Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Звіт

про виконання 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 №4

Виконав:

ст. гр. КІ-202

Гавриляк Д.В.

Прийняв:

Козак Н. Б.

Львів – 2022

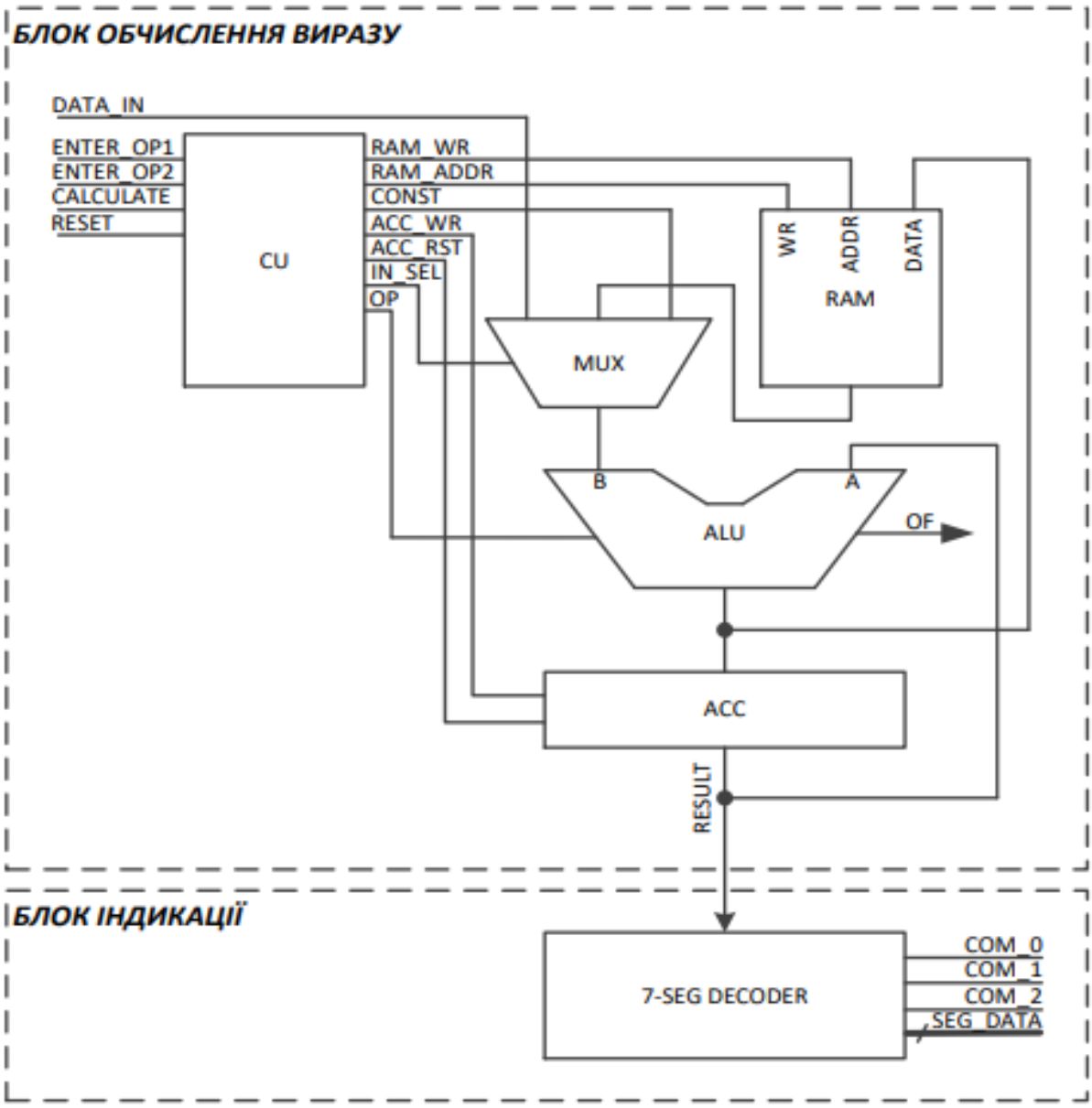
Мета: на базі стенда реалізувати цифровий автомат для обчислення виразу.

Завдання:

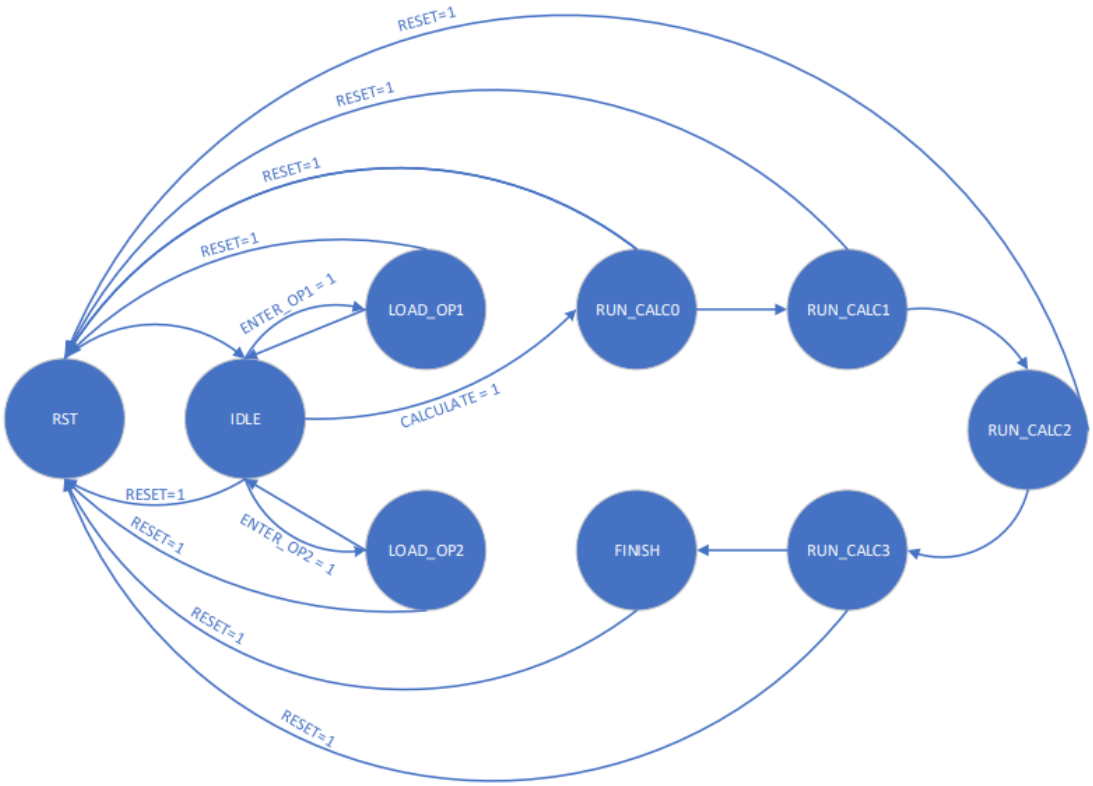
1) На базі стенда реалізувати цифровий автомат для обчислення значення виразу дотримуючись наступних вимог:

ВАРІАНТ	ВИРАЗ
4	((4 + OP1) xor OP2) – OP1

2) Пристрій повинен бути ітераційним АЛП повинен виконувати за один такт одну операцію та реалізованим згідно наступної структурної схеми:



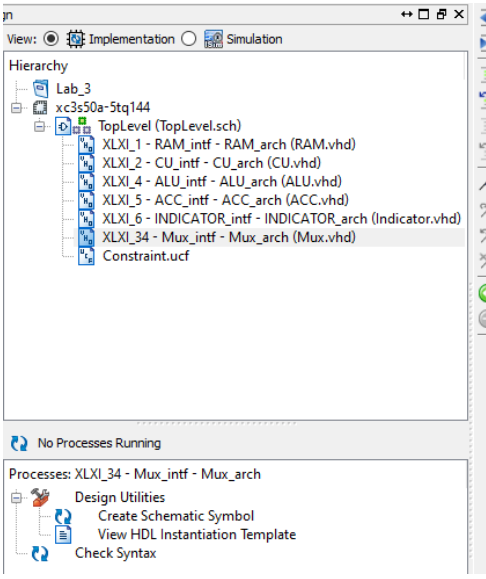
Малюнок 1 - Структурна схема автомата.



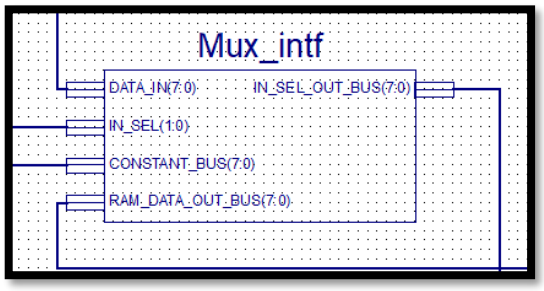
Малюнок 2 - граф станів блока керування.

Виконання роботи:

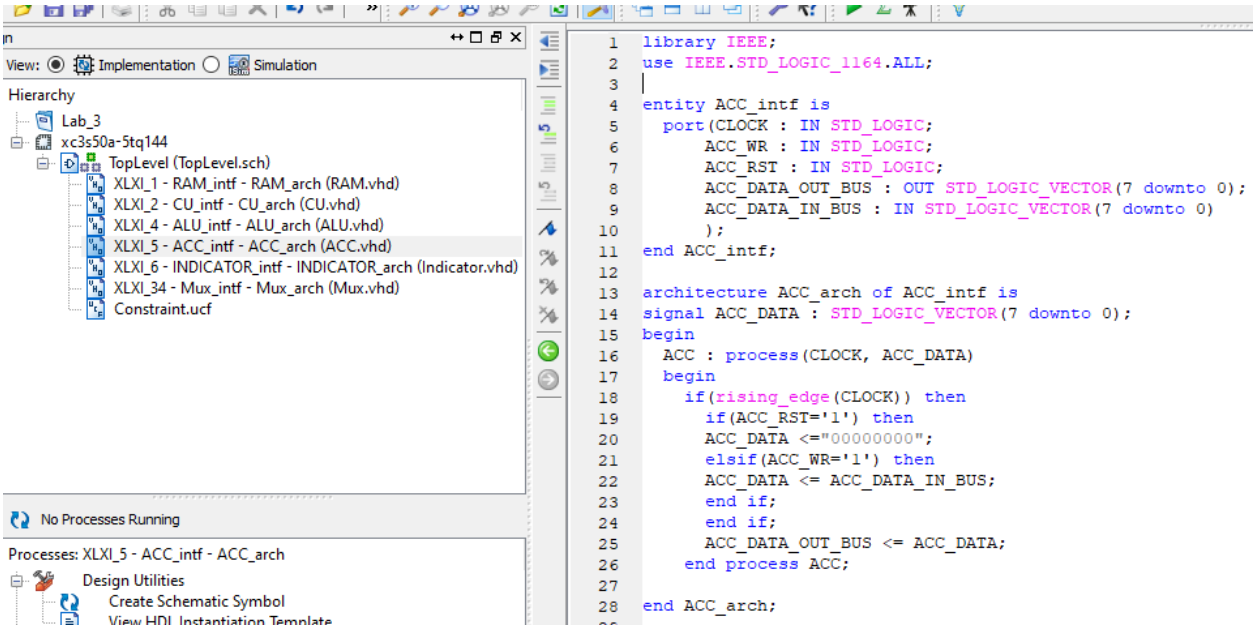
1) Створюю новий файл **Mux.vhd** в якому реалізовую мультиплексор:

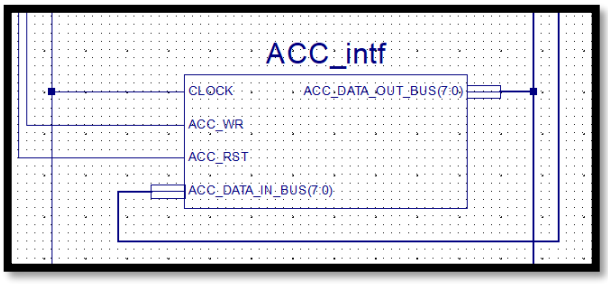



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity Mux_intf is
6  port (DATA_IN          : IN STD_LOGIC_VECTOR (7 downto 0);
7      IN_SEL           : IN STD_LOGIC_VECTOR (1 downto 0);
8      CONSTANT_BUS     : IN STD_LOGIC_VECTOR (7 downto 0);
9      RAM_DATA_OUT_BUS   : IN STD_LOGIC_VECTOR (7 downto 0);
10     IN_SEL_OUT_BUS   : OUT STD_LOGIC_VECTOR (7 downto 0)
11     );
12 end Mux_intf;
13
14 architecture Mux_arch of Mux_intf is
15
16 begin
17
18 INSEL_A_MUX : process (DATA_IN, CONSTANT_BUS, RAM_DATA_OUT_BUS, IN_SEL)
19     begin
20       if(IN_SEL = "00") then
21         IN_SEL_OUT_BUS <= DATA_IN;
22       elsif (IN_SEL = "01") then
23         IN_SEL_OUT_BUS <= RAM_DATA_OUT_BUS;
24       else
25         IN_SEL_OUT_BUS <= CONSTANT_BUS;
26       end if;
27  end process INSEL_A_MUX;
28
29 end Mux_arch;
30
```

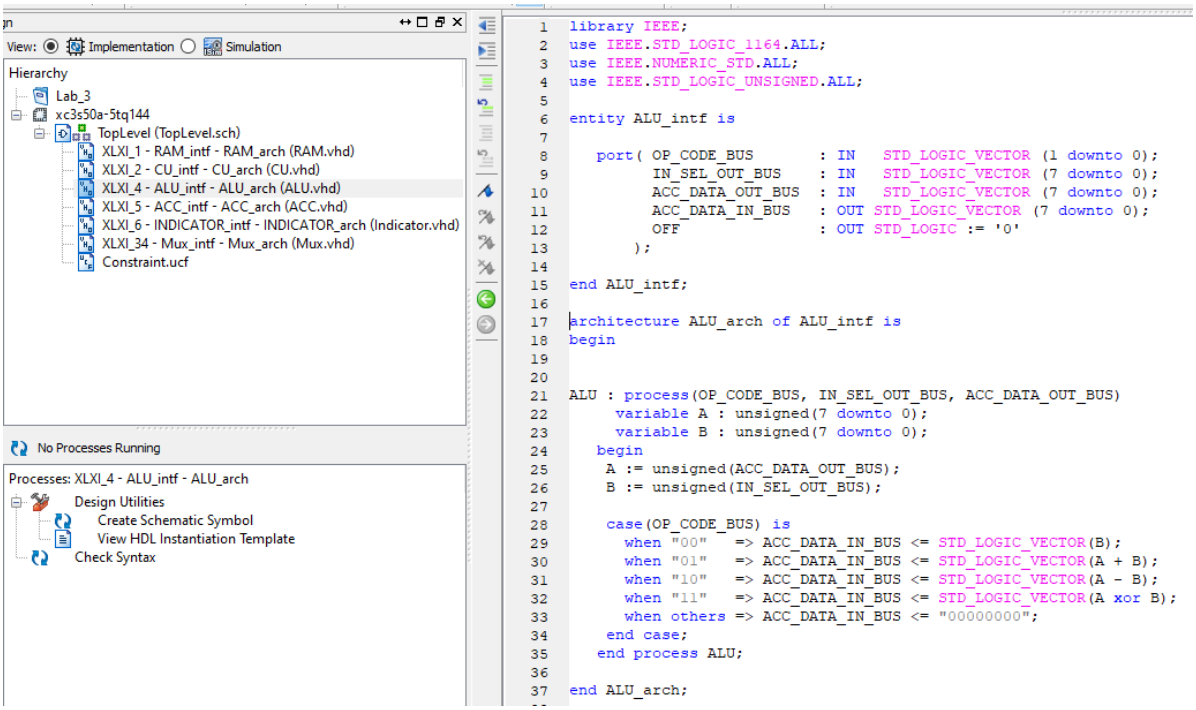


2) Створюю файл **ACC.vhd**, в якому реалізовую регістр:

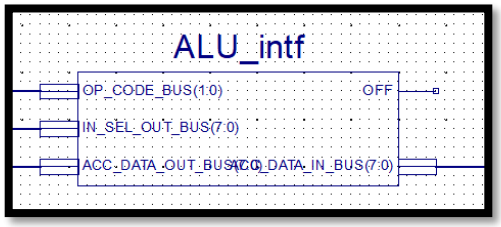




3) Створюю файл **ALU.vhd**, який реалізовує арифметико-логічний пристрій, що підтримує різні операції, а саме: « + », « - » та « xor »:



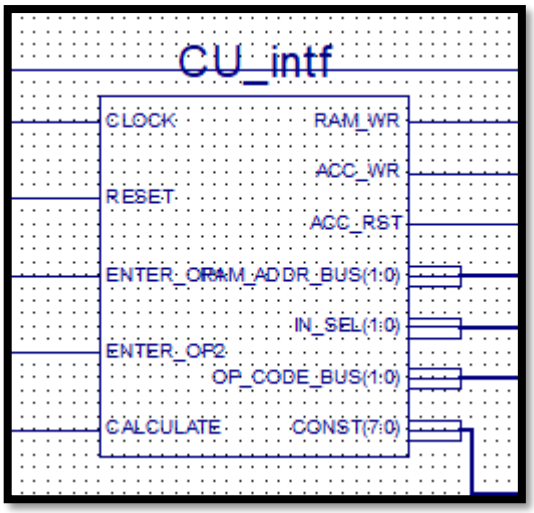

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity 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);
         OFF               : OUT STD_LOGIC := '0'
       );

end ALU_intf;

architecture ALU_arch of 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(A xor B);
      when others => ACC_DATA_IN_BUS <= "00000000";
    end case;
   end process ALU;

end ALU_arch;
```



4) Визначивши множину станів та умови переходу пристрою керування (CU), необхідних для обчислення виразу, створюю новий файл CU.vhd, який реалізовує пристрій керування, згідно визначеного алгоритму:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CU_intf 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);
    IN_SEL        : OUT STD_LOGIC_VECTOR (1 downto 0);
    ACC_WR        : OUT STD_LOGIC;
    ACC_RST       : OUT STD_LOGIC;
    OP_CODE_BUS   : OUT STD_LOGIC_VECTOR (1 downto 0);
    CONST         : OUT STD_LOGIC_VECTOR (7 downto 0)
    );
end CU_intf;

architecture CU_arch of CU_intf is

signal const_bus : std_logic_vector (7 downto 0) := "00000100";
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
CONST <= const_bus;
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

      cu_next_state <= cu_cur_state;

     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;


```

```
77
78    CU_OUTPUT_DECODE: process (cu_cur_state)
79    begin
80     case(cu_cur_state) is
81       when cu_rst     =>
82         IN_SEL        <= "00";
83         OP_CODE_BUS   <= "00";
84         RAM_ADDR_BUS  <= "00";
85         RAM_WR        <= '0';
86         ACC_RST       <= '1';
87         ACC_WR        <= '0';
88       when cu_idle     =>
89         IN_SEL        <= "00";
90         OP_CODE_BUS   <= "00";
91         RAM_ADDR_BUS  <= "00";
92         RAM_WR        <= '0';
93         ACC_RST       <= '0';
94         ACC_WR        <= '0';
95       when cu_load_op1   =>
96         IN_SEL        <= "00";
97         OP_CODE_BUS   <= "00";
98         RAM_ADDR_BUS  <= "00";
99         RAM_WR        <= '1';
100        ACC_RST       <= '0';
101        ACC_WR        <= '1';
102      when cu_load_op2   =>
103        IN_SEL        <= "00";
104        OP_CODE_BUS   <= "00";
105        RAM_ADDR_BUS  <= "01";
106        RAM_WR        <= '1';
107        ACC_RST       <= '0';
108        ACC_WR        <= '1';
109      when cu_run_calc0 =>
110        IN_SEL        <= "01";
111        OP_CODE_BUS   <= "00";
112        RAM_ADDR_BUS  <= "00";
113        RAM_WR        <= '0';
114        ACC_RST       <= '0';
115        ACC_WR        <= '1';
116      when cu_run_calc1 =>
117        IN_SEL        <= "10";
118        OP_CODE_BUS   <= "01";
116      when cu_run_calc1 =>
117        IN_SEL        <= "10";
118        OP_CODE_BUS   <= "01";
119        RAM_ADDR_BUS  <= "00";
120        RAM_WR        <= '0';
121        ACC_RST       <= '0';
122        ACC_WR        <= '1';
123      when cu_run_calc2 =>
124        IN_SEL        <= "01";
125        OP_CODE_BUS   <= "11";
126        RAM_ADDR_BUS  <= "01";
127        RAM_WR        <= '0';
128        ACC_RST       <= '0';
129        ACC_WR        <= '1';
130      when cu_run_calc3 =>
131        IN_SEL        <= "01";
132        OP_CODE_BUS   <= "10";
133        RAM_ADDR_BUS  <= "00";
134        RAM_WR        <= '0';
135        ACC_RST       <= '0';
136        ACC_WR        <= '1';
137      when cu_finish   =>
138        IN_SEL        <= "00";
139        OP_CODE_BUS   <= "00";
140        RAM_ADDR_BUS  <= "00";
141        RAM_WR        <= '0';
142        ACC_RST       <= '0';
143        ACC_WR        <= '0';
144      when others    =>
145        IN_SEL        <= "00";
146        OP_CODE_BUS   <= "00";
147        RAM_ADDR_BUS  <= "00";
148        RAM_WR        <= '0';
149        ACC_RST       <= '0';
150        ACC_WR        <= '0';
151    end case;
152   end process;
153
154 end CU_arch;
```

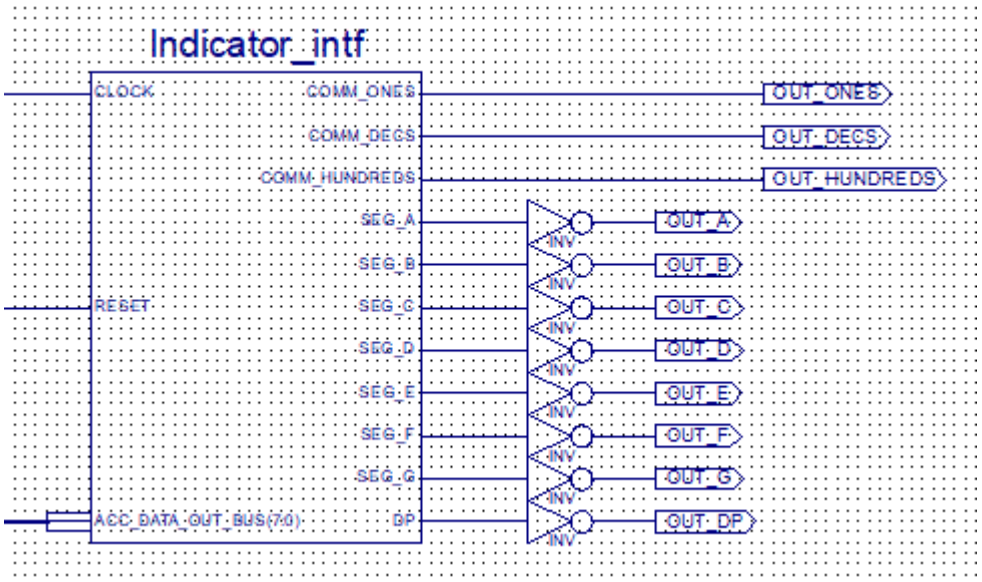


5) Створюю файл **Indicator.vhd,** в якому реалізовую блок індикації (7-SEG DECODER):

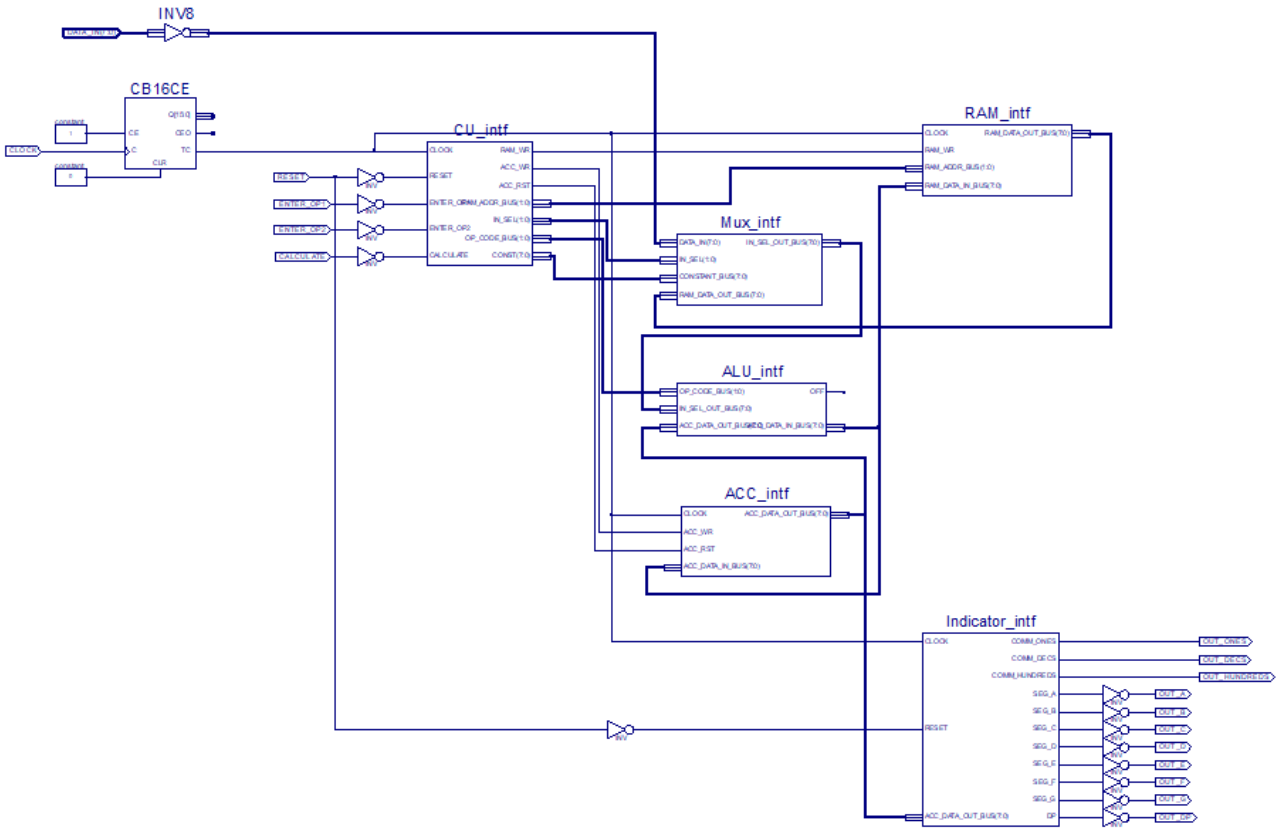
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity INDICATOR_intf is
port (ACC_DATA_OUT_BUS   : 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 INDICATOR_intf;

architecture INDICATOR_arch of INDICATOR_intf is

signal ONES_BUS     : STD_LOGIC_VECTOR (3 downto 0) := "0000";
signal DECS_BUS     : STD_LOGIC_VECTOR (3 downto 0) := "0001";
signal HUNDREDS_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;
```

```
35  begin
36       bcd     := (others => '0');
37       hex_src := ACC_DATA_OUT_BUS;
38
39  for i in hex_src'range loop
40     if bcd(3 downto 0) > "0100" then
41        bcd(3 downto 0) := bcd(3 downto 0) + "0011";
42     end if;
43
44     if bcd(7 downto 4) > "0100" then
45        bcd(7 downto 4) := bcd(7 downto 4) + "0011";
46     end if;
47
48     if bcd(11 downto 8) > "0100" then
49        bcd(11 downto 8) := bcd(11 downto 8) + "0011";
50     end if;
51
52     bcd := bcd( 10 downto 0)& hex_src(hex_src'left);
53     hex_src := hex_src(hex_src'left - 1 downto hex_src'right ) & '0';
54  end loop;
55
56  HUNDREDS_BUS <= bcd(11 downto 8);
57  DECS_BUS     <= bcd(7 downto 4);
58  ONES_BUS     <= bcd(3 downto 0);
59
60 end process bin_to_bcd ;
61
62 INDICATE   : process(CLOCK)
63     type DIGIT_TYPE is (ONES, DECS, HUNDREDS);
64
65     variable CUR_DIGIT     : DIGIT_TYPE := ONES;
66     variable DIGIT_VAL     : STD_LOGIC_VECTOR(3 downto 0) := "0000";
67     variable DIGIT_CTRL    : STD_LOGIC_VECTOR(6 downto 0) := "0000000";
68     variable COMMONS_CTRL  : STD_LOGIC_VECTOR(2 downto 0) := "000";
69
70      begin
71        if (rising_edge(CLOCK)) then
72          if(RESET = '0') then
73            case CUR_DIGIT is
74              when ONES =>
75                  DIGIT_VAL := ONES_BUS;
76                  CUR_DIGIT := DECS;
77                  COMMONS_CTRL := "001";
78              when DECS =>
79                  DIGIT_VAL := DECS_BUS;
80                  CUR_DIGIT := HUNDREDS;
81                  COMMONS_CTRL := "010";
82              when HUNDREDS =>
83                  DIGIT_VAL := HUNDREDS_BUS;
84                  CUR_DIGIT := ONES;
85                  COMMONS_CTRL := "100";
86              when others =>
87                  DIGIT_VAL := ONES_BUS;
88                  CUR_DIGIT := ONES;
89                  COMMONS_CTRL := "000";
90            end case;
91
92            case DIGIT_VAL is               --abcdefg
93              when "0000" => DIGIT_CTRL := "1111110";
94              when "0001" => DIGIT_CTRL := "0110000";
95              when "0010" => DIGIT_CTRL := "1101101";
96              when "0011" => DIGIT_CTRL := "1111001";
97              when "0100" => DIGIT_CTRL := "0110011";
98              when "0101" => DIGIT_CTRL := "1011011";
99              when "0110" => DIGIT_CTRL := "1011111";
100             when "0111" => DIGIT_CTRL := "1110000";
101             when "1000" => DIGIT_CTRL := "1111111";
102             when "1001" => DIGIT_CTRL := "1111011";
103             when others => DIGIT_CTRL := "0000000";
104           end case;
105         else
106           DIGIT_VAL := ONES_BUS;
107           CUR_DIGIT := ONES;
108           COMMONS_CTRL := "000";
109         end if;
```

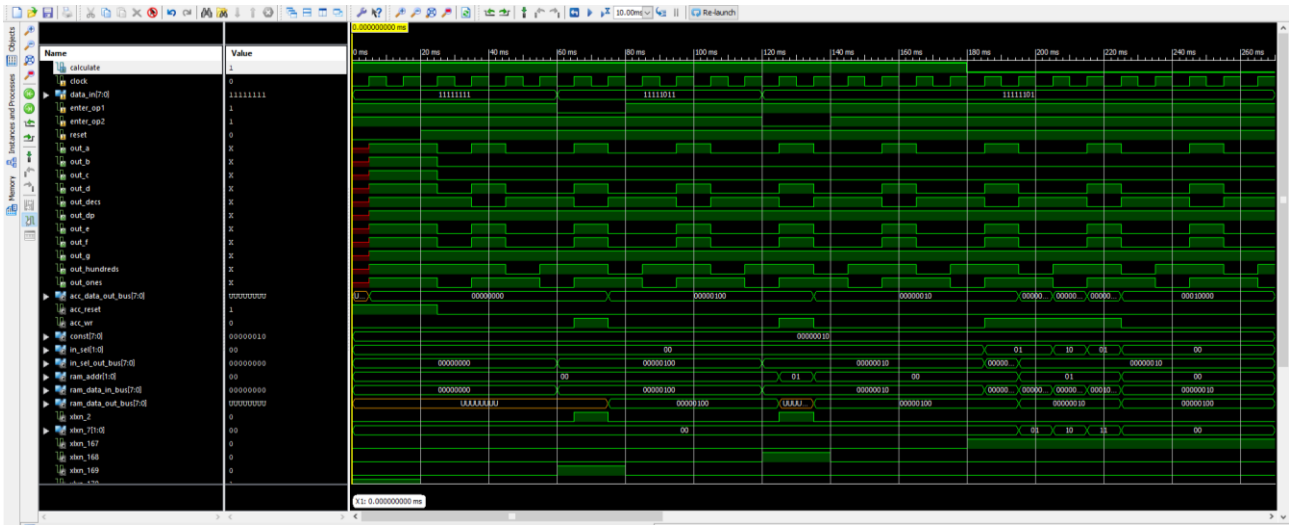
```
110
111         COMM_ONES     <= not COMMONS_CTRL(0);
112         COMM_DECS     <= not COMMONS_CTRL(1);
113         COMM_HUNDREDS <= not COMMONS_CTRL(2);
114
115         SEG_A <= DIGIT_CTRL(6);
116         SEG_B <= DIGIT_CTRL(5);
117         SEG_C <= DIGIT_CTRL(4);
118         SEG_D <= DIGIT_CTRL(3);
119         SEG_E <= DIGIT_CTRL(2);
120         SEG_F <= DIGIT_CTRL(1);
121         SEG_G <= DIGIT_CTRL(0);
122         DP    <= '0';
123
124       end if;
125   end process INDICATE;
126
127 end INDICATOR_arch;
```



6) Згенерувавши символи для раніше імплементованих компонентів створюю файл верхнього рівня **TopLevel.sch**, в якому виконую інтеграцію компонентів системи між собою та зі стендом Elbert V2 – Spartan 3A FPGA:



7) За допомогою симулятора ISim симулюю роботу пристрою:

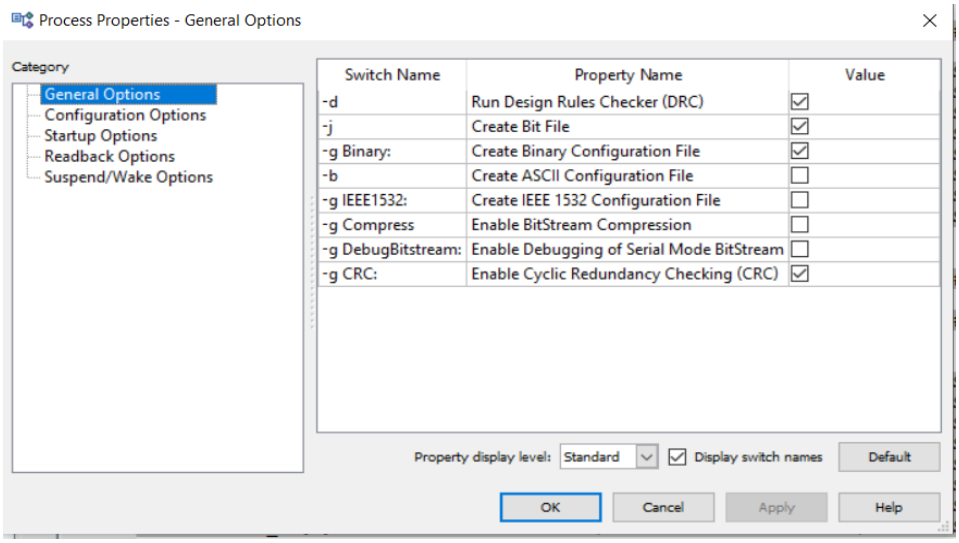


8) Призначив виводам схеми фізичні виводи цільової FPGA:

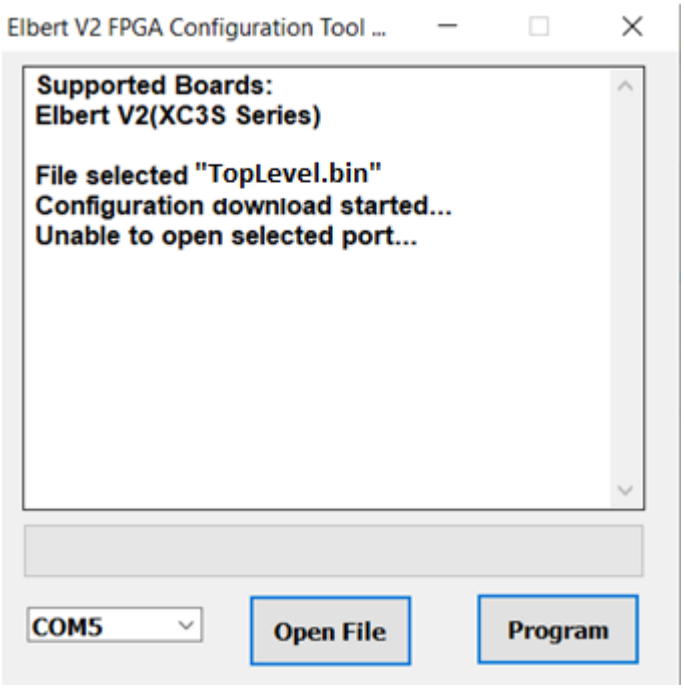
```
CONFIG VCCAUX = "3.3" ;

 # Clock 12 MHz
  NET "CLOCK"                 LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;

##########################################################################################################
#                                  Seven Segment Display
##########################################################################################################

    NET "OUT_A"     LOC = P117  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_B"     LOC = P116  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_C"     LOC = P115  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_D"     LOC = P113  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_E"     LOC = P112  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_F"     LOC = P111  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_G"     LOC = P110  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_DP"    LOC = P114  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

    NET "OUT_ONES"          LOC = P124  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_DECS"          LOC = P121  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_HUNDREDS"      LOC = P120  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

##########################################################################################################
#                                  DP Switches
##########################################################################################################

    NET "DATA_IN(0)"        LOC = P70   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(1)"        LOC = P69   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(2)"        LOC = P68   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(3)"        LOC = P64   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(4)"        LOC = P63   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(5)"        LOC = P60   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(6)"        LOC = P59   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA_IN(7)"        LOC = P58   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

##########################################################################################################
#                                  Switches
##########################################################################################################
    NET "ENTER_OP1"         LOC = P80   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "ENTER_OP2"         LOC = P79   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "CALCULATE"         LOC = P78   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "RESET"             LOC = P75   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
##########################################################################################################
```

9) Згенерував BIT файл для цільової FPGA.

Послідовно запустив процеси Synthesize-XST, Implement Design та Generate Programming File:



10) Запрограмував лабораторний стенд отриманим BIT файлом:



Висновок: виконуючи дану лабораторну роботу, я навчився на базі стенда Elbert V2 – Spartan 3A FPGA реалізовувати цифровий автомат для обчислення значення виразу.