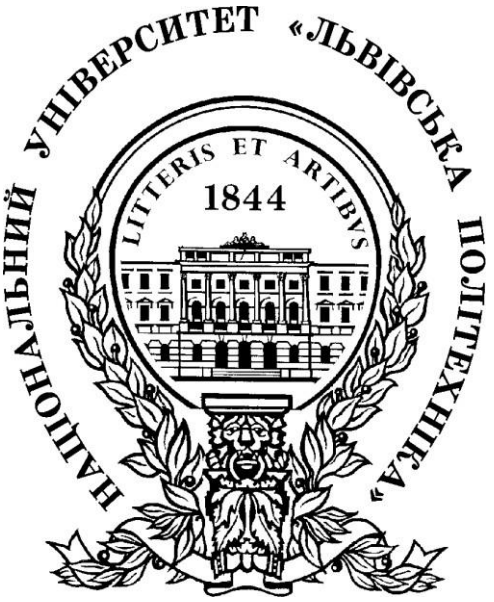
Міністерство освіти і науки України

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»

Варіант-7

Виконав:

Душний О.І.

Прийняв:

Козак Н.Б

Львів 2023

Мета: На базі стенда Elbert V2 – Spartan 3A FPGA реалізувати цифровий автомат для обчислення виразу.

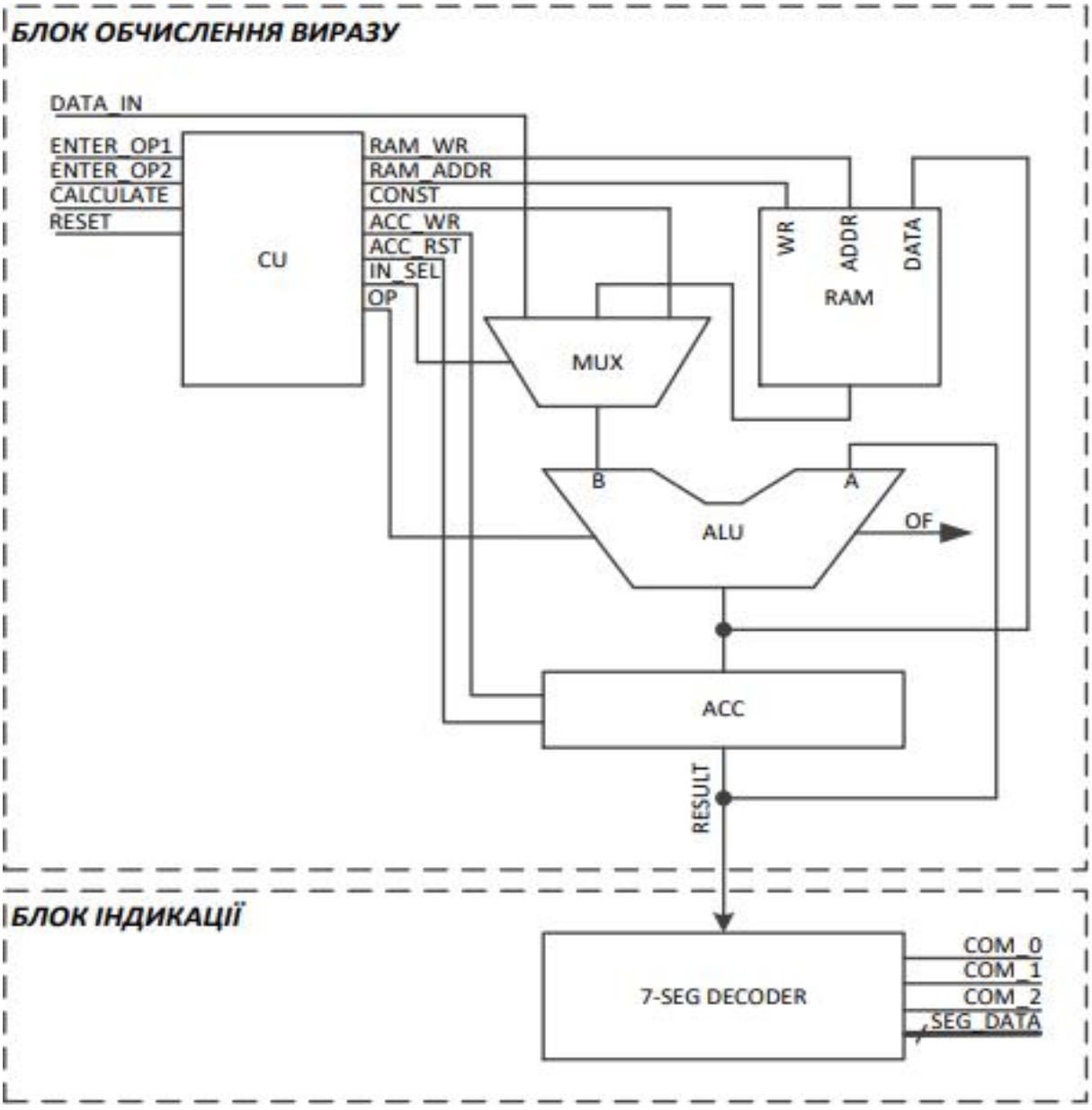
Завдання

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

1) Функціонал пристрою повинен бути реалізований згідно отриманого варіанту завдання:

7	((OP1 << 2) – OP2) + 4

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



Хід роботи

1. Створю файл Mux.vhd в якому реалізовую мультиплексор:

```
entity MUX is
    Port ( DATA_IN : in  STD_LOGIC_VECTOR (7 downto 0);
           RAM_IN : in  STD_LOGIC_VECTOR (7 downto 0);
           IN_SEL : in  STD_LOGIC_VECTOR (1 downto 0);
           MUX_DATA_OUT : OUT  STD_LOGIC_VECTOR (7 downto 0));
end MUX;

architecture Behavioral of MUX is

begin
    process(DATA_IN, IN_SEL, RAM_IN)
    begin
      if(IN_SEL = "00") then
         MUX_DATA_OUT <= DATA_IN;
      elsif(IN_SEL = "01") then
         MUX_DATA_OUT <= RAM_IN;
      elsif(IN_SEL = "10")then
         MUX_DATA_OUT <= "00000101";
      else
         MUX_DATA_OUT <= "00000000";
      end if;
    end process;
end Behavioral;
```

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

```
entity ACC is
    Port ( INPUT_BUS : in  STD_LOGIC_VECTOR (7 downto 0);
           OUTPUT_BUS : out  STD_LOGIC_VECTOR (7 downto 0);
           ACC_WR : in  STD_LOGIC;
           ACC_RST : in  STD_LOGIC;
           CLOCK : in STD_LOGIC);
end ACC;

architecture Behavioral of ACC is

signal ACC_DATA : STD_LOGIC_VECTOR(7 downto 0);

begin
process(CLOCK, INPUT_BUS, ACC_DATA)
    begin
      if (rising_edge(CLOCK)) then
         if(ACC_RST = '1') then
            ACC_DATA <= "00000000";
         elsif (ACC_WR = '1') then
            ACC_DATA <= INPUT_BUS;
         end if;
      end if;
      OUTPUT_BUS <= ACC_DATA;
    end process;
end Behavioral;
```

3. Створюю файл ALU.vhd, який реалізовує арифметико-логічний пристрій, що підтримує операції, необхідні для виконання виразу згідно з варіантом, а саме операції: «+», «-» та «SLL»:

```
entity ALU is
    Port ( A_BUS    : in   STD_LOGIC_VECTOR (7 downto 0);
           B_BUS    : in   STD_LOGIC_VECTOR (7 downto 0);
           OUT_BUS  : out  STD_LOGIC_VECTOR (7 downto 0);
           OP_CODE  : in   STD_LOGIC_VECTOR (1 downto 0);
           OVERFLOW : out  STD_LOGIC);
end ALU;

architecture Behavioral of ALU is

begin

process(OP_CODE, A_BUS, B_BUS)
      variable A : unsigned(7 downto 0);
      variable B : unsigned(7 downto 0);
    begin
        A := unsigned(A_BUS);
        B := unsigned(B_BUS);

        case(OP_CODE) is
           when "00"   =>
              OUT_BUS <= STD_LOGIC_VECTOR(B);
              OVERFLOW <= '0';

   ----------------------------------------------------
           when "01"   =>
           if ((A > 128 and B > 127) or (A > 127 and B > 128)) then
              OVERFLOW <= '1';
           else
              OVERFLOW <= '0';
           end if;
              OUT_BUS <= STD_LOGIC_VECTOR(A - B);

   ----------------------------------------------------
           when "10"   =>
           if (A < B) then
              OVERFLOW <= '1';
           else
              OVERFLOW <= '0';
           end if;
              OUT_BUS <= STD_LOGIC_VECTOR(A + B);

   ----------------------------------------------------
           when "11"   =>
              OUT_BUS <= STD_LOGIC_VECTOR(A sll 2);

              OVERFLOW <= '0';
   ----------------------------------------------------
           when others =>
              OUT_BUS <= "00000000";
              OVERFLOW <= '0';
        end case;
    end process;

end Behavioral;
```

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:

```
entity CU is
    Port ( 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);
           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);
           CLOCK          : IN   STD_LOGIC);
end CU;

architecture CU_arch 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
   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;

   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;

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         <= "01";
          OP_CODE_BUS    <= "00";
          RAM_ADDR_BUS   <= "00";
          RAM_WR         <= '0';
          ACC_RST        <= '0';
          ACC_WR         <= '1';
       when cu_run_calc1 =>
          IN_SEL         <= "00";
          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         <= "10";
          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 CU_arch;
```

5. Створюю файл RAM.vhd, в якому реалізовую пам’ять пристрою:

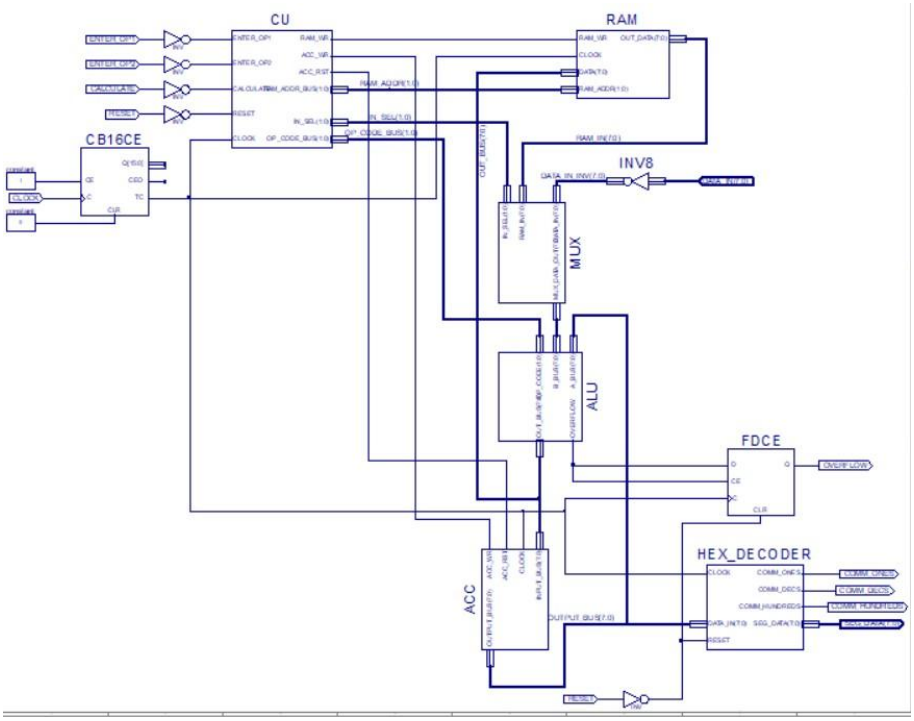
```
entity RAM is
    Port ( DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           RAM_ADDR : in  STD_LOGIC_VECTOR (1 downto 0);
           RAM_WR : in  STD_LOGIC;
           CLOCK : in STD_LOGIC;
           OUT_DATA : out  STD_LOGIC_VECTOR (7 downto 0));
end RAM;

architecture Behavioral of RAM is

type ram_type is array (3 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
signal RAM_UNIT         : ram_type;

begin

    process(CLOCK, RAM_ADDR, RAM_UNIT)
    begin
      if (rising_edge(CLOCK)) then
         if (RAM_WR = '1') then
            RAM_UNIT(conv_integer(RAM_ADDR)) <= DATA;
         end if;
      end if;
      OUT_DATA <= RAM_UNIT(conv_integer(RAM_ADDR));
    end process;


end Behavioral;
```

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

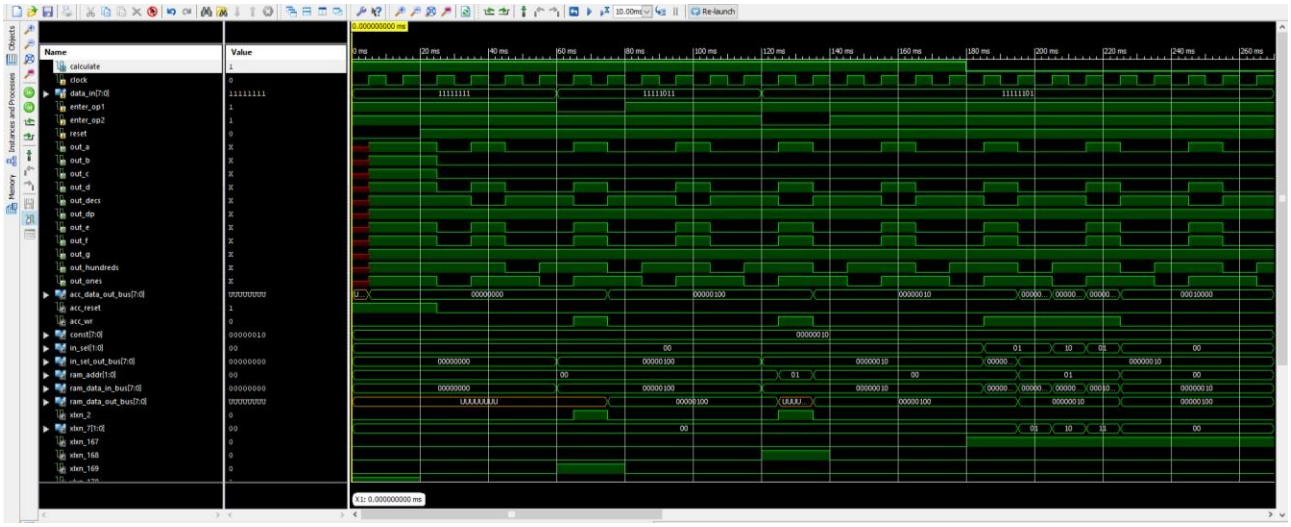
```
entity HEX_DECODER is
    Port ( DATA_IN : in  STD_LOGIC_VECTOR (7 downto 0);
           CLOCK : in STD_LOGIC;
           RESET : in STD_LOGIC;
           SEG_DATA : out  STD_LOGIC_VECTOR (7 downto 0) := "10000000";
           COMM_ONES       : OUT STD_LOGIC := '0';
           COMM_DECS       : OUT STD_LOGIC := '0';
           COMM_HUNDREDS   : OUT STD_LOGIC := '0');
end HEX_DECODER;

architecture Behavioral of HEX_DECODER 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

    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 ;

        HUNDREDS_BUS       <=  bcd (11 downto 8);
        DECS_BUS        <=  bcd (7 downto 4);
        ONES_BUS       <=  bcd (3 downto 0);

    end process;

    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 := HUNDREDS_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_DATA(6) <= (DIGIT_CTRL(6) xor '1');
        SEG_DATA(5) <= (DIGIT_CTRL(5) xor '1');
        SEG_DATA(4) <= (DIGIT_CTRL(4) xor '1');
        SEG_DATA(3) <= (DIGIT_CTRL(3) xor '1');
        SEG_DATA(2) <= (DIGIT_CTRL(2) xor '1');
        SEG_DATA(1) <= (DIGIT_CTRL(1) xor '1');
        SEG_DATA(0) <= (DIGIT_CTRL(0) xor '1');
        --SEG_DATA(7) <= '0';
        end if;
   end process;

end Behavioral;
```

7. 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:



8. За допомогою симулятора ISim симулюю роботу пристрою:

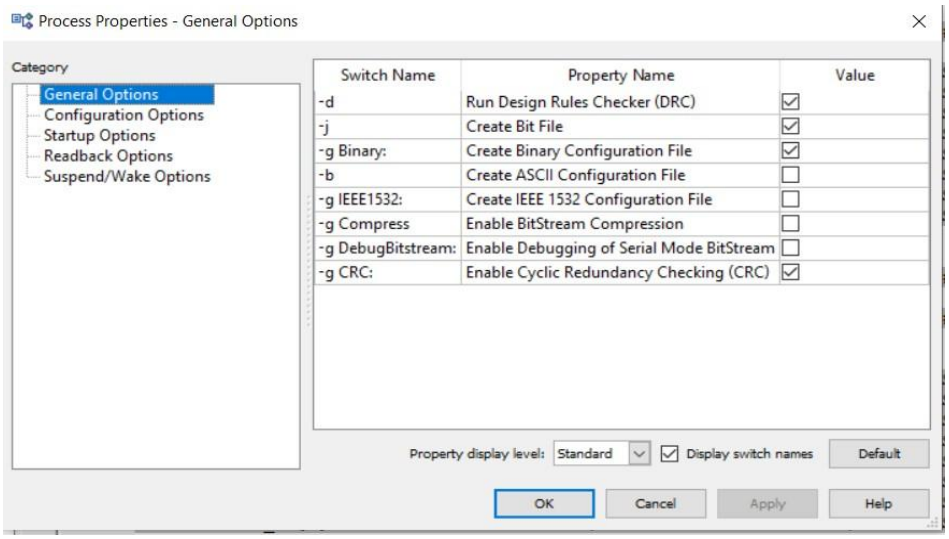


9. Призначив виводам схеми фізичні виводи цільової FPGA

```
##############################################################################################

    NET "SEG_DATA[6]"     LOC = P117  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[5]"     LOC = P116  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[4]"     LOC = P115  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[3]"     LOC = P113  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[2]"     LOC = P112  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[1]"     LOC = P111  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[0]"     LOC = P110  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_DATA[7]"     LOC = P114  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

    NET "COMM_HUNDREDS"       LOC = P124  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "COMM_DECS"           LOC = P121  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "COMM_ONES"           LOC = P120  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

##############################################################################################
#                                        LED
##############################################################################################

    NET "OVERFLOW"            LOC = P46   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(1)"         LOC = P47   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(2)"         LOC = P48   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(3)"         LOC = P49   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(4)"         LOC = P50   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(5)"         LOC = P51   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(6)"         LOC = P54   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    #NET "OUT_BUS(7)"         LOC = P55   | 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 "RESET"                 LOC = P78   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "CALCULATE"             LOC = P77   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

Згенерував BIT файл для цільової FPGA.

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



Висновок

На цій лабораторній роботі я навчився на базі стенда Elbert V2 – Spartan 3A FPGA, реалізовувати цифровий автомат для обчислення значення виразу.