

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 №3

Виконав:
ст. гр. КІ-201
Бовтач П.В

Прийняв:
ст. викладач
каф. ЕОМ
Козак Н. Б.

Львів 2024

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

Виконання роботи:

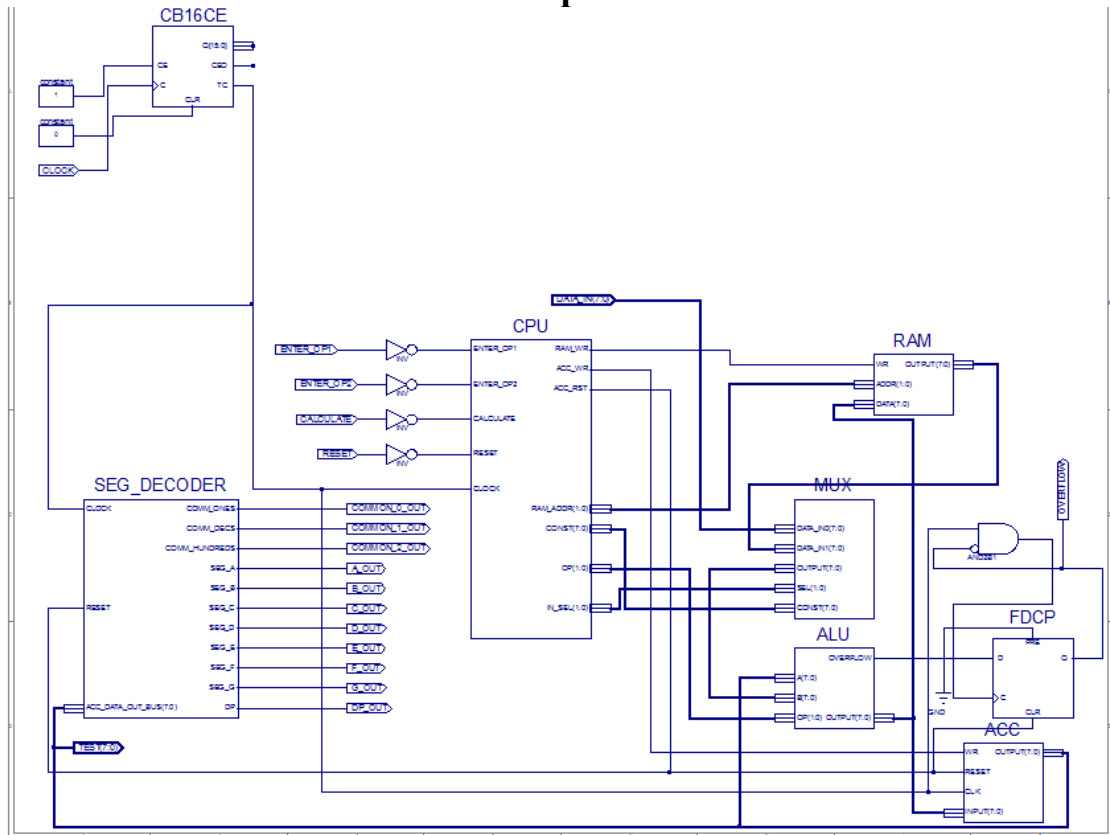


Рис. 1 – Top Level

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.  ENTITY ACCTest IS
9.  END ACCTest;
10.
11. ARCHITECTURE behavior OF ACCTest IS
12.
13. -- Component Declaration for the Unit Under Test (UUT)
14.
15. COMPONENT ACC
16. PORT(
17.   WR : IN std_logic;
18.   RESET : IN std_logic;
19.   CLK : IN std_logic;
20.   INPUT : IN std_logic_vector(7 downto 0);
21.   OUTPUT : OUT std_logic_vector(7 downto 0)
22. );
23. END COMPONENT;
24.

```

```

25.
26. --Inputs
27. signal WR : std_logic := '0';
28. signal RESET : std_logic := '0';
29. signal CLK : std_logic := '0';
30. signal INPUT : std_logic_vector(7 downto 0) := (others => '0');
31.
32. --Outputs
33. signal OUTPUT : std_logic_vector(7 downto 0);
34.
35. -- Clock period definitions
36. constant CLKP: time := 2 ps;
37.
38. BEGIN
39.
40. -- Instantiate the Unit Under Test (UUT)
41. uut: ACC PORT MAP (
42. WR => WR,
43. RESET => RESET,
44. CLK => CLK,
45. INPUT => INPUT,
46. OUTPUT => OUTPUT
47. );
48.
49. -- Clock process definitions
50. CLK_process :process
51. begin
52. CLK <= '0';
53. wait for CLKP/2;
54. CLK <= '1';
55. wait for CLKP/2;
56. end process;
57.
58.
59. -- Stimulus process
60. stim_proc: process
61. begin
62. RESET <= '1';
63. wait for 4 * CLKP;
64. RESET <= '0';
65. WR <= '0';
66. INPUT <= "00001111";
67. wait for CLKP;
68. assert OUTPUT = "00000000" severity failure;
69. WR <= '1';
70. wait for CLKP;
71. assert OUTPUT = "00001111" severity failure;
72. INPUT <= "11110000";
73. wait for CLKP;
74. assert OUTPUT = "11110000" severity failure;
75.
76. wait;
77. end process;
78.
79. END;

```

Файл ALU.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. use IEEE.STD_LOGIC_UNSIGNED.ALL;
8.
9. -- Uncomment the following library declaration if instantiating
10. -- any Xilinx primitives in this code.
11. --library UNISIM;
12. --use UNISIM.VComponents.all;
13.
14. entity ALU is
15. Port ( A : in STD_LOGIC_VECTOR(7 downto 0);
16. B : in STD_LOGIC_VECTOR(7 downto 0);
17. OP : in STD_LOGIC_VECTOR(1 downto 0);
18. OUTPUT : out STD_LOGIC_VECTOR(7 downto 0);
19. OVERFLOW: out STD_LOGIC);
20. end ALU;
21.
22.
23. architecture ALU_Behavioral of ALU is
24. signal ALUR: STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
25. signal Carry: STD_LOGIC := '0';
26. begin
27. process(A, B, OP)
28. begin
29. case (OP) is
30. when "01" => ALUR <= ("00000000" & A) + ("00000000" & B);
31. when "10" => ALUR <= ("00000000" & A) + ("11111111" & not B) + "0000000000000001";
32. when "11" => ALUR <= ("00000000" & A) and ("00000000" & B);
33. when others => ALUR <= ("00000000" & B);
34. end case;
35. end process;
36. OUTPUT <= ALUR(7 downto 0);
37. OVERFLOW <= ALUR(8) OR ALUR(9) OR ALUR(10) OR ALUR(11) OR ALUR(12) OR ALUR(13) OR
    ALUR(14) OR ALUR(15);
38. 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 <= "00000101";
116. ACC_WR <= '1';
117. ACC_RST <= '0';
118. IN_SEL <= "10";
119. OP <= "11";
120. when RUN_CALC2 =>
121. RAM_WR <= '0';
122. RAM_ADDR <= "01";
123. CONST <= "00000000";
124. ACC_WR <= '1';
125. ACC_RST <= '0';
126. IN_SEL <= "01";
127. OP <= "01";
128. when RUN_CALC3 =>
129. RAM_WR <= '0';
130. RAM_ADDR <= "00";
131. CONST <= "00001010";
132. ACC_WR <= '1';
133. ACC_RST <= '0';
134. IN_SEL <= "10";
135. OP <= "01";
136. when RUN_CALC4 =>
137. RAM_WR <= '0';
138. RAM_ADDR <= "00";
139. CONST <= "00000000";
140. ACC_WR <= '1';
141. ACC_RST <= '0';
142. IN_SEL <= "01";
143. OP <= "10";
144. when IDLE =>
145. RAM_WR <= '0';
146. RAM_ADDR <= "00";
147. CONST <= "00000000";
148. ACC_WR <= '0';
149. ACC_RST <= '0';
150. IN_SEL <= "00";
151. OP <= "00";
152. when others =>
153. RAM_WR <= '0';
154. RAM_ADDR <= "00";
155. CONST <= "00000000";
```

```
156. ACC_WR <= '0';
157. ACC_RST <= '0';
158. IN_SEL <= "00";
159. OP <= "00";
160. end case;
161. end process;
162. 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;

```

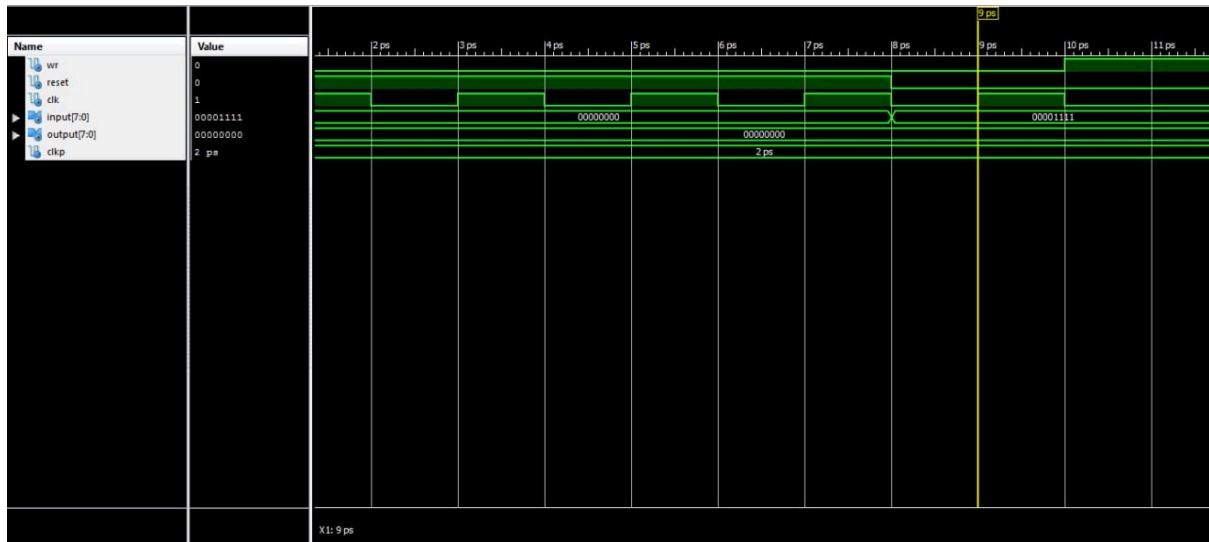


Рис. 2 – Часова діаграма ACC

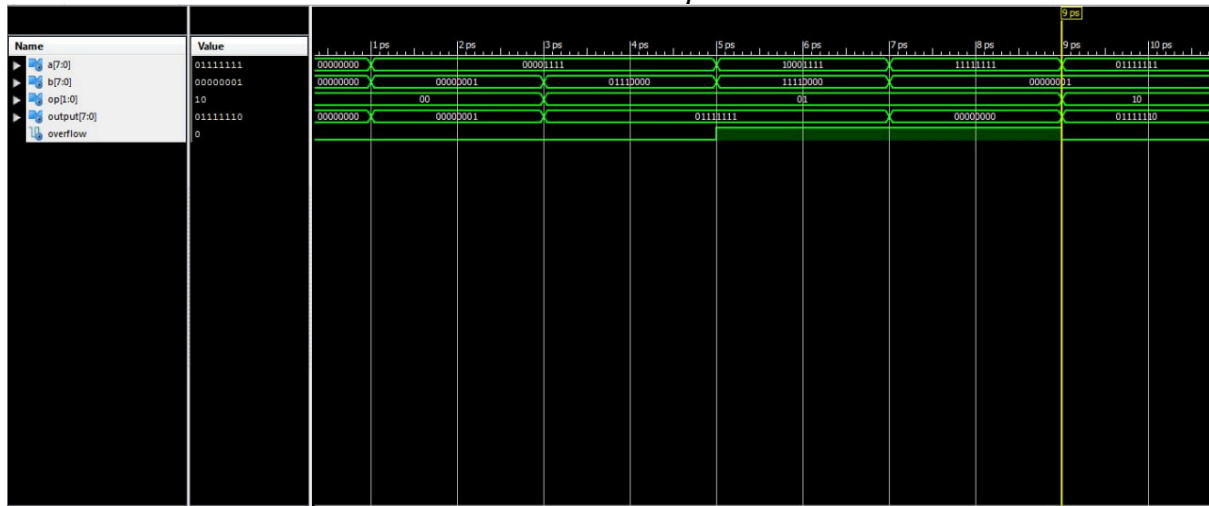


Рис. 3 – Часова діаграма ALU

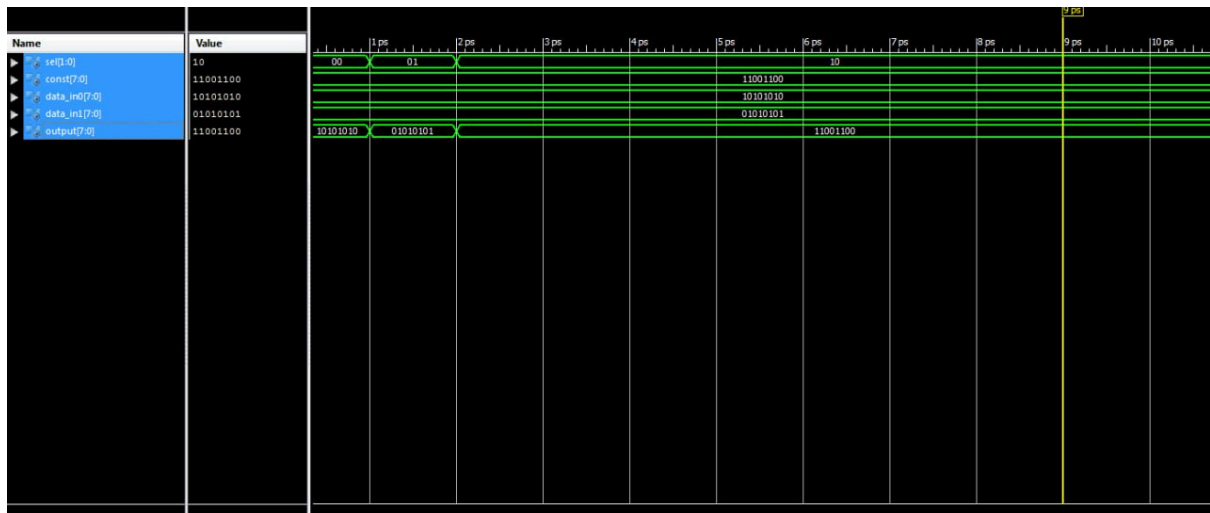


Рис. 4 – Часова діаграма MUX

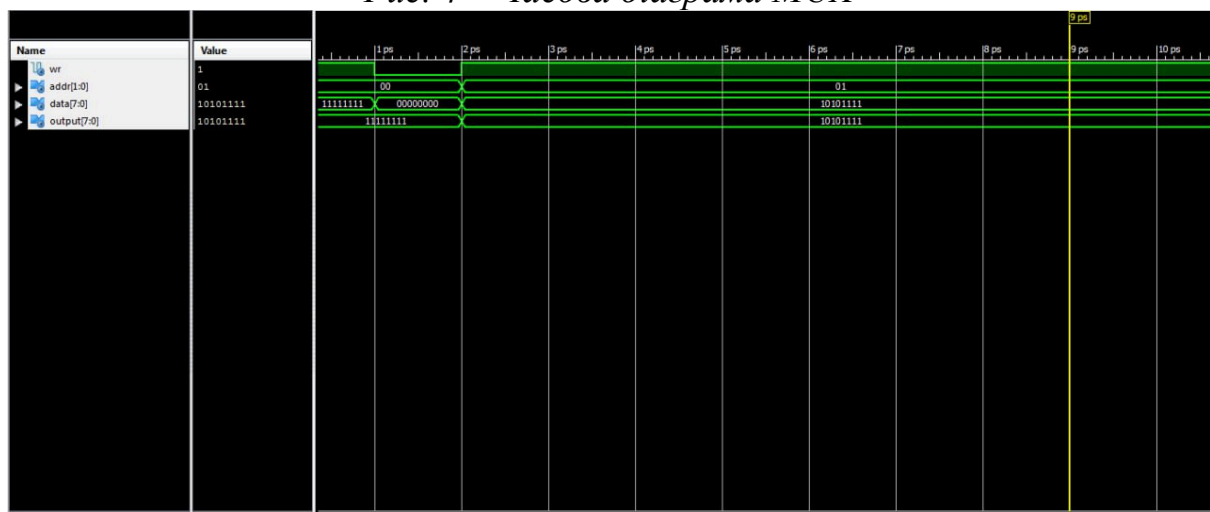


Рис. 5 – Часова діаграма RAM



Рис 6. – Часова діаграма SEG_DECODER

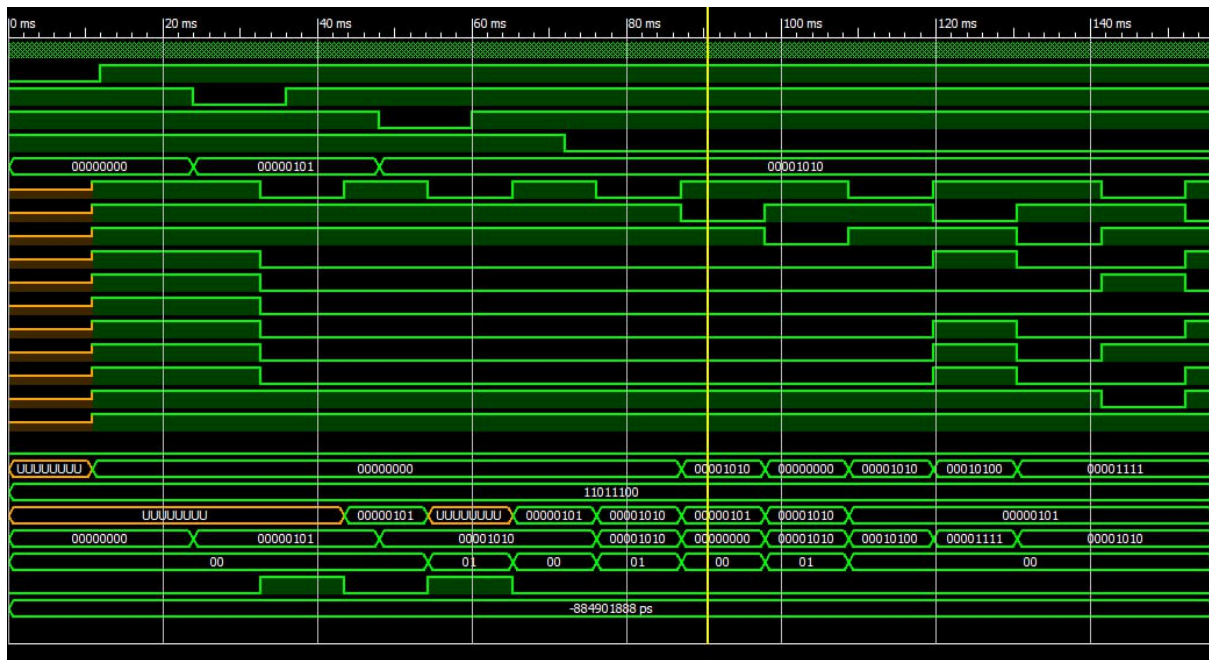


Рис 7. – Часова діаграма TopLevel

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;
29. RAMOUT: OUT STD_LOGIC_VECTOR(7 downto 0);
30. ALUOUT: OUT STD_LOGIC_VECTOR(7 downto 0);
31. RAMA: OUT STD_LOGIC_VECTOR(1 downto 0);
32. RAMWR: OUT STD_LOGIC;
33. OVERFLOW : OUT STD_LOGIC);
34. END COMPONENT;
35.
36. SIGNAL CLOCK : STD_LOGIC := '0';
37. SIGNAL RESET : STD_LOGIC;

```

```

38. SIGNAL ENTER_OP1 : STD_LOGIC;
39. SIGNAL ENTER_OP2 : STD_LOGIC;
40. SIGNAL CALCULATE : STD_LOGIC;
41. SIGNAL DATA_IN : STD_LOGIC_VECTOR (7 DOWNTO 0);
42. SIGNAL COMMON_0_OUT : STD_LOGIC;
43. SIGNAL COMMON_1_OUT : STD_LOGIC;
44. SIGNAL COMMON_2_OUT : STD_LOGIC;
45. SIGNAL A_OUT : STD_LOGIC;
46. SIGNAL B_OUT : STD_LOGIC;
47. SIGNAL C_OUT : STD_LOGIC;
48. SIGNAL D_OUT : STD_LOGIC;
49. SIGNAL E_OUT : STD_LOGIC;
50. SIGNAL F_OUT : STD_LOGIC;
51. SIGNAL G_OUT : STD_LOGIC;
52. SIGNAL DP_OUT : STD_LOGIC;
53. SIGNAL OVERFLOW : STD_LOGIC;
54. SIGNAL TEST: STD_LOGIC_VECTOR(7 downto 0);
55. SIGNAL TEST2: STD_LOGIC_VECTOR(7 downto 0);
56. signal RAMOUT: STD_LOGIC_VECTOR(7 downto 0);
57. signal ALUOUT: STD_LOGIC_VECTOR(7 downto 0);
58. signal RAMA: STD_LOGIC_VECTOR(1 downto 0);
59. signal RAMWR: STD_LOGIC;
60.
61. -- constant CLOCK_period : time := 166ns;
62. constant CLKP: time := 12ms;--24ms;
63.
64. BEGIN
65.
66. UUT: TopLevel PORT MAP(
67. CLOCK => CLOCK,
68. RESET => RESET,
69. ENTER_OP1 => ENTER_OP1,
70. ENTER_OP2 => ENTER_OP2,
71. CALCULATE => CALCULATE,
72. DATA_IN => DATA_IN,
73. COMMON_0_OUT => COMMON_0_OUT,
74. COMMON_1_OUT => COMMON_1_OUT,
75. COMMON_2_OUT => COMMON_2_OUT,
76. A_OUT => A_OUT,
77. B_OUT => B_OUT,
78. C_OUT => C_OUT,
79. D_OUT => D_OUT,
80. E_OUT => E_OUT,
81. F_OUT => F_OUT,
82. G_OUT => G_OUT,
83. DP_OUT => DP_OUT,
84. OVERFLOW => OVERFLOW,
85. TEST => TEST,
86. RAMOUT => RAMOUT,
87. ALUOUT => ALUOUT,
88. RAMA => RAMA,
89. RAMWR => RAMWR
90. );
91.
92. CLOCK_process: process
93. begin
94. 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 5 to 5 loop

```

```

104. lp2: for j in 10 to 10 loop
105. TEST2 <= std_logic_vector(to_unsigned(to_integer(to_unsigned(i, 8) sll 2) * j + j
    + 10, 8));
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;

```

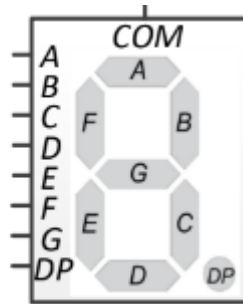


Рис.8 – 7-сегментний індикатор

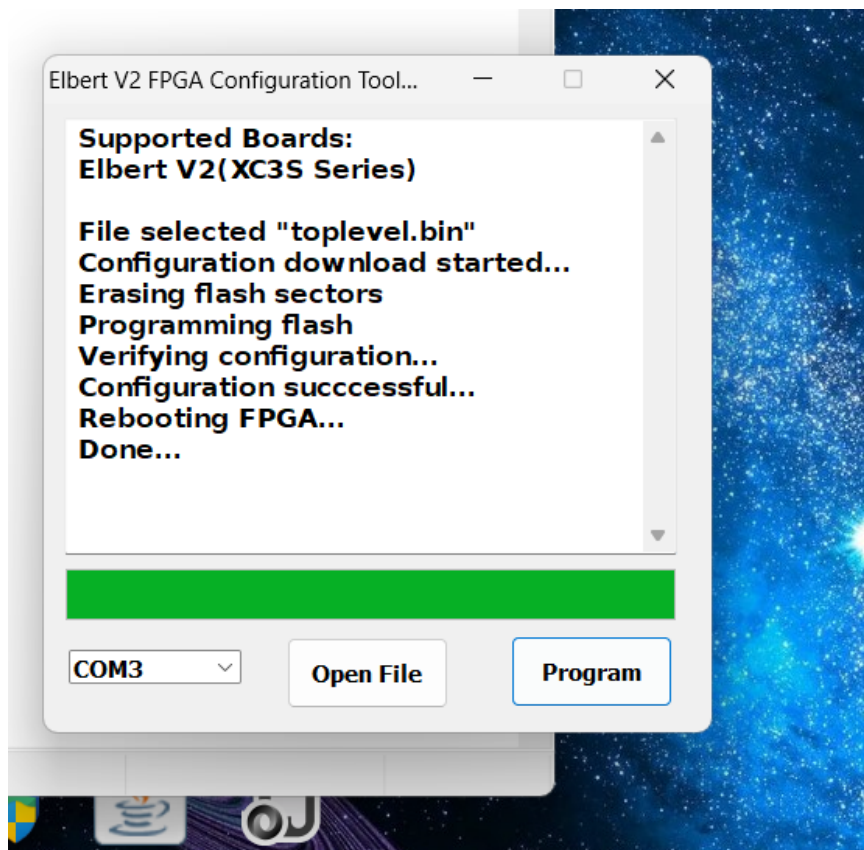


Рис.9 – Успішна прошивка

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