

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 №1

Виконав:
ст. гр. КІ-201
Бандрівський П.Р
Прийняв:
ст. викладач
каф. ЕОМ
Козак Н. Б.

Львів 2024

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

Виконання роботи:

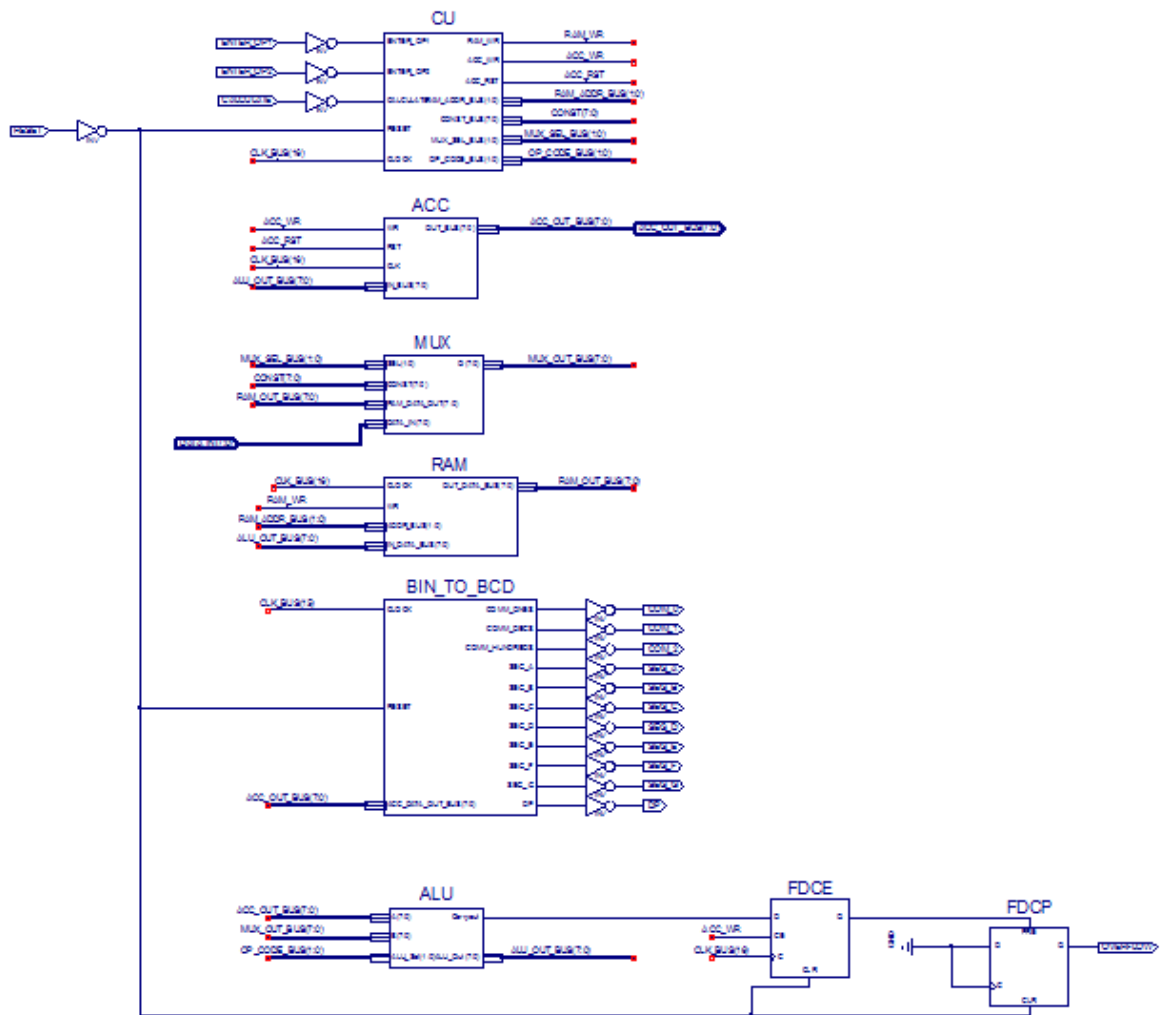


Рис. 1 – Top Level

Файл ACC.vhd

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ACC is

Port (WR : in STD_LOGIC;

RST : in STD_LOGIC;

CLK : in STD_LOGIC;

IN_BUS : in STD_LOGIC_VECTOR (7 downto 0);

OUT_BUS : 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 RST = '1' then
                DATA <= (others => '0');
            elsif WR = '1' then
                DATA <= IN_BUS;
            end if;
        end if;
    end process;

    OUT_BUS <= DATA;

end ACC_arch;

```

```

Файл ALU.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;
entity ALU is
    Port (
        A, B : in STD_LOGIC_VECTOR(7 downto 0);
        ALU_Sel : in STD_LOGIC_VECTOR(1 downto 0);
        ALU_Out : out STD_LOGIC_VECTOR(7 downto 0);
        Carryout : out std_logic
    );
end ALU;
architecture Behavioral of ALU is

    signal ALU_Result : std_logic_vector (8 downto 0);

begin
    process(A,B,ALU_Sel)
    begin
        case(ALU_Sel) is
            when "01" =>
                ALU_Result <= ('0' & A) + ('0' & B);

```

```

when "10" =>
  ALU_Result <= ('0' & A) and ('0' & B);
when "11" =>
  case(B) is
    when x"00" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 0);
    when x"01" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 1);
    when x"02" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 2);
    when x"03" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 3);
    when x"04" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 4);
    when x"05" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 5);
    when x"06" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 6);
    when x"07" => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 7);
    when others => ALU_Result <= std_logic_vector(unsigned(('0' & A))
sll 0);
  end case;
  ALU_Result(8) <= '0';
  when others => ALU_Result <= ('0' & B);
  end case;
end process;
ALU_Out <= ALU_Result(7 downto 0);
Carryout <= ALU_Result(8);
end Behavioral;

```

```

Файл CPU.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CU 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_BUS : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    CONST_BUS : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    ACC_WR : OUT STD_LOGIC;
    ACC_RST : OUT STD_LOGIC;
    MUX_SEL_BUS : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    OP_CODE_BUS : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
end CU;

```

architecture CU_arch of CU 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
    CONST_BUS <= "00001010";

```

```

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 (CUR_STATE, ENTER_OP1,
ENTER_OP2, CALCULATE)
begin
    --declare default state for next_state to avoid latches
    NEXT_STATE <= CUR_STATE; --default is to stay in current state
    --insert statements to decode next_state
    --below is a simple example
    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 =>
            MUX_SEL_BUS <= "00";
            OP_CODE_BUS <= "00";
            RAM_ADDR_BUS <= "00";
            RAM_WR <= '0';
            ACC_RST <= '1';
            ACC_WR <= '0';
        when IDLE =>
            MUX_SEL_BUS <= "00";
            OP_CODE_BUS <= "00";
            RAM_ADDR_BUS <= "00";

```

```

RAM_WR    <= '0';
ACC_RST   <= '0';
ACC_WR    <= '0';
when LOAD_OP1 =>
    MUX_SEL_BUS <= "00";
    OP_CODE_BUS <= "00";
    RAM_ADDR_BUS <= "00";
    RAM_WR      <= '1';
    ACC_RST     <= '0';
    ACC_WR      <= '1';
when LOAD_OP2 =>
    MUX_SEL_BUS <= "00";
    OP_CODE_BUS <= "00";
    RAM_ADDR_BUS <= "01";
    RAM_WR      <= '1';
    ACC_RST     <= '0';
    ACC_WR      <= '1';
when RUN_CALC0 =>
    MUX_SEL_BUS <= "01";
    OP_CODE_BUS <= "00";
    RAM_ADDR_BUS <= "00";
    RAM_WR      <= '0';
    ACC_RST     <= '0';
    ACC_WR      <= '1';
when RUN_CALC1 =>
    MUX_SEL_BUS <= "01";
    OP_CODE_BUS <= "01";
    RAM_ADDR_BUS <= "01";
    RAM_WR      <= '0';
    ACC_RST     <= '0';
    ACC_WR      <= '1';
when RUN_CALC2 =>
    MUX_SEL_BUS <= "11";
    OP_CODE_BUS <= "01";
    RAM_ADDR_BUS <= "00";
    RAM_WR      <= '0';
    ACC_RST     <= '0';
    ACC_WR      <= '1';
when RUN_CALC3 =>
    MUX_SEL_BUS <= "01";
    OP_CODE_BUS <= "10";
    RAM_ADDR_BUS <= "01";
    RAM_WR      <= '0';

```

```

    ACC_RST    <= '0';
    ACC_WR     <= '1';
when RUN_CALC4 =>
    MUX_SEL_BUS <= "01";
    OP_CODE_BUS <= "11";
    RAM_ADDR_BUS <= "00";
    RAM_WR     <= '0';
    ACC_RST    <= '0';
    ACC_WR     <= '1';
when FINISH  =>
    MUX_SEL_BUS <= "00";
    OP_CODE_BUS <= "00";
    RAM_ADDR_BUS <= "00";
    RAM_WR     <= '0';
    ACC_RST    <= '0';
    ACC_WR     <= '0';
when others  =>
    MUX_SEL_BUS <= "00";
    OP_CODE_BUS <= "00";
    RAM_ADDR_BUS <= "00";
    RAM_WR     <= '0';
    ACC_RST    <= '0';
    ACC_WR     <= '0';
end case;
end process;
end CU_arch;

```

```

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);
          RAM_DATA_OUT : in  STD_LOGIC_VECTOR (7 downto 0);
          DATA_IN : in  STD_LOGIC_VECTOR (7 downto 0);
          O : out  STD_LOGIC_VECTOR (7 downto 0));
end MUX;

architecture MUX_arch of MUX is
begin

```



```

PROCESS (SEL, CONST, RAM_DATA_OUT, DATA_IN)
BEGIN
  IF (SEL = "00") THEN
    O <= DATA_IN;
  ELSIF (SEL = "01") THEN
    O <= RAM_DATA_OUT;
  ELSE
    O <= CONST;
  END IF;
END PROCESS;

```

```

end MUX_arch;

```

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( CLOCK : STD_LOGIC;
        WR : IN STD_LOGIC;
        ADDR_BUS : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        IN_DATA_BUS : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        OUT_DATA_BUS : 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(CLOCK, ADDR_BUS, UNIT)
  begin
    if (rising_edge(CLOCK)) then
      if (WR = '1') then
        UNIT(conv_integer(ADDR_BUS)) <= IN_DATA_BUS;
      end if;
    end if;
    OUT_DATA_BUS <= UNIT(conv_integer(ADDR_BUS));
  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 BIN_TO_BCD 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 BIN_TO_BCD;
```

```
architecture Behavioral of BIN_TO_BCD 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    <= COMMONS_CTRL(0);
COMM_DECS    <= COMMONS_CTRL(1);
COMM_HUNDREDS <= COMMONS_CTRL(2);

SEG_A <= DIGIT_CTRL(6);
SEG_B <= DIGIT_CTRL(5);
SEG_C <= DIGIT_CTRL(4);
SEG_D <= DIGIT_CTRL(3);
SEG_E <= DIGIT_CTRL(2);

SEG_F <= DIGIT_CTRL(1);
SEG_G <= DIGIT_CTRL(0);
DP    <= '0';

```

```

end if;
end process INDICATE;

end Behavioral;

```

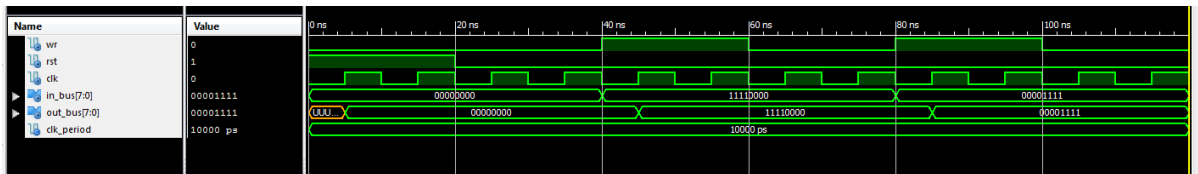


Рис. 2 – Часова діаграма ACC

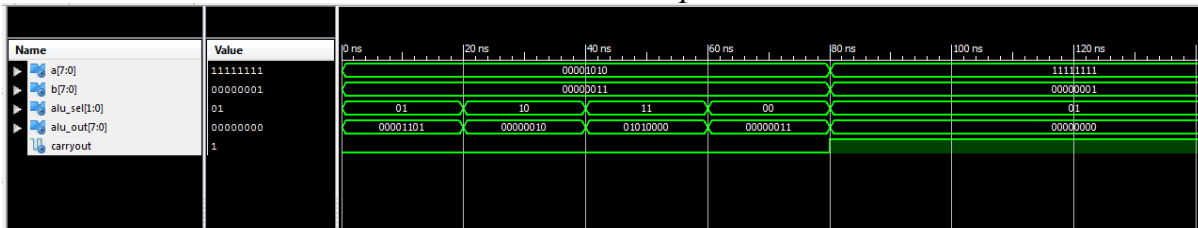


Рис. 3 – Часова діаграма ALU

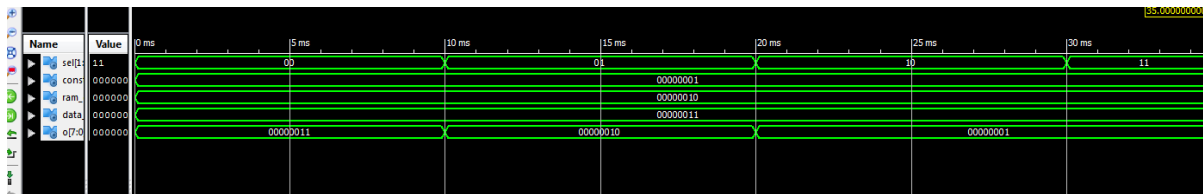


Рис. 4 – Часова діаграма MUX

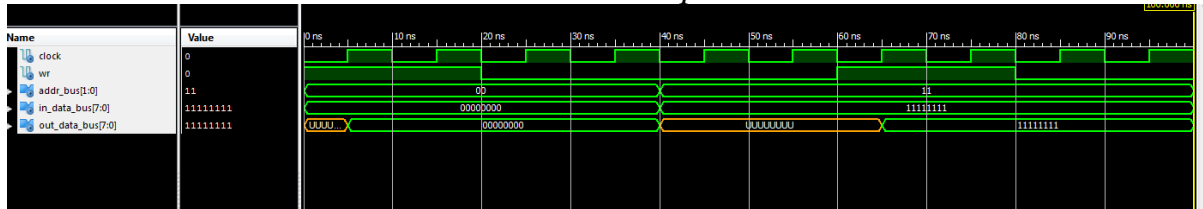


Рис. 5 – Часова діаграма RAM

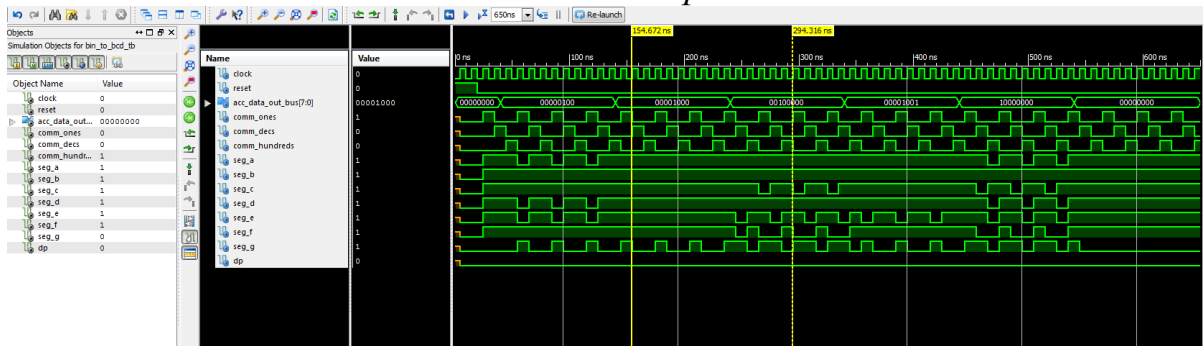


Рис 6. – Часова діаграма SEG_DECODER

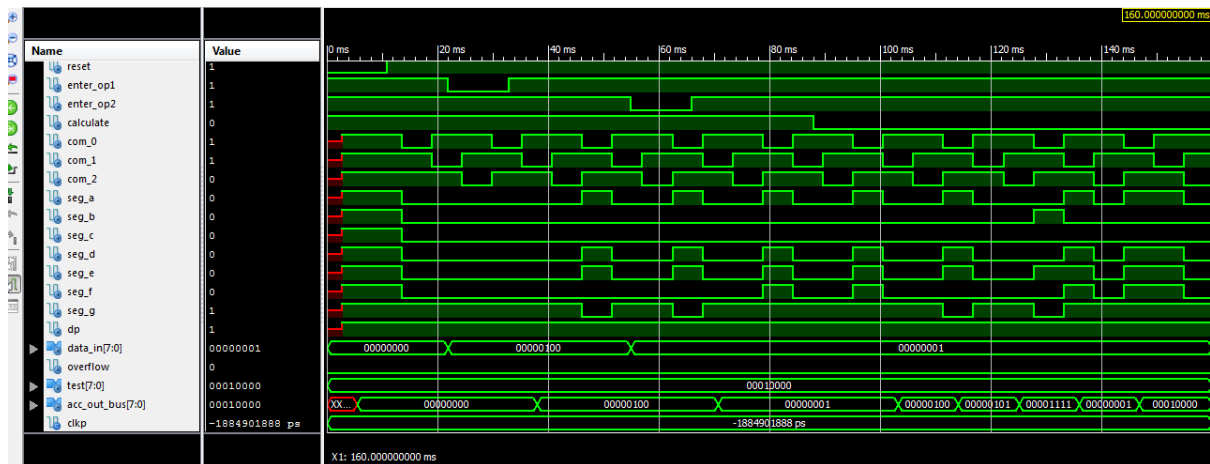


Рис 7. – Часова діаграма TopLevel

Файл TopLevelTest.vhd

-- * Test Bench - User Defined Section *

tb : PROCESS

BEGIN

lp1: for i in 4 to 4 loop

lp2: for j in 1 to 1 loop

ENTER_OP1 <= '1';

ENTER_OP2 <= '1';

CALCULATE <= '1';

case(std_logic_vector(to_unsigned(i, 8))) is

when x"00" =>

TEST <=

std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsigned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) + to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND std_logic_vector(to_unsigned(j, 8))) SLL 0);

when x"01" =>

TEST <=

std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsigned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) + to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND std_logic_vector(to_unsigned(j, 8))) SLL 1);

when x"02" =>

TEST <=

std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsigned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +

```

to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 2);
    when x"03" =>
        TEST <=
std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsig
ned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +
to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 3);
    when x"04" =>
        TEST <=
std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsig
ned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +
to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 4);
    when x"05" =>
        TEST <=
std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsig
ned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +
to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 5);
    when x"06" =>
        TEST <=
std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsig
ned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +
to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 6);
    when x"07" =>
        TEST <=
std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsig
ned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +
to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 7);
    when others =>
        TEST <=
std_logic_vector(signed(std_logic_vector(unsigned(std_logic_vector(to_unsig
ned(to_integer(unsigned(std_logic_vector(to_unsigned(i, 8)))) +
to_integer(unsigned(std_logic_vector(to_unsigned(j, 8)))) + 10, 8)))) AND
std_logic_vector(to_unsigned(j, 8))) SLL 0);
    end case;
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 * 2;
DATA_IN <= (std_logic_vector(to_unsigned(j, 8))); -- B
ENTER_OP2 <= '0';
wait for CLKP;
ENTER_OP2 <= '1';
wait for CLKP * 2;
CALCULATE <= '0'; -- START CALCULATION
REPORT "OP1 = (" & integer'image(i) & ") and OP2 = (" &
integer'image(j) & ") calculation started" SEVERITY NOTE;
wait for CLKP * 7;
assert ACC_OUT_BUS = TEST severity FAILURE;
REPORT "OP1 = (" & integer'image(i) & ") and OP2 = (" &
integer'image(j) & ") calculation finished" SEVERITY NOTE;
wait for CLKP;
end loop;
end loop;
WAIT; -- will wait forever
END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

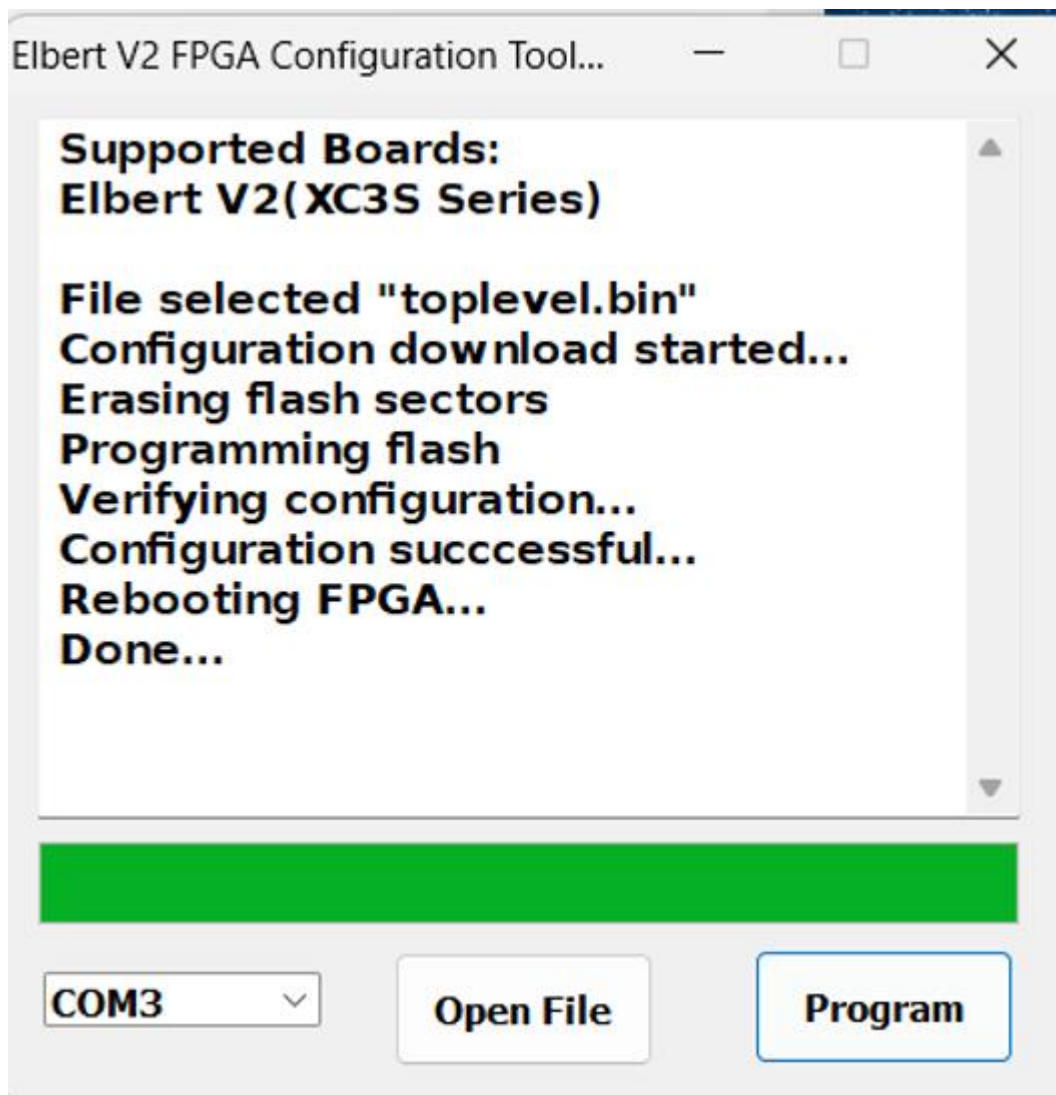



Рис.9 – Успішна прошивка

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