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 – Spartan3A FPGA»

Варіант 9

Виконав:

ст. гр. КІ-202

Іванюк Д. В.

Прийняв:

Козак Н. Б.

Львів 2024

Поведінковий опис цифрового автомата. Перевірка роботи автомата за допомогою стенда Elbert V2 – Spartan 3A FPGA

Мета роботи:

На базі стенда реалізувати цифровий автомат для обчислення значення виразу дотримуючись наступних вимог:

1. Функціонал пристрою повинен бути реалізований згідно отриманого варіанту завдання.
2. Пристрій повинен бути ітераційним АЛП повинен виконувати за один такт одну операцію та реалізованим згідно наступної структурної схеми(рис.3.1).
3. Кожен блок структурної схеми повинен бути реалізований на мові VHDL в окремому файлі Дозволено використовувати всі оператори.
4. Для кожного блока структурної схеми повинен бути згенерований символ.
5. Інтеграція структурних блоків в єдину систему та зі стендом.
6. Кожен структурний блок і схема вцілому повинні бути промодельовані за допомогою симулятора ISim.
7. Формування вхідних даних на шині DATA_IN повинно бути реалізовано за допомогою DIP перемикачів.
8. Керування пристроєм повинно бути реалізовано за допомогою PUSH BUTTON кнопок.
9. Індикація значень операндів при вводі та вивід результату обчислень повинні бути реалізовані за допомогою семи сегментних індикаторів Індикація переповнення в АЛП за допомогою LED D8.
10. Підготувати та захистити звіт.

Вхідні дані:

Пристрій повинен реалізувати обчислення наступного виразу:

9	((OP1 - 4) + OP2 + 15) or 2

Виконання роботи:

1. Створити новий .vhd файл, та реалізувати на ньому мультиплексор MUX.

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

entity MUX is
   Port ( SEL : in  STD_LOGIC_VECTOR (2 downto 0);
          CONST1 : in  STD_LOGIC_VECTOR (7 downto 0);
                       CONST2 : in  STD_LOGIC_VECTOR (7 downto 0);
                       CONST3 : 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, CONST1, CONST2, CONST3, RAM_DATA_OUT, DATA_IN)
          BEGIN
                    IF (SEL = "000") THEN
                              O <= DATA_IN;
                    ELSIF (SEL = "001") THEN
                              O <= RAM_DATA_OUT;
                    ELSIF (SEL = "010") THEN
                              O <= CONST1;
                    ELSIF (SEL = "011") THEN
                              O <= CONST2;
                    ELSIF (SEL = "100") THEN
                              O <= CONST3;
                    END IF;
          END PROCESS;

end MUX_arch;
```

2. Перевірити роботу мультиплексора за допомогою симулятора ISim.

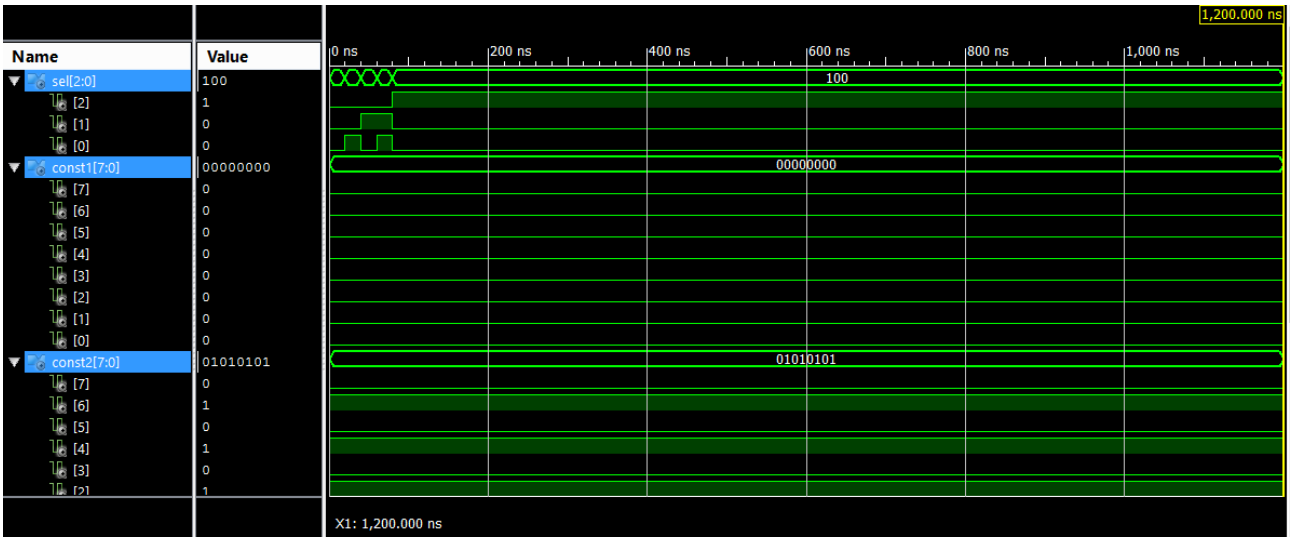

Рис.3.3. Симуляція мультиплексора в ISim.

3. Створити новий .vhd файл, та реалізувати на ньому регістр ACC.

```
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;
```

4. Перевірити роботу регістра ACC (запис/скидання) за допомогою симулятора ISim.

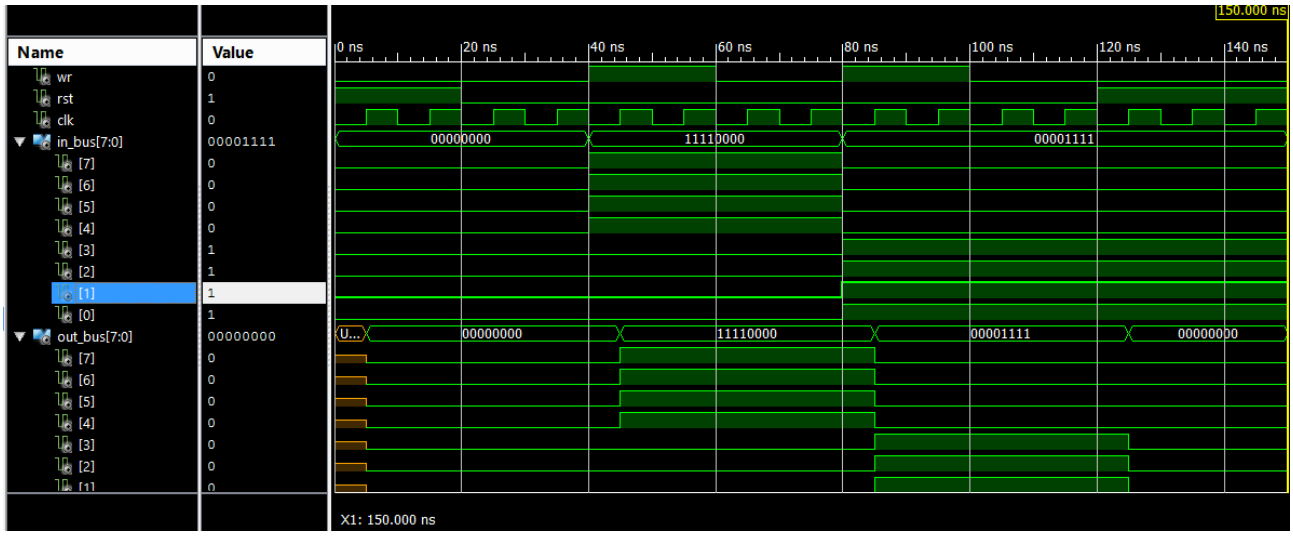

Рис.3.5. Симуляція регістра в ISim.

5. Визначити набір необхідних операції для обчислення індивідуального виразу.

Список набір операцій:

1. nop – передача даних із входу B на вихід АЛП.
2. “-“
3. “+”
4. “or”

6. Створити новий .vhd файл, та реалізувати на ньому АЛП ALU.

```
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) or ('0' & B);
  when "10" =>
          ALU_Result <= ('0' & A) + ('0' & B);
```

```
 when "11" =>
  ALU_Result <= ('0' & A) - ('0' & B);
 when others => ALU_Result <= ('0' & B);
 end case;
end process;
ALU_Out <= ALU_Result(7 downto 0);
Carryout <= ALU_Result(8);
end Behavioral;
```

7. Перевірити роботу АЛП за допомогою симулятора ISim.

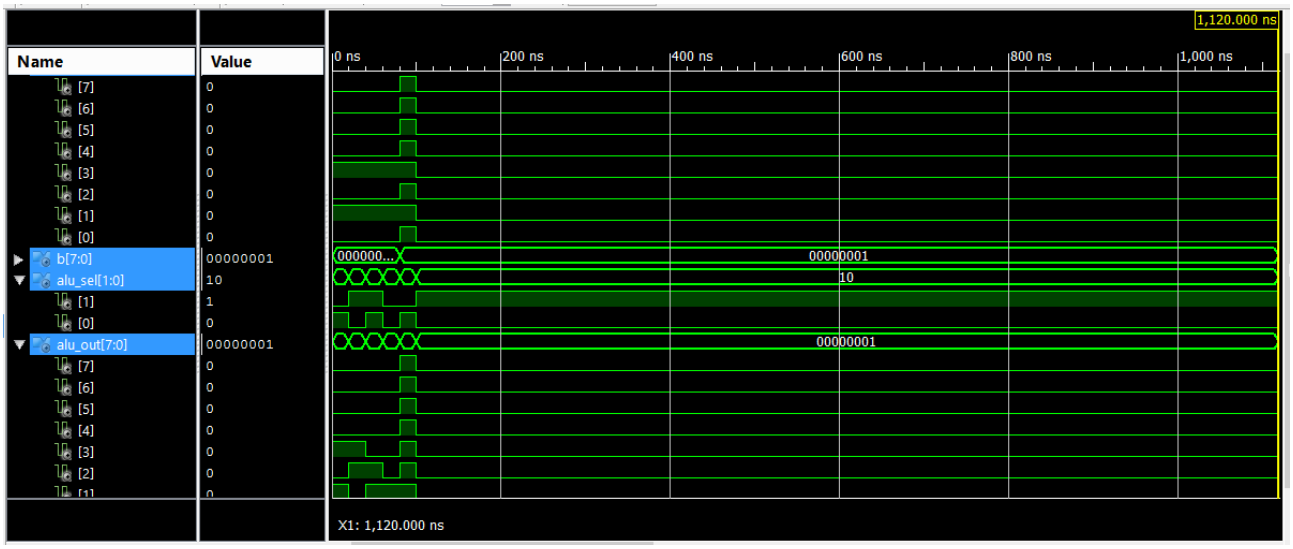

Рис.3.7. Симуляція АЛП в ISim.

8. Визначити множину станів та умови переходів пристрою керування необхідних для обчислення виразу.

Опис кожного зі станів:

- RST – скидання схеми до початкового стану.
- IDLE – стан очікування. Чекає на вхідні сигнали ENTER_OP1, ENTER_OP2 або CALCULATE.
- LOAD_OP1 – запис першого операнда OP1 в ОЗП.
- LOAD_OP2 – запис другого операнда OP2 в ОЗП.
- RUN_CALC0: ACC = RAM(0x00);
- RUN_CALC1: ACC = (ACC - 4);
- RUN_CALC2: ACC = (ACC + OP2);

- RUN_CALC3: ACC = (ACC + 15);
- RUN_CALC4: ACC = (ACC or 2);
- FINISH – індикація кінцевого результату.

9. Створити новий .vhd файл, та реалізувати на ньому блок керування CU.

```
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);
			CONST1_BUS : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
			CONST2_BUS : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
			CONST3_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(2 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
        CONST1_BUS <= "00000100";
        CONST2_BUS <= "00001111";
        CONST3_BUS <= "00000010";

        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  <= "000";
                OP_CODE_BUS <= "00";
                RAM_ADDR_BUS    <= "00";
                RAM_WR              <= '0';
                ACC_RST             <= '1';
                ACC_WR              <= '0';
            when IDLE           =>
```

```
            MUX_SEL_BUS <= "000";
            OP_CODE_BUS <= "00";
            RAM_ADDR_BUS        <= "00";
            RAM_WR                      <= '0';
            ACC_RST                     <= '0';
            ACC_WR                      <= '0';
        when LOAD_OP1       =>
            MUX_SEL_BUS <= "000";
            OP_CODE_BUS <= "00";
            RAM_ADDR_BUS        <= "00";
            RAM_WR                      <= '1';
            ACC_RST                     <= '0';
            ACC_WR                      <= '1';
        when LOAD_OP2       =>
            MUX_SEL_BUS <= "000";
            OP_CODE_BUS <= "00";
            RAM_ADDR_BUS        <= "01";
            RAM_WR                      <= '1';
            ACC_RST                     <= '0';
            ACC_WR                      <= '1';
        when RUN_CALC0 =>
            MUX_SEL_BUS <= "001";
            OP_CODE_BUS <= "00";
            RAM_ADDR_BUS        <= "00";
            RAM_WR                      <= '0';
            ACC_RST                     <= '0';
            ACC_WR                      <= '1';
        when RUN_CALC1 =>
            MUX_SEL_BUS <= "010";
            OP_CODE_BUS <= "11";
            RAM_ADDR_BUS        <= "00";
            RAM_WR                      <= '0';
            ACC_RST                     <= '0';
            ACC_WR                      <= '1';
```

```
when RUN_CALC2 =>
        MUX_SEL_BUS <= "001";
        OP_CODE_BUS <= "10";
        RAM_ADDR_BUS        <= "01";
        RAM_WR                      <= '0';
        ACC_RST                     <= '0';
        ACC_WR                      <= '1';
when RUN_CALC3 =>
        MUX_SEL_BUS <= "011";
        OP_CODE_BUS <= "10";
        RAM_ADDR_BUS        <= "00";
        RAM_WR                      <= '0';
        ACC_RST                     <= '0';
        ACC_WR                      <= '1';
when RUN_CALC4 =>
        MUX_SEL_BUS <= "100";
        OP_CODE_BUS <= "01";
        RAM_ADDR_BUS        <= "00";
        RAM_WR                      <= '0';
        ACC_RST                     <= '0';
        ACC_WR                      <= '1';
when FINISH    =>
        MUX_SEL_BUS <= "000";
        OP_CODE_BUS <= "00";
        RAM_ADDR_BUS        <= "00";
        RAM_WR                      <= '0';
        ACC_RST                     <= '0';
        ACC_WR                      <= '0';
when others             =>
        MUX_SEL_BUS <= "000";
        OP_CODE_BUS <= "00";
        RAM_ADDR_BUS        <= "00";
        RAM_WR                      <= '0';
        ACC_RST                     <= '0';
```

```
                ACC_WR                  <= '0';
            end case;
    end process;
end CU_arch;
```

10. Перевірити роботу блока керування за допомогою симулятора ISim.

Рис.3.9. Симуляція блоку керування в ISim.

11. Створити новий .vhd файл, та реалізувати на ньому ОЗП RAM.

```
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;
```

12. Перевірити роботу ОЗП RAM за допомогою симулятора ISim.

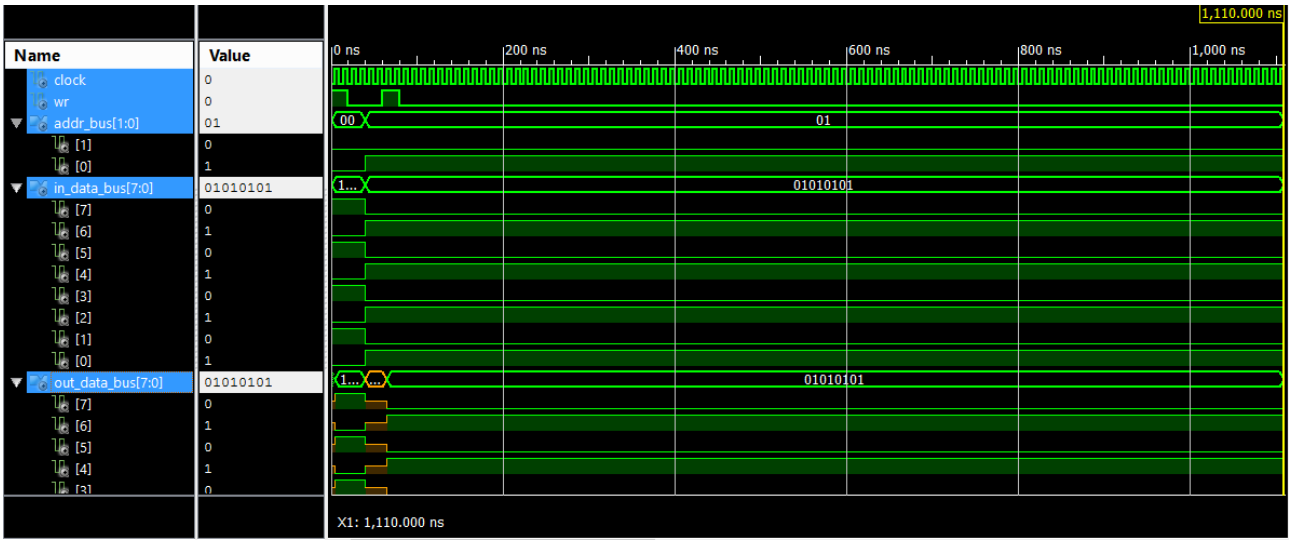

Рис.3.11. Симуляція ОЗП в Isim.

13. Створити новий .vhd файл, та реалізувати на ньому блок індикації 7-SEG DECODER.

```
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;
```

14. Перевірити роботу блока індикації за допомогою симулятора ISim.

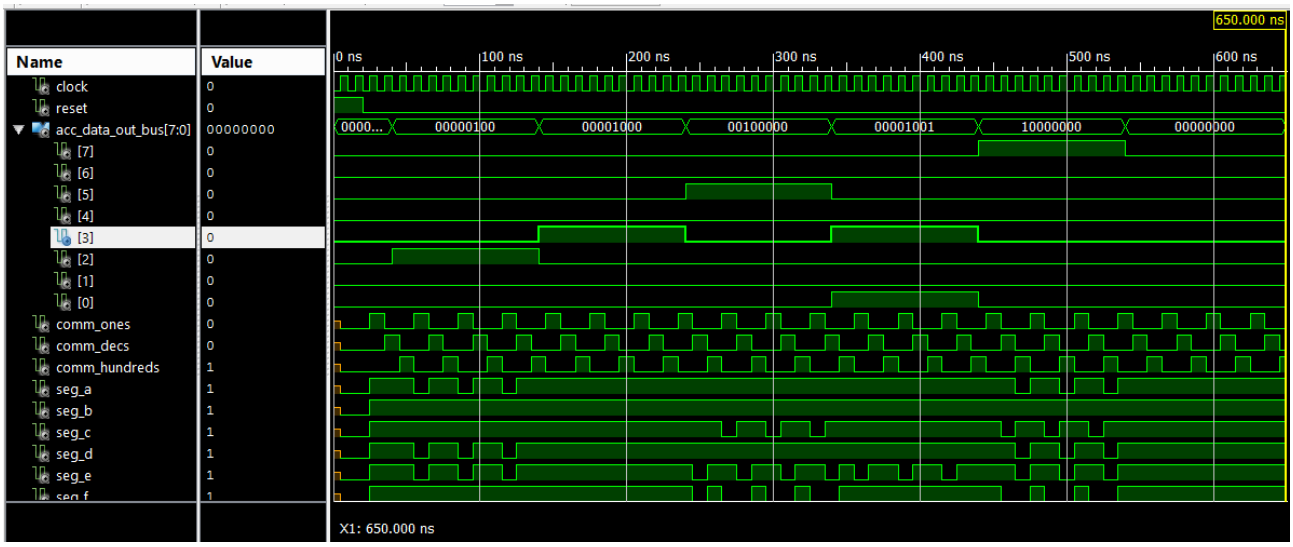

Рис.3.12. Симуляція блоку індикації в Isim.

15-16. Згенерувати символи імплементованих компонентів. Створити файл верхнього рівня та виконати інтеграцію компонентів системи між собою та зі стендом.

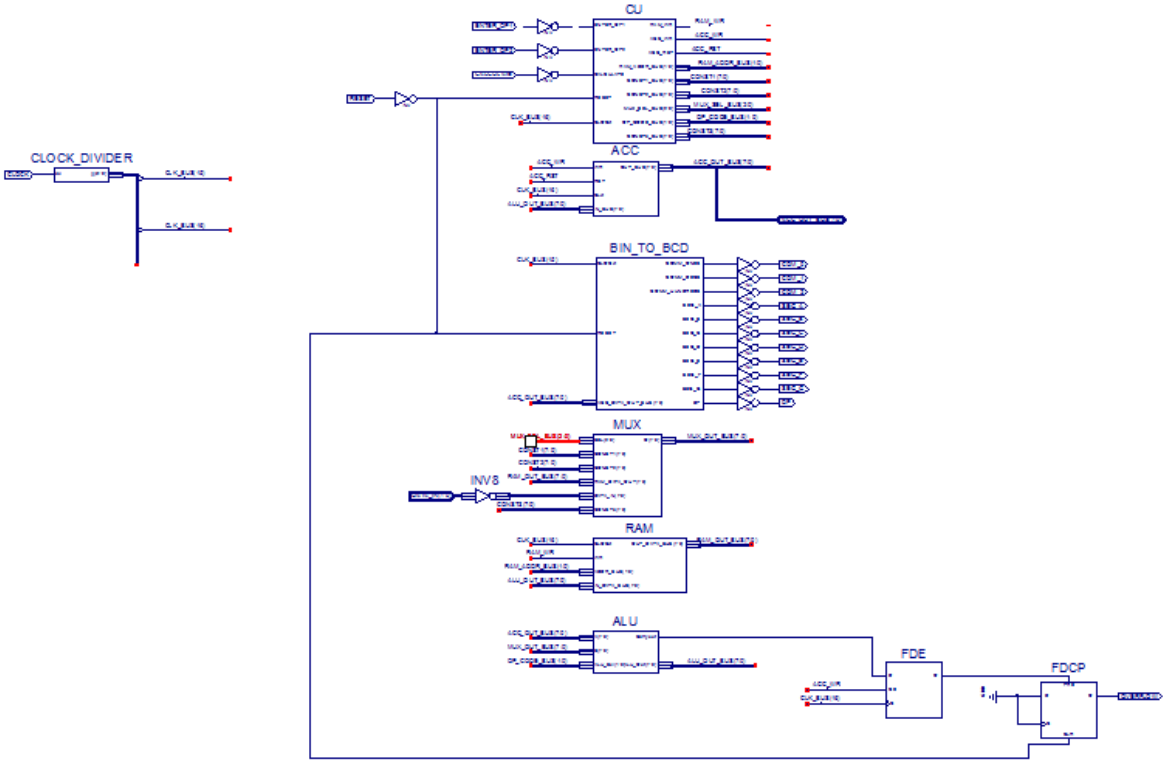

Рис.3.13. Зінтегровані між собою компоненти.

17. Перевірити роботу схеми в симуляторі ISim.

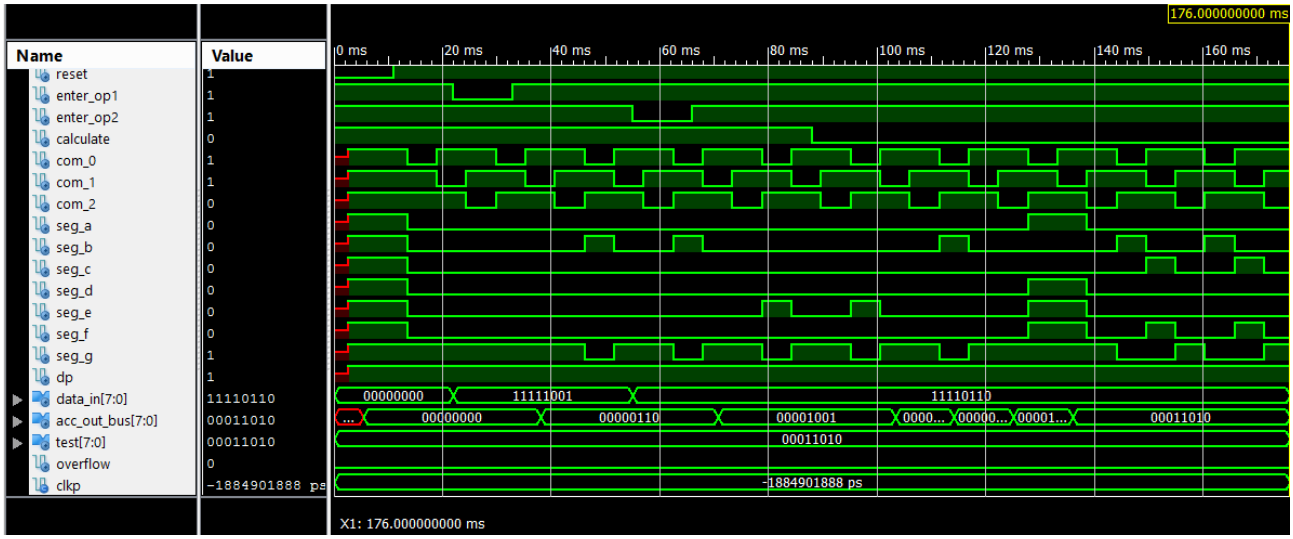

Рис.3.14. Симуляція виконання обчислень.

Висновок:

В ході виконання цієї лабораторної роботи я реалізував на базі стенда Elbert V2 – Spartan3A FPGA цифровий автомат згідно заданих вимог.