

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Кафедра ЕОМ
З лабораторної роботи № 1
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: студент гр. КІ-201
Панасевич Ю. В.

Прийняв:
Козак Н. Б.

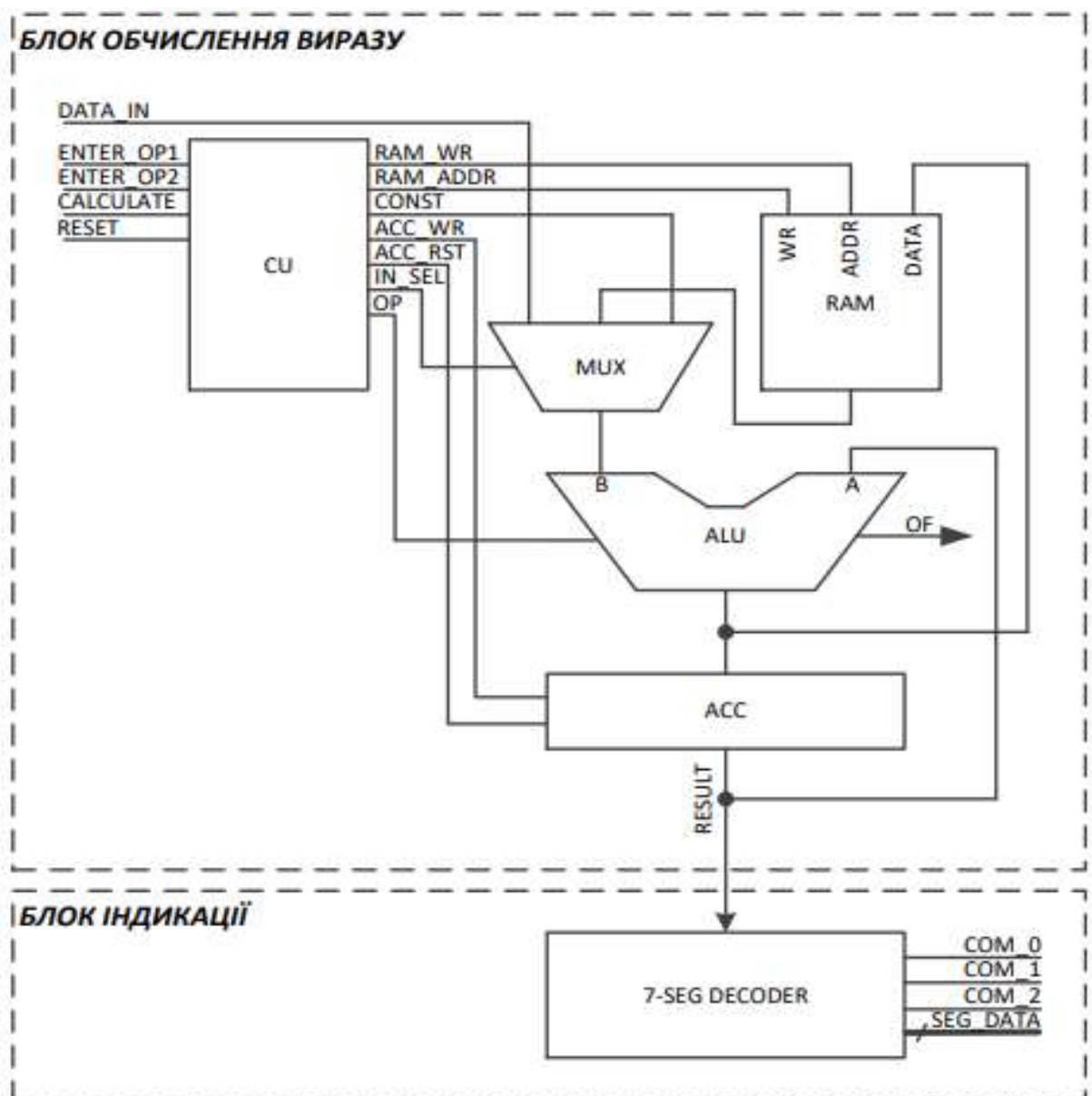
Мета: на базі стенда реалізувати цифровий автомат для обчислення виразу.

Завдання:

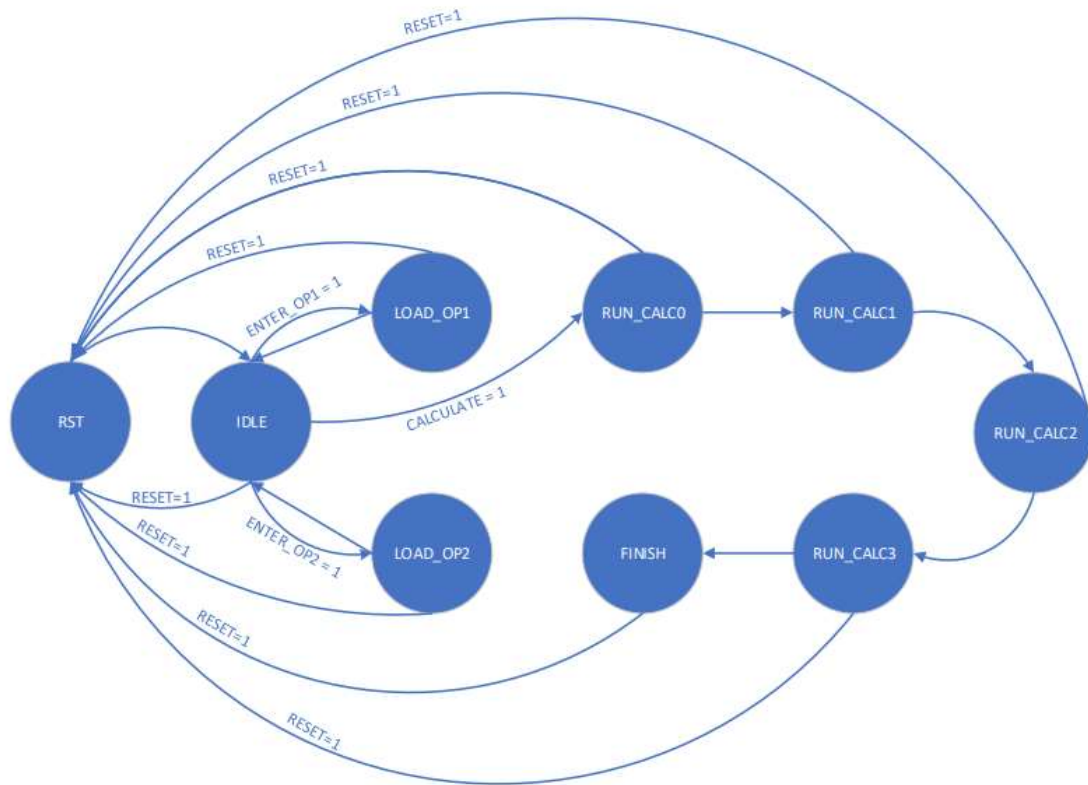
- 1) На базі стенда реалізувати цифровий автомат для обчислення значення виразу дотримуючись наступних вимог:

ВАРІАНТ	ВИРАЗ
4	$((4 + OP1) \text{ xor } OP2) - OP1$

- 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.

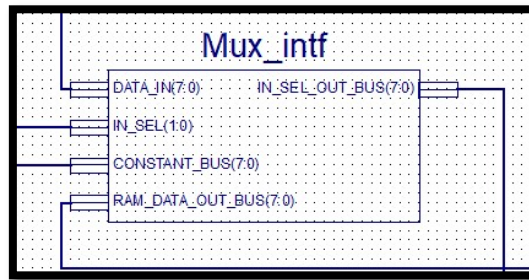
Виконання роботи:

1) Створюю новий файл **Mux.vhd** в якому реалізую 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

```

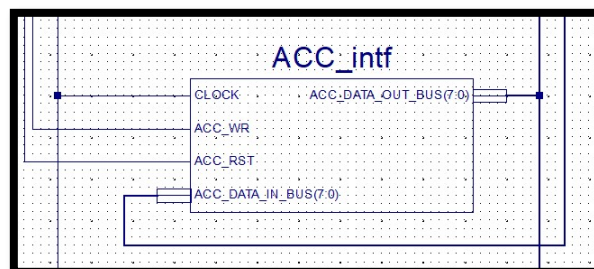


2) Створюю файл **ACC.vhd**, в якому реалізую регістр:

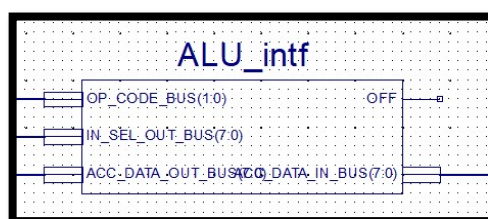
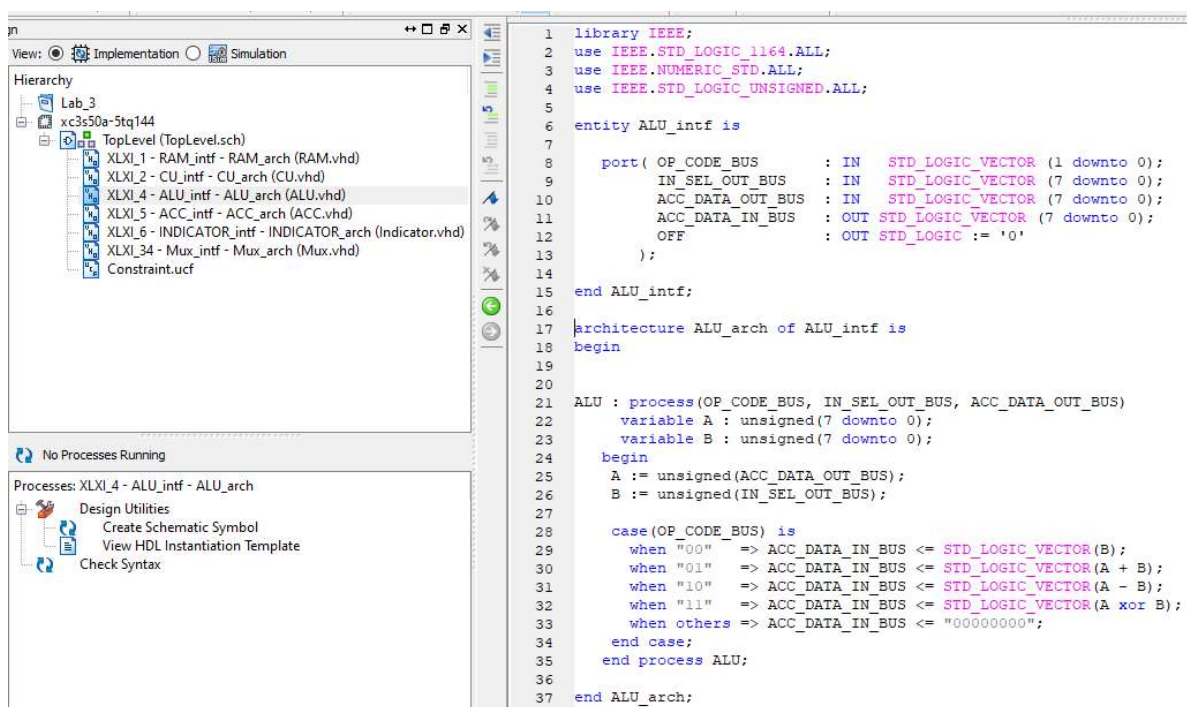
```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ACC_intf is
5      port(CLOCK : IN STD_LOGIC;
6            ACC_WR : IN STD_LOGIC;
7            ACC_RST : IN STD_LOGIC;
8            ACC_DATA_OUT_BUS : OUT STD_LOGIC_VECTOR(7 downto 0);
9            ACC_DATA_IN_BUS : IN STD_LOGIC_VECTOR(7 downto 0);
10     );
11 end ACC_intf;
12
13 architecture ACC_arch of ACC_intf is
14     signal ACC_DATA : STD_LOGIC_VECTOR(7 downto 0);
15 begin
16     ACC : process(CLOCK, ACC_DATA)
17     begin
18         if(rising_edge(CLOCK)) then
19             if(ACC_RST='1') then
20                 ACC_DATA <="00000000";
21             elsif(ACC_WR='1') then
22                 ACC_DATA <= ACC_DATA_IN_BUS;
23             end if;
24         end if;
25         ACC_DATA_OUT_BUS <= ACC_DATA;
26     end process ACC;
27 end ACC_arch;

```



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



- 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:


```

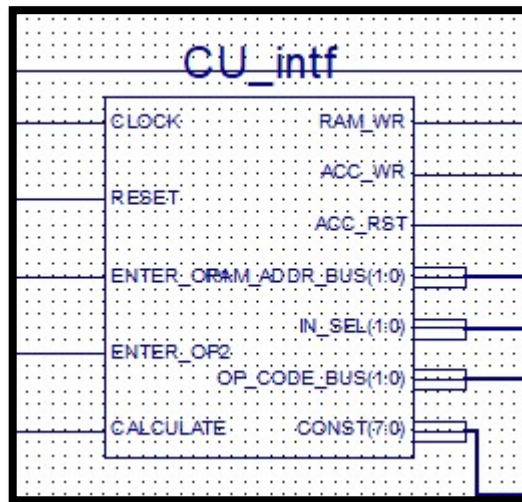
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity CU_intf is
5  port (CLOCK      : IN  STD_LOGIC;
6        RESET      : IN  STD_LOGIC;
7        ENTER_OP1   : IN  STD_LOGIC;
8        ENTER_OP2   : IN  STD_LOGIC;
9        CALCULATE    : IN  STD_LOGIC;
10       RAM_WR       : OUT STD_LOGIC;
11       RAM_ADDR_BUS : OUT STD_LOGIC_VECTOR (1 downto 0);
12       IN_SEL       : OUT STD_LOGIC_VECTOR (1 downto 0);
13       ACC_WR       : OUT STD_LOGIC;
14       ACC_RST      : OUT STD_LOGIC;
15       OP_CODE_BUS  : OUT STD_LOGIC_VECTOR (1 downto 0);
16       CONST        : OUT STD_LOGIC_VECTOR (7 downto 0)
17       );
18 end CU_intf;
19
20 architecture CU_arch of CU_intf is
21
22 signal const_bus : std_logic_vector (7 downto 0) := "00000100";
23 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);
24 signal cu_cur_state : cu_state_type;
25 signal cu_next_state : cu_state_type;
26
27 begin
28   CONST <= const_bus;
29   CU_SYNC_PROC: process (CLOCK)
30   begin
31     if (rising_edge(CLOCK)) then
32       if (RESET = '1') then
33         cu_cur_state <= cu_rst;
34       else
35         cu_cur_state <= cu_next_state;
36       end if;
37     end if;
38   end process;
39
40   CUNEXT_STATE_DECODE: process (cu_cur_state, ENTER_OP1, ENTER_OP2, CALCULATE)
41   begin
42
43     cu_next_state <= cu_cur_state;
44
45     case(cu_cur_state) is
46       when cu_rst =>
47         cu_next_state <= cu_idle;
48       when cu_idle =>
49         if (ENTER_OP1 = '1') then
50           cu_next_state <= cu_load_op1;
51         elsif (ENTER_OP2 = '1') then
52           cu_next_state <= cu_load_op2;
53         elsif (CALCULATE = '1') then
54           cu_next_state <= cu_run_calc0;
55         else
56           cu_next_state <= cu_idle;
57         end if;
58       when cu_load_op1 =>
59         cu_next_state <= cu_idle;
60       when cu_load_op2 =>
61         cu_next_state <= cu_idle;
62       when cu_run_calc0 =>
63         cu_next_state <= cu_run_calc1;
64       when cu_run_calc1 =>
65         cu_next_state <= cu_run_calc2;
66       when cu_run_calc2 =>
67         cu_next_state <= cu_run_calc3;
68       when cu_run_calc3 =>
69         cu_next_state <= cu_finish;
70       when cu_finish =>
71         cu_next_state <= cu_finish;
72       when others =>
73         cu_next_state <= cu_idle;
74     end case;
75   end process;
76
77

```

```

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";
119
120     when cu_run_calc1 =>
121       IN_SEL      <= "10";
122       OP_CODE_BUS <= "01";
123       RAM_ADDR_BUS <= "00";
124       RAM_WR      <= '0';
125       ACC_RST     <= '0';
126       ACC_WR      <= '1';
127     when cu_run_calc2 =>
128       IN_SEL      <= "01";
129       OP_CODE_BUS <= "11";
130       RAM_ADDR_BUS <= "01";
131       RAM_WR      <= '0';
132       ACC_RST     <= '0';
133       ACC_WR      <= '1';
134     when cu_run_calc3 =>
135       IN_SEL      <= "01";
136       OP_CODE_BUS <= "10";
137       RAM_ADDR_BUS <= "00";
138       RAM_WR      <= '0';
139       ACC_RST     <= '0';
140       ACC_WR      <= '1';
141     when cu_finish =>
142       IN_SEL      <= "00";
143       OP_CODE_BUS <= "00";
144       RAM_ADDR_BUS <= "00";
145       RAM_WR      <= '0';
146       ACC_RST     <= '0';
147       ACC_WR      <= '0';
148     when others =>
149       IN_SEL      <= "00";
150       OP_CODE_BUS <= "00";
151       RAM_ADDR_BUS <= "00";
152       RAM_WR      <= '0';
153       ACC_RST     <= '0';
154       ACC_WR      <= '0';
155   end case;
156 end process;
157
158 end CU_arch;

```



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

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity INDICATOR_intf is
7  port (ACC_DATA_OUT_BUS : IN  STD_LOGIC_VECTOR (7 downto 0);
8        CLOCK           : IN  STD_LOGIC;
9        RESET           : IN  STD_LOGIC;
10       COMM_ONES       : OUT  STD_LOGIC;
11       COMM_DECS       : OUT  STD_LOGIC;
12       COMM_HUNDREDS   : OUT  STD_LOGIC;
13       SEG_A           : OUT  STD_LOGIC;
14       SEG_B           : OUT  STD_LOGIC;
15       SEG_C           : OUT  STD_LOGIC;
16       SEG_D           : OUT  STD_LOGIC;
17       SEG_E           : OUT  STD_LOGIC;
18       SEG_F           : OUT  STD_LOGIC;
19       SEG_G           : OUT  STD_LOGIC;
20       DP              : OUT  STD_LOGIC
21     );
22 end INDICATOR_intf;
23
24 architecture INDICATOR_arch of INDICATOR_intf is
25
26   signal ONES_BUS      : STD_LOGIC_VECTOR (3 downto 0) := "0000";
27   signal DECS_BUS      : STD_LOGIC_VECTOR (3 downto 0) := "0001";
28   signal HUNDREDS_BUS  : STD_LOGIC_VECTOR (3 downto 0) := "0000";
29   begin
30
31   bin_to_bcd : process(ACC_DATA_OUT_BUS)
32     variable hex_src : STD_LOGIC_VECTOR (7 downto 0);
33     variable bcd      : STD_LOGIC_VECTOR (11 downto 0);
34
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;

```



```

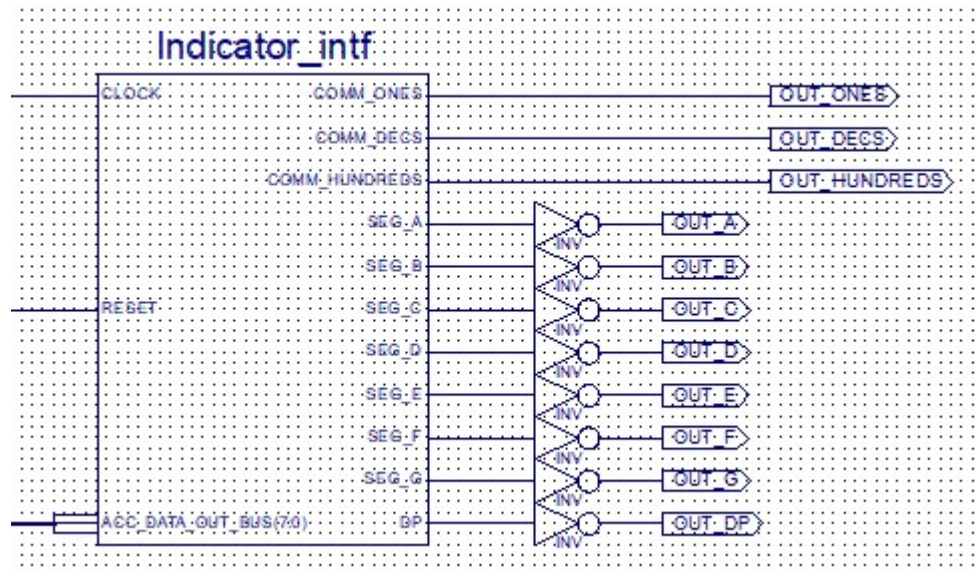
35 begin
36     bcd      := (others => '0');
37     hex_src := ACC_DATA_OUT_BUS;
38
39     for i in hex_src'range loop
40         if bcd(3 downto 0) > "0100" then
41             bcd(3 downto 0) := bcd(3 downto 0) + "0011";
42         end if;
43
44         if bcd(7 downto 4) > "0100" then
45             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) := "00000000";
68     variable COMMONS_CTRL   : STD_LOGIC_VECTOR(2 downto 0) := "000";
69
70
71 begin
72     if (rising_edge(CLOCK)) then
73         if(RESET = '0') then
74             case CUR_DIGIT is
75                 when ONES =>
76                     DIGIT_VAL := ONES_BUS;
77                     CUR_DIGIT := DECS;
78                     COMMONS_CTRL := "001";
79                 when DECS =>
80                     DIGIT_VAL := DECS_BUS;
81                     CUR_DIGIT := HUNDREDS;
82                     COMMONS_CTRL := "010";
83                 when HUNDREDS =>
84                     DIGIT_VAL := HUNDREDS_BUS;
85                     CUR_DIGIT := ONES;
86                     COMMONS_CTRL := "100";
87                 when others =>
88                     DIGIT_VAL := ONES_BUS;
89                     CUR_DIGIT := ONES;
90                     COMMONS_CTRL := "000";
91             end case;
92
93             case DIGIT_VAL is
94                 --abdefg
95                 when "0000" => DIGIT_CTRL := "1111110";
96                 when "0001" => DIGIT_CTRL := "0110000";
97                 when "0010" => DIGIT_CTRL := "1101101";
98                 when "0011" => DIGIT_CTRL := "1111001";
99                 when "0100" => DIGIT_CTRL := "0110011";
100                when "0101" => DIGIT_CTRL := "1011011";
101                when "0110" => DIGIT_CTRL := "1011111";
102                when "0111" => DIGIT_CTRL := "1110000";
103                when "1000" => DIGIT_CTRL := "1111111";
104                when "1001" => DIGIT_CTRL := "1111011";
105                when others => DIGIT_CTRL := "0000000";
106            end case;
107
108             else
109                 DIGIT_VAL := ONES_BUS;
110                 CUR_DIGIT := ONES;
111                 COMMONS_CTRL := "000";
112             end if;

```

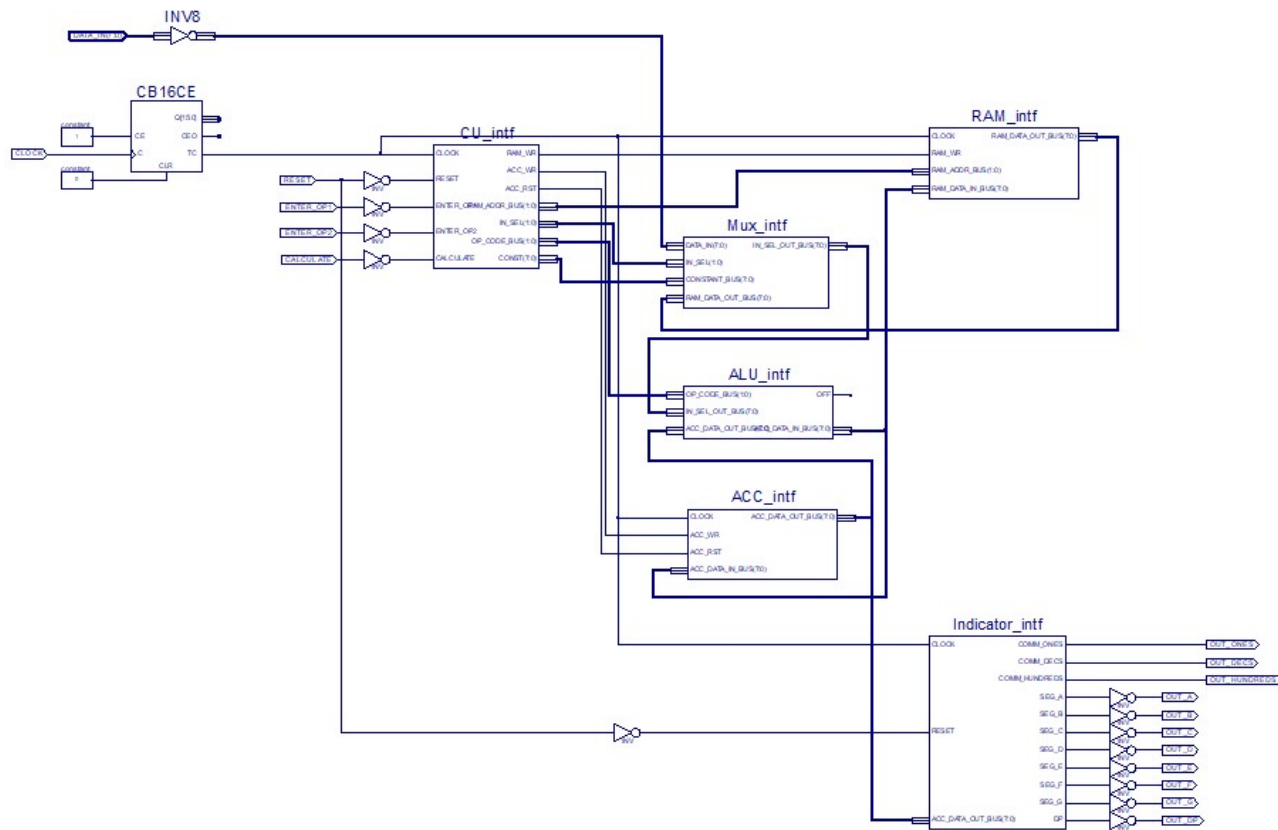
```

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;

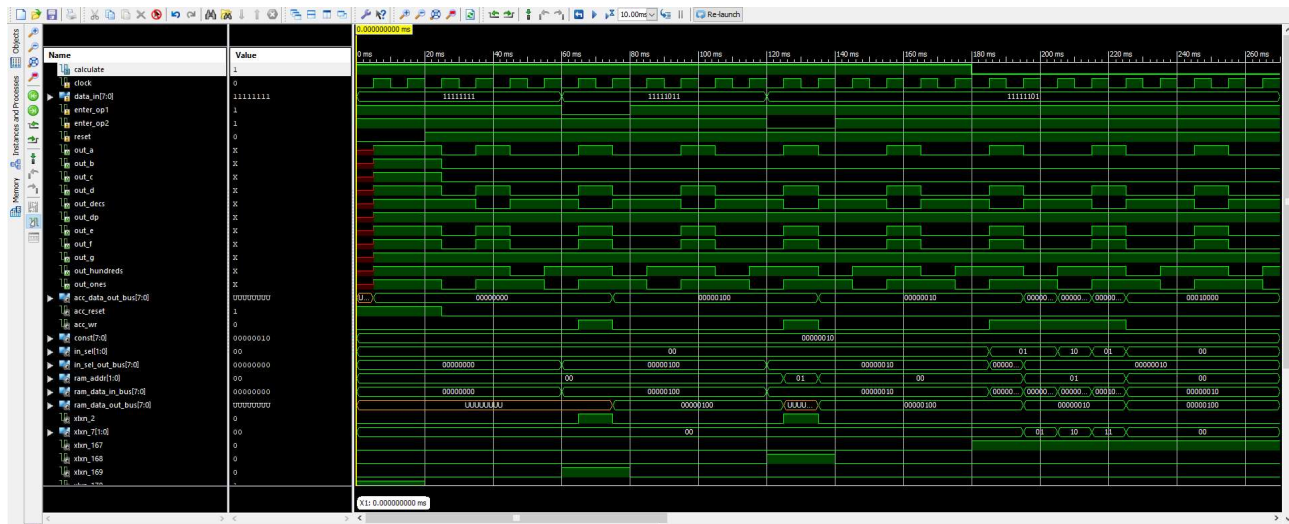
```



- 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 симую роботу пристрою:



8) Призначив виводам схеми фізичні виводи цільової FPGA:

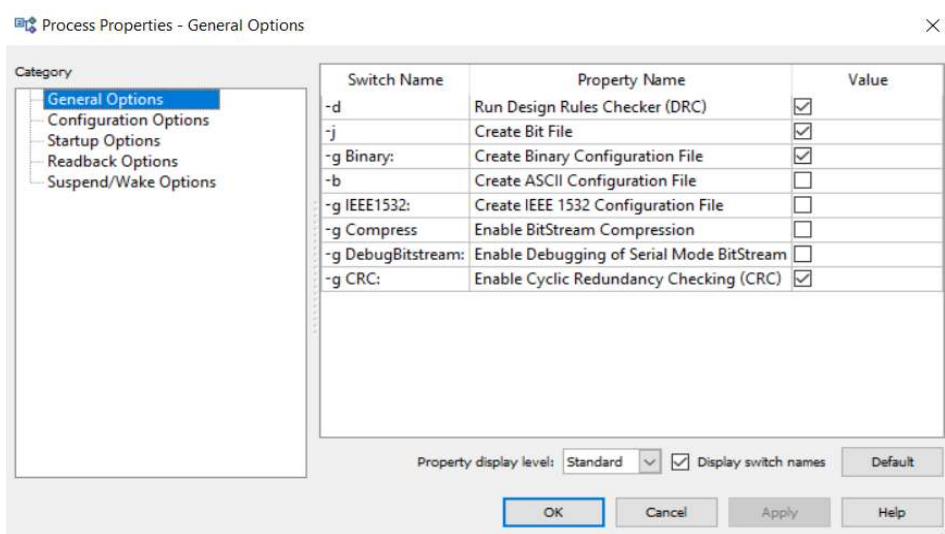
```

1 |CONFIG VCCAUX = "3.3" ;
2
3 |# Clock 12 MHz
4 |NET "CLOCK" LOC = P129 | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
5
6 |#####
7 |# Seven Segment Display
8 |#####
9
10 |NET "OUT_A" LOC = P117 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
11 |NET "OUT_B" LOC = P116 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
12 |NET "OUT_C" LOC = P115 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
13 |NET "OUT_D" LOC = P113 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
14 |NET "OUT_E" LOC = P112 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
15 |NET "OUT_F" LOC = P111 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
16 |NET "OUT_G" LOC = P110 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
17 |NET "OUT_DP" LOC = P114 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
18
19 |NET "OUT_ONES" LOC = P124 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
20 |NET "OUT_DECS" LOC = P121 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
21 |NET "OUT_HUNDREDS" LOC = P120 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
22
23 |#####
24 |# DP Switches
25 |#####
26
27 |NET "DATA_IN(0)" LOC = P70 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
28 |NET "DATA_IN(1)" LOC = P69 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
29 |NET "DATA_IN(2)" LOC = P68 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
30 |NET "DATA_IN(3)" LOC = P64 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
31 |NET "DATA_IN(4)" LOC = P63 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
32 |NET "DATA_IN(5)" LOC = P60 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
33 |NET "DATA_IN(6)" LOC = P59 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
34 |NET "DATA_IN(7)" LOC = P58 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
35
36 |#####
37 |# Switches
38 |#####
39 |NET "ENTER_OP1" LOC = P80 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
40 |NET "ENTER_OP2" LOC = P79 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
41 |NET "CALCULATE" LOC = P78 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
42 |NET "RESET" LOC = P75 | PULLUP | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
43 |#####

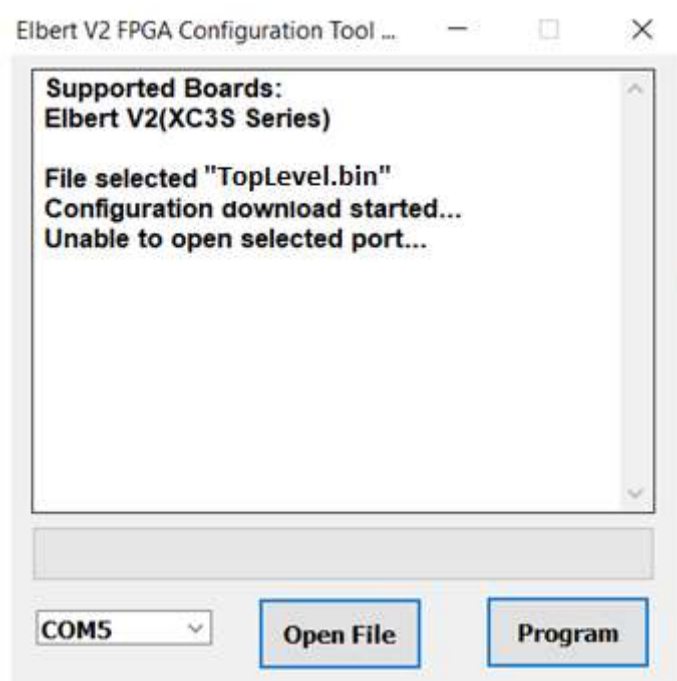
```

9) Згенерував BIT файл для цільової FPGA.

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



- 10) Запрограмував лабораторний стенд отриманим ВІТ файлом:



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