Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ

Звіт

до 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 №10

Виконав:
ст. гр. КІ-201
Гришканич А. М.
Прийняв:
Козак Н. Б.

Львів 2024

Мета роботи: На базі стенда реалізувати цифровий автомат для обчислення значення виразів.

Виконання роботи:

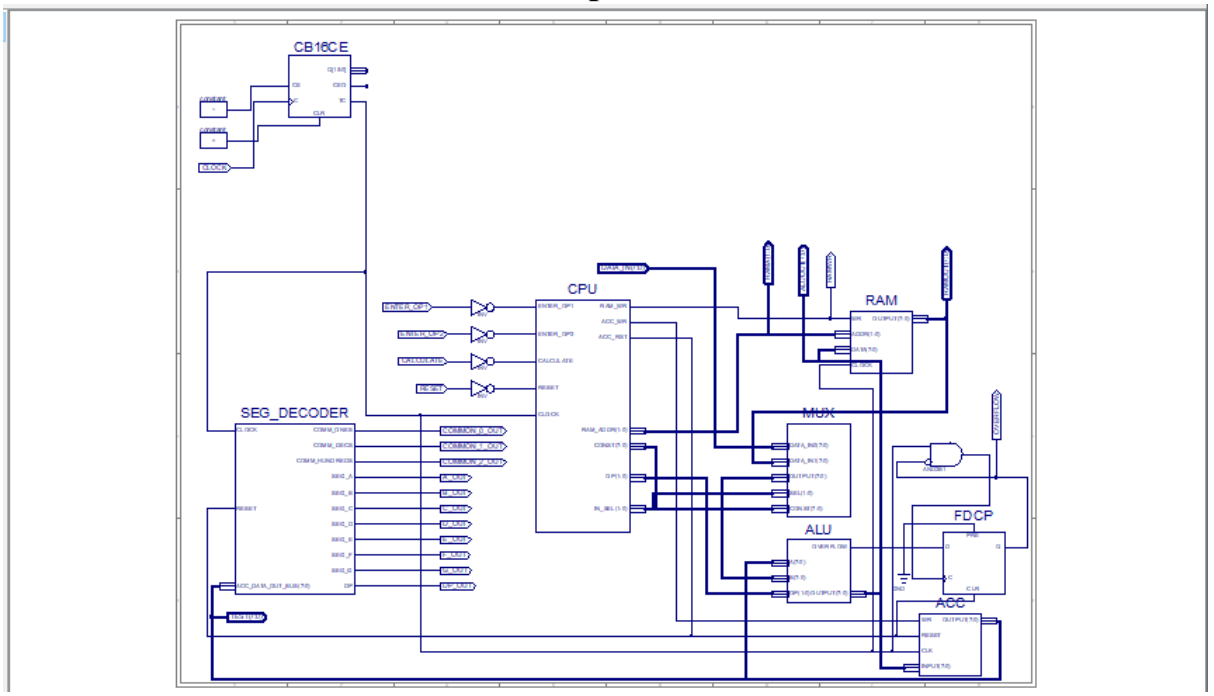

Рис. 1 – Top Level

Файл ACC.vhd

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  -- Uncomment the following library declaration if using
5.  -- arithmetic functions with Signed or Unsigned values
6.  --use IEEE.NUMERIC_STD.ALL;
7.
8.  -- Uncomment the following library declaration if instantiating
9.  -- any Xilinx primitives in this code.
10. --library UNISIM;
11. --use UNISIM.VComponents.all;
12.
13. entity ACC is
14.     Port ( WR    : in  STD_LOGIC;
15.            RESET  : in  STD_LOGIC;
16.            CLK   : in  STD_LOGIC;
17.            INPUT : in  STD_LOGIC_VECTOR (7 downto 0);
18.            OUTPUT : out  STD_LOGIC_VECTOR (7 downto 0));
19. end ACC;
20.
21. architecture ACC_arch of ACC is
22.     signal DATA : STD_LOGIC_VECTOR (7 downto 0);
23. begin
24.     process (CLK)
25.     begin
26.         if rising_edge(CLK) then
27.             if RESET = '1' then
28.                 DATA <= (others => '0');
29.             elsif WR = '1' then
```

```
                DATA <= INPUT;
            end if;
        end if;
    end process;


    OUTPUT <= DATA;

end ACC_arch;

```

Файл ALU.vhd

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ALU is
    Port ( A : in  STD_LOGIC_VECTOR(7 downto 0);
           B : in  STD_LOGIC_VECTOR(7 downto 0);
           OP : in  STD_LOGIC_VECTOR(1 downto 0);
           OUTPUT : out  STD_LOGIC_VECTOR(7 downto 0);
                       OVERFLOW: out STD_LOGIC);
end ALU;


architecture ALU_Behavioral of ALU is
        signal ALUR: STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
        signal Carry: STD_LOGIC := '0';
begin
        process(A, B, OP)
        begin
                case (OP) is
                        when "01" => ALUR <= ("00000000" & A) + ("00000000" & B);
                        when "10" => ALUR <= ("00000000" & A) + ("11111111" & not
    B) + "0000000000000001";
                        when "11" =>
                                case(B) is
                                        when x"00"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 0);
                                        when x"01"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 1);
                                        when x"02"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 2);
                                        when x"03"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 3);
                                        when x"04"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 4);
                                        when x"05"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 5);
                                        when x"06"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 6);
                                        when x"07"   => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 7);
                                        when others  => ALUR <=
    std_logic_vector(unsigned(("00000000" & A)) sll 0);
```

```
43.                             end case;
44.                         when others => ALUR <= ("00000000" & B);
45.                 end case;
46.         end process;
47.         OUTPUT <= ALUR(7 downto 0);
48.         OVERFLOW <= ALUR(8) OR ALUR(9) OR ALUR(10) OR ALUR(11) OR ALUR(12) OR
    ALUR(13) OR ALUR(14) OR ALUR(15);
49. end ALU_Behavioral;
```

Файл CPU.vhd

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.
5.  entity CPU is
6.          port( ENTER_OP1 : IN STD_LOGIC;
7.                          ENTER_OP2 : IN STD_LOGIC;
8.                          CALCULATE : IN STD_LOGIC;
9.                          RESET : IN STD_LOGIC;
10.                         CLOCK : IN STD_LOGIC;
11.                         RAM_WR : OUT STD_LOGIC;
12.                         RAM_ADDR : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
13.                         CONST : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
14.                         ACC_WR : OUT STD_LOGIC;
15.                         ACC_RST : OUT STD_LOGIC;
16.                         IN_SEL : OUT STD_LOGIC_VECTOR(1 downto 0);
17.                         OP : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
18. end CPU;
19.
20. architecture CPU_arch of CPU is
21.
22. type   STATE_TYPE is (RST, IDLE, LOAD_OP1, LOAD_OP2, RUN_CALC0, RUN_CALC1,
    RUN_CALC2, RUN_CALC3, RUN_CALC4, FINISH);
23. signal CUR_STATE  : STATE_TYPE;
24. signal NEXT_STATE : STATE_TYPE;
25.
26. begin
27.         SYNC_PROC: process (CLOCK)
28.    begin
29.       if (rising_edge(CLOCK)) then
30.          if (RESET = '1') then
31.             CUR_STATE <= RST;
32.          else
33.             CUR_STATE <= NEXT_STATE;
34.          end if;
35.       end if;
36.    end process;
37.
38.
39.         NEXT_STATE_DECODE: process (CLOCK, ENTER_OP1, ENTER_OP2, CALCULATE)
40.    begin
41.       NEXT_STATE <= CUR_STATE;
42.
43.                 case(CUR_STATE) is
44.                         when RST =>
45.                                 NEXT_STATE <= IDLE;
46.                         when IDLE             =>
47.                                 if (ENTER_OP1 = '1') then
48.                                         NEXT_STATE <= LOAD_OP1;
49.                                 elsif (ENTER_OP2 = '1') then
50.                                         NEXT_STATE <= LOAD_OP2;
```

```
51.                             elsif (CALCULATE = '1') then
52.                                     NEXT_STATE <= RUN_CALC0;
53.                             else
54.                                     NEXT_STATE <= IDLE;
55.                             end if;
56.                     when LOAD_OP1  =>
57.                             NEXT_STATE <= IDLE;
58.                     when LOAD_OP2  =>
59.                             NEXT_STATE <= IDLE;
60.                     when RUN_CALC0 =>
61.                             NEXT_STATE <= RUN_CALC1;
62.                     when RUN_CALC1 =>
63.                             NEXT_STATE <= RUN_CALC2;
64.                     when RUN_CALC2 =>
65.                             NEXT_STATE <= RUN_CALC3;
66.                     when RUN_CALC3 =>
67.                             NEXT_STATE <= RUN_CALC4;
68.                     when RUN_CALC4 =>
69.                             NEXT_STATE <= FINISH;
70.                     when FINISH    =>
71.                             NEXT_STATE <= FINISH;
72.                     when others                     =>
73.                             NEXT_STATE <= IDLE;
74.             end case;
75.     end process;
76.
77.         OUTPUT_DECODE: process (CUR_STATE)
78.         begin
79.                 case (CUR_STATE) is
80.                         when RST =>
81.                                 RAM_WR <= '0';
82.                                 RAM_ADDR <= "00";
83.                                 CONST <= "00000000";
84.                                 ACC_WR <= '0';
85.                                 ACC_RST <= '1';
86.                                 IN_SEL <= "00";
87.                                 OP <= "00";
88.                         when LOAD_OP1 =>
89.                                 RAM_WR <= '1';
90.                                 RAM_ADDR <= "00";
91.                                 CONST <= "00000000";
92.                                 ACC_WR <= '0';
93.                                 ACC_RST <= '1';
94.                                 IN_SEL <= "00";
95.                                 OP <= "00";
96.                         when LOAD_OP2 =>
97.                                 RAM_WR <= '1';
98.                                 RAM_ADDR <= "01";
99.                                 CONST <= "00000000";
100.                                ACC_WR <= '0';
101.                                ACC_RST <= '1';
102.                                IN_SEL <= "00";
103.                                OP <= "00";
104.                        when RUN_CALC0 =>
105.                                RAM_WR <= '0';
106.                                RAM_ADDR <= "01";
107.                                CONST <= "00000000";
108.                                ACC_WR <= '1';
109.                                ACC_RST <= '0';
110.                                IN_SEL <= "01";
111.                                OP <= "00";
112.                        when RUN_CALC1 =>
113.                                RAM_WR <= '0';
114.                                RAM_ADDR <= "00";
115.                                CONST <= "00000000";
116.                                ACC_WR <= '1';
```

```
                                ACC_RST <= '0';
                                IN_SEL <= "01";
                                OP <= "10";
                        when RUN_CALC2 =>
                                RAM_WR <= '0';
                                RAM_ADDR <= "00";
                                CONST <= "00000001";
                                ACC_WR <= '1';
                                ACC_RST <= '0';
                                IN_SEL <= "10";
                                OP <= "11";
                        when RUN_CALC3 =>
                                RAM_WR <= '0';
                                RAM_ADDR <= "00";
                                CONST <= "00000000";
                                ACC_WR <= '1';
                                ACC_RST <= '0';
                                IN_SEL <= "01";
                                OP <= "01";
                        when RUN_CALC4 =>
                                RAM_WR <= '0';
                                RAM_ADDR <= "00";
                                CONST <= "00001010";
                                ACC_WR <= '1';
                                ACC_RST <= '0';
                                IN_SEL <= "10";
                                OP <= "01";
                        when IDLE =>
                                RAM_WR <= '0';
                                RAM_ADDR <= "00";
                                CONST <= "00000000";
                                ACC_WR <= '0';
                                ACC_RST <= '0';
                                IN_SEL <= "00";
                                OP <= "00";
                        when others =>
                                RAM_WR <= '0';
                                RAM_ADDR <= "00";
                                CONST <= "00000000";
                                ACC_WR <= '0';
                                ACC_RST <= '0';
                                IN_SEL <= "00";
                                OP <= "00";
                end case;
        end process;
    end CPU_arch;
```

Файл MUX.vhd

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity MUX is
5.          PORT(
6.                  SEL: in STD_LOGIC_VECTOR(1 downto 0);
7.                  CONST: in STD_LOGIC_VECTOR(7 downto 0);
8.                  --CONST1: in STD_LOGIC_VECTOR()
9.                  DATA_IN0: in STD_LOGIC_VECTOR(7 downto 0);
10.                 DATA_IN1: in STD_LOGIC_VECTOR(7 downto 0);
11.                 OUTPUT: out STD_LOGIC_VECTOR(7 downto 0)
12.         );
13. end MUX;
14.
15. architecture Behavioral of MUX is
16. begin
```

```
17.         process (SEL, DATA_IN0, DATA_IN1, CONST)
18.         begin
19.                 if (SEL = "00") then
20.                         OUTPUT <= DATA_IN0;
21.                 elsif (SEL = "01") then
22.                         OUTPUT <= DATA_IN1;
23.                 else
24.                         OUTPUT <= CONST;
25.                 end if;
26.         end process;
27. end Behavioral;
```

Файл RAM.vhd

```
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.
7.  entity RAM is
8.          port(
9.                          WR : IN STD_LOGIC;
10.                         ADDR : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
11.                         DATA : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
12.                         CLOCK: IN STD_LOGIC;
13.                         OUTPUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
14.                         );
15. end RAM;
16.
17. architecture RAM_arch of RAM is
18.         type ram_type is array (3 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
19.         signal UNIT : ram_type;
20.
21. begin
22.         process(ADDR, CLOCK, UNIT)
23.          begin
24.                 if(rising_edge(CLOCK)) then
25.                         if (WR = '1') then
26.                                 UNIT(conv_integer(ADDR)) <= DATA;
27.                         end if;
28.                 end if;
29.                 OUTPUT <= UNIT(conv_integer(ADDR));
30.         end process;
31. end RAM_arch;
```

Файл SEG_DECODER.vhd

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  use IEEE.STD_LOGIC_ARITH.ALL;
4.  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5.
6.
7.  entity SEG_DECODER is
8.          port( CLOCK : IN STD_LOGIC;
9.                          RESET : IN STD_LOGIC;
10.                         ACC_DATA_OUT_BUS : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
11.                         COMM_ONES                       : OUT STD_LOGIC;
```

```
12.                       COMM_DECS             : OUT STD_LOGIC;
13.                       COMM_HUNDREDS   : OUT STD_LOGIC;
14.                       SEG_A    : OUT STD_LOGIC;
15.                       SEG_B    : OUT STD_LOGIC;
16.                       SEG_C    : OUT STD_LOGIC;
17.                       SEG_D    : OUT STD_LOGIC;
18.                       SEG_E    : OUT STD_LOGIC;
19.                       SEG_F    : OUT STD_LOGIC;
20.                       SEG_G    : OUT STD_LOGIC;
21.                       DP              : OUT STD_LOGIC);
22. end SEG_DECODER;
23.
24. architecture Behavioral of SEG_DECODER is
25.
26.        signal ONES_BUS : STD_LOGIC_VECTOR(3 downto 0) := "0000";
27.        signal DECS_BUS : STD_LOGIC_VECTOR(3 downto 0) := "0001";
28.        signal HONDREDS_BUS : STD_LOGIC_VECTOR(3 downto 0) := "0000";
29.
30. begin
31.        BIN_TO_BCD : process (ACC_DATA_OUT_BUS)
32.         variable hex_src : STD_LOGIC_VECTOR(7 downto 0) ;
33.         variable bcd     : STD_LOGIC_VECTOR(11 downto 0) ;
34.     begin
35.         bcd             := (others => '0') ;
36.         hex_src         := ACC_DATA_OUT_BUS;
37.
38.         for i in hex_src'range loop
39.             if bcd(3 downto 0) > "0100" then
40.                 bcd(3 downto 0) := bcd(3 downto 0) + "0011" ;
41.             end if ;
42.             if bcd(7 downto 4) > "0100" then
43.                 bcd(7 downto 4) := bcd(7 downto 4) + "0011" ;
44.             end if ;
45.             if bcd(11 downto 8) > "0100" then
46.                 bcd(11 downto 8) := bcd(11 downto 8) + "0011" ;
47.             end if ;
48.
49.             bcd := bcd(10 downto 0) & hex_src(hex_src'left) ; -- shift bcd + 1
    new entry
50.             hex_src := hex_src(hex_src'left - 1 downto hex_src'right) & '0' ; --
    shift src + pad with 0
51.         end loop ;
52.
53.         HONDREDS_BUS       <=  bcd (11 downto 8);
54.         DECS_BUS        <=  bcd (7 downto 4);
55.         ONES_BUS       <=  bcd (3 downto 0);
56.
57.     end process BIN_TO_BCD;
58.
59.         INDICATE : process(CLOCK)
60.               type DIGIT_TYPE is (ONES, DECS, HUNDREDS);
61.
62.               variable CUR_DIGIT     : DIGIT_TYPE := ONES;
63.               variable DIGIT_VAL      : STD_LOGIC_VECTOR(3 downto 0) := "0000";
64.               variable DIGIT_CTRL     : STD_LOGIC_VECTOR(6 downto 0) :=
    "0000000";
65.               variable COMMONS_CTRL  : STD_LOGIC_VECTOR(2 downto 0) := "000";
66.
67.               begin
68.                      if (rising_edge(CLOCK)) then
69.                             if(RESET = '0') then
70.                                    case CUR_DIGIT is
71.                                           when ONES =>
72.                                                  DIGIT_VAL := ONES_BUS;
73.                                                  CUR_DIGIT := DECS;
74.                                                  COMMONS_CTRL := "001";
```

```
75.                                             when DECS =>
76.                                                     DIGIT_VAL := DECS_BUS;
77.                                                     CUR_DIGIT := HUNDREDS;
78.                                                     COMMONS_CTRL := "010";
79.                                             when HUNDREDS =>
80.                                                     DIGIT_VAL := HONDREDS_BUS;
81.                                                     CUR_DIGIT := ONES;
82.                                                     COMMONS_CTRL := "100";
83.                                             when others =>
84.                                                     DIGIT_VAL := ONES_BUS;
85.                                                     CUR_DIGIT := ONES;
86.                                                     COMMONS_CTRL := "000";
87.                                     end case;
88.
89.                                     case DIGIT_VAL is               --abcdefg
90.                                             when "0000" => DIGIT_CTRL :=
    "1111110";
91.                                             when "0001" => DIGIT_CTRL :=
    "0110000";
92.                                             when "0010" => DIGIT_CTRL :=
    "1101101";
93.                                             when "0011" => DIGIT_CTRL :=
    "1111001";
94.                                             when "0100" => DIGIT_CTRL :=
    "0110011";
95.                                             when "0101" => DIGIT_CTRL :=
    "1011011";
96.                                             when "0110" => DIGIT_CTRL :=
    "1011111";
97.                                             when "0111" => DIGIT_CTRL :=
    "1110000";
98.                                             when "1000" => DIGIT_CTRL :=
    "1111111";
99.                                             when "1001" => DIGIT_CTRL :=
    "1111011";
100.                                            when others => DIGIT_CTRL :=
    "0000000";
101.                                    end case;
102.                            else
103.                                    DIGIT_VAL := ONES_BUS;
104.                                    CUR_DIGIT := ONES;
105.                                    COMMONS_CTRL := "000";
106.                            end if;
107.
108.                            COMM_ONES        <= not COMMONS_CTRL(0);
109.                            COMM_DECS        <= not COMMONS_CTRL(1);
110.                            COMM_HUNDREDS <= not COMMONS_CTRL(2);
111.
112.                            SEG_A <= not DIGIT_CTRL(6);
113.                            SEG_B <= not DIGIT_CTRL(5);
114.                            SEG_C <= not DIGIT_CTRL(4);
115.                            SEG_D <= not DIGIT_CTRL(3);
116.                            SEG_E <= not DIGIT_CTRL(2);
117.                            SEG_F <= not DIGIT_CTRL(1);
118.                            SEG_G <= not DIGIT_CTRL(0);
119.                            DP     <= '1';
120.
121.                    end if;
122.       end process INDICATE;
123.
124.   end Behavioral;
```

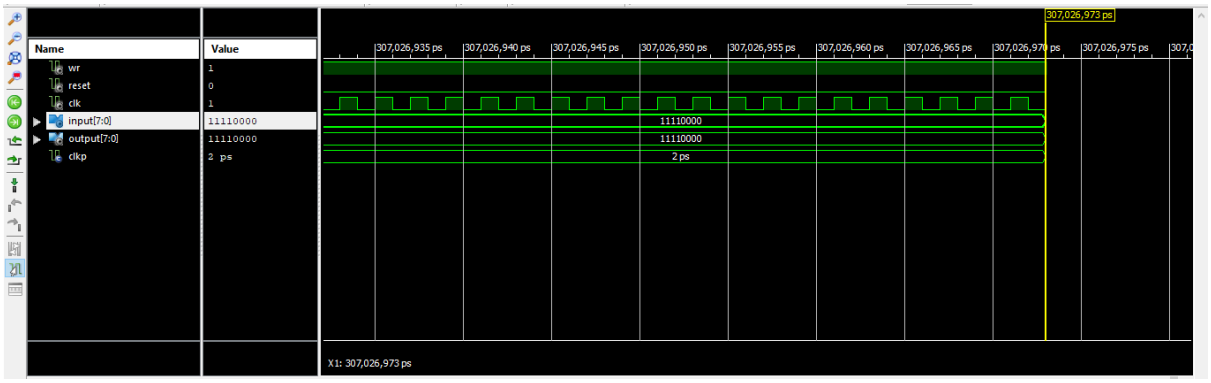

Рис. 2 – Часова діаграма ACC

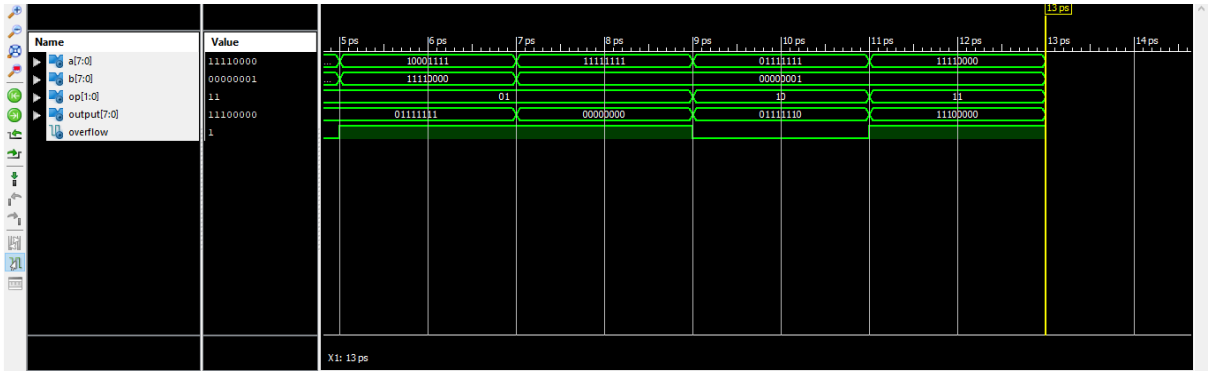

Рис. 3 – Часова діаграма ALU

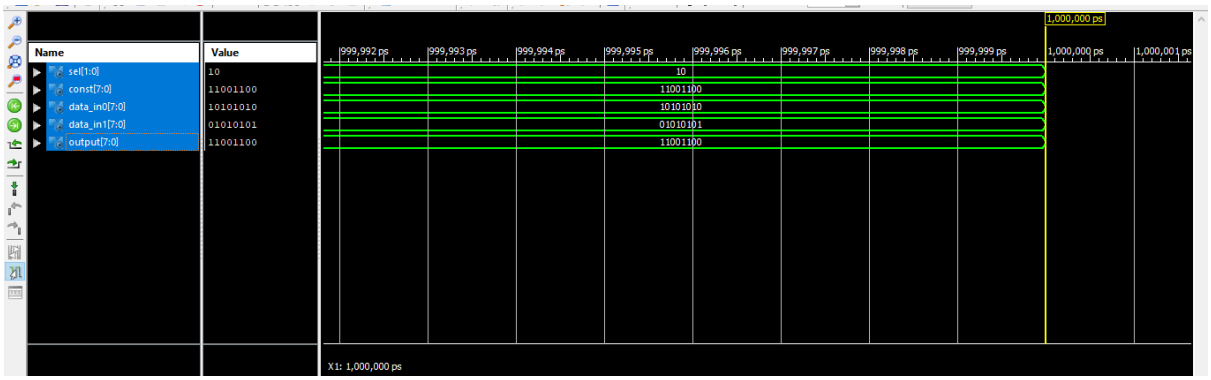

Рис. 4 – Часова діаграма MUX

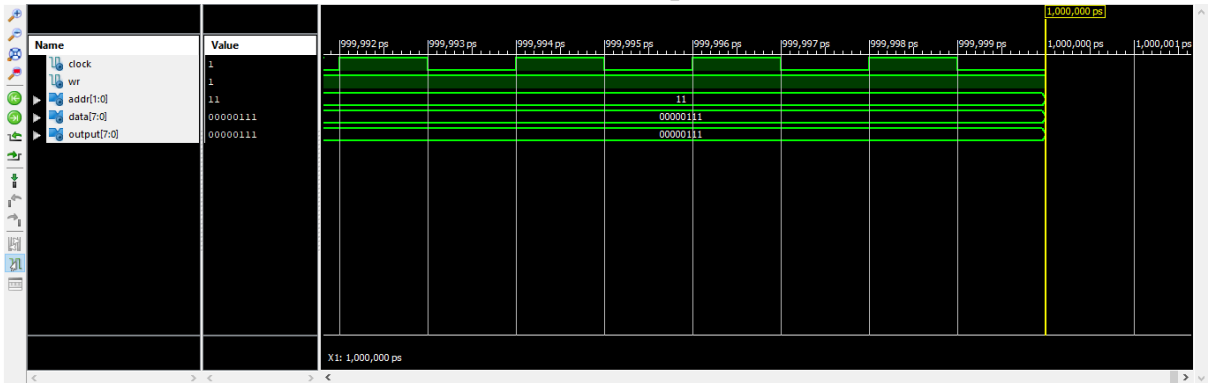

Рис. 5 – Часова діаграма RAM

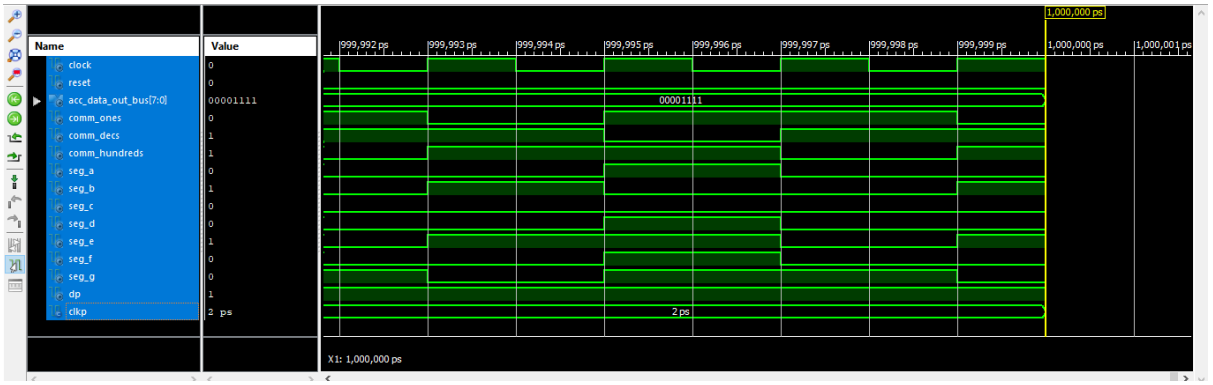

Рис 6. – Часова діграма SEG_DECODER

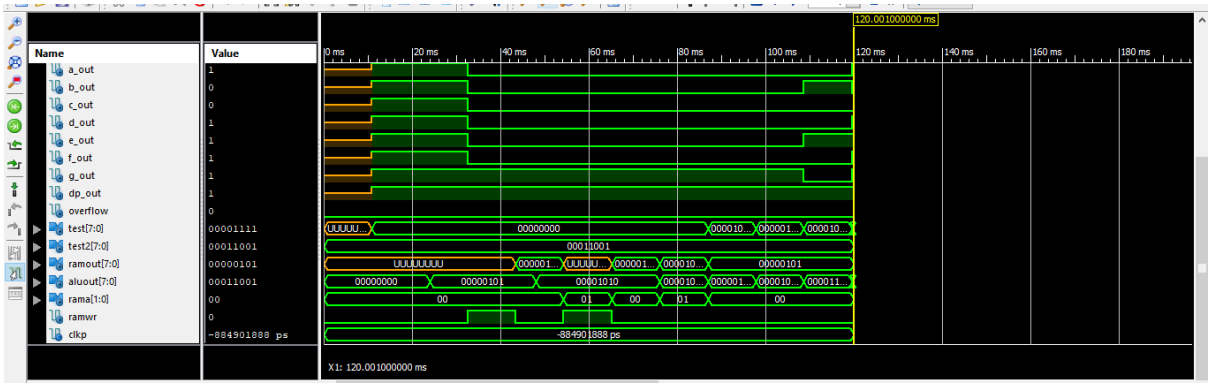

Рис 7. – Часова діграма TopLevel

Файл TopLevelTest.vhd

```
1.  LIBRARY ieee;
2.  USE ieee.std_logic_1164.ALL;
3.  USE ieee.numeric_std.ALL;
4.  LIBRARY UNISIM;
5.  USE UNISIM.Vcomponents.ALL;
6.  ENTITY TopLevel_TopLevel_sch_tb IS
7.  END TopLevel_TopLevel_sch_tb;
8.  ARCHITECTURE behavioral OF TopLevel_TopLevel_sch_tb IS
9.
10.    COMPONENT TopLevel
11.    PORT( CLOCK :       IN      STD_LOGIC;
12.          RESET         :       IN      STD_LOGIC;
13.          ENTER_OP1     :       IN      STD_LOGIC;
14.          ENTER_OP2     :       IN      STD_LOGIC;
15.          CALCULATE     :       IN      STD_LOGIC;
16.          DATA_IN       :       IN      STD_LOGIC_VECTOR (7 DOWNTO 0);
17.          COMMON_0_OUT          :       OUT     STD_LOGIC;
18.          COMMON_1_OUT          :       OUT     STD_LOGIC;
19.          COMMON_2_OUT          :       OUT     STD_LOGIC;
20.                         TEST: OUT STD_LOGIC_VECTOR(7 downto 0);
21.          A_OUT         :       OUT     STD_LOGIC;
22.          B_OUT         :       OUT     STD_LOGIC;
23.          C_OUT         :       OUT     STD_LOGIC;
24.          D_OUT         :       OUT     STD_LOGIC;
25.          E_OUT         :       OUT     STD_LOGIC;
26.          F_OUT         :       OUT     STD_LOGIC;
27.          G_OUT         :       OUT     STD_LOGIC;
28.          DP_OUT        :       OUT     STD_LOGIC;
```

```
                        RAMOUT: OUT STD_LOGIC_VECTOR(7 downto 0);
                        ALUOUT: OUT STD_LOGIC_VECTOR(7 downto 0);
                        RAMA: OUT STD_LOGIC_VECTOR(1 downto 0);
                        RAMWR: OUT STD_LOGIC;
          OVERFLOW    :       OUT     STD_LOGIC);
    END COMPONENT;

    SIGNAL CLOCK       :       STD_LOGIC := '0';
    SIGNAL RESET       :       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 COMMON_0_OUT        :       STD_LOGIC;
    SIGNAL COMMON_1_OUT        :       STD_LOGIC;
    SIGNAL COMMON_2_OUT        :       STD_LOGIC;
    SIGNAL A_OUT       :       STD_LOGIC;
    SIGNAL B_OUT       :       STD_LOGIC;
    SIGNAL C_OUT       :       STD_LOGIC;
    SIGNAL D_OUT       :       STD_LOGIC;
    SIGNAL E_OUT       :       STD_LOGIC;
    SIGNAL F_OUT       :       STD_LOGIC;
    SIGNAL G_OUT       :       STD_LOGIC;
    SIGNAL DP_OUT      :       STD_LOGIC;
    SIGNAL OVERFLOW    :       STD_LOGIC;
        SIGNAL TEST: STD_LOGIC_VECTOR(7 downto 0);
        SIGNAL TEST2: STD_LOGIC_VECTOR(7 downto 0);
        signal RAMOUT: STD_LOGIC_VECTOR(7 downto 0);
        signal ALUOUT: STD_LOGIC_VECTOR(7 downto 0);
        signal RAMA: STD_LOGIC_VECTOR(1 downto 0);
        signal RAMWR: STD_LOGIC;

--      constant CLOCK_period : time := 166ns;
        constant CLKP: time := 12ms;--24ms;

BEGIN

    UUT: TopLevel PORT MAP(
                CLOCK => CLOCK,
                RESET => RESET,
                ENTER_OP1 => ENTER_OP1,
                ENTER_OP2 => ENTER_OP2,
                CALCULATE => CALCULATE,
                DATA_IN => DATA_IN,
                COMMON_0_OUT => COMMON_0_OUT,
                COMMON_1_OUT => COMMON_1_OUT,
                COMMON_2_OUT => COMMON_2_OUT,
                A_OUT => A_OUT,
                B_OUT => B_OUT,
                C_OUT => C_OUT,
                D_OUT => D_OUT,
                E_OUT => E_OUT,
                F_OUT => F_OUT,
                G_OUT => G_OUT,
                DP_OUT => DP_OUT,
                OVERFLOW => OVERFLOW,
                TEST => TEST,
                RAMOUT => RAMOUT,
                ALUOUT => ALUOUT,
                RAMA => RAMA,
                RAMWR => RAMWR
    );

        CLOCK_process: process
    begin
                CLOCK <= '0';
```

```
95.               wait for 83ns;
96.               CLOCK <= '1';
97.               wait for 83ns;
98.   end process;
99.
100.    -- *** Test Bench - User Defined Section ***
101.       tb : PROCESS
102.       BEGIN
103.               lp1: for i in 2 to 2 loop
104.                       lp2: for j in 4 to 4 loop
105.                               TEST2 <= std_logic_vector((to_signed(j - i, 8) sll
    1) + i + 10);
106.                               ENTER_OP1 <= '1';
107.                               ENTER_OP2 <= '1';
108.                               CALCULATE <= '1';
109.                               DATA_IN <= (others => '0');
110.                               RESET <= '0';
111.                               wait for CLKP;
112.                               RESET <= '1';
113.                               wait for CLKP;
114.                               DATA_IN <= std_logic_vector(to_unsigned(i, 8)); -- A
115.                               ENTER_OP1 <= '0';
116.                               wait for CLKP;
117.                               ENTER_OP1 <= '1';
118.                               wait for CLKP;
119.                               DATA_IN <= std_logic_vector(to_unsigned(j, 8)); -- B
120.                               ENTER_OP2 <= '0';
121.                               wait for CLKP;
122.                               ENTER_OP2 <= '1';
123.                               wait for CLKP;
124.                               CALCULATE <= '0'; -- START CALCULATION
125.                               wait for CLKP* 7;
126.                               assert TEST = TEST2 severity FAILURE;
127.                               wait for CLKP;
128.                       end loop;
129.               end loop;
130.
131.          WAIT; -- will wait forever
132.       END PROCESS;
133.    -- *** End Test Bench - User Defined Section ***
134.
135.    END;
```

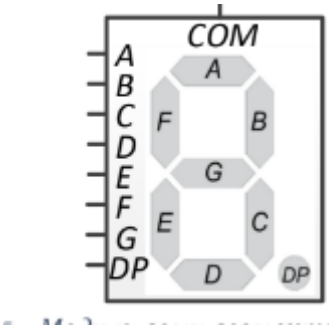

Рис.8 – 7-сегментний індикатор

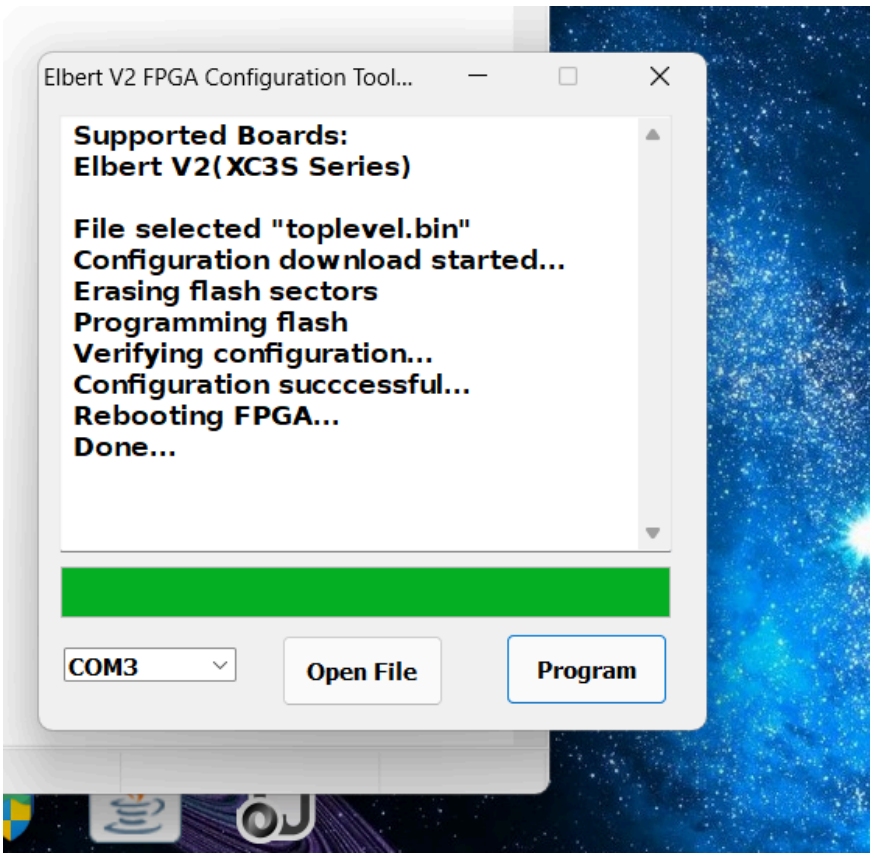

Рис.9 – Успішна прошивка

Висновок: Виконуючи дану лабораторну роботу я навчився реалізовувати цифровий автомат для обчислення значення виразів використовуючи засоби VHDL.