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 №9

Виконав:
ст. гр. КІ-201
Іванюк О.О.

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).

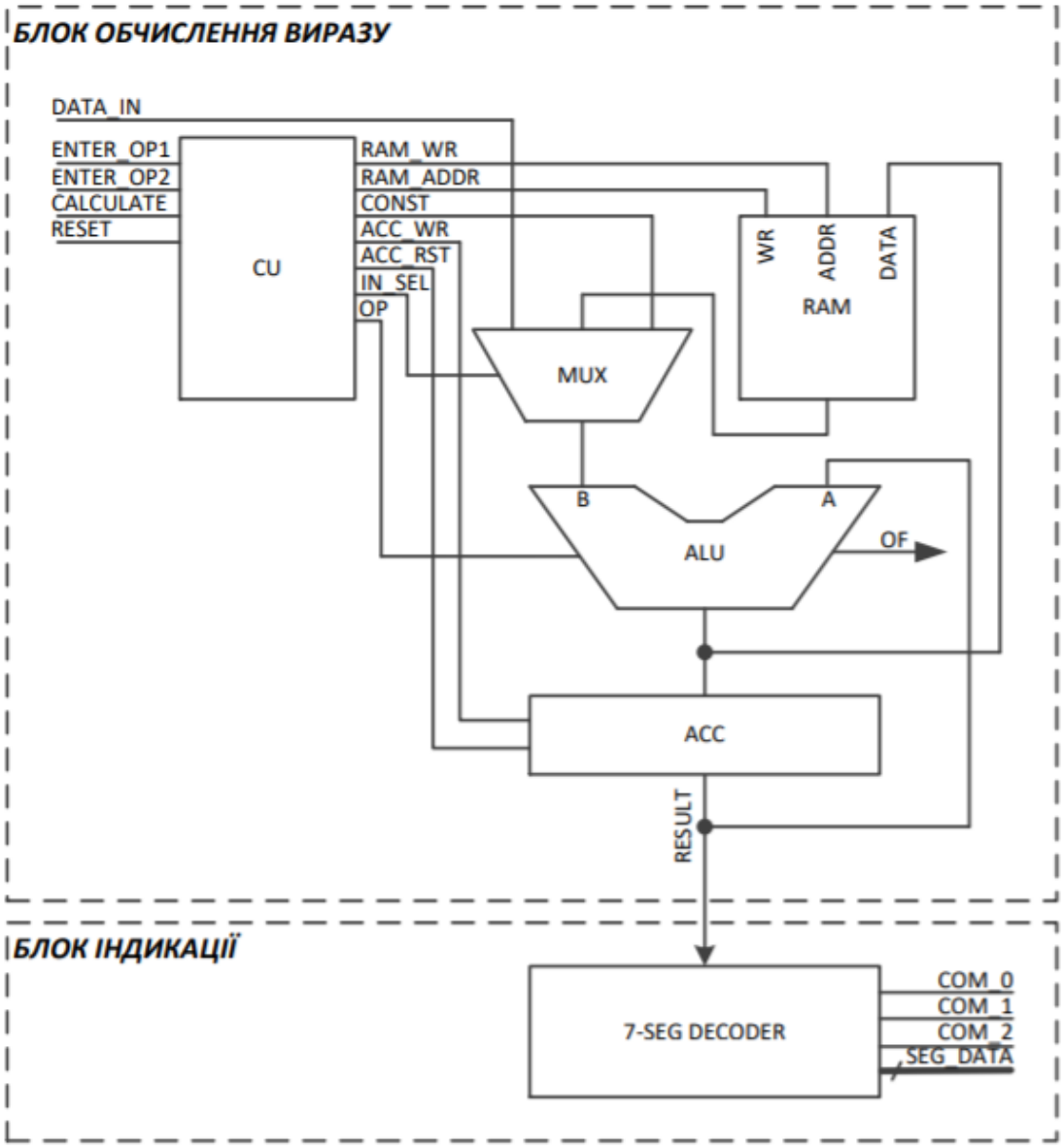


Рис. 1 - Структурна схема автомата.

Завдання:

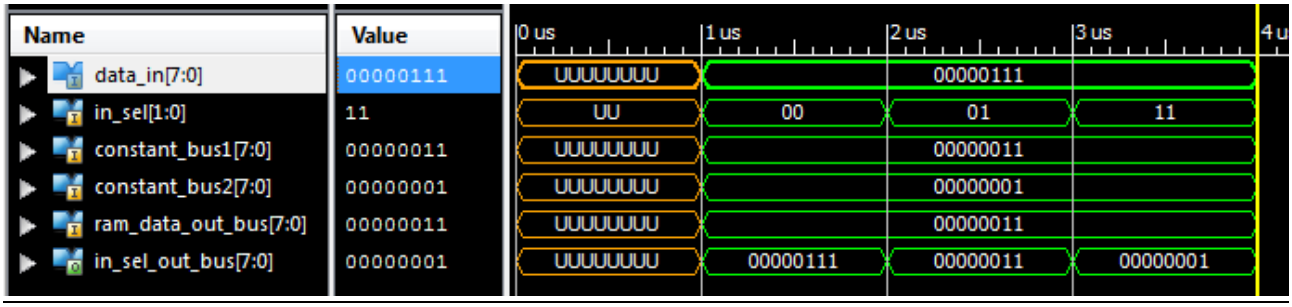
ВАРІАНТ	ВИРАЗ
9	((OP2 - 4) + OP1) or 2

Виконання:

1) Створив файл MUX.vhd.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity my_MuX_intf is
5  port (
6     DATA_IN           : in std_logic_vector(7 downto 0);
7     IN_SEL            : in std_logic_vector(1 downto 0);
8     CONSTANT_BUS1     : in std_logic_vector(7 downto 0);
9     CONSTANT_BUS2     : in std_logic_vector(7 downto 0);
10    RAM_DATA_OUT_BUS  : in std_logic_vector(7 downto 0);
11    IN_SEL_OUT_BUS    : out std_logic_vector(7 downto 0)
12    );
13 end my_MuX_intf;
14
15 architecture my_MuX_arch of my_MuX_intf is
16
17 begin
18
19    INSEL_A_MUX: process(DATA_IN, CONSTANT_BUS1, CONSTANT_BUS2, RAM_DATA_OUT_BUS, IN_SEL)
20       begin
21          if (IN_SEL = "00") then
22             IN_SEL_OUT_BUS <= DATA_IN;
23          elsif (IN_SEL = "01") then
24             IN_SEL_OUT_BUS <= RAM_DATA_OUT_BUS;
25          elsif (IN_SEL = "10") then
26             IN_SEL_OUT_BUS <= CONSTANT_BUS1;
27          else
28             IN_SEL_OUT_BUS <= CONSTANT_BUS2;
29          end if;
30       end process INSEL_A_MUX;
31
32
33 end my_MuX_arch;
```

Рис.1. Реалізація мультиплексора у файлі MUX.vhd

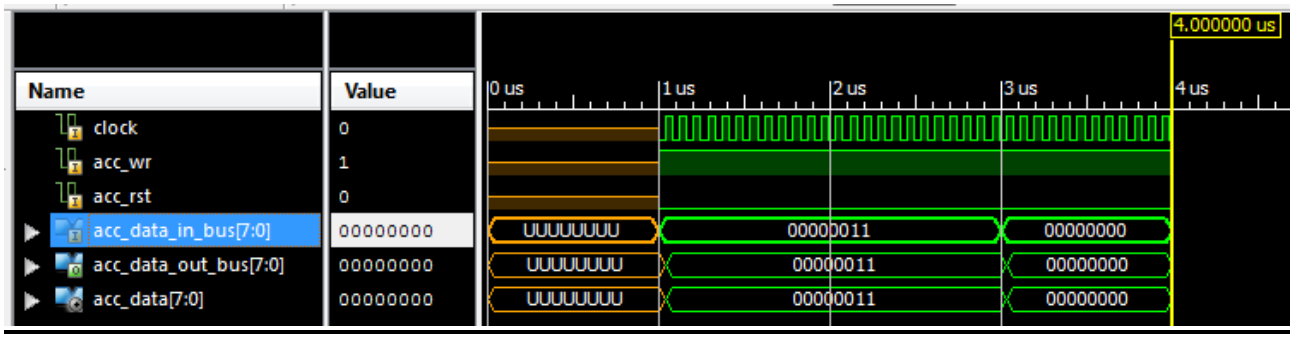


Симуляція роботи мультиплексора

2) Створив файл ACC.vhd.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity my_ACC_intf is
6  port (
7     CLOCK              : in std_logic;
8     ACC_WR             : in std_logic;
9     ACC_RST            : in std_logic;
10    ACC_DATA_IN_BUS    : in std_logic_vector (7 downto 0);
11    ACC_DATA_OUT_BUS   : out std_logic_vector(7 downto 0)
12 );
13 end my_ACC_intf;
14
15 architecture my_ACC_arch of my_ACC_intf is
16
17 signal ACC_DATA: std_logic_vector (7 downto 0);
18
19 begin
20
21 ACC : process (CLOCK, ACC_DATA)
22    begin
23       if (rising_edge(CLOCK)) then
24          if (ACC_RST = '1') then
25             ACC_DATA <= "00000000";
26          elsif (ACC_WR = '1') then
27             ACC_DATA <= ACC_DATA_IN_BUS;
28          end if;
29       end if;
30       ACC_DATA_OUT_BUS <= ACC_DATA;
31    end process ACC;
32
33 end my_ACC_arch;
```

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

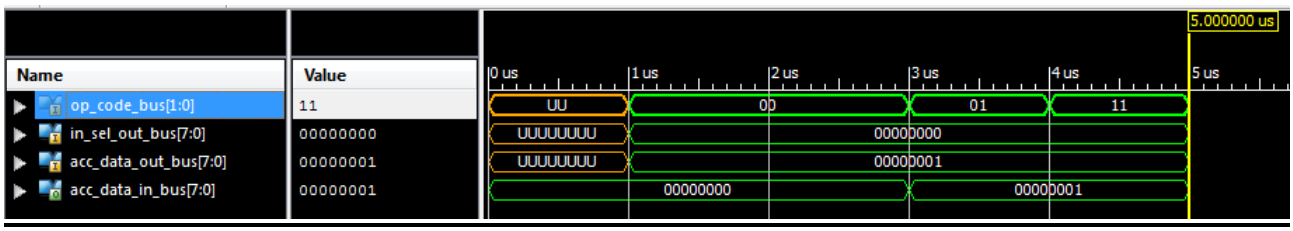


Симуляція роботи регістра

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).

```
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 my_ALU_intf is
7  port (
8     OP_CODE_BUS       : in std_logic_vector (1 downto 0);
9     IN_SEL_OUT_BUS    : in std_logic_vector (7 downto 0);
10    ACC_DATA_OUT_BUS  : in std_logic_vector (7 downto 0);
11    ACC_DATA_IN_BUS   : out std_logic_vector(7 downto 0)
12 );
13 end my_ALU_intf;
14
15 architecture my_ALU_arch of my_ALU_intf is
16
17 begin
18
19    ALU : process (OP_CODE_BUS, IN_SEL_OUT_BUS, ACC_DATA_OUT_BUS)
20       variable A: unsigned (7 downto 0);
21       variable B: unsigned (7 downto 0);
22    begin
23       A := unsigned (ACC_DATA_OUT_BUS);
24       B := unsigned (IN_SEL_OUT_BUS);
25
26       case (OP_CODE_BUS) is
27          when "00" => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR (B);
28          when "01" => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR (A + B);
29          when "10" => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR (A - B);
30          when "11" => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR (A or B);
31          when others => ACC_DATA_IN_BUS <= "00000000";
32       end case;
33
34    end process ALU;
35
36 end my_ALU_arch;
```

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);
CONSTANT_BUS2 : 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 <= "00000100";
CONSTANT_BUS2 <= "00000010";

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;
```

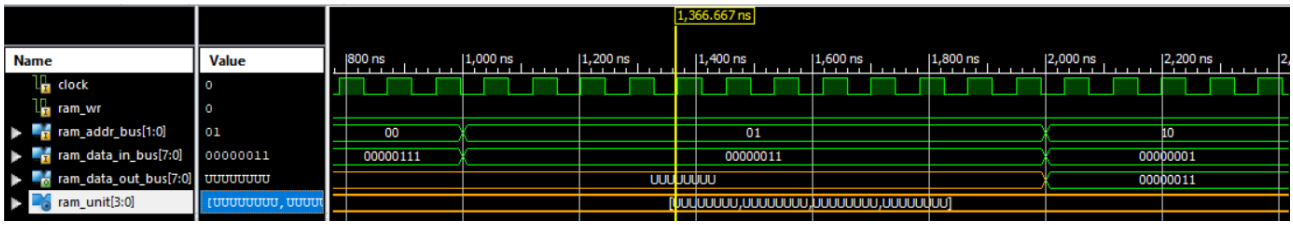
```
81      end process;
82
83
84    CU_OUTPUT_DECODE: process (cu_cur_state)
85      begin
86       case(cu_cur_state) is
87         when cu_rst     =>
88           IN_SEL          <= "00";
89           OP_CODE_BUS     <= "00";
90           RAM_ADDR_BUS    <= "00";
91           RAM_WR          <= '0';
92           ACC_RST         <= '1';
93           ACC_WR          <= '0';
94         when cu_idle     =>
95           IN_SEL          <= "00";
96           OP_CODE_BUS     <= "00";
97           RAM_ADDR_BUS    <= "00";
98           RAM_WR          <= '0';
99           ACC_RST         <= '0';
100          ACC_WR          <= '0';
101        when cu_load_op1   =>
102          IN_SEL          <= "00";
103          OP_CODE_BUS     <= "00";
104          RAM_ADDR_BUS    <= "00";
105          RAM_WR          <= '1';
106          ACC_RST         <= '0';
107          ACC_WR          <= '1';
108        when cu_load_op2   =>
109          IN_SEL          <= "00";
110          OP_CODE_BUS     <= "00";
111          RAM_ADDR_BUS    <= "01";
112          RAM_WR          <= '1';
113          ACC_RST         <= '0';
114          ACC_WR          <= '1';
115        when cu_run_calc0 =>
116          IN_SEL          <= "01";
117          OP_CODE_BUS     <= "00";
118          RAM_ADDR_BUS    <= "01";
119          RAM_WR          <= '0';
120          ACC_RST         <= '0';
121          ACC_WR          <= '1';
122        when cu_run_calc1 =>
123          IN_SEL          <= "10";
124          OP_CODE_BUS     <= "10";
125          RAM_ADDR_BUS    <= "00";
126          RAM_WR          <= '0';
127          ACC_RST         <= '0';
128          ACC_WR          <= '1';
129        when cu_run_calc2 =>
130          IN_SEL          <= "01";
131          OP_CODE_BUS     <= "01";
132          RAM_ADDR_BUS    <= "00";
133          RAM_WR          <= '0';
134          ACC_RST         <= '0';
135          ACC_WR          <= '1';
136        when cu_run_calc3 =>
137          IN_SEL          <= "11";
138          OP_CODE_BUS     <= "11";
139          RAM_ADDR_BUS    <= "00";
140          RAM_WR          <= '0';
141          ACC_RST         <= '0';
142          ACC_WR          <= '1';
143        when cu_finish   =>
144          IN_SEL          <= "00";
145          OP_CODE_BUS     <= "00";
146          RAM_ADDR_BUS    <= "00";
147          RAM_WR          <= '0';
148          ACC_RST         <= '0';
149          ACC_WR          <= '0';
150        when others    =>
151          IN_SEL          <= "00";
152          OP_CODE_BUS     <= "00";
153          RAM_ADDR_BUS    <= "00";
154          RAM_WR          <= '0';
155          ACC_RST         <= '0';
156          ACC_WR          <= '0';
157     end case;
158    end process;
159
160  end my_CU_arch;
```

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

5) Створив файл RAM.vhd.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  entity my_RAM_intf is
6  port (
7     CLOCK             : in std_logic;
8     RAM_WR            : in std_logic;
9     RAM_ADDR_BUS      : in STD_LOGIC_VECTOR  (1 downto 0);
10    RAM_DATA_IN_BUS   : in STD_LOGIC_VECTOR  (7 downto 0);
11    RAM_DATA_OUT_BUS  : out STD_LOGIC_VECTOR (7 downto 0)
12 );
13 end my_RAM_intf;
14
15 architecture my_RAM_arch of my_RAM_intf is
16
17 type ram_type is array (3 downto 0) of STD_LOGIC_VECTOR (7 downto 0);
18 signal RAM_UNIT : ram_type;
19
20 begin
21
22 RAM : process (CLOCK, RAM_ADDR_BUS, RAM_UNIT)
23    begin
24       if (rising_edge (CLOCK)) then
25          if (RAM_WR = '1') then
26             RAM_UNIT (conv_integer (RAM_ADDR_BUS)) <= RAM_DATA_IN_BUS;
27          end if;
28       end if;
29       RAM_DATA_OUT_BUS <= RAM_UNIT (conv_integer(RAM_ADDR_BUS));
30    end process RAM;
31
32
33 end my_RAM_arch;
34
```

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).

```
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
            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";
```

```
81              when HUNDREDS =>
82                  DIGIT_VAL := HONDREDS_BUS;
83                  CUR_DIGIT := ONES;
84                  COMMONS_CTRL := "100";
85              when others =>
86                  DIGIT_VAL := ONES_BUS;
87                  CUR_DIGIT := ONES;
88                  COMMONS_CTRL := "000";
89            end case;
90
91            case DIGIT_VAL is              --abcdefg
92              when "0000" => DIGIT_CTRL := "1111110";
93              when "0001" => DIGIT_CTRL := "0110000";
94              when "0010" => DIGIT_CTRL := "1101101";
95              when "0011" => DIGIT_CTRL := "1111001";
96              when "0100" => DIGIT_CTRL := "0110011";
97              when "0101" => DIGIT_CTRL := "1011011";
98              when "0110" => DIGIT_CTRL := "1011111";
99              when "0111" => DIGIT_CTRL := "1110000";
100             when "1000" => DIGIT_CTRL := "1111111";
101             when "1001" => DIGIT_CTRL := "1111011";
102             when others => DIGIT_CTRL := "0000000";
103           end case;
104         else
105           DIGIT_VAL := ONES_BUS;
106           CUR_DIGIT := ONES;
107           COMMONS_CTRL := "000";
108         end if;
109
110         COMM_ONES     <= COMMONS_CTRL(0);
111         COMM_DECS     <= COMMONS_CTRL(1);
112         COMM_HUNDREDS <= COMMONS_CTRL(2);
113
114         SEG_A <= DIGIT_CTRL(6);
115         SEG_B <= DIGIT_CTRL(5);
116         SEG_C <= DIGIT_CTRL(4);
117         SEG_D <= DIGIT_CTRL(3);
118         SEG_E <= DIGIT_CTRL(2);
119         SEG_F <= DIGIT_CTRL(1);
120         SEG_G <= DIGIT_CTRL(0);
121         DP   <= '0';
122
123       end if;
124   end process INDICATE;
125
126
127 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.

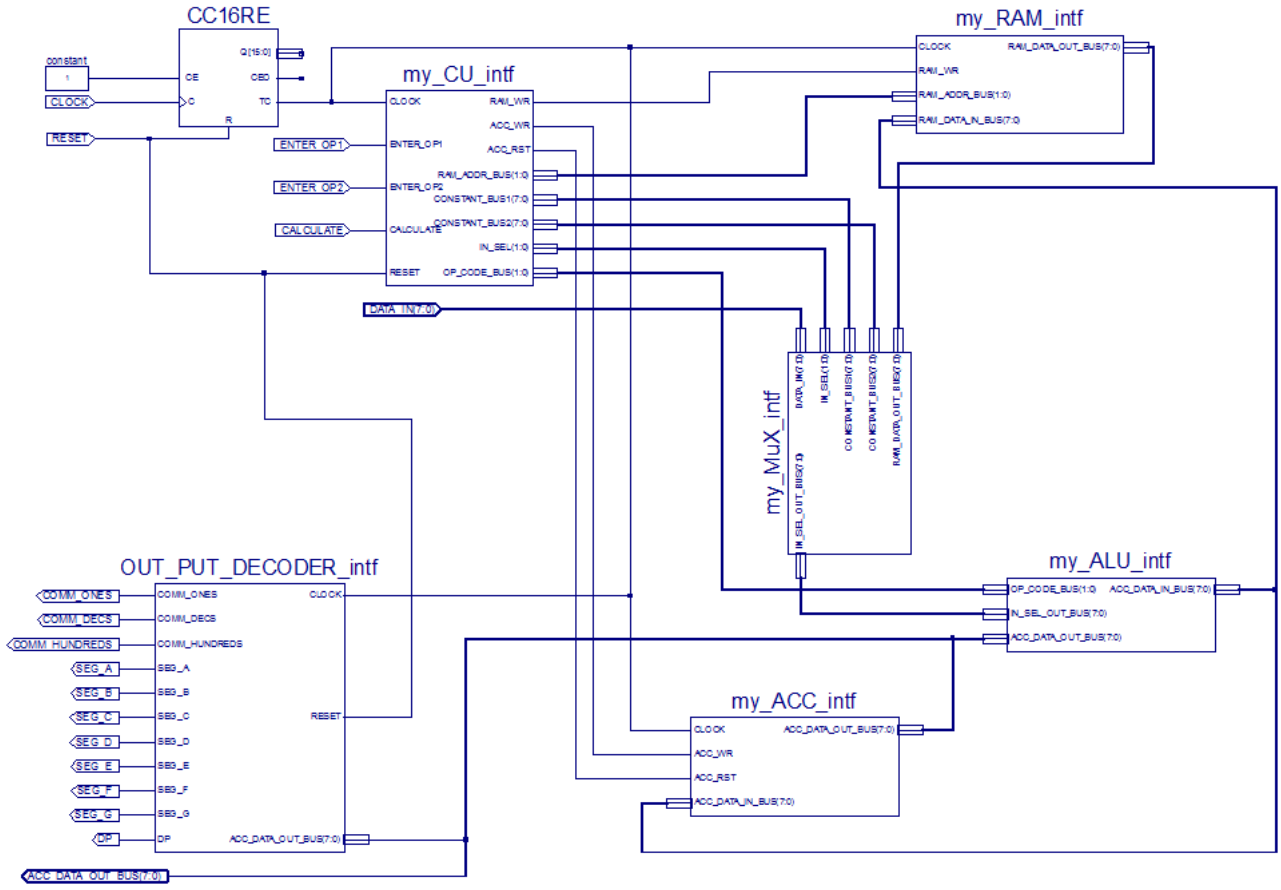


Рис.7. Схема з використанням імплементованих компонентів.

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_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,
      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
   );

```

```
81  CLK_process : process
82    begin
83      CLOCK <= '1';
84      wait for CLK_period/2;
85      CLOCK <= '0';
86      wait for CLK_period/2;
87    end process CLK_process;
88
89    stim_proc: process
90    begin
91    RESET <= '1';
92    ENTER_OP1 <= '0';
93    ENTER_OP2 <= '0';
94     CALCULATE <= '0';
95    DATA_IN   <=(others => '0');
96
97    wait for 2*CLK_period;
98    RESET <='0';
99
100   wait for 4*TC_period;
101   ENTER_OP1 <='1';
102   DATA_IN   <= op1;
103
104   wait for 2*TC_period;
105   ENTER_OP1 <='0';
106
107   wait for 4*TC_period;
108   ENTER_OP2 <='1';
109   DATA_IN   <= op2;
110
111   wait for 2*TC_period;
112   ENTER_OP2 <='0';
113   wait for 4*TC_period;
114
115   CALCULATE <= '1';
116   wait for 8*TC_period;
117    wait;
118   end process stim_proc; --1.835 s
119
120 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;

```

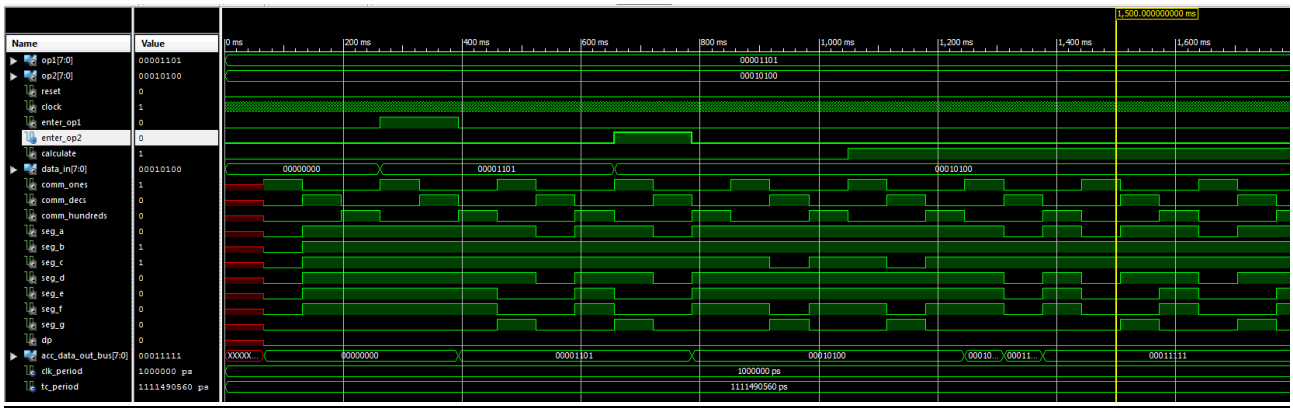
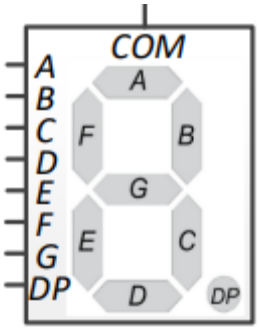


Рис.8. Часова діаграма згідно методичних вказівок.



Перевірка:
((OP2 - 4) + OP1) or 2
OP1 => 0000 1101
OP2 => 0001 0100
((0001 0100 – 0000 0100) + 0000 1101) or 0000 0010
0001 0100 – 0000 0100 = 0001 0000
0001 0000 + 0000 1101 = 0001 1101
0001 1101 or
0000 0010 = 0001 1111

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