Міністерство освіти і науки України

ДЕРЖАВНИЙ БІОТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Інститут «Кіберпорт»

Кафедра автоматизації та комп'ютерно-інтегрованих технологій

С. О. Тимчук, А. О. Панов

Методичні вказівки до виконання практичних робіт з дисципліни «Теорія цифрових автоматів»

для здобувачів другого (магістерського) рівня вищої освіти денної та заочної форм навчання за освітньо-професійною програмою зі спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології

Харків
2023

Міністерство освіти і науки України

ДЕРЖАВНИЙ БІОТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Інститут «Кіберпорт»

Кафедра автоматизації та комп'ютерно-інтегрованих технологій

С. О. Тимчук, А. О. Панов

Методичні вказівки до виконання практичних робіт з дисципліни «Теорія цифрових автоматів»

для здобувачів другого (магістерського) рівня вищої освіти денної та заочної форм навчання
за освітньо-професійною програмою зі спеціальності
151 Автоматизація та комп'ютерно-інтегровані технології

Затверджено
рішенням науково-методичної ради
інституту «Кіберпорт»
Протокол № 6
від «04» травня 2023 року

Харків
2023

УДК 510:621.9
Т 41

Схвалено на засіданні кафедри
автоматизації та комп'ютерно-інтегрованих технологій
Протокол № 8 від 28.04. 2023 р.

Рецензенти:

С. Я. Бовчалюк, канд. техн. наук, доцент кафедри електронних обчислювальних машин ХНУРЕ.
М. П. Кунденко, д-р. техн. наук, професор, зав. кафедри теплотехніки та енергоефективних технологій НТУ 'ХПІ'.

Т 41 Теорія цифрових автоматів: методичні вказівки до виконання практичних робіт з дисципліни «Теорія цифрових автоматів» для здобувачів другого (магістерського) рівня вищої освіти денної та заочної форм навчання за освітньо-професійною програмою зі спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» / С. О. Тимчук, А. О. Панов / - Електрон. дані. – Х.: ДБТУ, 2023. – 50 с.

Методичні вказівки включають 8 практичних робіт. Матеріал розкриває сутність реалізації процесу програмування інформаційних пристроїв дискретної дії. Майбутні фахівці повинні володіти теорією цифрових автоматів.

Видання призначене студентам другого (магістерського) рівня вищої освіти денної та заочної форм навчання спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології.

УДК 510:621.9

Відповідальний за випуск: С. О. Тимчук, д-р техн. наук, професор

ЗМІСТ

ВСТУП

Цифровий автомат (ЦА) - це умовна, формальна модель будь-якого інформаційного пристрою дискретного дії. Модель призначена для представлення пристрою на логічному рівні, тобто в ній не допускається використання будь-яких фізичних характеристик процесів, що відбуваються при роботі реального пристрою.

У зв'язку з таким спрощенням число помітних станів, в яких може опинитися ЦА, завжди обмежена. Перехід з одного стану в інший мислиться як миттєвий стрибок, без всяких проміжних станів, тоді як в реальній фізичній системі відповідний перехід повинен мати безліч перехідних фаз з зникаюче малими відмінностями подальшого стану від попереднього.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Мова AlteraHDL (AHDL) - мова опису апаратури, створений фірмою Altera в 1983 році. До теперішнього часу, пройшовши довгий шлях розвитку, він є ефективним засобом для:

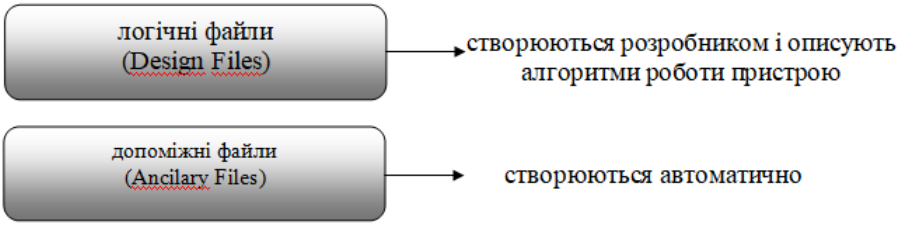
- поведінкового і структурного опису проектованого пристрою;
- документування проекту.

Система синтезу пакету MAX + plusII, в яку мову AHDL інтегрований, забезпечує його архітектурну незалежність: текстовий опис може бути без переробки синтезовано на базі будь-якої НВІС програмованої логіки, включеної в пакет MAX + plusII.

На відміну від таких мов як VHDL і VerilogHDL, орієнтованих на опис і моделювання системи (наприклад: процесор-пам'ять-проектована пристрій), мова AHDL більш простий у вивченні і оптимізований за своїми можливостями для проектування окремої НВІС. Однак, незважаючи на свою відносну простоту, він містить типовий для сучасних мов опису апаратури набір високорівневих конструкцій.

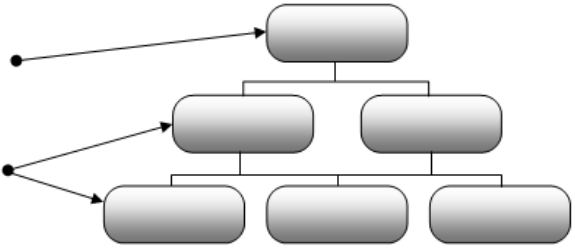
Проект

Під терміном "проект" в рамках пакету MAX + plusII розуміється набір файлів, пов'язаних з проектованим модулем, в якому виділяються дві групи файлів:



Проект може містити один логічний файл або кілька логічних файлів, що утворюють ієрархічний опис проектованого модуля. При ієрархічному описі серед безлічі логічних файлів розрізняють:

- файл верхнього рівня в ієрархії описів (Top –level Design Files);
- файли нижніх (одного або декількох) рівнів ієрархії (Low-level Design Files).



У файлі верхнього рівня задається архітектура модуля, визначається набір модулів, що входять до його складу як компоненти, і їх взаємозв'язок. Описи цих модулів містяться в логічних файлах нижчого рівня ієрархії. До їх складу, в свою чергу, у вигляді компонентів також можуть входити модулі, опису яких наведені в логічних файлах ще більш низького рівня ієрархії, і т.д.

Ім'я проекту має збігатися з ім'ям модуля верхнього рівня в ієрархії описів, а отже, і ім'ям логічного файлу, в якому зберігається його опис. Імена модулів нижніх рівнів ієрархії, в свою чергу, повинні збігатися з іменами файлів, в яких вони описані.

Логічний файл - це файл одного з наступних типів:

- **Graphic Design File** (стандартне розширення - **.gdf**)

Файл містить схему, створену в рамках пакету MAX + plusII;

- **AHDL Text Design File** (стандартне розширення - **.tdf**)

Файл містить текстовий опис модуля на мові AlteraHDL

- **Waveform Design File** (Спосіб стандартного розширення - **.wdf**)

Файл містить тимчасові діаграми вхідних і вихідних сигналів, створені в рамках пакету MAX + plusII;

- **VHDL Design File** (стандартне розширення - **.vdf**)

Файл містить текстовий опис модуля на мові VHDL;

- **Verilog Design File** (стандартне розширення - **.v**)

Файл містить текстовий опис модуля на мові VerilogHDL;

- **Orcad Schematic Files** (стандартне розширення - .**sch**);

Файл містить схему, створену в рамках пакету ORCAD;

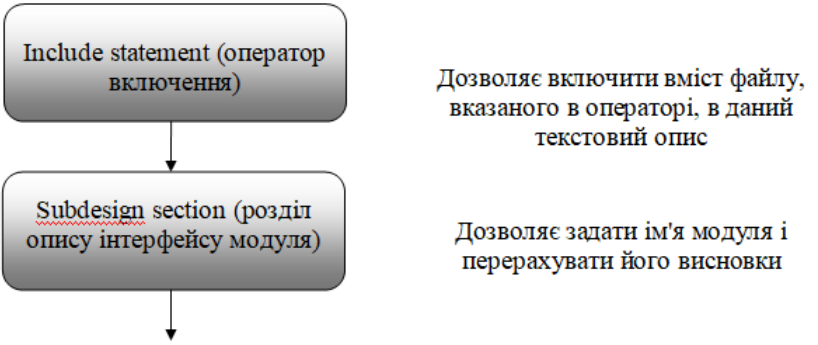
- **EDIF Input Files** (стандартне розширення - **.edf**);

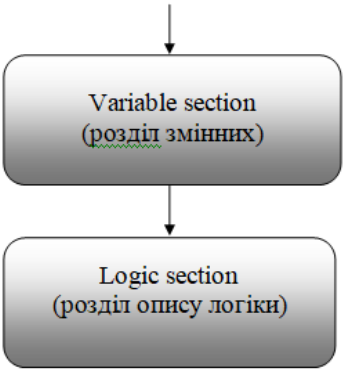
Файл містить опис в форматі EDIF 200 або 300;

- **Xilinx Netlist Format File** (стандартне розширення - **.xnf**);

Файл містить опис модуля, отримане в рамках пакету фірми Xilinx.

Типова структура програми на мові AHDL має наступний вигляд:

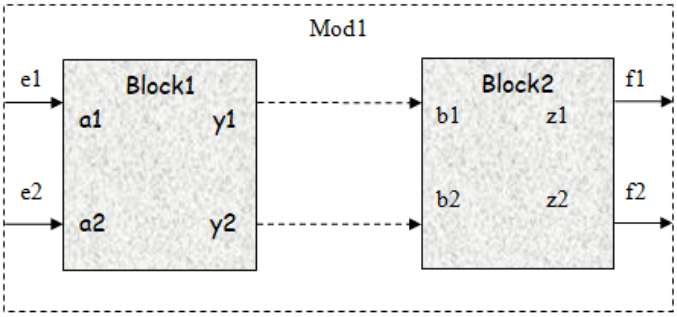




Дозволяє задавати внутрішні змінні модуля, призначені для використання в розділі опису логіки

Дозволяє задати алгоритм роботи проектованого модуля і встановити взаємозв'язок між включеними в поточний проект модулями нижніх рівнів

Наприклад, модулю (назвемо його mod1), який має структуру, представлену на малюнку:



відповідає наступний текстовий опис:

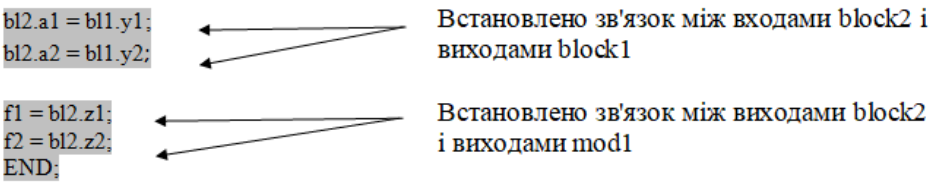
```
INCLUDE "block1.inc";
INCLUDE "block2.inc";

SUBDESIGN mod1
(
        e1, e2      : INPUT;
        f1, f2      : OUTPUT;
)

VARIABLE
bl1 : block1;
bl2 : block2;

BEGIN
bl1.a1 = e1;
bl1.a2 = e2;
```

Файли з назвами block1 і block2, які були описані раніше, будуть включені в поточний текстовий опис

Задано ім'я модуля
Задані входи
Задані виходи

Імена bl1 і bl1 будуть використані для позначення модулів block1 і block2

Встановлено зв'язок між входами block1 і входами mod1



Правила використання Include statement:

- Оператор починається з ключового слова INCLUDE, за яким в подвійних лапках (") вказується ім'я файлу включення (Include file). Далі ставиться крапка з комою (;).
- Якщо явно не задано розширення файлу включення, то компілятор шукає файл, який має задане ім'я і розширення .inc.
- Файл, вказане в операторі включення, не повинно містити шляху до файлу.
- У файлі з текстовим описом даний оператор може використовуватися необмежену кількість разів.

Правила використання Subdesign section:

- Розділ починається з ключового слова SUBDESIGN, за яким слідує ім'я модуля. Максимальна довжина імені 32 символу. Ім'я модуля повинно співпадати з ім'ям файлу, в якому зберігається його текстовий опис.
- Після імені модуля слід взятий у круглі дужки список його висновків.
- Типи висновків: INPUT - вхід, OUTPUT - вихід, BIDIR - двонаправлений висновок, MASHINE INPUT - вхід імпортованих станів автомата, MASHINE OUTPUT - вихід експортованих станів автомата. Висновки MASHINE INPUT і MASHINE OUTPUT не можуть бути використані в текстовому описі верхнього рівня ієрархії, тобто при описі модуля, висновки якого є висновками HBIC.
- Висновки перераховуються через кому в одну або кілька рядків. В кінці переліку однотипних висновків ставиться двокрапка, потім ключове слово, яке вказує тип висновків, і далі - крапка з комою.
- Після слова INPUT може бути зазначено базове значення (GND - логічний нуль, VCC - логічна одиниця) вхідного сигналу. Базове значення - значення, яке буде подаватися на вхід в тому випадку, якщо він виявиться непідключеним при використанні даного модуля як компонента при описі модуля більш високого рівня ієрархії. Якщо непідключеним виявиться вхід, для якого не вказано базове значення, то компілятор пакета MAX + plus II видасть повідомлення про помилку.

➢ У файлі з текстовим описом даний розділ може використовуватися тільки один раз.

Правила використання Variable section:

➢ Розділ починається з ключового слова VARIABLE. Далі вказується: символічне ім'я змінної, символ двокрапка, тип змінної, крапка з комою. Імена однотипних змінних можуть бути перераховані через кому.

➢ Допустимі типи змінних: NODE - лінія зв'язку; TRI_STATE_NODE - лінія зв'язку з трьома станами; модуль нижчого рівня ієрархії; примітив; кінцевий автомат; псевдонім кінцевого автомата.

➢ Розділ змінних може також містити оператор IF GENERATE, що дозволяє, в залежності від значення оцінюваного в операторі арифметичного виразу, управляти завданням змінних.

➢ У файлі з текстовим описом даний розділ може використовуватися тільки один раз.

Правила використання Logic section:

➢ Розділ починається з ключового слова BEGIN і закінчується ключовим словом END, за яким слідує крапка з комою.

➢ У розділі можуть бути використані:

o **Defaults Statement** (оператор завдання вихідних значень);

o **Boolean Equations** (логічні рівняння);

o **Boolean Control Equations** (логічні рівняння для керуючих сигналів);

o **Case Statement** (Оператор Case);

o **If Then Statement** (оператор If Then);

o **If Generate Statement** (оператор If Generate);

o **In-Line Logic Function reference** (безпосереднє звернення до модулів нижнього рівня в ієрархії описів і примітивам);

o **For Generate Statement** (оператор For Generate);

o **Truth Table Statement** (таблиця істинності);

o **Assert Statement** (оператор контролю).

➢ У файлі з текстовим описом даний розділ може використовуватися тільки один раз.

Для того щоб створити проект в «MAX + plus II» необхідно виконати наступні дії:

1. Увійдіть в меню File → New, виберіть текстовий редактор (Text Editor File) і натисніть OK (рис. 1).

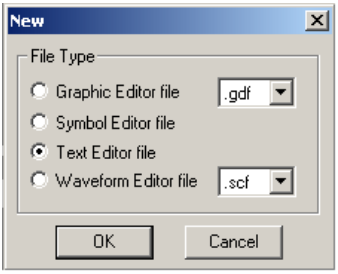


Рисунок 1

2. У вікні, введіть опис, наведене на рис. 2.

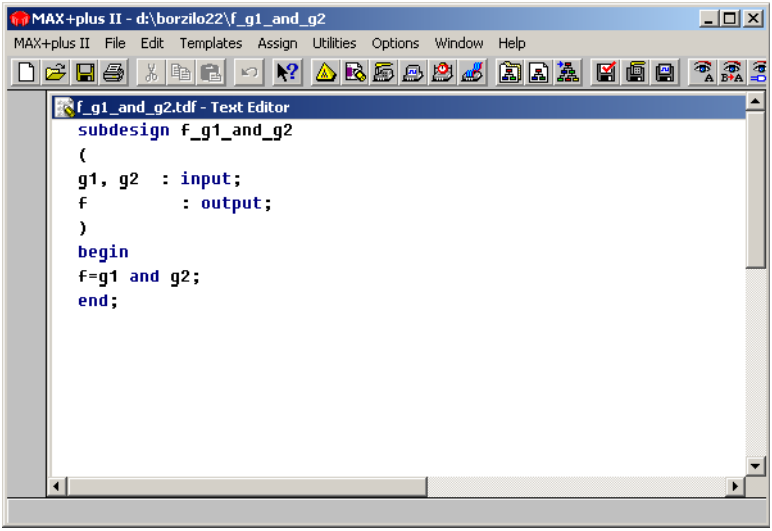


Рисунок 2

3. Збережіть даний файл з дозволом .tdf. При цьому назва файлу повинна збігатися з назвою модуля, яке зазначено після службового слова SUBDESIGN (f_g1_and_g2), а назви папок, в яких зберігається файл, повинні містити тільки латинські букви.

4. Для того, щоб при виконанні компіляції компілятор звертався до файлу f_g1_and_g2, зайдіть в меню File → Projects → Set Projects To Current File.

5. Перед тим, як приступити до компіляції проекту, Ви можете встановити наступні призначення:

- вибрати тип кристала ПЛІС (меню Assign → Device) (Рис. 3);
- призначити номери висновків ПЛІС, до яких будуть підключені сигнали проектованого модуля (меню Assign → Pin / Location / Chip) (Рис. 4).

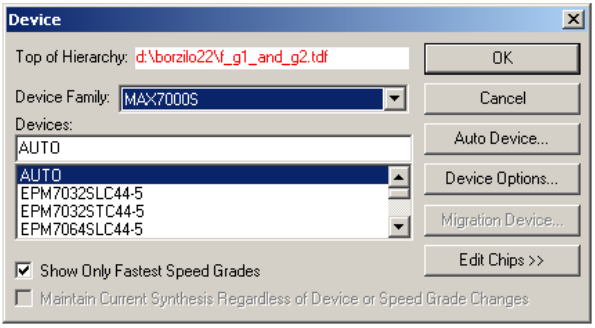


Рисунок 3

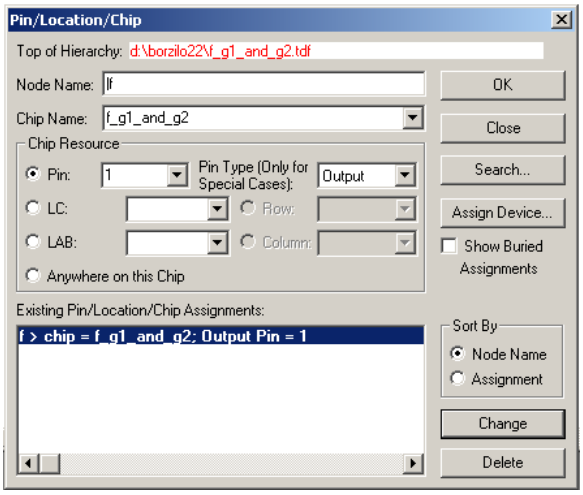


Рисунок 4

6. Відкрийте вікно компілятора (меню MAX + plus II → Compiler) і натисніть кнопку Start (Рис. 5).

7. Для оцінки часу затримки формування вихідного сигналу f скористайтеся тимчасовим аналізатором (меню MAX + plus II → Timing analyzer) (Рис. 6).

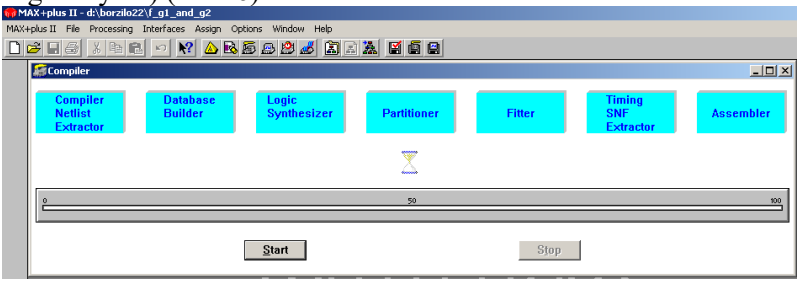


Рисунок 5

8. Увійдіть в меню MAX + plus II → Waveform editor і клацніть правою кнопкою мишки на робочій зоні редактора тимчасових діаграм. У відкрилася посиланням виберіть Enter Nodes From SNF. У вікні, клікніть на кнопці List, потім на кнопці з позначенням => і натисніть ОК (рис. 7). При цьому в редакторі з'являться рядки для відображення значень сигналів g1, g2 і f.

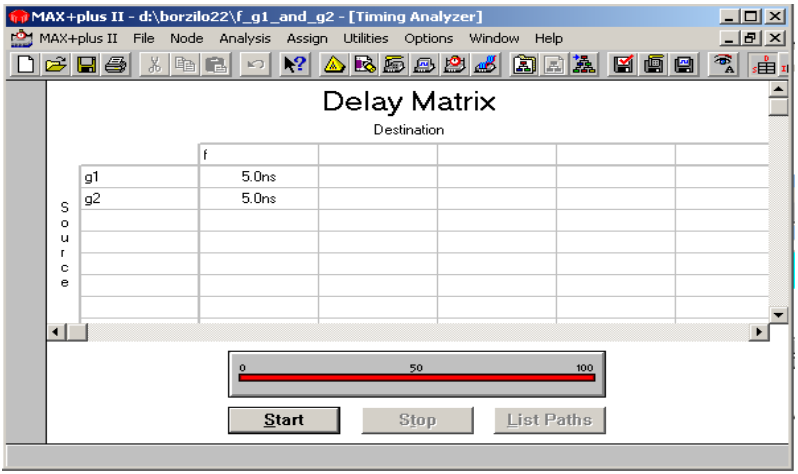


Рисунок 6

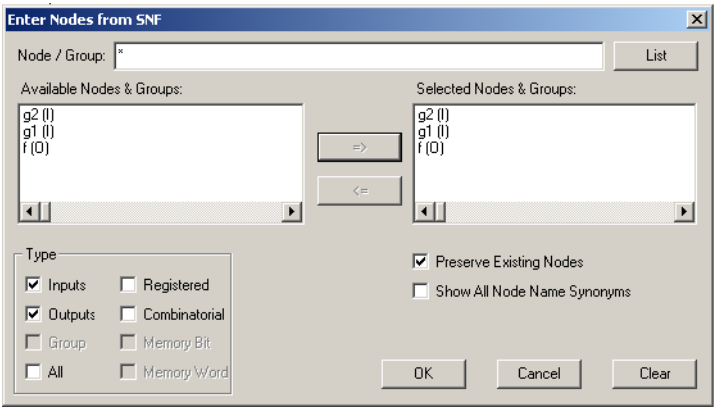


Рисунок 7

9. Збережіть файл f_g1_and_g2.scf, відкритий в редакторі тимчасових діаграм.

10. Задайте за допомогою інструментів, доступних на панелі, розташованій ліворуч, значення сигналів g1 і g2, і потім, для отримання результатів моделювання, запустіть стимулятор (меню MAX + plus II → Simulator) (рис. 8).

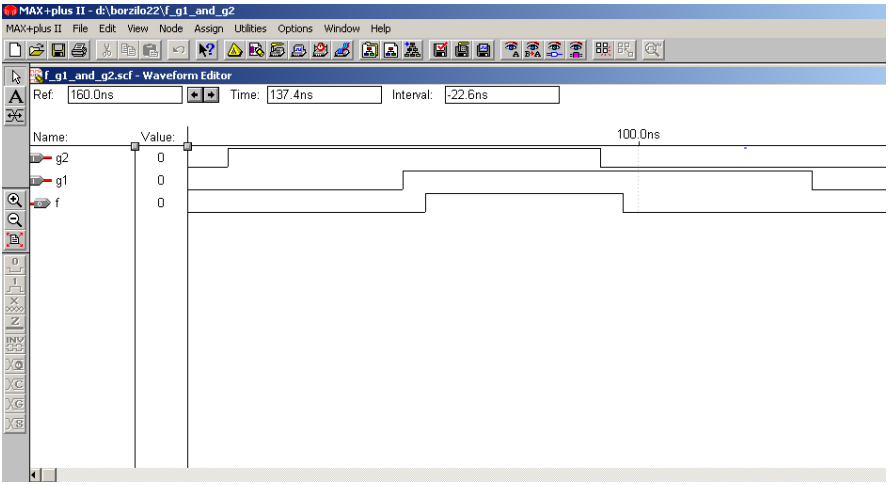


Рисунок 8

Таким чином, нам вдалося створити проект цифрового пристрою на основі ПЛІС, яке реалізує кон'юнкцію двох змінних.

Практичне заняття № 1
"Реалізація логічних функцій на AHDL"

Мета роботи: отримання практичних навичок у складанні програм реалізації логічних функцій.

Основні конструкції мови AHDL, що застосовуються для реалізації логічних функцій.

Способи реалізації логічних функцій розглянемо на конкретних прикладах. Нехай задана система логічних рівнянь:

$$\begin{cases} Out1 = a1 \wedge \overline{a0}; \\ Out2 = a1 \wedge \overline{a0} + b. \end{cases}$$

Для їх реалізації можливе використання наступного текстового опису:

```
SUBDESIGN boole_1
(
    a0, a1, b                 :INPUT;
    out1, out2                :OUTPUT;
)
BEGIN
      Out1=a1&!a0;
      Out2=out1#b;
END;
```

Представлене текстовий опис включає мінімально необхідний набір розділів:

- розділ інтерфейсу Subdesign section;
- розділ опису логіки Logic section.

У рівняннях використовувалися символи логічного множення "&", логічного додавання "#" і інверсії "!".

Однотипні змінні або висновки можуть бути об'єднані в *одновимірну*, *двовимірну* або *тимчасові* групи.

Застосування одновимірних груп ілюструється наступним прикладом:

```
SUBDESIGN group_1
(
    a[3..0],
    b[4..1],
    c, d, e, f                :INPUT;
    out[5..2]                 :OUTPUT;
)
BEGIN
      out[]=(a[]#b[1..4])&!(c, d, e, f);
 END;
```

У прикладі використано поразрядне застосування операторів #, &,!, Тому кількість розрядів в групах має бути однаково.

При посиланні на групу b її індекси перераховані в зворотному порядку, в зв'язку з чим компілятор сформує відповідне попередження.

Скорочена форма записи в групах (a []) застосовується при використанні всіх її членів і їх розташувань в тому ж порядку, в якому вони були задані.

При використанні двовимірних груп текстовий опис може виглядати наступним чином:

```
SUBDESIGN group_2
(
   a[2..1]                 :INPUT;
   b[1..0][2..1]           :OUTPUT;
)
BEGIN
      b[][]=(a[1..2], a[]);
 END;
```

Відзначимо, що групи зліва і праворуч від знака рівності мають однакове число розрядів.

Порядок виконання роботи:

1. Реалізувати проекти boole_1, group_1, group_2; виконати їх компіляцію і тимчасове моделювання.

2. Вирішити індивідуальне завдання.

Індивідуальні завдання.

1. $\begin{cases} Out1 = a1 \wedge \overline{a0 + a2} + \overline{\overline{a3 + a4} \wedge a5}; \\ Out2 = \overline{\overline{a2 \wedge a1} + \overline{a0 + a3}} + \overline{a4}; \\ Out3 = a4 + a2 + \overline{a1 \wedge a0} \wedge \overline{\overline{a3 \wedge a6} + \overline{a5 \wedge a7}}. \end{cases}$; $\begin{cases} z[3..1] = (a[2..0] \wedge \overline{b}) + c[4..2]; \\ y[4..0] = (d[5..1] + \overline{e[7..3]}) \wedge c[8..4]. \end{cases}$

2. $\begin{cases} Out1 = a3 + \overline{a0 \wedge \overline{a1} + a2} \wedge \overline{\overline{a6 \wedge a5} + \overline{a4} \wedge a7}; \\ Out2 = a1 \wedge \overline{\overline{a2 + a4} \wedge a0} + \overline{a3}; \\ Out3 = a0 + \overline{\overline{a2 \wedge a1} + \overline{a3 + a4}} + \overline{a6 \wedge \overline{a5}}. \end{cases}$; $\begin{cases} z[3..1] = (a[2..0] \wedge \overline{b}) \wedge c[4..2] \wedge m; \\ y[4..0] = (e[8..4] \wedge \overline{c[7..3]}) + d[5..1]. \end{cases}$

3. $\begin{cases} Out1 = \overline{a6 + \overline{a0 \wedge \overline{a1}} + a2} \wedge \overline{a3 \wedge a5} + \overline{a7} \wedge a4; \\ Out2 = a4 \wedge \overline{\overline{a2 + a1} \wedge a3} + \overline{a0} \wedge a5; \\ Out3 = \overline{\overline{a2 \wedge a1} + \overline{a0 + a5}} + \overline{a4 \wedge \overline{a6}} + a3. \end{cases}$; $\begin{cases} z[0..3] = c[4..1] \wedge m + (a[3..0] \wedge \overline{b}); \\ y[8..6] = (e[2..4] + \overline{c[1..3]}) \wedge d[5..3]. \end{cases}$

4. $\begin{cases} Out1 = \overline{a3 \wedge a5} + \overline{a7} \wedge a4 + a6 + \overline{a0 \wedge \overline{a1} + a2}; \\ Out2 = \overline{\overline{a2 + a1} \wedge a3} + \overline{\overline{a0} \wedge a5 + \overline{a4}}; \\ Out3 = \overline{a2 \wedge a1} + \overline{a3} + \overline{a4 \wedge \overline{a6}} \wedge a0 + a5. \end{cases}$; $\begin{cases} z[6..3] = a[4..1] \wedge m + (c[3..0] \wedge b); \\ y[4..6] = (d[2..4] + c[1..3]) \wedge e[5..3]. \end{cases}$

5. $\begin{cases} Out1 = \overline{a3 \wedge a5} + \overline{\overline{a7} \wedge a4} + a6 + \overline{a0 \wedge \overline{a1} + a2}; \\ Out2 = \overline{\overline{a2 + a1} \wedge a3} + \overline{a0} \wedge a5 + \overline{a4}; \\ Out3 = \overline{a2 \wedge a1} + \overline{a3} + \overline{a4 \wedge \overline{a6}} \wedge a0 + a5. \end{cases}$; $\begin{cases} z[0..1] = c[4..5] \wedge m + (a[3..2] \wedge b); \\ y[8..6] = (e[2..4] + g[1..3]) \wedge d[1..3]. \end{cases}$

6. $\begin{cases} Out1 = \overline{\overline{a7} \wedge a4} + a6 + \overline{a0 \wedge \overline{a1} + a2} + \overline{a3 \wedge a5}; \\ Out2 = \overline{a0} + \overline{\overline{a2 + a1} \wedge a3} \wedge a5 + \overline{a4}; \\ Out3 = \overline{a3} + \overline{a2 \wedge a1} + \overline{a4 \wedge \overline{a6}} \wedge a0 + a5. \end{cases}$; $\begin{cases} z[3..4] = c[6..5] \wedge m + (a[3..2] \wedge b); \\ y[7..6] = (e[2..3] + g[1..2]) \wedge d[2..3]. \end{cases}$

7. $\begin{cases} Out1 = \overline{\overline{a7} \wedge a4} + a6 + \overline{\overline{a0 \wedge \overline{a1} + a2} + \overline{a3 \wedge a5}}; \\ Out2 = a5 + \overline{a0} + \overline{\overline{a2 + a1} \wedge a3} + \overline{a4}; \\ Out3 = \overline{a3} + \overline{a4 \wedge \overline{a6}} \wedge a0 + a5 \wedge \overline{a2 \wedge a1}. \end{cases}$; $\begin{cases} z[5..0] = a[6..1] \wedge m + (c[7..2] \wedge b); \\ y[1..0] = (e[6..7] + g[8..7]) \wedge d[2..3]. \end{cases}$

8. $\begin{cases} Out1 = a1 \wedge \overline{a0 + a2} + \overline{\overline{a3 + a4} \wedge a5}; \\ Out2 = \overline{\overline{a2 \wedge a1} + \overline{a0 + a3}} + \overline{a4}; \\ Out3 = a4 + a2 + \overline{a1 \wedge a0} \wedge \overline{\overline{a3 \wedge a6} + \overline{a5 \wedge a7}}. \end{cases}$; $\begin{cases} z[3..1] = (a[2..0] \wedge \overline{b}) + c[4..2]; \\ y[4..0] = (d[5..1] + e\overline{[7..3]}) \wedge c[8..4]. \end{cases}$

9. $\begin{cases} Out1 = a3 + \overline{a0 \wedge \overline{a1} + a2} \wedge \overline{\overline{a6 \wedge a5} + \overline{a4} \wedge a7}; \\ Out2 = a1 \wedge \overline{\overline{a2 + a4} \wedge a0} + \overline{a3}; \\ Out3 = a0 + \overline{\overline{a2 \wedge a1} + \overline{a3 + a4}} + \overline{a6 \wedge \overline{a5}}. \end{cases}$; $\begin{cases} z[3..1] = (a[2..0] \wedge \overline{b}) \wedge c[4..2] \wedge m; \\ y[4..0] = (e[8..4] \wedge \overline{c[7..3]}) + d[5..1]. \end{cases}$

10. $\begin{cases} Out1 = \overline{a6 + \overline{a0 \wedge \overline{a1} + a2} \wedge \overline{a3 \wedge a5}} + \overline{a7} \wedge a4; \\ Out2 = a4 \wedge \overline{\overline{a2 + a1} \wedge a3} + \overline{a0} \wedge a5; \\ Out3 = \overline{\overline{a2 \wedge a1} + \overline{a0 + a5}} + \overline{a4 \wedge \overline{a6}} + a3. \end{cases}$; $\begin{cases} z[0..3] = c[4..1] \wedge m + (a[3..0] \wedge \overline{b}); \\ y[8..6] = (e[2..4] + \overline{c[1..3]}) \wedge d[5..3]. \end{cases}$

Практичне заняття № 2
"Оператори IF THEN, TABLE і CASE"

Мета роботи: отримання практичних навичок у складанні програм з використанням операторів TABLE, IF THEN і CASE.

Таблиця істинності (Truth Table Statement) логічної функції задається наступним чином:

```
TABLE
_node_name,_node_name =>_ node_name,_node_name;
_input_value,_input_value =>_output_value,_output_value;
_input_value,_input_value =>_output_value,_output_value;
_input_value,_input_value =>_output_value,_output_value;
END TABLE;
```

Відкриває таблицю ключове слово TABLE, а закривають ключові слова END TABLE, за якими слід крапка з комою.

Перша після слова TABLE рядок визначає форму таблиці. У ній через кому перераховуються аргументи (внутрішні змінні, входи або виходи модуля) і імена формованих логічних функцій (внутрішні змінні або виходи модуля). Аргументи і функції розділяє символ стрілка (=>). В кінці рядка ставиться крапка з комою.

У наступних рядках відповідно до заданої форми вказуються набори аргументів і значення логічних функцій.

Розглянемо використання таблиці істинності на прикладі опису пріоритетного шифратора.

```
SUBDESIGN prior
(
   high, middle, low                :INPUT;
   highest_level[1..0]              :OUTPUT;
)
BEGIN
     TABLE
   high, middle, low  =>  highest_level[1..0];
     1,  x,  x  =>  B"11" ;
     0,  1,  x  =>  B"10" ;
     0,  0,  1  =>  B"01" ;
     0,  0,  0  =>  B"00" ;
END TABLE;
END;
```

Вхід high в даному прикладі має найвищий пріоритет і при появі на ньому логічної "1" на виходах встановлюється комбінація "11" незалежно від того, які сигнали при цьому надійшли на інші входи. Входи middle і low мають, відповідно середній і нижчий

пріоритети і формують на виходах комбінації "10" і "01".

В даному прикладі використаний символ "x" - do not care, т. Е. Значення сигналу не враховується в цьому рядку таблиці (може бути будь-яким).

Оператор IF THEN дозволяє послідовно оцінити істинність декількох логічних виразів і відповідно до отриманих результатів виконати ті чи інші дії.

```
IF <group> THEN
        <statements>
.   {ELSIF <rgroup> THEN
            <statements>}
    [ELSE <rgroup>  THEN
            <statements>]
     END IF;
```

Вираз укладену в дужки {...} може вводиться, кілька разів, а вираз укладену в дужки [...] вводиться один раз або може бути відсутнім.

Приклад:

```
SUBDESIGN if_then
(
   high, middle, low                :INPUT;
   highest_level[1..0]              :OUTPUT;
)
BEGIN
        IF high==1
            THEN highest_level[1..0]=3;
         ELSIF middle == 1
              THEN highest_level[1..0]=2;
         ELSIF low == 1
              THEN highest_level[1..0]=1;
          ELSE highest_level[1..0]=0;
    END IF;
    END;
```

Відзначимо, що оператору IF THEN внутрішньо притаманна пріоритетність. Так, перевіряється першим вхід high має вищий пріоритет: якщо high = 1, то при будь-яких значеннях middle і low (їх значення навіть не перевіряються) групі виходів highest_level [1..0] буде присвоєно число В "11". Вхід middle має більш низький пріоритет, а вхід low - нижчий.

Оператор CASE дозволяє оцінити значення однорозрядною змінної (одно- розрядного виведення), групи змінних (групи висновків) і за результатами оцінки вибрати той чи інший оператор

для виконання.

CASE <rgroup> **IS**
WHEN <choices> => <statements>
{ **WHEN** <choices> => <statements>}
[**WHEN OTHERS** => <statements>]
END CASE;

Приклад:

```
SUBDESIGN case
(
   high, middle, low                :INPUT;
   highest_level[1..0]              :OUTPUT;
)
BEGIN
CASE (high, middle, low) IS
      WHEN B"1xx" => highest_level[1..0]=3;
      WHEN B"01x" => highest_level[1..0]=2;
      WHEN B"001" => highest_level[1..0]=1;
      WHEN OTHERS => highest_level[1..0]=0;
   END CASE;
   END;
```

Порядок виконання роботи:

1. Реалізувати проекти prior, if_then, case; виконати їх компіляцію і тимчасове моделювання

2. Вирішити індивідуальне завдання за допомогою оператора CASE або TABLE.

3. Вирішити завдання за допомогою оператора IF THEN: дана система рівнянь, яку реалізує цифровий пристрій. При формуванні сигналу z1 включити виконавчий механізм «A»; при відсутності сигналу z1 і наявності сигналу z2 включити виконавчий механізм «B»; при відсутності сигналів z1 і z2 і наявності сигналу z3 включити виконавчий механізм «C»; інакше включити виконавчий механізм «D».

Індивідуальні завдання.

1.а) y = [1, 3, 6, 8, 11, 15, 22, 23, 29, 30]; б) $\begin{cases} z1 = c \wedge a \wedge b; \\ z2 = a \wedge b; \\ z3 = a + b. \end{cases}$

2.а) y = [2, 5, 8, 12, 15, 23, 26, 27, 28, 29]; б) $\begin{cases} z1 = c; \\ z2 = a \wedge b; \\ z3 = \overline{c + a} + b. \end{cases}$

3.а) y = [3, 8, 9, 10, 14, 18, 21, 25, 27, 28]; б) $\begin{cases} z1 = \bar{c} \wedge \bar{a}; \\ z2 = a \wedge b; \\ z3 = b + c. \end{cases}$

4.а) y = [1, 3, 7, 15, 16, 18, 27, 28, 29, 31]; б) $\begin{cases} z1 = \bar{c} \wedge \bar{a} \wedge b; \\ z2 = a \wedge b; \\ z3 = a + c. \end{cases}$

5.а) y = [3, 6, 8, 11, 15, 22, 23, 29, 30, 31]; б) $\begin{cases} z1 = c \wedge \bar{a} \wedge b; \\ z2 = a \wedge b; \\ z3 = a + b. \end{cases}$

6.а) y = [6, 8, 11, 15, 17, 22, 23, 25, 26, 27]; б) $\begin{cases} z1 = c \wedge a \wedge \bar{b}; \\ z2 = a \wedge b; \\ z3 = a + b. \end{cases}$

7.а) y = [5, 7, 8, 12, 16, 23, 25, 27, 28, 29]; б) $\begin{cases} z1 = c; \\ z2 = a \wedge b; \\ z3 = \overline{c + a} + b. \end{cases}$

8.а) y = [4, 8, 9, 12, 14, 18, 22, 25, 27, 30]; б) $\begin{cases} z1 = c \wedge a; \\ z2 = a \wedge b; \\ z3 = b + c. \end{cases}$

9.а) y = [1, 3, 7, 11, 12, 18, 22, 28, 29, 31]; б) $\begin{cases} z1 = c \wedge \bar{a} \wedge b; \\ z2 = a \wedge b; \\ z3 = a + c. \end{cases}$

10.а) y = [9, 11, 12, 13, 15, 22, 23, 29, 30, 31]; б) $\begin{cases} z1 = b; \\ z2 = a \wedge c; \\ z3 = \overline{a + b}. \end{cases}$

Практичне заняття № 3
"Типові комбінаційні схеми"

Мета роботи: отримання практичних навичок у складанні програм реалізації типових комбінаційних схем.

Конструкції мови AHDL дозволяють ефективно описувати різноманітні цифрові пристрої: шифратори і дешифратори, мультиплексори і демультиплексори і т. Д.

Нижче представлений приклад опису трьохрозрядного шифратора, що забезпечує перетворення восьмирозрядного унітарного коду в двійковий трьохрозрядний код:

```
SUBDESIGN cd
(
   in[8..1]                    :INPUT;
   Binary_Code[3..1]           :OUTPUT;
)
BEGIN
     TABLE
      in[]          => Binary_Code[];
     b"00000001" => 0;
     b"00000010" => 1;
     b"00000100" => 2;
     b"00001000" => 3;
     b"00010000" => 4;
     b"00100000" => 5;
     b"01000000" => 6;
     b"10000000" => 8;
     END TABLE;
END;
```

В даному прикладі значення логічних функцій Binary_Code [3..1] задано не на всіх наборах аргументів. За замовчуванням, на всіх незаданих в таблиці істинності наборах аргументів групі виходів Binary_Code [3..1] буде присвоєно значення В "000".

Мультиплексор - безконтактний комутатор, що забезпечує передачу сигналу з одного з його входів на вихід пристрою, при чому вибір входу здійснюється кодом (адресою) входу. Опис мультиплексора розглянемо на прикладі:

```
SUBDESIGN mp
(
   in[1..4], m[1..2]                  :INPUT;
   out                                :OUTPUT;
)
BEGIN
      CASE x[] IS
       WHEN B“11” => in1 = out;
       WHEN B“10” => in2 = out;
       WHEN B“01” => in3 = out;
       WHEN B“00” => in4 = out;
      END CASE;
END
```

Порядок виконання роботи:

1. Реалізувати проекти cd і mp; виконати їх компіляцію і тимчасове моделювання

2. Реалізувати комбінаційну схему відповідно до індивідуального завданням, виконати її компіляцію і тимчасове моделювання.

Індивідуальні завдання.

1.Повне трьохрозрядний дешифратор.

2.Демультіплексор з шістьма інформаційними виходами.

3. Повний чотирьохрозрядний шифратор.

4. Мультиплексор з сімома інформаційними входами.

5.Четирехразрядний дешифратор, що виконує перетворення чотирирозрядний двійкового коду в шестнадцатізначний унітарний код.

6. Демультиплексор з вісьмома інформаційними виходами.

7.Трехразрядний дешифратор, що виконує перетворення трехразрядного двійкового коду в восьмизарядний унітарний код.

8. Демультиплексор з чотирма інформаційними виходами.

9.Четирехразрядний шифратор, з дванадцятьма входами.

10. Мультиплексор з вісьмома інформаційними входами.

Практичне заняття № 4
"Примітиви"

Мета роботи: отримання практичних навичок у складанні програм з використанням примітивів.

Для використання в текстовому описі модуля примітиву необхідно звернеться до вбудованого в пакет функціональному опису даного примітиву, і зіставити його з висновками числа, константи, змінні або висновки модуля.

Приклад:

```
SUBDESIGN mktreg
(
X1[7..0], ENA, CLK              :INPUT;
 Y[7..0]                        :OUTPUT;
)
VARIABLE
FF[7..0]:DFFE;
BEGIN
      FF[].CLK=CLK;
      FF[].ENA=ENA;
      FF[].D=X1[];
      Y[]=FF[].q;
END;
```

В даному прикладі використаний примітив тригера DFFE - D-тригер зі входом Ena (Enable).

Нижче представлений файл bur_reg.tdf, що описує паралельний восьмирозрядний регістр. Для його опису використаний примітив D-тригера з додатковим входом Ena (Enable). Перший рядок логічної секції в даному описі забезпечує підключення входів синхронізації D-тригерів до сигналу clk. Другий рядок забезпечує підключення сигналу завантаження регістра load до входів ena D-тригерів. Третій рядок забезпечує підключення інформаційних входів d [7..0] регістра до входів d D-тригерів. Четвертий рядок забезпечує підключення інформаційних виходів q D-тригерів до виходів q [7..0] регістра.

```
SUBDESIGN bur_reg
(
   clk, load, d[7..0]                  :INPUT;
   q[7..0]                             :OUTPUT;
)
VARIABLE
ff[7..0]        :DFFE;
BEGIN
      ff[].clk=clk;
      ff[].ena=load;
      ff[].d=d[];
      q[]=ff[].q;

END;
```

Порядок виконання роботи:

1. Реалізувати проекти mktreg і bur_reg; виконати їх компіляцію і тимчасове моделювання.

2. Реалізувати регістр відповідно до індивідуального завданням, виконати його компіляцію і тимчасове моделювання.

Індивідуальні завдання:

1. Паралельний чотирьохрозрядний регістр з входом завантаження і асинхронним скиданням.

2. Послідовний чотирьохрозрядний регістр зсуву (зрушення вправо), з синхронною установкою в «1».

3. Послідовний восьмизарядний регістр зсуву (зрушення вліво), з синхронним скиданням.

4. Паралельний шестнадцятиразрядний регістр з входом завантаження і синхронним скиданням.

5. Послідовний восьмизарядний регістр зсуву (зрушення вліво), з входом завантаження і синхронним скиданням.

6. Послідовний чотирьохрозрядний регістр зсуву (зрушення вправо), з входом установки регістра в стан «1011».

7. Паралельний чотирьохрозрядний регістр з асинхронним скиданням.

8. Послідовний чотирьохрозрядний регістр зсуву (зрушення вліво), з синхронною установкою в «1».

9. Послідовний восьмизарядний регістр зсуву (зрушення вліво), з синхронним скиданням.

10. Паралельний шестнадцятиразрядний регістр з входом завантаження і асинхронним скиданням.

Практичне заняття № 5
"Лічильники"

Мета роботи: отримання практичних навичок у складанні програм опису лічильників.

Лічильник - цифровий пристрій для рахунку сигналів (імпульсів), що надійшли на його вхід. Лічильники застосовуються для формування послідовності адресованих команд (програмний лічильник), для рахунку кількості виконуваних операцій, для рахунку подій, що відбуваються на керовані об'єкти. Файл ahdcnt.tdf, представлений нижче, описує 16-ти бітний завантажується лічильник з можливістю обнулення.

Приклад:

```
SUBDESIGN ahdcnt
(
 clk, load, ena, clr, d[15..0]          :INPUT;
 q[15..0]                               :OUTPUT;
)
VARIABLE
count[15..0]:DFF;
BEGIN
      count[].clk=clk ;
      count[].clrn=!clr;
      IF load THEN
            count[].d=d[];
       ELSIF ena THEN
            count[].d=count[].q+1;
       ELSE
            count[].d=count[].q;
     END IF;
       q[]=count[];
 END;
```

В даному прикладі 16 D-тригерів оголошені в секції VARIABLE. Конструкція IF THEN забезпечує вибір режиму функціонування лічильника: режим завантаження, якщо на вхід load поданий сигнал логічної «1», режим рахунку, якщо на вхід ena поданий сигнал логічної «1», або режим очікування.

Порядок виконання роботи:

1. Реалізувати проект ahdcnt; виконати його компіляцію і тимчасове моделювання.

2. Реалізувати лічильник відповідно до індивідуального завданням, виконати його компіляцію і тимчасове моделювання.

Індивідуальні завдання:

1. 32-х бітний завантажується лічильник з можливістю обнулення.
2. 8-ми бітний лічильник з входом установки в стан 38h.
3. 16-ти бітний лічильник зворотного відліку з можливістю обнулення.
4. 4-х бітний завантажується лічильник зворотного відліку з можливістю обнулення.
5. 16-ти бітний лічильник з входом установки в стан 36.
6. 32-ти бітний завантажується лічильник з можливістю обнулення.
7. 4-х бітний лічильник з входом установки в стан b "1001".
8. 8-ми бітний завантажується лічильник з можливістю обнулення.
9. 64-х бітний лічильник з входом установки в стан 166h.
10. 64-х бітний завантажується лічильник зворотного відліку з можливістю обнулення.

Практичне заняття № 6
"Реалізація проектів з ієрархічною структурою"

Мета роботи: отримання практичних навичок у складанні програм реалізації проектів з ієрархічною структурою.

Об'єднання різних цифрових пристроїв з використанням принципів ієрархічного опису проектів дозволяє реалізувати на основі ПЛІС як стандартні, так і нетрадиційні архітектури керуючих автоматів різного призначення. Як приклад розглянемо проект, що описує схему восьмирозрядного модуля багаторазової перевірки правильності прийняття управлінських рішень (МПУР). Дана схема дозволяє виключити помилкове формування керуючих рішень при наявності короткочасних спотворень вхідних або внутрішніх сигналів ПЛІС, обумовлених впливом електромагнітних завад, дрібязгом контактів, «змаганнями» сигналів і т. Д. Структурна схема модуля МПУР представлена на рис. 6.1 і містить наступні функціональні вузли:

- два регістра пам'яті reg, які містять інформаційні входи X1 [7..0], інформаційні виходи Y [7..0], вхід сигналу синхронізації clk і вхід синхронної завантаження LOAD;

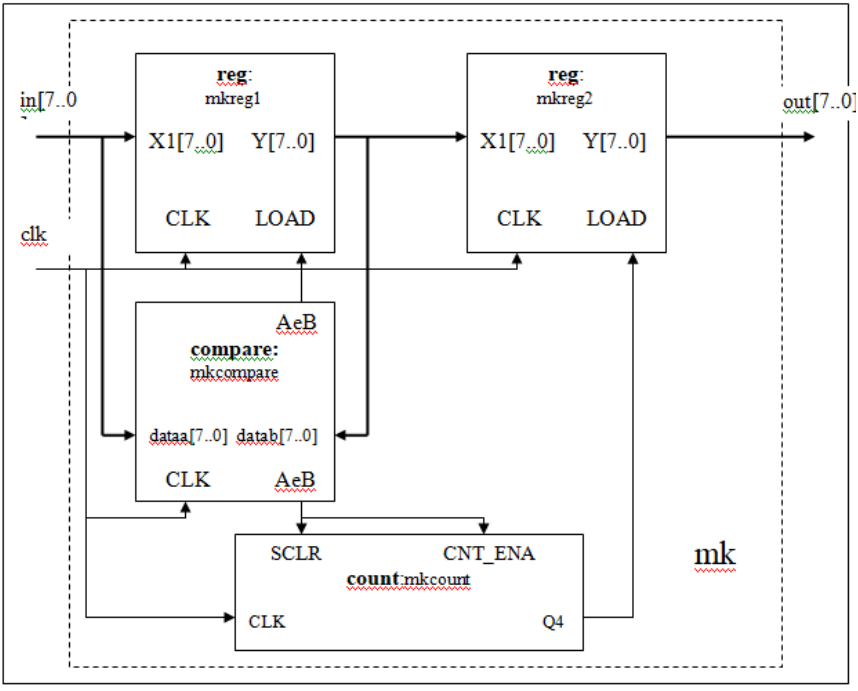


Рисунок 6.1

- схему порівняння compare, яка містить інформаційні входи dataa [7..0] і datab [7..0], вхід сигналу синхронізації CLK і вихід AeB, що приймає значення логічної 1 при dataa [7..0] = = datab [7 ..0];

- лічильник count, який містить вхід сигналу синхронізації CLK, вхід синхронного скидання SCLR, вхід дозволу рахунку CNT_ENA і вихід Q4.

Файл верхнього рівня mk містить зазначені вище функціональні вузли і описує взаємодії між ними.

У разі зміни одного або декількох сигналів групи in [7..0] на виході AeB схеми порівняння compare з'являється сигнал логічного 0. В результаті цього здійснюється скидання лічильника count (по входу SCLR) і завантаження першого регістра reg (по входу LOAD). Сигнал AeB знову приймає значення логічної 1, яка надходить на входи SCLR і CNT_ENA лічильника count. Дорахував до 16, лічильник видає на вихід Q4 сигнал логічної 1, що забезпечує можливість завантаження другого регістра reg і формування на виходах out [7..0] відповідної інформації. У тому випадку, якщо інформація на входах in [7..0] зміниться раніше, ніж лічильник дорахував до 16, буде вироблено його обнуління і чергова завантаження першого регістра. Таким чином, схема забезпечує 16-ти кратний контроль правильності прийняття управлінських рішень. Період перевірки визначається частотою синхронізації.

Текстовий опис модуля МПУР виглядає наступним чином.

```
SUBDESIGN reg
(
 X1[7..0], LOAD, CLK        :INPUT;
 Y[7..0]                    :OUTPUT;
)
VARIABLE
   FF[7..0] : DFFE;
BEGIN
   FF[].CLK=CLK;
   FF[].ENA=LOAD;
   FF[].D=X1[];
   Y[]=FF[].q;
END;
```

```
SUBDESIGN compare
(
 dataa[7..0], datab[7..0], CLK        :INPUT;
 AeB                                  :OUTPUT;
)
VARIABLE
   tr : DFF;
BEGIN
   tr.CLK=CLK;
   IF dataa[] = = datab[]
   THEN tr.D=b"1";
   ELSE tr.D=b"0";
   END IF;
   AeB=tr.q;
END;

SUBDESIGN count
(
 cnt_en, SCLR, CLK        :INPUT;
 q[4..0]                  :OUTPUT;
)
VARIABLE
   FF[4..0] : DFFE;
BEGIN
   tr[].PRN=VCC;
   tr[].ENA= cnt_en;
   tr[].CLRN=sclr;
   tr[].clk=CLK;
   tr[].D=tr[].Q+1;
   q[]=tr[].Q;
END;

INCLUDE "reg";
INCLUDE "count";
INCLUDE "compare";
SUBDESIGN mk
(
in[7..0], clk       :INPUT;
 out[7..0]          :OUTPUT;
)
VARIABLE
        mkreg1:reg;
```

```
        mkreg2:reg;
        mkcount:count;
        mkcomp:compare;
BEGIN
mkreg1.X1[]=in[];
mkreg1.LOAD=!mkcomp.AeB;
mkreg1.CLK=clk;
mkcount.CLK=clk;
mkcount.cnt_en=mkcomp.AeB;
mkcount.SCLR=mkcomp.AeB;
mkcomp.dataa[]=in[];
mkcompdatab[]=mkreg1.Y[];
mkcomp.CLK=clk;
out[]=mkreg2.Y[];
END;
```

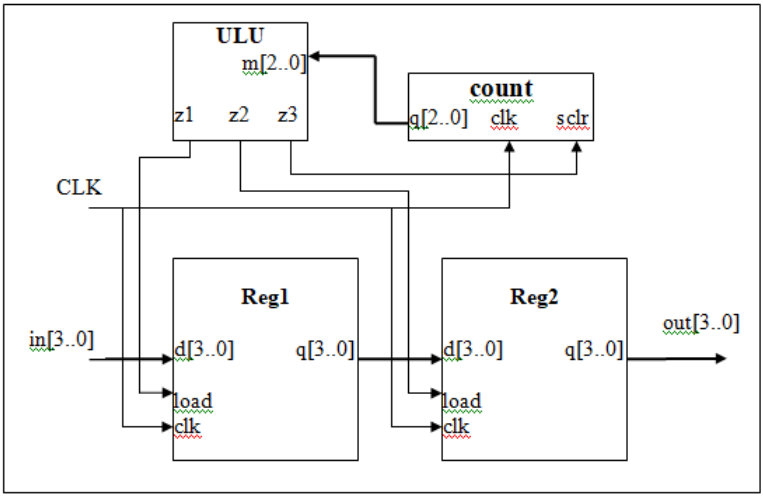
Порядок виконання роботи:

1. Реалізувати проект mk; виконати його компіляцію і тимчасове моделювання.

2. Реалізувати пристрій відповідно до індивідуального завданням, виконати його компіляцію і тимчасове моделювання.

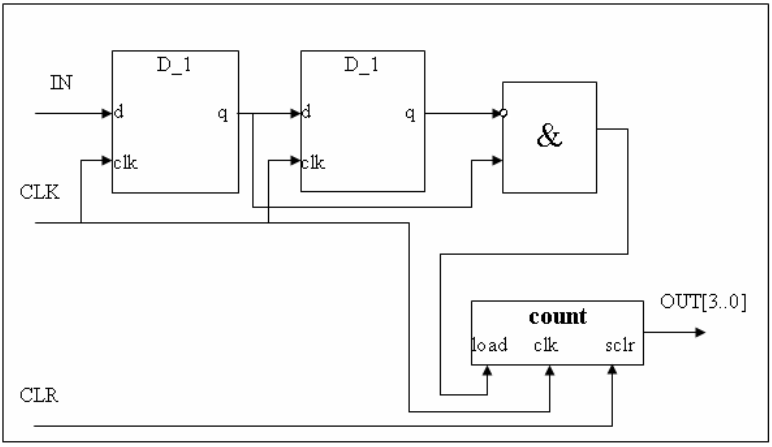
Індивідуальні завдання:

1.



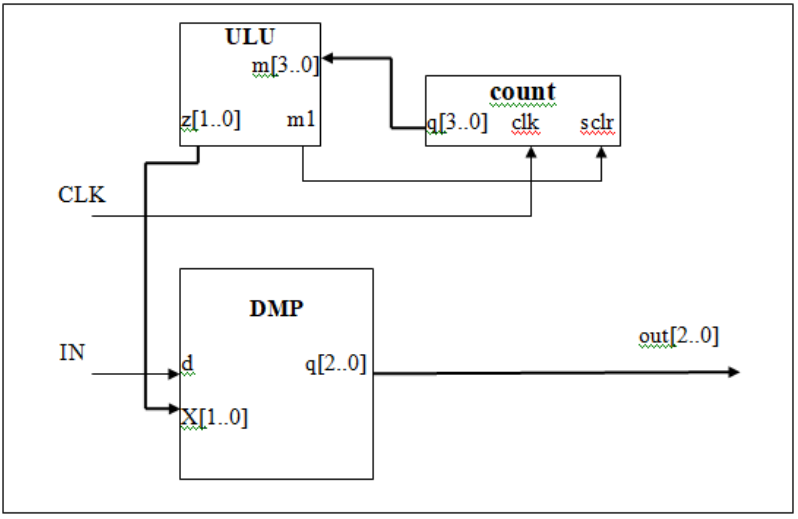
При появі на виході лічильника сигналу b "011" завантажити інформацію з входів in [3..0] в регістр Reg1. У момент коли на виході лічильника з'явиться комбінація b "101" завантажити дані в регістр Reg2; скинути лічильник.

2.



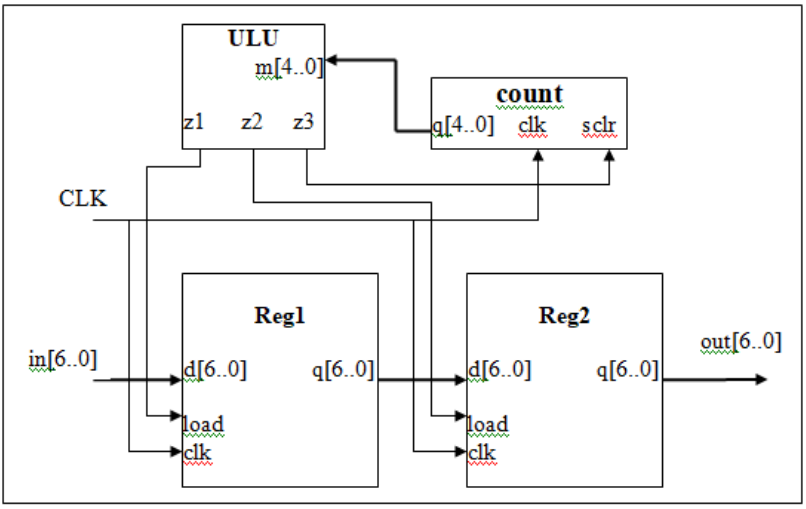
Реалізувати пристрій, що складається з схеми виділення переднього фронту і лічильника.

3.



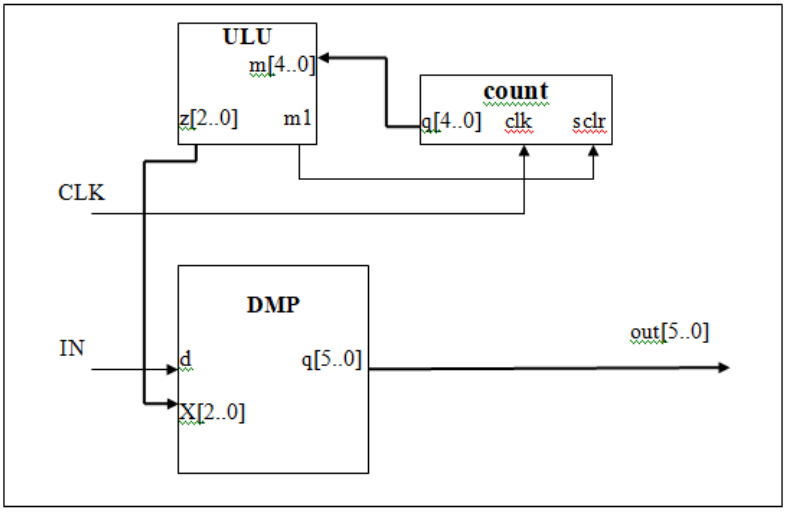
Реалізувати схему управління Демультиплексори. Змінювати керуючі сигнали демультиплексор через кожні два такту синхронізації. Скинути лічильник після появи на виході z [1..0] комбінації "11".

4.



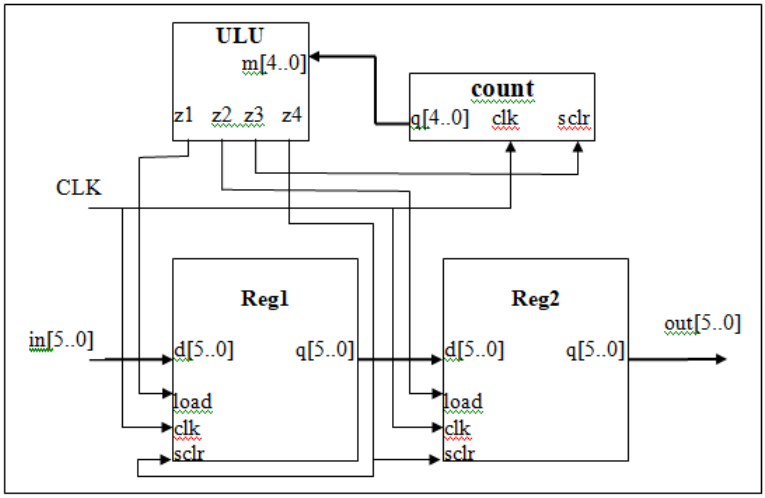
При появі на виході лічильника сигналу 9h завантажити інформацію з входів in [6..0] в регістр Reg1. У момент, коли на виході лічильника з'явиться комбінація 12h завантажити дані в регістр Reg2; скинути лічильник в момент, коли на його виході з'явиться комбінація 16h.

5.



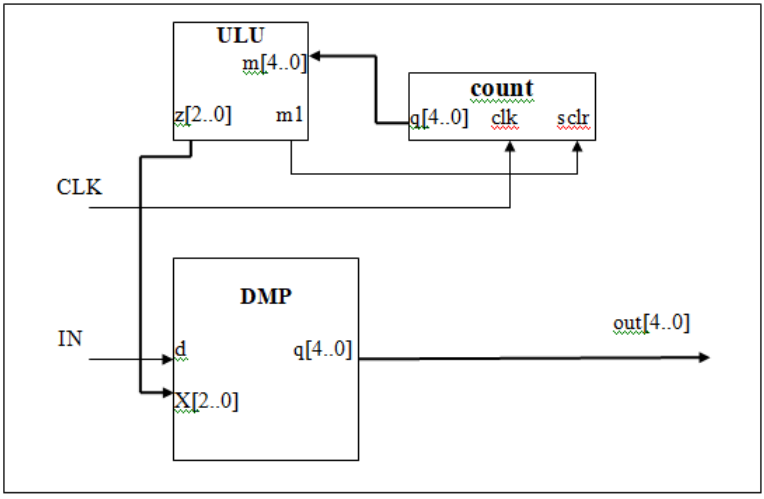
Реалізувати схему управління Демультиплексори. Змінювати керуючі сигнали демультиплексор через кожні два такту синхронізації. Скинути лічильник після появи на виході z [2..0] комбінації "111".

6.



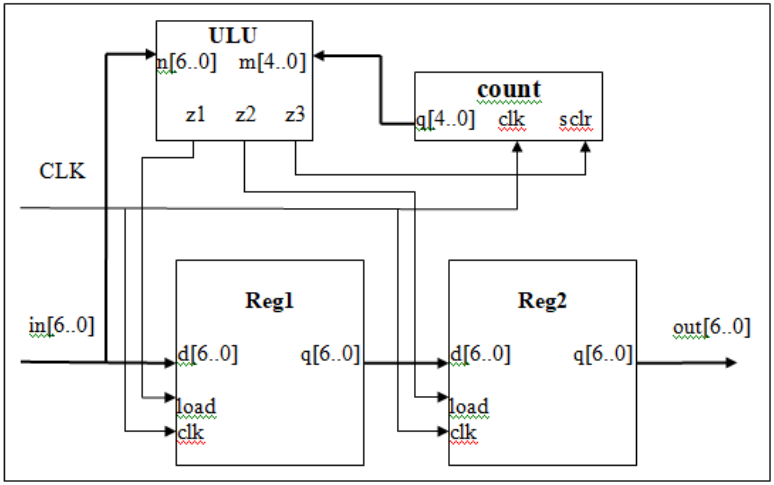
При появі на виході лічильника 10 завантажити інформацію з входів in [5..0] в регістр Reg1. У момент коли на виході лічильника з'явиться 20 завантажити дані в регістр Reg2, скинути регістр Reg1; коли на виході лічильника буде 30 скинути лічильник і скинути регістр Reg2.

7.



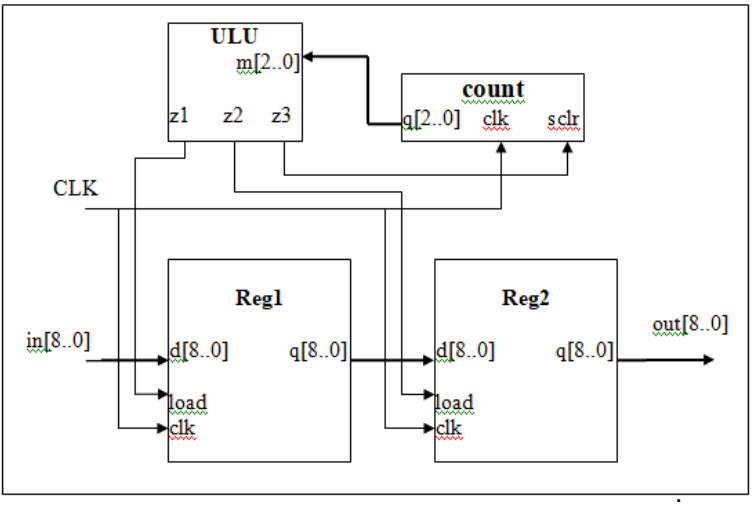
Реалізувати схему управління Демультиплексори. Змінювати керуючі сигнали демультиплексор через кожні два такту синхронізації. Скинути лічильник після появи на виході z [2..0] комбінації "001" (лічильник зворотного відліку).

8.



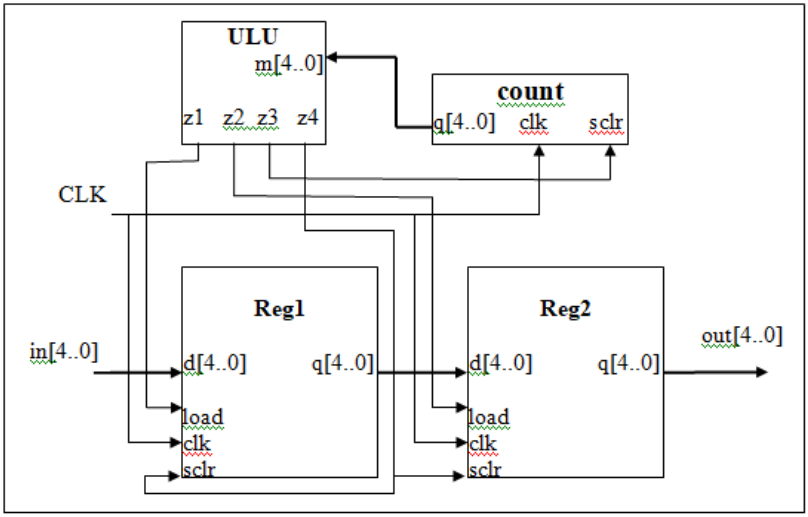
При появі на виході лічильника сигналу b "00010" завантажити інформацію з входів in [6..0] в регістр Reg1. У момент, коли на виході лічильника з'явиться комбінація b "00100" завантажити дані в регістр Reg2; скинути лічильник в момент, скинути лічильник, якщо на вході in [6..0] з'явиться комбінація b "1111111" або коли на виході лічильника з'явиться комбінація b "01000".

9.



При появі на виході лічильника сигналу b "010" завантажити інформацію з входів in [8..0] в регістр Reg1. У момент коли на виході лічильника з'явиться комбінація b "110" завантажити дані в регістр Reg2; скинути лічильник.

10.



При появі на виході лічильника 6 завантажити інформацію з входів in [4..0] в регістр Reg1. У момент коли на виході лічильника з'явиться 18 завантажити дані в регістр Reg2, скинути регістр Reg1; коли на виході лічильника буде 28 скинути лічильник і скинути регістр Reg2.

Практичне заняття № 7
"Реалізація автомата Мура"

Мета роботи: отримання практичних навичок в реалізації кінцевих автоматів.

Реалізацію автомата Мура розглянемо на прикладі схеми включення лампи з кнопкою без фіксації. Нехай лампа Z світиться після непарного числа натискання кнопки X і не світиться після парного числа натискань.

Як бачите, один і той же дію (натискання кнопки X) в різних випадках призводить до різних наслідків: в одному випадку лампа включається, а в іншому вимикається. Для того щоб реалізувати подібний алгоритм, необхідна наявність пристрою, який, крім реалізації логічних рівнянь, могло б запам'ятовувати передісторію вхідних впливів (в даному випадку - це натискання / відпускання кнопки).

Структурна схема автомата Мура показана на рис. 7.1. Даний автомат складається з вхідних комбінаційної схеми CL_{φ}, елемента пам'яті у вигляді регістра RG і вихідний комбінаційної схеми CL_{ψ}.

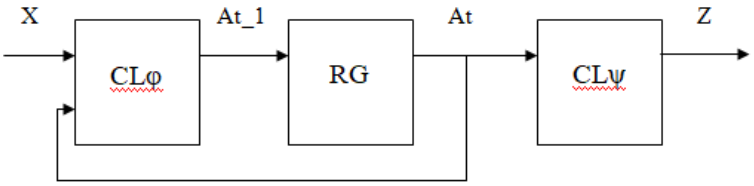


Рисунок 7.1

Для завдання кінцевих автоматів широко використовуються графи переходів, таблиці переходів і системи логічних рівнянь. Основними елементами графа є стану автомата (їх називають вершинами графа), які позначаються кружками з номером стану, і подіями (їх називають переходами), які позначаються стрілками. На рис 7.2. показаний граф переходів для автомата, що описує алгоритм з лампою.

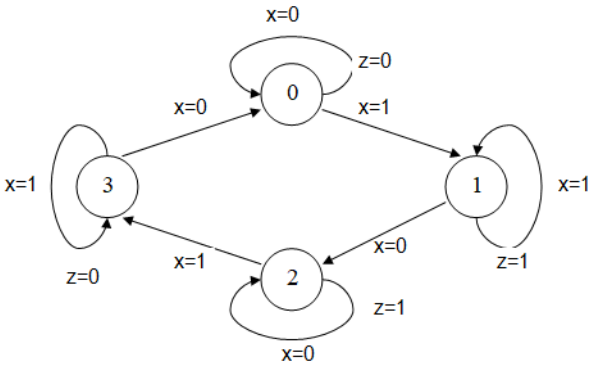


Рисунок 7. 2.

```
SUBDESIGN vhodn_preobr
(
X_1,  At[1..2]          :INPUT;
At_1[1..2]              :OUTPUT;
)
BEGIN
CASE  (At[], X)  IS
WHEN (b"00", b"0") => At_1[]=b"00";
WHEN (b"00", b"1") => At_1[]=b"01";
WHEN (b"01", b"1") => At_1[]=b"01";
WHEN (b"01", b"0") => At_1[]=b"10";
WHEN (b"10", b"0") => At_1[]=b"10";
WHEN (b"10", b"1") => At_1[]=b"11";
WHEN (b"11", b"1") => At_1[]=b"11";
WHEN (b"11", b"0") => At_1[]=b"00";
END CASE;
END;

SUBDESIGN reg
(
clk,  At_1[1..2]        :INPUT;
At[1..2]                :OUTPUT;
)
VARIABLE
FF[2..1]:DFF;
BEGIN
FF[].clk=clk;
FF[].D= At_1[];
At[]=FF[].q;
END;
```

```
SUBDESIGN vyhodn_preobr
(
At[1..2]        :INPUT;
Z               :OUTPUT;
)
BEGIN
TABLE
At[] => Z;
b”00” => 0;
b”01” => 1;
b”10” => 1;
b”11” => 0;
END TABLE;
END;

INCLUDE “vhodn_preobr”;
INCLUDE “reg”;
INCLUDE “vyhodn_preobr”;
SUBDESIGN am
(
X_1, clk        :INPUT;
Z               :OUTPUT;
)
VARIABLE
amvhodn_preobr: vhodn_preobr;
amreg: reg;
amvyhodn_preobr: vyhodn_preobr;
BEGIN
amvhodn_preobr.X_1=X_1;
amreg. At_1[]=amvhodn_preobr. At_1[];
amvhodn_preobr.At[]=amreg.At[];
amreg.clk=clk;
amvyhodn_preobr.At[]=amreg.At[];
Z= amvyhodn_preobr.Z;
END;
```

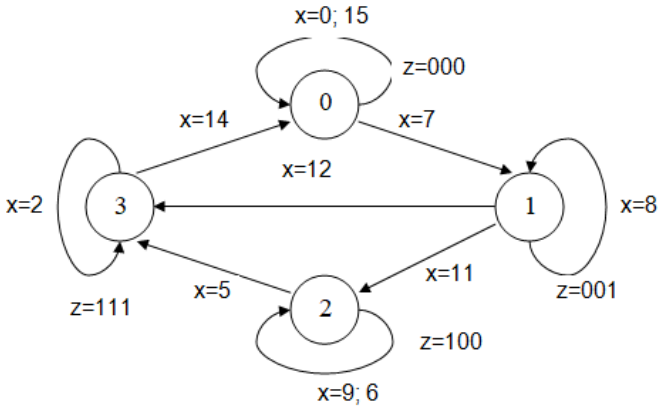
Порядок виконання роботи:

1. Реалізувати проект am; виконати його компіляцію і тимчасове моделювання.

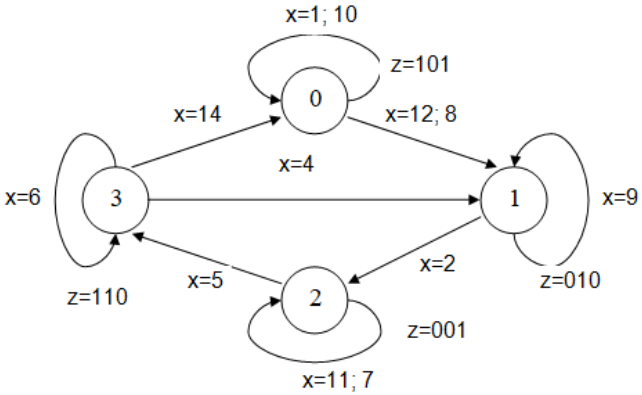
2. Вирішити індивідуальне завдання.

Індивідуальні завдання:

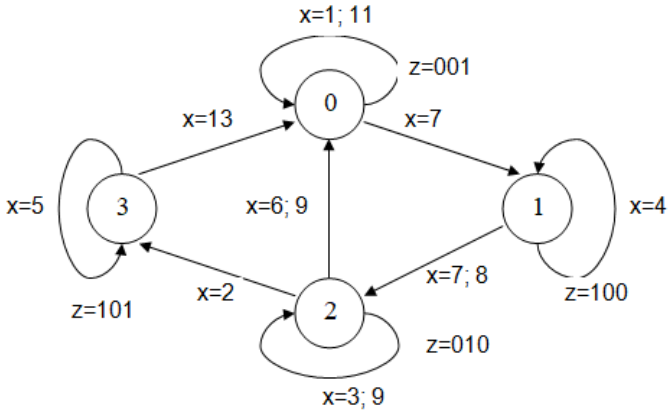
1.



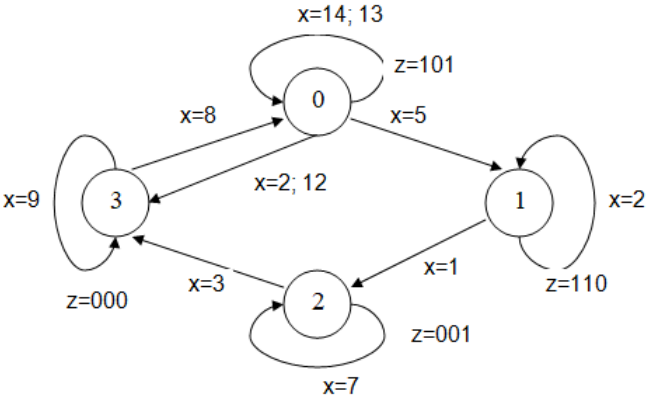
2.



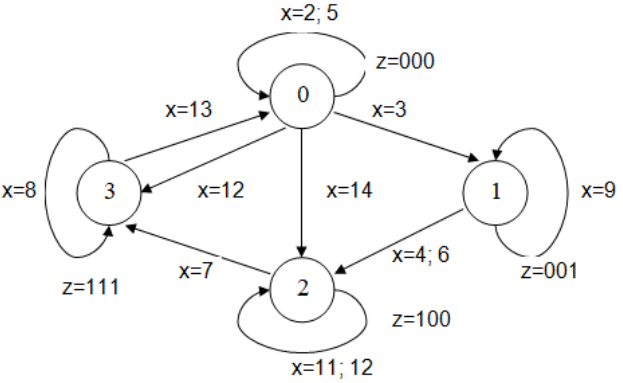
3.



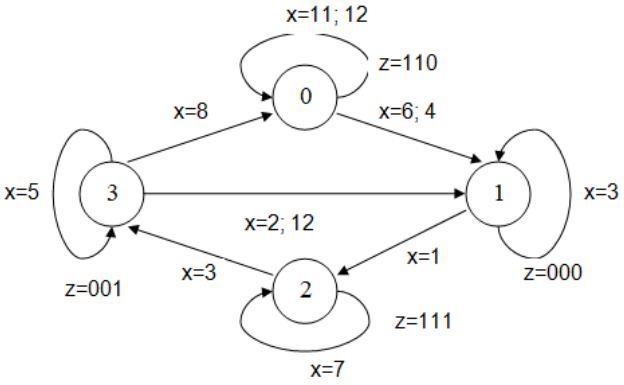
4.


x=14; 13
z=101
0
x=8
x=5
x=9
3
x=2; 12
1
x=2
x=1
x=3
z=110
z=000
2
z=001
x=7

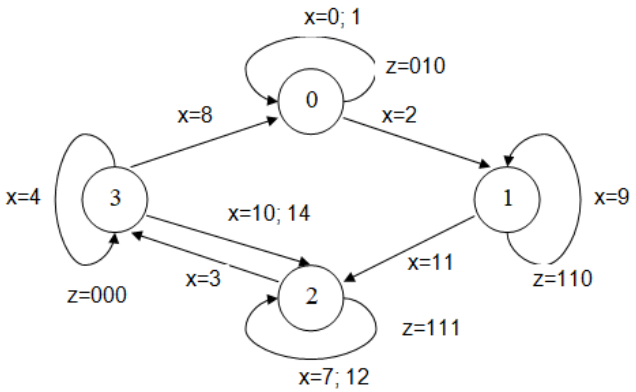
5.


x=2; 5
z=000
0
x=13
x=3
x=8
3
x=12
x=14
1
x=9
x=4; 6
x=7
z=111
z=001
2
z=100
x=11; 12

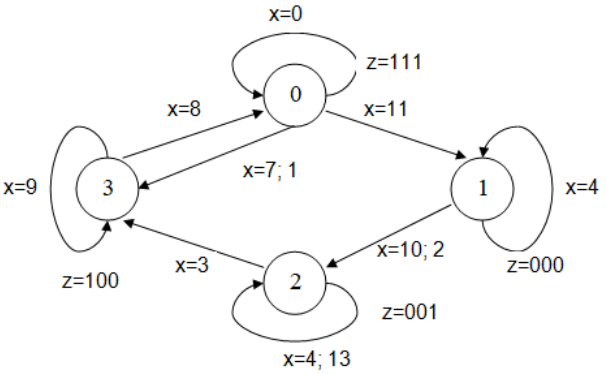
6.


x=11; 12
z=110
0
x=8
x=6; 4
x=5
3
1
x=3
x=2; 12
x=1
x=3
z=001
z=000
2
z=111
x=7

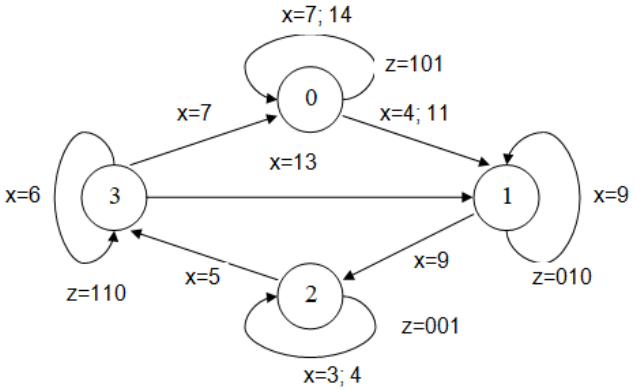
7.


x=0; 1
z=010
0
x=8
x=2
x=4
3
1
x=9
x=10; 14
x=11
x=3
z=000
2
z=110
z=111
x=7; 12

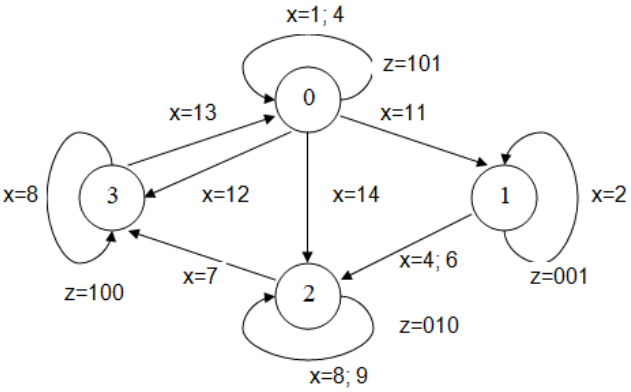
8.


x=0
z=111
0
x=8
x=11
x=9
3
x=7; 1
1
x=4
x=10; 2
x=3
z=100
2
z=000
z=001
x=4; 13

9.


x=7; 14
z=101
0
x=7
x=4; 11
x=13
x=6
3
1
x=9
x=9
x=5
z=110
2
z=010
z=001
x=3; 4

10.


x=1; 4
z=101
0
x=13
x=11
x=8
3
x=12
x=14
1
x=2
x=7
x=4; 6
z=100
z=001
2
z=010
x=8; 9

Практичне заняття № 8
"Реалізація автомата Мілі"

Мета роботи: отримання практичних навичок в реалізації кінцевих автоматів.

Структурна схема автомата Мілі показана на рис. 8.1. Даний автомат складається з вхідних комбінаційної схеми CL_{φ}, елемента пам'яті у вигляді регістра RG і вихідний комбінаційної схеми CL_{ψ}. На відміну від автомата Мура (див. П. Р. №7), вихідна комбінаційна схема автомата Мілі (CL_{ψ}) має безпосередній зв'язок з входами автомата.

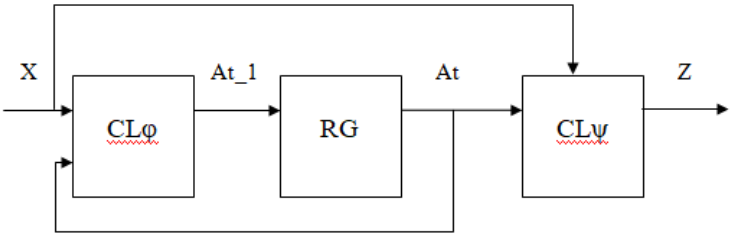


Рисунок 8. 1

На рис 8.2. показаний граф переходів для автомата Мілі.

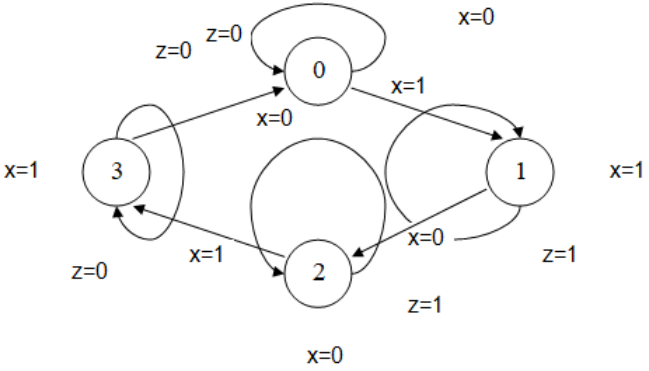


Рисунок 8. 2

```
SUBDESIGN vhodn_preobr
(
X_1,  At[1..2]          :INPUT;
At_1[1..2]              :OUTPUT;
)
BEGIN
CASE  (At[], X)  IS
WHEN (b"00", b"0") => At_1[]=b"00";
WHEN (b"00", b"1") => At_1[]=b"01";
WHEN (b"01", b"1") => At_1[]=b"01";
WHEN (b"01", b"0") => At_1[]=b"10";
WHEN (b"10", b"0") => At_1[]=b"10";
WHEN (b"10", b"1") => At_1[]=b"11";
WHEN (b"11", b"1") => At_1[]=b"11";
WHEN (b"11", b"0") => At_1[]=b"00";
END CASE;
END;

SUBDESIGN reg
(
clk,  At_1[1..2]        :INPUT;
At[1..2]                :OUTPUT;
)
VARIABLE
FF[2..1]:DFF;
BEGIN
FF[].clk=clk;
FF[].D= At_1[];
At[]=FF[].q;
END;
SUBDESIGN vyhodn_preobr
(
At[1..2]        :INPUT;
Z               :OUTPUT;
)
BEGIN
TABLE
At[] => Z;
b"00" => 0;
b"01" => 1;
b"10" => 1;
b"11" => 0;
```

```
END TABLE;
END;

INCLUDE "vhodn_preobr";
INCLUDE "reg";
INCLUDE "vyhodn_preobr";
SUBDESIGN am
(
X_1, clk        :INPUT;
Z               :OUTPUT;
)
VARIABLE
amvhodn_preobr: vhodn_preobr;
amreg: reg;
amvyhodn_preobr: vyhodn_preobr;
BEGIN
amvhodn_preobr.X_1=X_1;
amreg. At_1[]=amvhodn_preobr. At_1[];
amvhodn_preobr.At[]=amreg.At[];
amreg.clk=clk;
amvyhodn_preobr.At[]=amreg.At[];
Z= amvyhodn_preobr.Z;
END;
```

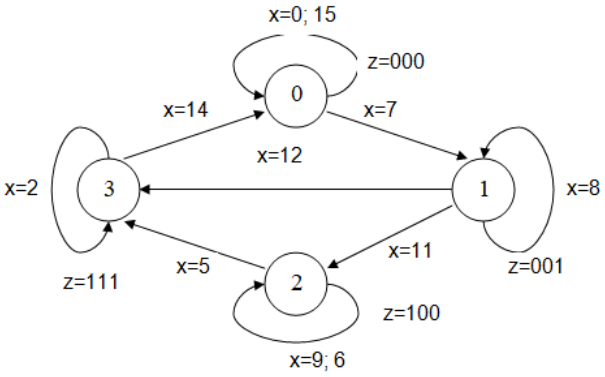
Порядок виконання роботи:

1. Реалізувати проект am; виконати його компіляцію і тимчасове моделювання.

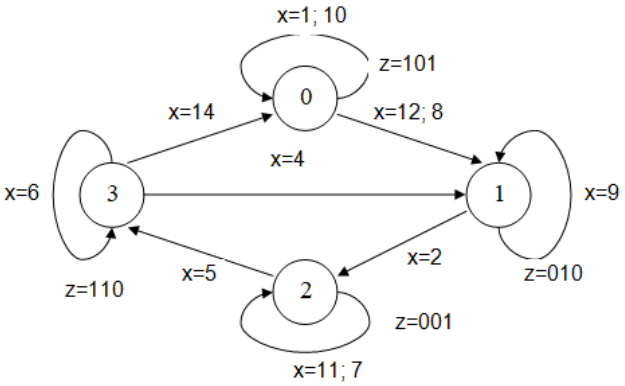
2. Вирішити індивідуальне завдання.

Індивідуальні завдання:

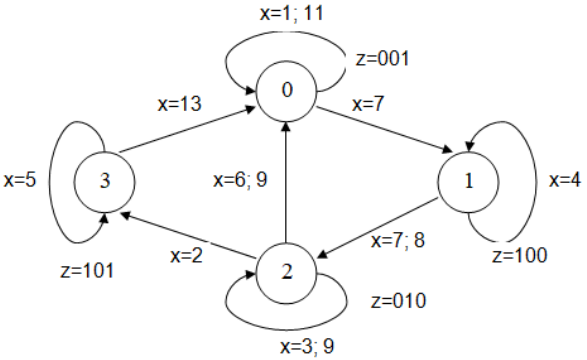
1.



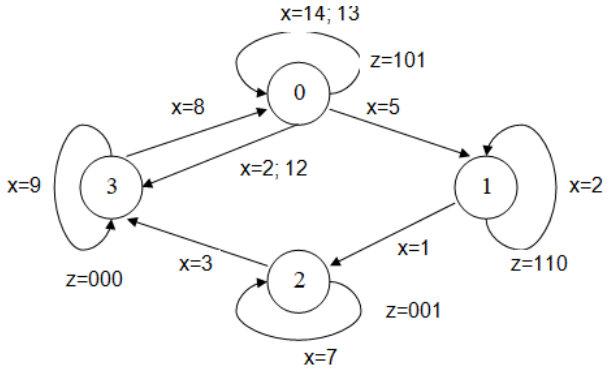
2.


x=1; 10
z=101
0
x=14
x=12; 8
x=4
x=6
3
1
x=9
x=2
x=5
z=010
z=110
2
z=001
x=11; 7

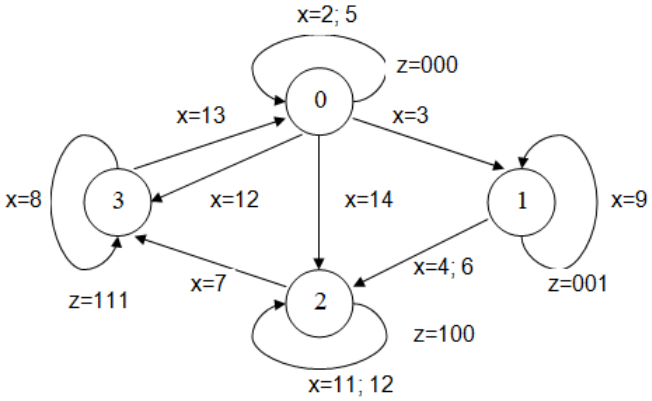
3.


x=1; 11
z=001
0
x=13
x=7
x=5
3
x=6; 9
1
x=4
x=7; 8
x=2
z=101
z=100
2
z=010
x=3; 9

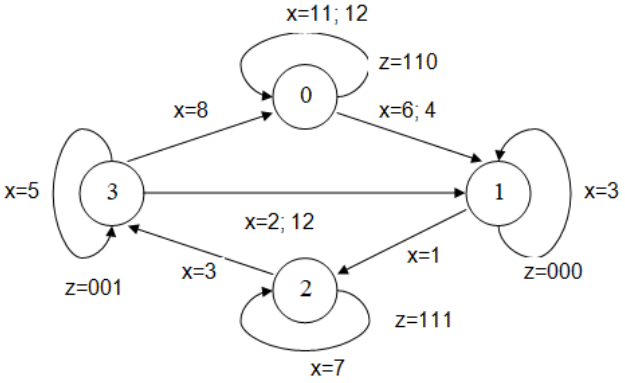
4.


x=14; 13
z=101
0
x=8
x=5
x=9
3
x=2; 12
1
x=2
x=1
x=3
z=000
z=110
2
z=001
x=7

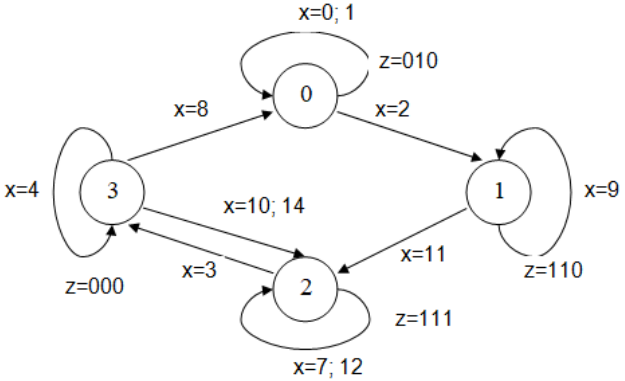
5.


x=2; 5
z=000
0
x=13
x=3
x=8
3
x=12
x=14
1
x=9
x=7
x=4; 6
z=111
2
z=001
z=100
x=11; 12

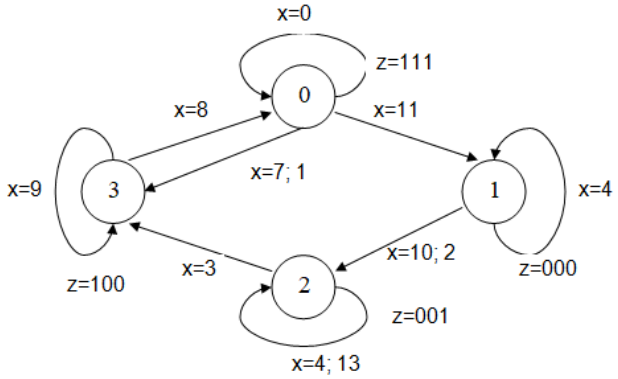
6.


x=11; 12
z=110
0
x=8
x=6; 4
x=5
3
1
x=3
x=2; 12
x=1
x=3
2
z=001
z=000
z=111
x=7

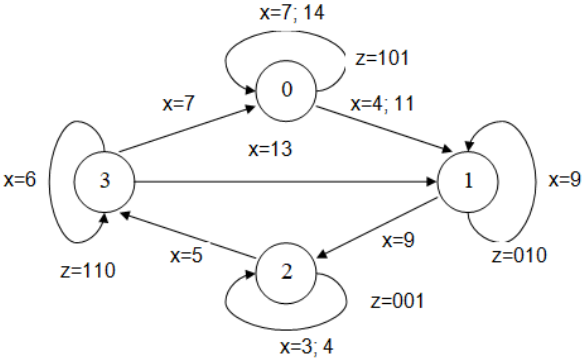
7.


x=0; 1
z=010
0
x=8
x=2
x=4
3
1
x=9
x=10; 14
x=11
x=3
z=110
z=000
2
z=111
x=7; 12

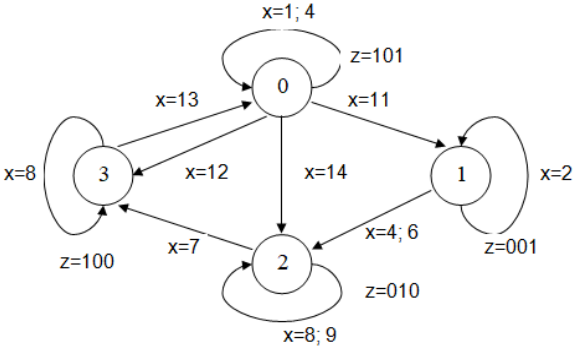
8.


x=0
z=111
0
x=8
x=11
x=9
3
x=7; 1
1
x=4
x=10; 2
z=000
x=3
z=100
2
z=001
x=4; 13

9.


x=7; 14
z=101
0
x=7
x=4; 11
x=13
x=6
3
1
x=9
x=9
x=5
z=010
z=110
2
z=001
x=3; 4

10.


x=1; 4
z=101
0
x=13
x=11
x=8
3
x=12
x=14
1
x=2
x=4; 6
x=7
z=001
z=100
2
z=010
x=8; 9

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фурман І.А. Організація та програмування мікроконтролерів / Фурман І.А., Краснобаєв В.А., Скороделов В.В., Рисований А.М: Підручник. – Харків: Еспада, 2005. – 248 с.

2. Теорія цифрових автоматів та формальних мов. Вступний курс : навч. посібник / Гавриленко С. Ю., Клименко А. М., Любченко Н.Ю. та ін. – Харків : НТУ "ХПІ", 2011. – 176 с.

3. Малиновський М. Л., Фурман І. О., Бовчалюк С. Я. / Проектування цифрових пристроїв ПЛІС: Підручник для ВНЗ. – Харків: Факт, 2006. – 164 с.

4. Методологія системного підходу: навч. посібник / [С. О. Кошман, С. О. Мороз, В. М. Курчанов та ін.]; під загальною редакцією С. О. Кошмана. – Харків: ХНТУСГ, 2016. – 125 с.

5. Малиновський М. Л. / Управління об'єктами критичного використання на основі ПЛІС: Монографія. – Харків: ФАКТ, 2008. – 224 с.

6. Фурман І. О., Малиновський М. Л., Джулганов В. Г. Та ін. / Мікроконтролерні засоби програмного керування / Під загальною редакцією І. О. Фурман: Підручник для студентів ВНЗ. – Харків: Факт, 2007 – 486 с.

7. Теорія цифрових автоматів: метод. вказівки до виконання практ. робіт для студентів другого (магістерського) рівня вищої освіти денної та заоч. форм навч. спец. 151 Автоматизація та комп’ютерно-інтегровані технології ; Харків. нац. техн. у-т сіл. госп-ва ім. П. Василенка ; уклад.: С. О. Тимчук, А. О. Панов. – Харків : [б. в.], 2020.– 56 с.

Навчальне видання

Методичні вказівки
до виконання практичних робіт з дисципліни
Теорія цифрових автоматів

Тимчук Сергій Олександрович
ПАНОВ Антон Олександрович

Формат 60×84/16. Гарнітура Times New Roman
Папір для цифрового друку. Друк ризографічний.
Ум. друк. арк. 2,67. Наклад 20 пр.
ДБТУ
61002, м. Харків, вул. Алчевських, 44