Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Архітектура комп'ютерів-3. Мікропроцесорні засоби

Домашня

Модульна контрольна робота

Виконала

студентка групи ІВ-71

Молчанова В.С.

Залікова книжка №7110

Перевірила

доцент

Ткаченко В.В.

Київ – 2020

**Технічне завдання на модульну роботу**

**Проектування мікропроцесорних систем**

**Студ.** Молчанової В.С. **гр.** ІВ-71

№ залікової книжки (7110)10, (11011 11000110)2

|  |  |
| --- | --- |
| Вибір елементної бази | 1816ВЕ51 |
| Організація шини |  |
| Вибір системи команд | Комплексна |
|  |  |
| Вектор переривання  (видає викладач bbb) | 106 |
| Кількість ЗУ | 111 |
| КПП. КПДП | Централізовані |
| Адреси для інтерфейсу зовніш. Пристрою (ВВ55) та ЗП. | ППА: 5020h, 5021h, 5022h, 5023h  ЗП3: 774 h |
| Зовнішня пам'ять програм | 6 сторінок по 16 Кб |
| Зовнішня пам'ять даних | 16 сторінок по 256 б |
| Функціонально структурна схема | МПС |
| Функціональна схем | Контролер пріоритетного переривання (піраміда) |
|  |  |
|  |  |
|  |  |

Завдання видав Ткаченко В.В.

Завдання прийняв Молчанова В.С.

Зміст

[Частина 1 4](#_Toc40438863)

[1. Опис мікропроцесорної системи 4](#_Toc40438864)

[2. Карта розподілу пам’яті даних 6](#_Toc40438865)

[3. Опис піраміди КПП 7](#_Toc40438866)

[Частина 2 8](#_Toc40438867)

[1. Лабораторна робота №1 (знайомство із середовищем моделювання ModelSim) 8](#_Toc40438868)

[2. Лабораторна робота №2 (дослідження комбінаційних пристроїв) 9](#_Toc40438870)

[3. Лабораторна робота №3 (комбінаційні пристрої) 13](#_Toc40438872)

[Частина 3 23](#_Toc40438875)

[1. Структурна схема мікропроцесорної системи 23](#_Toc40438876)

[2. Функціональна схема контролера пріоритетного переривання 24](#_Toc40438877)

[Висновки 25](#_Toc40438878)

# Частина 1

## 1. Опис мікропроцесорної системи

На рисунку 1 зображена мікропроцесорна система на основі мікроконтролера МК51, який складається з:

1. Резидентної пам’яті програм (РПП) об’ємом *4Кб*. З адреси 0000h розпочинає виконуватися програма під час системного скидання. Інші призначені для зберігання початкових адрес підпрограм обслуговування переривань від зовнішніх сигналів, таймерів-лічильників або послідовного інтерфейсу.
2. Резидентної пам’яті даних (РПД) об’ємом *128б*, яка поділяється на
   1. Оперативний запам’ятовуючий пристрій (ОЗП), що містить чотири банки регістрів загального призначення, до регістрів яких команди можуть звертатися за їх іменами *R7* – *R0* та шістнадцять байт (*20h* – *2Fh*), що утворюють область комірок, до яких можливе застосування прямої побітової адресації.
   2. Регістри спеціальних функцій *SFR*, до складу якого входять
      1. Восьмирозрядний акумулятор *ACC*, який використовується як допоміжний регістр під час виконання операцій
      2. Регістр-розширювач аккумулятора *B*
      3. Регістр слова стану програми *PSW*
      4. Покажчик стека *SP*
      5. Регістр-покажчик даних DPTR
      6. Регістри таймерів-лічильників
      7. Буфер прийомопередавача
      8. Регістри управління
3. Арифметико-логічного пристрою (АЛП), завдяки якому МК виконує арифметичні та логічні дії
4. Блока таймерів-лічильників
5. Пристрою управління та синхронізації
6. Блока переривань і послідовного порту
7. Чотирьох портів вводу-виводу (*P0, P1, P2, P3*).

До МК підключено:

1. *16* сторінок **зовнішньої пам’яті даних** (ЗПД) об’ємом *256б*. Кожна сторінка пам’яті даних обирається через порт *P1* безпосередньо через вихід порту (або через дешифратор). Першасторінка використовується для зберігання адрес *111* зовнішніх пристроїв. Зовнішні пристрої належать загальному адресному простору пам’яті даних.
2. *6* сторінок **зовнішньої пам’яті програм** (ЗПП) об’ємом *16Кб*.
3. *111* **зовнішніх пристроїв** (ЗП).
4. **Контролер пріоритетних переривань** (КПП) – реалізує зовнішні векторні переривання. Коли зовнішній пристрій став готовим до взаємодії, він надсилає на контролер запит на переривання *IRQ*. По цих запитах КПП видає на процесор сигнал вимоги переривання *INT*. Після його отримання процесор перериває виконання поточної програми наступним чином:
   1. Завершує виконання поточної команди.
   2. Зберігає у стеку стан програми, що переривається, та адресу повернення до неї.
   3. Видає на КПП сигнал підтвердження переривання *IACK*
   4. Зчитує вектор, що виставляє на шину даних КПП, по ньому обчислює адресу переходу до підпрограми обробки переривання та переходить на її першу команду.
   5. Виконує підпрограму, остання команда якої відновлює стан перерваної основної програми за інформацією, збереженою в стеці.
5. **Контролер прямого доступу до пам’яті даних** (КПДП) – реалізує режим прямого доступу до пам’яті, який допомагає прискорити обмін даними між елементами системи, адже під час роботи КПДП може паралельно виконувати завдання, що не вимагають доступу до пам’яті. Відбувається це наступним чином:
   1. КПДП надсилає сигнал вимоги захоплення шини (*HRQ -* Hold Request)
   2. Процесор отримує цей сигнал, виконує попередню ініціалізацію контролера для чого пересилає по шині даних в контролер прямого доступу інформацію, необхідну для управління обміном (адресу комірки пам’яті, в якій розміщується перший байт даних, що записуються або зчитуються, адресу порту, загальну кількість даних, що передаються, напрям передачі та інше) та відключається від шини даних і шини адреси, надаючи їх контролеру для організації обміну за допомогою сигналу підтвердження захоплення шини (*HLDA* – Hold Acknowledge).
   3. КПДП виконує обмін даними між ЗП та СП, який координується за допомогою сигналів вимоги ПДП (*DRQ* – DMA Request) та підтвердження ПДП (*DACK* – DMA Acknowledge).
   4. Процесор отримує сигнал від контролера і переходить до виконання основної програми.
6. **Програмований периферійний адаптер** (ППА) - забезпечує ввід/вивід за трьома додатковими восьмирозрядними портами РА, РВ, РС. по шині даних відбувається не тільки обмін даними, але і пересилання з МК в ППА управляючих слів, генерованих програмним забезпеченням процесора, а також передача в МК інформації про стан периферійного обладнання. Налаштування внутрішньої організації ППА – тобто підключення його до конкретного обладнання виконується за допомогою управляючого слова режиму роботи (УСРР), що міститься у регістрі управляючого слова (РУС)
7. **Селектори адрес** (СА), що за адресою, виставленою на шині визначають, до якого з елементів системи треба підключитись.
8. **Дешифратор** (DC) для вибору сторінок ЗПД та ЗПП

## 2. Карта розподілу пам’яті даних

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | ЗПД0 | |
| FFFFh | 1111 | 1111 | 1111 | 1111 |  | |
| -------- | ------ | ------ | ------ | ------ |
| 5027h | 0101 | 0000 | 0010 | 011**1** | КПДП | |
| 5026h | 0101 | 0000 | 0010 | 011**0** |
| 5025h | 0101 | 0000 | 0010 | 010**1** | КПП | |
| 5024h | 0101 | 0000 | 0010 | 010**0** |
| 5023h | 0101 | 0000 | 0010 | 00**11** | РУС | ППА |
| 5022h | 0101 | 0000 | 0010 | 00**10** | РС |
| 5021h | 0101 | 0000 | 0010 | 00**01** | РВ |
| 5020h | 0101 | 0000 | 0010 | 00**00** | РА |
| -------- | ------ | ------ | ------ | ------ |  | |
| 084Bh | 0000 | 1000 | 0100 | 101**1** | ЗП110 (РД) | Зовнішні пристрої |
| 084Ah | 0000 | 1000 | 0100 | 101**0** | ЗП110 (РС) |
| 0849h | 0000 | 1000 | 0100 | 100**1** | ЗП109 (РД) |
| 0848h | 0000 | 1000 | 0100 | 100**0** | ЗП109 (РС) |
| -------- | ------ | ------ | ------ | ------ |  |
| 0775h | 0000 | 0111 | 0111 | 010**1** | ЗП3 (РД) |
| 0774h | 0000 | 0111 | 0111 | 010**0** | ЗП3 (РС) |
| 0773h | 0000 | 0111 | 0111 | 001**1** | ЗП2 (РД) |
| 0772h | 0000 | 0111 | 0111 | 001**0** | ЗП2 (РС) |
| 0771h | 0000 | 0111 | 0111 | 000**1** | ЗП1 (РД) |
| 0770h | 0000 | 0111 | 0111 | 000**0** | ЗП1 (РС) |
| 076Fh | 0000 | 0111 | 0110 | 111**1** | ЗП0 (РД) |
| 076Eh | 0000 | 0111 | 0110 | 111**0** | ЗП0 (РС) |
| -------- | ------ | ------ | ------ | ------ |  | |
| 0000h | 0000 | 0000 | 0000 | 000 |  | |

## 3. Опис піраміди КПП

Для того аби підключити 111 ЗП необхідно використати 111/8 = 13,875 ≈ 14 схем *КМ1804ВН1*( *PIC8259A*). Для формування старших розрядів вектора переривання за допомогою шифратора необхідно підключити на його входи 14 схем *PIC8259A,* тому використаємо 2 схеми *КМ1804ВР3* (*IC8257А)*. Піраміду КПП з використанням цих елементів зображено на рисунку 2, на ній червоним кольором показаний шлях формування вектора переривання для зовнішнього пристрою *106*.

# Частина 2

## Лабораторна робота №1 (знайомство із середовищем моделювання ModelSim)

*Завдання****:*** Изменяя значение входных сигналов и временные параметры в файле **Stim.do**, проверьте работоспособность тестируемого устройства.

### Виконання роботи

Лістинг файлу **half\_adder\_rtl.v**:

`timescale 1 ns/1 ps

module half\_adder\_rtl(And, Xor, A, B);

output And, Xor;

input A, B;

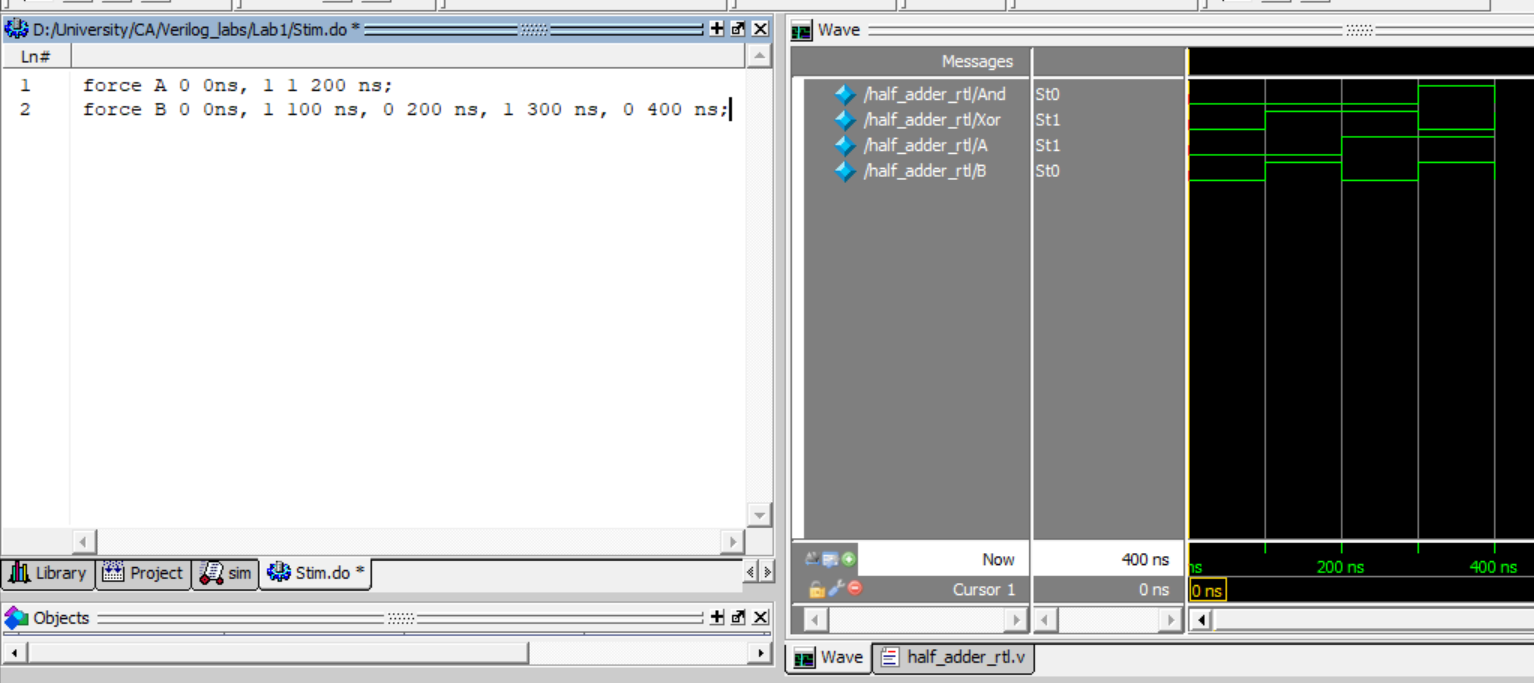
wire And, Xor;

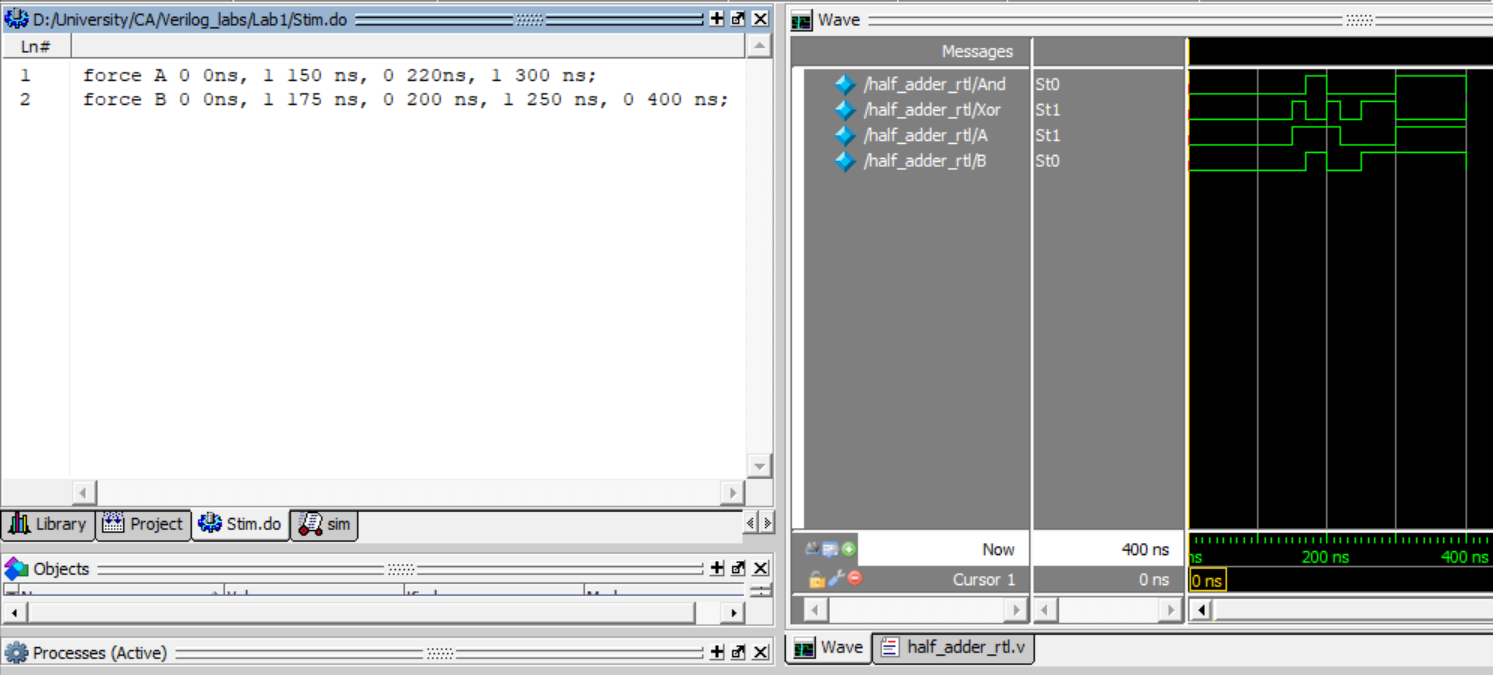
assign And = A & B;

assign Xor = A ^ B;

endmodule

Тестування при різних параметрах в файлі **Stim.do:**





*Висновок:* програма працює належним чином.

## Лабораторна робота №2 (дослідження комбінаційних пристроїв)

*Варіант завдання****:***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| *h*10 | *h*9 | *h*8 | *h*7 | *h*6 | *h*5 | *h*4 | *h*3 | *h*2 | *h*1 |

Таблиця істинності перемикальних функцій

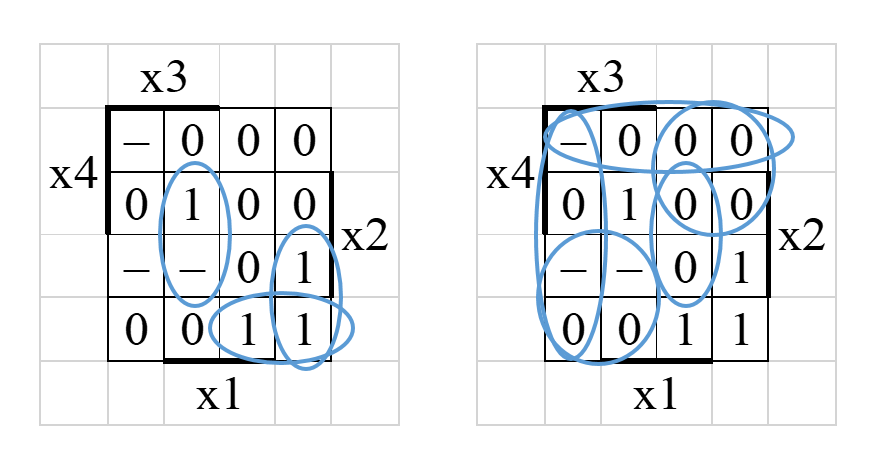
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***x*4** | ***x*3** | ***x*2** | ***x*1** | ***f*** |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | – |
| 0 | 1 | 1 | 1 | – |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | – |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Елементний базис

|  |  |  |  |
| --- | --- | --- | --- |
| ***h*10** | ***h*9** | ***h*8** | **Логічні елементи** |
| 1 | 1 | 1 | І-НЕ, АБО-НЕ |

### Виконання роботи

1. Мінімізуємо перемикальну функцію *f* методом діаграм Вейча:



2. Виходячи з діаграми, записуємо МДНФ функції та її заперечення, з яких отримаємо 8 нормальних форм:



















3. Виходячи з елементного базису, побудуємо комбінаційні схеми для отриманих форм І-НЕ/І-НЕ та АБО-НЕ/АБО-НЕ:

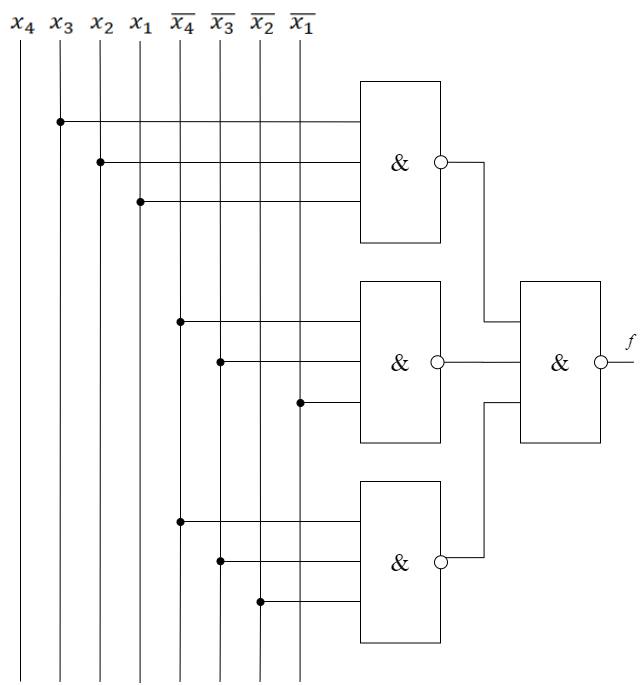


Схема для форми І-НЕ/І-НЕ

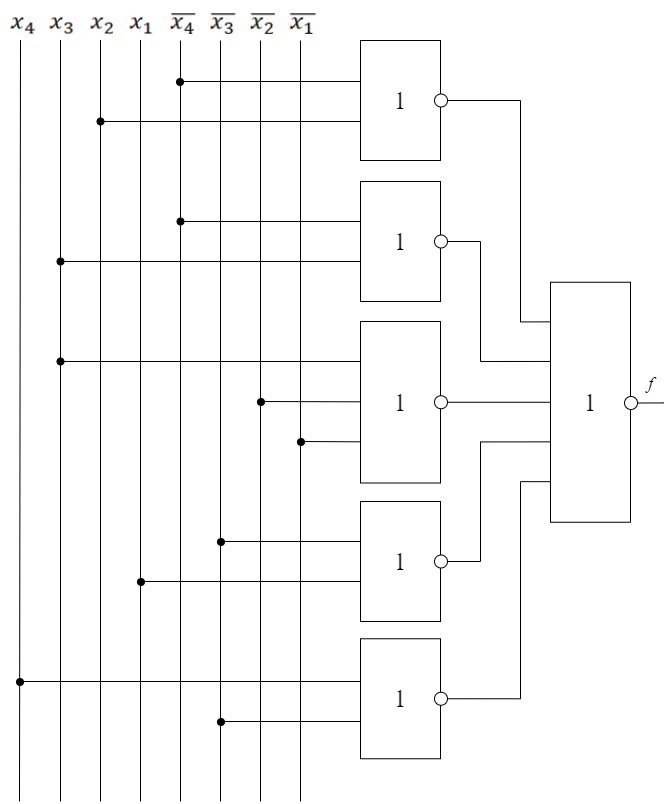


Схема для форми АБО-НЕ/АБО-НЕ

4. Виконаємо моделювання роботи комбінаційних схем в програмі ModelSim. Для цього створимо модулі, що будуть відповідати заданим елементам:

Лістинг файлу **My\_NOR2.v**:

module My\_NOR2(OUT, IN1, IN2);

input IN1, IN2;

output OUT;

wire OUT;

assign OUT = ~(IN1 | IN2);

endmodule

Лістинг файлу **My\_NOR3.v**:

module My\_NOR3(OUT, IN1, IN2, IN3);

input IN1, IN2, IN3;

output OUT;

wire OUT;

assign OUT = ~(IN1 | IN2 | IN3);

endmodule

Лістинг файлу **My\_NOR5.v**:

module My\_NOR5(OUT, IN1, IN2, IN3, IN4, IN5);

input IN1, IN2, IN3, IN4, IN5;

output OUT;

wire OUT;

assign OUT = ~(IN1 | IN2 | IN3 | IN4 | IN5);

endmodule

Лістинг файлу **My\_NAND3.v**:

module My\_NAND3(OUT, IN1, IN2, IN3);

input IN1, IN2, IN3;

output OUT;

wire OUT;

assign OUT = ~(IN1 & IN2 & IN3);

endmodule

Створимо модуль перемикальної функціїї, де поєднаємо модулі відповідно до побудованих комбінаційних схем:

Лістинг файлу **My\_function.v**:

module My\_function(f\_nand, f\_nor, x4, x3, x2, x1);

input x4, x3, x2, x1;

output f\_nand, f\_nor;

wire s1, s2, s3, s4, s5, s6, s7, s8;

`include "My\_NAND3.v";

`include "My\_NOR2.v";

`include "My\_NOR3.v";

`include "My\_NOR5.v";

// Function on NAND basis

My\_NAND3 my\_nand3\_1( .OUT (s1), .IN1 (x3), .IN2 (x2), .IN3 (x1) );

My\_NAND3 my\_nand3\_2( .OUT (s2), .IN1 (~x4), .IN2 (~x3), .IN3 (~x1) );

My\_NAND3 my\_nand3\_3( .OUT (s3), .IN1 (~x4), .IN2 (~x3), .IN3 (~x2) );

My\_NAND3 my\_nand3\_4( .OUT (f\_nand), .IN1 (s1), .IN2 (s2), .IN3 (s3) );

// Function on NOR basis

My\_NOR2 my\_nor2\_1( .OUT (s4), .IN1 (~x4), .IN2 (x2));

My\_NOR2 my\_nor2\_2( .OUT (s5), .IN1 (~x4), .IN2 (x3));

My\_NOR3 my\_nor3\_1( .OUT (s6), .IN1 (x3), .IN2 (~x2), .IN3 (~x1) );

My\_NOR2 my\_nor2\_3( .OUT (s7), .IN1 (~x3), .IN2 (x1));

My\_NOR2 my\_nor2\_4( .OUT (s8), .IN1 (x4), .IN2 (~x3));

My\_NOR5 my\_nor5\_1( .OUT (f\_nor), .IN1 (s4), .IN2 (s5), .IN3 (s6), .IN4 (s7), .IN5 (s8) );

endmodule

Створимо макрос для перевірки значення функції для всіх можливих комбінацій вхідних параметрів:

Лістинг файлу **Sim.do**:

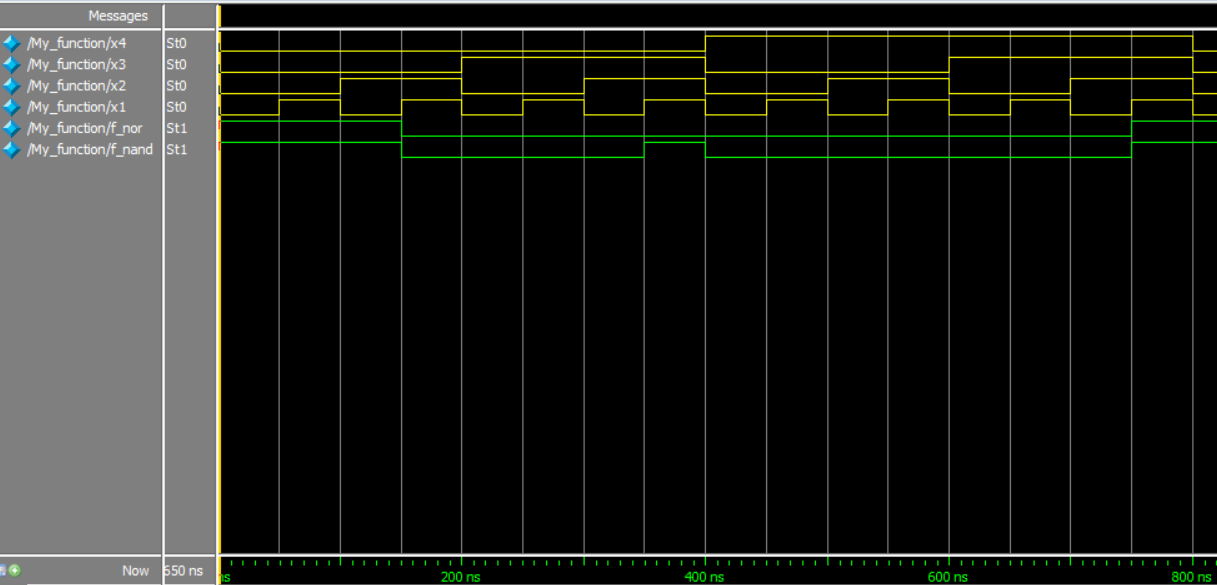
force x4 0 0ns, 1 400ns, 0 800 ns;

force x3 0 0ns, 1 {200ns} -repeat 400 ns;

force x2 0 0ns, 1 {100ns} -repeat 200 ns;

force x1 0 0ns, 1 {50ns} -repeat 100 ns;

Та виконаємо симуляцію:



*Висновки:* як можна бачити з результатів симуляції, мінімальні форми були знайдені коректно, обидва варіанта побудови функції дають однаковий результат, який відповідає заданому у таблиці, за вийнятком невизначених у таблиці значень.

## Лабораторна робота №3 (комбінаційні пристрої)

*Завдання* *1 (варіант 101)*: Реалізувати 4-х розрядний демультиплексор 1-в-5 (кількість інформаційних виходів – 5, розрядність інформаційної шини – 4 біт) і перевірити його роботу в середовищі ModelSim.

### Виконання завдання 1

Для того щоб реалізувати демультиплексор на 5 інформаційних виходів, потрібно  адресних входів. Побудуємо таблицю істинності:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A2 | A1 | A0 | Q4 | Q3 | Q2 | Q1 | Q0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | D |
| 0 | 0 | 1 | 0 | 0 | 0 | D | 0 |
| 0 | 1 | 0 | 0 | 0 | D | 0 | 0 |
| 0 | 1 | 1 | 0 | D | 0 | 0 | 0 |
| 1 | 0 | 0 | D | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Реалізуємо 3-розрядний дешифратор.

Лістинг файлу **My\_decoder3.v**:

module My\_Decoder3(OUT, IN1, IN2, IN3);

input IN1, IN2, IN3;

output [7:0] OUT;

assign OUT[0] = ~IN1 & ~IN2 & ~IN3;

assign OUT[1] = ~IN1 & ~IN2 & IN3;

assign OUT[2] = ~IN1 & IN2 & ~IN3;

assign OUT[3] = ~IN1 & IN2 & IN3;

assign OUT[4] = IN1 & ~IN2 & ~IN3;

assign OUT[5] = IN1 & ~IN2 & IN3;

assign OUT[6] = IN1 & IN2 & ~IN3;

assign OUT[7] = IN1 & IN2 & IN3;

endmodule

Використаємо його для створення демультиплексора відповідно завданню.

Лістинг файлу **My\_Demultiplexor\_1\_to\_5.v**:

module My\_Demultiplexor\_1\_to\_5(Q4, Q3, Q2, Q1, Q0, D, A);

input [2:0] A;

input [3:0] D;

output [3:0] Q4;

output [3:0] Q3;

output [3:0] Q2;

output [3:0] Q1;

output [3:0] Q0;

wire [7:0] adr;

genvar i;

`include "My\_Decoder3.v";

My\_Decoder3 decode(.OUT (adr), .IN1 (A[2]), .IN2 (A[1]), .IN3 (A[0]));

for(i=0; i<4; i=i+1)

begin

assign Q4[i] = adr[4] & D[i];

end

for(i=0; i<4; i=i+1)

begin

assign Q3[i] = adr[3] & D[i];

end

for(i=0; i<4; i=i+1)

begin

assign Q2[i] = adr[2] & D[i];

end

for(i=0; i<4; i=i+1)

begin

assign Q1[i] = adr[1] & D[i];

end

for(i=0; i<4; i=i+1)

begin

assign Q0[i] = adr[0] & D[i];

end

endmodule

Створимо *testbench* для тестування роботи системи.

Лістинг файлу **Test\_Demultiplexor.v**:

`timescale 1ns/1ns;

module Test\_Demultiplexor; // Top Level Testbench

wire [2:0] A;

wire [3:0] D;

reg [2:0] A\_r;

reg [3:0] D\_r;

wire [3:0] Q4, Q3, Q2, Q1, Q0;

`include "My\_Demultiplexor\_1\_to\_5.v"

My\_Demultiplexor\_1\_to\_5 demultiplexor (Q4, Q3, Q2, Q1, Q0, D, A);

initial

begin

$display("\t\t Time Q4 Q3 Q2 Q1 Q0 A D");

$monitor($time,,,,,Q4,,,,,Q3,,,,,Q2,,,,,Q1,,,,,Q0,,,,,A,,,,,D);

#400 $finish;

end

initial

begin

D\_r = 10;

A\_r = 0;

#50 A\_r = 1;

#50 A\_r = 2;

#50 A\_r = 3;

#50 A\_r = 4;

#50 A\_r = 5;

#50 A\_r = 6;

#50 A\_r = 7;

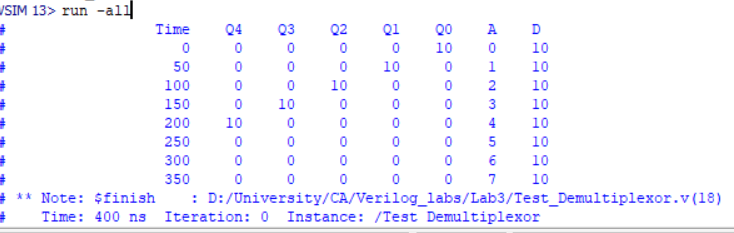
end

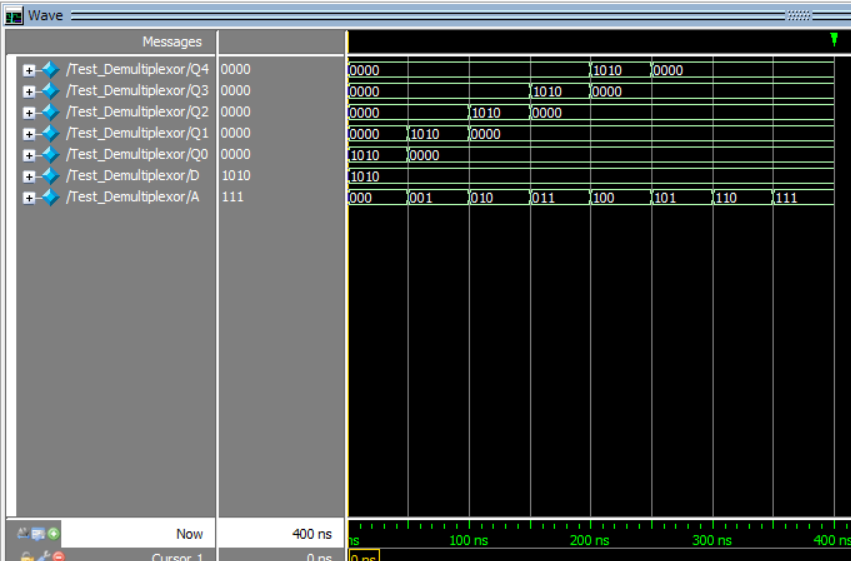
assign A = A\_r;

assign D = D\_r;

endmodule

Виконаємо симуляцію та проаналізуємо її результати:





*Висновок:* можна побачити, що вивід тестового файлу після виконання симуляції ідентичний до побудованої раніше таблиці істинності, отже програма працює правильно.

*Завдання* *2 (варіант 101)*: Розробіть фрагмент АЛБ для виконання наступних мікрооперацій:

|  |  |  |  |
| --- | --- | --- | --- |
| **Блок №1** | **Блок №2** | **Блок №3** | **Блок №4** |
|  |  |  |  |

Розробити тестовий рівень для тестування роботи системи.

### Виконання завдання 2

Побудуємо таблиці істинності для 1-розрядних суматора (зліва) та від’ємника (справа), які будуть використовуватися у блоках №2 та №4 відповідно. У таблиці для від’ємника через *ZI*=-(1+*CI*)=~*CI* позначена наявність позики з попереднього розряду:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **CIN** | **SUM** | **COUT** |  | **A** | **B** | **CIN** | **ZI** | **SUB** | **ZOUT** |
| 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |  | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |  | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |  | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |  | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |  | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |  | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 0 | 0 | 0 |

Створимо модулі для блоків відповідно до варіанту а також головний тестовий модуль.

Лістинг файлу **Test\_Blocks.v**:

`timescale 1ns/1ns;

module My\_Block1(ALB1, R, S);

input [3:0] R;

input [3:0] S;

output [3:0] ALB1;

assign ALB1 = R ^ S;

endmodule

module My\_Block2(ALB2, C2, R, S, CI);

input [3:0] R;

input [3:0] S;

input CI;

output [3:0] ALB2;

output C2;

wire [3:0] C;

bitsum sum1(.SUM (ALB2[0]), .COUT (C[0]), .A (R[0]), .B(S[0]), .CIN(CI));

bitsum sum2(.SUM (ALB2[1]), .COUT (C[1]), .A (R[1]), .B(S[1]), .CIN(C[0]));

bitsum sum3(.SUM (ALB2[2]), .COUT (C[2]), .A (R[2]), .B(S[2]), .CIN(C[1]));

bitsum sum4(.SUM (ALB2[3]), .COUT (C[3]), .A (R[3]), .B(S[3]), .CIN(C[2]));

assign C2 = C[3];

endmodule

module My\_Block3(ALB3, R, S);

input [3:0] R;

input [3:0] S;

output [3:0] ALB3;

assign ALB3 = ~R & S;

endmodule

module My\_Block4(ALB4, Z4, R, S, CI);

input [3:0] R;

input [3:0] S;

input CI;

output [3:0] ALB4;

output Z4;

wire [3:0] Z;

bitsub sub1(.SUB (ALB4[0]), .ZOUT (Z[0]), .A (R[0]), .B(S[0]), .ZIN(~CI));

bitsub sub2(.SUB (ALB4[1]), .ZOUT (Z[1]), .A (R[1]), .B(S[1]), .ZIN(Z[0]));

bitsub sub3(.SUB (ALB4[2]), .ZOUT (Z[2]), .A (R[2]), .B(S[2]), .ZIN(Z[1]));

bitsub sub4(.SUB (ALB4[3]), .ZOUT (Z[3]), .A (R[3]), .B(S[3]), .ZIN(Z[2]));

assign Z4 = Z[3];

endmodule

module bitsum (SUM, COUT, A, B, CIN);

input A, B, CIN;

output SUM, COUT;

wire A, B, SUM, Res;

wire c1, c2, CIN, COUT;

xor(Res, A, B);

xor(SUM, CIN, Res);

and(c1, A, B);

and(c2, CIN, Res);

or(COUT, c1, c2);

endmodule

module bitsub (SUB, ZOUT, A, B, ZIN);

input A, B, ZIN;

output SUB, ZOUT;

wire A, B, SUB, Res;

wire z1, z2, z3, z4, ZIN, ZOUT;

xor(Res, A, B);

xor(SUB, ZIN, Res);

and(z1, ~A, B);

and(z2, ~A, ZIN);

and(z3, B, ZIN);

or(z4, z1, z2);

or(ZOUT, z3, z4);

endmodule

module Test\_Blocks; // Top Level Testbench

wire Ci, C2, Z4;

wire [3:0] R, S;

reg [3:0] R\_r, S\_r;

reg Ci\_r;

wire [3:0] ALB1, ALB2, ALB3, ALB4;

My\_Block1 block1(ALB1, R, S);

My\_Block2 block2(ALB2, C2, R, S, Ci);

My\_Block3 block3(ALB3, R, S);

My\_Block4 block4(ALB4, Z4, R, S, Ci);

initial

begin

$display("\t\t Time R S Ci R^S R+S+CI C2 ~R&S R-S-1+CI Z4"); $monitor($time,,,,,R,,,,,S,,,,,Ci,,,,,ALB1,,,,,,,,ALB2,,,,,,,C2,,,,,,,ALB3,,,,,,,,,,,ALB4,,,,,,,Z4);

#400 $finish;

end

initial

begin

R\_r = 0;

#50 R\_r = 4;

#50 R\_r = 11;

#50 R\_r = 10;

#50 R\_r = 7;

#50 R\_r = 6;

#50 R\_r = 5;

#50 R\_r = 8;

end

initial

begin

S\_r = 0;

#100 S\_r = 5;

#100 S\_r = 4;

#100 S\_r = 3;

end

initial

begin

Ci\_r = 0;

#200 Ci\_r = 1;

end

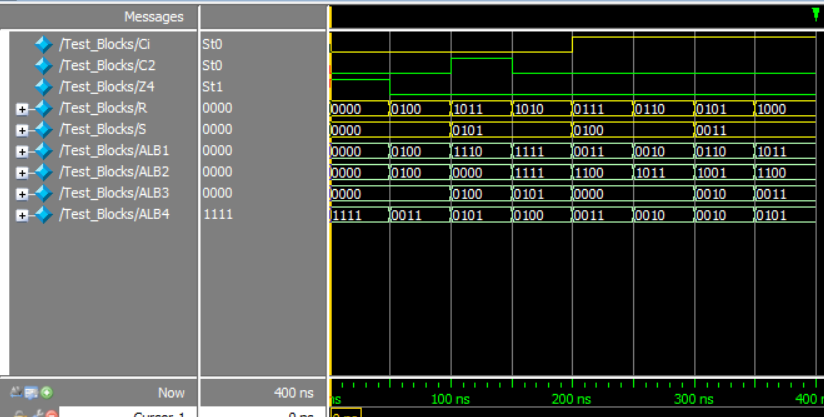
assign R = R\_r;

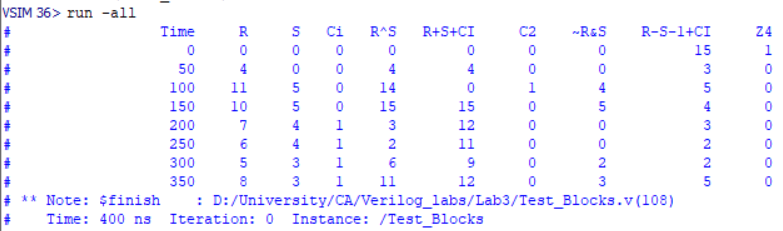
assign S = S\_r;

assign Ci = Ci\_r;

endmodule

Виконаємо симуляцію та проаналізуємо її результати:





*Висновок:* можна побачити, що результати виконання симуляції правильні (перевіряти результат блоків №2 та №4 зручніше по таблиці з десятковими результатами, а блоків №1 та №3 – по графіку з бінарними числами).

*Завдання* *3*: побудувати дешифратор для семисегментногоіндикатора. Активний рівень сигналу для увімкнення сегменту – логічний «0».

### Виконання завдання 3

Переробимо 3-розрядний дешифратор з минулого завдання у 4-розрядний.

Лістинг файлу **My\_decoder4.v**:

module My\_Decoder4(OUT, IN3, IN2, IN1, IN0);

input IN3, IN2, IN1, IN0;

output [15:0] OUT;

assign OUT[ 0] = ~IN3 & ~IN2 & ~IN1 & ~IN0;

assign OUT[ 1] = ~IN3 & ~IN2 & ~IN1 & IN0;

assign OUT[ 2] = ~IN3 & ~IN2 & IN1 & ~IN0;

assign OUT[ 3] = ~IN3 & ~IN2 & IN1 & IN0;

assign OUT[ 4] = ~IN3 & IN2 & ~IN1 & ~IN0;

assign OUT[ 5] = ~IN3 & IN2 & ~IN1 & IN0;

assign OUT[ 6] = ~IN3 & IN2 & IN1 & ~IN0;

assign OUT[ 7] = ~IN3 & IN2 & IN1 & IN0;

assign OUT[ 8] = IN3 & ~IN2 & ~IN1 & ~IN0;

assign OUT[ 9] = IN3 & ~IN2 & ~IN1 & IN0;

assign OUT[10] = IN3 & ~IN2 & IN1 & ~IN0;

assign OUT[11] = IN3 & ~IN2 & IN1 & IN0;

assign OUT[12] = IN3 & IN2 & ~IN1 & ~IN0;

assign OUT[13] = IN3 & IN2 & ~IN1 & IN0;

assign OUT[14] = IN3 & IN2 & IN1 & ~IN0;

assign OUT[15] = IN3 & IN2 & IN1 & IN0;

endmodule

Для зручності побудуємо таблицю відповідності сегментів числам (одиниця позначає відповідність, у коді вона перетвориться на потрібний логічний 0):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| a | 1 |  | 1 |  |  | 1 | 1 | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |
| b | 1 | 1 | 1 |  | 1 |  |  | 1 | 1 | 1 | 1 |  |  | 1 |  |  |
| c | 1 | 1 |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | 1 |  |  |
| d | 1 |  | 1 |  |  | 1 | 1 |  | 1 | 1 |  | 1 | 1 | 1 | 1 | 1 |
| e | 1 |  | 1 |  |  |  | 1 |  | 1 |  | 1 | 1 | 1 | 1 | 1 | 1 |
| f | 1 |  |  |  | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 | 1 |  | 1 | 1 |
| g |  |  | 1 |  | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 |

Створимо модуль перетворення двійкових чисел у сегменти та тестовий модуль.

Лістинг файлу **Test\_decoder.v**:

`timescale 1ns/1ns;

module Bin\_to\_segment(a, b, c, d, e, f, g, IN);

input [3:0] IN;

output a, b, c, d, e, f, g;

wire [15:0] dec;

`include "My\_Decoder4.v";

My\_Decoder4 decoder(dec, IN[3], IN[2], IN[1], IN[0]);

assign a = ~(dec[ 0] | dec[ 2] | dec[ 5] | dec[ 6] | dec[ 7] | dec[ 8] | dec[ 9] | dec[10] | dec[12] | dec[13] | dec[14] | dec[15]);

assign b = ~(dec[ 0] | dec[ 1] | dec[ 2] | dec[ 4] | dec[ 7] | dec[ 8] | dec[ 9] | dec[10] | dec[13]);

assign c = ~(dec[ 0] | dec[ 1] | dec[ 4] | dec[ 5] | dec[ 6] | dec[ 7] | dec[ 8] | dec[ 9] | dec[10] | dec[11] | dec[13]);

assign d = ~(dec[ 0] | dec[ 2] | dec[ 5] | dec[ 6] | dec[ 8] | dec[ 9] | dec[11] | dec[12] | dec[13] | dec[14] | dec[15]);

assign e = ~(dec[ 0] | dec[ 2] | dec[ 6] | dec[ 8] | dec[10] | dec[11] | dec[12] | dec[13] | dec[14] | dec[15]);

assign f = ~(dec[ 0] | dec[ 4] | dec[ 5] | dec[ 6] | dec[ 8] | dec[ 9] | dec[10] | dec[11] | dec[12] | dec[14] | dec[15]);

assign g = ~(dec[ 2] | dec[ 4] | dec[ 5] | dec[ 6] | dec[ 8] | dec[ 9] | dec[10] | dec[11] | dec[13] | dec[14] | dec[15]);

endmodule

module Test\_decoder; // Top Level Testbench

wire [3:0] IN;

reg [3:0] IN\_r;

wire a, b, c, d, e, f, g;

genvar i;

Bin\_to\_segment segment(a, b, c, d, e, f, g, IN);

initial

begin

$display("\t\t Time IN3 IN2 IN1 IN0 a b c d e f g");

$monitor($time,,,,,IN[3],,,,,IN[2],,,,,IN[1],,,,,IN[0],,,,,,,a,,,,,b,,,,,c,,,,,d,,,,,e,,,,,f,,,,,g);

#800 $finish;

end

initial

begin

IN\_r = 0;

#50 IN\_r = 1;

#50 IN\_r = 2;

#50 IN\_r = 3;

#50 IN\_r = 4;

#50 IN\_r = 5;

#50 IN\_r = 6;

#50 IN\_r = 7;

#50 IN\_r = 8;

#50 IN\_r = 9;

#50 IN\_r = 10;

#50 IN\_r = 11;

#50 IN\_r = 12;

#50 IN\_r = 13;

#50 IN\_r = 14;

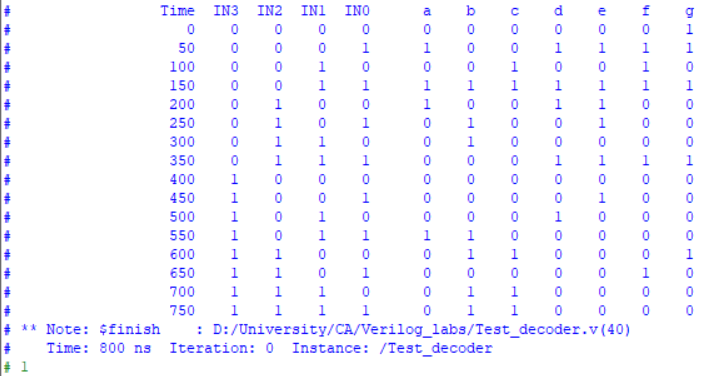
#50 IN\_r = 15;

end

assign IN = IN\_r;

endmodule

Виконаємо симуляцію та проаналізуємо її результати:



*Висновок:* можна побачити, що результати виконання симуляції відповідають побудованій раніше таблиці.

# Частина 3

## Структурна схема мікропроцесорної системи

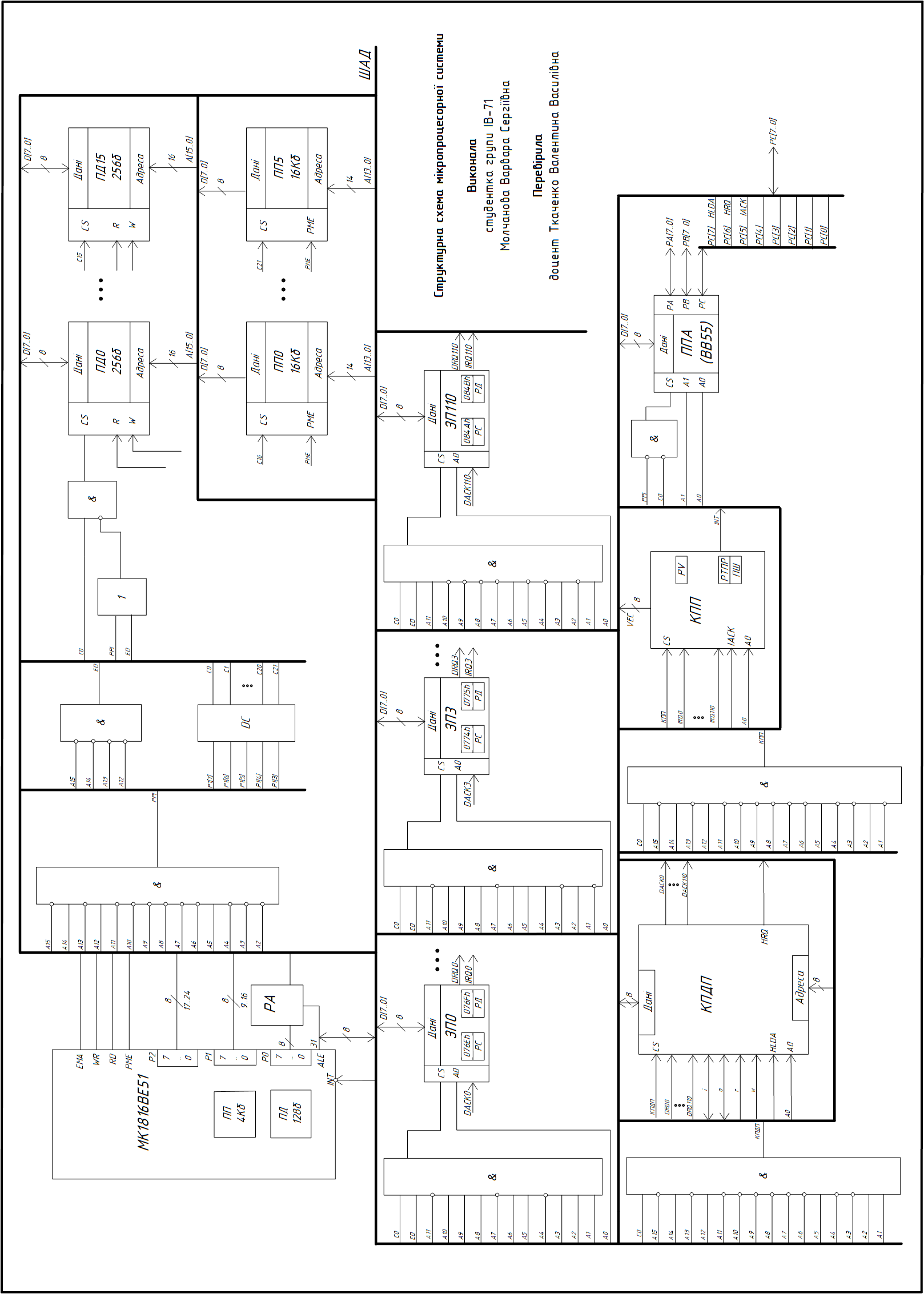


Рисунок 1. Структурна схема мікропроцесорної системи

## Функціональна схема контролера пріоритетного переривання

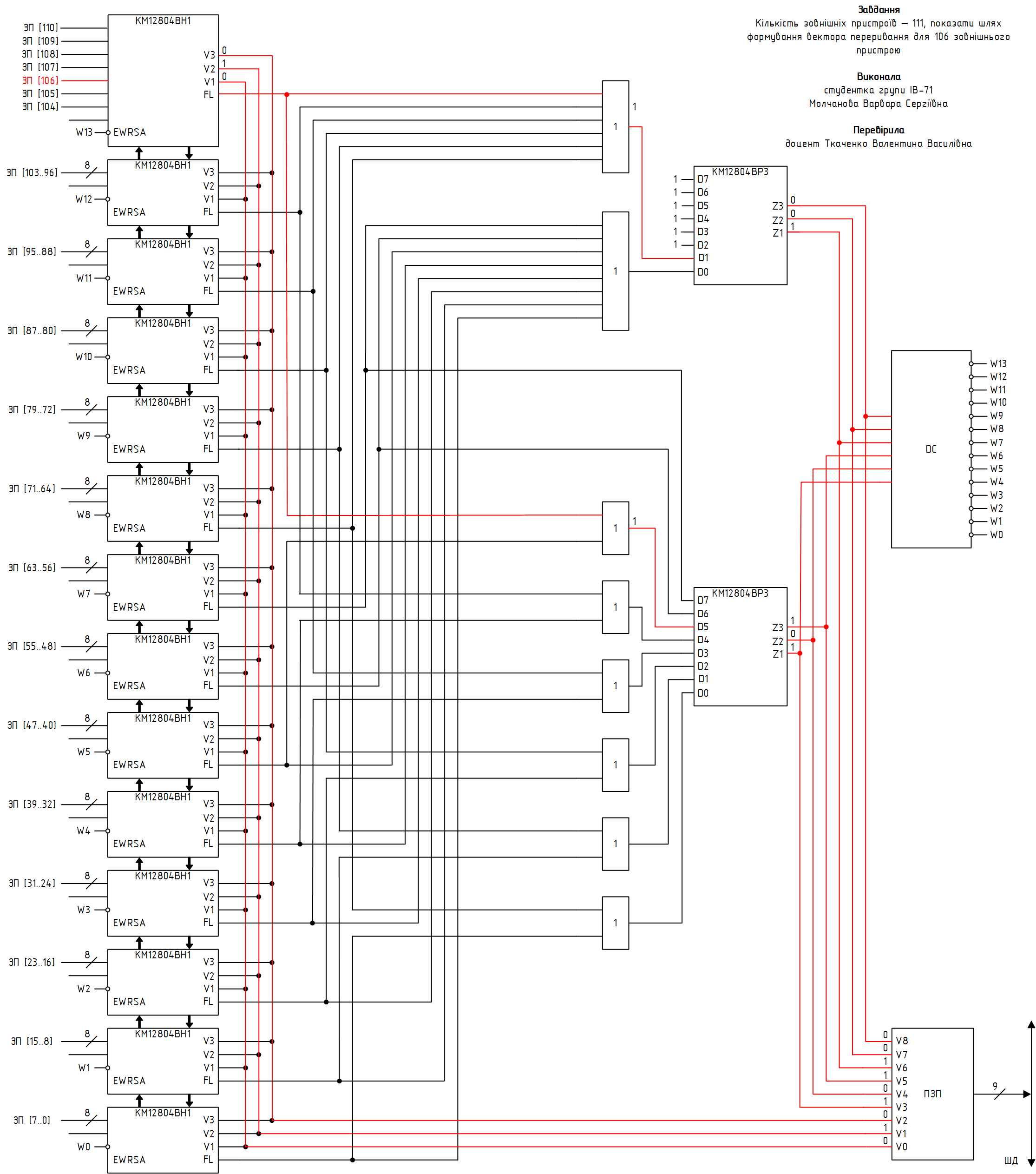


Рисунок 2. Піраміда КПП

# Висновки

У даній роботі було побудовано мікропроцесорну систему на основі мікроконтролера МК51, зображену на рисунку 1. Також було побудовано селектори адреси для цих робочих блоків та побудовано контролер пріоритетних переривань (піраміду КПП), зображений на рисунку 2.