PLUG-IN MANAGER системи проектування MAX+PLUS II

Завдання 2. Користуючись засобами MEGA WIZARD PLUG-IN MANAGER розробити проект, який реалізовує логічну функцію, задану таблицею істинності.

Входи			Вихід
C	B	A	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Виконання завдання. Запишемо логічну функціє у вигляді досконалої диз'юнктивної нормальної форми

$$F = \overline{C}\,\overline{B}A + \overline{C}B\overline{A} + C\overline{B}\,\overline{A} + CBA.$$

Виконаємо спрощення

$$F = \overline{C}\,\overline{B}A + \overline{C}B\overline{A} + C\overline{B}\,\overline{A} + CBA = \overline{C}(\overline{B}A + B\overline{A}) + C(\overline{B}\,\overline{A} + BA) =$$

$$= \overline{C}(B \oplus A) + C\overline{B \oplus A} = C \oplus B \oplus A.$$

Таким чином, обчислення функції зводиться реалізації тривходового елемента додавання за модулем 2 (елемент xor3). Оскільки в бібліотеці примітивів MAX+PLUS II відсутній елемент xor3, то його створимо за допомогою засобів Mega Wizard Plug-In Manager.

Створення елемента в Mega Wizard Plug-In Manage здійснюється наступним чином.

Вибираємо MegaWizard Plug-In Manager з меню File. У вікні, що з'явилося, потрібно поставити галочку навпроти Create a new custom megafunction variation і натиснути Next (рис. 1.14).

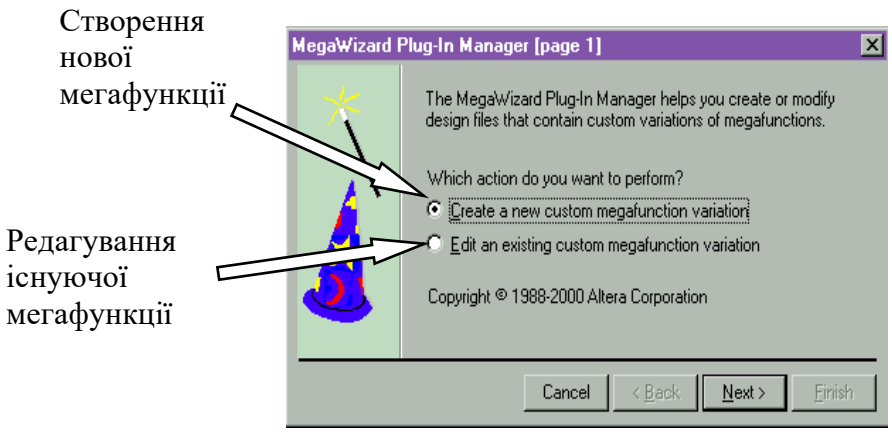


Рисунок 1.14 − Перший крок у створенні мегафункції

У наступному вікні потрібно вибрати мегафункцію, на базі якої буде створений новий елемент, мова опису вихідного файлу і його ім'я.

Виберемо в розділі Схеми (Gates) мегафункцію LPM_XOR і вихідний файл – AHDL. Назвемо мегафункцію xor3 (рис. 1.15). Ім’я xor3 можна ввести у папку, у якій створюємо проект. Потрібний файл можна легко знайти за допомогою кнопки Browse…

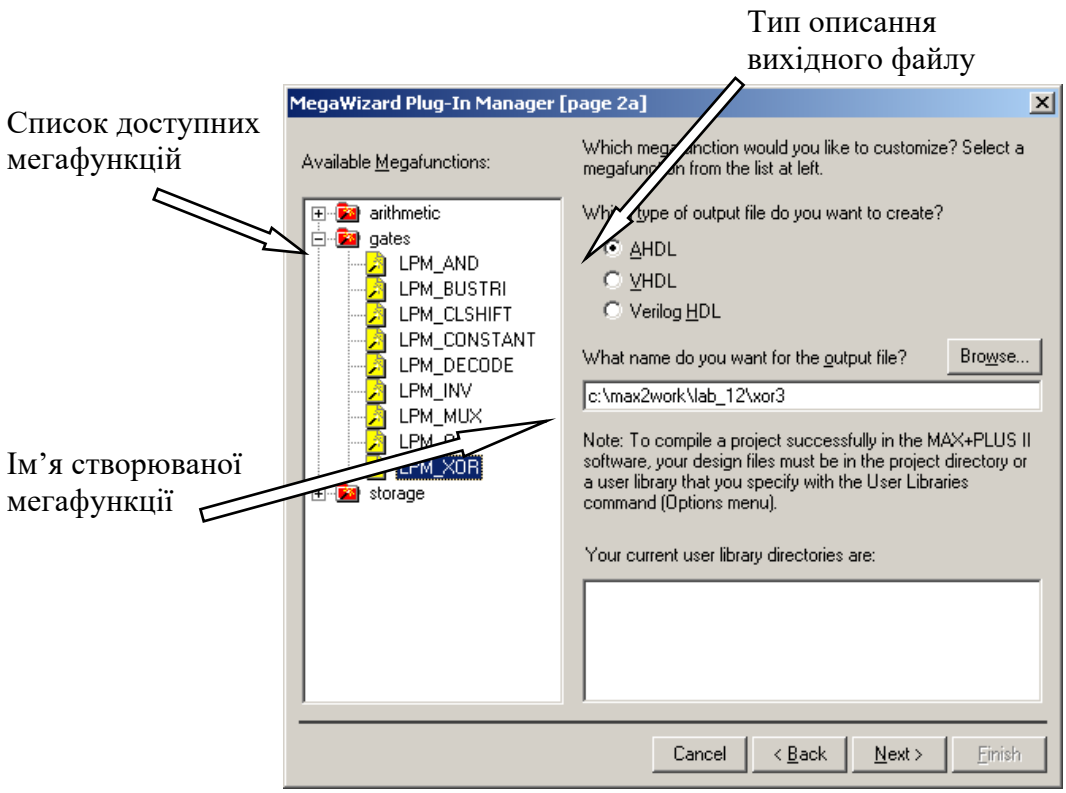


Рисунок 1. 15. – Другий крок у створенні мегафункції

Натиснувши Next – потрапимо на третю сторінку створення мегафункцій (рис. 1.16).

У цьому вікні потрібно вибрати кількість входів для даних, ширину вхідних даних (у бітах), а також можна задати відображення виводів як шин і зробити розміри символу якнайменші.

Укажемо кількість входів - 3, ширину вихідних даних - 1 біт.

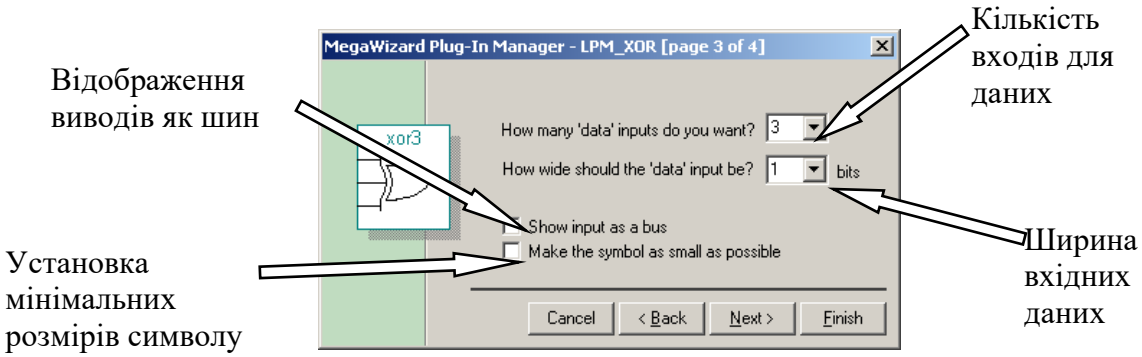


Рисунок 1. 16. – Третій крок у створенні мегафункції

Натиснувши на кнопку Finish - закінчимо створення мегафункції. Якщо натиснути на Next, то потрапимо на четверту сторінку (рис. 1.17), де можна побачити які файли були створені для даної мегафункції й закінчити роботу.

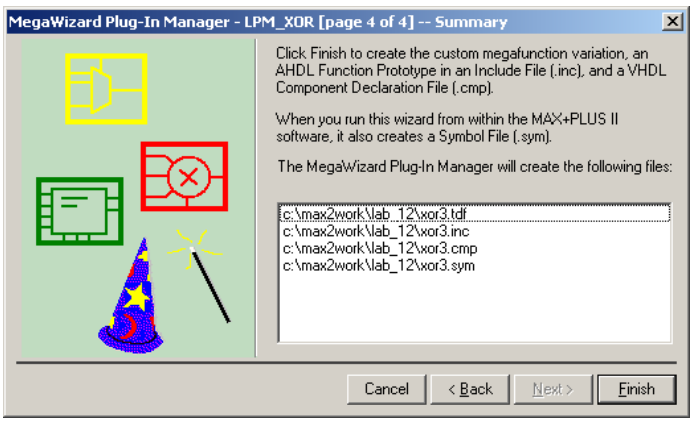


Рисунок 1. 17 – Список файлів, які створені для мегафункції xor3

На рис. 1.18 наведено вигляд створеного символу, якщо його вставляти у вікно графічного редактора

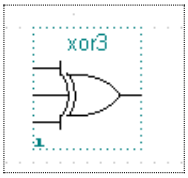


Рисунок 1.18 – Вигляд створеного символу

Зауважимо, що у даному випадку, вузлам присвоюються імена за замовчуванням: вхідні — data0, data1, data2, вихідний — result.

На рис. 1.19 наведено вигляд проекту з використанням створеної мегафункції.

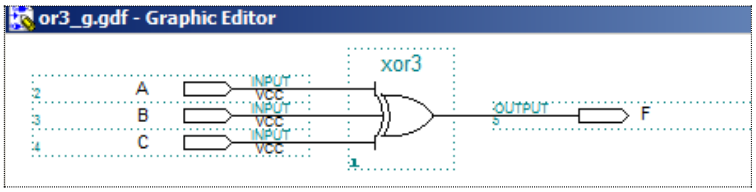


Рисунок 1.19 – Проект з використанням створеної мегафункції

Результати моделювання створеного символу, наведено на рис. 1.20.

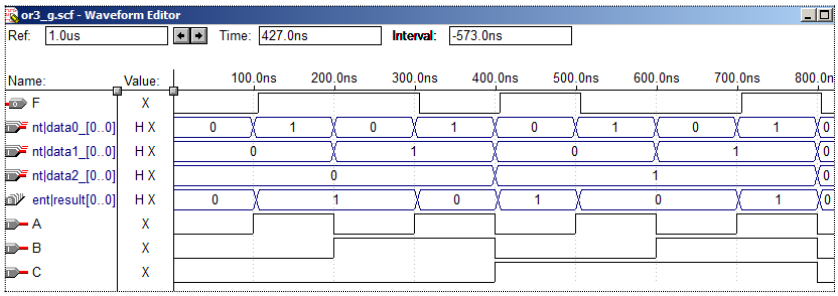


Рисунок 1.20 – Результати моделювання створеної схеми

Перші п’ять рядків видаються при моделюванні автоматично, чотири з яких є імена присвоєні за замовчуванням. Із них чотири, які виведені у цифровій формі, відповідають одновимірним шинам. Функція *Y* виведена у графічному вигляді, що відповідає однопровідному виводу. Останні три рядки виведено у ручному режимі. Вони ілюструють вхідні сигнали у графічному режимі, що теж відповідає однопровідним виводам.

Нижче наведено два текстові файли в AHDL-коді, які формуються системою. Це файл xor.inc —include file і файл xor3 — AHDL програма.

```
        --Файл xor.inc
FUNCTION xor3
(
        data0,
        data1,
        data2
)
RETURNS (result);
```

```
	--Файл xor3.tdf
INCLUDE "lpm_xor.inc";
SUBDESIGN xor3
(
	data0  : INPUT;
	data1  : INPUT;
	data2  : INPUT;
	result  : OUTPUT;
)
VARIABLE
	lpm_xor_component : lpm_xor WITH (LPM_SIZE = 3, LPM_WIDTH =
1);
BEGIN
	result = lpm_xor_component.result[0..0];
	lpm_xor_component.data[2..2][0..0] = data2;
	lpm_xor_component.data[1..1][0..0] = data1;
	lpm_xor_component.data[0..0][0..0] = data0;
END;
```

2. Завдання для самостійної роботи

Завдання 2.1. Створити елемент за допомогою MEGA WIZARD PLUG-IN MANAGER логічної функції відповідно варіанту. Назви функцій наведено в табл. 2.1.

Таблиця 2.1 − Назви функцій

№ в-ту	Функція	№ в-ту	Функція	№ в-ту	Функція	№ в-ту	Функція
1	and5	4	and9	7	and7	10	xor7
2	or9	5	or7	8	or5	11	xor8
3	xor4	6	xor5	9	xor6	12	xor12

Завдання 2.2. Нижче наведені програми набрати в тексовому редакторі програмного комплексу MAX+plus II з використанням логічних операторів, відкомпілювати їх, провести верифікацю та побудувати символи у відповідності із графічним зображення символу.

Таблиця 2.2 − Варіанти завдань

№ варіанту	Програма на моаі AHDL	№ варіанту	Програма на моаі AHDL

1	SUBDESIGN la2_1
	(
	x7,x6,x5,x4,x3,x2,x1,x0:input;
	Y:output;
	)
	BEGIN
	Y=!(x7&x6#x5&x4&x3#x2&x1&x0);
	END;
		2	SUBDESIGN la2_2
			(
			x7,x6,x5,x4,x3,x2,x1,x0:input;
			Y:output;
			)
			BEGIN
			Y=!(x7&x6&x5#x4&x3#x2&x1&x0);
			END;

3	SUBDESIGN la2_3
	(
	x7,x6,x5,x4,x3,x2,x1,x0:input;
	Y:output;
	)
	BEGIN
	Y=!(x7&x6#x5&x4&x3#x2&x1&x0);
	END;
		4	SUBDESIGN la2_4
			(
			x7,x6,x5,x4,x3,x2,x1,x0:input;
			Y:output;
			)
			BEGIN
			Y=!(x7&x6)#x5&x4&x3#x2&x1&x0;
			END;

5	SUBDESIGN ir_1
	(
	X9,X8,x7,x6,x5,x4,x3,x2,x1,x0:input;
	Y:output;
	)
	BEGIN
	Y=!(x9&x8#x7&x6&x5#x4&x3&x2#x1&x0);
	END;
		6	SUBDESIGN ir_2
			(
			X9,X8,x7,x6,x5,x4,x3,x2,x1,x0:input;
			Y:output;
			)
			BEGIN
			Y=!(x9&x8&x7#x6&x5#x4&x3&x2#x1&x0);
			END;

7	SUBDESIGN ir_3
	(
	X9,X8,x7,x6,x5,x4,x3,x2,x1,x0:input;
	Y:output;
	)
	BEGIN
	Y=!(x9&x8&x7#x6&x5&x4&#3&x2#x1&x0);
	END;
		8	SUBDESIGN ir_4
			(
			X9,X8,x7,x6,x5,x4,x3,x2,x1,x0:input;
			Y:output;
			)
			BEGIN
			Y=!(x9&x8&x7#x6&x5#x4&x3#x2&x1&x0);
			END;

Лабораторна робота № 5

Тема: Проектування суматорів у системі проектування Max+plus II з використанням мови опису апаратури AHDL

Мета роботи: Набуття практичних навичок створення проектування напівсуматора, повного двійкового суматора та багаторозрядного суматора у програмному комплексі MAX+PLUS II з використанням мови опису апаратури AHDL.

1. Короткі теоретичні відомості

1.1. Двійковий напівсуматор

Робота напівсуматора описується таблиця істинності (табл. 1.1).

Таблиця 1.1

b	a	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Логічні рівняння

Варіант 1	Варіант 2
$S = \bar{b}a + b\bar{a}$	$S = b \oplus a$
$Co = ba$	$Co = ba$

Програми мовою AHDL з використанням символів логічних операторів наведено в таблиці 1.2

Таблиця 1.2

Варіант 1	Варіант 2

SUBDESIGN hsm
(
a,b:INPUT;
S,Co:OUTPUT;
)
BEGIN
S=!a&b#a&!b;
Co=b&a;
END;
	SUBDESIGN hsm
	(
	a,b:INPUT;
	S,Co:OUTPUT;
	)
	BEGIN
	S=b$a;
	Co=b&a;
	END;

Результати моделювання роботи напівсуматора наведено на рис. 1.1.

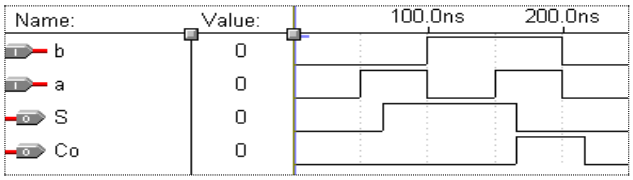


Рисунок 1.1 – Результати моделювання

Схемна реалізація напівсуматора в середовищі MAX+PLUS II. Схема і символ напівсуматора наведено на рис. 1.2.

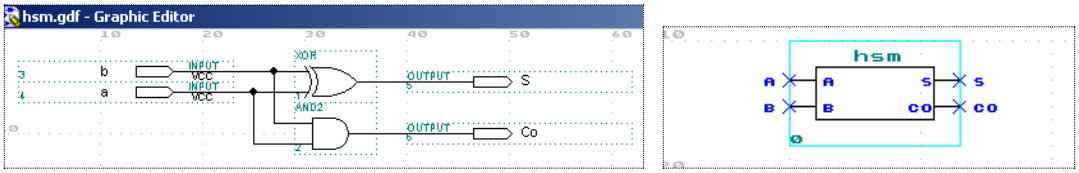


Рисунок 1.2 – Схема і символ напівсуматора

Результати моделювання наведено на рис. 1.3.

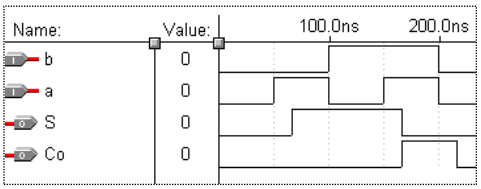


Рисунок 1.3 – Результати моделювання

1.2. Повний двійковий суматор

Робота повного двійкового суматора описується наступною таблицею істинності (табл.1.3).

Таблиця 1.3

Ci	b	a	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

К-карта для S

	001		100
010		111	

К-карта для Co

	101	
011	111	110

Склейки

011	101	110
111	111	111
*11	1*1	11*

Рисунок 1.4 – К-карти

2. Практична реалізація суматорів різної розрядності

Логічні рівняння для повного двійкового суматор складені за таблицею істинності:

$$S = \overline{Ci}\,\overline{b}a + \overline{Ci}b\overline{a} + Ci\overline{b}\,\overline{a} + Ciba, \quad Co = \overline{Ci}ba + Ci\overline{b}a + Cib\overline{a} + Ciba. \qquad (2.1)$$

Виконаємо спрощення рівнянь (1.1). Рівняння для суми виразимо через функцію додавання за модулем 2, а рівняння для переносу спростимо двома способами: методом К-карт (друга формула в (1.2)) і на основі властивостей логічних функцій (друга формула в (1.3)).

$$S = \overline{Ci}\left(b\overline{a} + \overline{b}a\right) + Ci\left(ba + \overline{b}\,\overline{a}\right) = \overline{Ci}\left(b \oplus a\right) + Ci\left(\overline{b \oplus a}\right) = Ci \oplus b \oplus a,$$

$$Co = ba + Cia + Cib; \qquad (2.2)$$

$$S = Ci \oplus b \oplus a; \quad Co = \left(\overline{Ci} + Ci\right)ba + Ci\left(\overline{b}a + b\overline{a}\right) = ba + Ci\left(b \oplus a\right). \qquad (2.3)$$

2.1. Програма реалізації повного двійкового суматора мовою AHDL з використанням логічних операторів або логічних символів (формули (2.1)) наведена в табл.2.1.

Таблиця 2.1

1. Використання логічних операторів	2. Використання логічних символів

SUBDESIGN sm_1
(a,b,Ci:INPUT;
S,Co:OUTPUT;)
BEGIN
S=NOT Ci AND NOT b AND A
OR NOT Ci AND b AND NOT A
OR Ci AND NOT b AND NOT A
OR Ci AND b AND A;
Co=NOT Ci AND b AND a
OR Ci AND NOT b AND a
OR Ci AND b AND NOT a
OR Ci AND b AND a;
END;
	SUBDESIGN sm_1
	(a,b,Ci:INPUT;
	S,Co:OUTPUT;)
	BEGIN
	S=! Ci & ! b & A
	# ! Ci & b & ! A
	# Ci & ! b & ! A
	# Ci & b & A;
	Co=! Ci & b & a
	# Ci & ! b & a
	# Ci & b & ! a
	# Ci & b & a;
	END;

Символ повного двійкового суматора та результати моделювання наведено на рис.2.1.

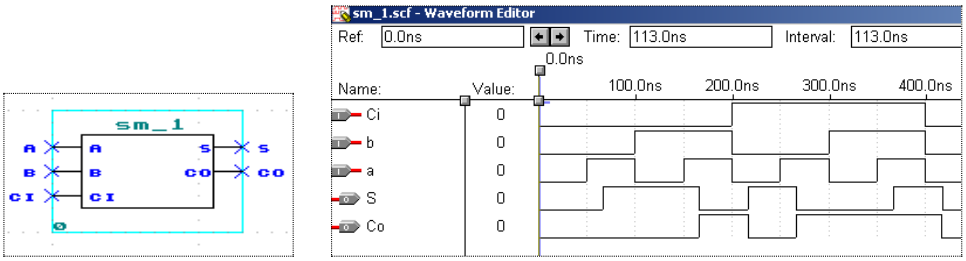


Рисунок 2.1 – Символ повного двійкового суматора та результати моделювання

2.2. Схемна реалізація повного двійкового суматора в середовищі MAX+PLUS II на основі формул (2.1) наведена на рис.2.2.

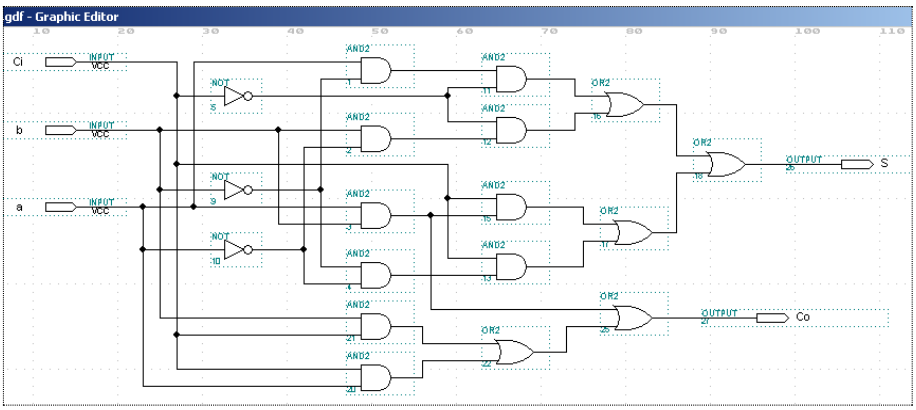


Рисунок 2.2 – Схемна реалізація повного двійкового суматора в середовищі MAX+PLUS II на основі формул (2.1)

Результати моделювання повного двійкового суматора наведено на рис.2.3.

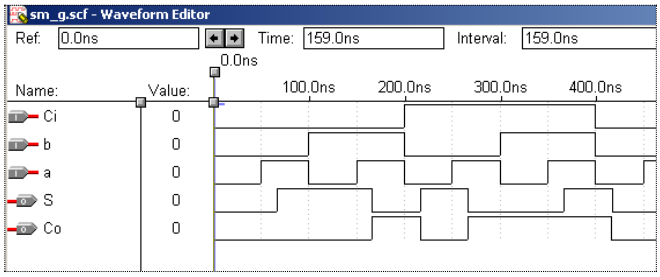


Рисунок 2.3 – Результати моделювання повного двійкового суматора

2.3. Програми реалізації повного суматора мовою AHDL за спрощеними логічними формулами (2.2) з використанням логічних операторів та логічних символів наведено в табл.2.2.

Таблиця 2.2

1. Використання логічних операторів	2. Використання логічних символів

SUBDESIGN sm_2
(a,b,Ci:INPUT;
S,Co:OUTPUT;)
BEGIN
S=Ci XOR b XOR a;
Co= b AND a OR Ci AND b OR Ci AND a;
END;
	SUBDESIGN sm_2
	(a,b,Ci:INPUT;
	S,Co:OUTPUT;)
	BEGIN
	  S=Ci $ b $ a;
	  Co= b & a # Ci & b # Ci & a);
	END;

Результати моделювання повного суматора мовою AHDL за спрощеними логічними формулами (2.2) наведено на рис.2.4.

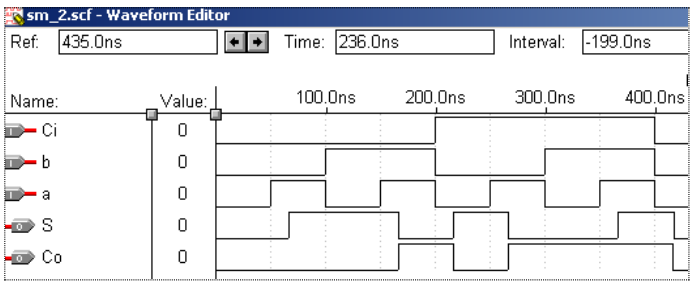


Рисунок 2.4 – Результати моделювання повного суматора мовою AHDL за спрощеними логічними формулами (2.2)

Схемна реалізація повного суматора за спрощеними формулами (2.2) в середовищі MAX+PLUS II та результати моделювання наведено на рис.2.5

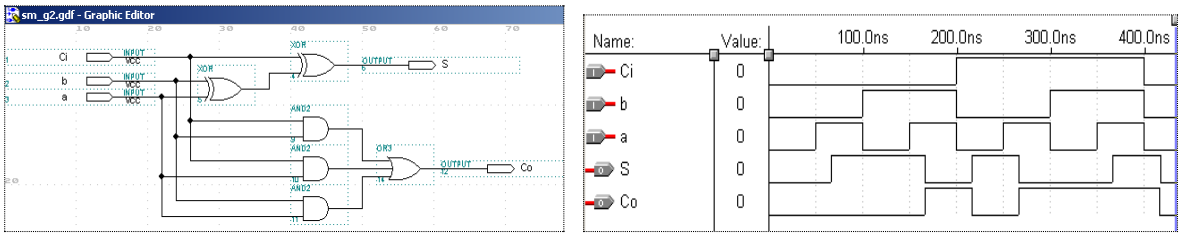


Рисунок 2.5– Схемна реалізація повного суматора за спрощеними формулами (2.2) в середовищі MAX+PLUS II та результати моделювання

2.4. Програми реалізації повного суматора мовою AHDL за спрощеними логічними формулами (2.3) з використанням логічних операторів та логічних символів наведено в табл.2.3.

Таблиця 2.3

1. Використання логічних операторів	2. Використання логічних символів

SUBDESIGN sm_3
(a,b,Ci:INPUT;
S,Co:OUTPUT;)
BEGIN
S=Ci XOR b XOR a;
Co= b AND a OR Ci AND (b XOR a);
END;
	SUBDESIGN sm_3
	(a,b,Ci:INPUT;
	S,Co:OUTPUT;)
	BEGIN
	S=Ci $ b $ a;
	Co= b & a # Ci & (b $ a);
	END;

Результати моделювання повного суматора мовою AHDL за спрощеними логічними формулами (2.3) наведено на рис.2.6.

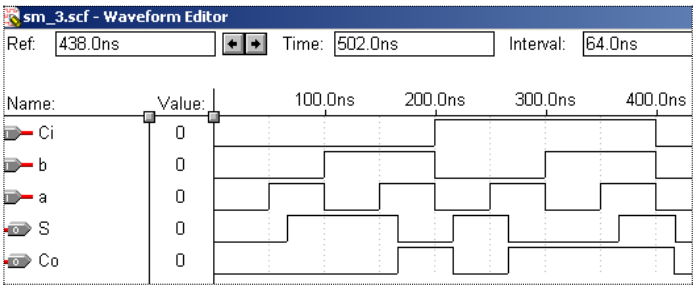


Рисунок 2.6 – Результати моделювання повного суматора мовою AHDL за спрощеними логічними формулами (2.3)

2.5. Схемна реалізація повного суматора за спрощеними формулами (2.3) в середовищі MAX+PLUS II наведена на рис.2.7.

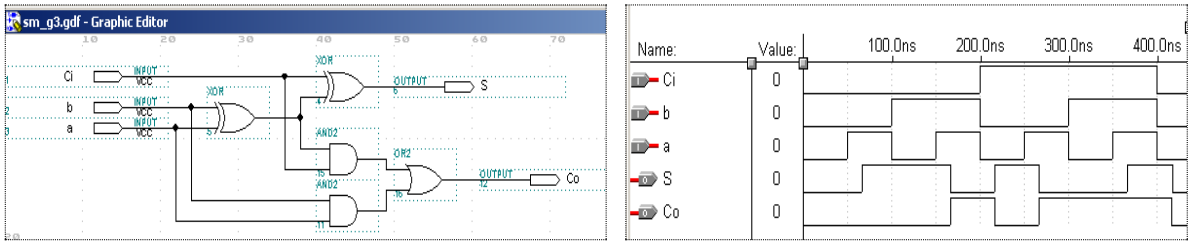


Рисунок 2.7 – Схемна реалізація повного суматора за спрощеними формулами (2.3) в середовищі MAX+PLUS II

2.6. Програмна реалізація повного суматора мовою AHDL з використанням таблиці істинності (табл. 1.3)

```
SUBDESIGN sm_4
(a,b,Ci:INPUT;
S,Co:OUTPUT;)
BEGIN
TABLE
Ci,b,a=>S,Co;
  0,0,0=>0,0;
  0,0,1=>1,0;
  0,1,0=>1,0;
  0,1,1=>0,1;
  1,0,0=>1,0;
  1,0,1=>0,1;
  1,1,0=>0,1;
  1,1,1=>1,1;
 END TABLE;
END;
```

Результати моделювання повного суматора мовою AHDL з використанням таблиці істинності наведено на рис.2.8.

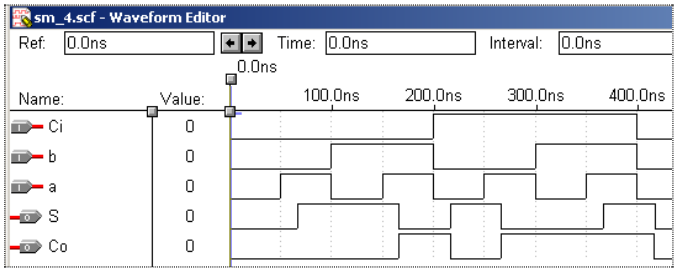


Рисунок 2.8 – Результати моделювання повного суматора мовою AHDL з використанням таблиці істинності

2.7. Схемна реалізація повного двійкового суматора з використанням символу напівсуматора (hsm) і елемента OR2 наведена на рис.2.9.

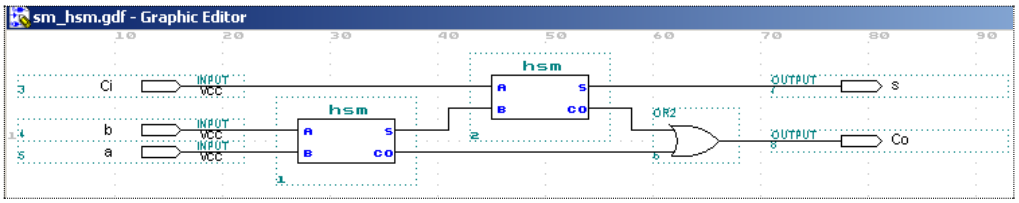


Рисунок 2.9 – Схемна реалізація повного двійкового суматора з використанням символу напівсуматора (hsm) і елемента OR2

Результати моделювання повного двійкового суматора з використанням символу напівсуматора (hsm) і елемента OR2 наведено на рис.2.10.

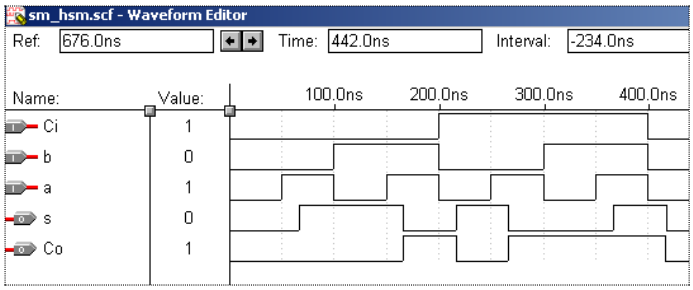


Рисунок 2.10 – Результати моделювання повного двійкового суматора з використанням символу напівсуматора (hsm) і елемента OR2

3. Схемна реалізація трирозрядного двійкового суматора з використанням символів напівсуматора (hsm) і повного двійкового суматора (sm_1)

Схемна реалізація трирозрядного двійкового суматора з використанням символів напівсуматора (hsm) і повного двійкового суматора (sm_1) наведена на рис.2.11.

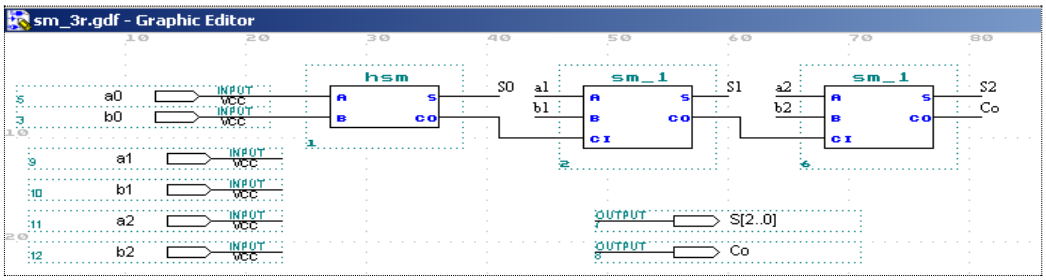


Рисунок 2.11 – Схемна реалізація трирозрядного двійкового суматора з використанням напівсуматора (hsm) і повного двійкового суматора (sm_1)

Результати моделювання трирозрядного двійкового суматора з використанням символів напівсуматора (hsm) і повного двійкового суматора (sm_1) наведено на рис.2.12.

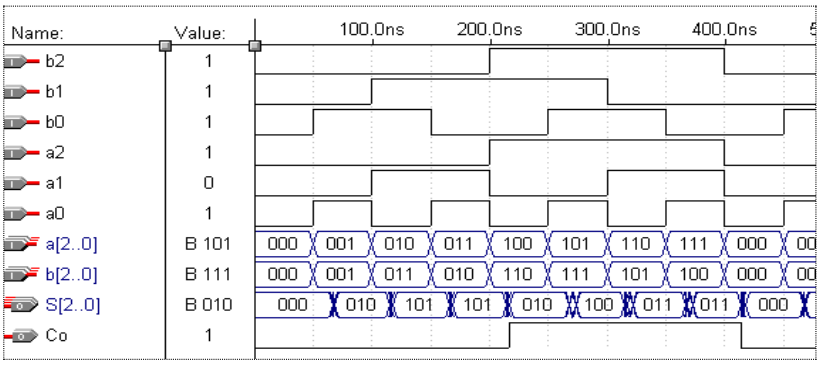


Рисунок 2.12 – Результати моделювання трирозрядного двійкового суматора з використанням символів напівсуматора (hsm) і повного двійкового суматора (sm_1)

4. N-розрядний параметризований двійковий суматор

Суматор даного типу — комбінаційний пристрій, що виконує операцію арифметичного додавання двох N-розрядних двійкових чисел a[], b[] і вхідного коду переносу Ci. Нижче наведено опис такого суматора на вентильному рівні.

Примітив CARRY явно вказує компілятору на необхідність використання схеми ланцюгового переносу логічного елемента HBIC сімейства FLEX. При розміщені HBIC сімейства MAX компілятор ігнорує примітив CARRY.

```
PARAMETERS (N=7);
SUBDESIGN sm_Nr_p
(a[N..0], b[N..0], Ci: INPUT;
        S[N..0], Co: OUTPUT;)
VARIABLE
        C[N+1..0]:NODE;
BEGIN
 C[0]=Ci;
 FOR i IN 0 TO N GENERATE
        S[i]=a[i]$b[i]$C[i];
        C[i+1]=CARRY(a[i]&b[i]#C[i]&(a[i]#b[i]));
 END GENERATE;
  Co=C[N+1];
END;
```

Результати моделювання параметризованого двійкового суматора при $N = 7$ наведено на рис.2.13.

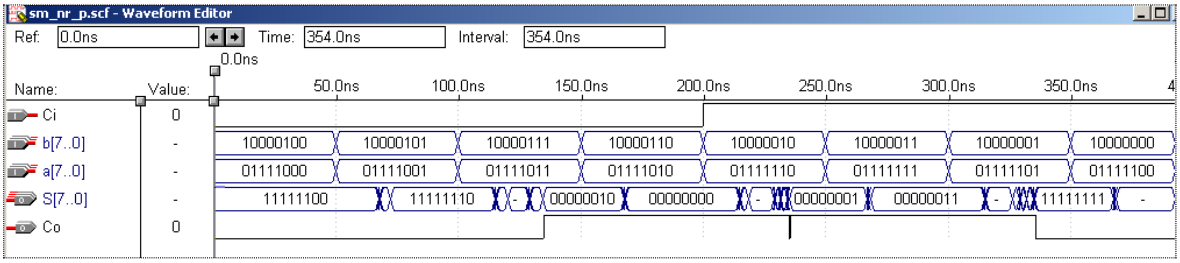


Рисунок 2.13 – Результати моделювання параметризованого двійкового суматора при $N = 7$

Результати моделювання параметризованого двійкового суматора при $N = 2$ наведено на рис.2.14.

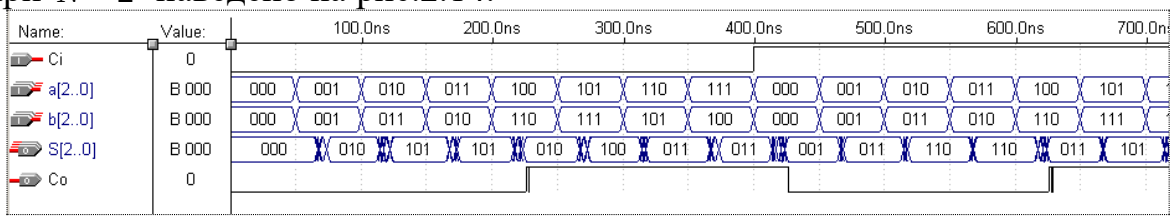


Рисунок 2.14 – Результати моделювання параметризованого двійкового суматора при $N = 2$

5. Восьмирозрядний двійковий суматор, реалізований на основі операції додавання шин

```
SUBDESIGN add_8r
(
A[7..0],B[7..0],Ci:INPUT;
S[7..0],Co:OUTPUT;)
BEGIN
(Co,S[])=(B"0",A[])+(B"0",B[])+(B"00000000",Ci);
END;
```

Результати моделювання восьмирозрядного двійкового суматора, реалізованого на основі операції додавання шин наведено на рис.2.15.

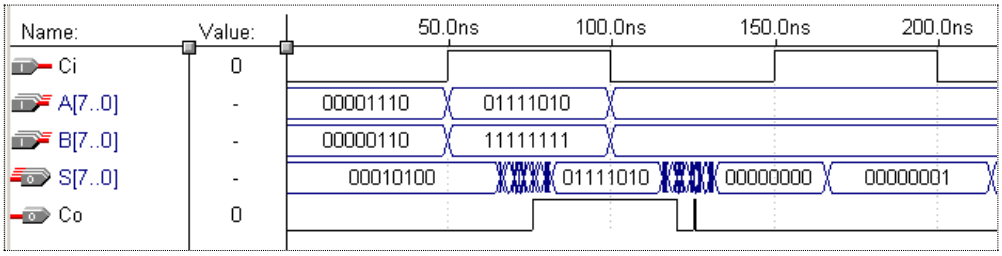


Рисунок 2.15 – Результати моделювання восьмирозрядного двійкового суматора, реалізованого на основі операції додавання шин

6. N-розрядний параметризований двійковий суматор, реалізований на основі операції додавання шин

```
PARAMETERS (N=7);
SUBDESIGN add_8r_p
( A[N..0],B[N..0],Ci:INPUT;
S[N..0],Co:OUTPUT; )
VARIABLE
zero[N..0]:NODE;
BEGIN
zero[]=0;
(Co,S[])=(B"0",A[])+(B"0",B[])+(zero[],Ci);
END;
```

Результати моделювання N-розрядного параметризованого двійкового суматора, реалізованого на основі операції додавання шин наведено на рис.2.16.

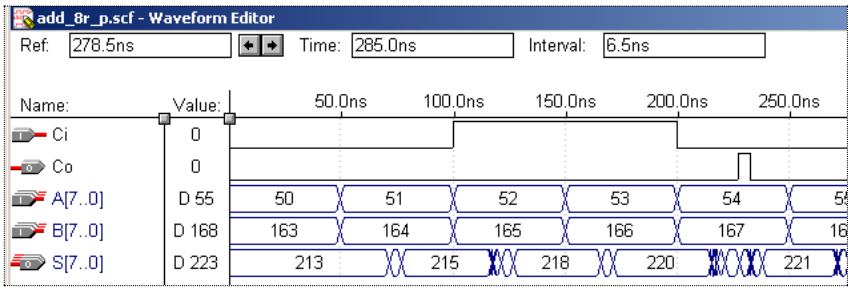


Рисунок 2.16 – Результати моделювання N-розрядного параметризованого двійкового суматора, реалізованого на основі операції додавання шин

7. Перетворення прямого коду в обернений і навпаки

Параметризована програма переведення прямого коду в обернений. В наведеній програмі знак числа задається в N+1-му розряді.

```
PARAMETERS (N=7);
SUBDESIGN Pr_ob_1
( A[N+1..0]:INPUT;
 B[N+1..0]:OUTPUT; )
BEGIN
 IF A[N+1]==VCC
THEN
          B[N+1]=VCC;
          B[N..0]=!A[N..0];
        ELSE
          B[]=A[];
END IF;
END;
```

Результати моделювання параметризованої програми переведення прямого коду в обернений наведено на рис.2.17.

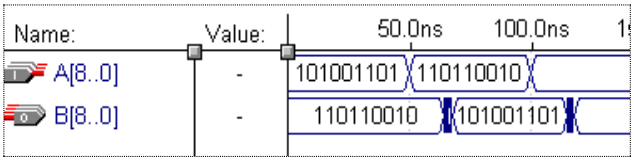


Рисунок 2.17 – Результати моделювання параметризованої програми переведення
прямого коду в обернений

8. Перетворення прямого коду в доповняльний і навпаки

Параметризована програма переведення восьмирозрядного прямого коду в доповняльний. В наведеній програмі знак числа задається в N+1-му розряді.

```
PARAMETERS (N=7);
SUBDESIGN Pr_dop_1
( A[N+1..0]:INPUT;
 B[N+1..0]:OUTPUT; )
BEGIN
 IF A[N+1]==VCC
THEN
          B[N+1]=VCC;
          B[N..0]=!A[N..0]+B"00000001";
        ELSE
           B[]=A[];
END IF;
END;
```

Результати моделювання параметризованої програми переведення прямого коду в доповняльний наведено на рис.2.18.

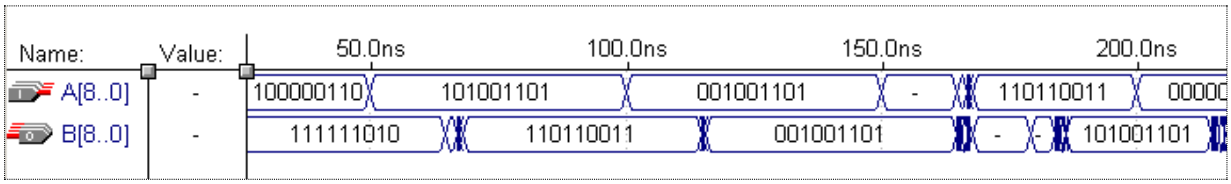


Рисунок 2.18 – Результати моделювання параметризованої програми переведення
доповняльний коду в обернений

9. Восьмирозрядний двійковий віднімач, побудований з використанням доповняльного коду

```
PARAMETERS (N=7);
SUBDESIGN sub_8r_1
( A[N+1..0], B[N+1..0]: INPUT;
Sp[N+1..0], Sg, Sd[N+1..0]: OUTPUT; )
VARIABLE
R[N+1..0],Co: NODE;
BEGIN
 (Co, R[])=(B"0",A[])+(B"0",!B[])+B"0000000001";
IF R[N+1]==B"1" THEN
Sg=B"1";
Sp[]=!R[]+B"000000001";
Sp[N+1]=B"1";
ELSE
Sg=B"0";
Sp[]=R[];
END IF;
Sd[]=R[];
END;
```

Результати моделювання восьмирозрядного двійкового віднімача, побудованого з використанням доповняльного коду наведено на рис.2.19.

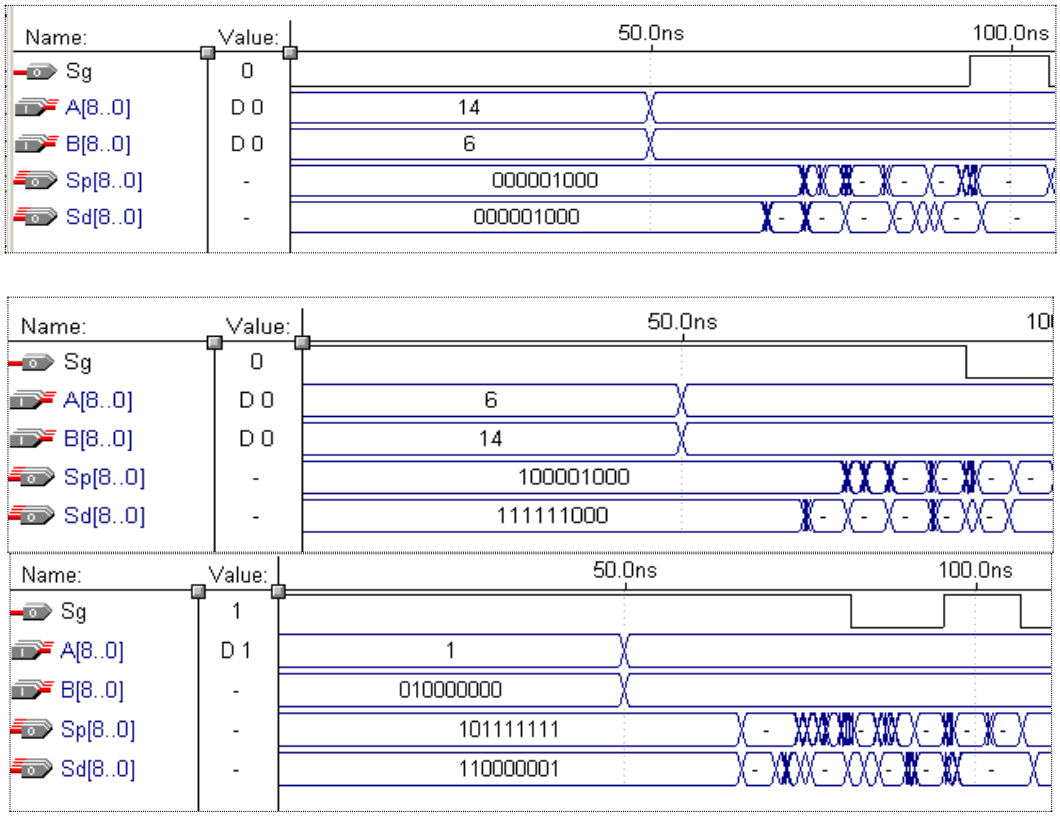


Рисунок 2.19 – Результати моделювання восьмирозрядного двійкового віднімача, побудованого з використанням доповняльного коду

10. Восьмирозрядний двійковий віднімач, реалізований на основі операції віднімання

```
PARAMETERS (N=7);
SUBDESIGN sub_8r_p
( A[N..0],B[N..0],Ci:INPUT;
S[N..0],Co:OUTPUT; )
VARIABLE
zero[N..0]:NODE;
BEGIN
zero[]=0;
(Co,S[])=(B"0",A[])-(B"0",B[])+(zero[],Ci);
END;
```

Результати моделювання восьмирозрядного двійкового віднімача, реалізованого на основі операції віднімання наведено на рис.2.20.

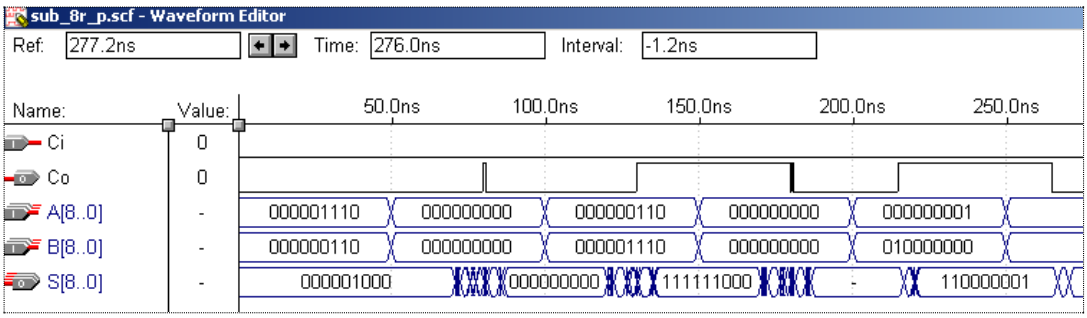


Рисунок 2.20 – Результати моделювання восьмирозрядного двійкового віднімача, реалізованого на основі операції віднімання

11. Побудова чотирирозрядного двійкового суматора з використанням Include файлу

Побудова Include файлу повного двійкового суматора

Завдання. Створити Include файл повного двійкового суматора, робота якого описується програмою

```
        SUBDESIGN FS
(
a, b, ci : INPUT;
        s, co    : OUTPUT;
 )
BEGIN
        s = a $ b $ ci;
        co = a & b # a & ci # b & ci;
END;
```

Результати моделювання Include файлу повного двійкового суматора наведено на рис.2.21.

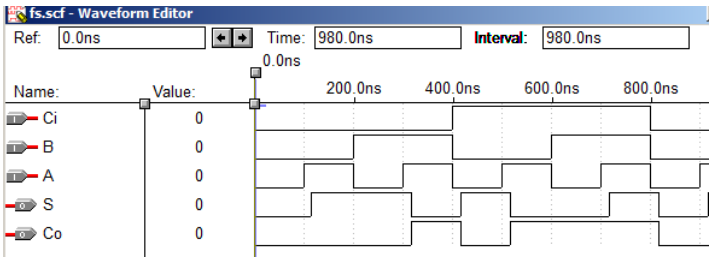


Рисунок 2.21 – Результати моделювання Include файлу повного двійкового суматора

Для створення Include файлу потрібно відкомпілювати проект (програму) і в опції **File** лівою клавішею миші клацнути на **Create Default Include File.** В результаті одержимо файл **fs.inc**, відкривши який, знайдемо відповідну функцію:

```
FUNCTION fs (a, b, ci)
  RETURNS (s, co);
```

Вигляд заголовка функції дає інформаціє про вхідні і вихідні вузли, що використовується при звертанні до **Include** файлу в логічному блоці програми.

Звертання до файлу здійснюється в тексті програми за допомогою запису

```
INCLUDE "fs.inc";
```

Один із варіантів звернення до **Include** файлу розглянемо на прикладі проектуваня чотирирозрядного двійкового суматора.

12. Проектування чотирирозрядного двійкового суматора з використанням INCLUDE файлу однорозрядного суматора

```
--FUNCTION fs (a, b, ci)
  --RETURNS (s, co);
INCLUDE "FS.inc";
SUBDESIGN sum_4
(  A[3..0], B[3..0]      : INPUT;
        SS[4..0]         : OUTPUT; )
VARIABLE
        Sum[3..0]: fs;
BEGIN
        Sum[0].( a, b, ci ) = ( A[0], B[0], GND );
        SS[0] = Sum[0].s;
        FOR i IN 1 to 3 GENERATE
                Sum[i].( a, b, ci ) = ( A[i], B[i], Sum[i-1].co );
                SS[i] = Sum[i].s;
        END GENERATE;
        SS[4] = Sum[3].co;
END;
```

Результати моделювання чотирирозрядного двійкового суматора з використанням INCLUDE файлу однорозрядного суматора наведено на рис.2.22.

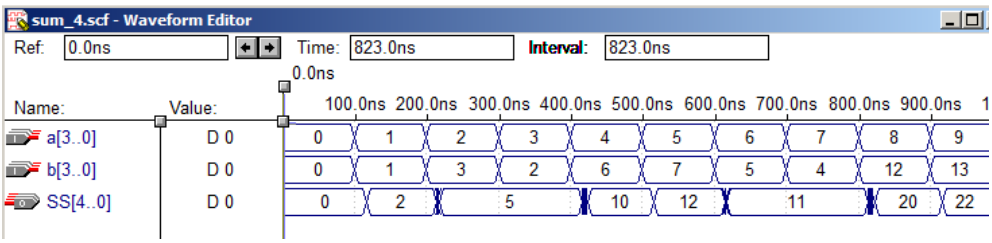


Рисунок 2.22 – Результати моделювання чотирирозрядного двійкового суматора з використанням INCLUDE файлу однорозрядного суматора

Завдання для самостійної роботи

1. Індивідуально проробити усі наведені проекти. У варіативних випадках (1 або 2) студенти з непарним номером у журналі академгрупи виконують варіант 1, а студенти з парним номером ─ варіант 2.
2. Користуючись таблицею часових затримок проаналізувати роботу подібних проектів.
3. Провести власні дослідження роботи віднімачів, реалізованих за допомогою доповняльного коду.

Контрольні питання

1. Наведіть основні способи створення проектів двійкових напівсуматорів.
2. Наведіть основні способи створення проектів повних однорозрядних двійкових суматорів.
3. Напишіть програму реалізації N-розрядного параметризованого двійкового суматора.
4. Напишіть програму реалізації восьмирозрядного двійкового суматора, реалізованого на основі операції додавання шин.
5. Напишіть програму реалізації N-розрядного двійкового суматора, реалізованого на основі операції додавання шин.
6. Як реалізовується проект перетворення прямого коду в обернений і навпаки?
7. Як реалізовується проект перетворення доповняльного коду в обернений і навпаки?
8. Напишіть програму реалізації восьмирозрядного двійкового віднімача з використанням доповняльного коду.
9. Напишіть програму реалізації восьмирозрядного двійкового віднімача з використанням операції віднімання.
10. Назвіть основні дії, які потрібно виконати для створення чотирирозрядного двійкового суматора з використанням INCLUDE файлу.

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