МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до практичних занять
з дисципліни
„КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРИСТРОЇВ ЦИФРОВОЇ ЕЛЕКТРОНІКИ"

для студентів спеціальності
176 „Мікро- та наносистемна техніка“,
освітня програма: „Мікро- та наноелектронні прилади і пристрої“
першого (бакалаврського) рівня вищої освіти
денної й заочної форм навчання

2025

Методичні вказівки до практичних занять з дисципліни "Комп'ютерне моделювання та проєктування пристроїв цифрової електроніки" для студентів спеціальності 176 „Мікро- та наносистемна техніка“, освітня програма: „Мікро- та наноелектронні прилади і пристрої“ першого (бакалаврського) рівня вищої освіти денної й заочної форм навчання / Укл.: Ніна НАГОРНА. – Запоріжжя: НУ «Запорізька політехніка», 2025. – 52 с.

Укладач: Ніна НАГОРНА, ст. викладач,

Рецензент: Валентин ПОГОСОВ, проф., д-р фіз.-мат. наук

Відповідальний за випуск: Андрій КОРОТУН, канд. фіз.-мат. наук, професор

Затверджено
на засіданні кафедри
інформаційної безпеки та
наноелектроніки

Протокол № 5
від “ 22 ” січня 2025 р.

Рекомендовано до видання
НМК ФІБЕК
Протокол № 7
від “ 24 ” лютого 2025 р.

ЗМІСТ

1 ПРАКТИЧНЕ ЗАНЯТТЯ № 1
„МОДЕЛІ ПОСЛІДОВНІСНИХ СХЕМ“

Мета роботи - ознайомлення з формальним підходом до опису і проєктуванню послідовнісних схем довільного типу, розгляд прикладів проєктування схем з використанням моделі Мура.

1.1 Теоретичні відомості

1.1.1 Моделі синхронної послідовнісної схеми

У послідовнісних схемах існує кінцеве число логічних станів. Тому їх називають кінцевими автоматами. Стан послідовнісних схем представляють двійковими сигналами, які називаються змінними станів.

Розглянемо модель Мілі (Mealy model), яка є моделлю послідовнісної логічної схеми на *D*-тригерах (рис. 1.1). У моделі можуть бути використані також інші типи тригерів.

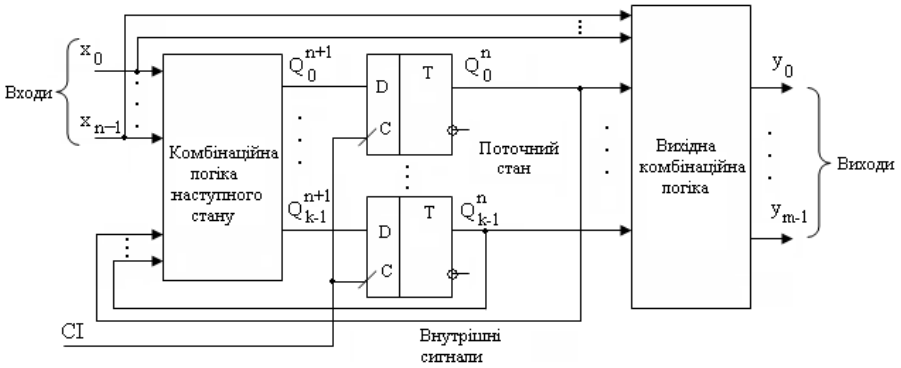


Рисунок 1.1 – Модель Мілі

У моделі Мілі сигнали Q^n визначають поточний стан схеми. Це змінні поточних станів. Сигнали Q^{n+1} – змінні наступних станів.

Сигнали Q^{n+1} – це комбінаційні функції вхідних сигналів X і поточних станів Q^n.

Особливості моделі: поточні стани Q^n змінюються у момент надходження активуючого переходу тактового імпульсу *C*. Тому зміна вхідного сигналу *X* не впливатиме на Q^n в моменти відсутності імпульсу *C*.

Вихідні сигнали *Y* описуються комбінаційними функціями вхідних сигналів і поточних станів. Тому зміни *Y* можуть відбуватися при зміні *X* незалежно від *C*.

Цієї проблеми легко уникнути, якщо сигнали *Y* залежатимуть тільки від поточних станів Q^n.

Подібний підхід реалізується в моделі Мура (Moore model), показаної на рис. 1.2.

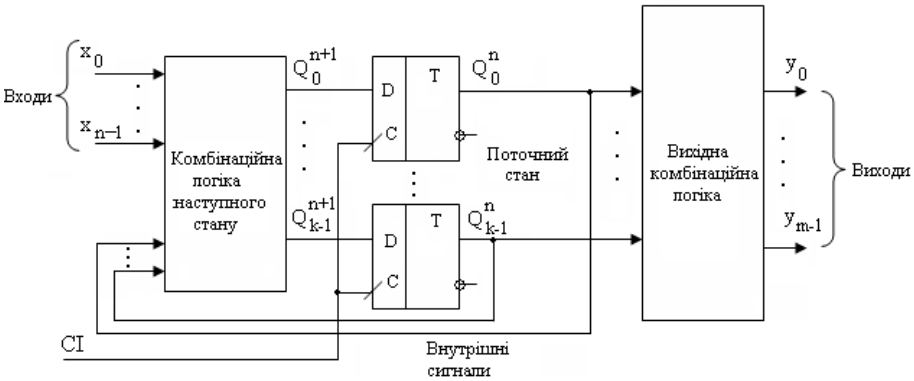


Рисунок 1.2 - Модель Мура

При проєктуванні послідовнісних схем передусім необхідно визначити кількість тригерів в схемі. У загальному випадку, при кількості станів від $2^{k-1}+1$ до 2^k необхідно мати *k* змінних стану і, отже, *k* тригерів.

Якщо використовуються тригери з перемиканням по позитивному фронту, то всі зміни станів відбуватимуться під час переходів рівня тактового сигналу з 0 в 1. Якщо ж застосовуються тригери з перемиканням по негативному фронту, то зміни відбуватимуться по задньому фронту тактового сигналу, тобто при переході з 1 в 0.

У будь-якому випадку, після приходу активуючого переходу тактового імпульсу необхідно забезпечити достатню затримку до моменту надходження переходу наступного тактового імпульсу, щоб сигнали встигли пройти схеми комбінаційної логіки і встановити значення змінних наступного стану.

Проєктування послідовнісної синхронної схеми можна виконувати як з використанням моделі Мілі, так і з використанням моделі Мура, але в моделі Мілі зміни вхідних сигналів можуть впливати на стан виходів безпосередньо у момент зміни, а не синхронно з приходом тактового імпульсу. Тому в цьому аспекті поведінка моделей може відрізнятися.

Лічильники – це схеми, поведінка яких описується моделлю Мура, оскільки виходи схеми – це безпосередньо виходи тригерів.

При проєктуванні послідовнісних схем необхідно визначити функції змінних наступних станів, визначувані вектором Q^{n+1}, і функції вихідних станів, визначувані вектором Y.

1.1.2 Проєктування схеми лічильника на основі моделі Мура

Далі на прикладі задачі розглядається проєктування схеми з використанням моделі Мура.

Задача. Розробити лічильник на основі D-тригерів із двома довільними рахунковими послідовностями.

Нехай A – вхід керування. При $A = 0$ формується послідовність 00, 01, 11. При $A = 1$ – послідовність 00, 11, 01.

Розв'язок. На основі умови задачі будується граф станів моделі Мура для лічильника. Кожна із заданих послідовностей має три стани, але при вмиканні схеми може виникнути і четвертий стан.

На графі станів лічильника (рис. 1.3) у вузлах вказані номери станів і значення сигналів на виходах схеми; над дугами проставлені значення сигналів керування A, необхідні для переходу із одного стану в інший. В даному випадку номери станів вибрані довільно.

На основі графа станів будується таблиця станів (табл. 1.1).

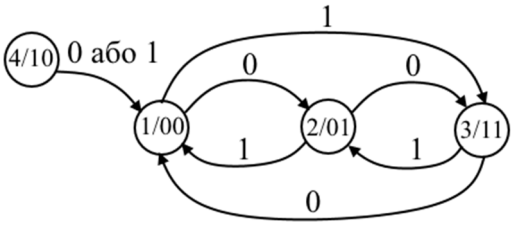


Рисунок 1.3 - Граф станів лічильника

Таблиця 1.1 - Таблиця станів лічильника

Поточний стан Q^n	Наступний стан Q^{n+1}	
	$A = 0$	$A = 1$
1	2	3
2	3	1
3	1	2
4	1	1

Призначення станам значень змінних. Для кожного стану необхідно вибрати значення змінних поточних станів. В цілях зручності стану 1 привласнюється комбінація значень змінних $Q_1^n Q_0^n$ = 00; стану 2 - $Q_1^n Q_0^n$ = 01; стану 3 - $Q_1^n Q_0^n$ = 11; стану 4 - $Q_1^n Q_0^n$ = 10. Вказане привласнення відповідає вихідним станам, що вимагаються в умові задачі. Результати призначень зведені в табл. 1.2.

Таблиця 1.2 - Таблиця призначених станів

Поточний стан $Q_1^n Q_0^n$	Наступний стан $Q_1^{n+1} Q_0^{n+1}$	
	$A = 0$	$A = 1$
00	01	11
01	11	00
11	00	01
10	00	00

Далі слід визначити вхідні функції тригерів з використанням табл. 1.2. Карти Карно для обох вхідних функцій і одержані на їх основі вказані функції приводяться на рис. 1.4.

$Q_1^n Q_0^n$ / A	00	01	11	10
0	0	1	0	0
1	1	0	0	0

$Q_1^n Q_0^n$ / A	00	01	11	10
0	1	1	0	0
1	1	0	1	0

$$D_1 = Q_1^{n+1} = \overline{Q}_1^n Q_0^n \overline{A} + \overline{Q}_1^n \overline{Q}_0^n A \qquad D_0 = Q_0^{n+1} = \overline{Q}_1^n \overline{A} + \overline{Q}_1^n \overline{Q}_0^n A + Q_1^n Q_0^n A$$

Рисунок 1.4 – Карти Карно і мінімізовані вхідні функції тригерів

Мінімізація функції неповна. Член $\overline{Q}_1^n \overline{Q}_0^n A$ вибраний тому, що він зустрічається в Q_1^{n+1}.

1.2 Завдання

1.2.1 На основі одержаної в задачі з пункту 1.2.2 моделі лічильника синтезуйте його логічну структуру.

1.2.2 Розробіть логічну структуру лічильника із рахунковою послідовністю 11, 01, 10.

1.3 Зміст звіту

Звіт повинен містити:

- мету роботи;
- моделі Мілі і Мура;
- схему лічильника із задачі 1.1;
- граф станів лічильника із рахунковою послідовністю 11, 01, 10; таблиці станів і призначених станів лічильника; вхідні функції тригерів; схему лічильника.

1.4 Контрольні запитання

1. Чи можна в моделях Мілі та Мура використовувати *JK*-тригери*?*
2. Скільки тригерів буде в схемі лічильника із 48 станами?
3. Чим відрізняються моделі Мілі та Мура?
4. Чи може бути призначення станам значень змінних довільним?
5. Коли доцільна неповна мінімізація вхідних функцій тригерів?
6. Чим відрізняється комбінаційна логіка від послідовнісної?

2 ПРАКТИЧНЕ ЗАНЯТТЯ № 2 „ПРОЄКТУВАННЯ СИНХРОННИХ СХЕМ НА ОСНОВІ МОДЕЛЕЙ МУРА І МІЛІ“

Мета роботи - ознайомлення із алгоритмом синтезу послідовнісних схем та з підходами до кодування станів; розгляд конкретних реалізацій алгоритму з використанням моделей Мура і Мілі.

2.1 Теоретичні відомості

2.1.1 Алгоритм проєктування електронної схеми

Алгоритм синтезу послідовнісної схеми містить наступні кроки:

- побудова діаграми станів;
- побудова таблиці станів;
- присвоєння станам схеми комбінацій значень змінних стану;
- побудова таблиці призначених станів;
- отримання вхідних функцій тригерів (за допомогою карт Карно);
- отримання за картами Карно вихідних функцій;
- побудова логічної схеми.

Реалізація приведеного алгоритму розглядається на прикладі задачі.

Задача. Синтезувати схему, що детектує певну кодову послідовність, яка надходить на вхід даних. Логічний стан цього входу може змінюватися після кожного тактового імпульсу. Схема має один вихід, стан якого буде рівним 1 тільки тоді, коли на вході з’являється задана послідовність (рис. 2.1).

Нехай, потрібно визначити наявність на вході кодової послідовності 0110011. Причому стан на виході *Y* повинен встановлюватися в 1 при появі на вході останнього біта послідовності.

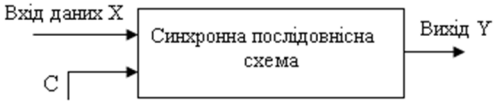


Рисунок 2.1 – Узагальнений вид детектора двійкової послідовності

Розв'язок. Послідовність можна реалізувати за 8 кроків, що передбачає наявність восьми станів. При побудові графа станів (рис. 2.2) використана модель Мура, в якій кожному стану відповідає конкретне значення вихідного сигналу *Y*.

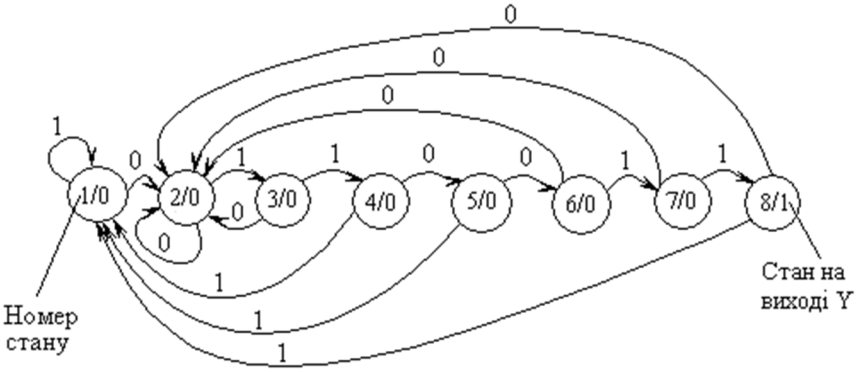


Рисунок 2.2 – Граф станів детектора двійкової послідовності

Початковим станом є стан 1, в якому $Y = 0$. Схема змінює свій стан, якщо на вході з'являється 0 (точніше, якщо сигнал «0» на вході існує під час приходу тактового імпульсу).

Це перший 0 послідовності, який потрібно знайти. Після його появи схема переходить у стан 2. При цьому на виході зберігається значення 0. Поява наступних станів вхідної послідовності приводить до переходу схеми в стани 3, 4, 5, 6, 7 і 8. У останньому восьмому стані на виході схеми встановлюється 1.

Після цього наступна 1 повертає схему в початковий стан із номером 1. Якщо ж приймається 0, то він може бути першим нулем нової послідовності, тому виконується перехід до стану 2.

Вся решта переходів в графі станів, що мають місце до прийому останнього біта послідовності, є переходами у стан 1, якщо приймається 1, і у стан 2, якщо приймається 0 (який розглядається як перший нуль послідовності).

Таблиця станів (переходів) детектора приведена в табл. 2.1.

Призначення станам значень змінних. Для опису восьми різних станів потрібні 3 змінні стану, нехай $Q3$, $Q2$, $Q1$. Необхідно

кожному із станів призначити певні значення змінних. Призначення може бути довільним.

Таблиця 2.1 - Таблиця станів детектора послідовності

Поточний стан	Наступний стан		Вихідний стан
	X=0	X=1	*Y*
1	2	1	0
2	2	3	0
3	2	4	0
4	5	1	0
5	6	1	0
6	2	7	0
7	2	8	0
8	2	1	1

Існує безліч варіантів таких призначень. Оптимального способу вибору варіанту не існує. Простішу функцію можна знайти, якщо на карті Карно є клітини із суміжними 1.

При призначенні можна використовувати нижченаведені правила.

Правило №1. При переході в наступний однаковий стан призначаються коди, відмінні значеннями тільки однієї змінної.

Правило №2. Наступним станам, в які може переходити поточний стан, слід призначати коди із однією змінною, що розрізняється.

Правило №3. Станам із однаковим значенням на виході слід призначати коди, відмінні значеннями тільки однієї змінної.

Правило №3 застосовують тоді, коли не можна використовувати правила №1 і №2.

Ілюстрації до приведених правил показані на рис. 2.3.

Рисунок 2.3 - Ілюстрації до правил №1, №2, №3

Вказані правила є рекомендаціями і не можуть гарантувати отримання схеми із мінімальною кількістю логічних елементів.

Для даної задачі, застосувавши правило №1 для станів 1, 4, 5 і 8 (всі вони переходять у стан 1) і станів 3, 6, 7 (вони переходять в один і той же стан 2), можна отримати наступні призначення:

стан 1 000;	стан 2 111;
стан 4 001;	стан 3 101;
стан 5 010;	стан 6 110;
стан 8 100;	стан 7 011.

Присвоєння стану 1 коду 000 полегшує ініціалізацію тригерів в цьому стані (за допомогою входів скиду).

У табл. 2.4 приведені коди призначених станів.

Таблиця 2.4 - Таблиця призначених станів детектора

Поточний стан $Q_3^n Q_2^n Q_1^n$	Наступний стан $Q_3^{n+1} Q_2^{n+1} Q_1^{n+1}$		Значення на виході Y
	X = 0	X = 1	
000	111	000	0
111	111	101	0
101	111	001	0
001	010	000	0
010	110	000	0
110	111	011	0
011	111	100	0
100	111	000	1

Вид вихідної функції детектора нескладно визначити із останнього рядка табл. 2.4 за кодом поточного стану:

$$Y = Q_3^n \overline{Q}_2^n \overline{Q}_1^n .$$

Функції збудження тригерів визначаються з використанням карт Карно.

XQ_3^n \ $Q_2^n Q_1^n$	00	01	11	10
00	1	0	1	1
01	1	1	1	1
11	0	0	1	0
10	0	0	1	0

$$D_3 = Q_3^{n+1} = \overline{X}\overline{Q}_1^n + \overline{X}Q_3^n + Q_2^n Q_1^n$$

XQ_3^n \ $Q_2^nQ_1^n$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	1
10	0	0	0	0

$$D_2 = Q_2^{n+1} = \overline{X} + Q_3^n Q_2^n \overline{Q_1^n}$$

XQ_3^n \ $Q_2^nQ_1^n$	00	01	11	10
00	1	0	1	0
01	1	1	1	1
11	0	1	1	1
10	0	0	0	0

$$D_1 = Q_1^{n+1} = Q_3^n Q_1^n + Q_3^n Q_2^n \overline{Q_1^n} + \overline{X}\,\overline{Q_2^n Q_1^n} + \overline{X} Q_2^n Q_1^n$$

Користуючись отриманими вище функціями, можна зобразити схему детектора послідовності.

2.1.2 Проєктування схем на основі моделі Мілі

Модель Мілі відрізняється від моделі Мура тим, що кожному стану не обов'язково відповідає один набір вихідних значень, тому вихідний сигнал залежить як від поточного стану, так і від сигналів на входах, що перемикають схему в даний стан.

На графі станів Мілі над дугами, спрямованими до стану, вказуються як вихідні, так і вхідні сигнали, які призводять до переходу в даний стан.

Часто для вирішення однієї і тієї ж задачі граф станів моделі Мілі має менше станів, ніж аналогічний граф моделі Мура, тому змінних стану і тригерів менше. При цьому вихідні функції можуть мати складнішу форму, тому що в них використовуються як змінні стану, так і вхідні змінні.

Суттєво те, що у випадку зміни вхідних сигналів в проміжках між надходженням тактових імпульсів можуть змінюватися значення виходів, що може впливати на схемні рішення системи в цілому.

Нижче приводиться алгоритм рішення задачі пункту 2.2.1 (синтез детектора двійкової послідовності 0110011) з використанням моделі Мілі. Граф станів моделі Мілі показаний на рис. 2.4.

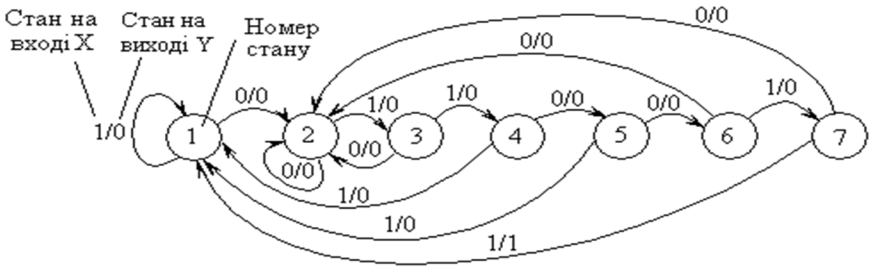


Рисунок 2.4 - Граф станів моделі Мілі для детектора двійкової послідовності

Таблиці станів моделі Мілі і призначених станів приведені в табл. 2.5 і табл. 2.6.

Таблиця 2.5 -Таблиця станів детектора

Поточний стан	Наступний стан		Наступний вихідний стан *Y*	
	X = 0	X = 1	X = 0	X = 1
1	2	1	0	0
2	2	3	0	0
3	2	4	0	0
4	5	1	0	0

Таблиця 2.6 - Таблиця призначених станів детектора

Поточний стан $Q_3^n Q_2^n Q_1^n$	Наступний стан $Q_3^{n+1} Q_2^{n+1} Q_1^{n+1}$		Наступний вихідний стан Y	
	X = 0	X = 1	X = 0	X = 1
000	111	000	0	0
111	111	101	0	0
101	111	001	0	0
001	010	000	0	0
010	110	000	0	0
110	111	011	0	0
011	111	000	0	1

У табл. 2.6 стан 100 не використовується, він ніколи не з'явиться, якщо при ввімкненні всі тригери обнулити спільним сигналом скиду 000. У табл. 2.6 використовуються ті ж призначення, що і в моделі Мура.

Отримана на основі табл. 2.6 вихідна функція має вигляд:

$$Y = X\overline{Q_3^n}Q_2^nQ_1^n .$$

Функції збудження тригерів визначаються за допомогою карт Карно.

XQ_3^n \ $Q_2^nQ_1^n$	00	01	11	10
00	1	0	1	0
01	*	1	1	1
11	*	1	1	1
10	0	0	0	0

XQ_3^n \ $Q_2^nQ_1^n$	00	01	11	10
00	1	1	1	1
01	*	1	1	1
11	*	0	0	1
10	0	0	0	0

XQ_3^n \ $Q_2^nQ_1^n$	00	01	11	10
00	1	0	1	1
01	*	1	1	1
11	0	0	1	0
10	0	0	0	0

$$D_1 = Q_1^{n+1} = Q_3^n + \overline{X}\overline{Q_2^n}\overline{Q_1^n} + \overline{X}Q_2^nQ_1^n ;$$

$$D_2 = Q_2^{n+1} = \overline{X} + Q_3^n\overline{Q_1^n} ;$$

$$D_3 = Q_3^{n+1} = \overline{X}\overline{Q_1^n} + \overline{X}Q_3^n + Q_3^nQ_2^nQ_1^n + \overline{X}Q_2^nQ_1^n .$$

2.2 Завдання

2.2.1 З використанням розробленої у задачі пункту 2.1.1 на основі автомата Мура моделі детектора двійкової послідовності синтезуйте його логічну структуру.

2.2.2 З використанням розробленої у пункті 2.1.2 на основі автомата Мілі моделі детектора двійкової послідовності синтезуйте його логічну структуру.

2.2.3 На основі автомата Мілі реалізуйте модель детектора двійкової послідовності 10110 і синтезуйте його логічну структуру.

2.3 Зміст звіту

Звіт повинен містити:

- мету роботи;
- алгоритм синтезу послідовнісної схеми;
- логічну структуру детектора двійкової послідовності 0110011 на основі автомата Мура (модель синтезована у задачі пункту 2.2.1);
- логічну структуру детектора двійкової послідовності 0110011 на основі автомата Мілі (модель синтезована у пункті 2.2.2);
- граф станів детектора двійкової послідовності 10110; таблиці станів і призначених станів детектора; вхідні функції тригерів; схему детектора.

2.4 Контрольні запитання

1. Задані чотири стани: 1, 2, 3, 4, для яких відомо, що стан 1 переходить у стан 2, стан 2 переходить у стан 3, стан 3 переходить у стан 4, стан 4 переходить у стан 1. Першому стану привласнюється код 00. Призначте значення змінних стану двома різними способами.
2. У чому різниця між графами станів моделі Мура і моделі Мілі?
3. Опишіть алгоритм синтезу послідовнісної схеми.
4. На основі чого отримуються функції збудження тригерів?
5. За якими правилами відбувається призначення станам значень змінних?

3 ПРАКТИЧНЕ ЗАНЯТТЯ № 3
„ОПИСИ ОБ'ЄКТІВ НА МОВІ VHDL“

Мета роботи - ознайомлення з видами опису систем на структурному і поведінковому рівнях; ознайомлення з представленням опису системи у вигляді дерев ієрархії різних типів; розгляд двох типів описів об'єкта на мові VHDL: опис об'єкта "у цілому" (*entity*) і опис архітектури об'єкта (*architecture*); ознайомлення з типами сигналів.

3.1 Види опису цифрової системи

VHDL - це могутня мова, яка дозволяє описувати поведінку, тобто алгоритми функціонування цифрових систем, а також проводити функціонально-структурний опис систем, має засоби для опису паралельних асинхронних процесів, регулярних структур і в той же час має всі ознаки мови програмування високого рівня - дозволяє створювати свої типи даних, має широкий набір арифметичних і логічних операцій.

Цифрова система на мові VHDL може бути описана на структурному і поведінковому рівнях.

Структурний опис – це опис системи у вигляді сукупності компонент (підсхем, елементів) і зв'язків між компонентами.

Компоненти системи у структурному описі можуть складатися з декількох частин більш низького рівня ієрархії.

Поведінковий опис задає алгоритм, що реалізовує система.

Приклад. На вхідні полюси цифрової системи S (рис. 3.1) подаються два дворозрядні числа $a = (a2, a1)$, $b = (b2, b1)$, де $a2$, $b2$ – старші розряди чисел a, b; x – сигнал керування.

Якщо $x = 0$, то система S складає числа a, b і видає чотирирозрядний результат $d = (d4, d3, d2, d1) = (0, c2, s2, s1)$, де $c2$ – сигнал переносу.

Якщо $x = 1$, то система S перемножує числа a, b і видає чотирирозрядний результат $d = (d4, d3, d2, d1)$, де $d = a \times b$.

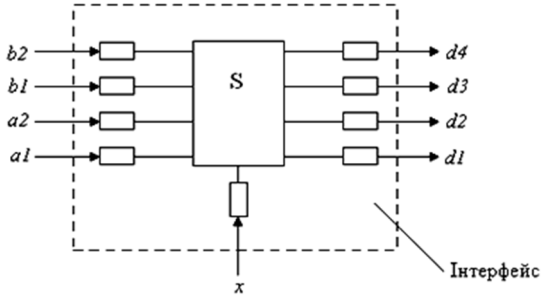


Рисунок 3.1 – Система *S* та її інтерфейс

Виконання структурного опису системи *S*. На структурному рівні в систему входять: дворозрядний помножувач, суматор, пристрій керування, а також схема диз'юнктивного об'єднання вихідних сигналів (рис. 3.2).

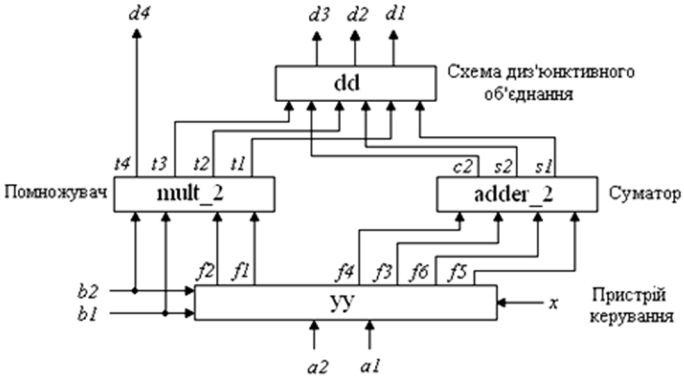


Рисунок 3.2 – Структура цифрової системи *S*

Алгоритм роботи пристрою керування:

якщо $x = 0$, то

$$(f4, f3) = (a2, a1),$$
$$(f6, f5) = (b2, b1),$$
$$(f2, f1) = (0, 0),$$

тобто числа *a, b* подаються на входи суматора;

якщо $x = 1$, то

$$(f4, f3) = (0, 0),$$
$$(f6, f5) = (0, 0),$$

$$(f2, f1) = (a2, a1),$$

тобто числа *a, b* подаються на входи помножувача.

Нехай опис системи матиме ім'я *VLSI*_1. Тоді ієрархія структурного опису системи *S* буде мати вид, представлений на рис. 3.3.

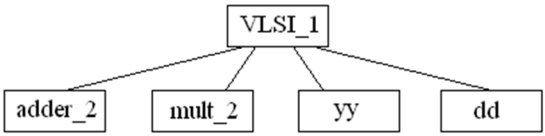


Рисунок 3.3 – Дерево ієрархії структурного опису системи *S*

Структура блоку mult_2 представлена на рис. 3.4.

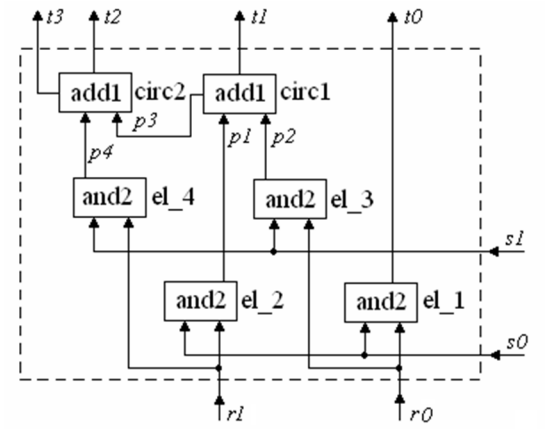


Рисунок 3.4 – Структура дворозрядного помножувача mult_2

Вхідні сигнали блоку mult_2:

$$r1, r0, s1, s0.$$

Сигнали $r1$, $r0$ інтерпретуються як дворозрядне ціле число $r = (r1, r0)$, сигнали $s1$, $s0$ – як дворозрядне число $s = (s1, s0)$.

Вихідні сигнали блоку mult_2:

$$t = (t3, t2, t1, t0) = (r1, r0)*(s1, s0).$$

В блоці mult_2 елемент add1 – однорозрядний півсуматор, функціонування якого описується таблицею істинності (табл. 3.1).

Таблиця 3.1 – Таблиця істинності півсуматора

*b*1	*b*2	*c*1	*s*1
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Математичний запис булевих функцій, представлених у табл. 3.1:

$$s1 = b1 \oplus b2 = (\overline{b1} \wedge b2) \vee (b1 \wedge \overline{b2}),$$

$$c1 = b1 \wedge b2.$$

Запис на мові VHDL, що описує функціонування елемента add1:

*s*1 <= ((*b*1 **and** (**not** *b*2)) **or** (**not** *b*1) **and** *b*2);

*c*1 <= *b*1 **and** *b*2;

де **and**, **or**, **not** – логічні оператори.

Елемент and2 – двовходовий кон'юнктор. Опис функції and2 на мові VHDL:

y <= *x*1 **and** *x*2;

де *x*1, *x*2 – вхідні сигнали;
y – вихідний сигнал.

У дерево проєкту схеми помножувача mult_2 входять:

- елемент and2;
- підсхема add1.

У свою чергу, підсхема add1 включає:

- or2 – двовходовий диз'юнктор;
- and2 – двовходовий кон'юнктор;
- inv – інвертор.

Елементи and2, or2, inv є ***листями*** проєкту. Вони не мають складових частин і називаються ***примітивами*** проєкту.

Примітив описується тільки на поведінковому рівні.

Об'єктами проєкту для дворозрядного помножувача mult_2 є add1, and2.

Позначення кореня дерева (mult_2) є ***ім'ям*** проєкту.

Кожен об'єкт проєкту має 2 різні типи описів:

- опис об'єкта «в цілому» (*entity*);
- опис архітектури об'єкта (*architecture*).

Архітектура - це структура системи на функціональному рівні її опису.

Архітектурне тіло (*architecture*) визначає тіло об'єкта. У архітектурному тілі описуються функції або структура об'єкта проєкту.

Опис об'єкта «в цілому» на мові VHDL носить назву ***«інтерфейс» об'єкта***. Він складається з імені об'єкта і опису портів (входів і виходів) об'єкта.

Наприклад, опис об'єкта проєкту and2 має вигляд:

```
entity and2 is                          – – декларація імені об'єкта проєкту;
port(x1, x2: in BIT;                                – – декларація вхідних портів;
      y2: out BIT);                     – – декларація вихідного порту;
end and2;
architecture functional of and2 is      – – декларація архітектури;
begin
          y <= x1 and x2;               – – опис функції об'єкта;
end functional;
```

У даному прикладі BIT – це тип сигналу.

Як видно з опису об'єкта and2, з ключового слова **architecture** починається алгоритмічна частина опису (архітектурне тіло). Архітектурне тіло має своє унікальне ім'я functional, яке зв'язується (**is**) з інтерфейсом схеми.

Опис об'єкта add1 має вигляд:

```
entity add1 is
port (b1, b2: in BIT;
      c1, s1: out BIT);
end add1;
architecture struct_1 of add1 is
begin
          s1 <= ((b1 and (not b2)) or ((not b1) and b2));
          c1 <= b1 and b2;
end struct_1;
```

Опис об’єкта проєкту mult_2 має вигляд:

```
 entity mult_2 is
port (s1, s0, r1, r0: in BIT;
      t3, t2, t1, t0: out BIT);
end mult_2;
architecture structure of mult_2 is
component
add1 port (b1, b2: in BIT;
           c1, s1: out BIT);
end component;
signal p1, p2, p3, p4: BIT;
begin
        t0 <= r0 and s0;                    – – елемент el_1;
        p1 <= r1 and s0;                    – – елемент el_2;
        p2 <= r0 and s1;                    – – елемент el_3;
        p4 <= r1 and s1;                    – – елемент el_4;

        circ1: add1 port map (p1, p2, p3, t1);
        circ2: add1 port map (p3, p4, t3, t2);
end structure;
```

Пояснення.

У описі архітектури проєкту mult_2 оголошуються (декларуються) дві підсхеми (компоненти): circ1, circ2.

Після ключового слова **begin** приводяться екземпляри описів, кожний з яких має унікальну мітку (circ1, circ2 – мітки), а також карту портів (**port map**).

Карта портів відображає зв’язок між входами, виходами описів компонента і екземплярами компонента.

У даному описі використовувалося поняття компонента (підсхеми) для add1, тоді як логічні елементи ”І” схеми описані на функціональному рівні (без використовування поняття компонента).

Можливість використовування змішаних описів є важливою корисною особливістю мови VHDL.

Опис дворозрядного суматора adder_2. Дворозрядний суматор (рис. 3.5) складається з двох підсхем: add1 і add2, де add1 – однорозрядний півсуматор, який вже розглядався, а add2 – однорозрядний суматор, функціонування якого описується таблицею істинності (табл. 3.2).

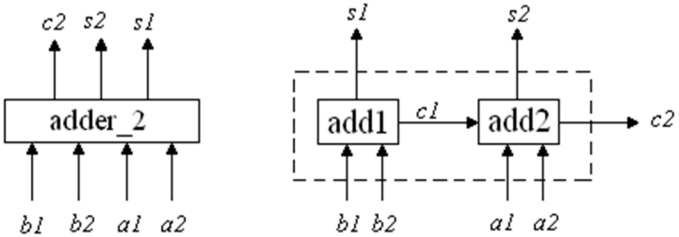
Умовне позначення дворозрядного суматора: **adder_2** (рис. 3.5).

Вихідні сигнали суматора формуються за формулою:

$$(a1, b1) + (a2, b2) = (c2, s2, s1).$$

Таблиця 3.2 – Таблиця істинності однорозрядного суматора

*c*1	*a*1	*a*2	*c*2	*s*2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



а б

а – умовне позначення;

б - схема у вигляді каскадного з'єднання однорозрядного півсуматора add1 та однорозрядного суматора add2

Рисунок 3.5 - Дворозрядний суматор $(a1, b1) + (a2, b2) = (c2, s2, s1)$

У дерево проєкту для підсхеми adder_2 входять підсхеми add1, add2. VHDL – опис підсхеми adder_2 має вид:

```
entity adder_2 is
port (a1, b1, a2, b2: in BIT;
      c2, s2, s1: out BIT);
end adder_2;

architecture structure of adder_2 is
component
add1
```

```
port (b1, b2: in BIT;
      c1, s1: out BIT);
end component;

component add2
port (c1, a1, a2: in BIT;
      c2, s2: out BIT);
end component;
signal c1: BIT;
begin
          circ1: add1
          port map(b1, b2, c1, s1);
          circ2: add2
          port map(c1, a1, a2, c2, s2);
end structure;
```

Можна помітити, що в різні підсхеми входить add1, при цьому підсхема add1, що входить у суматор adder_2, є листом проєкту, тому описана на поведінковому рівні.

Підсхема add1, що входить у помножувач mult_2, описана на структурному рівні.

VHDL – код для опису системи *S* має вид:

```
entity vlsi_1 is
port (a2, a1, b2, b1, x: in BIT;
      d4, d3, d2, d1: out BIT);
end vlsi_1;
architecture structure of vlsi_1 is
component adder_2
port (a1, b1, a2, b2: in BIT;
          , s2, s1: out BIT);
end component;
component mult_2
port (s1, s0, r1, r0: in BIT;
      t3, t2, t1, t0: out BIT);
end component;
component dd
port (x1, x2, x3, x4, x5, x6: in BIT;
      y1, y2, y3: out BIT);
end component;
component yy
port (a2, a1, b2, b1, x: in BIT;
      f6, f5, f4, f3, f2, f1: out BIT);
end component;
```

```
signal f1, f2, f3, f4, f5, f6, t4, t3, t2, t1, c2, s2, s1: BIT;
begin
        circ1: yy
        port map(a2, a1, b2, b1, x, f6, f5, f4, f3, f2, f1);
        circ2: mult_2
        port map(b2, b1, f2, f1, d4, t3, t2, t1);
        circ3: adder_2
        port map(f4, f3, f6, f5, c2, s2, s1);
        circ4: dd
        port map(s1, t1, s2, t2, c2, t3, d1, d2, d3);
end structure;
```

3.2 Використання САПР «MAX+plus II» для розробки цифрових пристроїв на ПЛІС

САПР «MAX+plus II» являє собою інтегроване середовище для розробки цифрових пристроїв на базі програмованих логічних інтегрованих схем (ПЛІС) фірми «Intel» і забезпечує виконання всіх етапів, необхідних для випуску готових виробів:

- створення проєктів пристроїв;
- синтез структур і трасування внутрішніх зв'язків ПЛІС;
- підготовку даних для програмування або конфігурування ПЛІС (компіляцію);
- верифікацію проєктів (моделювання і часовий аналіз);
- програмування або конфігурування ПЛІС.

Розробка функціонально-структурної схеми за її алгоритмічним описом називається ***високорівневим синтезом*** на відміну від логічного синтезу, коли за функціонально-структурним описом цифрової системи треба розробити логічну схему із заданих базисних логічних елементів.

Програма, яка здійснює синтез схеми на основі VHDL-опису, називається компілятором. Однак в системах моделювання VHDL-кодів під компіляцією також розуміється перетворення VHDL-коду в проміжну мову, яку безпосередньо використовують програми моделювання.

3.3 Процедура розробки проєкту в САПР MAX+plus II

Процедура розробки проєкту в САПР MAX+plus II полягає у виконанні проєктувальником нижченаведених поетапних дій.

Створення робочої папки для розміщення файлів проєкту. У директорії *MAXWORK* необхідно створити робочу папку, наприклад, під ім'ям *vlsi*_1.

Створення директорії проєкту. Директорія створюється за допомогою завдання послідовності команд *File*|*Project*|*Name* і введенням імені проєкту (наприклад, *vlsi*_1). При цьому вибирається створена робоча папка.

Створення текстового файлу. Якщо текстовий файл, створений в будь-якому текстовому редакторі (наприклад, в редакторі Word), вже існує, то необхідно його вміст записати в буфер пам'яті за допомогою введення команд *Правка*|*Виділити все*|*Ctrl+C*.

Далі слід активізувати текстовий редактор, задавши послідовність команд *Max+plusII*|*Text Editor*, а також перемістити інформацію з буфера пам'яті за допомогою натиснення поєднання клавіш *Ctrl+V*.

Якщо введений файл є програмою, написаною на мові VHDL, то його слід зберегти в директорії проєкту з розширенням *.vhd*. При цьому задається послідовність команд *File*|*Save As* і вводиться ім'я файлу (наприклад, *vlsi*_1.*vhd*).

Виконання компіляції файлу. Компіляція виконується шляхом запуску додатку *Compiler*. За наявності помилок в програмі, їх слід усунути і виконати повторну компіляцію.

Створення Include-файлу і символу бібліотечного елемента. Include-файл створюється командами *File*|*Create Default Include File* і записується в бібліотеку користувача. Прочитати цей файл і уточнити назви входів і виходів можна за допомогою виклику додатку *Hierarchy Display*. При цьому відображаються всі модулі проєкту і їх взаємозв'язки, а також всі типи файлів, сформовані в процесі обробки проєкту. Include-файл відображається з розширенням *.inc*. Його активізація дозволяє проглянути вміст вказаного файлу.

Створення графічного файлу. Для виклику графічного редактора потрібно в меню *Menager* вибрати *Max+PlusII*|*Grafic Editor*.

Графічному файлу із схемою необхідно командою *File|Save As* привласнити ім'я з розширенням *.gdf* (наприклад, *vlsi_*1.*gdf*).

Після того, як функціональні блоки введені, потрібно ввести символи вхідних і вихідних портів. Їх необхідно імпортувати з бібліотеки примітивів. Для цього необхідно двічі клацнути мишею по порожньому полю графічного редактора. Відкриється діалогове вікно, в якому в меню *Symbol Libraries* вказана бібліотека знаходиться за адресою *c:\program file\maxplusiiv10.2\max2lib\prim*. Після подвійного клацання за вказаною адресою в меню *Symbol File* з'явиться список логічних елементів. З вказаного списку необхідно вибрати примітиви портів, які зберігаються в бібліотеці під іменами *input* і *output*.

Далі необхідно привласнити імена всім портам.

Симуляція. Симуляція – це процес функціонального моделювання роботи схеми. Перед виконанням моделювання необхідно створити тестові вектори, тобто задати значення вхідних сигналів. Для цієї мети можна використати редактор діаграм, який вибирається послідовністю команд *Max+PlusII|Waveform Editor*.

Коли вікно редактора відкриється, створюється файл (наприклад, з назвою *vlsi_*1.*scf*) послідовністю *File|Save As* і вказівкою назви файлу (наприклад, *vlsi_*1.*scf*) в рядку *File Name* діалогового вікна, що відкрилося.

Далі визначаються вхідні і вихідні сигнальні лінії схеми для процесу симуляції. Для цього використовуються сигнальні лінії, занесені в *SNF-файл (Simulator Netlist File*), створений на етапі компіляції схеми. Необхідно відкрити список доступних в *SNF-файлі* сигнальних ліній за допомогою введення *Node|Enter Node from SNF*. Відкриється екран з двома вікнами: *Available Nodes & Groups* і *Selected Nodes & Groups*. Після натиснення *List* в першому вікні з'явиться список вхідних і вихідних ліній з *SNF-файлу.* Необхідно скопіювати список вхідних і вихідних ліній в друге вікно, тобто створити список вибираних сигнальних ліній. Після введення *OK* у вікні графічного редактора відобразяться вхідні і вихідні лінії.

Далі задається кінцевий час симуляції введенням *File|End Time* і інтервал часової сітки *Options|Grid Size*.

Для установки значень вхідних сигналів можна скористатися одним із способів: за допомогою вертикального репера встановити клацанням з протяганням тривалість сигналу, а потім ввести його

значення за допомогою кнопки символом "1" на лівій інструментальній панелі.

Для запуску пакету моделювання потрібно або ввести *File|Simulator*, або клацнути по кнопці симулятора на головній інструментальній панелі.

Якщо результати моделювання виявилися успішними, можна за допомогою виклику додатку *File|Timing Analyzer* відобразити таблицю *"Delay Matrix"*, в якій записані затримки формування вихідних сигналів щодо вхідних сигналів.

Призначення ресурсів. Для призначення ресурсів фізичних пристроїв і проглядання результатів розводки, зроблених компілятором, викликається порівневий планувальник *File|Floorplan Editor*. У вікні планувальника можна побачити тип мікросхеми, яка вибиралася в проєкті автоматично (при необхідності тип ПЛІС можна вибрати) і умовне графічне зображення вибраної ПЛІС з вказівкою під'єднуваних входів і виходів схеми.

3.4 Завдання

Розробіть проєкт під ім'ям *vlsi*_1. Порядок дій при розробці проєкту описаний у підрозділі 3.4. Текстовий файл, який описує систему *S* (додаток А), надається викладачем.

При створенні графічного файлу використовуйте рис. 3.2.

При функціональному моделюванні роботи схеми, представленої графічним файлом, необхідно задати значення вхідних сигналів на різних часових інтервалах відповідно табл. 3.3.

3.5 Зміст звіту

Звіт повинен містити:

- мету роботи;
- зображення системи *S* та її інтерфейсу (рис. 3.1);
- структуру цифрової системи *S* (рис. 3.2);
- часові діаграми роботи системи *S*;
- висновки.

Таблиця 3.3 – Значення коду (x, $b2$, $b1$, $a2$, $a1$)

№ вар.	Часові інтервали, нс			
	0…80	80…160	160…240	240…300
	x, $b2$, $b1$, $a2$, $a1$	x, $b2$, $b1$, $a2$, $a1$	x, $b2$, $b1$, $a2$, $a1$	x, $b2$, $b1$, $a2$, $a1$
1	0 0 0 1 0	0 1 1 1 0	1 1 1 0 1	1 1 0 0 1
2	0 1 1 1 1	0 1 0 0 0	1 0 1 0 0	1 0 1 0 1
3	0 0 0 0 1	0 1 0 1 0	1 1 0 0 1	1 0 0 1 0
4	0 1 0 0 1	0 0 1 1 1	1 0 1 1 1	1 1 0 1 0
5	0 0 0 1 0	0 1 0 1 0	1 1 0 0 1	1 0 1 1 1
6	0 1 0 1 0	0 0 1 1 1	1 0 0 1 0	1 0 0 0 1
7	0 0 1 0 0	0 1 0 1 0	1 1 1 1 0	1 0 1 1 1
8	0 1 0 0 1	0 1 0 1 0	1 0 0 1 0	1 0 1 0 1
9	0 0 1 1 0	0 1 0 0 1	1 1 0 1 1	1 0 0 0 1
10	0 1 0 0 1	0 0 1 1 1	1 0 1 0 0	1 1 0 1 0
11	0 0 1 1 0	0 1 0 0 1	1 1 0 0 1	1 0 0 0 1
12	0 1 0 1 0	0 0 0 1 0	1 0 0 1 0	1 1 1 0 0
13	0 0 1 0 1	0 1 0 0 1	1 1 0 0 0	1 0 0 0 1
14	0 1 0 1 0	0 1 1 1 0	1 0 0 1 0	1 1 0 1 0
15	0 0 0 0 1	0 1 1 0 0	1 1 0 0 1	1 0 0 1 0
16	0 1 0 1 0	0 0 0 0 1	1 0 0 1 0	1 0 0 0 1
17	0 0 0 1 0	0 1 0 1 0	1 1 0 1 0	1 0 1 1 0
18	0 1 1 1 1	0 1 0 0 1	1 0 1 0 1	1 1 1 1 1
19	0 0 1 0 1	0 1 1 0 1	1 1 0 1 0	1 0 1 0 0
20	0 1 1 0 1	0 1 0 0 0	1 0 0 0 1	1 0 1 0 1
21	0 0 0 1 0	0 1 0 0 1	1 1 0 1 0	1 0 1 1 1
22	0 1 1 1 0	0 0 0 1 0	1 0 1 1 1	1 1 0 1 0
23	0 0 0 1 0	0 1 0 0 0	1 1 1 0 0	1 0 0 0 1
24	0 1 0 0 1	0 1 0 0 1	1 0 0 1 0	1 0 0 0 1
25	0 0 1 0 1	0 1 1 1 0	1 1 0 0 0	1 0 0 0 1
26	0 1 0 1 0	0 0 1 1 1	1 0 1 0 0	1 1 0 1 0
27	0 0 0 0 1	0 1 0 1 0	1 1 1 0 1	1 0 0 0 1
28	0 1 1 0 0	0 1 0 0 0	1 1 1 0 1	1 1 0 0 1
29	0 0 0 1 0	0 1 1 1 1	1 1 1 0 1	1 0 1 0 1
30	0 1 1 1 0	0 1 0 1 0	1 0 1 1 0	1 0 0 1 1

3.6 Контрольні запитання

1. Що таке VHDL-опис, VHDL-код?
2. Чим структурний опис системи відрізняється від поведінкового?
3. Що таке проєкт, лист проєкту, примітив проєкту?
4. Що таке ієрархія проєкту?
5. Що таке високорівневий синтез?
6. Що таке логічний синтез?
7. Що таке архітектура об'єкта, архітектурне тіло (**architecture**)?
8. Що таке структура схеми, функція схеми, поведінка схеми?
9. Опишіть процедуру розробки проєкту в САПР MAX+PLUS II.
10. Чи можна на мові VHDL написати програму знаходження факторіалу натурального числа?
11. Чи правильне твердження: "VHDL має багато можливостей для моделювання аналогових схем"?
12. Чи правильне твердження: "Коментар в мові VHDL починається і закінчується двома дефісами"?
13. Скільки приведених нижче операторів відповідає виразу "заперечення кон'юнкції X, Y", якщо відомо, що оператор **not** має найвищий пріоритет в порівнянні з іншими логічними операторами (тобто виконується першим)?
    а) Z<=**not** X **and not** Y;
    б) Z<=**not** (X **and**Y);
    в) Z<=**not** X **and** Y;
14. Чи виконують приведені нижче оператори однакові функції?

    Z<=(A **nand** B) **nand** C;
    Z<=A **nand** (B **nand** C);
    Z<=**not** X **and** Y;

15. Чи представляє тривходову NAND – комірку приведений нижче вираз?

    A **nand** B **nand** C;

16. Чим відрізняється процес симуляції від процесу компіляції?

4 ПРАКТИЧНЕ ЗАНЯТТЯ № 4
„РЕАЛІЗАЦІЯ АВТОМАТНИХ VHDL-МОДЕЛЕЙ“

Мета роботи - ознайомлення з поняттям ”підмножина мови VHDL, що синтезується”; розгляд методики опису кінцевого автомата на мові VHDL; виконання проєкту пристрою, що реалізує заданий автомат з використанням VHDL-програми та аналіз його часових параметрів.

4.1 Теоретичні відомості

4.1.1 Підмножина мови VHDL, що синтезується

На основі VHDL-моделі синтезується схема, функції якої відповідають алгоритму VHDL-моделі.

Синтез здійснюється за допомогою спеціальних програм, званих синтезаторами. Для кожного типу кристалів PLD або FPGA в синтезаторі є цільова бібліотека.

Основні кроки синтезу: створення проєкту і установка опцій синтезу. ***Проєкт*** - це сукупність початкових VHDL-описів, необхідних пакетів, бібліотек, а також деякі внутрішні представлення, що вимагаються для роботи синтезатора.

Основними опціями синтезу є складність схеми «**Area**» (площа), швидкість «**Speed**», тип кодування даних (двійкове «**Binary**», унітарне «**One hot**», випадкове «**Random**», кодом Грея «**Gray**»), частота «**Clock Frequency**».

Тип Bit_vector визначає масив бітів.

Приклади:

```
bit_vector(0 to 3);          -- зростаючий  діапазон
bit_vector(7 downto 0);      -- убуваючий діапазон
```

Приклад:

```
Siqual DataBus: bit_vector (7 downto 0);
1 0 0 1 0 1 0 1
7 6 5 4 3 2 1 0 № разряда
DataBus = “10010101”;
   DataBus(7) = “1”;
   :
   DataBus(0) = “1”;
```

4.1.2 Моделювання VHDL-описів

Виконання VHDL-програми здійснюється за допомогою спеціальної програми - системи моделювання, яка включає:

- організацію проєкту (визначення директорії проєкту, розміщення в ній VHDL-кодів, необхідних пакетів, бібліотек тощо);
- компіляцію (**compile**) - перетворення VHDL-кодів у внутрішнє представлення, яке моделюється (компіляція не є синтезом логічної схеми);
- збирання (**link**) проєкту;
- моделювання (**run**), тобто виконання VHDL-кодів, представлених у внутрішній формі;
- візуалізацію результатів.

Після того, як проєктувальник переконається в коректності VHDL-моделі, виконується її схемна реалізація, тобто виконується синтез схеми. Зазвичай це інтегрована схема типу ПЛІС, або схема типу вентильної матриці.

При синтезі оператори мови VHDL замінюються компілятами.

Компілят - це підсхема, що реалізовує цілком певний оператор, наприклад, оператор складання.

При синтезі дані типу **bit** відповідають провідникам, типу **bit_vector** - шинам.

Проте перехід до відповідної логічної схеми здійснюється не для всієї мови VHDL, а тільки для деякої підмножини цієї мови, званої підмножиною, що синтезується.

Тільки для VHDL-моделі цифрової системи, описаної на підмножині мови VHDL, що синтезується, можна синтезувати схему.

Логічні оператори **and**, **or**, **xor**, **nand**, **nxor**, **not** входять в підмножину, що синтезується.

Логічні оператори **and**, **or**, **xor** мають однакове старшинство і виконуються зліва направо у виразах.

Логічний оператор **not** має найвищий ранг і виконується раніше інших операторів

Z <= A and not B or C; $Z = A\overline{B} + C$

В логічних операторах використовуються наступні типи даних: **boolean**; **bit**; **bit_vector**; **std_logic**; **std_logic_vector**; **std_ulogic**; **std_ulogic_vector**.

Тип **bit_vector** визначає масив бітів.

Приклади:

```
bit_vector(0 to 3);          --зростаючий діапазон
bit_vector(7 downto 0);  -- убуваючий діапазон
```

Приклад:

```
siqual DataBus: bit_vector (7 downto 0);
1 0 0 1 0 1 0 1
7 6 5 4 3 2 1 0 № розряду
dataBus = "10010101";
dataBas(7)= "1";
.
dataBas(0)= "1";
```

4.1.3 Типи даних std_logic, std_logic_vector

Для опису вхідних і вихідних даних (кодів) синтезованих схем використовуються тільки типи даних: **std_logic**, **std_logic_vector**.

Ці дані можуть приймати значення з багатозначної логіки:

'**U**' - не ініціалізоване;
'**X**' - невідоме значення (сильне джерело сигналу);
'**0**' - логічний 0 (сильне джерело сигналу);
'**1**' - логічна 1 (сильне джерело сигналу);
'**Z**' - високий імпеданс (ланцюг не підключений до джерела);
'**W**' - невідоме значення (слабке джерело сигналу);
'**L**' - логічний 0 (слабке джерело сигналу);
'**H**' - логічна1 (слабке джерело сигналу);
'**-**' - невизначене значення (байдужий стан).

Визначення типів **std_logic**, **std_logic_vector** міститься у пакеті std_logic_1164, там же міститься визначення підтипів, функцій на випадок багатозначної логіки. Щоб використовувати пакет std_logic_1164, необхідно задати:

```
Library IEEE;
Use IEEE.std_logic_1164.all;
```

Мова VHDL передбачає паралельну роботу процесів у часі. Тому сигналу одночасно можуть бути привласнені різні значення (у разі

наявності у сигналу декількох джерел). Для визначення значення сигналу у цій ситуації використовується ***функція перекриття***, яка визначає закон формування результуючого значення сигналу для всіх можливих комбінацій джерел сигналів. Принцип перекриття сигналів представлено на рис. 4.1.

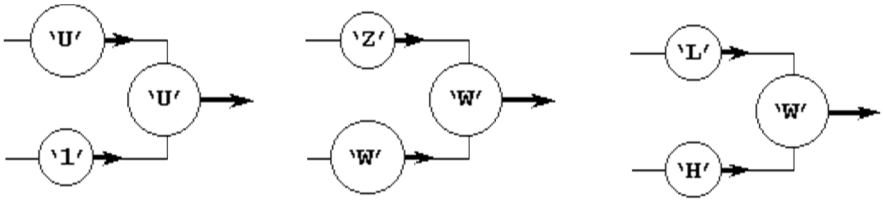


Рисунок 4.1 – Перекриття сигналів типу *Std_Logic*

Серед стандартних типів сигналів операція перекриття у VHDL визначена тільки для типів **std_logic** та **std_logic_vector**.

Основне призначення типу **std_logic** - це дати можливість розробнику робити багатократні привласнення одному і тому ж сигналу.

Тип **std_logic** є підтипом типу **std_ulogic**.

Тип **std_logic** визначається в пакеті **std_logic_1164** так:

Subtype std_logic is resolved std_ulogic;

Тип **std_ulogic** є недозволяємим перераховуємим типом з множиною значень {'U','X','0','1','Z','W','L','H',' -'}. Тобто сигнали цього типу не повинні мати багато джерел, їх визначення міститься в тому ж пакеті std_logic_1164.

4.1.4 VHDL - моделі кінцевих автоматів

ПЛІС, виконані за архітектурою FPGA, мають достатнє число тригерів, тому використовування автоматних моделей дозволяє одержати достатньо швидкодіючу і в той же час наочну реалізацію пристрою при прийнятних витратах ресурсів.

Нижче розглядається приклад проєктування схеми на базі автомата Мілі.

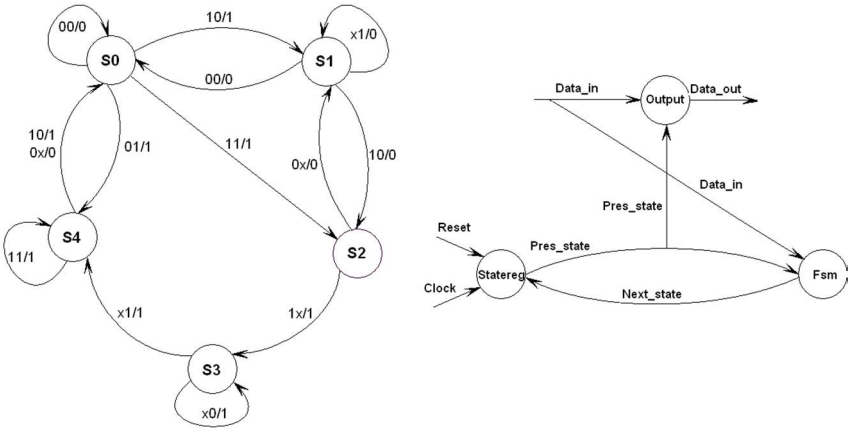
VHDL – код реалізує поведінку кінцевого автомата, заданого графом (рис. 4.2,а). Архітектурний опис цього кінцевого автомата містить два внутрішні сигнали: *Pres_state* і *Next_state*. Сигнал *Pres_state* призначений для зберігання поточного стану автомата. Фізично він реалізується у формі тригерів (регістра відповідної розрядності). Сигнал *Next_state* використовується для визначення наступного стану – стану, який стане поточним у наступному такті. Значення цього сигналу визначається на базі значень вхідних сигналів і поточного стану автомата. Фізично цей сигнал реалізується у формі ліній зв'язку.

Поведінка автомата представляється у вигляді сукупності трьох процесів (рис. 4.2, б).

В процесі *Fsm* визначається наступний стан автомата. Фізична реалізація такого процесу, як правило, є комбінаційною схемою. В загальному випадку в тілі процесу можуть з'являтися оператори умовного переходу, в гілках яких визначаються значення для різних наборів сигналів. Тоді в схемі можуть з'явитися клямки.

Процес *Statereg* має список чутливості, в який входять сигнали *Reset* (обнулення) і *Clock* (синхросигнал). В процесі *Statereg* виконується перехід із стану у стан за відповідним фронтом сигналу *Clock*. За сигналом *Reset* = 0 автомат встановлюється в початковий стан *ST*0.

Скидання автомата в початковий стан дозволяє забезпечити стабільну і безвідмовну роботу цифрового пристрою. Таким чином завжди забезпечується ініціалізація автомата в заздалегідь певному стані при першому тактовому імпульсі. Якщо скидання не передбачене, неможливо визначити, з якого стану почнеться функціонування, це може привести до збоїв в роботі всієї системи. Ймовірність виникнення такої ситуації збільшується при вмиканні живлення системи. Тому настійно рекомендується використовувати схеми скидання і початкової установки при проєктуванні пристроїв на ПЛІС.



а б

а –граф автомата;
б - опис функціонування кінцевого автомата у вигляді сукупності процесів

Рисунок 4.2 – Кінцевий автомат Мілі з п’ятьма станами

В процесі *Output* відповідно до поточного стану автомата визначається стан виходу *Data_out*.

4.2 Завдання

4.2.1 Розробка проєкту Mealy

Виконайте VHDL-програму автомата Мілі з п’ятьма станами (додаток Б). Послідовності сигналів на шині **data_in** задайте у шістнадцятковій системі числення відповідно табл. 4.1. Часовий інтервал дії кожної з послідовностей задайте довільно.

Приклад часової діаграми, отриманої при виконанні VHDL-програми, представлений на рис. 4.3. Тривалість періоду синхросигналу на діаграмі становить 50 нс. Кінцевий час моделювання Ткін= 1 мкс.

Зробіть порівняльний аналіз отриманої під час роботи часової діаграми і графа автомата (рис. 4.2, а). Зробіть висновок щодо правильності функціонування розробленого пристрою.

Таблиця 4.1 - Послідовності сигналів на шині **data_in**

№ вар.	Послідовності	№ вар.	Послідовності
1	0 2 1 3 2	16	0 3 2 1 3
2	2 3 0 2 1	17	3 0 2 0 1
3	3 2 1 0 1	18	2 3 2 0 3
4	1 3 2 1 0	19	1 3 2 1 0
5	3 0 2 1 3	20	2 0 1 2 3
6	0 1 3 2 0	21	2 1 2 0 3
7	1 0 2 3 1	22	0 3 2 3 1
8	3 1 2 3 0	23	3 0 2 1 0
9	2 1 3 0 3	24	0 3 2 3 1
10	1 0 2 1 3	25	1 0 3 2 3
11	2 1 0 3 2	26	3 1 2 1 0
12	1 3 1 2 0	27	3 2 0 1 3
13	2 3 1 0 2	28	3 2 1 1 3
14	2 0 2 1 3	29	0 2 3 1 2
15	1 2 0 3 2	30	0 1 2 1 3

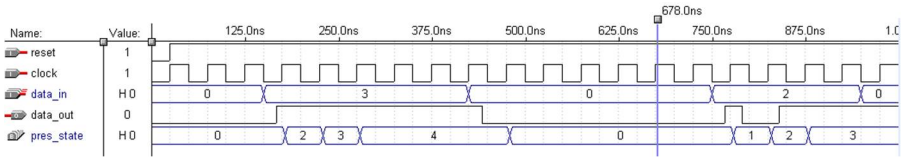


Рисунок 4.3 – Приклад часової діаграми роботи автомата Мілі з п’ятьма станами

4.2.2 Аналіз часових параметрів проєкту

4.2.2.1 Робота з аналізатором часових параметрів Timing Analyzer

З меню **MAX+plus II** виберіть команду **Timing Analyzer** (аналізатор часових параметрів). Запуск **Timing Analyzer** приводить до відкриття його вікна і появи на верхній панелі трьох додаткових піктограм: **Delay Matrix** (матриця затримок), **Setup/Hold Matrix** (матриця часів передустановки і утримання сигналів) і **Registered Performance** (швидкодія регістрової логіки). При цьому піктограма **Delay Matrix** є активізованою. Після натиснення кнопки Start

аналізатор часових параметрів обчислює затримки розповсюдження сигналів між вхідними і вихідними контактами поточного проєкту.

Якщо шляхи розповсюдження сигналу мають різну довжину, то в комірці **Delay Matrix** з'являються два значення затримок, відповідні найдовшому і найкоротшому шляхам. Це означає, що в схемі є змагання сигналів. Коли в початковому проєктному файлі джерело і приймач сигналу розділяються інформаційним входом *D*-тригера, затримка обчислюється через *Clock* (тактуючий) або *Preset* (встановлюючий) входи, а не через *D* (інформаційний) вхід.

При активізації **Setup/Hold Matrix** і кнопки Start визначте мінімально допустимі значення часів передустановки і утримання сигналів для інформаційних входів тригерів.

При активізації **Registered Performance** і кнопки *Start* визначте затримки в логіці між регістрами, мінімальний період і максимальну частоту тактового сигналу.

4.2.2.2 Робота в редакторі фізичного розміщення Floorplan Editor

Викличте редактор з основного меню **MAX+plus II**. В меню **Layout** представлені два варіанти зображення мікросхеми: **Device View** (показує всі контакти на корпусі мікросхеми та їх функції) і **LAB View** (вид логічних блоків, що показує логічні блоки *LAB* і логічні комірки *LC* усередині блоків, а також комірки вводу-виводу - *I/O cell*).

Виберіть команду **LAB View**. Прогляньте на екрані для кожного задіяного елементу вхідні і вихідні зв'язки. Для цього встановіть курсор на відповідну комірку і активізуйте її клацанням клавіші миші. Використовуючи піктограми  , розташовані зліва, виведіть на екран вхідні і вихідні зв'язки кожного елементу, розташованого в одній зайнятій комірці, а також зв'язки між елементами.

4.2.2.3 Отримання Delay Matrix

Для розрахунку **Delay Matrix** того вигляду, який Вам потрібен (бажано з більшою кількістю стовпців і рядків), на екрані редактора **Floorplan Editor** послідовно активізуйте вибрані комірки, викличте клацанням правої клавіші миші спливаюче меню, виберіть команду

Timing Analysis та одну з вкладок: **Sourse** (джерело сигналу, розташовується у рядку матриці), **Destination** (приймач сигналу, розташовується у стовпці матриці), **Cutoff** (вирізати з матриці). Потім запустіть **Timing Analyzer**.

4.3 Зміст звіту

Звіт повинен містити:

- мету роботи;
- граф кінцевого автомата Мілі з п'ятьма станами;
- опис функціонування кінцевого автомата у вигляді сукупності процесів; часову діаграму роботи автомата Мілі;
- матрицю затримок;
- мінімально допустимі значення часів передустановки і утримання сигналів для інформаційних входів тригерів;
- затримки в логіці між регістрами, мінімальний період і максимальну частоту тактового сигналу;
- висновки.

4.4 Контрольні запитання

1. Чим відрізняється підмножина мови VHDL, що синтезується, від множини мови VHDL?
2. Які параметри оптимального синтезу схеми можна змінити за допомогою послідовності команд **Assign/Global Progect Logic Synthesis**?
3. На якому етапі розробки проєкту створюються компіляти?
4. Чи можна стверджувати, що провіднику у схемі відповідає змінна типу **bit_vector**? Поясніть відповідь.
5. Навіщо необхідна передустановка сигналів для інформаційних входів тригерів?
6. Як сформувати матрицю затримок бажаної форми?
7. Які процеси використовуються при описі поведінки автомата?
8. В чому полягає робота з аналізатором часових параметрів Timing Analyzer?
9. В чому полягає робота з редактором фізичного розміщення Floorplan Editor?

5 ПРАКТИЧНЕ ЗАНЯТТЯ № 5
„ПРОЄКТУВАННЯ ЕЛЕКТРОННИХ СХЕМ НА ПЛІС“

Мета роботи - ознайомлення з особливостями ПЛІС; розгляд принципів опису кінцевих автоматів; отримання навичок синтезу функціональних схем пристроїв при наявності автоматної моделі; розробка проєкту пристрою з використанням САПР.

5.1 Теоретичні відомості

5.1.1 Особливості програмованих логічних інтегрованих схем

Програмовані логічні інтегровані схеми (ПЛІС) з’явилися як альтернатива ***програмованим логічним матрицям*** (ПЛМ). Від останніх ПЛІС відрізняються як за архітектурою, так і за технологією виготовлення.

ПЛМ є матрицею багатовходових (декілька десятків входів) логічних елементів з тригерами, в яких перемичками програмуються конституенти одиниць диз’юнктивних нормальних форм функцій цих елементів. Спочатку перемички виконувалися у вигляді перепалюваних тонких провідничків. Тепер перемички виконуються у вигляді МОН-транзисторів з плаваючим затвором, як в електрично перепрограмовуваному запам’ятовувальному пристрої (ППЗП), тобто ПЛМ виготовляються за технологією флеш-пам’яті. Великі ПЛМ (CPLD) відрізняються тільки тим, що декілька ПЛМ зібрані на одному кристалі і об’єднані програмованим полем зв’язків.

ПЛІС є матрицею маловходових (від двох до п’яти входів) логічних елементів, тригерів, відрізків ліній зв’язку, що сполучаються перемичками з польових транзисторів. Судячи з англійської назви - Field Programmable Gate Array (FPGA) - ПЛІС програмуються зміною рівня на затворах польових транзисторів, що використовуються для програмування. Ці транзистори підключені до входів тригерів одного довгого зсувового регістра, який заповнюється при програмуванні ПЛІС. Деякі з ділянок цього регістра можуть також виконувати роль комірок постійного запам’ятовувального пристрою (ПЗП).

Прошивка звичайно зберігається в постійному запам’ятовувальному пристрої, що стоїть поряд з ПЛІС і після

приєднання живлення або за сигналом скидання вона автоматично переписується в програмуючий зсувовий регістр ПЛІС.

Цей процес називається конфігурацією ПЛІС. Оскільки основу ПЛІС складають тригери, що зберігають прошивку, то ПЛІС виготовляються за технологією мікросхем статичного оперативного запам'ятовувального пристрою (ОЗП).

В порівнянні з CPLD, ПЛІС виграють у:

- необмеженій кількості перепрограмувань;
- логічній ємності;
- малому енергоспоживанні.

Як правило, ПЛІС мають на два - три порядки більшу ємність у кількості еквівалентних логічних вентилів, ніж CPLD, і, як статичне ОЗП, майже не споживають енергії за відсутності перемикань. Крім того, у ПЛІС на порядок вищий рівень надійності (нижча інтенсивність відмов), ніж у CPLD. До недоліків ПЛІС відносять необхідність зовнішньої ПЗП-прошивки, а також необхідність наявності генератора синхросерії. Але 8-виводове ПЗП займає на платі значно менше місця, ніж сама ПЛІС з багатьма сотнями виводів. Те ж стосується і генератора синхросерії.

Багато сумнівів у користувачів виникає із захистом проєкту від копіювання. Дійсно, прошивка ПЛІС зберігається в зовнішньому ПЗП, вміст якого просто копіюється. Але змінити або розшифрувати прошивку, наприклад, для приховання авторства або відновлення схеми, практично неможливо, оскільки семантика бітів у файлі прошивки - секрет фірми, а необережна зміна її може вивести ПЛІС з ладу. Якщо потрібен захист, то завантаження програми виконують за допомогою зовнішньої CPLD, яка автоматично забезпечує захист проєкту. У ПЛІС нових поколінь передбачається шифрування прошивки, наприклад, за допомогою вбудованого шифрувача DES із забезпеченням збереження ключа за допомогою батареї.

5.1.2 Опис алгоритму роботи кінцевого автомата

A finite state machine (FSM) - кінцевий автомат є моделлю поведінки, що складається з кінцевої кількості станів, переходів між цими станами і операцій. Кінцевий автомат - це абстрактна модель пристрою з примітивною оперативною пам'яттю.

Кінцеві автомати використовуються в багатьох областях, наприклад, в електротехніці, лінгвістиці, інформатиці, біології, математиці, логіці. У інформатиці кінцеві автомати широко використовуються в моделюванні поведінки додатків, проєктуванні апаратних цифрових систем, розробці програмного забезпечення, компіляторів, мережевих протоколів.

Алгоритм роботи кінцевого автомата зручно описувати графом (рис. 5.1).

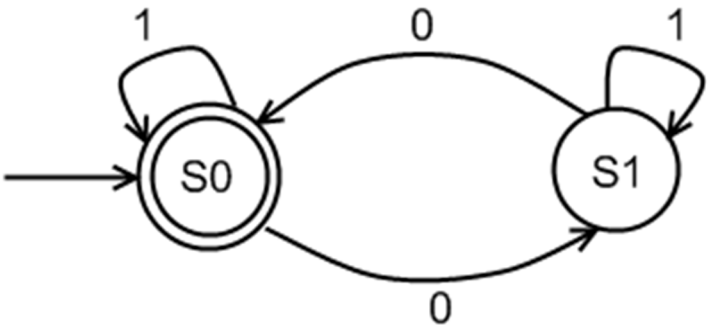


Рисунок 5.1 - Граф КА визначення парності кількості 0 в двійковому числі

Кінцевий автомат у кожен конкретний момент може знаходитися тільки в одному стані. Кожен тактовий імпульс може привести до переходу автомата в інший стан. Правила переходу визначаються комбінаційною схемою, званою логікою переходу. Наступний стан визначається як функція поточного стану. Стан виходу автомата у разі автомата Милі визначається за допомогою логіки формування виходу.

Стан вводу (іноді розуміється як приймаючий стан) - стан, в якому машина успішно виконала свою процедуру. Він зазвичай відображається подвійним колом.

Початковий стан *S*0 автомата, граф якого представлений на рис. 5.1, визначається як приймаючий стан. Цей автомат дасть правильний кінцевий стан, якщо двійкове число містить парне число нулів, або в числі немає нулів. Приклади рядків, прийнятих цим КА - епсілон (порожній рядок), 1, 11, 11..., 00, 010, 1010, 10110, і так далі.

5.2 Завдання

5.2.1 Постановка задачі

Синхронний автомат використовує два імпульси *Out*1 та *Out*2, що не перекриваються. Автомат приймає чотири стани: *Indle*, *Delay*, *Next*, *Done*. Граф автомата показаний на рис. 5.2.

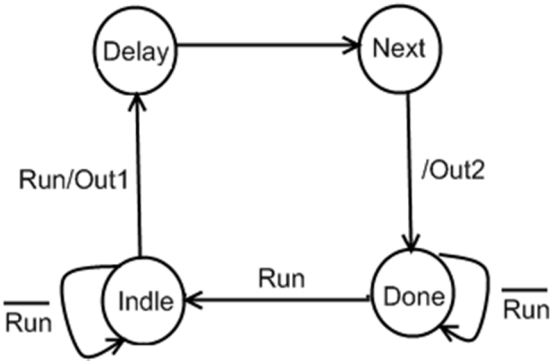


Рисунок 5.2 – Граф автомата з чотирма станами

Отже, для кодування станів потрібно два тригери. При використовуванні методу двійкового кодування можна записати систему рівнянь, що описує роботу автомата (символ ”¬” відповідає операції НЕ*)*:

$$
\begin{aligned}
&\text{Indle} = \neg S1 \cdot \neg S0;\\
&\text{Delay} = \neg S1 \cdot S0;\\
&\text{Next} = S1 \cdot S0;\\
&\text{Done} = S1 \cdot \neg S0;\\
&S0 := (\text{Indle} \cdot \text{Run}) + \text{Delay};\\
&S1 := (\text{Done} \cdot \neg \text{Run}) + \text{Delay} + \text{Next};\\
&\text{Out1} := \text{Indle} \cdot \text{Run};\\
&\textit{Out2: = Next.}
\end{aligned}
\qquad (5.1)
$$

У представленій системі рівнянь символ "=" позначає комбінаційну схему, відповідальну за перехід за станами, а символ

":=" позначає вихід тригера, необхідний для зберігання коду поточного стану автомата і вихідних сигналів.

Рівняння для вихідного сигналу *Out*1 є функцією як стану, так і вхідного сигналу *Run*. Кінцевий автомат з таким видом стробування виходів називається автоматом Милі.

5.2.2 Синтез функціональної схеми пристрою

5.2.2.1 На основі заданого графа побудуйте таблицю станів розробляємого автомата і таблицю призначених станів (з урахуванням перших чотирьох рівнянь системи (5.1)). Форми таблиць приведені у табл. 5.1 і табл. 5.2. У табл. 5.2 для позначення станів сигналу *Run* використайте тризначну логіку (0, 1, X).

Таблиця 5.1 – Форма таблиці станів автомата

Поточний стан	Наступний стан		Наступні вихідні стани		
			Out1		*Out2*
	Run = 0	*Run* = 1	*Run* = 0	*Run* = 1	*Run* = x
Indle					
Delay					
Next					
Done					

Таблиця 5.2 - Форма таблиці призначених станів автомата

Вхідний сигнал *Run*	Поточний стан		Наступний стан		Вихідні сигнали	
	$S1^n$	$S0^n$	$S1^{n+1}$	$S0^{n+1}$	*Out1*	*Out2*

5.2.2.2 За допомогою карт Карно мінімізуйте рівняння для змінних $S1^{n+1}$, $S0^{n+1}$.

5.2.3 Проєктування і тестування проекту пристрою на основі моделі кінцевого автомата

5.2.3.1 Створення проєкту. Створіть робочу папку для розміщення файлів проєкту. Створіть директорію проєкту.

Створіть графічний файл, для чого на основі приведених рівнянь (5.1) побудуйте схему з використанням графічного редактора MAX+ plus II. У схемі передбачте три вхідні порти (*Run* для введення даних, синхровхід *Clk* і *Reset* для скидання тригерів у початковий стан), чотири вихідні порти: два для виведення однобітових вихідних даних *Out*1 і *Out*2, два для контролю станів автомата (сигнали *S*0, *S*1). У схемі використовуйте *D*-тригери з інверсним входом скидання (чотири *D*-тригери відповідно до останніх чотирьох рівнянь системи логічних рівнянь, а також *D*-тригер, що прив'язує сигнал *Run* до синхроімпульсу).

5.2.3.2 Виконайте компіляцію графічного файлу.

5.2.3.3 Виконайте функціональне моделювання роботи розробленої схеми. На часовій діаграмі повинні бути відображеними три вхідні порти і чотири вихідні порти. Сигнал *Reset* на першому такті повинен бути одиничним, а на наступних тактах – нульовим. Задайте кінцевий час моделювання (приблизно 65 нс). Послідовність вхідних сигналів *Run* задайте відповідно до варіанту (табл. 5.3).

5.2.4 Синтез функціональної схеми пристрою на основі мінімізованих рівнянь

5.2.4.1 Виконайте дії відповідно підпункту 5.2.3.1 з використанням отриманих у підпункті 5.2.2.2 мінімізованих рівнянь.

5.2.4.2 Виконайте дії відповідно до підпунктів 5.2.3.2 та 5.2.3.3.

5.2.4.3 Порівняйте результати моделювання схем на основі немінімізованих та мінімізованих рівнянь.

5.3 Зміст звіту

Звіт повинен містити:

- мету роботи;
- граф автомата з чотирма станами;

– рівняння системи (5.1), заповнені таблиці 5.1 та 5.2;
часові діаграми роботи системи за підпунктами 5.2.3.3 та 5.2.4.2;
– висновки за підпунктом 5.2.4.3.

Таблиця 5.3 - Послідовність вхідних сигналів *Run*

№ вар.	Потактова послідовність вхідних сигналів *Run*	№ вар.	Потактова послідовність вхідних сигналів *Run*
1	0 0 1 1 0 0 1 0 0 1 0 1	16	1 0 0 0 0 0 1 1 1 1 0 1
2	1 1 0 0 1 1 1 1 0 1 0 1	17	1 1 1 0 0 0 1 1 0 1 0 1
3	0 1 0 1 0 0 1 0 1 1 0 1	18	1 0 0 0 0 0 1 0 1 0 0 1
4	1 0 0 1 1 0 0 0 1 1 0 0	19	1 0 1 1 1 0 1 1 1 1 0 0
5	1 0 0 1 0 0 1 1 1 1 0 1	20	1 0 1 0 1 0 1 1 1 0 0 1
6	0 1 1 1 0 0 1 1 0 1 1 0	21	0 0 1 0 1 1 1 1 0 0 0 1
7	1 1 0 1 0 0 0 1 1 0 0 1	22	1 0 1 0 1 0 1 1 1 1 0 1
8	1 0 1 1 0 0 0 1 0 0 0 1	23	1 1 1 0 0 0 1 0 1 0 0 1
9	0 0 0 1 1 0 1 1 1 1 1 0	24	1 0 1 1 0 0 1 1 1 1 0 1
10	1 1 1 0 0 0 0 1 0 1 0 1	25	1 0 0 0 0 0 1 0 0 1 0 1
11	1 0 1 1 1 0 1 0 1 1 0 0	26	1 1 0 1 0 1 1 1 0 1 0 1
12	0 0 1 1 0 0 1 0 0 1 0 1	27	1 1 0 1 0 0 0 1 1 1 0 1
13	1 1 1 0 0 1 0 1 0 0 0 1	28	1 0 1 1 0 0 1 1 1 1 0 1
14	1 0 1 0 1 0 1 0 1 1 0 1	29	1 1 1 0 0 0 1 1 0 0 0 1
15	1 1 0 1 0 0 1 0 0 1 1 0	30	1 0 1 0 1 0 1 0 0 1 0 1

5.4 Контрольні запитання

1. Які особливості мають ПЛІС?
2. Які особливості мають ПЛМ?
3. Навіщо на платі поруч з мікросхемою типу FPGA розташовують ПЗП?
4. Які переваги мають ПЛІС порівняно з CPLD?
5. Зобразіть граф автомата, який визначає наявність у числі парної кількості одиниць або їх відсутність.

6. Скільки тригерів повинно бути у схемі автомата, яка його реалізовує, якщо автомат має 9 станів?
7. Які порти має програма з додатку Б?
8. Наведіть алфавіт тризнакової логіки.
9. У яких випадках автомат може знаходитись у приймаючому стані?
10. Навіщо у розробленій Вами схемі використовується вхід *Reset*?
11. Як за складністю відрізняються схеми, що побудовані на основі немінімізованих та мінімізованих рівнянь?
12. З яких етапів складається розробка проєкту пристрою?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. **Діденко, Ю.В.**, Проєктування напівпровідникових приладів та інтегральних мікросхем. Навчальний посібник / Ю.В. Діденко, А.Т. Орлов, Д.Д. Татарчук. - Видавництво КПІ ім. Ігоря Сікорського, 2022. - 164 с.

2. **Макаренко, В.В.** Моделювання та аналіз цифрових схем / В.В. Макаренко, Є.З. Маланчук, А.В. Рудик, В.М. Співак. - Рівне: НУВГТІ, 2017. - 454 с.

Додаток А

VHDL – код, що реалізує алгоритм роботи системи *S*

```
entity vlsi_1 is                                    --VHDL – код для опису системи S
port (a2,a1,b2,b1,x: in BIT;
    d4,d3,d2,d1: out BIT);
end vlsi_1;
architecture structure of vlsi_1 is
component adder_2                                   --декларація компонента
port (a1,b1,a2,b2: in BIT;
       c2,s2,s1: out BIT);
end component;
component mult_2                                    --декларація компонента
port (s1,s0,r1,r0: in BIT;
      t3,t2,t1,t0: out BIT);
end component;
component dd                                        --декларація компонента
port (x1,x2,x3,x4,x5,x6: in BIT;
      y1,y2,y3: out BIT);
end component;
component yy                                        --декларація компонента
port (a2,a1,b2,b1,x: in BIT;
    f6,f5,f4,f3,f2,f1: out BIT);
end component;
signal f1,f2,f3,f4,f5,f6,t4,t3,t2,t1, c2, s2, s1: BIT;   --декларація внутрішніх сигналів
begin
    circ1: yy
    port map(a2,a1,b2,b1,x,f6,f5,f4,f3,f2,f1);
    circ2: mult_2
    port map(b2,b1,f2,f1,d4,t3,t2,t1);
    circ3: adder_2
    port map(f4,f3,f6,f5,c2,s2,s1);
    circ4: dd
    port map(s1,t1,s2,t2,c2,t3,d1,d2,d3);
end structure;
entity adder_2 is                                   --опис об'єкта adder_2
port (a1,b1,a2,b2: in BIT;
      c2,s2,s1: out BIT);
end adder_2;
architecture structure of adder_2 is
component
add1
port (b1,b2: in BIT;
       c1,s1: out BIT);
end component;
component add2
port (c1,a1,a2: in BIT;
       c2,s2: out BIT);
end component;
signal c1: BIT;
begin
    circ1: add1
    port map(b1,b2,c1,s1);
    circ2: add2
    port map(c1,a1,a2,c2,s2);
end structure;
entity mult_2 is                                    -- опис об'єкта  mult_2
port (s1,s0,r1,r0: in BIT;
```

```
        t3,t2,t1,t0: out BIT);

    end mult_2;
    architecture structure of mult_2 is
    component
    add1 port (b1,b2: in BIT;
          c1,s1: out BIT);
    end component;
    signal p1,p2,p3,p4: BIT;
    begin
        t0 <= r0 and s0;                          -- элемент el_1;
        p2 <= r0 and s1;                          -- элемент el_3;
        p1 <= r1 and s0;                          -- элемент el_2;
        p4 <= r1 and s1;                          -- элемент el_4;
        circ1: add1 port map (p1,p2,p3,t1);
        circ2: add1 port map (p3,p4,t3,t2);
    end structure;
    entity dd is                                          -- опис об'єкта  dd
    port (x1,x2,x3,x4,x5,x6: in BIT;
          y1,y2,y3: out BIT);
    end dd;
    architecture struct_2 of dd is
    begin
                y1 <= x1 or x2;
                y2 <= x3 or x4;
                y3 <= x5 or x6;
    end struct_2;
    entity add1 is                                -- опис об'єкта  add1
    port (b1, b2: in BIT;
           c1, s1: out BIT);
    end add1;
    architecture struct_3 of add1 is
    begin
        s1 <= ((b1 and (not b2)) or ((not b1) and b2));
        c1 <= b1 and b2;
    end struct_3;
    entity yy is                                          -- опис об'єкта  yy
    port (a2,a1,b2,b1,x: in BIT;
           f6,f5,f4,f3,f2,f1: out BIT);
    end yy;
    architecture struct_4 of yy is
    begin
                f1 <= x and a1;
                f2 <= x and a2;
                f3 <= not x and a1;
                f4 <= not x and a2;
                f5 <= not x and b1;
                f6 <= not x and b2;
    end struct_4;
    entity add2 is                                -- опис об'єкта  add2
    port (c1, a1, a2: in BIT;
           c2, s2: out BIT);
    end add2;
    architecture struct_6 of add2 is
    begin
    s2<=((not c1) and (not a1) and a2)or((not c1) and a1 and (not a2)) or (c1 and (not a1) and (not a2)) or (c1 and
a1 and a2) ;
    c2<=((not c1) and  a1 and a2) or ( c1 and (not a1) and a2) or (c1 and a1 and (not a2)) or (c1 and a1 and a2) ;
    end struct_6;
```

Додаток Б

Автомат Мілі з п’ятьма станами

```
library ieee;        -- Автомат Мілі з 5 станами
use ieee.std_logic_1164.all;
entity mealy is
port (clock, reset: in std_logic;
data_out: out std_logic;
data_in: in std_logic_vector (1 downto 0));
end mealy;
architecture behave of mealy is
type state_values is (st0, st1, st2, st3, st4);
signal pres_state, next_state: state_values;
begin
statereg: process (clock, reset)     --FSM регістр
begin
if (reset = '0') then
pres_state <= st0;
elsif (clock'event and clock = '1') then
pres_state <= next_state;
end if;
end process statereg;
fsm; process (pres_state, data_in)
begin
case pres_state is
when st0 =>
case data_in is
when "00" => next_state <= st0;
when "01" => next_state <= st4;
when "10" => next_state <= st1;
when "11" => next_state <= st2;
when others => next_state <= st0;
end case;
when st1 =>
case data_in is
when "00" => next_state <= st0;
when "10" => next_state <= st2;
when others => next_state <= st1;
end case;
when st2 =>
case data_in is
when "00" => next_state <= st1;
when "01" => next_state <= st1;
```

```
when "10" => next_state <= st3;
when "11" => next_state <= st3;
when others => next_state <= st0;
end case;
when st3 =>
case data_in is
when "01" => next_state <= st4;
when "11" => next_state <= st4;
when others => next_state <= st3;
end case;
when st4 =>
case data_in is
when "11" => next_state <= st4;
when others => next_state <= st0;
end case;
when others => next_state <= st0;
end case;
end process fsm;
outputs: process (pres_state, data_in)
begin
case pres_state is
when st0 =>
case data_in is
when "00" =>  data_out <= '0';
when others => data_out <= '1';
end case;
when st1 => data_out <= '0';
when st2 =>
case data_in is
when "00" =>  data_out <= '0';
when "01" =>  data_out <= '0';
when others => data_out <= '1';
end case;
when st3 => data_out <= '1';
when st4 =>
case data_in is
when "10" =>  data_out <= '1';
when "11" =>  data_out <= '1';
when others => data_out <= '0';
end case;
when others => data_out <= '0';
end case;
end process outputs;
end
```