МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Звіт

З лабораторної роботи №3

З дисципліни: Моделювання комп'ютерних систем

Виконав:

Студент з КІ-201

Бенітез Гектор

Перевірив:

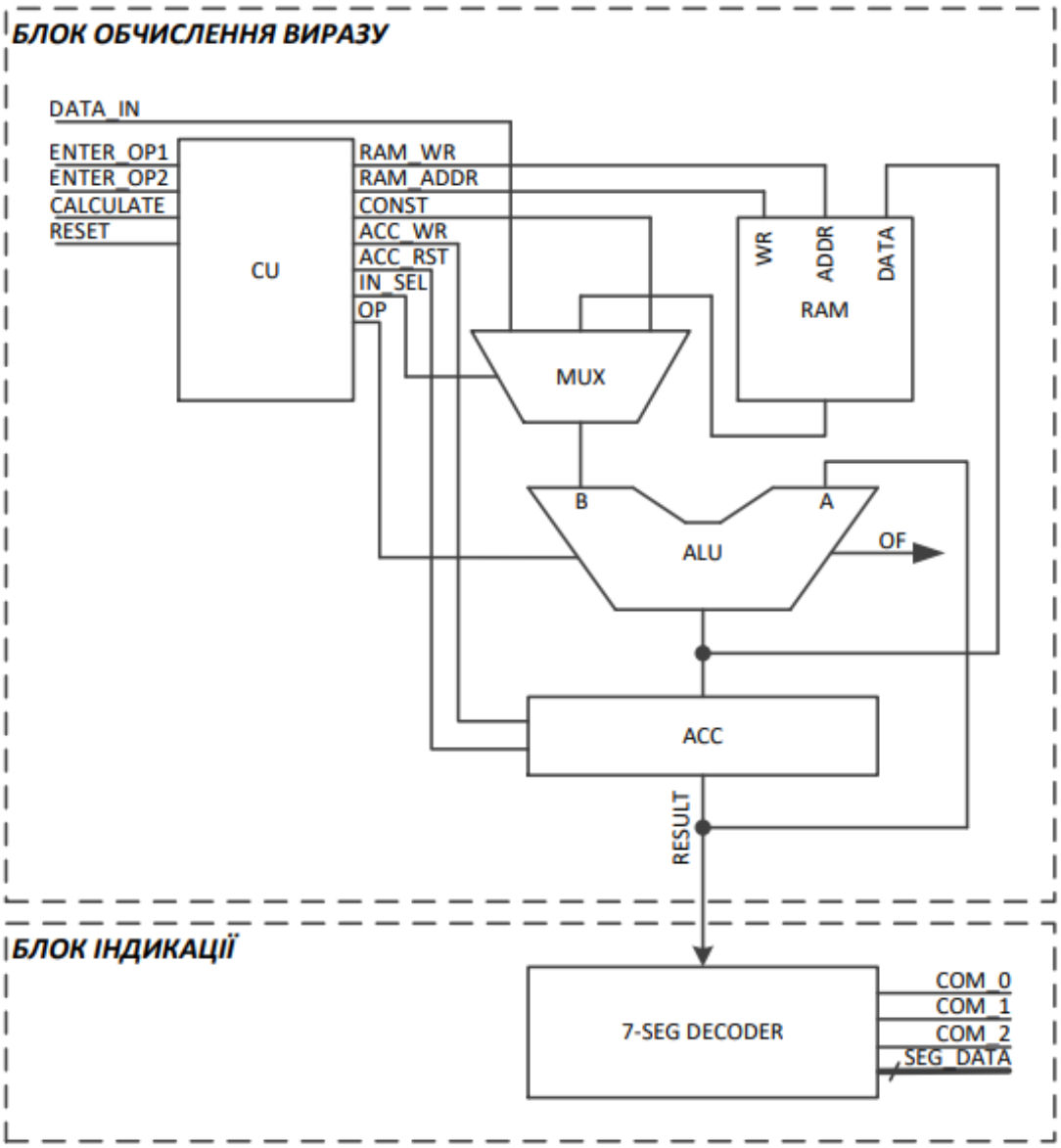
Козак Н.Б.

Львів – 2022

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

Мета роботи: На базі стенда Elbert V2- Spartan 3A FPGA реалізувати цифровий автомат для обчислення значення виразу дотримуючись наступних вимог:

1. Функціонал пристрою повинен бути реалізований згідно отриманого варіанту завдання Дивись розділ ЗАВДАННЯ
2. Пристрій повинен бути ітераційним АЛП повинен виконувати за один такт одну операцію та реалізованим згідно наступної структурної схеми



Малюнок 1 - Структурна схема автомата.

3. Кожен блок структурної схеми повинен бути реалізований на мові 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 повинна бути виконана за допомогою
6. Кожен структурний блок і схема вцілому повинні бути промодельовані за допомогою симулятора
7. Формування вхідних даних на шині повинно бути реалізовано за допомогою перемикачів елемент на стенді Див Додаток інформація про стенд наймолодший розряд значення операнда найстарший розряд значення операнда
8. Керування пристроєм повинно бути реалізовано за допомогою кнопок елементи на стенді Див Додаток інформація про стенд запис першого операнда в пам ять даних автомата запис другого операнда в пам ять даних автомата запуск процесу обчислення скидання автомата у початковий стан
9. Індикація значень операндів при вводі та вивід результату обчислень повинні бути реалізовані за допомогою семи сегментних індикаторів Індикація переповнення в АЛП за допомогою на стенді Див Додаток інформація про стенд
10. Підготувати і захистити звіт

ЗАВДАННЯ

8	((OP1 * OP2) >> 1) + OP1	

варіант – 8.

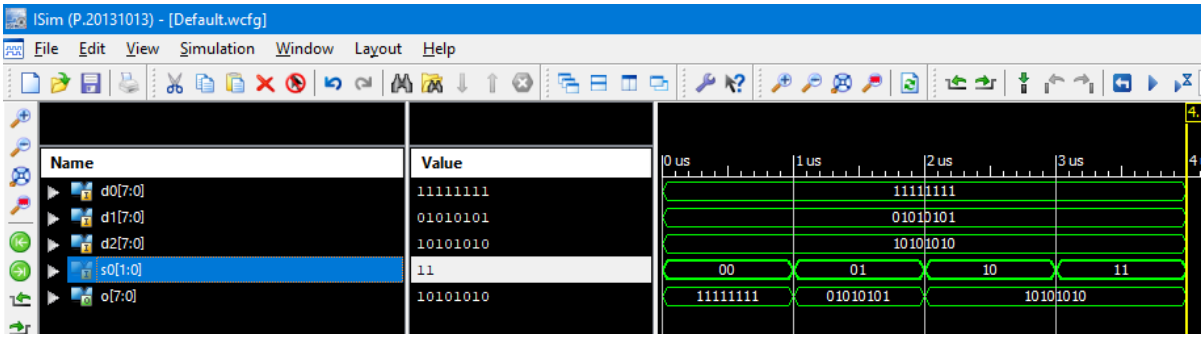
Мій вираз : ((OP1*OP2)>>1)+OP1.

Виконання роботи:

1. Створив VHDL мультиплексор

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

-- Uncomment the following library declaration i
-- arithmetic functions with Signed or Unsigned
--use IEEE.NUMERIC_STD.ALL;

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

entity MUX_3_1 is
port(
   D0,D1,D2 : in std_logic_vector(7 downto 0);
   S0 : in std_logic_vector(1 downto 0);
   O : out std_logic_vector(7 downto 0)
);

end MUX_3_1;

architecture Behavioral of MUX_3_1 is

begin
process (D0,D1,D2,S0)
   begin
   if (S0 = "00") then
   O <= D0;
   elsif(S0 = "01") then
   O <= D1;
   elsif(S0 = "10") then
   O <= D2;
   end if;
end process;
end Behavioral;
```

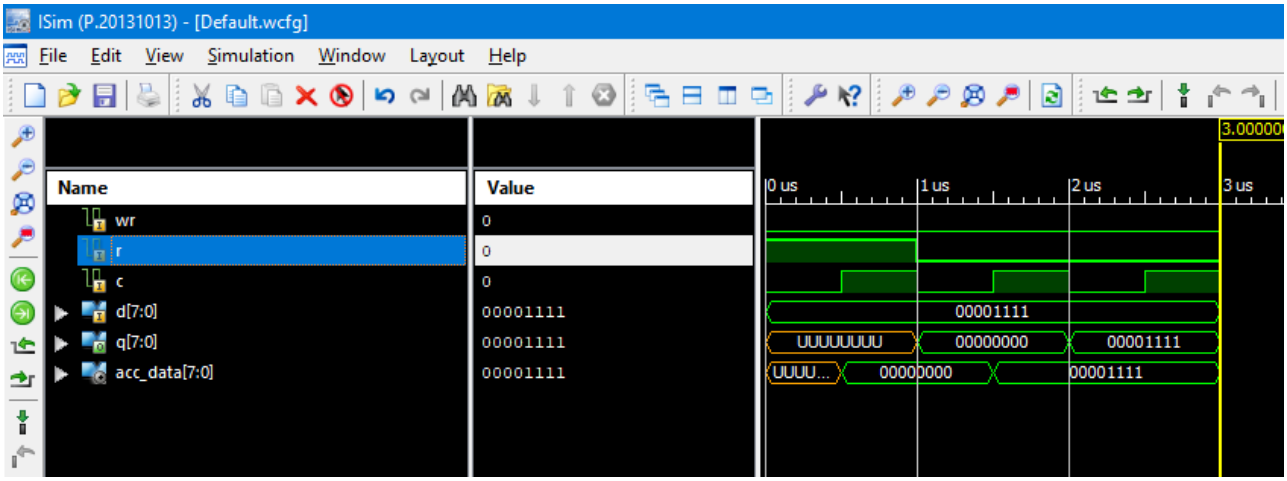
2. Промоделював роботу з усіма можливими наборами сигналів



3. Створив VHDL файл який реалізує регістр ACC

```
--use UNISIM.VComponents.all;

entity ACC is
port(
       WR : in STD_LOGIC;
       R : in STD_LOGIC;
       C : in STD_LOGIC;
       D : in STD_LOGIC_VECTOR(7 downto 0);
       Q : out STD_LOGIC_VECTOR(7 downto 0)
        );

end ACC;

architecture Behavioral of ACC is
signal ACC_DATA : STD_LOGIC_VECTOR(7 downto 0);
begin
   process(C,R,WR,D)
   begin
         if (rising_edge(C)) then
         if(R = '1') then
            ACC_DATA <= "00000000";
         elsif (WR = '0') then
            ACC_DATA <= D;
         end if;
      end if;
      Q <= ACC_DATA;
   end process;
end Behavioral;
```

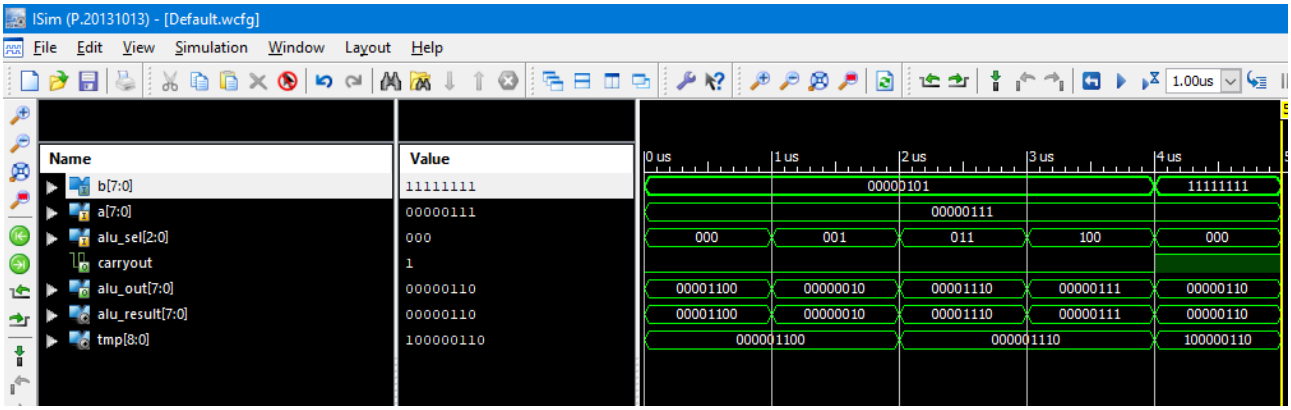
4. Промоделював роботу схеми



5. Створив VHDL файл який реалізує ALU

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_arith.all;
use IEEE.NUMERIC_STD.ALL;
entity ALU is
port(
		B : in STD_LOGIC_vector (7 downto 0);
		A : in STD_LOGIC_vector (7 downto 0);
		ALU_SEL : in STD_LOGIC_vector (2 downto 0);
		CARRYOUT : out STD_LOGIC := '0';
		ALU_OUT : out STD_LOGIC_vector (7 downto 0)
		);
end ALU;

architecture Behavioral of ALU is
signal ALU_Result : std_logic_vector (7 downto 0);
signal tmp: std_logic_vector (8 downto 0);
begin

	process(A, B, ALU_SEL)
	variable ALU_Mul_Result : std_logic_vector (15 downto 0);
	begin
	case ALU_SEL is
		when "000" => --add
		ALU_Result <= STD_LOGIC_VECTOR(A+B);
		tmp <= ('0' & A) + ('0' & B);
		when "001" => -- Subs
		ALU_Result <= A - B ;

		when "010" => --B*A
		ALU_Mul_Result := A * B;
		ALU_Result <= ALU_Mul_Result(7 downto 0);
		if (ALU_Mul_Result(15 downto 8) > 0) then
		tmp(8)  <= '1';
		else
		tmp(8) <= '0';
		end if;

		when "011" => --A<<1
		ALU_Result(7 downto 1)<=A(6 downto 0);
		ALU_Result(0)<='0';
		tmp <= A & '0';

		when "100" => --B>>1
		ALU_Result(6 downto 0)<=B(7 downto 1);
		ALU_Result(7)<='0';
		tmp <= B & '0';
		when "101" => --A>>1
		ALU_Result(6 downto 0)<=A(7 downto 1);
		ALU_Result(7)<='0';
		tmp <= A & '0';

		when "110" => -- A
		ALU_Result <=A;
		when "111" => -- B
		ALU_Result <=B;
		when others => ALU_Result <= A+B;
		end case;
	end process;
	ALU_Out <= ALU_Result; -- ALU out

	Carryout <= tmp(8); -- Carryout flag
end Behavioral;
```

6. Промоделював роботу схеми



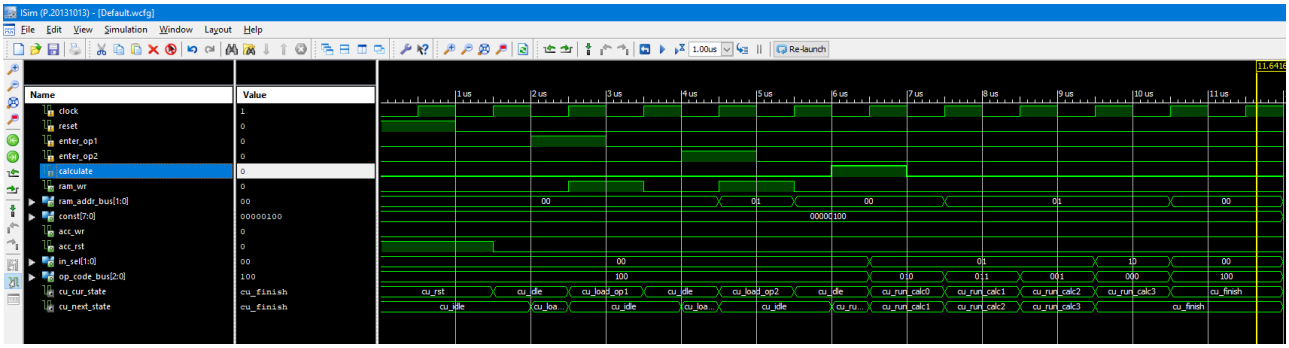
7. Створив VHDL файл який реалізує CU

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

entity CU is
port(
                CLOCK           : IN STD_LOGIC;
                RESET           : IN STD_LOGIC;
                ENTER_OP1       : IN STD_LOGIC;
                ENTER_OP2       : IN STD_LOGIC;
                CALCULATE       : IN STD_LOGIC;
                RAM_WR                  : OUT STD_LOGIC;
                RAM_ADDR_BUS                    : 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_CODE_BUS             : OUT STD_LOGIC_VECTOR(2 downto 0)

);
end CU;

architecture Behavioral of CU is
type   cu_state_type is (cu_rst, cu_idle, cu_load_op1, cu_load_op2, cu_run_calc0, cu_run_calc1, cu_run_calc2, cu_run_calc3, cu_finish);
signal cu_cur_state  : cu_state_type;
signal cu_next_state : cu_state_type;
begin

CU_SYNC_PROC: process (CLOCK)
  Begin

CU_SYNC_PROC: process (CLOCK)
  begin
    if (rising_edge(CLOCK)) then
      if (RESET = '1') then
        cu_cur_state <= cu_rst;
      else
        cu_cur_state <= cu_next_state;
      end if;
    end if;
  end process;
```

```
CUNEXT_STATE_DECODE: process (cu_cur_state, ENTER_OP1, ENTER_OP2, CALCULATE)
  begin
    --declare default state for next_state to avoid latches
    cu_next_state <= cu_cur_state;  --default is to stay in current state
    --insert statements to decode next_state
    --below is a simple example
                case(cu_cur_state) is
                        when cu_rst                     =>
                                cu_next_state <= cu_idle;
                        when cu_idle                    =>
                                if (ENTER_OP1 = '1') then
                                        cu_next_state <= cu_load_op1;
                                elsif (ENTER_OP2 = '1') then
                                        cu_next_state <= cu_load_op2;
                                elsif (CALCULATE = '1') then
                                        cu_next_state <= cu_run_calc0;
                                else
                                        cu_next_state <= cu_idle;
                                end if;
                        when cu_load_op1        =>
                                cu_next_state <= cu_idle;
                        when cu_load_op2        =>
                                cu_next_state <= cu_idle;
                        when cu_run_calc0 =>
                                cu_next_state <= cu_run_calc1;
                        when cu_run_calc1 =>
                                cu_next_state <= cu_run_calc2;
                        when cu_run_calc2 =>
                                cu_next_state <= cu_run_calc3;
                                when cu_run_calc3 =>
                                cu_next_state <= cu_finish;
                        when cu_finish          =>
                                cu_next_state <= cu_finish;
                        when others                     =>
                                cu_next_state <= cu_idle;
                end case;
  end process;


  CU_OUTPUT_DECODE: process (cu_cur_state)
  begin
                case(cu_cur_state) is
                        when cu_rst                     =>
                                IN_SEL                  <= "00";
                                OP_CODE_BUS     <= "110";
                                RAM_ADDR_BUS    <= "00";
                                RAM_WR                  <= '0';
                                ACC_RST                 <= '1';
                                ACC_WR                  <= '0';
                        when cu_idle                    =>
                                IN_SEL                  <= "00";
                                OP_CODE_BUS     <= "111";
                                RAM_ADDR_BUS    <= "00";
                                RAM_WR                  <= '0';
                                ACC_RST                 <= '0';
                                ACC_WR                  <= '1';
                        when cu_load_op1        =>
                                IN_SEL                  <= "00";
```

```
                        OP_CODE_BUS     <= "111";
                        RAM_ADDR_BUS    <= "00";
                        RAM_WR                  <= '1';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when cu_load_op2        =>
                        IN_SEL                  <= "00";
                        OP_CODE_BUS     <= "111";
                        RAM_ADDR_BUS    <= "01";
                        RAM_WR                  <= '1';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when cu_run_calc0 =>
                        IN_SEL                  <= "01"; --load OP1
                        OP_CODE_BUS     <= "111"; --B
                        RAM_ADDR_BUS    <= "00";
                        RAM_WR                  <= '0';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when cu_run_calc1 =>
                        IN_SEL                  <= "01";
                        OP_CODE_BUS     <= "010";  --A*B
                        RAM_ADDR_BUS    <= "01";
                        RAM_WR                  <= '0';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when cu_run_calc2 =>
                        IN_SEL                  <= "01";  --
                        OP_CODE_BUS     <= "101"; --A>>1
                        RAM_ADDR_BUS    <= "01";
                        RAM_WR                  <= '0';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when cu_run_calc3 =>
                        IN_SEL                  <= "01";  --
                        OP_CODE_BUS     <= "000"; --A+B
                        RAM_ADDR_BUS    <= "00";
                        RAM_WR                  <= '0';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when cu_finish          =>
                        IN_SEL                  <= "00";
                        OP_CODE_BUS     <= "110";
                        RAM_ADDR_BUS    <= "00";
                        RAM_WR                  <= '0';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
                when others             =>
                        IN_SEL                  <= "00";
                        OP_CODE_BUS     <= "000";
                        RAM_ADDR_BUS    <= "00";
                        RAM_WR                  <= '0';
                        ACC_RST                 <= '0';
                        ACC_WR                  <= '0';
        end case;
  end process;
        CONST <= "00000100";
end Behavioral;
```

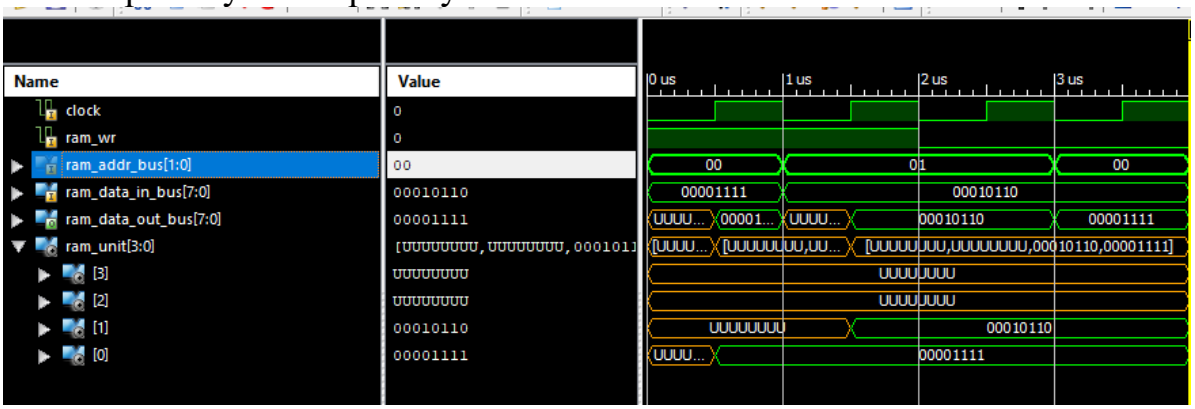
8. Просимулював роботу



9. Створив VHDL файл який реалізує RAM

```
19  ----------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_UNSIGNED.ALL;
23
24
25  entity RAM is
26  port(
27     CLOCK : in std_logic;
28     RAM_WR : in std_logic;
29     RAM_ADDR_BUS : in std_logic_vector(1 downto 0);
30     RAM_DATA_IN_BUS : in std_logic_vector(7 downto 0);
31     RAM_DATA_OUT_BUS : out std_logic_vector(7 downto 0)
32  );
33  end RAM;
34
35  architecture Behavioral of RAM is
36  type ram_type is array (3 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
37  signal RAM_UNIT : ram_type;
38  begin
39
40     RAM : process(CLOCK, RAM_ADDR_BUS, RAM_UNIT)
41      begin
42        if (rising_edge(CLOCK)) then
43           if (RAM_WR = '1') then
44              RAM_UNIT(conv_integer(RAM_ADDR_BUS)) <= RAM_DATA_IN_BUS;
45           end if;
46        end if;
47        RAM_DATA_OUT_BUS <= RAM_UNIT(conv_integer(RAM_ADDR_BUS));
48     end process RAM;
49
50
51  end Behavioral;
52
```

10. Просимулював роботу



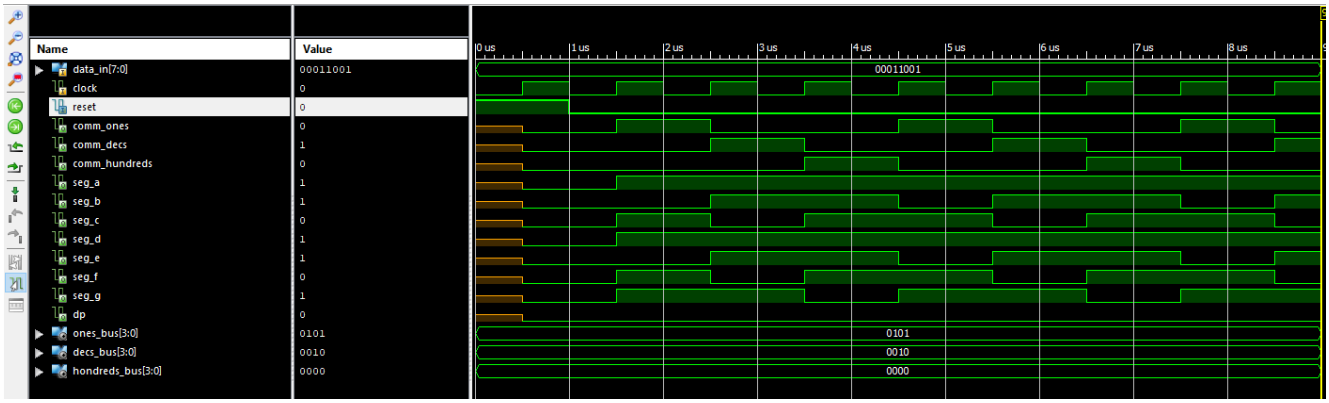
11. Створив VHDL файл який реалізує SegmentDecoder

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

entity SegmentDecoder is
port (
 DATA_IN : in STD_LOGIC_VECTOR(7 downto 0)  ;
 CLOCK : in std_logic;
                RESET                   : IN STD_LOGIC;
        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 SegmentDecoder;

architecture Behavioral of SegmentDecoder 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 (DATA_IN)
        variable hex_src : STD_LOGIC_VECTOR(7 downto 0) ;
    variable bcd     : STD_LOGIC_VECTOR(11 downto 0) ;
  begin
    bcd           := (others => '0') ;
        hex_src        := DATA_IN;
    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;
```

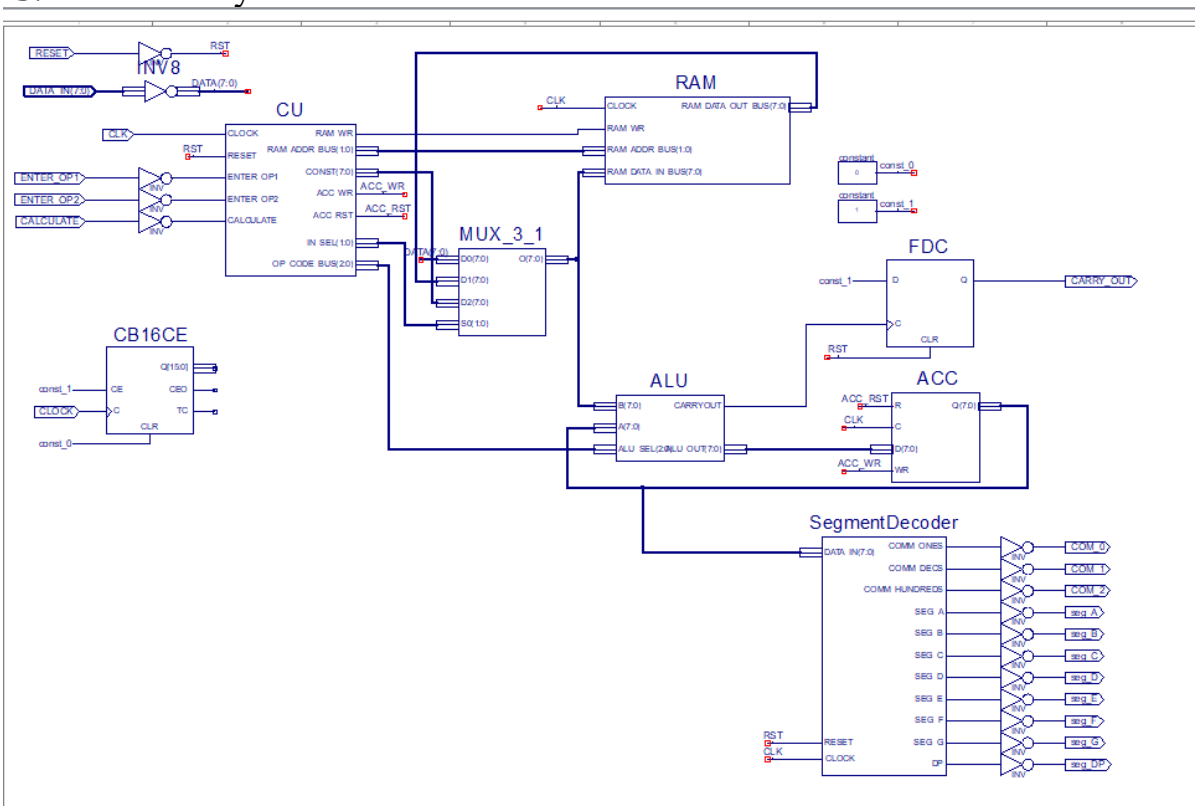
```
INDICATE : process(CLOCK)
        type DIGIT_TYPE is (ONES, DECS, HUNDREDS);

        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 := "1111110";
                                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;
```

12.Просимулював роботу



13. Склав схему



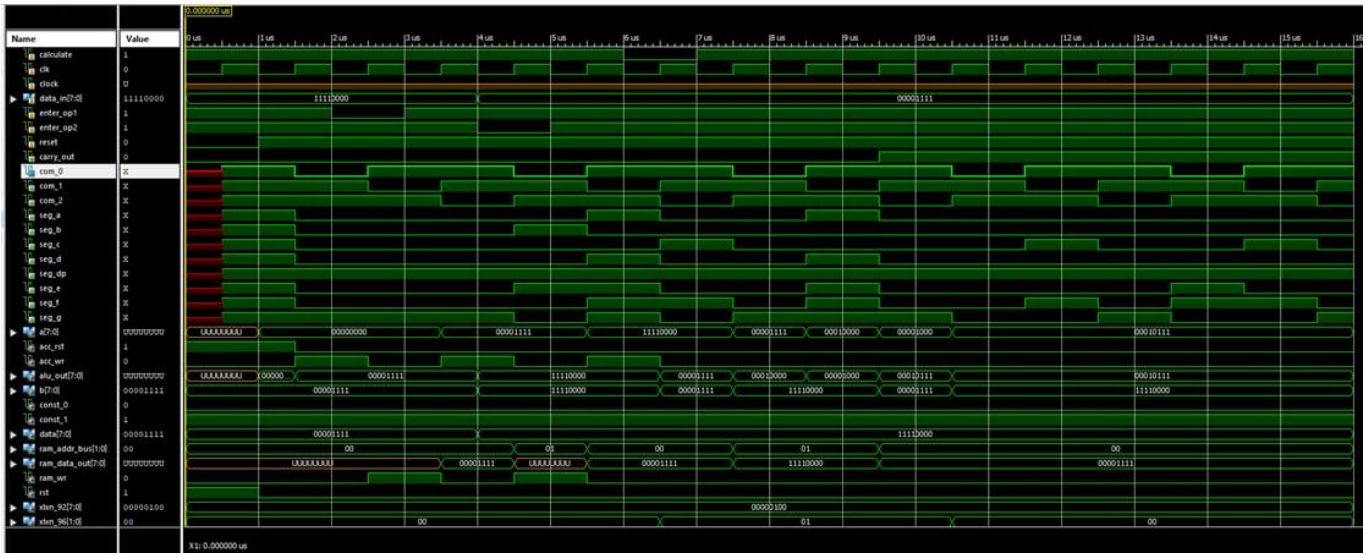
14. Просимулював роботу
((OP1* OP2)>>1)+OP1

При OP1 = 0000 1111, OP2 = 1111 0000

0000 1111 * 1111 0000 = 1110 0001 0000

0001 0000 >> 1 = 0000 1000

0000 1000 + 0000 1111 = 0001 0111



Висновок: На даній лабораторній роботі я на базі стенда Elbert V2-Spartan 3A FPGA реалізував цифровий автомат для обчислення значення виразу.