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 - Spartan 3A FPGA*»

Виконав: ст. гр. КІ-201

Олеш С.Б.

Прийняв:

Козак Н.Б.

Львів – 2023

Мета: 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 (ALU) повинен виконувати за один такт одну операцію), та реалізованим згідно наступної структурної схеми.
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 Schematic символ.
5. Інтеграція структурних блоків в єдину систему та зі стендом *Elbert V2- Spartan 3A FPGA* повинна бути виконана за допомогою *ISE WebPACK™ Schematic Capture*.
6. Кожен структурний блок і схема в цілому повинні бути промодельовані за допомогою симулятора *ISim*.
7. Формування вхідних даних на шині DATA_IN повинно бути реалізовано за допомогою DIP перемикачів (елемент P7 на стенді *Elbert V2- Spartan 3A FPGA* Див. Додаток - 2 (інформація про стенд)):
    a. P7[8] - наймолодший розряд значення операнда.
    b. P7[1] - найстарший розряд значення операнда.
8. Керування пристроєм повинно бути реалізовано за допомогою PUSH BUTTON кнопок (елементи SW1, SW2, SW3, SW6 на стенді *Elbert V2- Spartan 3A FPGA* Див. Додаток - 2 (інформація про стенд)):
    a. SW1 - запис першого операнду в пам’ять даних автомата.
    b. SW2 - запис другого операнду в пам’ять даних автомата.
    c. SW3 - запуск процесу обчислення.
    d. SW6 - скидання автомата у початковий стан.
9. Індикація значень операндів при вводі, та вивід результату обчислень повинні бути реалізовані за допомогою семи сегментних індикаторів S1 – S3. Індикація переповнення в АЛП - за допомогою LED D8 на стенді *Elbert V2- Spartan 3A FPGA* Див. Додаток - 2 (інформація про стенд).
10. Підготувати і захистити звіт.

Завдання:

Варіант 3

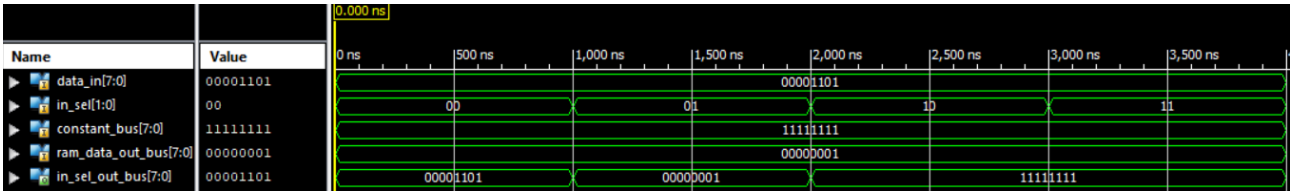
ВАРІАНТ	ВИРАЗ
0	((OP1 – OP2) + 4) << OP2
1	((OP1 + 2) * OP2) << OP1
2	((OP1 or OP2) + OP2) – 3
3	((OP2 and 5) + OP2) – OP1
4	((4 + OP1) xor OP2) – OP1
5	((1 << OP1) + OP2) – OP1
6	((OP1 + OP2) – 2) << OP2
7	((OP1 << 2) – OP2) + 4
8	((OP1 * OP2) >> 1) + OP1
9	((OP2 - 4) + OP1) or 2
10	((OP2 – OP1) << 1) + OP1
11	((OP1 * 2) + OP2) >> 1
12	((OP1 xor OP2) + OP2) – 1

Хід роботи:

1. Створив VHDL мультиплексор.

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

entity my_MuX_intf is
port(
      DATA_IN          :  in  std_logic_vector(7 downto 0);
      IN_SEL           :  in  std_logic_vector(1 downto 0);
      CONSTANT_BUS     :  in  std_logic_vector(7 downto 0);
      RAM_DATA_OUT_BUS :  in  std_logic_vector(7 downto 0);
      IN_SEL_OUT_BUS   :  out std_logic_vector(7 downto 0)
      );
end my_MuX_intf;

architecture my_MuX_arch of my_MuX_intf is

begin
    INSEL_A_MUX : process(DATA_IN, CONSTANT_BUS, RAM_DATA_OUT_BUS, IN_SEL)
       begin
            if(IN_SEL = "00") then
                  IN_SEL_OUT_BUS <= DATA_IN;
            elsif(IN_SEL = "01") then
                  IN_SEL_OUT_BUS <= RAM_DATA_OUT_BUS;
            else
                  IN_SEL_OUT_BUS <= CONSTANT_BUS;
            end if;
       end process INSEL_A_MUX;
end my_MuX_arch;
```

2. Симуляція роботи мультиплексора.

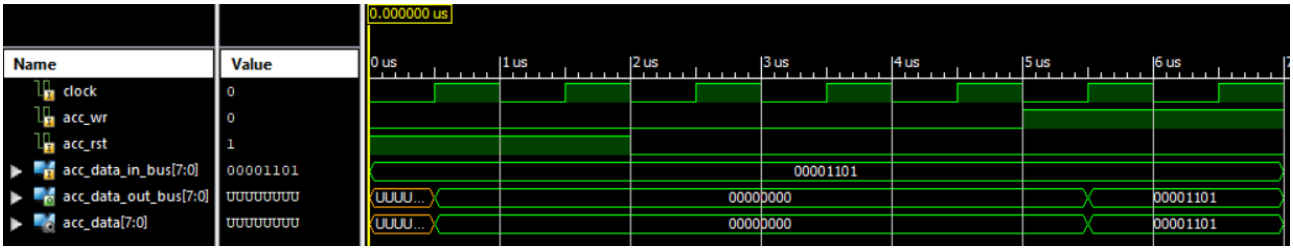


3. Створив VHDL файл який реалізує регістр ACC.

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


entity my_ACC_intf is
port(
      CLOCK            : in  std_logic;
      ACC_WR           : in  std_logic;
      ACC_RST          : in  std_logic;
      ACC_DATA_IN_BUS  : in  std_logic_vector(7 downto 0);
      ACC_DATA_OUT_BUS : out std_logic_vector(7 downto 0)
);
end my_ACC_intf;
```

```
architecture my_ACC_arch of my_ACC_intf is

signal ACC_DATA : std_logic_vector(7 downto 0);

begin

ACC : process(CLOCK, ACC_DATA)
	begin
		if (rising_edge(CLOCK)) then
			if(ACC_RST = '1') then
				ACC_DATA <= "00000000";
			elsif (ACC_WR = '1') then
				ACC_DATA <= ACC_DATA_IN_BUS;
			end if;
		end if;
		ACC_DATA_OUT_BUS <= ACC_DATA;
	end process ACC;

end my_ACC_arch;
```

4. Симуляція роботи регістра.

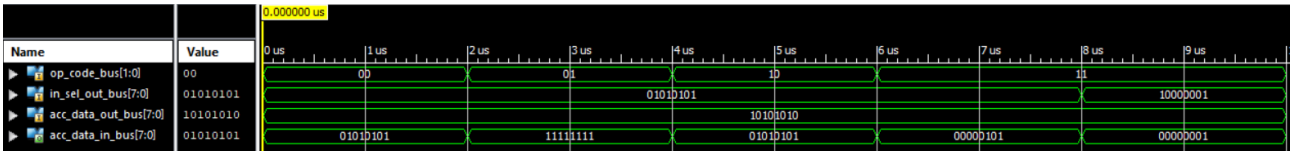


5. Створив VHDL файл який реалізує ALU.

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

entity my_ALU_intf is
port(
	OP_CODE_BUS	: in  std_logic_vector(1 downto 0);
	IN_SEL_OUT_BUS	: in  std_logic_vector(7 downto 0);
	ACC_DATA_OUT_BUS : in  std_logic_vector(7 downto 0);
	ACC_DATA_IN_BUS  : out std_logic_vector(7 downto 0)
);
end my_ALU_intf;

architecture my_ALU_arch of my_ALU_intf is

begin

 ALU : process(OP_CODE_BUS, IN_SEL_OUT_BUS, ACC_DATA_OUT_BUS)
		variable A : unsigned(7 downto 0);
		variable B : unsigned(7 downto 0);
	begin
		A := unsigned(ACC_DATA_OUT_BUS);
		B := unsigned(IN_SEL_OUT_BUS);

		case(OP_CODE_BUS) is
			when "00"    => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(B);
			when "01"    => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A + B);
			when "10"    => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(A - B);
```

```
                    when "11"    => ACC_DATA_IN_BUS <= STD_LOGIC_VECTOR(B and
"00000101");
                    when others => ACC_DATA_IN_BUS <= "00000000";
             end case;

       end process ALU;

end my_ALU_arch;
```

6. Симуляція роботи ALU.



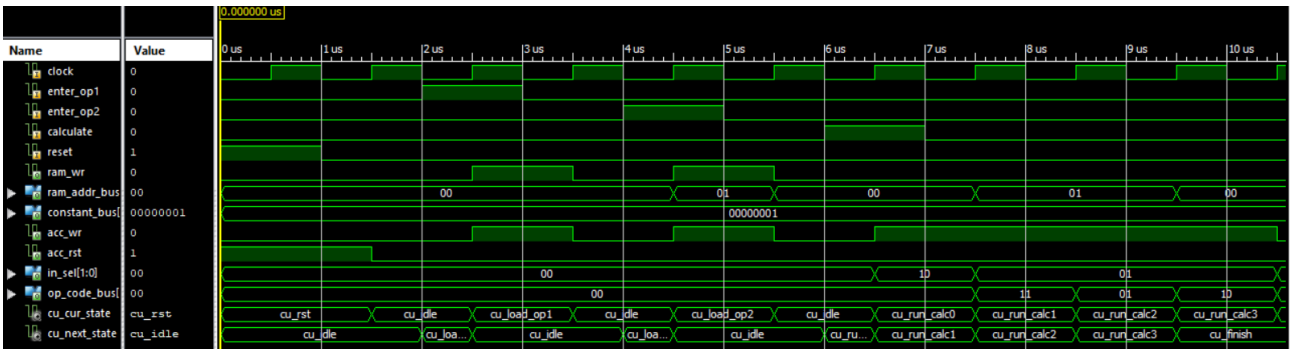
7. Створив VHDL файл який реалізує CU.

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

entity my_CU_intf is
port(
CLOCK        :  in std_logic;
ENTER_OP1    :  in std_logic;
ENTER_OP2    :  in std_logic;
CALCULATE    :  in std_logic;
RESET        :  in std_logic;

RAM_WR       : out std_logic;
RAM_ADDR_BUS : out std_logic_vector(1 downto 0);
CONSTANT_BUS : 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(1 downto 0)
);
end my_CU_intf;

architecture my_CU_arch of my_CU_intf 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
CONSTANT_BUS <= "00000001";

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  <= "00";
                        RAM_ADDR_BUS <= "00";
                        RAM_WR             <= '0';
                        ACC_RST                  <= '1';
                        ACC_WR             <= '0';
                  when cu_idle             =>
                        IN_SEL                     <= "00";
                        OP_CODE_BUS  <= "00";
                        RAM_ADDR_BUS <= "00";
                        RAM_WR             <= '0';
                        ACC_RST                  <= '0';
                        ACC_WR             <= '0';
                  when cu_load_op1   =>
                        IN_SEL                     <= "00";
                        OP_CODE_BUS  <= "00";
                        RAM_ADDR_BUS <= "00";
                        RAM_WR             <= '1';
                        ACC_RST                  <= '0';
                        ACC_WR             <= '1';
                  when cu_load_op2   =>
                        IN_SEL                     <= "00";
                        OP_CODE_BUS  <= "00";
                        RAM_ADDR_BUS <= "01";
```

```
                RAM_WR              <= '1';
                ACC_RST                 <= '0';
                ACC_WR              <= '1';
            when cu_run_calc0 =>
                IN_SEL                  <= "10";
                OP_CODE_BUS  <= "00";
                RAM_ADDR_BUS <= "00";
                RAM_WR              <= '0';
                ACC_RST                 <= '0';
                ACC_WR              <= '1';
            when cu_run_calc1 =>
                IN_SEL                  <= "01";
                OP_CODE_BUS  <= "11";
                RAM_ADDR_BUS <= "01";
                RAM_WR              <= '0';
                ACC_RST                 <= '0';
                ACC_WR              <= '1';
            when cu_run_calc2 =>
                IN_SEL                  <= "01";
                OP_CODE_BUS  <= "01";
                RAM_ADDR_BUS <= "01";
                RAM_WR              <= '0';
                ACC_RST                 <= '0';
                ACC_WR              <= '1';
            when cu_run_calc3 =>
                IN_SEL                  <= "01";
                OP_CODE_BUS  <= "10";
                RAM_ADDR_BUS <= "00";
                RAM_WR              <= '0';
                ACC_RST                 <= '0';
                ACC_WR              <= '1';
            when cu_finish      =>
                IN_SEL                  <= "00";
                OP_CODE_BUS  <= "00";
                RAM_ADDR_BUS <= "00";
                RAM_WR              <= '0';
                ACC_RST                 <= '0';
                ACC_WR              <= '0';
            when others         =>
                IN_SEL                  <= "00";
                OP_CODE_BUS  <= "00";
                RAM_ADDR_BUS <= "00";
                RAM_WR              <= '0';
                ACC_RST                 <= '0';
                ACC_WR              <= '0';
        end case;
    end process;
end my_CU_arch;
```

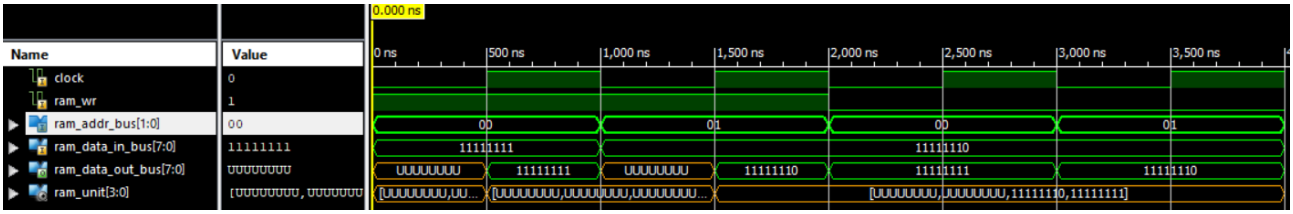
8. Симуляція роботи керуючого автомата.



9. Створив VHDL файл який реалізує RAM.

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

entity my_RAM_intf is
port(
CLOCK            : in  std_logic;
RAM_WR           : in  std_logic;
RAM_ADDR_BUS     : in  STD_LOGIC_VECTOR(1 downto 0);
RAM_DATA_IN_BUS  : in  STD_LOGIC_VECTOR(7 downto 0);
RAM_DATA_OUT_BUS : out STD_LOGIC_VECTOR(7 downto 0)
);
end my_RAM_intf;

architecture my_RAM_arch of my_RAM_intf is

type ram_type is array (3 downto 0) of STD_LOGIC_VECTOR(7 downto 0);
signal RAM_UNIT                 : ram_type;

begin
--when reset will init const
RAM : process(CLOCK, RAM_ADDR_BUS, RAM_UNIT)
       begin
             if (rising_edge(CLOCK)) then
                   if (RAM_WR = '1') then
                         RAM_UNIT(conv_integer(RAM_ADDR_BUS)) <= RAM_DATA_IN_BUS;
                   end if;
             end if;
             RAM_DATA_OUT_BUS <= RAM_UNIT(conv_integer(RAM_ADDR_BUS));
       end process RAM;

end my_RAM_arch;
```

10.Симуляція роботи RAM.



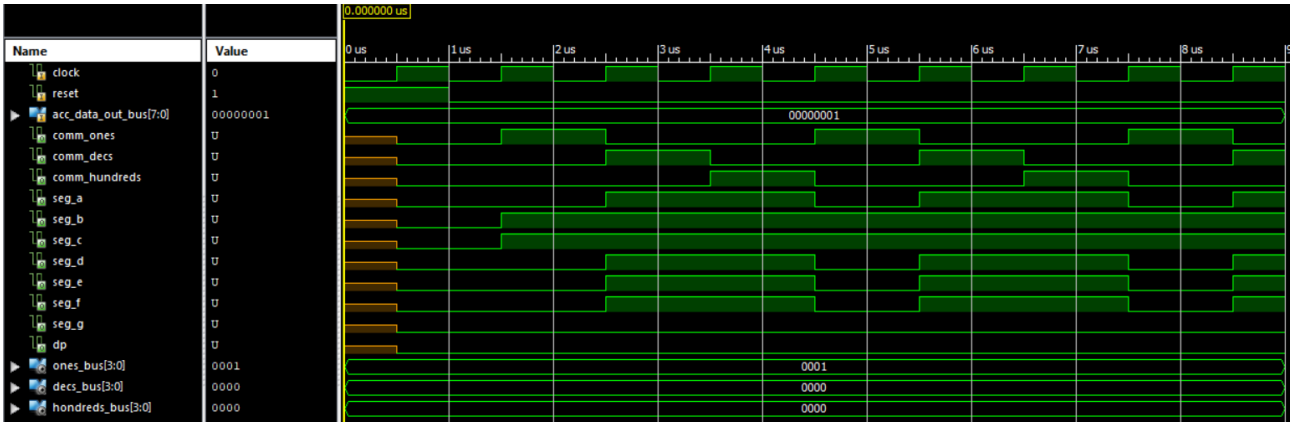
11. Створив VHDL файл який реалізує Sev_Seg_Decoder.

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

entity OUT_PUT_DECODER_intf 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;
```

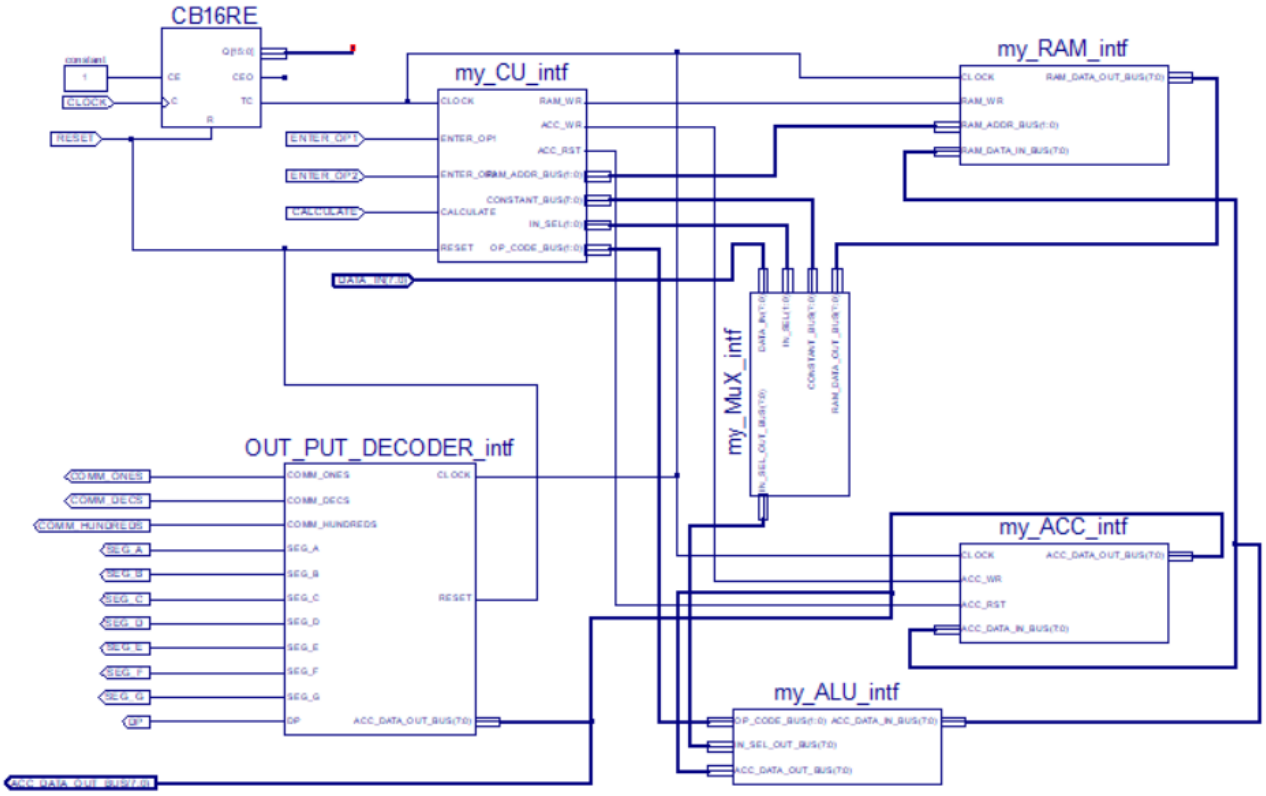
```
SEG_D                       : OUT STD_LOGIC;
SEG_E                       : OUT STD_LOGIC;
SEG_F                       : OUT STD_LOGIC;
SEG_G                       : OUT STD_LOGIC;
DP                      : OUT STD_LOGIC
);
end OUT_PUT_DECODER_intf;

architecture OUT_PUT_DECODER_arch of OUT_PUT_DECODER_intf 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 OUT_PUT_DECODER_arch;
```

12. Симуляція роботи декодера.

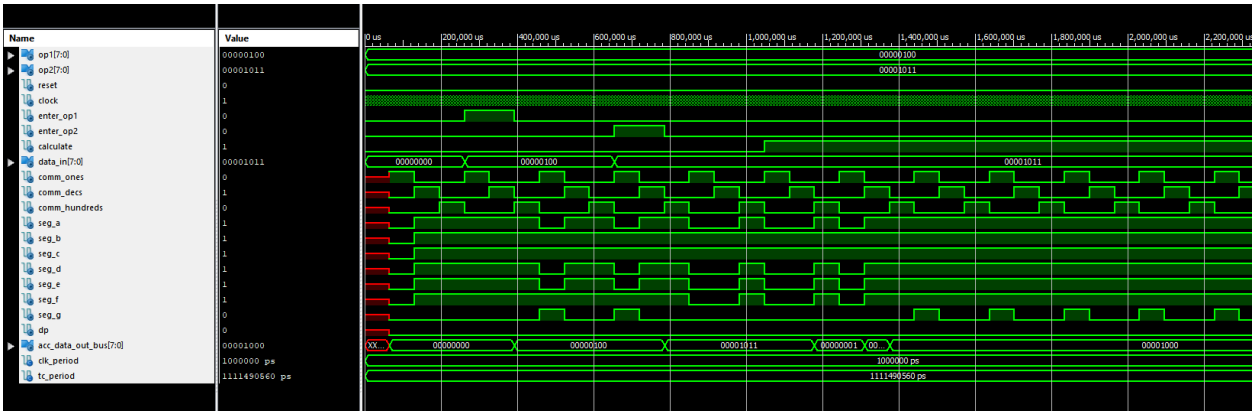


13. Склав схему.



14. Симуляція роботи схеми.

((OP2 and 5) + OP2) – OP1
При OP1 = 0000 0100, OP2 = 0000 1011;
0000 1011 and 0000 0101 = 0000 0001;
0000 0001 + 0000 1011= 0000 1100;
0000 1100 – 0000 0100 = 0000 1000.



Висновок: під час виконання цієї лабораторної роботи я реалізував цифровий автомат для обчислення значення виразу у середовищі Xilinx ISE і стендом Elbert V2 - Spartan 3A FPGA. Я реалізував схему автомату та провів симуляцію його роботи.