Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра ЕОМ



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*»

Виконав: ст. гр. КІ-201

Олеш С.Б.

Прийняв:

Козак Н.Б.

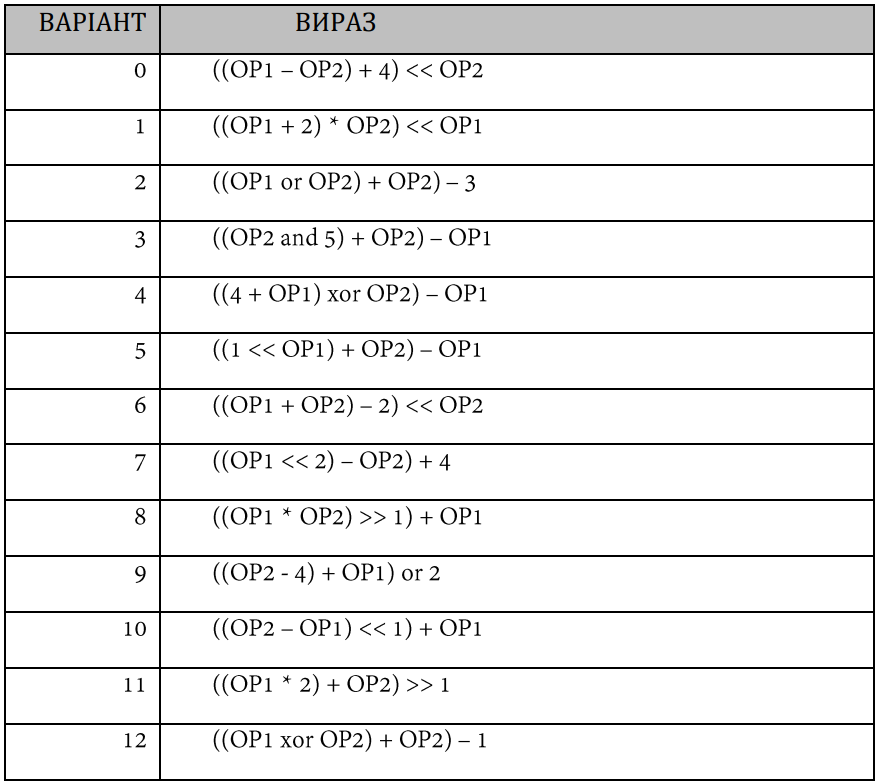
Львів – 2023

**Мета:** на базі стенда *Elbert V2- Spartan 3A FPGA* реалізувати цифровий автомат для обчислення значення виразу дотримуючись наступних вимог:

1. Функціонал пристрою повинен бути реалізований згідно отриманого варіанту завдання Дивись розділ ЗАВДАННЯ.
2. Пристрій повинен бути ітераційним (АЛП (ALU) повинен виконувати за один такт одну операцію), та реалізованим згідно наступної структурної схеми.
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 Schematic символ.
5. Інтеграція структурних блоків в єдину систему та зі стендом *Elbert V2- Spartan 3A FPGA* повинна бути виконана за допомогою *ISE WebPACKTM Schematic Capture*.
6. Кожен структурний блок і схема в цілому повинні бути промодельовані за допомогою симулятора *ISim*.
7. Формування вхідних даних на шині DATA\_IN повинно бути реалізовано за допомогою DIP перемикачів (елемент P7 на стенді *Elbert V2- Spartan 3A FPGA* Див. Додаток - 2 (інформація про стенд)):
   1. P7[8] - наймолодший розряд значення операндa.
   2. P7[1] - найстарший розряд значення операндa.
8. Керування пристроєм повинно бути реалізовано за допомогою PUSH BUTTON кнопок (елементи SW1, SW2, SW3, SW6 на стенді *Elbert V2- Spartan 3A FPGA* Див. Додаток - 2 (інформація про стенд)):
   1. SW1 - запис першого операнду в пам’ять даних автомата.
   2. SW2 - запис другого операнду в пам’ять даних автомата.
   3. SW3 - запуск процесу обчислення.
   4. SW6 - скидання автомата у початковий стан.
9. Індикація значень операндів при вводі, та вивід результату обчислень повинні бути реалізовані за допомогою семи сегментних індикаторів S1 – S3. Індикація переповнення в АЛП - за допомогою LED D8 на стенді *Elbert V2- Spartan 3A FPGA* Див. Додаток - 2 (інформація про стенд).
10. Підготувати і захистити звіт.

**Завдання:**

**Варіант 3**



**Хід роботи:**

1. Створив VHDL мультиплексор.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity my\_MuX\_intf is

port(

DATA\_IN : in std\_logic\_vector(7 downto 0);

IN\_SEL : in std\_logic\_vector(1 downto 0);

CONSTANT\_BUS : in std\_logic\_vector(7 downto 0);

RAM\_DATA\_OUT\_BUS : in std\_logic\_vector(7 downto 0);

IN\_SEL\_OUT\_BUS : out std\_logic\_vector(7 downto 0)

);

end my\_MuX\_intf;

architecture my\_MuX\_arch of my\_MuX\_intf is

begin

INSEL\_A\_MUX : process(DATA\_IN, CONSTANT\_BUS, RAM\_DATA\_OUT\_BUS, IN\_SEL)

begin

if(IN\_SEL = "00") then

IN\_SEL\_OUT\_BUS <= DATA\_IN;

elsif(IN\_SEL = "01") then

IN\_SEL\_OUT\_BUS <= RAM\_DATA\_OUT\_BUS;

else

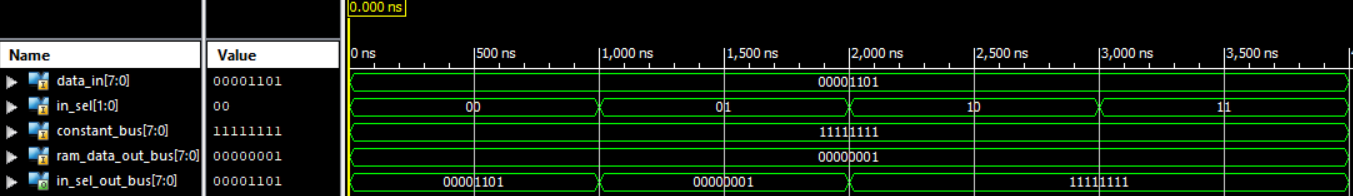
IN\_SEL\_OUT\_BUS <= CONSTANT\_BUS;

end if;

end process INSEL\_A\_MUX;

end my\_MuX\_arch;

1. Симуляція роботи мультиплексора.

****

1. Створив VHDL файл який реалізує регістр АСС.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity my\_ACC\_intf is

port(

CLOCK : in std\_logic;

ACC\_WR : in std\_logic;

ACC\_RST : in std\_logic;

ACC\_DATA\_IN\_BUS : in std\_logic\_vector(7 downto 0);

ACC\_DATA\_OUT\_BUS : out std\_logic\_vector(7 downto 0)

);

end my\_ACC\_intf;

architecture my\_ACC\_arch of my\_ACC\_intf is

signal ACC\_DATA : std\_logic\_vector(7 downto 0);

begin

ACC : process(CLOCK, ACC\_DATA)

begin

if (rising\_edge(CLOCK)) then

if(ACC\_RST = '1') then

ACC\_DATA <= "00000000";

elsif (ACC\_WR = '1') then

ACC\_DATA <= ACC\_DATA\_IN\_BUS;

end if;

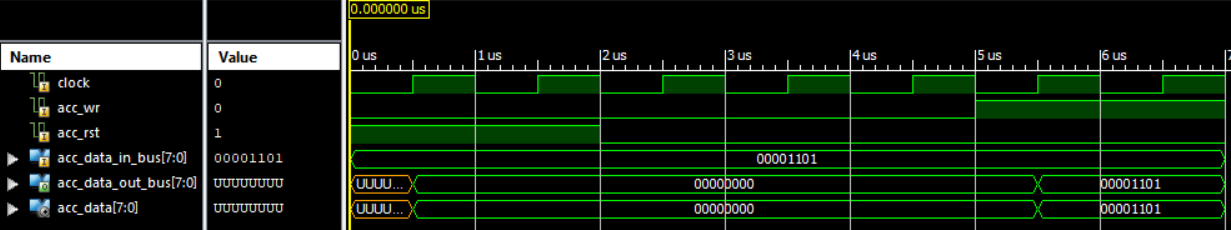
end if;

ACC\_DATA\_OUT\_BUS <= ACC\_DATA;

end process ACC;

end my\_ACC\_arch;

1. Симуляція роботи регістра.

****

1. Створив VHDL файл який реалізує АLU.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity my\_ALU\_intf is

port(

OP\_CODE\_BUS : in std\_logic\_vector(1 downto 0);

IN\_SEL\_OUT\_BUS : in std\_logic\_vector(7 downto 0);

ACC\_DATA\_OUT\_BUS : in std\_logic\_vector(7 downto 0);

ACC\_DATA\_IN\_BUS : out std\_logic\_vector(7 downto 0)

);

end my\_ALU\_intf;

architecture my\_ALU\_arch of my\_ALU\_intf is

begin

ALU : process(OP\_CODE\_BUS, IN\_SEL\_OUT\_BUS, ACC\_DATA\_OUT\_BUS)

variable A : unsigned(7 downto 0);

variable B : unsigned(7 downto 0);

begin

A := unsigned(ACC\_DATA\_OUT\_BUS);

B := unsigned(IN\_SEL\_OUT\_BUS);

case(OP\_CODE\_BUS) is

when "00" => ACC\_DATA\_IN\_BUS <= STD\_LOGIC\_VECTOR(B);

when "01" => ACC\_DATA\_IN\_BUS <= STD\_LOGIC\_VECTOR(A + B);

when "10" => ACC\_DATA\_IN\_BUS <= STD\_LOGIC\_VECTOR(A - B);

when "11" => ACC\_DATA\_IN\_BUS <= STD\_LOGIC\_VECTOR(B and "00000101");

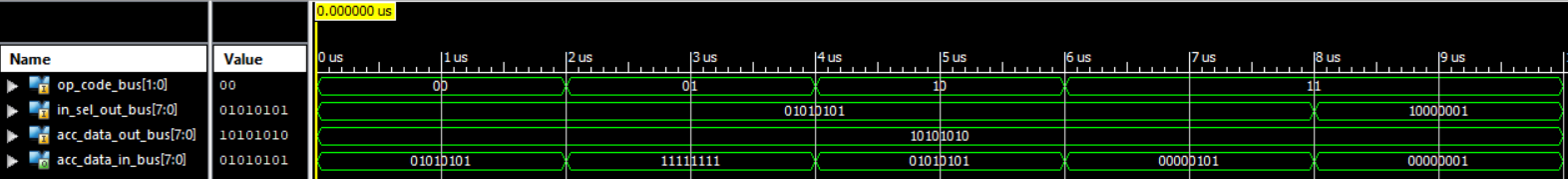
when others => ACC\_DATA\_IN\_BUS <= "00000000";

end case;

end process ALU;

end my\_ALU\_arch;

1. Симуляція роботи ALU.

****

1. Створив VHDL файл який реалізує CU.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity my\_CU\_intf is

port(

CLOCK : in std\_logic;

ENTER\_OP1 : in std\_logic;

ENTER\_OP2 : in std\_logic;

CALCULATE : in std\_logic;

RESET : in std\_logic;

RAM\_WR : out std\_logic;

RAM\_ADDR\_BUS : out std\_logic\_vector(1 downto 0);

CONSTANT\_BUS : out std\_logic\_vector(7 downto 0);

ACC\_WR : out std\_logic;

ACC\_RST : out std\_logic;

IN\_SEL : out std\_logic\_vector(1 downto 0);

OP\_CODE\_BUS : out std\_logic\_vector(1 downto 0)

);

end my\_CU\_intf;

architecture my\_CU\_arch of my\_CU\_intf is

type cu\_state\_type is (cu\_rst, cu\_idle, cu\_load\_op1, cu\_load\_op2, cu\_run\_calc0, cu\_run\_calc1, cu\_run\_calc2, cu\_run\_calc3, cu\_finish);

signal cu\_cur\_state : cu\_state\_type;

signal cu\_next\_state : cu\_state\_type;

begin

CONSTANT\_BUS <= "00000001";

CU\_SYNC\_PROC: process (CLOCK)

begin

if (rising\_edge(CLOCK)) then

if (RESET = '1') then

cu\_cur\_state <= cu\_rst;

else

cu\_cur\_state <= cu\_next\_state;

end if;

end if;

end process;

CUNEXT\_STATE\_DECODE: process (cu\_cur\_state, ENTER\_OP1, ENTER\_OP2, CALCULATE)

begin

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

cu\_next\_state <= cu\_cur\_state; --default is to stay in current state

--insert statements to decode next\_state

--below is a simple example

case(cu\_cur\_state) is

when cu\_rst =>

cu\_next\_state <= cu\_idle;

when cu\_idle =>

if (ENTER\_OP1 = '1') then

cu\_next\_state <= cu\_load\_op1;

elsif (ENTER\_OP2 = '1') then

cu\_next\_state <= cu\_load\_op2;

elsif (CALCULATE = '1') then

cu\_next\_state <= cu\_run\_calc0;

else

cu\_next\_state <= cu\_idle;

end if;

when cu\_load\_op1 =>

cu\_next\_state <= cu\_idle;

when cu\_load\_op2 =>

cu\_next\_state <= cu\_idle;

when cu\_run\_calc0 =>

cu\_next\_state <= cu\_run\_calc1;

when cu\_run\_calc1 =>

cu\_next\_state <= cu\_run\_calc2;

when cu\_run\_calc2 =>

cu\_next\_state <= cu\_run\_calc3;

when cu\_run\_calc3 =>

cu\_next\_state <= cu\_finish;

when cu\_finish =>

cu\_next\_state <= cu\_finish;

when others =>

cu\_next\_state <= cu\_idle;

end case;

end process;

CU\_OUTPUT\_DECODE: process (cu\_cur\_state)

begin

case(cu\_cur\_state) is

when cu\_rst =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '1';

ACC\_WR <= '0';

when cu\_idle =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_load\_op1 =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '1';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_load\_op2 =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '1';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_run\_calc0 =>

IN\_SEL <= "10";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_run\_calc1 =>

IN\_SEL <= "01";

OP\_CODE\_BUS <= "11";

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_run\_calc2 =>

IN\_SEL <= "01";

OP\_CODE\_BUS <= "01";

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_run\_calc3 =>

IN\_SEL <= "01";

OP\_CODE\_BUS <= "10";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_finish =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when others =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "00";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

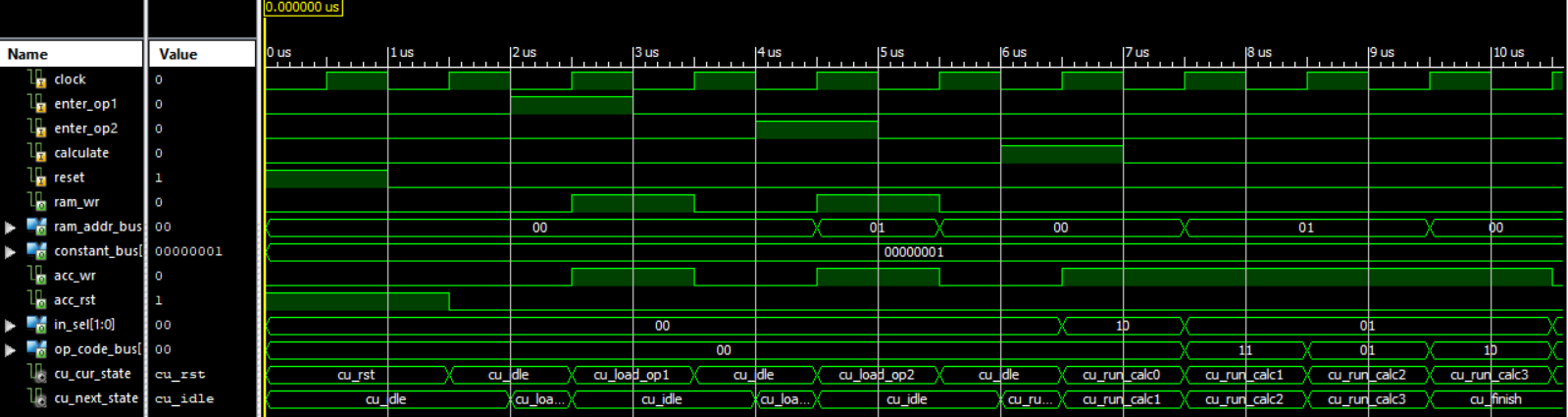
ACC\_WR <= '0';

end case;

end process;

end my\_CU\_arch;

1. Симуляція роботи керуючого автомата.



1. Створив VHDL файл який реалізує RAM.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity my\_RAM\_intf is

port(

CLOCK : in std\_logic;

RAM\_WR : in std\_logic;

RAM\_ADDR\_BUS : in STD\_LOGIC\_VECTOR(1 downto 0);

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

RAM\_DATA\_OUT\_BUS : out STD\_LOGIC\_VECTOR(7 downto 0)

);

end my\_RAM\_intf;

architecture my\_RAM\_arch of my\_RAM\_intf is

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

signal RAM\_UNIT : ram\_type;

begin

--when reset will init const

RAM : process(CLOCK, RAM\_ADDR\_BUS, RAM\_UNIT)

begin

if (rising\_edge(CLOCK)) then

if (RAM\_WR = '1') then

RAM\_UNIT(conv\_integer(RAM\_ADDR\_BUS)) <= RAM\_DATA\_IN\_BUS;

end if;

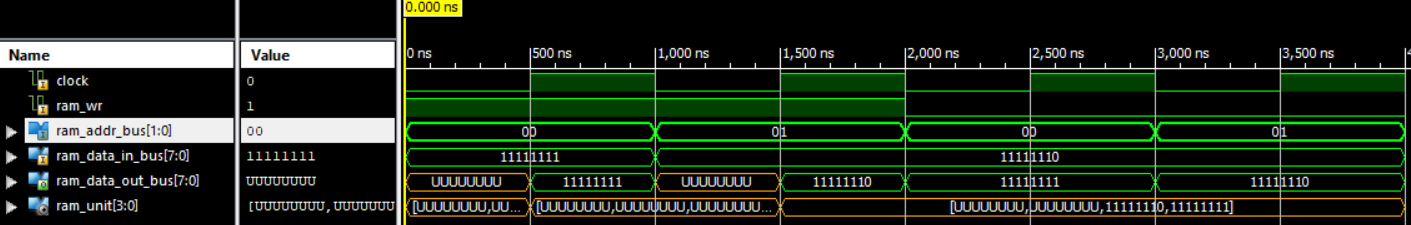
end if;

RAM\_DATA\_OUT\_BUS <= RAM\_UNIT(conv\_integer(RAM\_ADDR\_BUS));

end process RAM;

end my\_RAM\_arch;

1. Симуляція роботи RAM.

****

1. Створив VHDL файл який реалізує Sev\_Seg\_Decoder.

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity OUT\_PUT\_DECODER\_intf 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 OUT\_PUT\_DECODER\_intf;

architecture OUT\_PUT\_DECODER\_arch of OUT\_PUT\_DECODER\_intf 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);

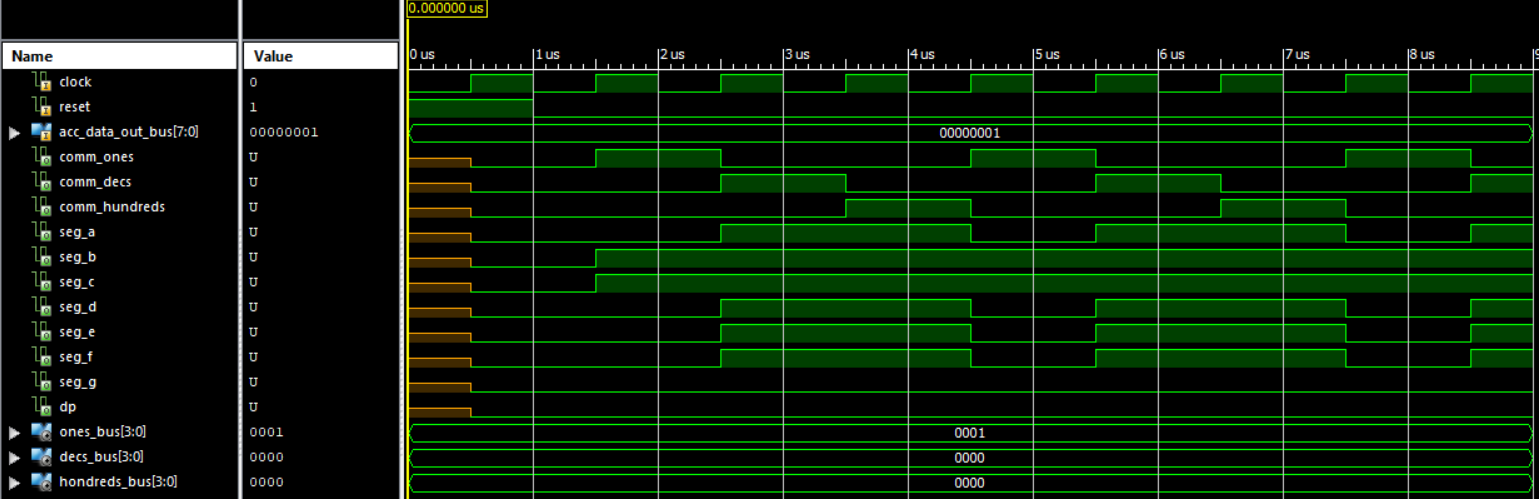
DP <= '0';

end if;

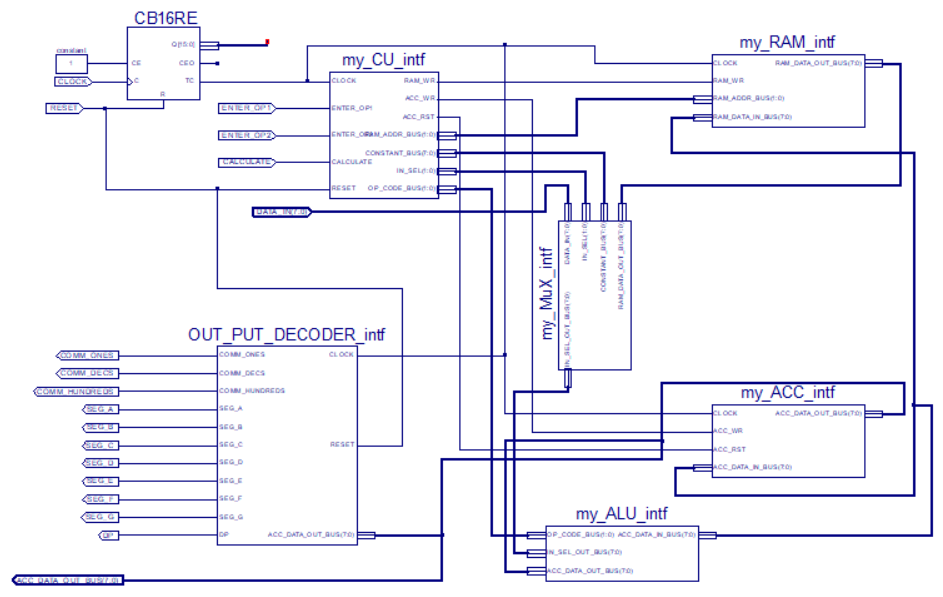
end process INDICATE;

end OUT\_PUT\_DECODER\_arch;

1. Симуляція роботи декодера.



1. Склав схему.

****

1. Симуляція роботи схеми.

((OP2 and 5) + OP2) – OP1

При ОР1 = 0000 0100, ОР2 = 0000 1011;

0000 1011 and 0000 0101 = 0000 0001;

0000 0001 + 0000 1011= 0000 1100;

0000 1100 – 0000 0100 = 0000 1000.

****

**Висновок:** під час виконання цієї лабораторної роботи я реалізував цифровий автомат для обчислення значення виразу у середовищі Xilinx ISE і стендом Elbert V2 - Spartan 3A FPGA. Я реалізував схему автомату та провів симуляцію його роботи.