**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп’ютерних технологій, автоматики та метрології**

**кафедра “Електронних обчислювальних машин”**

****

**Звіт**

З лабораторної роботи №3

З дисципліни: «Моделювання комп’ютерних систем»

На тему: «Поведінковий опис цифрового автомата Перевірка роботи автомата за допомогою стенда Elbert V2 – Spartan 3A FPGA»

***Варіант - 12***

Виконав: ст. гр. КІ-202

Лєсніков М.А.

Прийняв:

Козак Н.Б.

**Львів 2024**

**ЛАБОРАТОРНА РОБОТА №3**

**Поведінковий опис цифрового автомата Перевірка роботи автомата за допомогою стенда Elbert V2 – Spartan 3A FPGA**

**Мета роботи:**

На базі стенда реалізувати цифровий автомат для обчислення

значення виразу дотримуючись наступних вимог:

1. Функціонал пристрою повинен бути реалізований згідно отриманого варіанту завдання.
2. Пристрій повинен бути ітераційним АЛП повинен виконувати за один такт одну операцію та реалізованим згідно наступної структурної схеми(рис.3.1).
3. Кожен блок структурної схеми повинен бути реалізований на мові VHDL в окремому файлі Дозволено використовувати всі оператори.
4. Для кожного блока структурної схеми повинен бути згенерований символ.
5. Інтеграція структурних блоків в єдину систему та зі стендом.
6. Кожен структурний блок і схема вцілому повинні бути промодельовані за допомогою симулятора ISim.
7. Формування вхідних даних на шині DATA\_IN повинно бути реалізовано за допомогою DIP перемикачів.
8. Керування пристроєм повинно бути реалізовано за допомогою PUSH BUTTON кнопок.
9. Індикація значень операндів при вводі та вивід результату обчислень повинні бути реалізовані за допомогою семи сегментних індикаторів Індикація переповнення в АЛП за допомогою LED D8.
10. Підготувати та захистити звіт.

**Вхідні параметри**

Пристрій повинен реалізувати обчислення такого виразу:



**Виконання роботи:**

1. Створити новий .vhd файл, та реалізувати на ньому мультиплексор MUX.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity MUX is

Port ( SEL : in STD\_LOGIC\_VECTOR (1 downto 0);

CONST1 : in STD\_LOGIC\_VECTOR (7 downto 0);

CONST2 : in STD\_LOGIC\_VECTOR (7 downto 0);

RAM\_DATA\_OUT : in STD\_LOGIC\_VECTOR (7 downto 0);

DATA\_IN : in STD\_LOGIC\_VECTOR (7 downto 0);

O : out STD\_LOGIC\_VECTOR (7 downto 0));

end MUX;

architecture MUX\_arch of MUX is

begin

PROCESS (SEL, CONST1, CONST2, RAM\_DATA\_OUT, DATA\_IN)

BEGIN

IF (SEL = "00") THEN

O <= DATA\_IN;

ELSIF (SEL = "01") THEN

O <= RAM\_DATA\_OUT;

ELSIF (SEL = "10") THEN

O <= CONST1;

ELSE

O <= CONST2;

END IF;

END PROCESS;

end MUX\_arch;

2. Перевірити роботу мультиплексора за допомогою симулятора ISim.

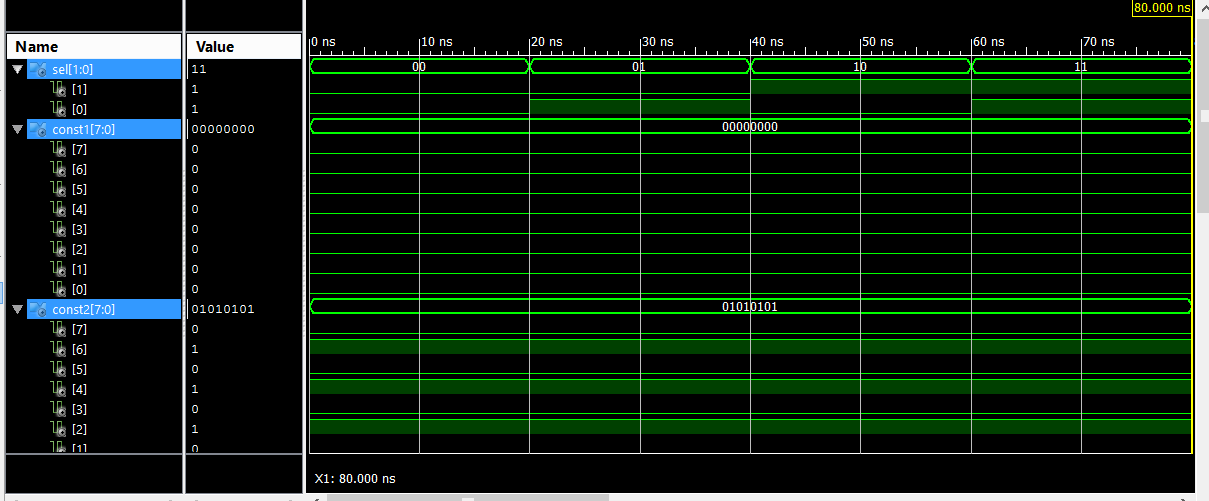


Рис.3.3. Симуляція мультиплексора в ISim.

3. Створити новий .vhd файл, та реалізувати на ньому регістр ACC.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity ACC is

Port ( WR : in STD\_LOGIC;

RST : in STD\_LOGIC;

CLK : in STD\_LOGIC;

IN\_BUS : in STD\_LOGIC\_VECTOR (7 downto 0);

OUT\_BUS : out STD\_LOGIC\_VECTOR (7 downto 0));

end ACC;

architecture ACC\_arch of ACC is

signal DATA : STD\_LOGIC\_VECTOR (7 downto 0);

begin

process (CLK)

begin

if rising\_edge(CLK) then

if RST = '1' then

DATA <= (others => '0');

elsif WR = '1' then

DATA <= IN\_BUS;

end if;

end if;

end process;

OUT\_BUS <= DATA;

end ACC\_arch;

4. Перевірити роботу регістра ACC (запис/скидання) за допомогою симулятора ISim.

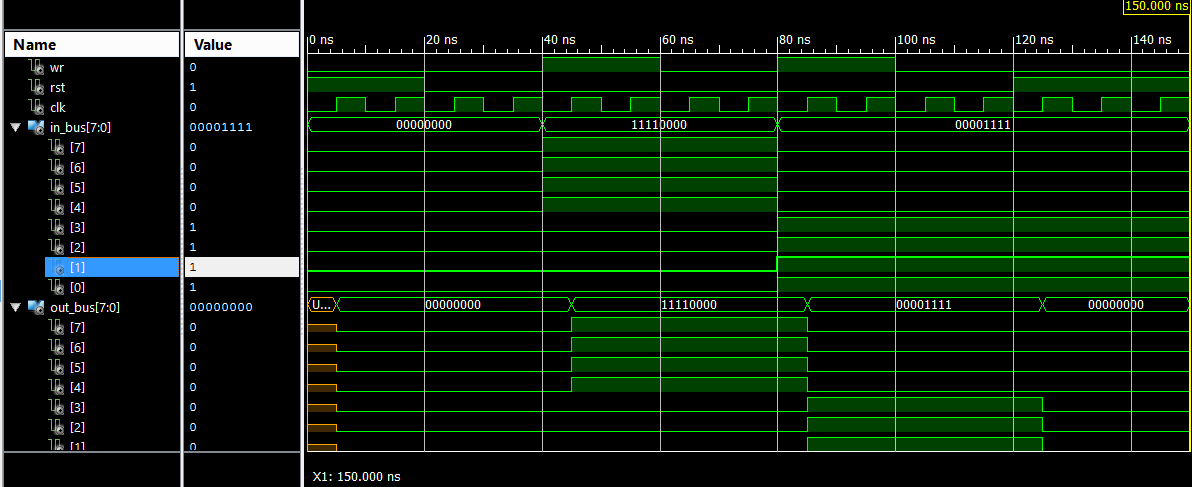


Рис.3.5. Симуляція регістра в ISim.

5. Визначити набір необхідних операції для обчислення індивідуального виразу.

Список набір операцій:

1. nop – передача даних із входу B на вихід АЛП.
2. “xor“
3. “+”
4. “-”

6. Створити новий .vhd файл, та реалізувати на ньому АЛП ALU.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use ieee.NUMERIC\_STD.all;

entity ALU is

Port (

A, B : in STD\_LOGIC\_VECTOR(7 downto 0);

ALU\_Sel : in STD\_LOGIC\_VECTOR(1 downto 0);

ALU\_Out : out STD\_LOGIC\_VECTOR(7 downto 0);

Carryout : out std\_logic

);

end ALU;

architecture Behavioral of ALU is

signal ALU\_Result : std\_logic\_vector (8 downto 0);

begin

process(A,B,ALU\_Sel)

begin

case(ALU\_Sel) is

when "01" =>

ALU\_Result <= ('0' & A) xor ('0' & B);

when "10" =>

ALU\_Result <= ('0' & A) + ('0' & B);

when "11" =>

ALU\_Result <= ('0' & A) - ('0' & B);

when others => ALU\_Result <= ('0' & B);

end case;

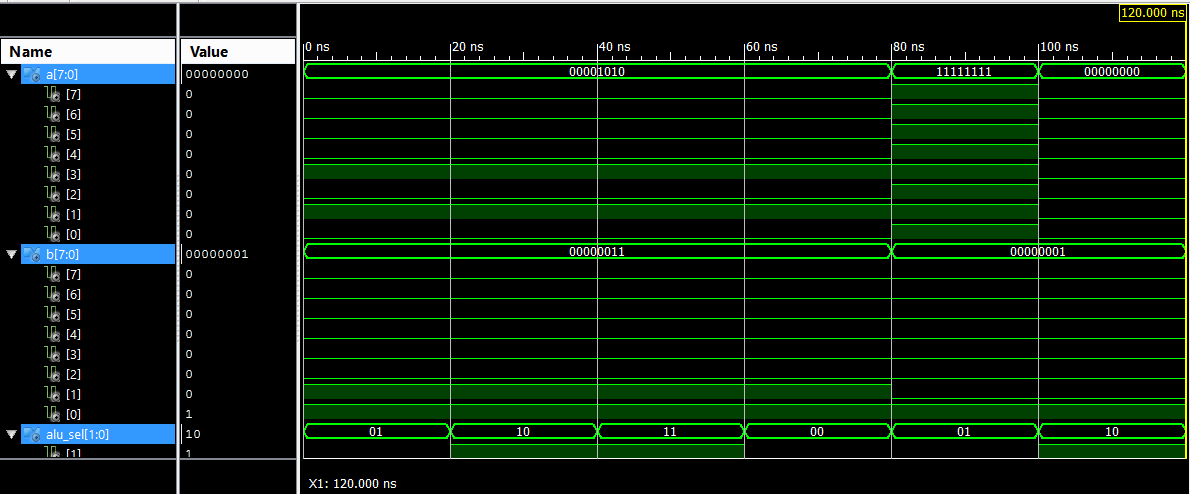
end process;

ALU\_Out <= ALU\_Result(7 downto 0);

Carryout <= ALU\_Result(8);

end Behavioral;

7. Перевірити роботу АЛП за допомогою симулятора ISim.



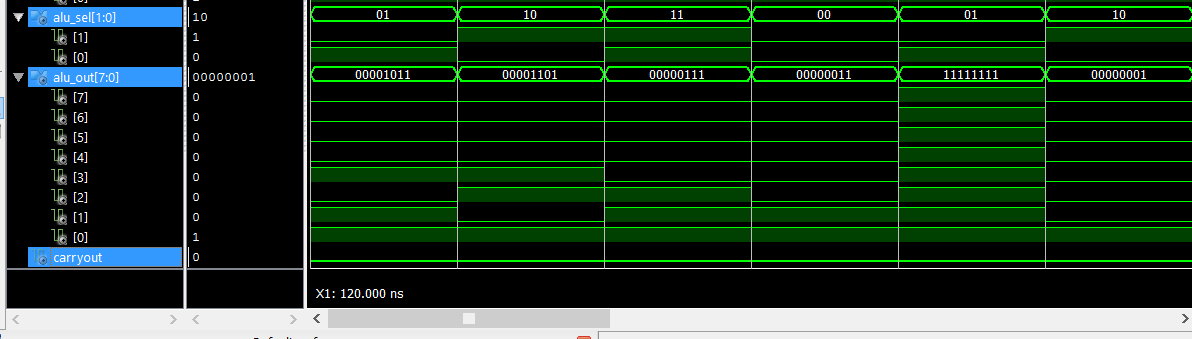


Рис.3.7. Симуляція АЛП в ISim.

8. Визначити множину станів та умови переходів пристрою керування необхідних для обчислення виразу.

Опис кожного зі станів:

* RST – скидання схеми до початкового стану.
* IDLE – стан очікування. Чекає на вхідні сигнали ENTER\_OP1, ENTER\_OP2 або CALCULATE.
* LOAD\_OP1 – запис першого операнда OP1 в ОЗП.
* LOAD\_OP2 – запис другого операнда OP2 в ОЗП.
* RUN\_CALC0: ACC = RAM(0x00);
* RUN\_CALC1: ACC = (ACC xor OP2);
* RUN\_CALC2: ACC = (ACC + OP1);
* RUN\_CALC3: ACC = (ACC + 15);
* RUN\_CALC4: ACC = (ACC - 3);
* FINISH – індикація кінцевого результату.

9. Створити новий .vhd файл, та реалізувати на ньому блок керування CU.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CU is

port( ENTER\_OP1 : IN STD\_LOGIC;

ENTER\_OP2 : IN STD\_LOGIC;

CALCULATE : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

CLOCK : IN STD\_LOGIC;

RAM\_WR : OUT STD\_LOGIC;

RAM\_ADDR\_BUS : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

CONST1\_BUS : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

CONST2\_BUS : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ACC\_WR : OUT STD\_LOGIC;

ACC\_RST : OUT STD\_LOGIC;

MUX\_SEL\_BUS : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

OP\_CODE\_BUS : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0));

end CU;

architecture CU\_arch of CU is

type STATE\_TYPE is (RST, IDLE, LOAD\_OP1, LOAD\_OP2, RUN\_CALC0, RUN\_CALC1, RUN\_CALC2, RUN\_CALC3, RUN\_CALC4, FINISH);

signal CUR\_STATE : STATE\_TYPE;

signal NEXT\_STATE : STATE\_TYPE;

begin

CONST1\_BUS <= "00001111";

CONST2\_BUS <= "00000001";

SYNC\_PROC: process (CLOCK)

begin

if (rising\_edge(CLOCK)) then

if (RESET = '1') then

CUR\_STATE <= RST;

else

CUR\_STATE <= NEXT\_STATE;

end if;

end if;

end process;

NEXT\_STATE\_DECODE: process (CUR\_STATE, ENTER\_OP1, ENTER\_OP2, CALCULATE)

begin

--declare default state for next\_state to avoid latches

NEXT\_STATE <= CUR\_STATE; --default is to stay in current state

--insert statements to decode next\_state

--below is a simple example

case(CUR\_STATE) is

when RST =>

NEXT\_STATE <= IDLE;

when IDLE =>

if (ENTER\_OP1 = '1') then

NEXT\_STATE <= LOAD\_OP1;

elsif (ENTER\_OP2 = '1') then

NEXT\_STATE <= LOAD\_OP2;

elsif (CALCULATE = '1') then

NEXT\_STATE <= RUN\_CALC0;

else

NEXT\_STATE <= IDLE;

end if;

when LOAD\_OP1 =>

NEXT\_STATE <= IDLE;

when LOAD\_OP2 =>

NEXT\_STATE <= IDLE;

when RUN\_CALC0 =>

NEXT\_STATE <= RUN\_CALC1;

when RUN\_CALC1 =>

NEXT\_STATE <= RUN\_CALC2;

when RUN\_CALC2 =>

NEXT\_STATE <= RUN\_CALC3;

when RUN\_CALC3 =>

NEXT\_STATE <= RUN\_CALC4;

when RUN\_CALC4 =>

NEXT\_STATE <= FINISH;

when FINISH =>

NEXT\_STATE <= FINISH;

when others =>

NEXT\_STATE <= IDLE;

end case;

end process;

OUTPUT\_DECODE: process (CUR\_STATE)

begin

case(CUR\_STATE) is

when RST =>

MUX\_SEL\_BUS <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '1';

ACC\_WR <= '0';

when IDLE =>

MUX\_SEL\_BUS <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when LOAD\_OP1 =>

MUX\_SEL\_BUS <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '1';

ACC\_RST <= '0';

ACC\_WR <= '1';

when LOAD\_OP2 =>

MUX\_SEL\_BUS <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '1';

ACC\_RST <= '0';

ACC\_WR <= '1';

when RUN\_CALC0 =>

MUX\_SEL\_BUS <= "01";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when RUN\_CALC1 =>

MUX\_SEL\_BUS <= "01";

OP\_CODE\_BUS <= "01";

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when RUN\_CALC2 =>

MUX\_SEL\_BUS <= "01";

OP\_CODE\_BUS <= "10";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when RUN\_CALC3 =>

MUX\_SEL\_BUS <= "10";

OP\_CODE\_BUS <= "10";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when RUN\_CALC4 =>

MUX\_SEL\_BUS <= "11";

OP\_CODE\_BUS <= "11";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when FINISH =>

MUX\_SEL\_BUS <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when others =>

MUX\_SEL\_BUS <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

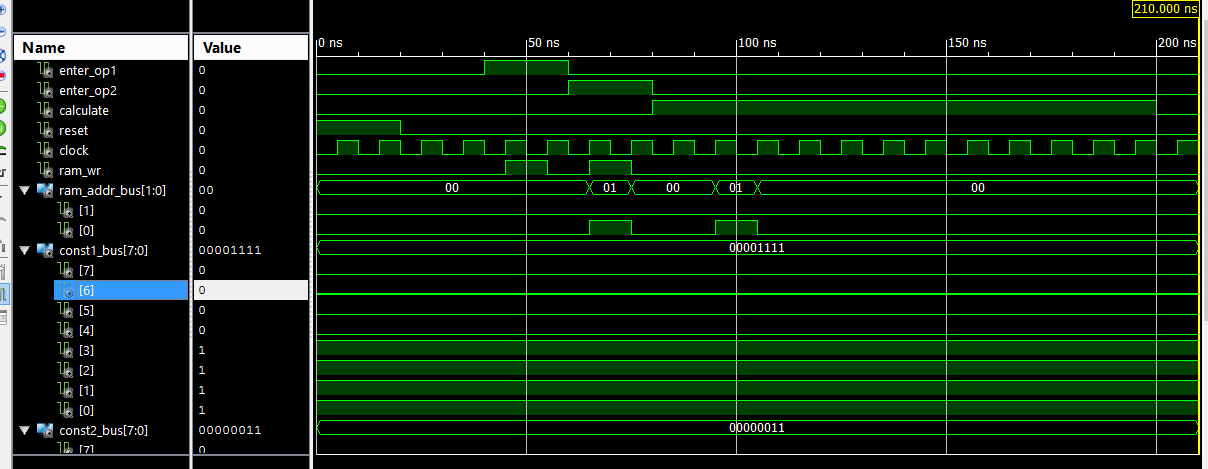
ACC\_WR <= '0';

end case;

end process;

end CU\_arch;

10. Перевірити роботу блока керування за допомогою симулятора ISim.



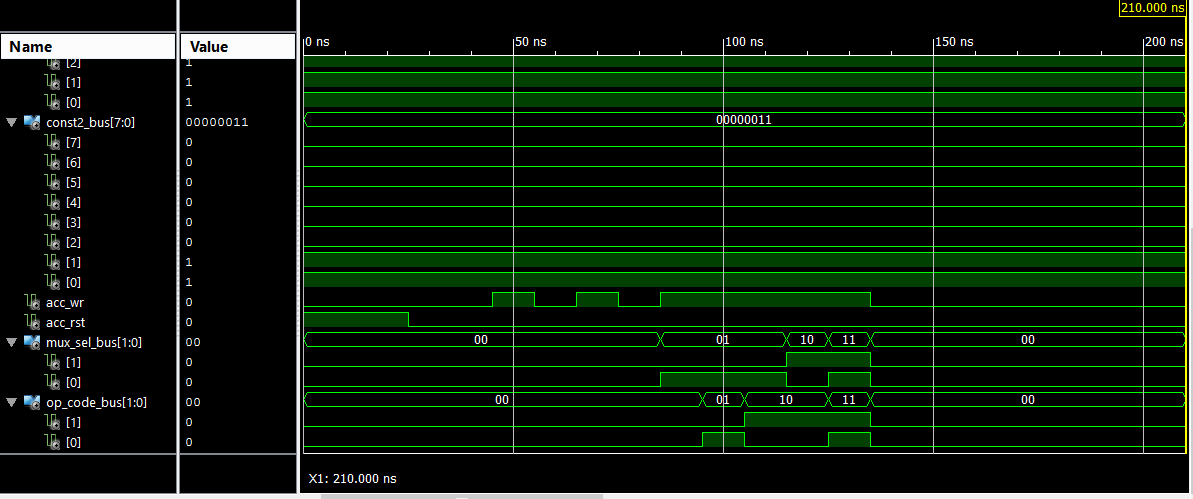


Рис.3.9. Симуляція блоку керування в ISim.

11. Створити новий .vhd файл, та реалізувати на ньому ОЗП RAM.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity RAM is

port( CLOCK : STD\_LOGIC;

WR : IN STD\_LOGIC;

ADDR\_BUS : IN STD\_LOGIC\_VECTOR(1 DOWNTO 0);

IN\_DATA\_BUS : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);

OUT\_DATA\_BUS : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0));

end RAM;

architecture RAM\_arch of RAM is

type ram\_type is array (3 downto 0) of STD\_LOGIC\_VECTOR(7 downto 0);

signal UNIT : ram\_type;

begin

process(CLOCK, ADDR\_BUS, UNIT)

begin

if (rising\_edge(CLOCK)) then

if (WR = '1') then

UNIT(conv\_integer(ADDR\_BUS)) <= IN\_DATA\_BUS;

end if;

end if;

OUT\_DATA\_BUS <= UNIT(conv\_integer(ADDR\_BUS));

end process;

end RAM\_arch;

12. Перевірити роботу блока керування за допомогою симулятора ISim.

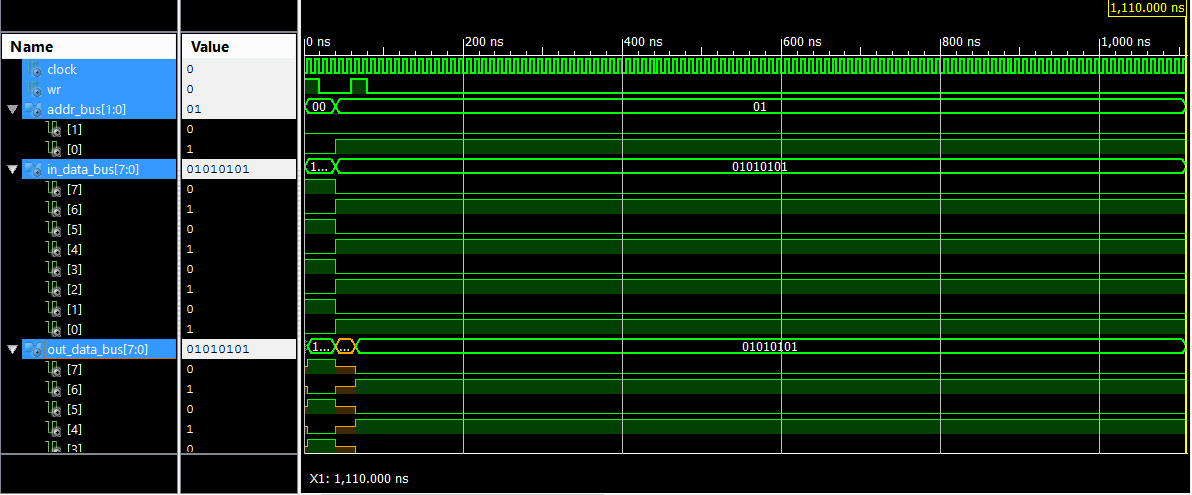


Рис.3.11. Симуляція ОЗП в Isim.

13. Створити новий .vhd файл, та реалізувати на ньому блок індикації 7-SEG DECODER.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity BIN\_TO\_BCD is

port( CLOCK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

ACC\_DATA\_OUT\_BUS : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);

COMM\_ONES : OUT STD\_LOGIC;

COMM\_DECS : OUT STD\_LOGIC;

COMM\_HUNDREDS : OUT STD\_LOGIC;

SEG\_A : OUT STD\_LOGIC;

SEG\_B : OUT STD\_LOGIC;

SEG\_C : OUT STD\_LOGIC;

SEG\_D : OUT STD\_LOGIC;

SEG\_E : OUT STD\_LOGIC;

SEG\_F : OUT STD\_LOGIC;

SEG\_G : OUT STD\_LOGIC;

DP : OUT STD\_LOGIC);

end BIN\_TO\_BCD;

architecture Behavioral of BIN\_TO\_BCD is

signal ONES\_BUS : STD\_LOGIC\_VECTOR(3 downto 0) := "0000";

signal DECS\_BUS : STD\_LOGIC\_VECTOR(3 downto 0) := "0001";

signal HONDREDS\_BUS : STD\_LOGIC\_VECTOR(3 downto 0) := "0000";

begin

BIN\_TO\_BCD : process (ACC\_DATA\_OUT\_BUS)

variable hex\_src : STD\_LOGIC\_VECTOR(7 downto 0) ;

variable bcd : STD\_LOGIC\_VECTOR(11 downto 0) ;

begin

bcd := (others => '0') ;

hex\_src := ACC\_DATA\_OUT\_BUS;

for i in hex\_src'range loop

if bcd(3 downto 0) > "0100" then

bcd(3 downto 0) := bcd(3 downto 0) + "0011" ;

end if ;

if bcd(7 downto 4) > "0100" then

bcd(7 downto 4) := bcd(7 downto 4) + "0011" ;

end if ;

if bcd(11 downto 8) > "0100" then

bcd(11 downto 8) := bcd(11 downto 8) + "0011" ;

end if ;

bcd := bcd(10 downto 0) & hex\_src(hex\_src'left) ; -- shift bcd + 1 new entry

hex\_src := hex\_src(hex\_src'left - 1 downto hex\_src'right) & '0' ; -- shift src + pad with 0

end loop ;

HONDREDS\_BUS <= bcd (11 downto 8);

DECS\_BUS <= bcd (7 downto 4);

ONES\_BUS <= bcd (3 downto 0);

end process BIN\_TO\_BCD;

INDICATE : process(CLOCK)

type DIGIT\_TYPE is (ONES, DECS, HUNDREDS);

variable CUR\_DIGIT : DIGIT\_TYPE := ONES;

variable DIGIT\_VAL : STD\_LOGIC\_VECTOR(3 downto 0) := "0000";

variable DIGIT\_CTRL : STD\_LOGIC\_VECTOR(6 downto 0) := "0000000";

variable COMMONS\_CTRL : STD\_LOGIC\_VECTOR(2 downto 0) := "000";

begin

if (rising\_edge(CLOCK)) then

if(RESET = '0') then

case CUR\_DIGIT is

when ONES =>

DIGIT\_VAL := ONES\_BUS;

CUR\_DIGIT := DECS;

COMMONS\_CTRL := "001";

when DECS =>

DIGIT\_VAL := DECS\_BUS;

CUR\_DIGIT := HUNDREDS;

COMMONS\_CTRL := "010";

when HUNDREDS =>

DIGIT\_VAL := HONDREDS\_BUS;

CUR\_DIGIT := ONES;

COMMONS\_CTRL := "100";

when others =>

DIGIT\_VAL := ONES\_BUS;

CUR\_DIGIT := ONES;

COMMONS\_CTRL := "000";

end case;

case DIGIT\_VAL is --abcdefg

when "0000" => DIGIT\_CTRL := "1111110";

when "0001" => DIGIT\_CTRL := "0110000";

when "0010" => DIGIT\_CTRL := "1101101";

when "0011" => DIGIT\_CTRL := "1111001";

when "0100" => DIGIT\_CTRL := "0110011";

when "0101" => DIGIT\_CTRL := "1011011";

when "0110" => DIGIT\_CTRL := "1011111";

when "0111" => DIGIT\_CTRL := "1110000";

when "1000" => DIGIT\_CTRL := "1111111";

when "1001" => DIGIT\_CTRL := "1111011";

when others => DIGIT\_CTRL := "0000000";

end case;

else

DIGIT\_VAL := ONES\_BUS;

CUR\_DIGIT := ONES;

COMMONS\_CTRL := "000";

end if;

COMM\_ONES <= COMMONS\_CTRL(0);

COMM\_DECS <= COMMONS\_CTRL(1);

COMM\_HUNDREDS <= COMMONS\_CTRL(2);

SEG\_A <= DIGIT\_CTRL(6);

SEG\_B <= DIGIT\_CTRL(5);

SEG\_C <= DIGIT\_CTRL(4);

SEG\_D <= DIGIT\_CTRL(3);

SEG\_E <= DIGIT\_CTRL(2);

SEG\_F <= DIGIT\_CTRL(1);

SEG\_G <= DIGIT\_CTRL(0);

DP <= '0';

end if;

end process INDICATE;

end Behavioral;

14. Перевірити роботу блока індикації за допомогою симулятора ISim.

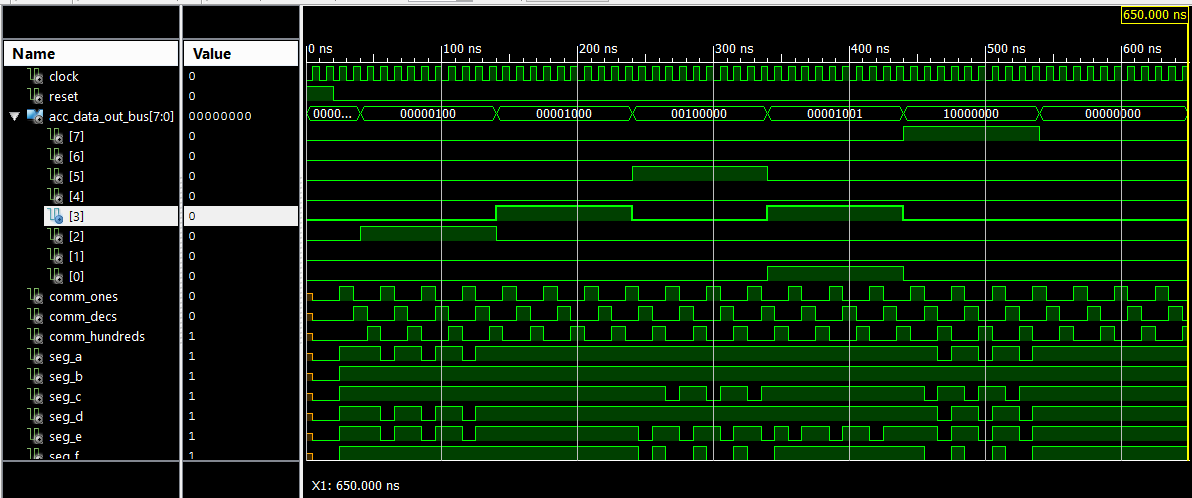


Рис.3.12. Симуляція блоку індикації в Isim.

15-16. Згенерувати символи імплементованих компонентів. Створити файл верхнього рівня та виконати інтеграцію компонентів системи між собою та зі стендом.

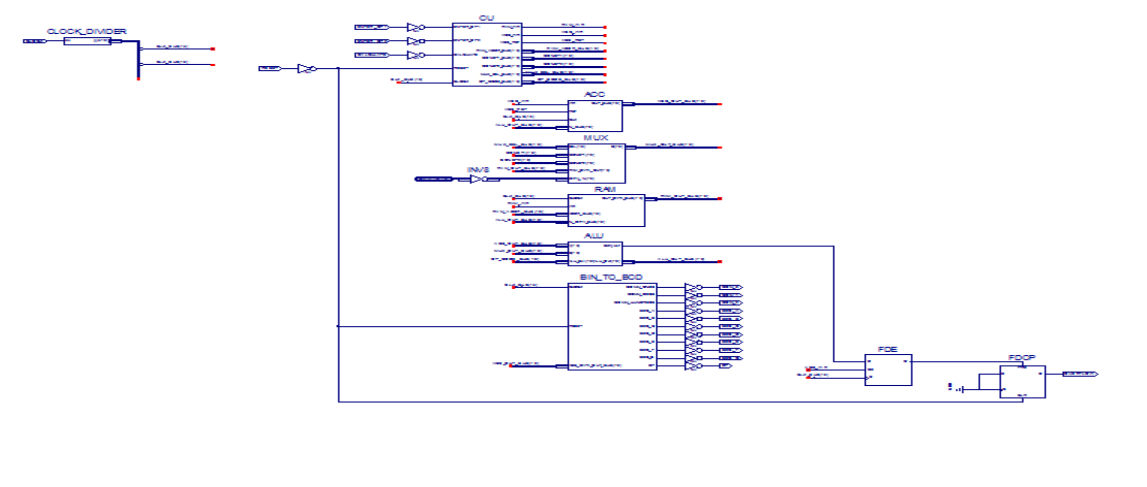


Рис.3.13. Зінтегровані між собою компоненти.

17. Перевірити роботу схеми в симуляторі ISim.

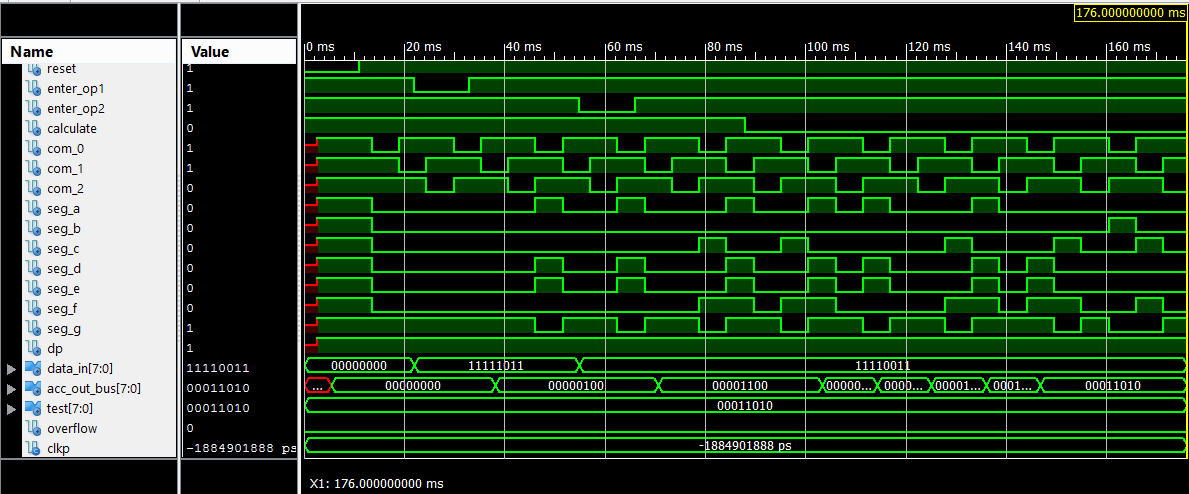


Рис.3.14. Симуляція виконання обчислень.

**Висновок:**

В ході виконання цієї лабораторної роботи я реалізував на базі стенда Elbert V2 – Spartan3A FPGA цифровий автомат згідно заданих вимог.