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 №19

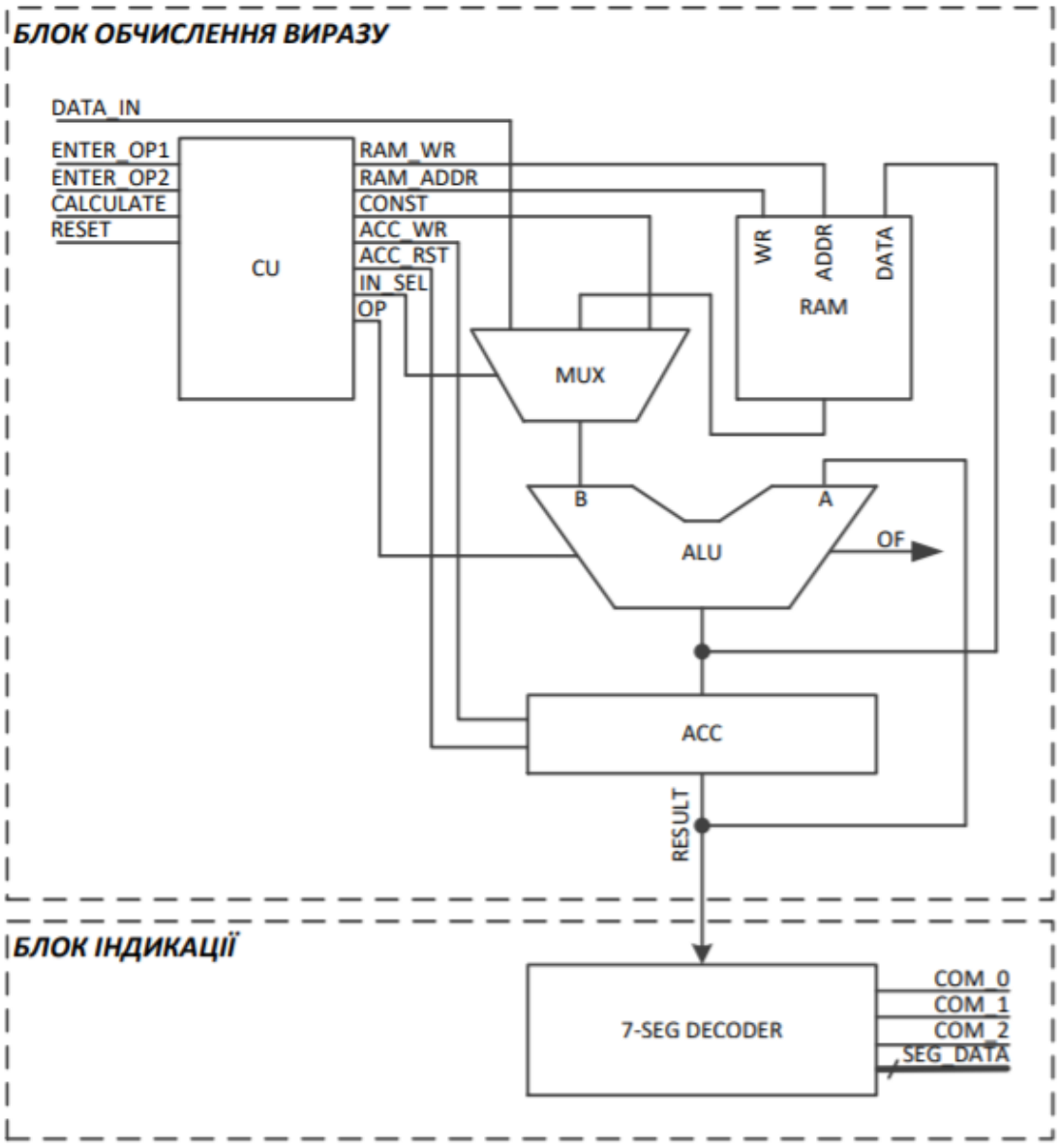
Виконав:
ст.гр. КІ-201
Салагуб А. О.
Прийняв:
Козак Н.Б.

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).



Малюнок 1 - Структурна схема автомата.

Завдання:

ВАРІАНТ	ВИРАЗ
5	((1 << OP1) + OP2) – OP1

Виконання:

1) Створив файл MUX.vhd і реалізував у ньому мультиплексор MUX (рис.1).

```
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_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 my_MuX_intf;
13
14 architecture my_MuX_arch of my_MuX_intf is
15
16 begin
17     INSEL_A_MUX : process(DATA_IN, CONSTANT_BUS, RAM_DATA_OUT_BUS, IN_SEL)
18     begin
19       if(IN_SEL = "00") then
20          IN_SEL_OUT_BUS <= DATA_IN;
21       elsif(IN_SEL = "01") then
22          IN_SEL_OUT_BUS <= RAM_DATA_OUT_BUS;
23       else
24          IN_SEL_OUT_BUS <= CONSTANT_BUS;
25       end if;
26     end process INSEL_A_MUX;
27 end my_MuX_arch;
28
```

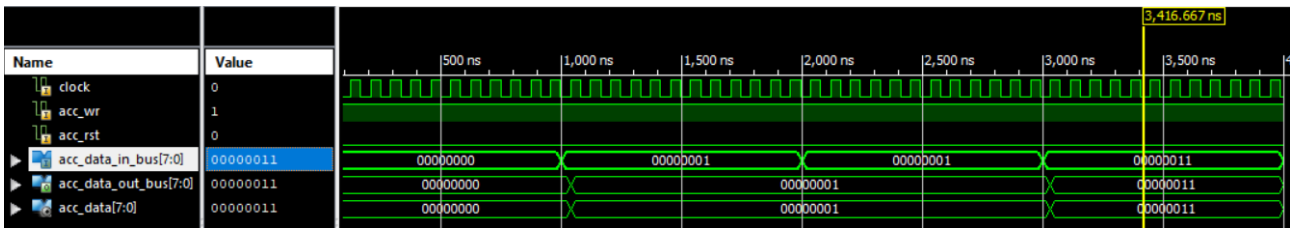
Рис.1. Реалізація мультиплексора у файлі MUX.vhd

Симуляція роботи мультиплексора.

2) Створив файл ACC.vhd і реалізував у ньому регістр ACC (рис.2).

```
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;
34
```

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

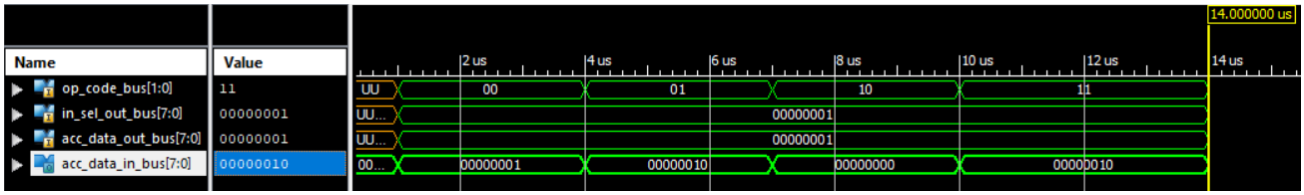


Симуляція роботи регістра.

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"   =>
31               case(B) is
32                  when x"00"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 0);
33                  when x"01"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 1);
34                  when x"02"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 2);
35                  when x"03"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 3);
36                  when x"04"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 4);
37                  when x"05"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 5);
38                  when x"06"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 6);
39                  when x"07"   => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 7);
40                  when others  => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A sll 0);
41               end case;
42          when others => ACC_DATA_IN_BUS <= "00000000";
43       end case;
44
45     end process ALU;
46
47 end my_ALU_arch;
```

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



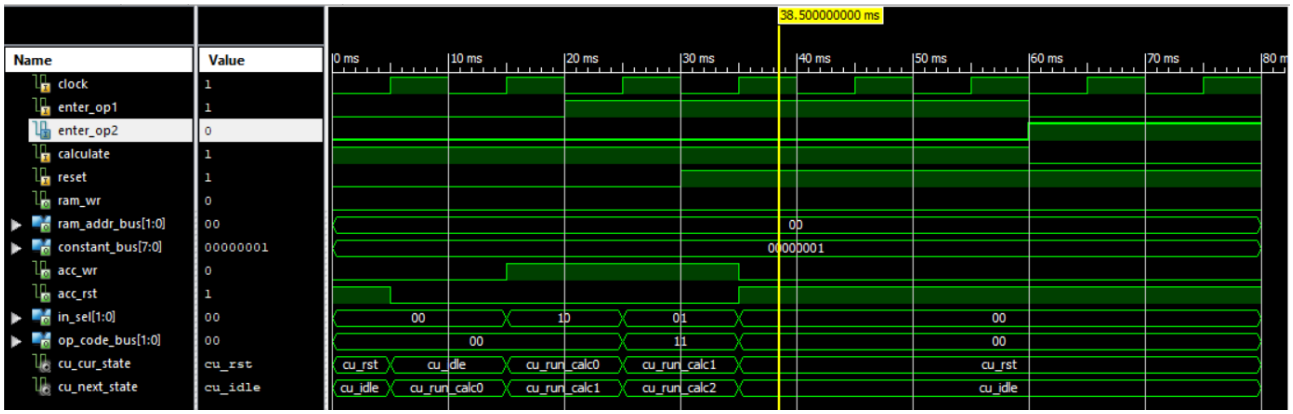
Симуляція роботи АЛП

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

```
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_BUS  : 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_BUS <= "00000001";

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;
   end process;
```

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

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

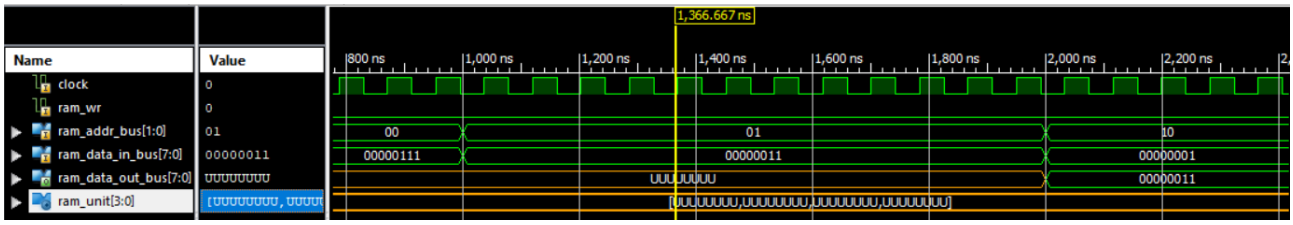


Симуляція роботи керуючого автомата

5) Створив файл RAM.vhd і реалізував у ньому пам’ять пристрою (RAM) (рис.5).

```
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 --when reset will init const
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;
```

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



Симуляція роботи RAM

6) Створив файл OUT_PUT_DECODER.vhd і реалізував в ньому блок індикації (7-SEG DECODER) (рис.6).

```
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 OUT_PUT_DECODER_intf is
7  port(
8  CLOCK            : IN STD_LOGIC;
9  RESET            : IN STD_LOGIC;
10 ACC_DATA_OUT_BUS : IN std_logic_vector(7 downto 0);
11
12 COMM_ONES        : OUT STD_LOGIC;
13 COMM_DECS        : OUT STD_LOGIC;
14 COMM_HUNDREDS    : OUT STD_LOGIC;
15 SEG_A            : OUT STD_LOGIC;
16 SEG_B            : OUT STD_LOGIC;
17 SEG_C            : OUT STD_LOGIC;
18 SEG_D            : OUT STD_LOGIC;
19 SEG_E            : OUT STD_LOGIC;
20 SEG_F            : OUT STD_LOGIC;
21 SEG_G            : OUT STD_LOGIC;
22 DP               : OUT STD_LOGIC
23 );
24 end OUT_PUT_DECODER_intf;
25
26 architecture OUT_PUT_DECODER_arch of OUT_PUT_DECODER_intf is
27 signal ONES_BUS : STD_LOGIC_VECTOR(3 downto 0) := "0000";
28 signal DECS_BUS : STD_LOGIC_VECTOR(3 downto 0) := "0001";
29 signal HONDREDS_BUS : STD_LOGIC_VECTOR(3 downto 0) := "0000";
30
31 begin
32
33    BIN_TO_BCD : process (ACC_DATA_OUT_BUS)
34        variable hex_src : STD_LOGIC_VECTOR(7 downto 0) ;
```

```
35          variable bcd     : STD_LOGIC_VECTOR(11 downto 0) ;
36      begin
37          bcd             := (others => '0') ;
38          hex_src         := ACC_DATA_OUT_BUS;
39
40          for i in hex_src'range loop
41              if bcd(3 downto 0) > "0100" then
42                  bcd(3 downto 0) := bcd(3 downto 0) + "0011" ;
43              end if ;
44              if bcd(7 downto 4) > "0100" then
45                  bcd(7 downto 4) := bcd(7 downto 4) + "0011" ;
46              end if ;
47              if bcd(11 downto 8) > "0100" then
48                  bcd(11 downto 8) := bcd(11 downto 8) + "0011" ;
49              end if ;
50
51              bcd := bcd(10 downto 0) & hex_src(hex_src'left) ; -- shift bcd + 1 new entry
52              hex_src := hex_src(hex_src'left - 1 downto hex_src'right) & '0' ; -- shift src + pad with 0
53          end loop ;
54
55          HONDREDS_BUS       <=  bcd (11 downto 8);
56          DECS_BUS        <=  bcd (7 downto 4);
57          ONES_BUS       <=  bcd (3 downto 0);
58
59      end process BIN_TO_BCD;
60
61      INDICATE : process(CLOCK)
62        type DIGIT_TYPE is (ONES, DECS, HUNDREDS);
63
64        variable CUR_DIGIT     : DIGIT_TYPE := ONES;
65        variable DIGIT_VAL     : STD_LOGIC_VECTOR(3 downto 0) := "0000";
66        variable DIGIT_CTRL    : STD_LOGIC_VECTOR(6 downto 0) := "0000000";
67        variable COMMONS_CTRL  : STD_LOGIC_VECTOR(2 downto 0) := "000";
68
69          begin
70             if (rising_edge(CLOCK)) then
71                if(RESET = '0') then
72                   case CUR_DIGIT is
73                      when ONES =>
74                           DIGIT_VAL := ONES_BUS;
75                           CUR_DIGIT := DECS;
76                           COMMONS_CTRL := "001";
77                      when DECS =>
78                           DIGIT_VAL := DECS_BUS;
79                           CUR_DIGIT := HUNDREDS;
80                           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;
128
129
```

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

7) Згенерував символи для імплементованих компонентів і створив схему у файлі Top_level.sch (рис.7).

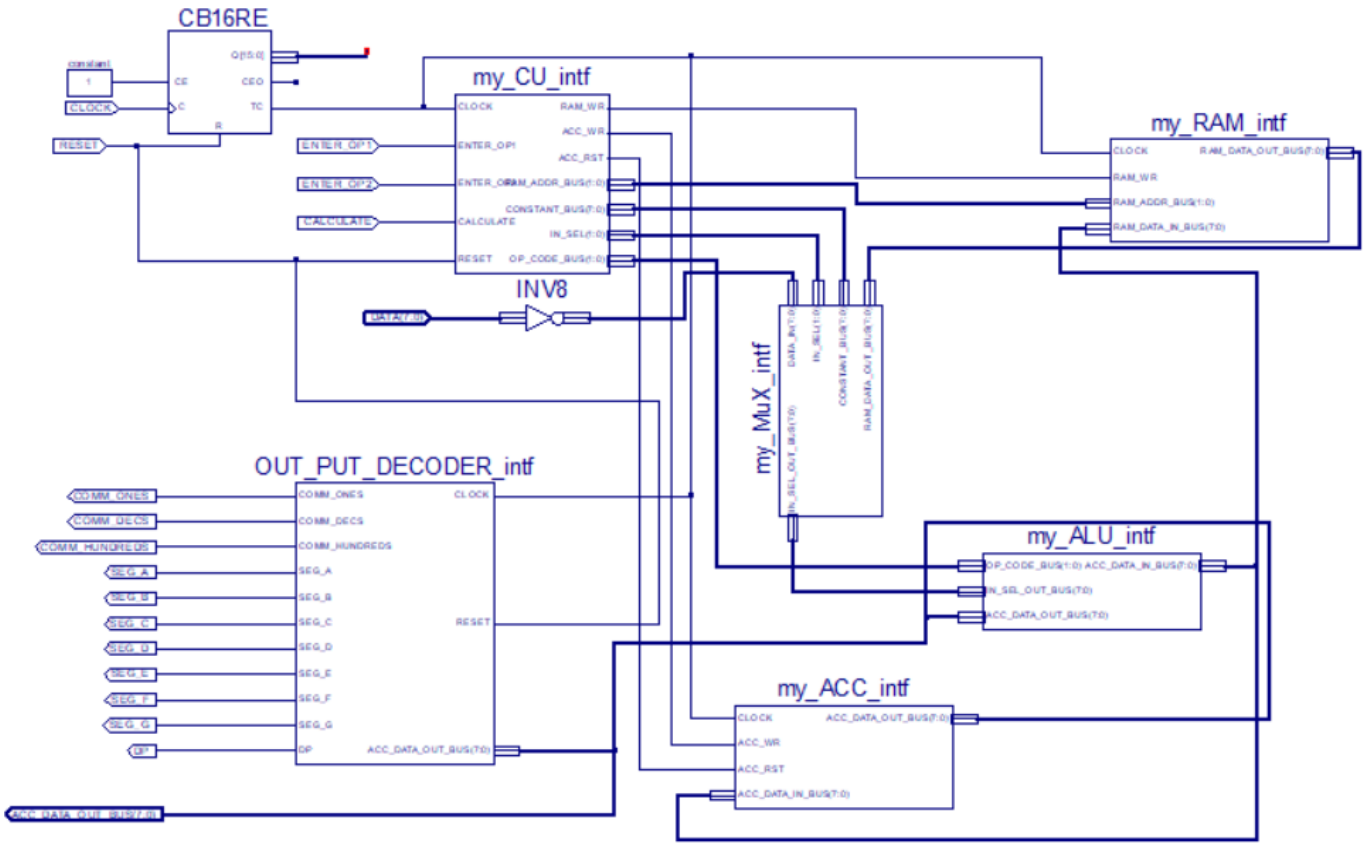


Рис.7. Схема з використанням імплементованих компонентів

8) Test bench TB_TOPLEVEL.vhd.

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

   COMPONENT TOP_LEVEL
   PORT(  RESET    :  IN STD_LOGIC;
          CLOCK    :  IN STD_LOGIC;
          ENTER_OP1   :  IN STD_LOGIC;
          ENTER_OP2   :  IN STD_LOGIC;
          CALCULATE   :  IN STD_LOGIC;
          DATA :  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;
          ACC_DATA_OUT_BUS : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   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;
```

```
35    signal ENTER_OP1  :     STD_LOGIC;
36    signal ENTER_OP2  :     STD_LOGIC;
37    signal CALCULATE  :     STD_LOGIC;
38    signal DATA :     STD_LOGIC_VECTOR (7 DOWNTO 0);
39    signal COMM_ONES  :     STD_LOGIC;
40    signal COMM_DECS  :     STD_LOGIC;
41    signal COMM_HUNDREDS :     STD_LOGIC;
42    signal SEG_A   :     STD_LOGIC;
43    signal SEG_B   :     STD_LOGIC;
44    signal SEG_C   :     STD_LOGIC;
45    signal SEG_D   :     STD_LOGIC;
46    signal SEG_E   :     STD_LOGIC;
47    signal SEG_F   :     STD_LOGIC;
48    signal SEG_G   :     STD_LOGIC;
49    signal DP      :     STD_LOGIC;
50    signal ACC_DATA_OUT_BUS :  STD_LOGIC_VECTOR(7 DOWNTO 0);
51
52
53     constant CLK_period: time := 1 us;
54     constant TC_period: time := 65536 us;
55
56
57
58  BEGIN
59
60     UUT: TOP_LEVEL PORT MAP(
61     RESET => RESET,
62        CLOCK => CLOCK,
63        ENTER_OP1 => ENTER_OP1,
64        ENTER_OP2 => ENTER_OP2,
65        CALCULATE => CALCULATE,
66        DATA => DATA,
67        COMM_ONES => COMM_ONES,
68        COMM_DECS => COMM_DECS,
```

```
69          COMM_HUNDREDS => COMM_HUNDREDS,
70          SEG_A => SEG_A,
71          SEG_B => SEG_B,
72          SEG_C => SEG_C,
73          SEG_D => SEG_D,
74          SEG_E => SEG_E,
75          SEG_F => SEG_F,
76          SEG_G => SEG_G,
77          DP => DP,
78          ACC_DATA_OUT_BUS => ACC_DATA_OUT_BUS
79
80     );
81
82  CLK_process : process
83     begin
84        CLOCK <= '1';
85        wait for CLK_period/2;
86        CLOCK <= '0';
87        wait for CLK_period/2;
88     end process CLK_process;
89
90
91
92     stim_proc: process
93     begin
94     RESET <= '1';
95     ENTER_OP1 <= '0';
96     ENTER_OP2 <= '0';
97     CALCULATE <= '0';
98     DATA   <=(others => '0');
99
100    wait for 2*CLK_period;
101    RESET <='0';
102
```

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

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

    wait for 4*TC_period;
    ENTER_OP2 <='1';
    DATA    <= 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;
```

9) 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(0)"        LOC = P70   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA(1)"        LOC = P69   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA(2)"        LOC = P68   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA(3)"        LOC = P64   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA(4)"        LOC = P63   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "DATA(5)"        LOC = P60   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

```
35      NET "DATA(6)"         LOC = P59    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
36      NET "DATA(7)"         LOC = P58    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
37
38  ##############################################################################################################
39  #                                        Switches
40  ##############################################################################################################
41
42      NET "ENTER_OP1"           LOC = P80    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
43      NET "ENTER_OP2"           LOC = P79    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
44      NET "CALCULATE"            LOC = P78    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
45      NET "RESET"             LOC = P75    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
46
```

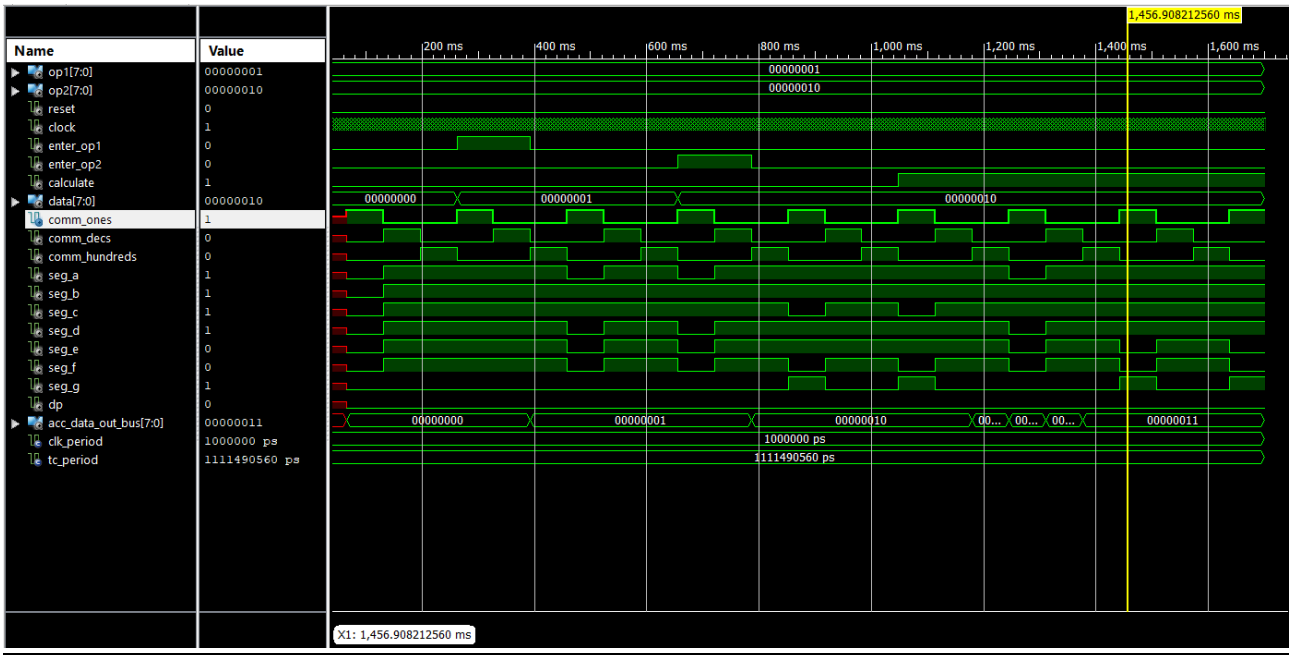
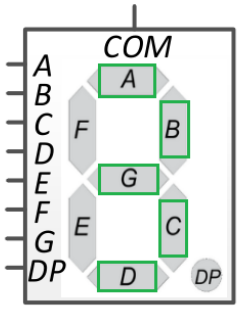


Рис.8. Часова діаграма згідно методичних вказівок

Перевірка:
((1 << OP1) + OP2) – OP1
OP1 => 1
OP2 => 2
((1 << 1) + 1) – 1
1 << 1 = 2
2 + 2 = 4
4 – 1 = 3_{10} = 00000011_2
В результаті на всі сегменти окрім e та f подалася одиниця, тобто число 3



7-сегментний індикатор.

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