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 №0

Виконав:
ст. гр. КІ-201
Абросімов А.С.

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 (ALU) повинен виконувати за один такт одну операцію), та реалізованим згідно наступної структурної схеми (Малюнок 1).

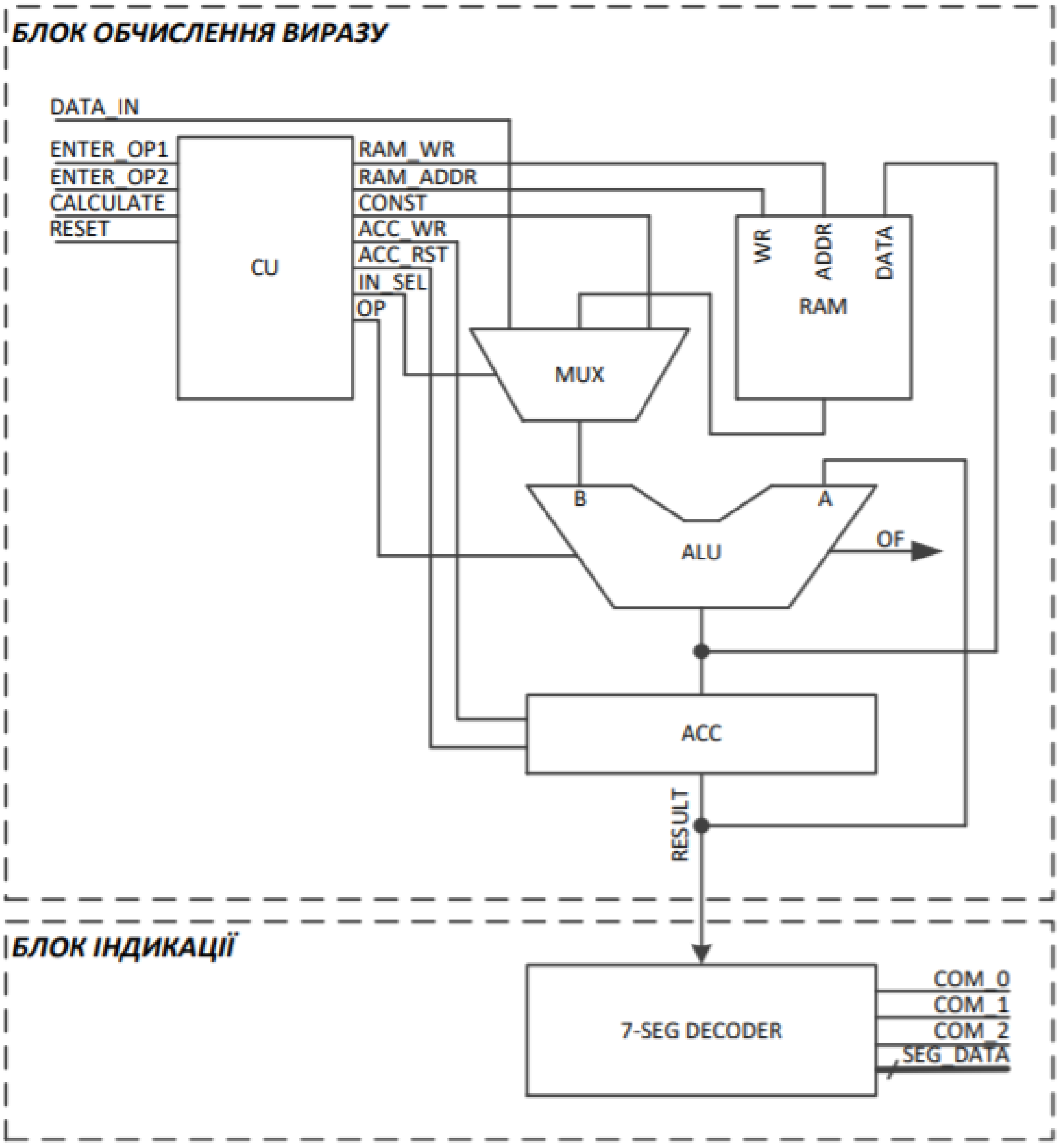


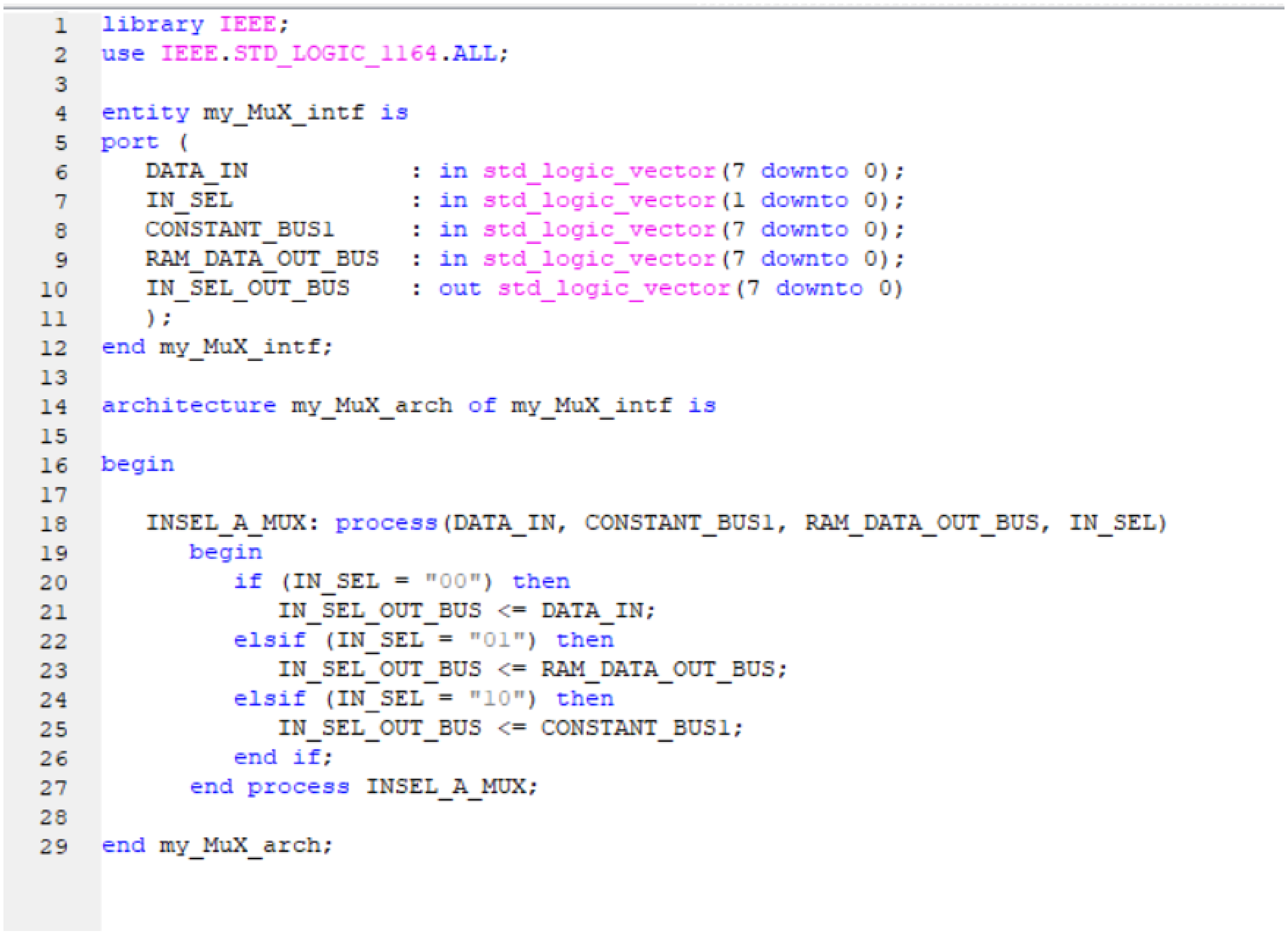
Рис. 1 - Структурна схема автомата.

Завдання:

ВАРІАНТ	ВИРАЗ
<u>0</u>	((OP1 – OP2) + 4) << OP2

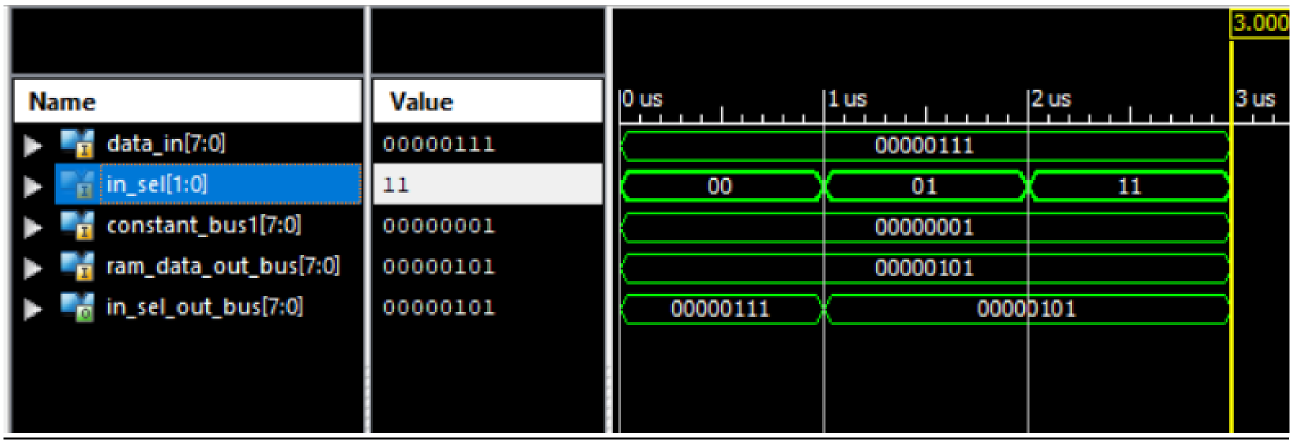
Виконання:

1) Створив файл MUX.vhd.



```vhdl
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_BUS1       : 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_BUS1, 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;
        elsif (IN_SEL = "10") then
          IN_SEL_OUT_BUS <= CONSTANT_BUS1;
        end if;
      end process INSEL_A_MUX;

end my_MuX_arch;
```

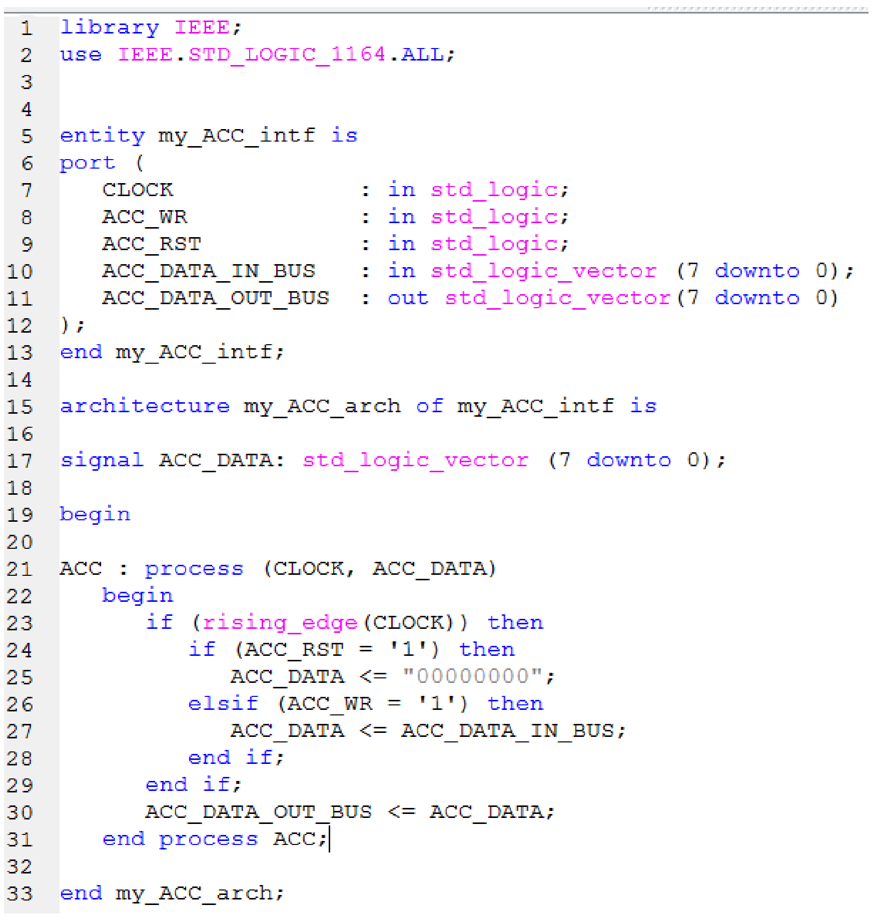
Рис.1. Реалізація мультиплексора у файлі MUX.vhd



Name	Value
data_in[7:0]	00000111
in_sel[1:0]	11
constant_bus1[7:0]	00000001
ram_data_out_bus[7:0]	00000101
in_sel_out_bus[7:0]	00000101

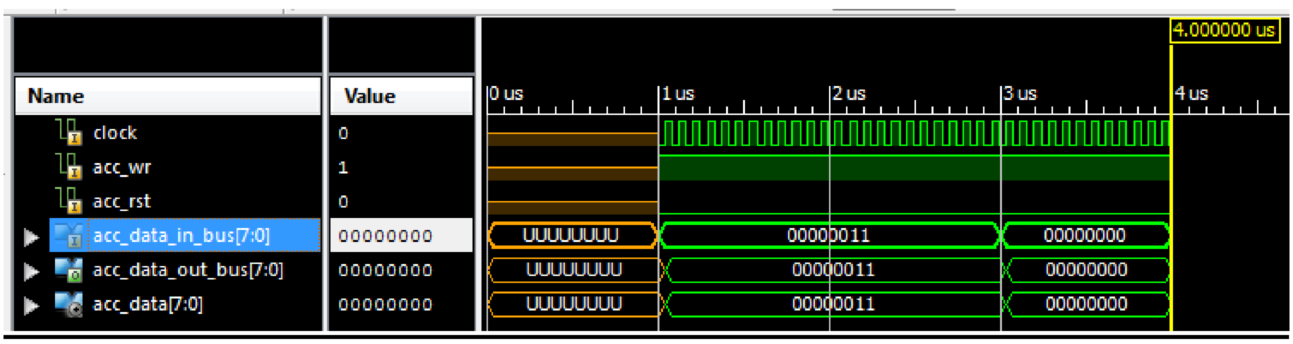
Симуляція роботи мультиплексора

2) Створив файл ACC.vhd.



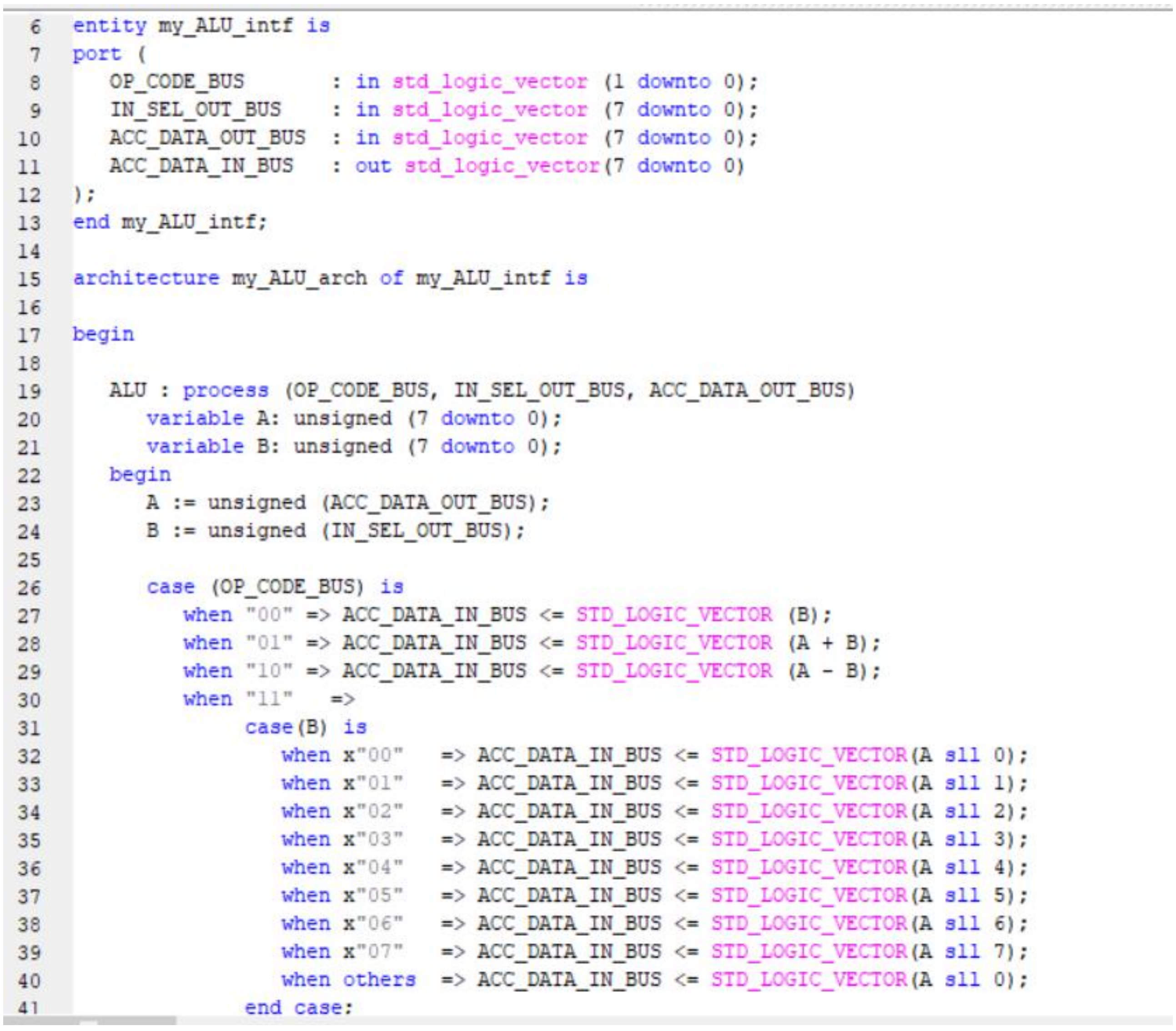
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


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;
```

Рис.2. Реалізація регістра ACC у файлі ACC.vhd



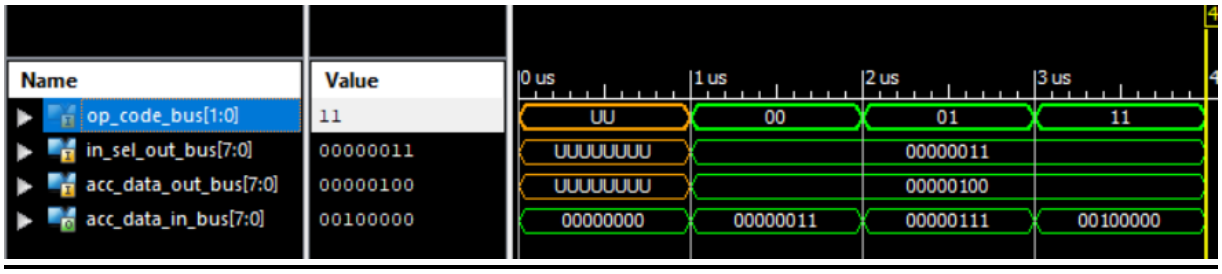
Симуляція роботи регістра

3) Визначив набір необхідних операцій для виконання виразу згідно свого варіанту і реалізував АЛП(ALU) у файлі ALU.vhd з підтримкою визначеного набору операцій (рис.3).



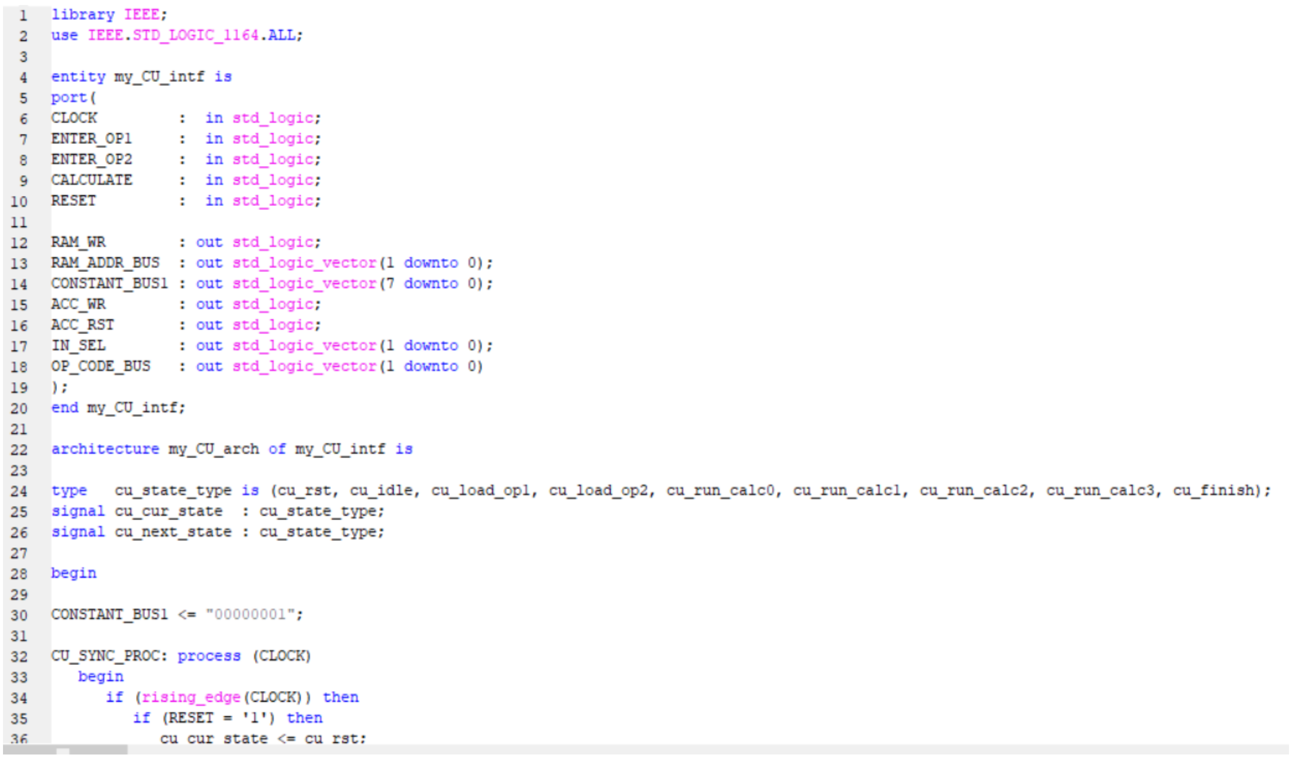
```
entity my_ALU_intf is
port (
   OP_CODE_BUS       : in std_logic_vector (1 downto 0);
   IN_SEL_OUT_BUS    : in std_logic_vector (7 downto 0);
   ACC_DATA_OUT_BUS  : in std_logic_vector (7 downto 0);
   ACC_DATA_IN_BUS   : out std_logic_vector(7 downto 0)
);
end my_ALU_intf;

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"   =>
              case(B) is
                 when x"00"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 0);
                 when x"01"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 1);
                 when x"02"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 2);
                 when x"03"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 3);
                 when x"04"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 4);
                 when x"05"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 5);
                 when x"06"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 6);
                 when x"07"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 7);
                 when others  => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 0);
              end case;
```

Рис.3. Реалізація АЛП(ALU) у файлі ALU.vhd з підтримкою визначеного набору операцій.

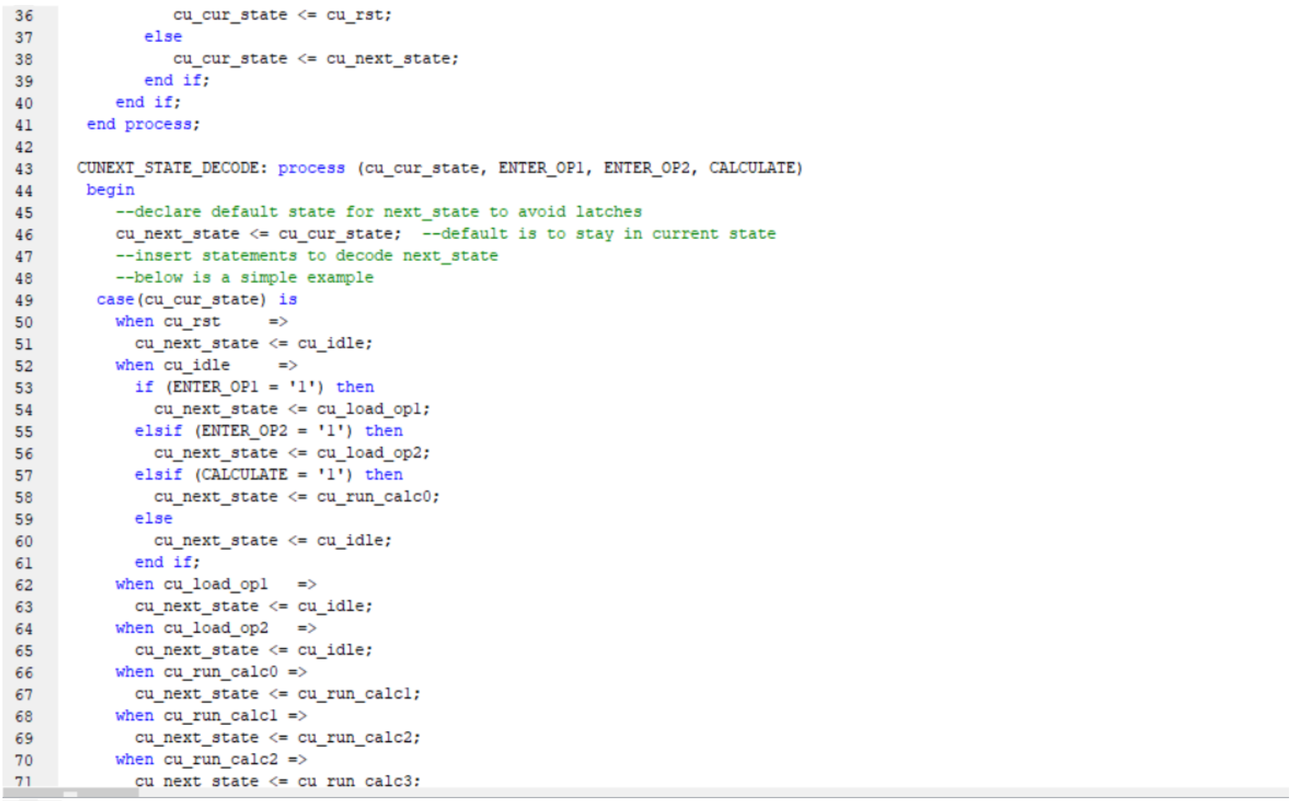


Симуляція роботи АЛП

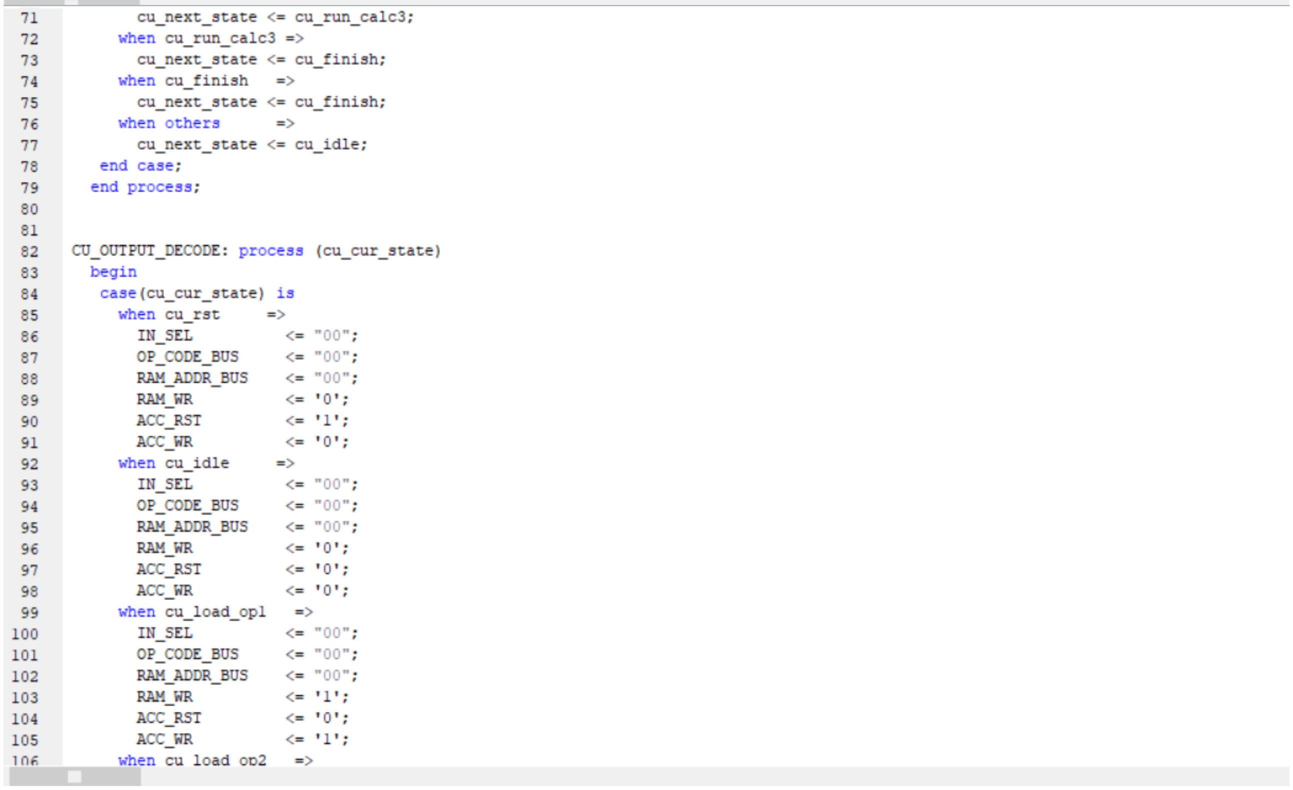
4) Визначив множину станів і реалізував пристрій керування (CU) у файлі CU.vhd.



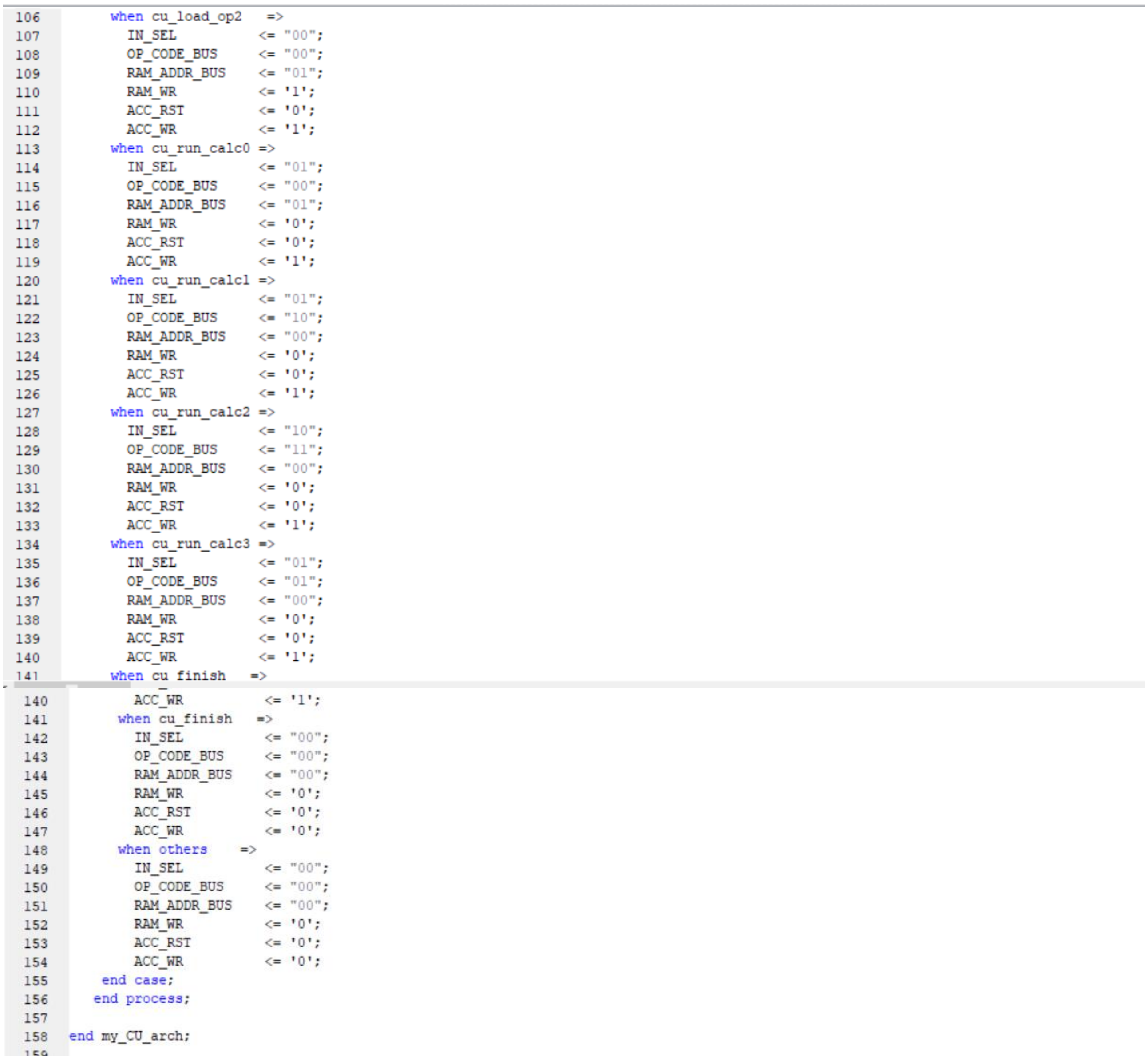
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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_BUS1  : 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_BUS1 <= "00000001";

CU_SYNC_PROC: process (CLOCK)
   begin
      if (rising_edge(CLOCK)) then
         if (RESET = '1') then
            cu_cur_state <= cu_rst;
```



```vhdl
            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;
```



```vhdl
        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    =>
```

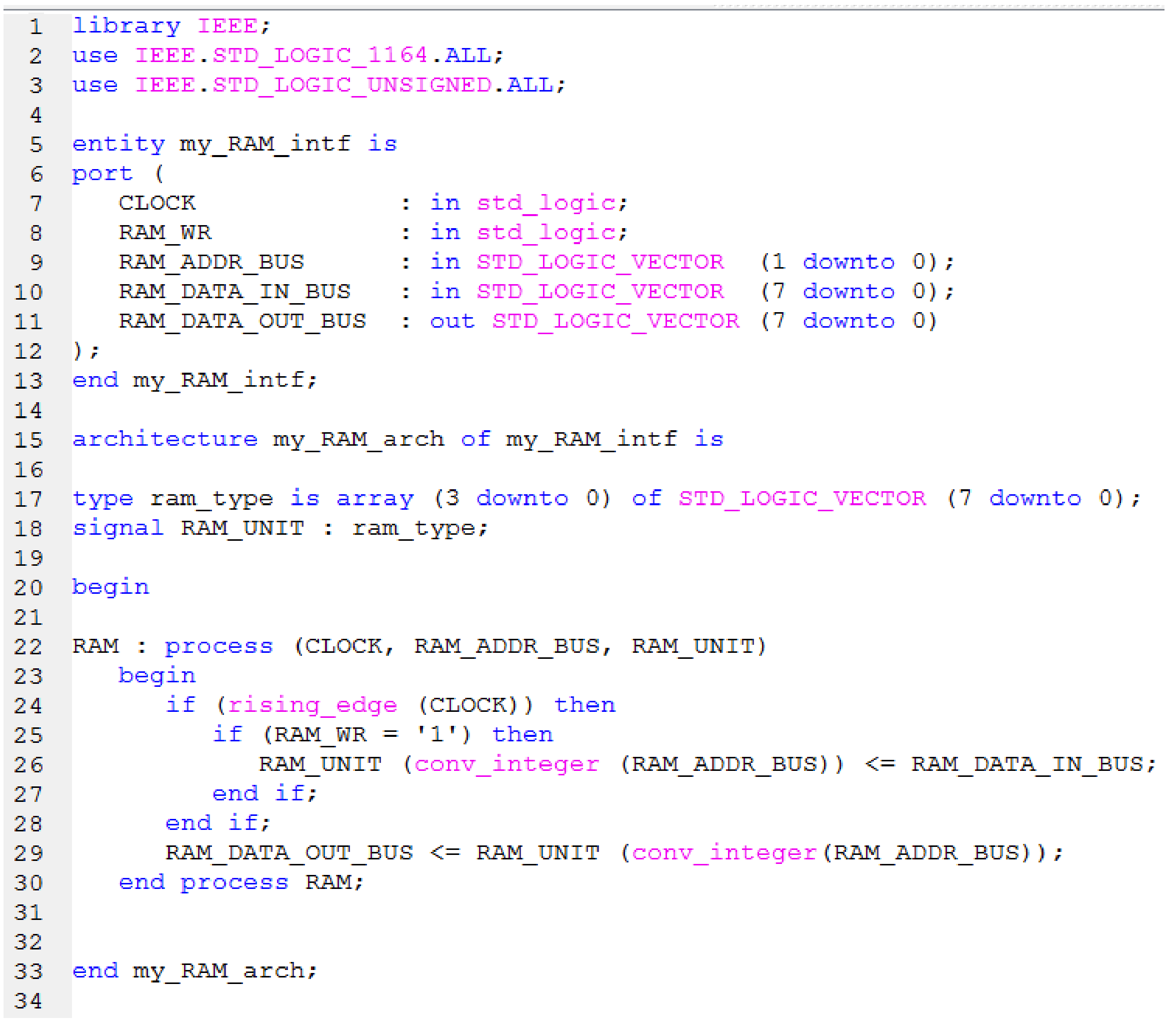


```vhdl
        when cu_load_op2   =>
          IN_SEL          <= "00";
          OP_CODE_BUS     <= "00";
          RAM_ADDR_BUS    <= "01";
          RAM_WR          <= '1';
          ACC_RST         <= '0';
          ACC_WR          <= '1';
        when cu_run_calc0 =>
          IN_SEL          <= "01";
          OP_CODE_BUS     <= "00";
          RAM_ADDR_BUS    <= "01";
          RAM_WR          <= '0';
          ACC_RST         <= '0';
          ACC_WR          <= '1';
        when cu_run_calc1 =>
          IN_SEL          <= "01";
          OP_CODE_BUS     <= "10";
          RAM_ADDR_BUS    <= "00";
          RAM_WR          <= '0';
          ACC_RST         <= '0';
          ACC_WR          <= '1';
        when cu_run_calc2 =>
          IN_SEL          <= "10";
          OP_CODE_BUS     <= "11";
          RAM_ADDR_BUS    <= "00";
          RAM_WR          <= '0';
          ACC_RST         <= '0';
          ACC_WR          <= '1';
        when cu_run_calc3 =>
          IN_SEL          <= "01";
          OP_CODE_BUS     <= "01";
          RAM_ADDR_BUS    <= "00";
          RAM_WR          <= '0';
          ACC_RST         <= '0';
          ACC_WR          <= '1';
        when cu_finish   =>
```

```vhdl
          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;
```

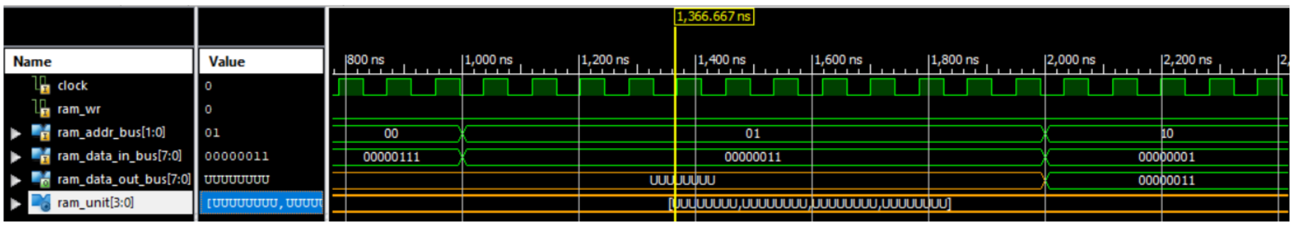
Рис.4. Реалізація пристрою керування (CU) у файлі CU.vhd

5) Створив файл RAM.vhd.



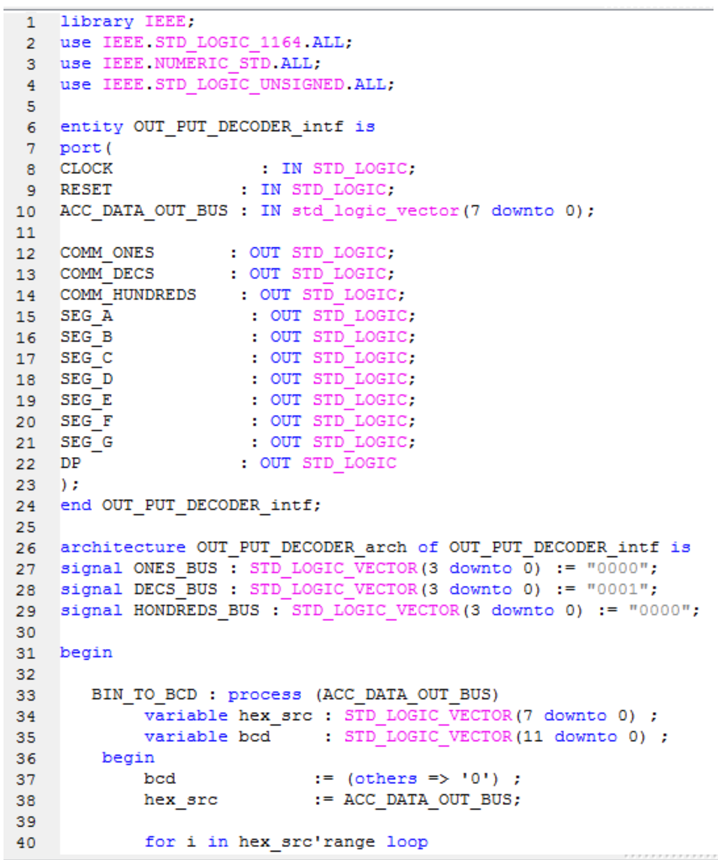
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

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;

```

Рис.5. Реалізація пам’яті пристрою (RAM) у файлі RAM.vhd

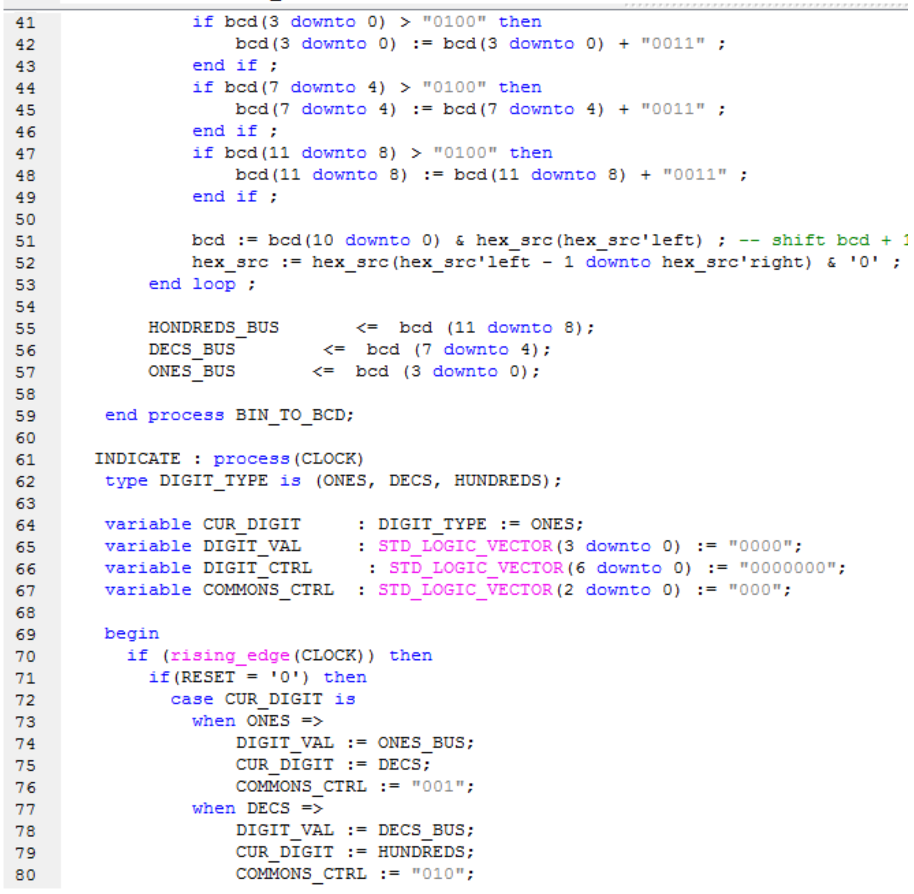


Симуляція роботи RAM

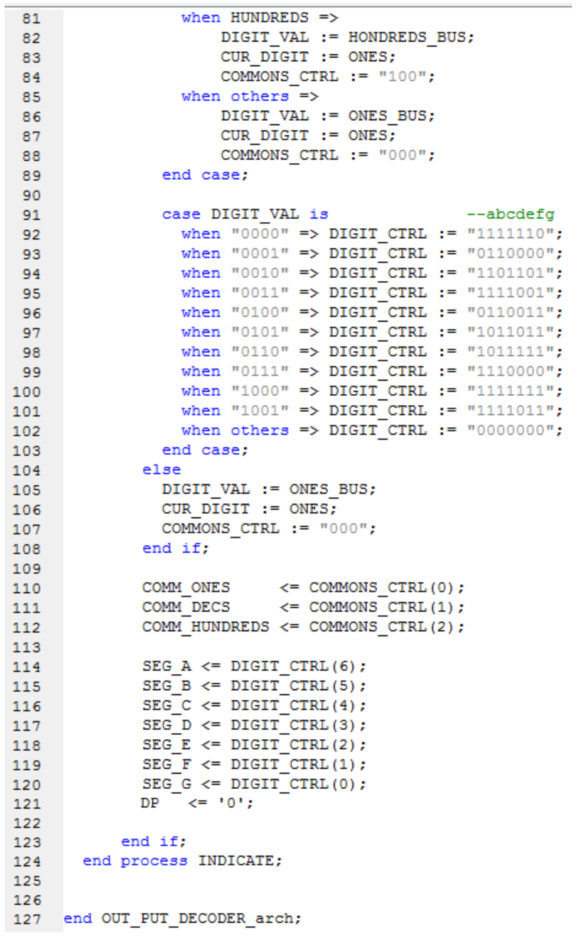
5) Створив файл OUT_PUT_DECODER.vhd і реалізував в ньому блок індикації (7-SEG DECODER).



```vhdl
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
```



```vhdl
            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
            hex_src := hex_src(hex_src'left - 1 downto hex_src'right) & '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;
```

Рис.6. Реалізація блоку індикації (7-SEG DECODER) в файлі OUT_PUT_DECODER.vhd

6) Згенерував символи для імплементованих компонентів і створив схему у файлі Top_level.sch.

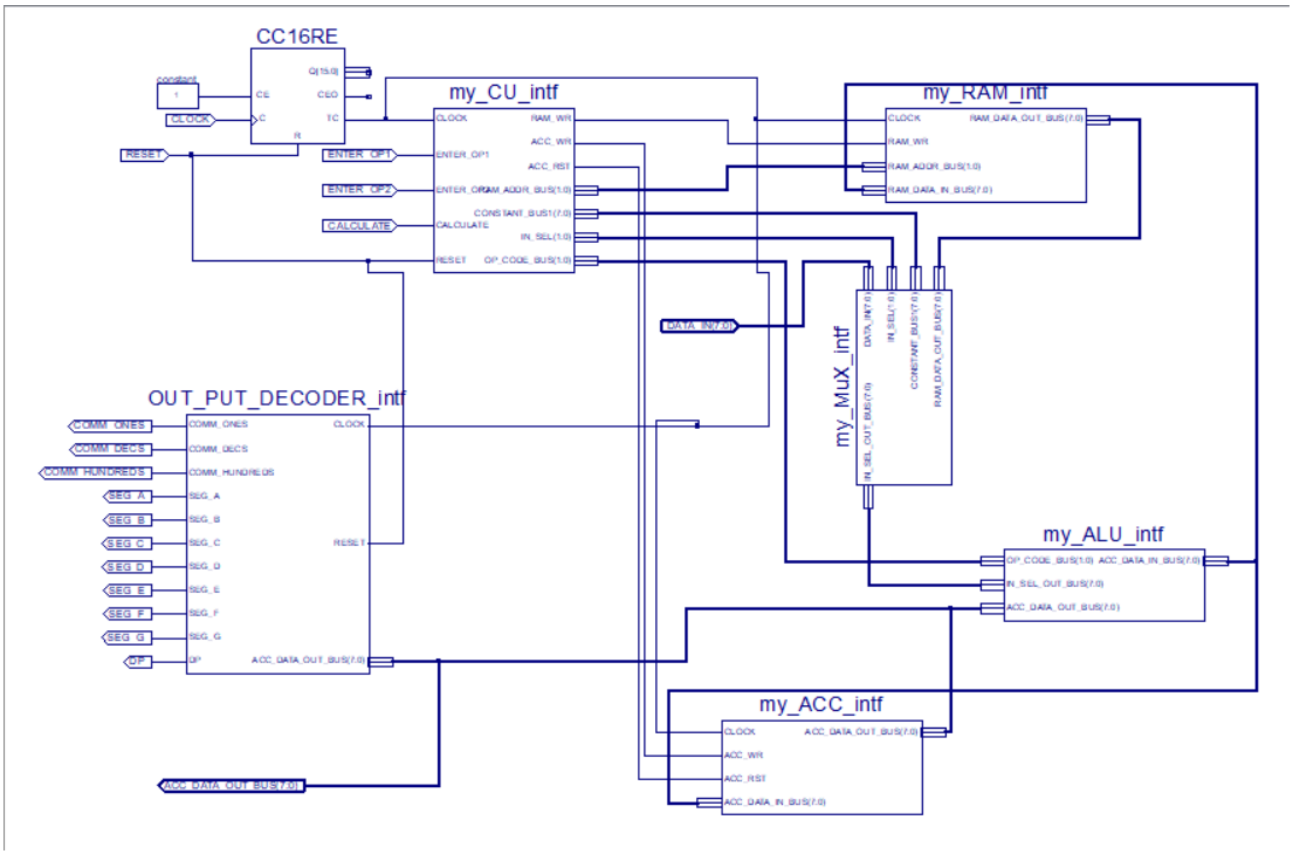
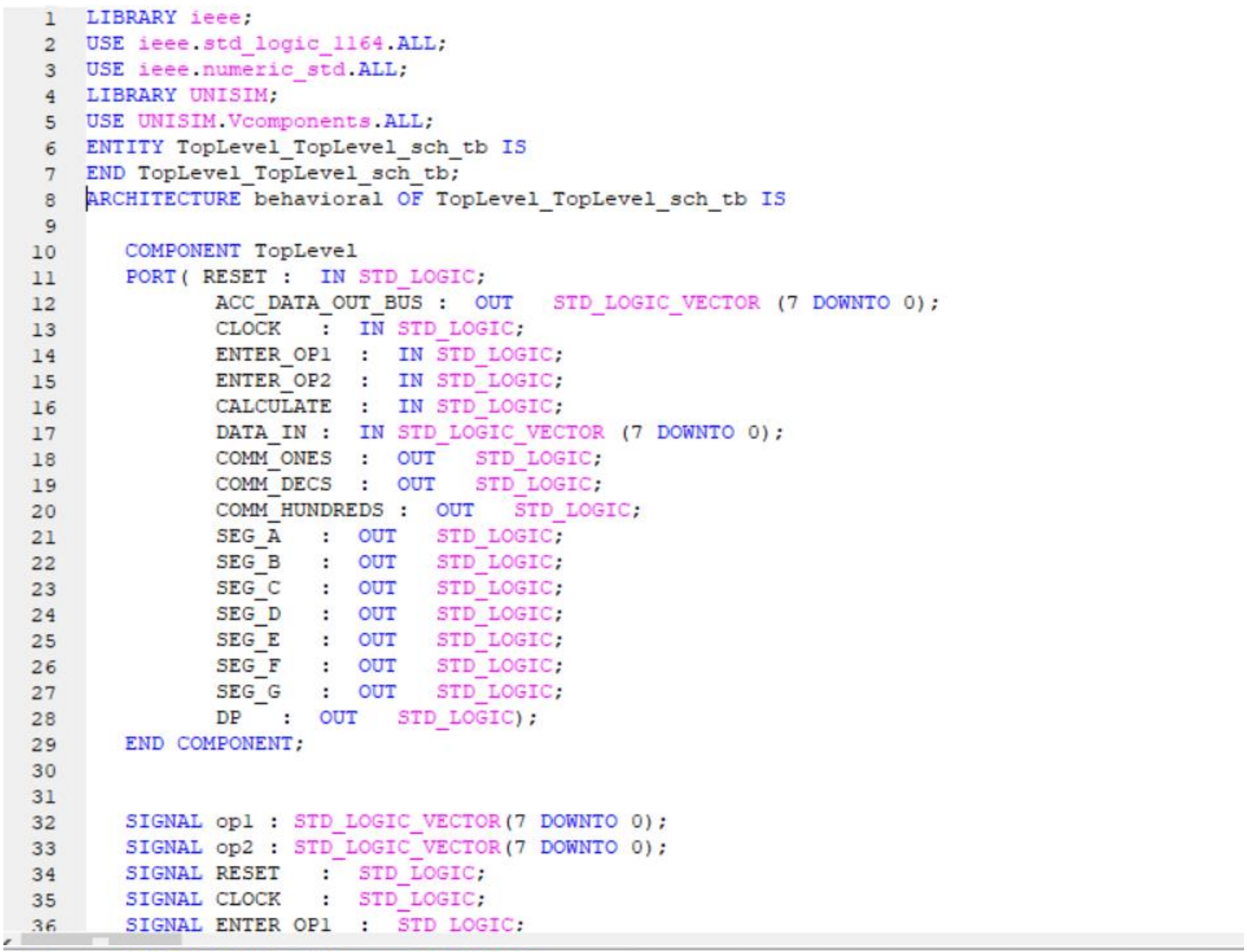
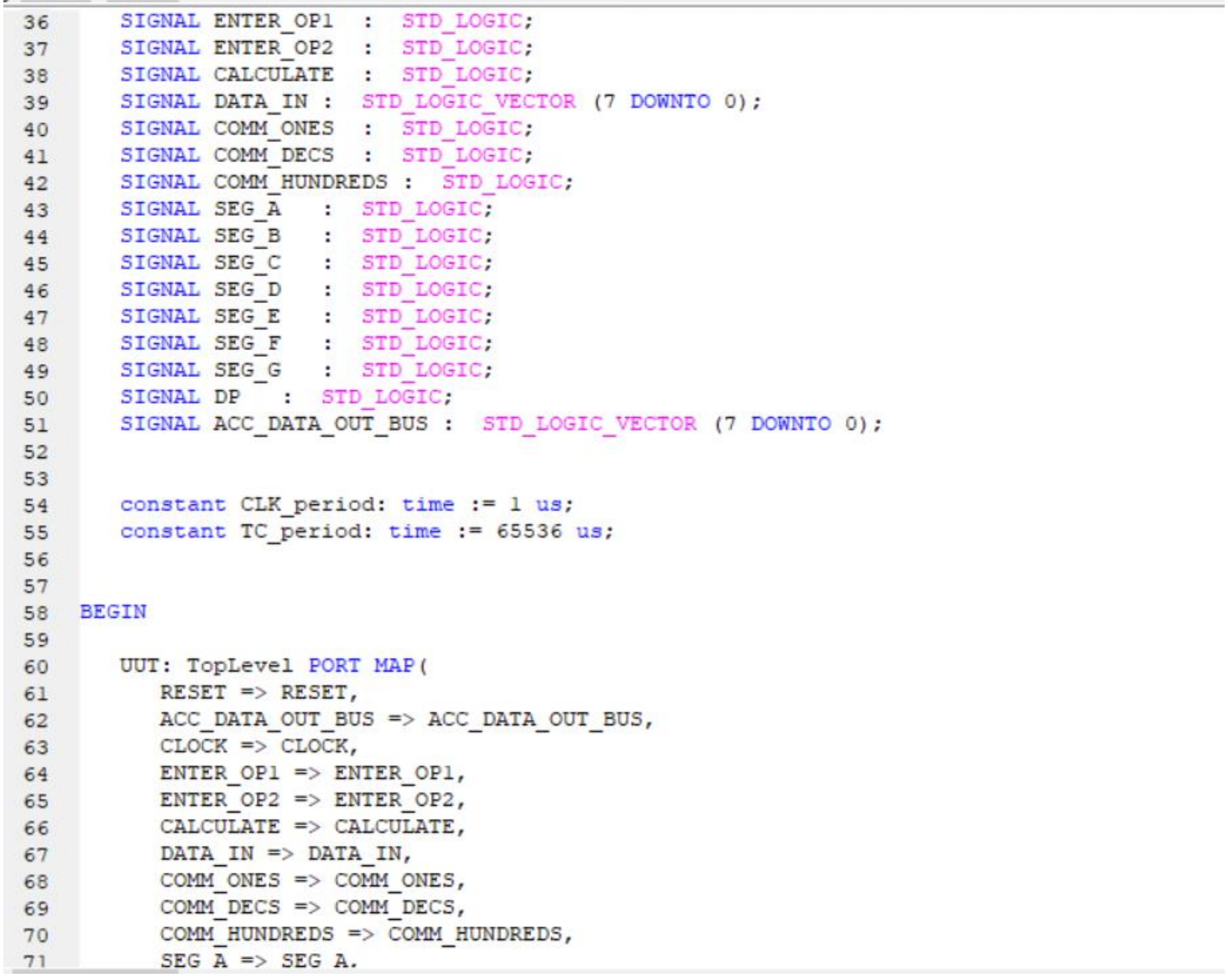


Рис.7. Схема з використанням імплементованих компонентів.

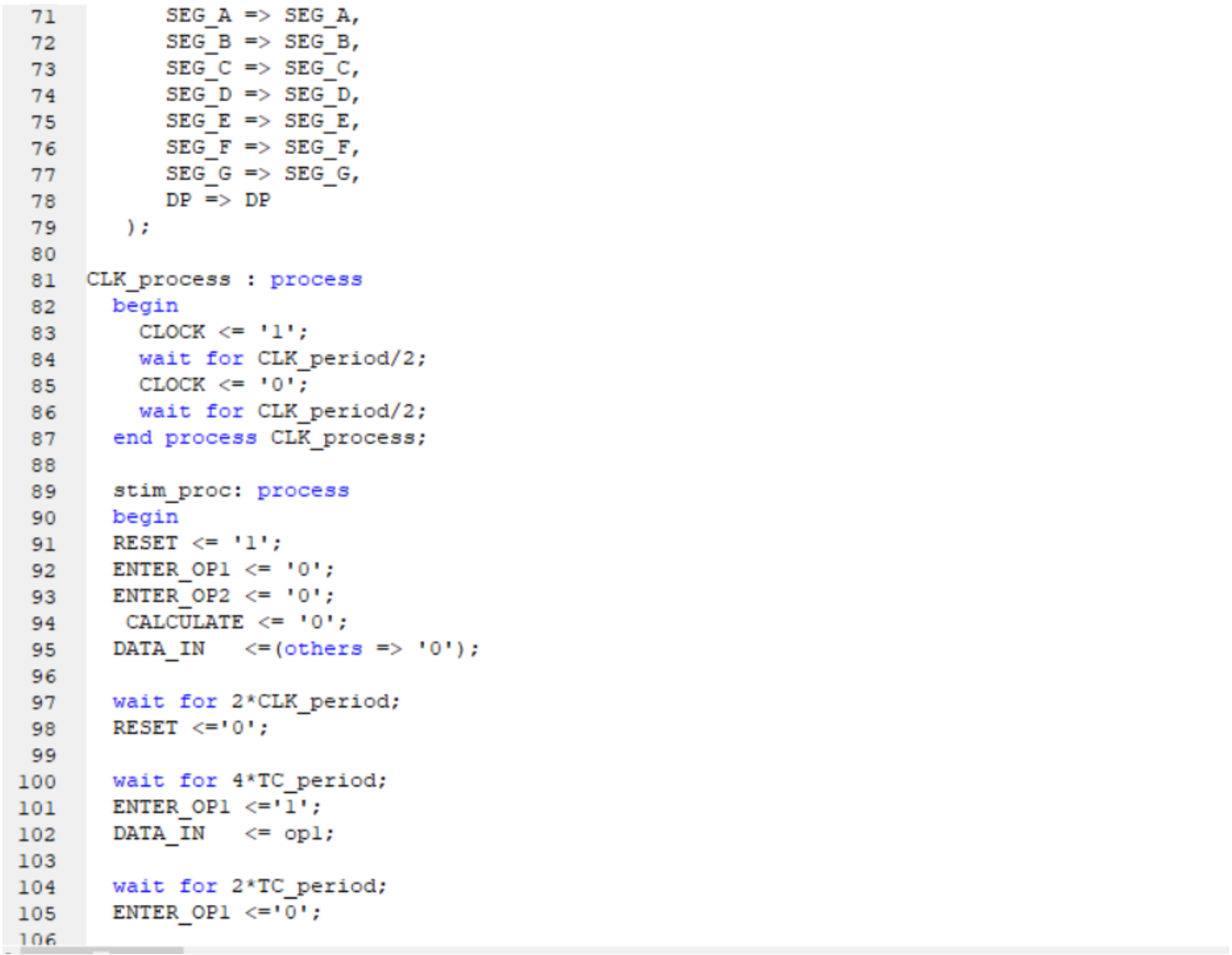
7) *testbench.vhd.*



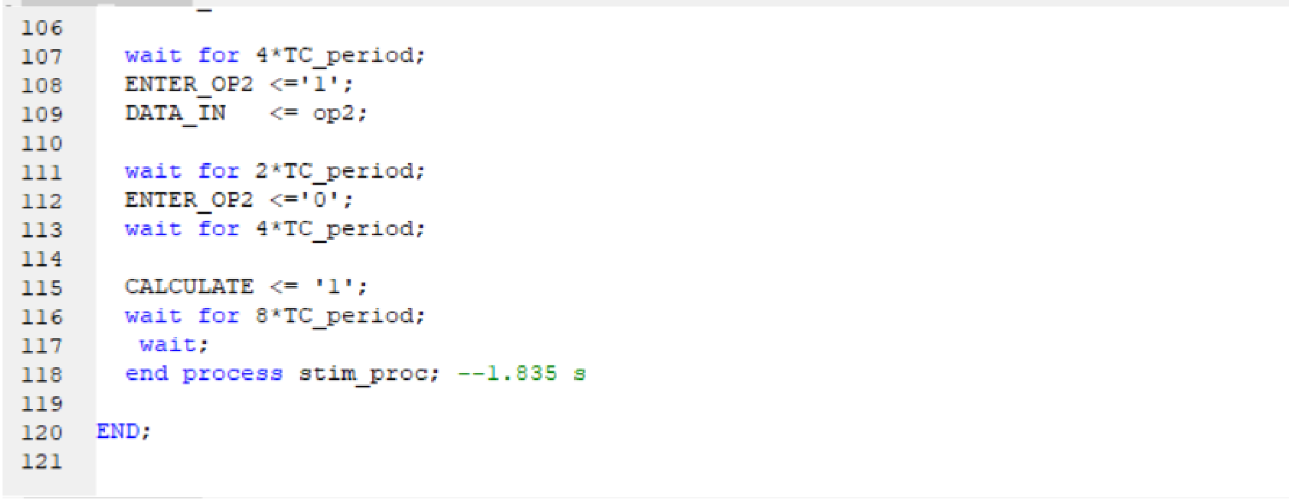
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY TopLevel_TopLevel_sch_tb IS
END TopLevel_TopLevel_sch_tb;
ARCHITECTURE behavioral OF TopLevel_TopLevel_sch_tb IS

   COMPONENT TopLevel
   PORT( RESET  :  IN STD_LOGIC;
          ACC_DATA_OUT_BUS :  OUT   STD_LOGIC_VECTOR (7 DOWNTO 0);
          CLOCK    :  IN STD_LOGIC;
          ENTER_OP1  :  IN STD_LOGIC;
          ENTER_OP2  :  IN STD_LOGIC;
          CALCULATE  :  IN STD_LOGIC;
          DATA_IN :  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 COMPONENT;


   SIGNAL op1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL op2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL RESET    :  STD_LOGIC;
   SIGNAL CLOCK    :  STD_LOGIC;
   SIGNAL ENTER_OP1  :  STD_LOGIC;
```



```
   SIGNAL ENTER_OP1   :  STD_LOGIC;
   SIGNAL ENTER_OP2   :  STD_LOGIC;
   SIGNAL CALCULATE   :  STD_LOGIC;
   SIGNAL DATA_IN :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL COMM_ONES   :  STD_LOGIC;
   SIGNAL COMM_DECS   :  STD_LOGIC;
   SIGNAL COMM_HUNDREDS :  STD_LOGIC;
   SIGNAL SEG_A    :  STD_LOGIC;
   SIGNAL SEG_B    :  STD_LOGIC;
   SIGNAL SEG_C    :  STD_LOGIC;
   SIGNAL SEG_D    :  STD_LOGIC;
   SIGNAL SEG_E    :  STD_LOGIC;
   SIGNAL SEG_F    :  STD_LOGIC;
   SIGNAL SEG_G    :  STD_LOGIC;
   SIGNAL DP    :  STD_LOGIC;
   SIGNAL ACC_DATA_OUT_BUS :  STD_LOGIC_VECTOR (7 DOWNTO 0);


   constant CLK_period: time := 1 us;
   constant TC_period: time := 65536 us;


BEGIN

   UUT: TopLevel PORT MAP(
      RESET => RESET,
      ACC_DATA_OUT_BUS => ACC_DATA_OUT_BUS,
      CLOCK => CLOCK,
      ENTER_OP1 => ENTER_OP1,
      ENTER_OP2 => ENTER_OP2,
      CALCULATE => CALCULATE,
      DATA_IN => DATA_IN,
      COMM_ONES => COMM_ONES,
      COMM_DECS => COMM_DECS,
      COMM_HUNDREDS => COMM_HUNDREDS,
      SEG_A => SEG_A.
```

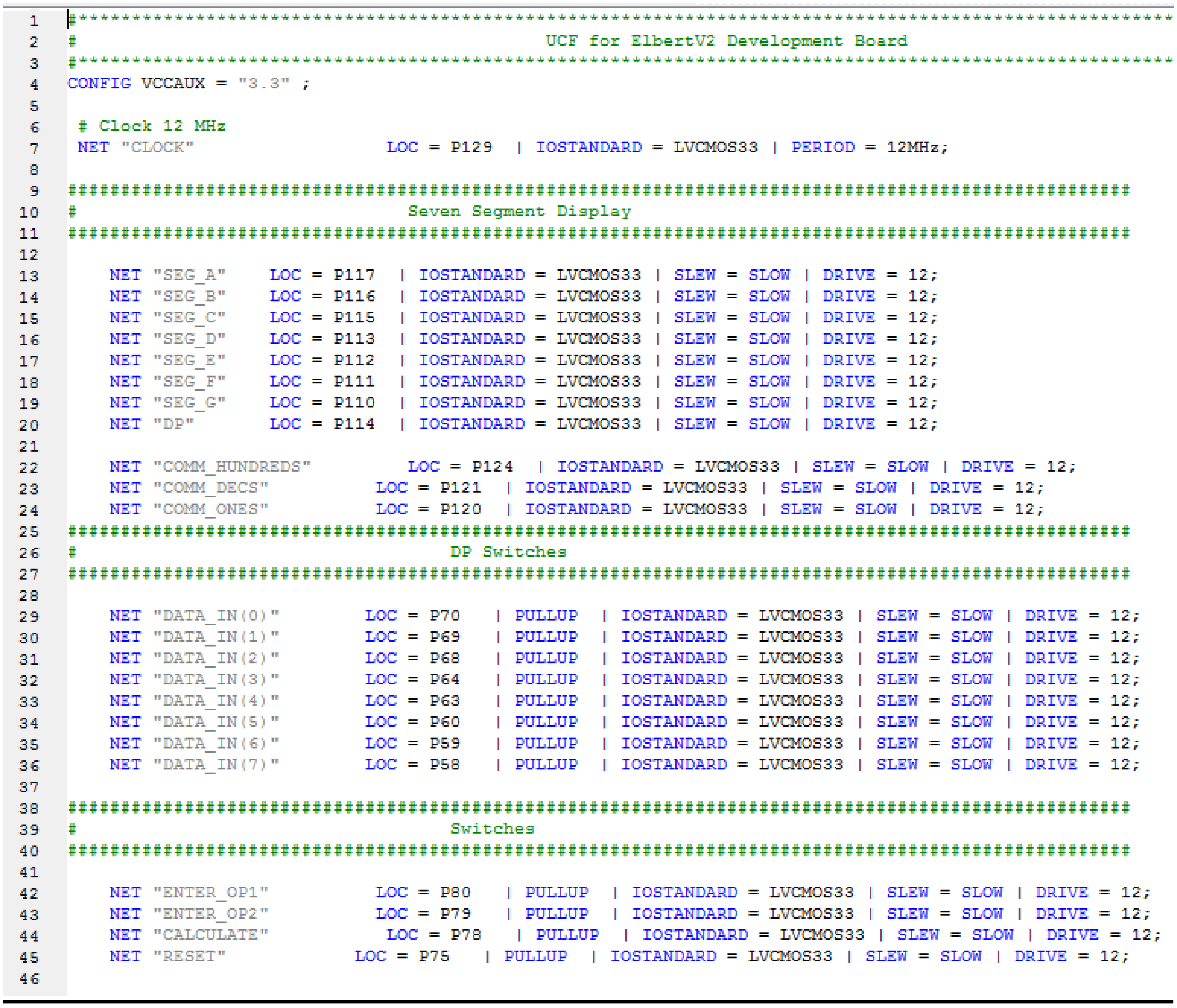


```vhdl
        SEG_A => SEG_A,
        SEG_B => SEG_B,
        SEG_C => SEG_C,
        SEG_D => SEG_D,
        SEG_E => SEG_E,
        SEG_F => SEG_F,
        SEG_G => SEG_G,
        DP => DP
    );

CLK_process : process
  begin
    CLOCK <= '1';
    wait for CLK_period/2;
    CLOCK <= '0';
    wait for CLK_period/2;
  end process CLK_process;

  stim_proc: process
  begin
  RESET <= '1';
  ENTER_OP1 <= '0';
  ENTER_OP2 <= '0';
   CALCULATE <= '0';
  DATA_IN   <=(others => '0');

  wait for 2*CLK_period;
  RESET <='0';

  wait for 4*TC_period;
  ENTER_OP1 <='1';
  DATA_IN   <= op1;

  wait for 2*TC_period;
  ENTER_OP1 <='0';
```



```vhdl

  wait for 4*TC_period;
  ENTER_OP2 <='1';
  DATA_IN   <= op2;

  wait for 2*TC_period;
  ENTER_OP2 <='0';
  wait for 4*TC_period;

  CALCULATE <= '1';
  wait for 8*TC_period;
   wait;
  end process stim_proc; --1.835 s

END;

```

8) Constraints



```
#*******************************************************************************************************
#                                            UCF for ElbertV2 Development Board
#*******************************************************************************************************
CONFIG VCCAUX = "3.3" ;

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

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

    NET "SEG_A"    LOC = P117  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_B"    LOC = P116  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_C"    LOC = P115  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_D"    LOC = P113  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_E"    LOC = P112  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_F"    LOC = P111  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_G"    LOC = P110  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DP"       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;
####################################################################################################
#                                   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;

```

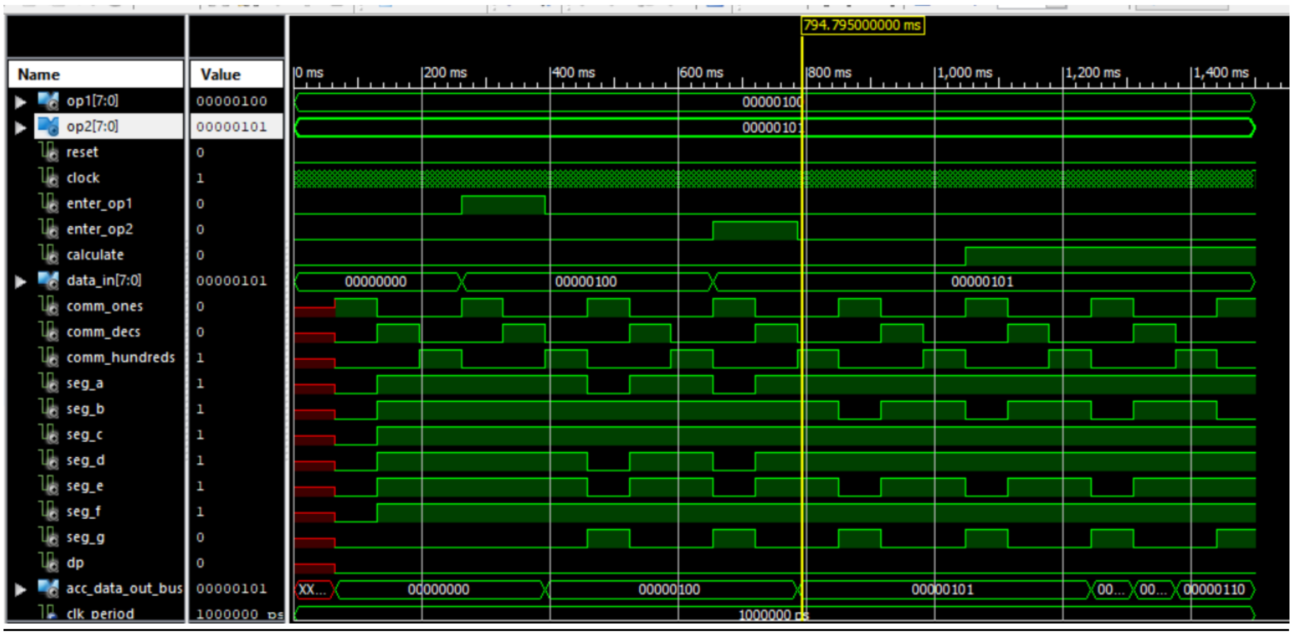
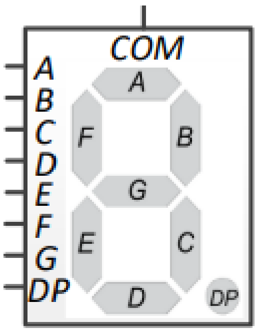


Рис.8. Часова діаграма згідно методичних вказівок.



Перевірка:

((OP1 – OP2) + 4) << OP2

OP1 => 0000 0001

OP2 => 0000 0001

((0000 0001– 0000 0001) + 0000 0100) << 0000 0001

0000 0001– 0000 0001 = 0000 0000

0000 0001 + 0000 0001= 0000 0010

0000 0010 << 0000 0001 = 0000 1000

Висновок:

На цій лабораторній роботі реалізував цифровий автомат для обчислення значення виразу та симулював його роботу.