**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**



**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

Заяць А. Я.

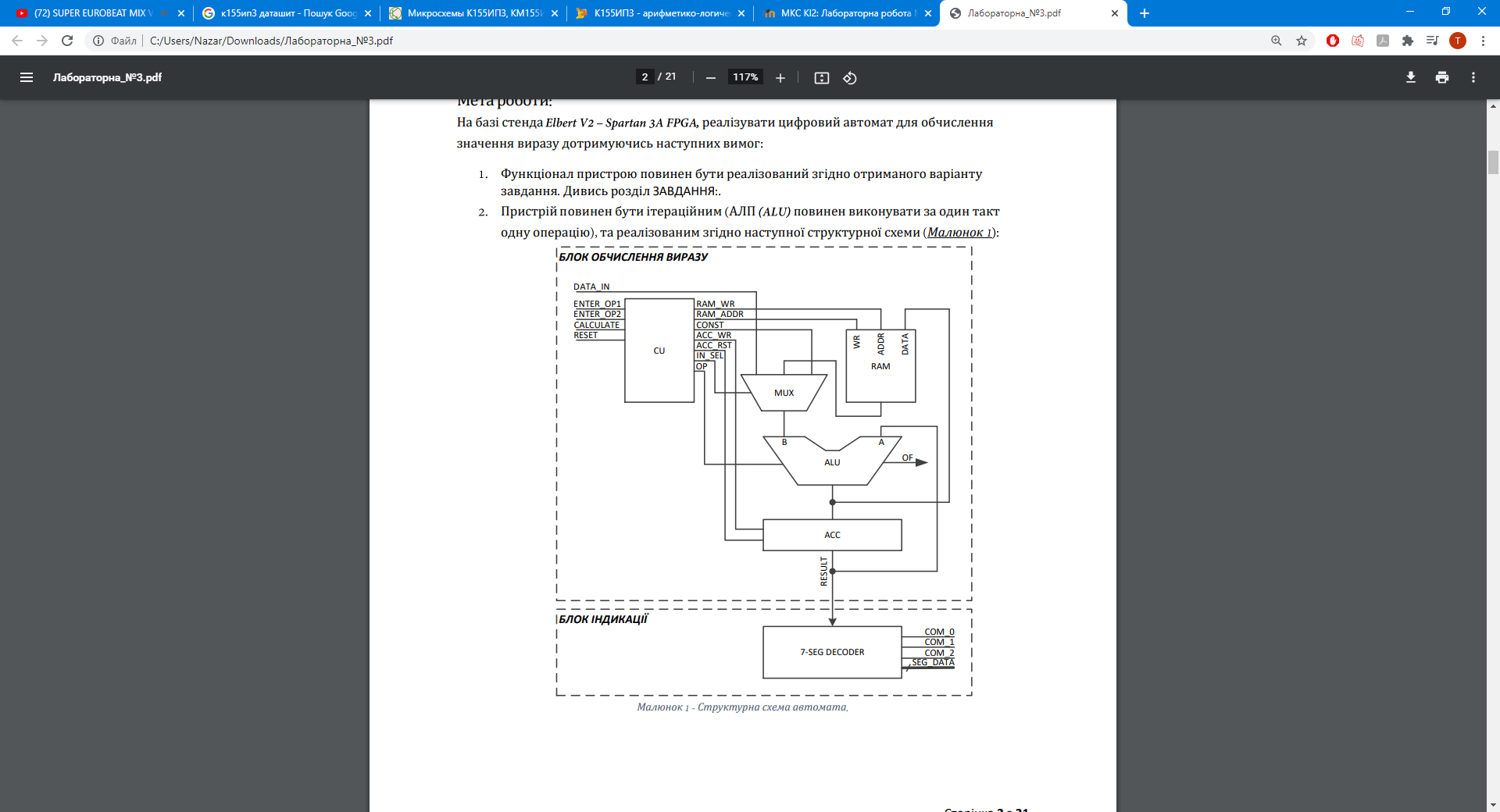
Прийняв:

Козак Н. Б.

Львів – 2023

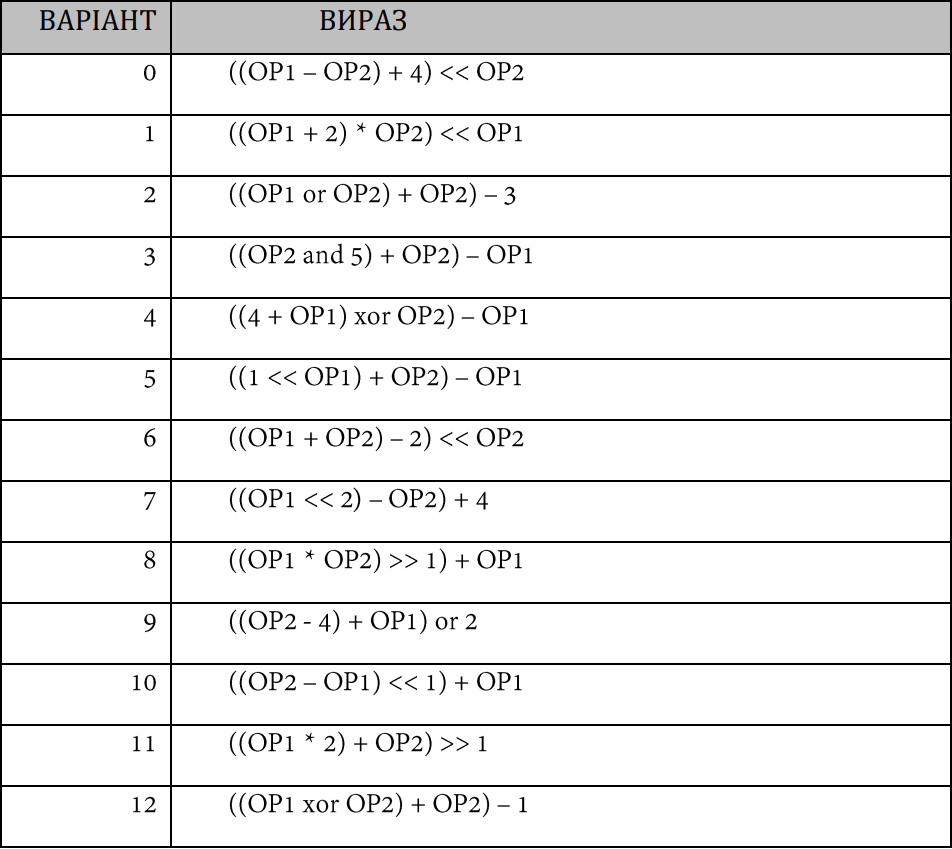
**Мета роботи:** На базі стенда Elbert V2- Spartan 3A FPGA реалізувати цифровий автомат для обчислення значення виразу дотримуючись наступних вимог:

1. Функціонал пристрою повинен бути реалізований згідно отриманого варіанту завдання Дивись розділ ЗАВДАННЯ
2. Пристрій повинен бути ітераційним АЛП повинен виконувати за один такт одну операцію та реалізованим згідно наступної структурної схеми



1. Кожен блок структурної схеми повинен бути реалізований на мові в окремому файлі Дозволено використовувати всі оператори
2. Для кожного блока структурної схеми повинен бути згенерований символ
3. Інтеграція структурних блоків в єдину систему та зі стендом повинна бути виконана за допомогою
4. Кожен структурний блок і схема вцілому повинні бути промодельовані за допомогою симулятора
5. Формування вхідних даних на шині повинно бути реалізовано за допомогою перемикачів елемент на стенді Див Додаток інформація про стенд наймолодший розряд значення операнда найстарший розряд значення операнда
6. Керування пристроєм повинно бути реалізовано за допомогою кнопок елементи на стенді Див Додаток інформація про стенд запис першого операнда в пам ять даних автомата запис другого операнда в пам ять даних автомата запуск процесу обчислення скидання автомата у початковий стан
7. Індикація значень операндів при вводі та вивід результату обчислень повинні бути реалізовані за допомогою семи сегментних індикаторів Індикація переповнення в АЛП за допомогою на стенді Див Додаток інформація про стенд
8. Підготувати і захистити звіт

**ЗАВДАННЯ**

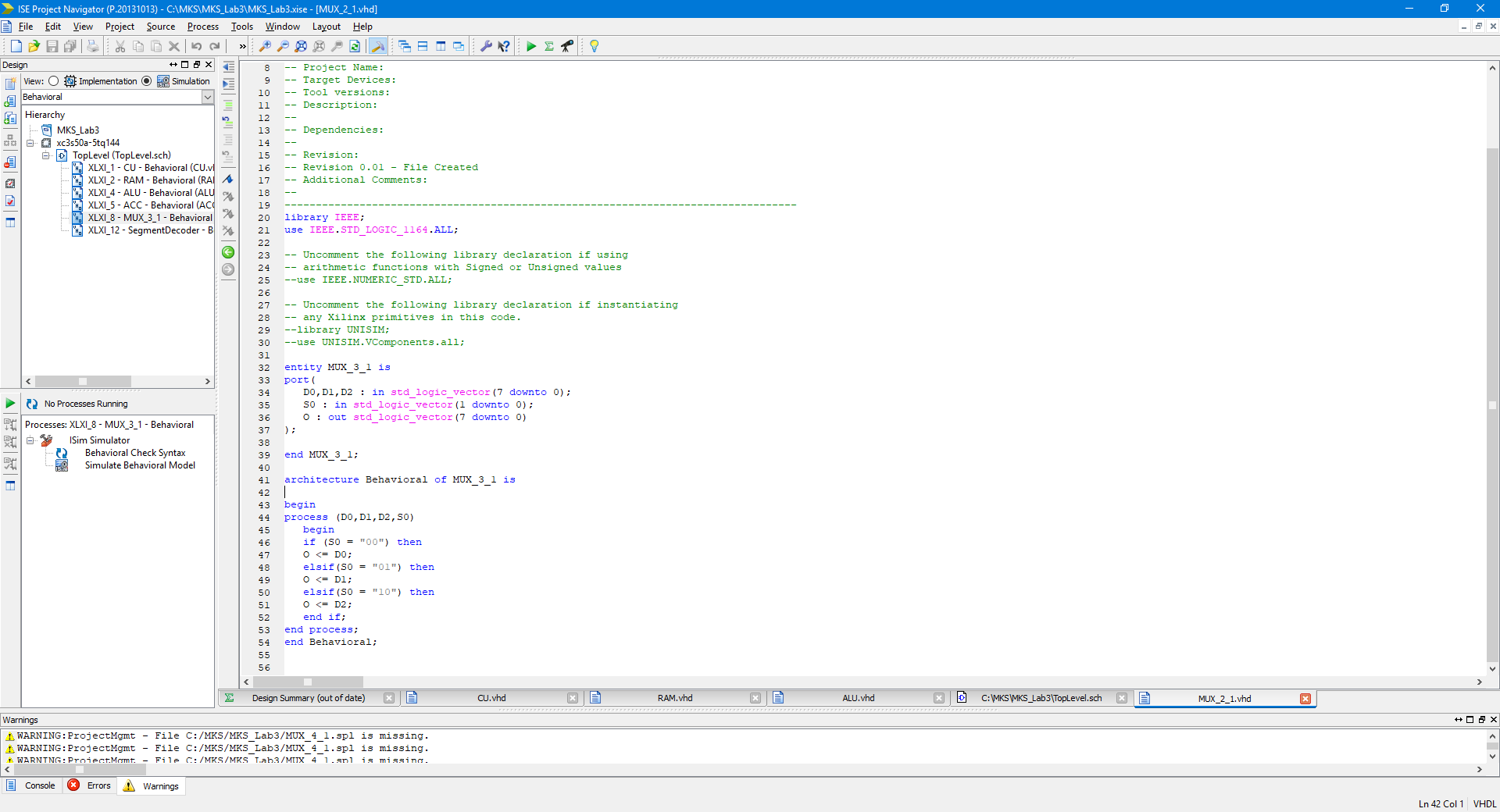


Мій варіант – 8.

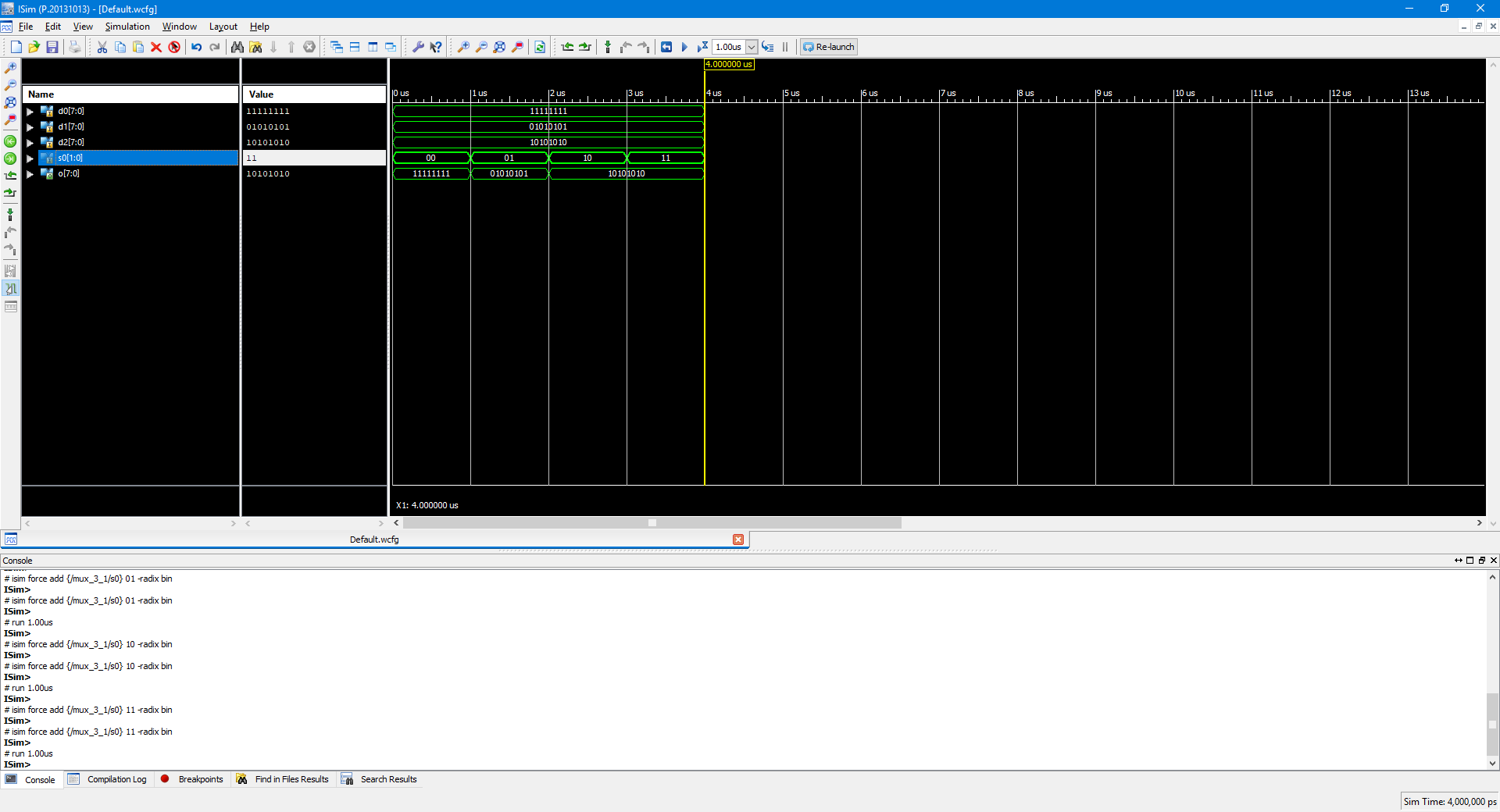
Мій вираз : ((OP1\*ОР2)>>1)+OP1.

**Виконання роботи:**

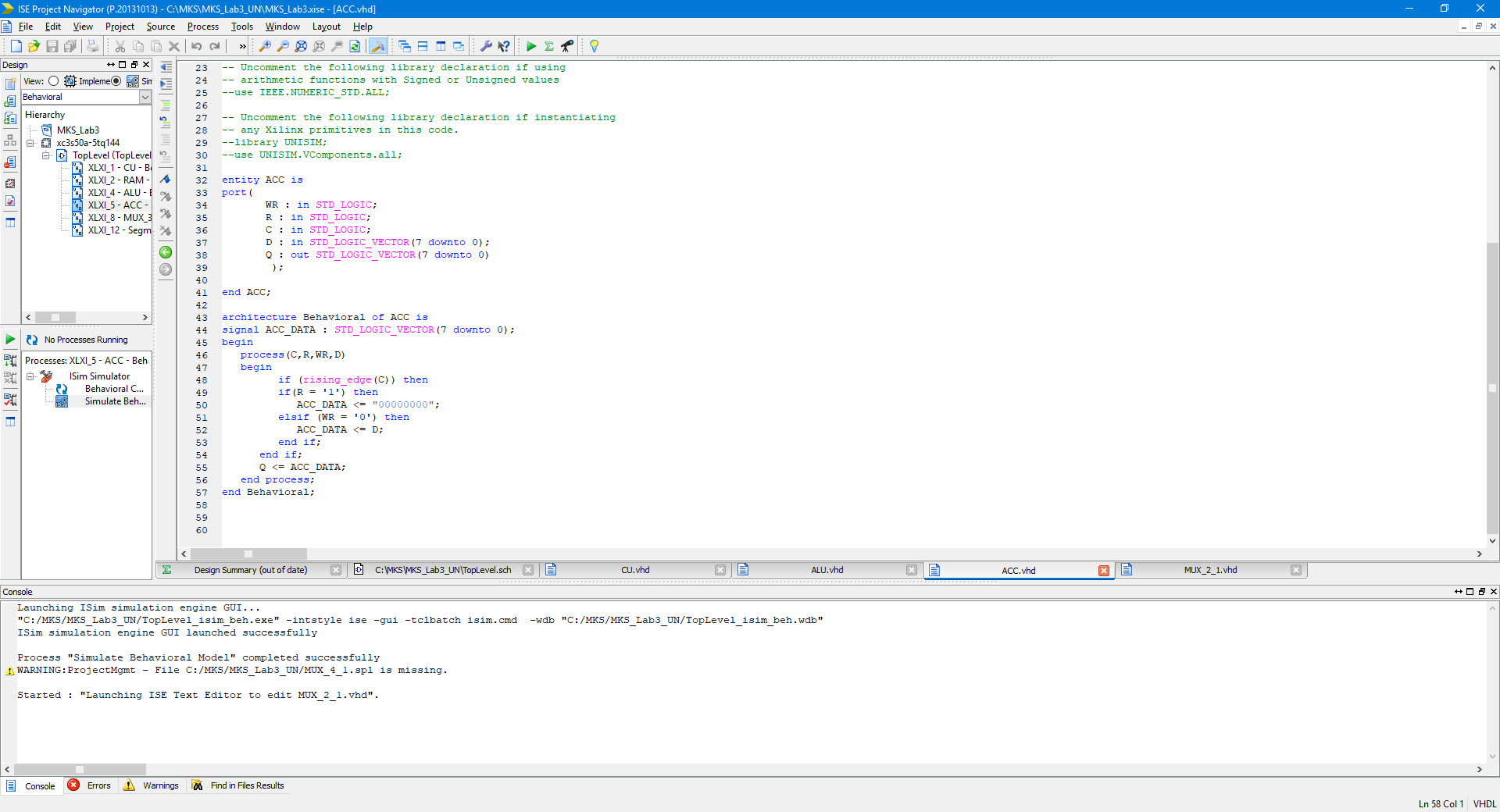
1. Створив VHDL мультиплексор



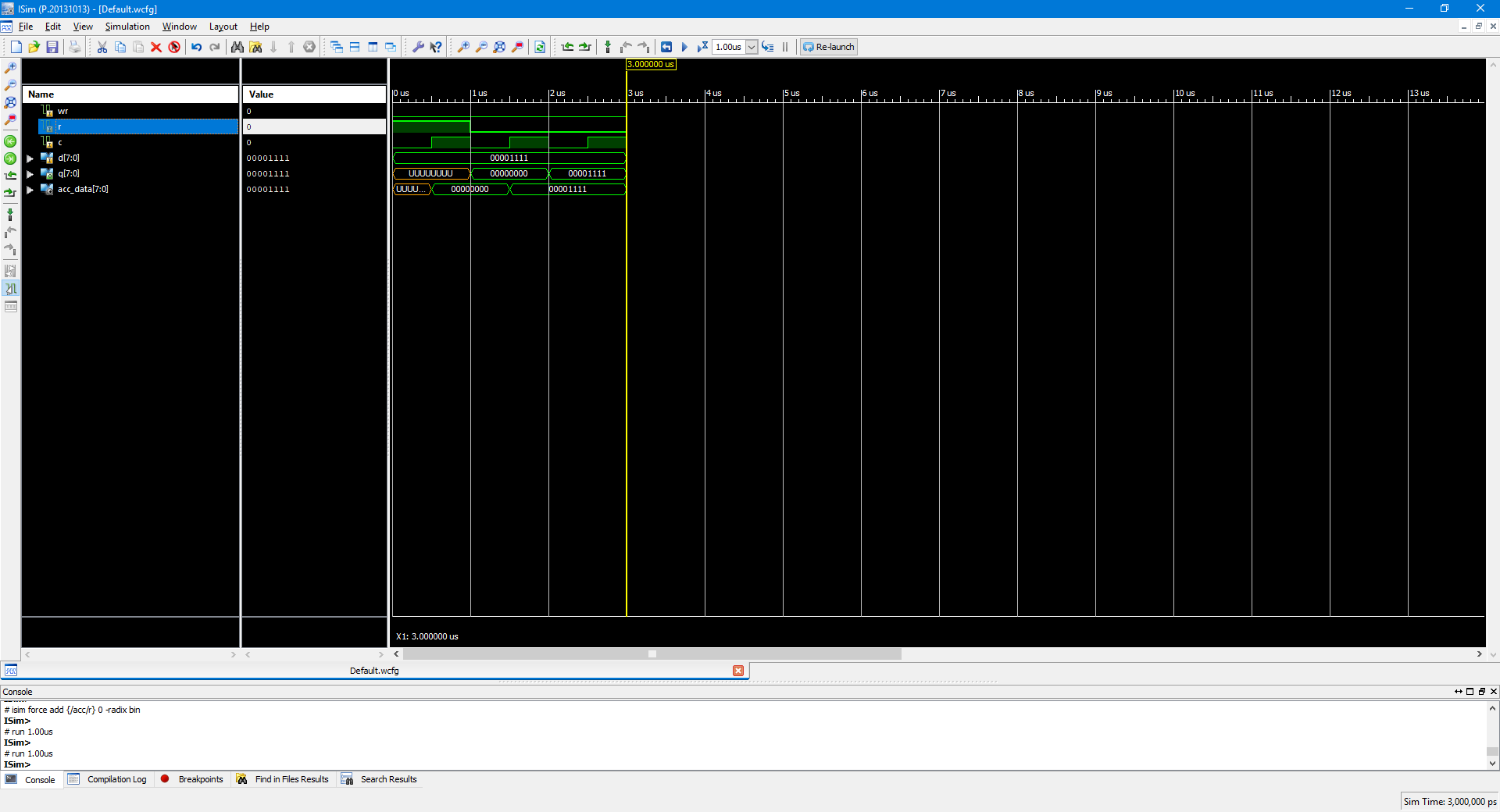
1. Промоделював роботу з усіма можливими наборами сигналів



1. Створив VHDL файл який реалізує регістр АСС



1. Промоделював роботу схеми



1. Створив VHDL файл який реалізує АLU

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

use ieee.std\_logic\_arith.all;

use IEEE.NUMERIC\_STD.ALL;

entity ALU is

port(

B : in STD\_LOGIC\_vector (7 downto 0);

A : in STD\_LOGIC\_vector (7 downto 0);

ALU\_SEL : in STD\_LOGIC\_vector (2 downto 0);

CARRYOUT : out STD\_LOGIC := '0';

ALU\_OUT : out STD\_LOGIC\_vector (7 downto 0)

);

end ALU;

architecture Behavioral of ALU is

signal ALU\_Result : std\_logic\_vector (7 downto 0);

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

begin

process(A, B, ALU\_SEL)

variable ALU\_Mul\_Result : std\_logic\_vector (15 downto 0);

begin

case ALU\_SEL is

when "000" => --add

ALU\_Result <= STD\_LOGIC\_VECTOR(A+B);

tmp <= ('0' & A) + ('0' & B);

when "001" => -- Subs

ALU\_Result <= A - B ;

when "010" => --B\*A

ALU\_Mul\_Result := A \* B;

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

if (ALU\_Mul\_Result(15 downto 8) > 0) then

tmp(8) <= '1';

else

tmp(8) <= '0';

end if;

when "011" => --A<<1

ALU\_Result(7 downto 1)<=A(6 downto 0);

ALU\_Result(0)<='0';

tmp <= A & '0';

when "100" => --B>>1

ALU\_Result(6 downto 0)<=B(7 downto 1);

ALU\_Result(7)<='0';

tmp <= B & '0';

when "101" => --A>>1

ALU\_Result(6 downto 0)<=A(7 downto 1);

ALU\_Result(7)<='0';

tmp <= A & '0';

when "110" => -- A

ALU\_Result <=A;

when "111" => -- B

ALU\_Result <=B;

when others => ALU\_Result <= A+B;

end case;

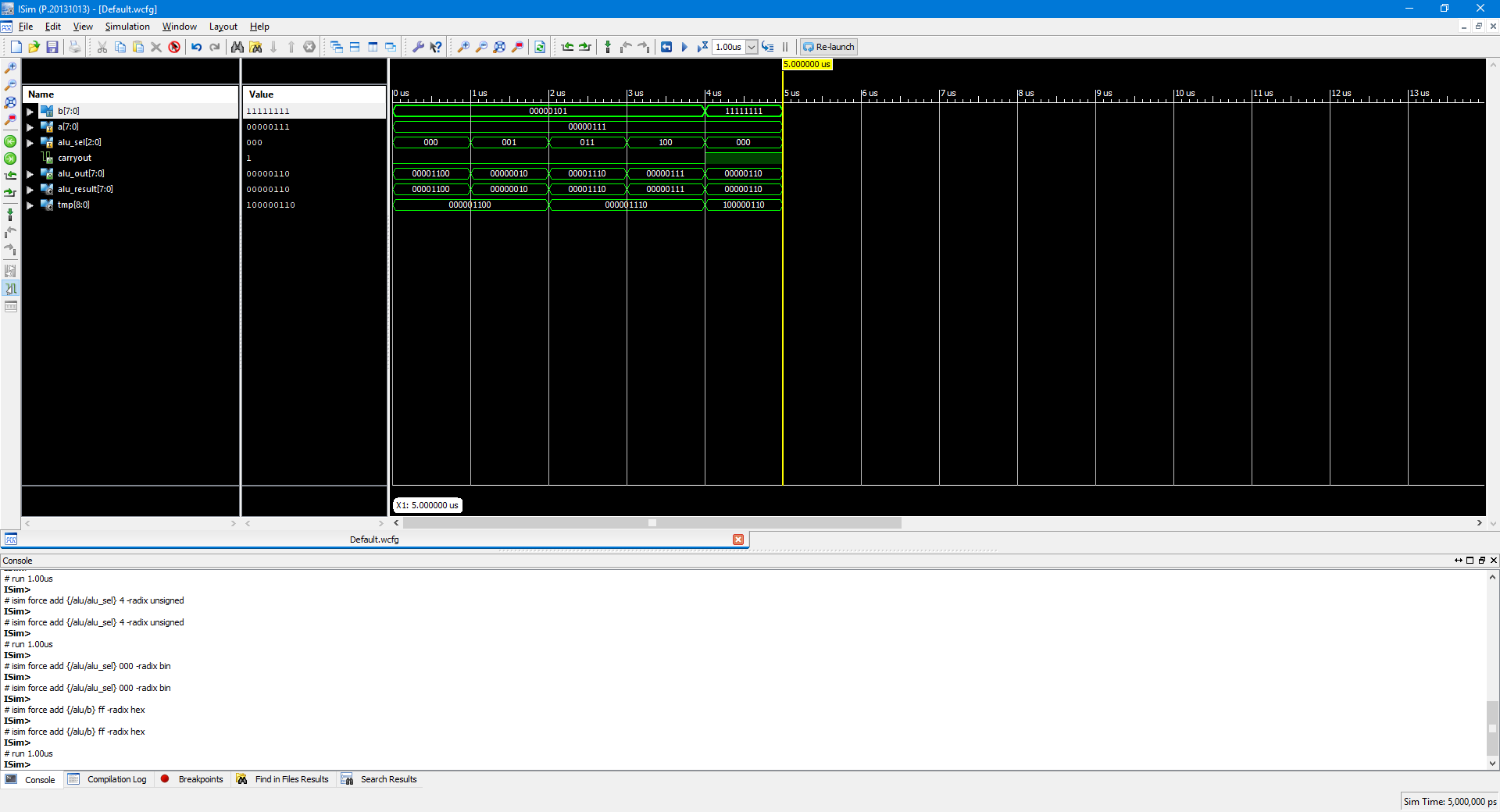
end process;

ALU\_Out <= ALU\_Result; -- ALU out

Carryout <= tmp(8); -- Carryout flag

end Behavioral;

1. Промоделював роботу схеми



1. Створив VHDL файл який реалізує CU

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CU is

port(

CLOCK : IN STD\_LOGIC;

RESET : IN STD\_LOGIC;

ENTER\_OP1 : IN STD\_LOGIC;

ENTER\_OP2 : IN STD\_LOGIC;

CALCULATE : IN STD\_LOGIC;

RAM\_WR : OUT STD\_LOGIC;

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

CONST : 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(2 downto 0)

);

end CU;

architecture Behavioral of CU 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

CU\_SYNC\_PROC: process (CLOCK)

Begin

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 <= "110";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '1';

ACC\_WR <= '0';

when cu\_idle =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "111";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '1';

when cu\_load\_op1 =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "111";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '1';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_load\_op2 =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "111";

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '1';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_run\_calc0 =>

IN\_SEL <= "01"; --load OP1

OP\_CODE\_BUS <= "111"; --B

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_run\_calc1 =>

IN\_SEL <= "01";

OP\_CODE\_BUS <= "010"; --A\*B

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_run\_calc2 =>

IN\_SEL <= "01"; --

OP\_CODE\_BUS <= "101"; --A>>1

RAM\_ADDR\_BUS <= "01";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_run\_calc3 =>

IN\_SEL <= "01"; --

OP\_CODE\_BUS <= "000"; --A+B

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when cu\_finish =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "110";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

when others =>

IN\_SEL <= "00";

OP\_CODE\_BUS <= "000";

RAM\_ADDR\_BUS <= "00";

RAM\_WR <= '0';

ACC\_RST <= '0';

ACC\_WR <= '0';

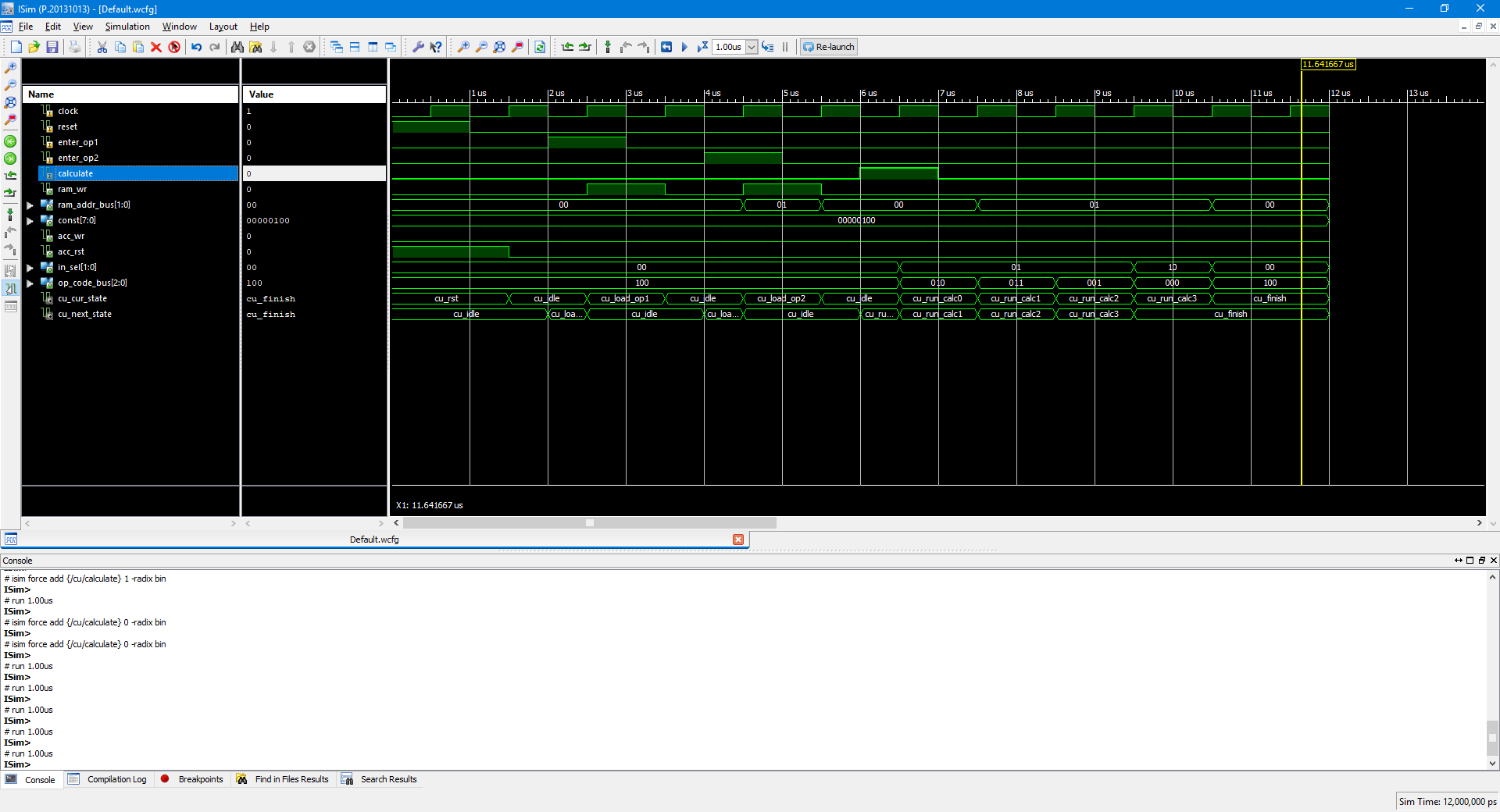
end case;

end process;

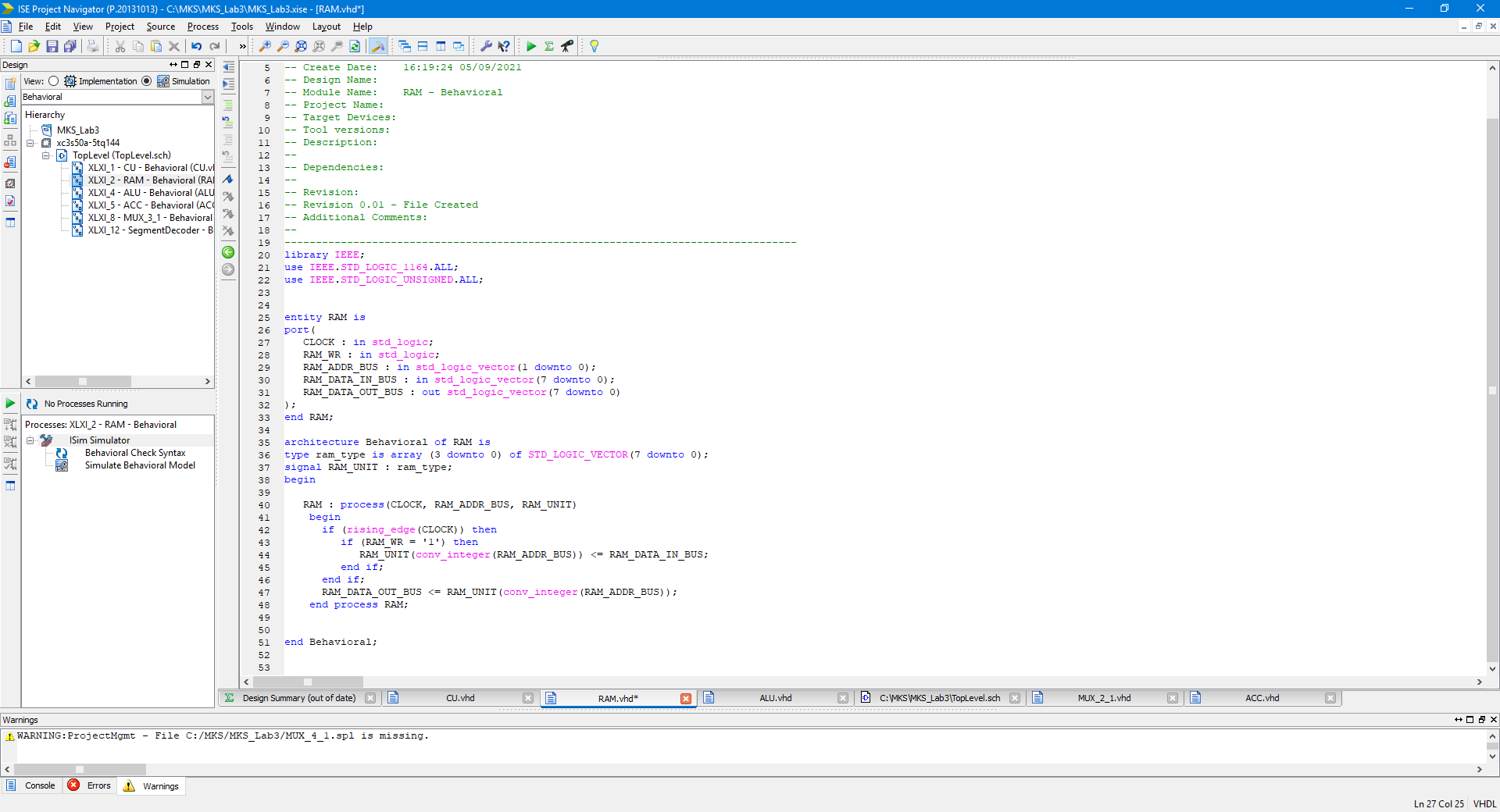
CONST <= "00000100";

end Behavioral;

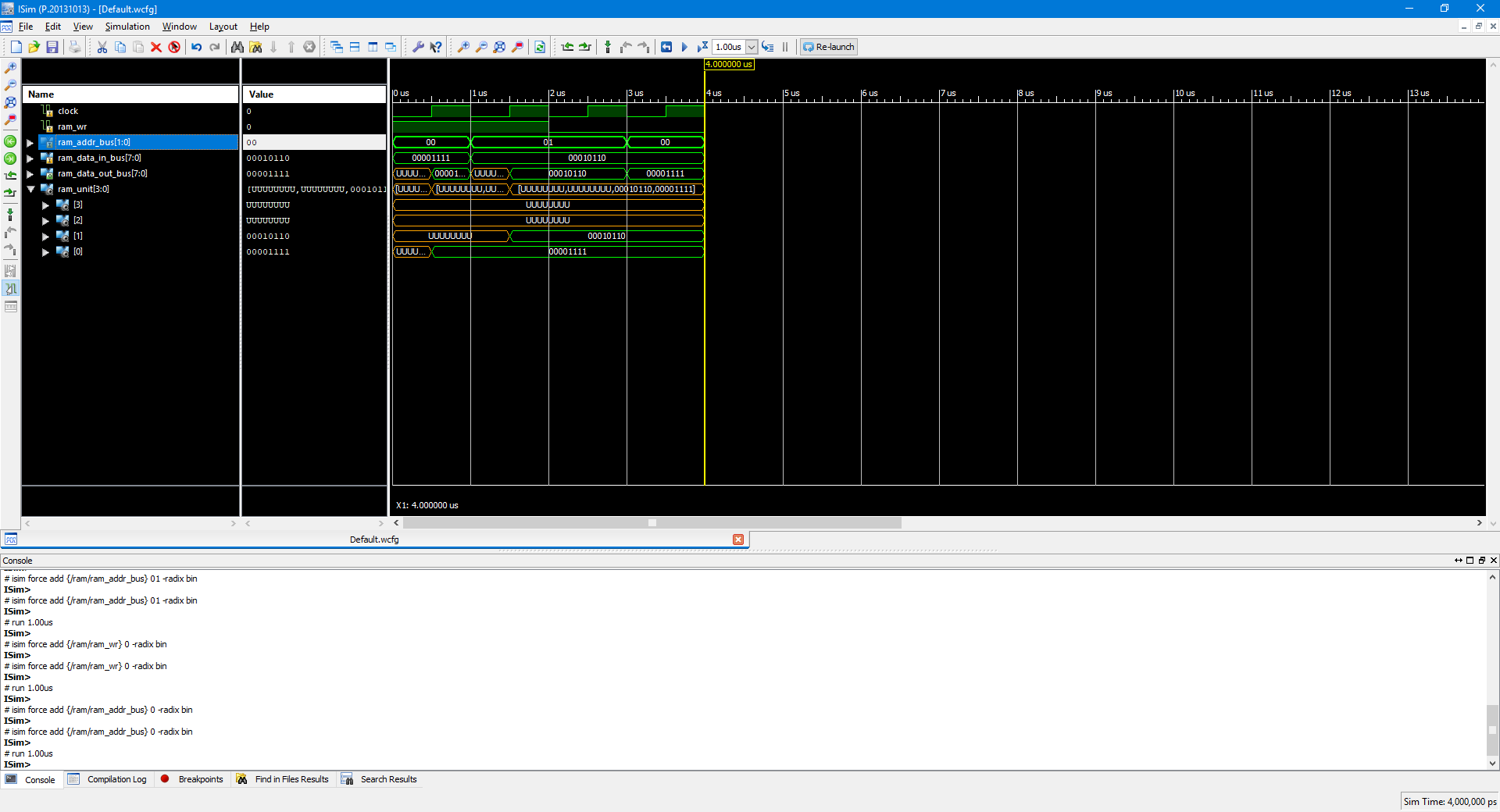
1. Просимулював роботу



1. Створив VHDL файл який реалізує RAM



1. Просимулював роботу



1. Створив VHDL файл який реалізує SegmentDecoder

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity SegmentDecoder is

port (

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

CLOCK : in std\_logic;

RESET : IN STD\_LOGIC;

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 SegmentDecoder;

architecture Behavioral of SegmentDecoder 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 (DATA\_IN)

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

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

begin

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

hex\_src := DATA\_IN;

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 := "1111110";

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);

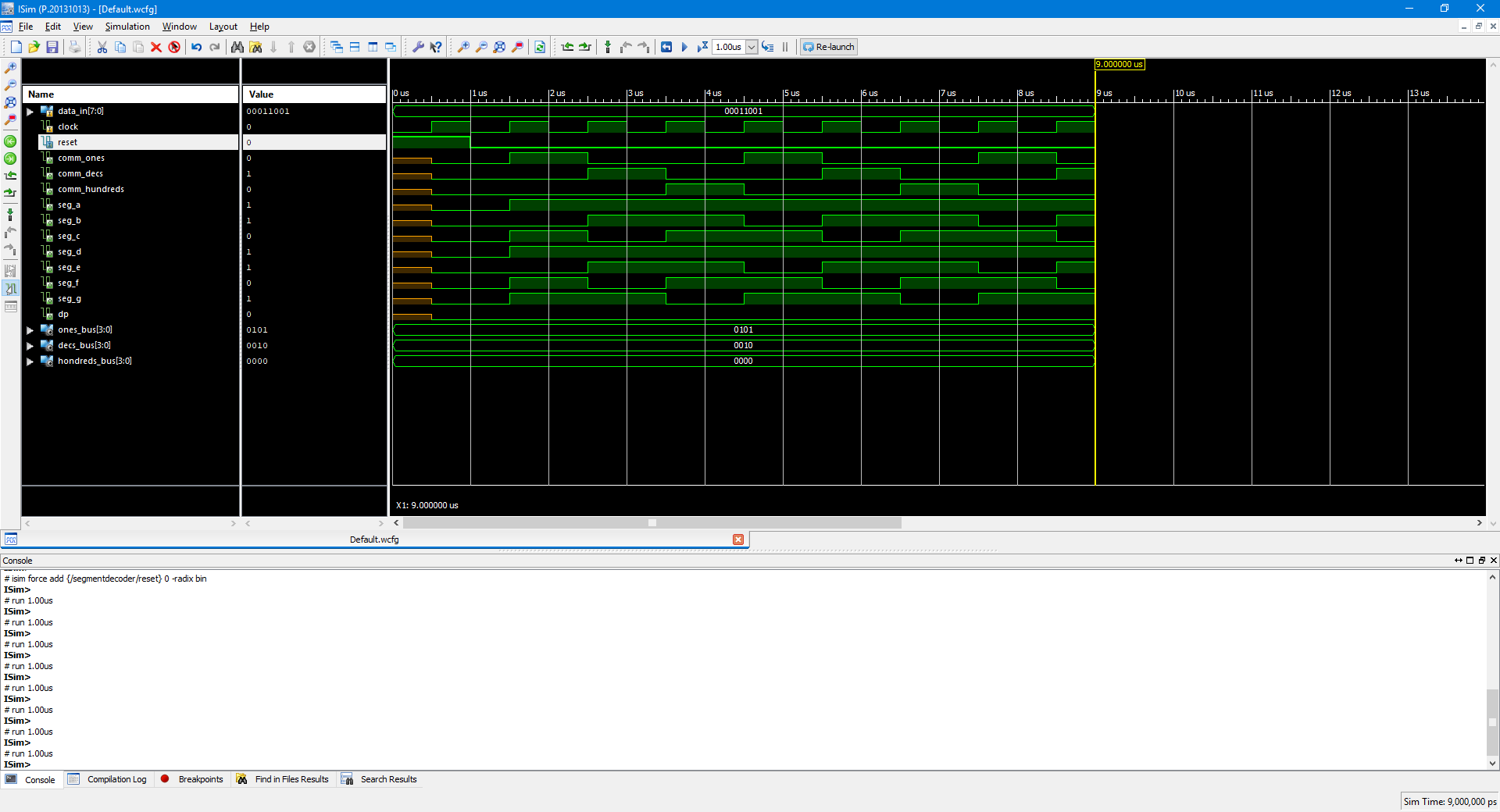
DP <= '0';

end if;

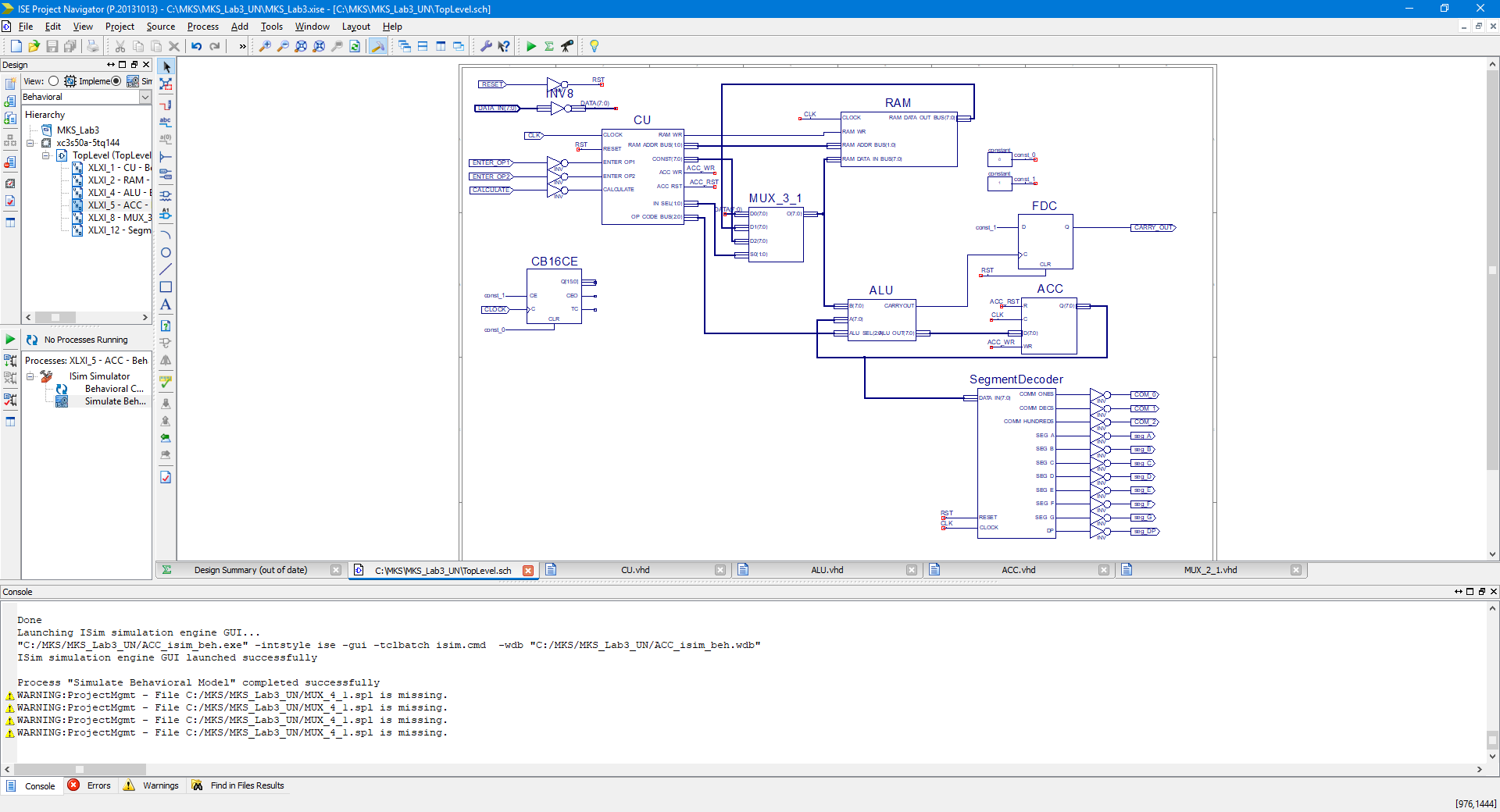
end process INDICATE;

end Behavioral;

1. Просимулював роботу



1. Склав схему



1. Просимулював роботу

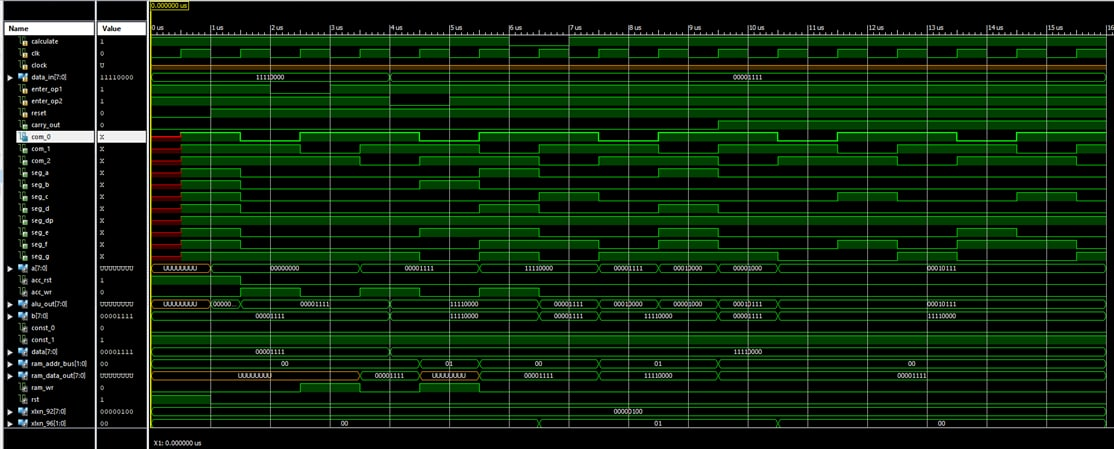
((OP1\* ОР2)>>1)+OP1

При ОР1 = 0000 1111, ОР2 = 1111 0000

0000 1111 \* 1111 0000 = 1110 0001 0000

0001 0000 >> 1 = 0000 1000

0000 1000 + 0000 1111 = 0001 0111



**Висновок:** На даній лабораторній роботі я на базі стенда Elbert V2-Spartan 3A FPGA реалізував цифровий автомат для обчислення значення виразу.