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 №15**

Виконав:

ст. гр. КІ-201

Коростенська С.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:**

**Зображення, що містить схема, текст, План, Креслення

Автоматично згенерований опис**

*Рис. 1 – Top Level*

|  |
| --- |
| Файл ACC.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity ACC is  Port ( WR : in STD\_LOGIC;  RESET : in STD\_LOGIC;  CLK : in STD\_LOGIC;  INPUT : in STD\_LOGIC\_VECTOR (7 downto 0);  OUTPUT : out STD\_LOGIC\_VECTOR (7 downto 0));  end ACC;  architecture ACC\_arch of ACC is  signal DATA : STD\_LOGIC\_VECTOR (7 downto 0);  begin  process (CLK)  begin  if rising\_edge(CLK) then  if RESET = '1' then  DATA <= (others => '0');  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;  use IEEE.NUMERIC\_STD.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.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  component FullAdder8 is  Port ( A: in STD\_LOGIC\_VECTOR (7 downto 0);  B: in STD\_LOGIC\_VECTOR (7 downto 0);  Ci: in STD\_LOGIC;  S: out STD\_LOGIC\_VECTOR (7 downto 0);  Co: out STD\_LOGIC);  end component;  signal AdderOut: STD\_LOGIC\_VECTOR(7 downto 0) := (others => '0');  signal Carry: STD\_LOGIC;  signal SEL: STD\_LOGIC := '0';  signal MuxB: STD\_LOGIC\_VECTOR(7 downto 0) := (others => '0');  begin  MuxB <= B when (SEL = '0') else not B;  Adder: FullAdder8 port map(  A => A,  B => MuxB,  Ci => SEL,  S => AdderOut,  Co => Carry  );  ALU\_EX: process (A, B, OP, Carry, AdderOut)  begin  case (OP) is  when "00" => OUTPUT <= A nor B; OVERFLOW <= '0';  when "01" => SEL <= '0'; OUTPUT <= AdderOut; OVERFLOW <= Carry;  when "10" => SEL <= '1'; OUTPUT <= AdderOut; OVERFLOW <= not Carry;  when others => OUTPUT <= B; OVERFLOW <= '1';  end case;  end process;  end ALU\_Behavioral; |

|  |
| --- |
| Файл CPU.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity CPU is  port( ENTER\_OP1 : IN STD\_LOGIC;  ENTER\_OP2 : IN STD\_LOGIC;  CALCULATE : IN STD\_LOGIC;  RESET : IN STD\_LOGIC;  CLOCK : IN STD\_LOGIC;  RAM\_WR : OUT STD\_LOGIC;  RAM\_ADDR : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);  CONST : 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 : OUT STD\_LOGIC\_VECTOR(1 DOWNTO 0));  end CPU;    architecture CPU\_arch of CPU is  type STATE\_TYPE is (RST, IDLE, LOAD\_OP1, LOAD\_OP2, RUN\_CALC0, RUN\_CALC1, RUN\_CALC2, RUN\_CALC3, RUN\_CALC4, FINISH);  signal CUR\_STATE : STATE\_TYPE;  signal NEXT\_STATE : STATE\_TYPE;  begin  SYNC\_PROC: process (CLOCK)  begin  if (rising\_edge(CLOCK)) then  if (RESET = '1') then  CUR\_STATE <= RST;  else  CUR\_STATE <= NEXT\_STATE;  end if;  end if;  end process;      NEXT\_STATE\_DECODE: process (CLOCK)  begin  NEXT\_STATE <= CUR\_STATE;    case(CUR\_STATE) is  when RST =>  NEXT\_STATE <= IDLE;  when IDLE =>  if (ENTER\_OP1 = '1') then  NEXT\_STATE <= LOAD\_OP1;  elsif (ENTER\_OP2 = '1') then  NEXT\_STATE <= LOAD\_OP2;  elsif (CALCULATE = '1') then  NEXT\_STATE <= RUN\_CALC0;  else  NEXT\_STATE <= IDLE;  end if;  when LOAD\_OP1 =>  NEXT\_STATE <= IDLE;  when LOAD\_OP2 =>  NEXT\_STATE <= IDLE;  when RUN\_CALC0 =>  NEXT\_STATE <= RUN\_CALC1;  when RUN\_CALC1 =>  NEXT\_STATE <= RUN\_CALC2;  when RUN\_CALC2 =>  NEXT\_STATE <= RUN\_CALC3;  when RUN\_CALC3 =>  NEXT\_STATE <= RUN\_CALC4;  when RUN\_CALC4 =>  NEXT\_STATE <= FINISH;  when FINISH =>  NEXT\_STATE <= FINISH;  when others =>  NEXT\_STATE <= IDLE;  end case;  end process;  OUTPUT\_DECODE: process (CUR\_STATE)  begin  case (CUR\_STATE) is  when RST =>  RAM\_WR <= '0';  RAM\_ADDR <= "00";  CONST <= "00000000";  ACC\_WR <= '0';  ACC\_RST <= '1';  IN\_SEL <= "00";  OP <= "00";  when LOAD\_OP1 =>  RAM\_WR <= '1';  RAM\_ADDR <= "00";  CONST <= "00000000";  ACC\_WR <= '0';  ACC\_RST <= '1';  IN\_SEL <= "00";  OP <= "01";  when LOAD\_OP2 =>  RAM\_WR <= '1';  RAM\_ADDR <= "01";  CONST <= "00000000";  ACC\_WR <= '0';  ACC\_RST <= '1';  IN\_SEL <= "00";  OP <= "01";  when RUN\_CALC0 =>  RAM\_WR <= '0';  RAM\_ADDR <= "00";  CONST <= "00000000";  ACC\_WR <= '1';  ACC\_RST <= '0';  IN\_SEL <= "01";  OP <= "01";  when RUN\_CALC1 =>  RAM\_WR <= '0';  RAM\_ADDR <= "01";  CONST <= "00000000";  ACC\_WR <= '1';  ACC\_RST <= '0';  IN\_SEL <= "01";  OP <= "00";  when RUN\_CALC2 =>  RAM\_WR <= '0';  RAM\_ADDR <= "01";  CONST <= "00000000";  ACC\_WR <= '1';  ACC\_RST <= '0';  IN\_SEL <= "01";  OP <= "01";  when RUN\_CALC3 =>  RAM\_WR <= '0';  RAM\_ADDR <= "00";  CONST <= "00001010";  ACC\_WR <= '1';  ACC\_RST <= '0';  IN\_SEL <= "10";  OP <= "01";  when RUN\_CALC4 =>  RAM\_WR <= '0';  RAM\_ADDR <= "00";  CONST <= "00000011";  ACC\_WR <= '1';  ACC\_RST <= '0';  IN\_SEL <= "10";  OP <= "10";  when IDLE =>  RAM\_WR <= '0';  RAM\_ADDR <= "00";  CONST <= "00000000";  ACC\_WR <= '0';  ACC\_RST <= '0';  IN\_SEL <= "00";  OP <= "01";  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; |

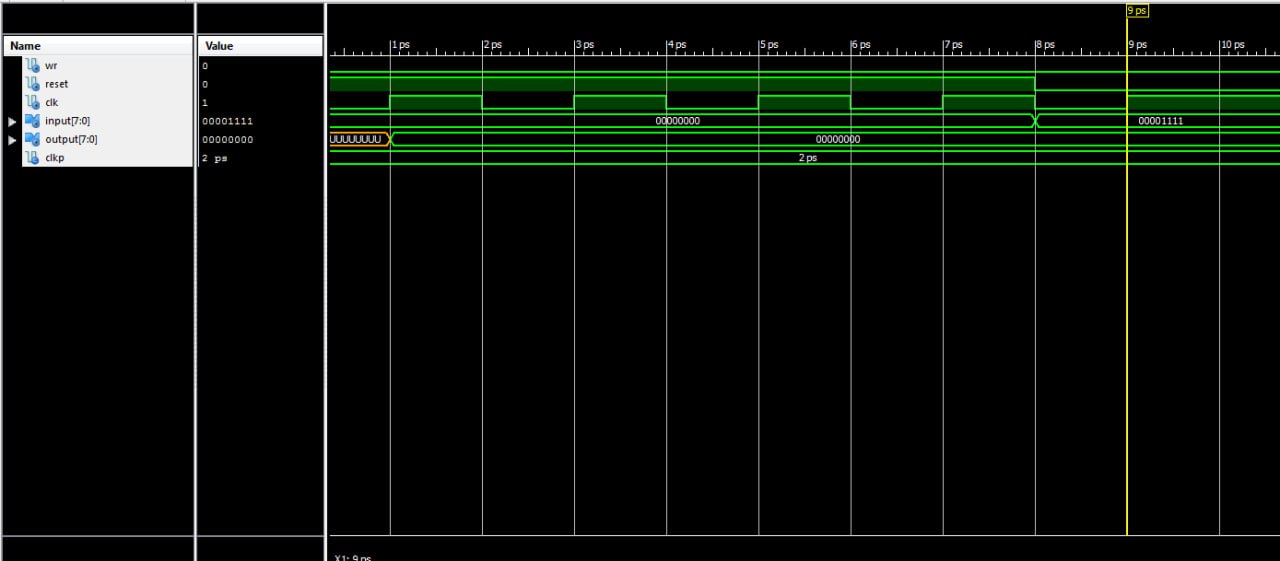
|  |
| --- |
| Файл FullAdder.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity FullAdder is  Port (A: in STD\_LOGIC;  B: in STD\_LOGIC;  Ci: in STD\_LOGIC;  S: out STD\_LOGIC;  Co: out STD\_LOGIC);  end FullAdder;  architecture Behavioral of FullAdder is  begin  S <= (A xor B) xor Ci;  Co <= (A and B) or ((A xor B) and Ci);  end Behavioral; |

|  |
| --- |
| Файл FullAdder8.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity FullAdder8 is  Port ( A: in STD\_LOGIC\_VECTOR (7 downto 0);  B: in STD\_LOGIC\_VECTOR (7 downto 0);  Ci: in STD\_LOGIC;  S: out STD\_LOGIC\_VECTOR (7 downto 0);  Co: out STD\_LOGIC);  end FullAdder8;  architecture Behavioral of FullAdder8 is  component FullAdder is  Port (A: in STD\_LOGIC;  B: in STD\_LOGIC;  Ci: in STD\_LOGIC;  S: out STD\_LOGIC;  Co: out STD\_LOGIC);  end component;  signal carry: STD\_LOGIC\_VECTOR(8 downto 0) := (others => '0');  begin  carry(0) <= Ci;  FullAdderGenerate: for i in 0 to 7 generate  adder: FullAdder port map(  A => A(i),  B => B(i),  Ci => carry(i),  S => S(i),  Co => carry(i + 1)  );  end generate FullAdderGenerate;  Co <= carry(8);  end Behavioral; |

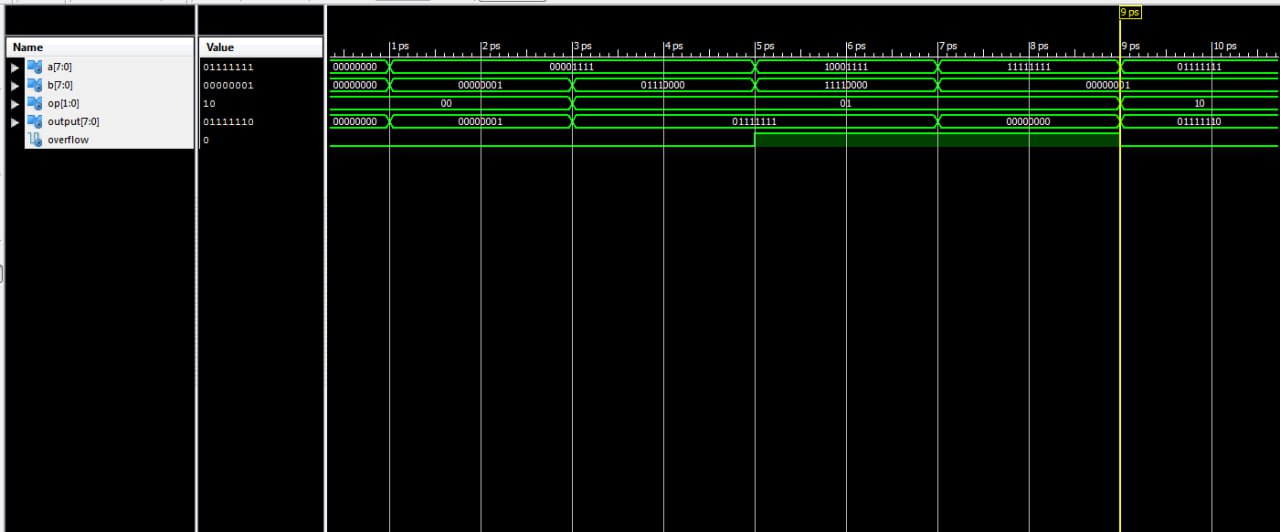
|  |
| --- |
| Файл MUX.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity MUX is  PORT(  SEL: in STD\_LOGIC\_VECTOR(1 downto 0);  CONST: in STD\_LOGIC\_VECTOR(7 downto 0);  --CONST1: in STD\_LOGIC\_VECTOR()  DATA\_IN0: in STD\_LOGIC\_VECTOR(7 downto 0);  DATA\_IN1: in STD\_LOGIC\_VECTOR(7 downto 0);  OUTPUT: out STD\_LOGIC\_VECTOR(7 downto 0)  );  end MUX;  architecture Behavioral of MUX is  begin  process (SEL, DATA\_IN0, DATA\_IN1, CONST)  begin  if (SEL = "00") then  OUTPUT <= DATA\_IN0;  elsif (SEL = "01") then  OUTPUT <= DATA\_IN1;  else  OUTPUT <= CONST;  end if;  end process;  end Behavioral; |

|  |
| --- |
| Файл RAM.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity RAM is  port(  WR : IN STD\_LOGIC;  ADDR : IN STD\_LOGIC\_VECTOR(1 DOWNTO 0);  DATA : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);  OUTPUT : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0)  );  end RAM;  architecture RAM\_arch of RAM is  type ram\_type is array (3 downto 0) of STD\_LOGIC\_VECTOR(7 downto 0);  signal UNIT : ram\_type;  begin  process(ADDR, UNIT, WR, DATA)  begin  if (WR = '1') then  UNIT(conv\_integer(ADDR)) <= DATA;  end if;  OUTPUT <= UNIT(conv\_integer(ADDR));  end process;  end RAM\_arch; |

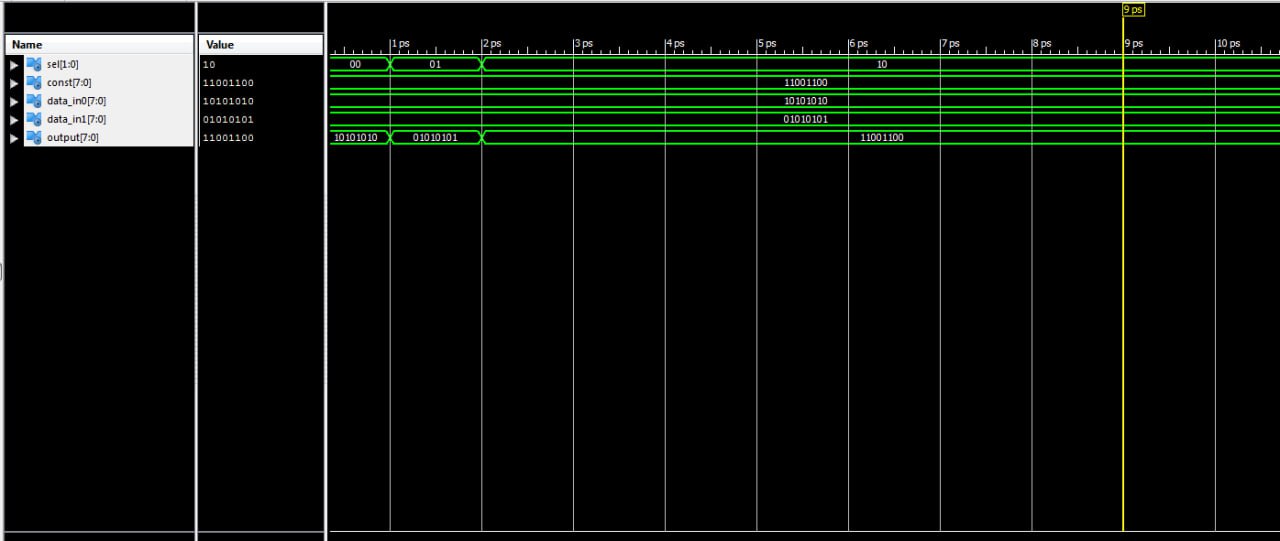
|  |
| --- |
| Файл SEG\_DECODER.vhd  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity SEG\_DECODER is  port( CLOCK : IN STD\_LOGIC;  RESET : IN STD\_LOGIC;  ACC\_DATA\_OUT\_BUS : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0);  COMM\_ONES : 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 SEG\_DECODER;  architecture Behavioral of SEG\_DECODER 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 new entry  hex\_src := hex\_src(hex\_src'left - 1 downto hex\_src'right) & '0' ; -- shift src + pad with 0  end loop ;  HONDREDS\_BUS <= bcd (11 downto 8);  DECS\_BUS <= bcd (7 downto 4);  ONES\_BUS <= bcd (3 downto 0);    end process BIN\_TO\_BCD;    INDICATE : process(CLOCK)  type DIGIT\_TYPE is (ONES, DECS, HUNDREDS);    variable CUR\_DIGIT : DIGIT\_TYPE := ONES;  variable DIGIT\_VAL : STD\_LOGIC\_VECTOR(3 downto 0) := "0000";  variable DIGIT\_CTRL : STD\_LOGIC\_VECTOR(6 downto 0) := "0000000";  variable COMMONS\_CTRL : STD\_LOGIC\_VECTOR(2 downto 0) := "000";    begin  if (rising\_edge(CLOCK)) then  if(RESET = '0') then  case CUR\_DIGIT is  when ONES =>  DIGIT\_VAL := ONES\_BUS;  CUR\_DIGIT := DECS;  COMMONS\_CTRL := "001";  when DECS =>  DIGIT\_VAL := DECS\_BUS;  CUR\_DIGIT := HUNDREDS;  COMMONS\_CTRL := "010";  when HUNDREDS =>  DIGIT\_VAL := HONDREDS\_BUS;  CUR\_DIGIT := ONES;  COMMONS\_CTRL := "100";  when others =>  DIGIT\_VAL := ONES\_BUS;  CUR\_DIGIT := ONES;  COMMONS\_CTRL := "000";  end case;    case DIGIT\_VAL is --abcdefg  when "0000" => DIGIT\_CTRL := "1111110";  when "0001" => DIGIT\_CTRL := "0110000";  when "0010" => DIGIT\_CTRL := "1101101";  when "0011" => DIGIT\_CTRL := "1111001";  when "0100" => DIGIT\_CTRL := "0110011";  when "0101" => DIGIT\_CTRL := "1011011";  when "0110" => DIGIT\_CTRL := "1011111";  when "0111" => DIGIT\_CTRL := "1110000";  when "1000" => DIGIT\_CTRL := "1111111";  when "1001" => DIGIT\_CTRL := "1111011";  when others => DIGIT\_CTRL := "0000000";  end case;  else  DIGIT\_VAL := ONES\_BUS;  CUR\_DIGIT := ONES;  COMMONS\_CTRL := "000";  end if;    COMM\_ONES <= not COMMONS\_CTRL(0);  COMM\_DECS <= not COMMONS\_CTRL(1);  COMM\_HUNDREDS <= not COMMONS\_CTRL(2);    SEG\_A <= not DIGIT\_CTRL(6);  SEG\_B <= not DIGIT\_CTRL(5);  SEG\_C <= not DIGIT\_CTRL(4);  SEG\_D <= not DIGIT\_CTRL(3);  SEG\_E <= not DIGIT\_CTRL(2);  SEG\_F <= not DIGIT\_CTRL(1);  SEG\_G <= not DIGIT\_CTRL(0);  DP <= '1';    end if;  end process INDICATE;  end Behavioral; |



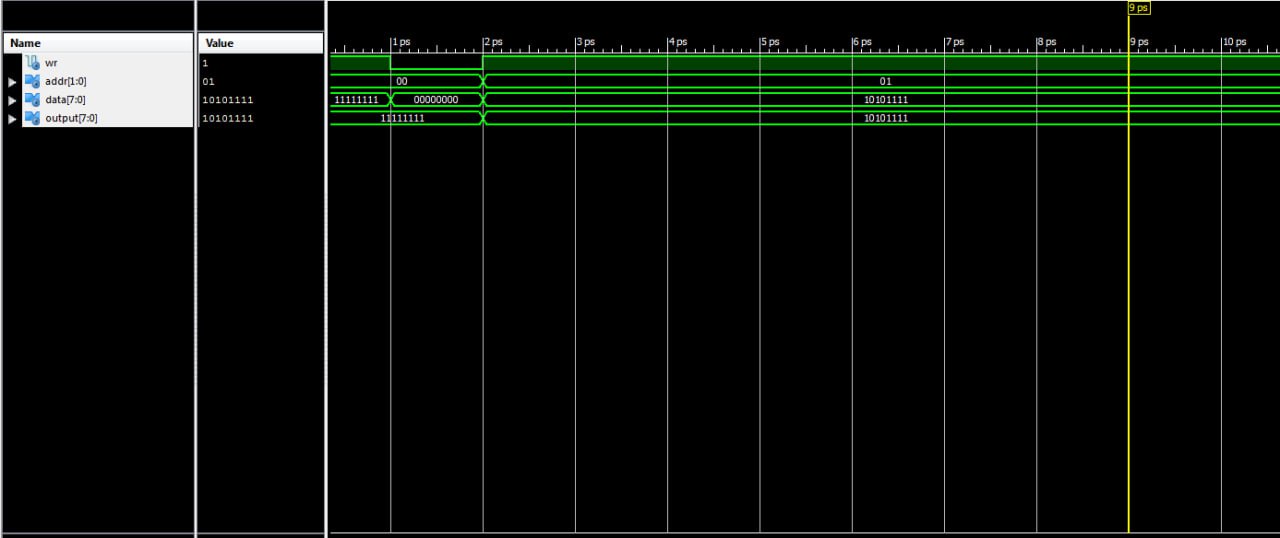
*Рис. 2 – Часова діаграма ACC*

**

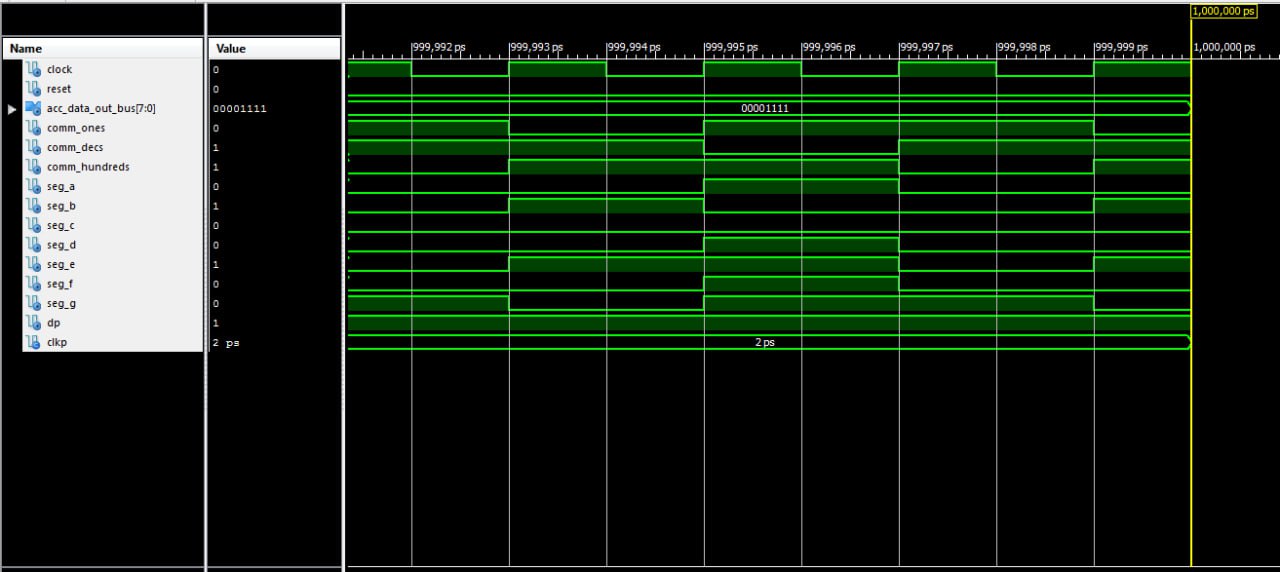
*Рис. 3 – Часова діаграма ALU*

**

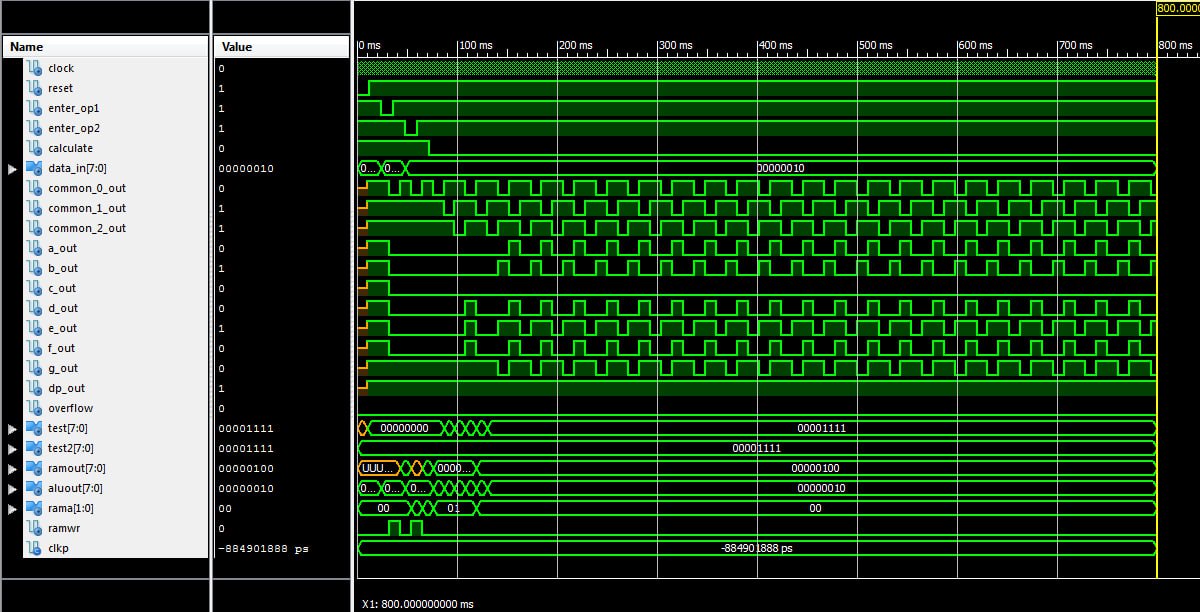
*Рис. 4 – Часова діаграма MUX*

**

*Рис. 5 – Часова діаграма RAM*

**

*Рис 6. – Часова діграма SEG\_DECODER*

**

*Рис 7. – Часова діграма TopLevel*

|  |
| --- |
| *Файл TopLevelTest.vhd*  *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( CLOCK : IN STD\_LOGIC;*  *RESET : 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);*  *COMMON\_0\_OUT : OUT STD\_LOGIC;*  *COMMON\_1\_OUT : OUT STD\_LOGIC;*  *COMMON\_2\_OUT : OUT STD\_LOGIC;*  *TEST: OUT STD\_LOGIC\_VECTOR(7 downto 0);*  *A\_OUT : OUT STD\_LOGIC;*  *B\_OUT : OUT STD\_LOGIC;*  *C\_OUT : OUT STD\_LOGIC;*  *D\_OUT : OUT STD\_LOGIC;*  *E\_OUT : OUT STD\_LOGIC;*  *F\_OUT : OUT STD\_LOGIC;*  *G\_OUT : OUT STD\_LOGIC;*  *DP\_OUT : 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 TEST1: STD\_LOGIC\_VECTOR(7 downto 0);*  *SIGNAL TEST2: STD\_LOGIC\_VECTOR(7 downto 0);*    *-- constant CLOCK\_period : time := 166ns;*  *constant CLKP: time := 24ms;--48ms;*  *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*  *);*    *CLOCK\_process: process*  *begin*  *CLOCK <= '0';*  *wait for 83ns;*  *CLOCK <= '1';*  *wait for 83ns;*  *end process;*  *-- \*\*\* Test Bench - User Defined Section \*\*\**  *tb : PROCESS*  *BEGIN*  *--ENTER\_OP1 <= '0';*  *--ENTER\_OP2 <= '0';*  *--CALCULATE <= '0';*  *--DATA\_IN <= (others => '0');*  *--RESET <= '1';*  *--wait for CLKP \* 4;*  *--RESET <= '0';*  *--DATA\_IN <= "00000000"; -- A*  *--ENTER\_OP1 <= '1';*  *--wait for CLKP \* 4;*  *--ENTER\_OP1 <= '0';*  *--wait for CLKP \* 4;*  *--DATA\_IN <= "00000001"; -- B*  *--ENTER\_OP2 <= '1';*  *--wait for CLKP \* 4;*  *--ENTER\_OP2 <= '0';*  *--wait for CLKP \* 4;*  *--CALCULATE <= '1'; -- START CALCULATION*    *lp1: for i in 1 to 1 loop*  *lp2: for j in 2 to 2 loop*  *TEST1 <= std\_logic\_vector(to\_unsigned(i, 8)) nor std\_logic\_vector(to\_unsigned(j, 8));*  *TEST2 <= std\_logic\_vector(to\_unsigned(to\_integer(signed(std\_logic\_vector(to\_signed(i, 8)) nor std\_logic\_vector(to\_unsigned(j, 8)))) + j + 7, 8));*  *ENTER\_OP1 <= '1';*  *ENTER\_OP2 <= '1';*  *CALCULATE <= '1';*  *DATA\_IN <= (others => '0');*  *RESET <= '0';*  *wait for CLKP;*  *RESET <= '1';*  *wait for CLKP;*  *DATA\_IN <= std\_logic\_vector(to\_unsigned(i, 8)); -- A*  *ENTER\_OP1 <= '0';*  *wait for CLKP;*  *ENTER\_OP1 <= '1';*  *wait for CLKP;*  *DATA\_IN <= std\_logic\_vector(to\_unsigned(j, 8)); -- B*  *ENTER\_OP2 <= '0';*  *wait for CLKP;*  *ENTER\_OP2 <= '1';*  *wait for CLKP;*  *CALCULATE <= '0'; -- START CALCULATION*  *wait for CLKP\* 7;*  *assert TEST = TEST2 severity FAILURE;*  *wait for CLKP;*  *end loop;*  *end loop;*    *WAIT; -- will wait forever*  *END PROCESS;*  *-- \*\*\* End Test Bench - User Defined Section \*\*\**  *END;* |

*Зображення, що містить текст, знімок екрана, схема, Шрифт

Автоматично згенерований опис*

*Рис.8 – 7-сегментний індикатор*

Зображення, що містить текст, знімок екрана, монітор, програмне забезпечення

Автоматично згенерований опис

*Рис.9 – Успішна прошивка*

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