Міністерство освіти і науки України
Національний університет „Львівська політехніка”

Звіт з лабораторної роботи № 3
З дисципліни:
“Моделювання комп’ютерних систем”

Виконав:
студент групи КІ-202
Ключко Д.С.

Прийняв:
Козак Н.Б.

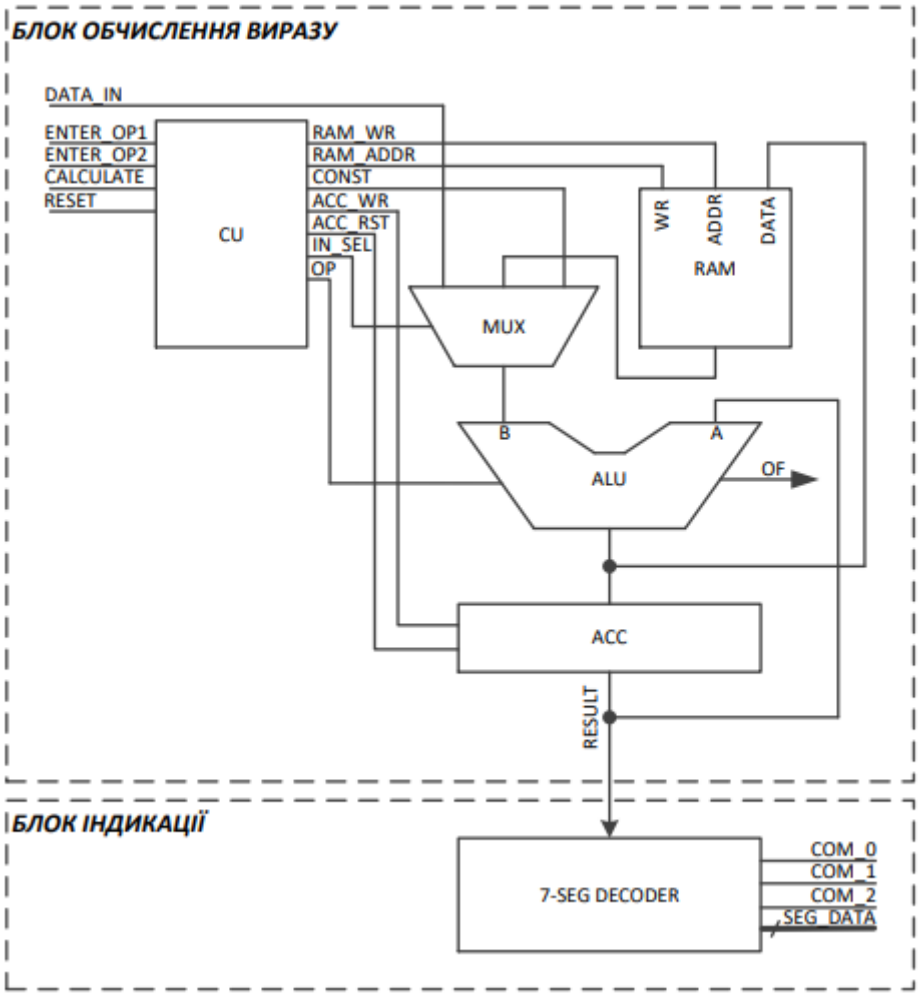
Львів – 2023

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 :



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 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 S1-S3. Індикація переповнення в АЛП за допомогою LED D8 на стенді Elbert V2 – Spartan 3A FPGA.

10. Підготувати і захистити звіт.

Завдання

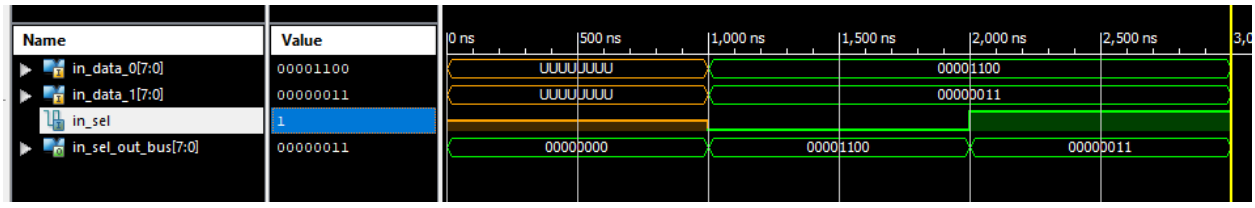
11	((OP1 * 2) + OP2) >> 1

Виконання роботи:

1. Створення файлу VHDL , який реалізовує логіку мультиплексора

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

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

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

entity MUX_module is
Port(
   signal IN_DATA_0        : IN STD_LOGIC_VECTOR(7 downto 0);
   signal IN_DATA_1        : IN STD_LOGIC_VECTOR(7 downto 0);
   signal IN_SEL           : IN STD_LOGIC;

   signal IN_SEL_OUT_BUS   : OUT STD_LOGIC_VECTOR(7 downto 0)
);
end MUX_module;

architecture Behavioral of MUX_module is

begin

   IN_SEL_OUT_BUS <= IN_DATA_0 when IN_SEL = '0' else
                     IN_DATA_1 when IN_SEL = '1' else
                     "00000000";


end Behavioral;
```

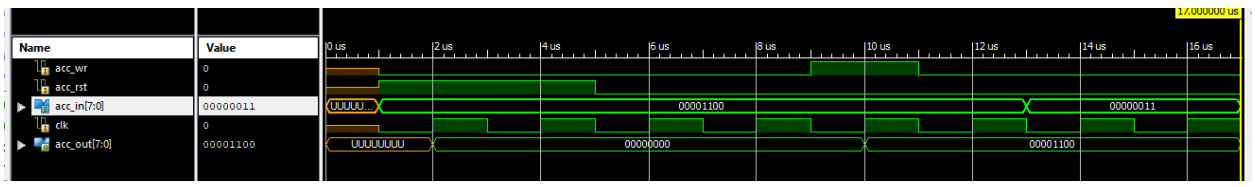
2. Модуляція роботи



3. Створення файлу VHDL , який реалізовує логіку акумулятора

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

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

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

entity ACC_module is
port(
   ACC_WR      : IN STD_LOGIC;
   ACC_RST     : IN STD_LOGIC;
   ACC_IN      : IN STD_LOGIC_VECTOR(7 downto 0);

   CLK         : IN STD_LOGIC;

   ACC_OUT     : OUT STD_LOGIC_VECTOR(7 downto 0)
);
end ACC_module;

architecture Behavioral of ACC_module is

begin

   ALU: process(CLK)
   begin
      if rising_edge(CLK) then
      if(ACC_RST = '1') then
            ACC_OUT <= "00000000";
         elsif (ACC_WR = '1') then
            ACC_OUT <= ACC_IN;
         end if;
      end if;

   end process ALU;

end Behavioral;
```

4. Модуляція роботи

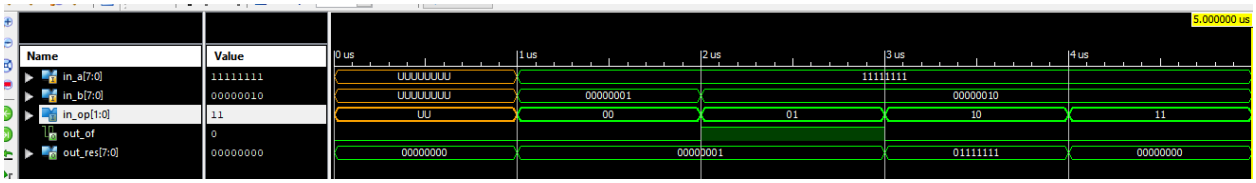


5. Створення файлу VHDL , який реалізовує логіку АЛУ

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity ALU_Module is
port(
   IN_A     : IN  STD_LOGIC_VECTOR(7 downto 0);
   IN_B     : IN  STD_LOGIC_VECTOR(7 downto 0);
   IN_OP    : IN  STD_LOGIC_VECTOR(1 downto 0);

   OUT_OF   : OUT STD_LOGIC;
   OUT_RES  : OUT STD_LOGIC_VECTOR(7 downto 0)
);
end ALU_Module;

architecture Behavioral of ALU_Module is


begin

   ALU : process(IN_OP, IN_A, IN_B)
      variable A : unsigned(8 downto 0);
      variable B : unsigned(8 downto 0);
    begin
      A := unsigned('0'&IN_A);
      B := unsigned('0'&IN_B);

      case(IN_OP) is
         when "00"   =>
                        OUT_RES <= STD_LOGIC_VECTOR(B(7 downto 0));
                        OUT_OF <= '0';
```

```
         when "00"   =>
                        OUT_RES <= STD_LOGIC_VECTOR(B(7 downto 0));
                        OUT_OF <= '0';
         when "01"   =>
                        B := unsigned(unsigned(A) + unsigned(B));
                        OUT_RES <= STD_LOGIC_VECTOR(B(7 downto 0));
                        case STD_LOGIC(B(8)) is
                              when '0' => OUT_OF <= '0';
                              when '1' => OUT_OF <= '1';
                              when others => OUT_OF <= '0';
                        end case;
         when "10"   =>
                        A := A srl 1;
                        OUT_RES <= STD_LOGIC_VECTOR(A(7 downto 0));
                        OUT_OF  <= '0';
         when others =>
                        OUT_RES <= "00000000";
                        OUT_OF  <= '0';
      end case;
    end process ALU;

end Behavioral;
```

6. Модуляція роботи



7. Створення файлу VHDL , який реалізовує логіку блока керування

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity CU_module is
port(
   ENTER_OP1      : IN  STD_LOGIC;
   ENTER_OP2      : IN  STD_LOGIC;
   CALC           : IN  STD_LOGIC;
   RESET          : IN  STD_LOGIC;
   CLK            : IN  STD_LOGIC;

   RAM_WR         : OUT STD_LOGIC;
   RAM_ADDR       : OUT STD_LOGIC_VECTOR(1 downto 0);

   ACC_WR         : OUT STD_LOGIC;
   ACC_RST        : OUT STD_LOGIC;
   IN_SELECT      : OUT STD_LOGIC;
   OP_SELECT      : OUT STD_LOGIC_VECTOR(1 downto 0)

);
end CU_module;

architecture Behavioral of CU_module 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
```

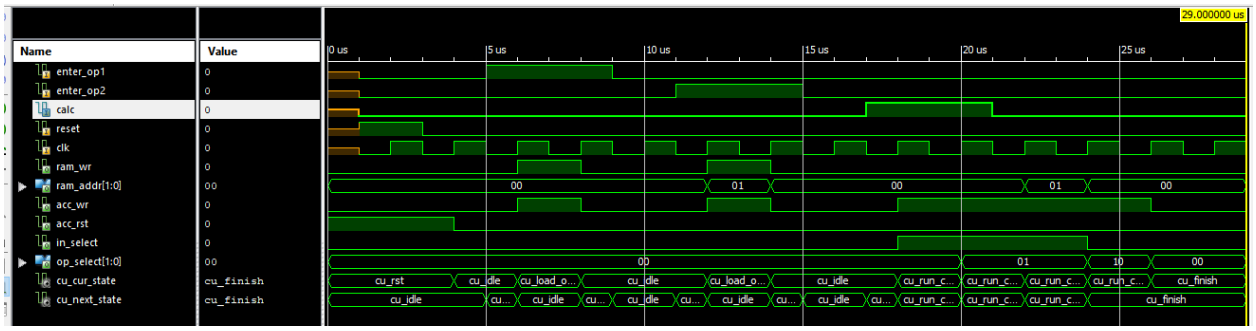
```
CU_SYNC_PROC: process (CLK)
begin
   if (rising_edge(CLK)) then
      if (RESET = '1') then
         cu_cur_state <= cu_rst;
      else
         cu_cur_state <= cu_next_state;
      end if;
   end if;
end process CU_SYNC_PROC;

CUNEXT_STATE_DECODE: process (cu_cur_state, ENTER_OP1, ENTER_OP2, CALC)
```

```
CUNEXT_STATE_DECODE: process (cu_cur_state, ENTER_OP1, ENTER_OP2, CALC)
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 (CALC = '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 CUNEXT_STATE_DECODE;
```

```
CU_OUTPUT_DECODE: process (cu_cur_state)
begin
   case(cu_cur_state) is
      when cu_rst       =>
         IN_SELECT      <= '0';
         OP_SELECT      <= "00";
         RAM_ADDR       <= "00";
         RAM_WR         <= '0';
         ACC_RST        <= '1';
         ACC_WR         <= '0';
      when cu_idle      =>
         IN_SELECT      <= '0';
         OP_SELECT      <= "00";
         RAM_ADDR       <= "00";
         RAM_WR         <= '0';
         ACC_RST        <= '0';
         ACC_WR         <= '0';
      when cu_load_op1  =>
         IN_SELECT      <= '0';
         OP_SELECT      <= "00";
         RAM_ADDR       <= "00";
         RAM_WR         <= '1';
         ACC_RST        <= '0';
         ACC_WR         <= '1';
      when cu_load_op2  =>
         IN_SELECT      <= '0';
         OP_SELECT      <= "00";
         RAM_ADDR       <= "01";
         RAM_WR         <= '1';
         ACC_RST        <= '0';
         ACC_WR         <= '1';
      when cu_run_calc0 =>
         IN_SELECT      <= '1';
         OP_SELECT      <= "00";
         RAM_ADDR       <= "00";
         RAM_WR         <= '0';
         ACC_RST        <= '0';
         ACC_WR         <= '1';
```

```
        when cu_run_calc1 =>
           IN_SELECT      <= '1';
           OP_SELECT      <= "01";
           RAM_ADDR       <= "00";
           RAM_WR         <= '0';
           ACC_RST        <= '0';
           ACC_WR         <= '1';
        when cu_run_calc2 =>
           IN_SELECT      <= '1';
           OP_SELECT      <= "01";
           RAM_ADDR       <= "01";
           RAM_WR         <= '0';
           ACC_RST        <= '0';
           ACC_WR         <= '1';
        when cu_run_calc3 =>
           IN_SELECT      <= '0';
           OP_SELECT      <= "10";
           RAM_ADDR       <= "00";
           RAM_WR         <= '0';
           ACC_RST        <= '0';
           ACC_WR         <= '1';
        when cu_finish    =>
           IN_SELECT      <= '0';
           OP_SELECT      <= "00";
           RAM_ADDR       <= "00";
           RAM_WR         <= '0';
           ACC_RST        <= '0';
           ACC_WR         <= '0';
        when others     =>
           IN_SELECT      <= '0';
           OP_SELECT      <= "00";
           RAM_ADDR       <= "00";
           RAM_WR         <= '0';
           ACC_RST        <= '0';
           ACC_WR         <= '0';
      end case;
   end process CU_OUTPUT_DECODE;

end Behavioral;
```

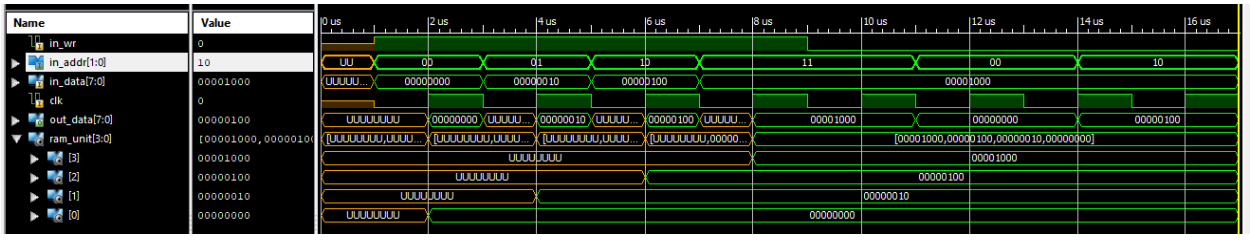
8. Модуляція роботи



9. Створення файлу VHDL , який реалізовує логіку RAM

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity RAM_module is
port(
   IN_WR     : IN  STD_LOGIC;
   IN_ADDR   : IN  STD_LOGIC_VECTOR(1 downto 0);
   IN_DATA   : IN  STD_LOGIC_VECTOR(7 downto 0);
   CLK       : IN  STD_LOGIC;

   OUT_DATA : OUT STD_LOGIC_VECTOR(7 downto 0)
);
end RAM_module;

architecture Behavioral of RAM_module is

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


begin

   RAM: process(CLK, IN_ADDR, RAM_UNIT)
   begin
      if(rising_edge(CLK)) then
      if (IN_WR = '1') then
            RAM_UNIT(conv_integer(IN_ADDR)) <= IN_DATA;
         end if;
      end if;
      OUT_DATA <= RAM_UNIT(conv_integer(IN_ADDR));
   end process RAM;

end Behavioral;
```

10. Модуляція роботи



11. Створення файлу VHDL , який реалізовує логіку декодера для 7-сегментного індикатора.

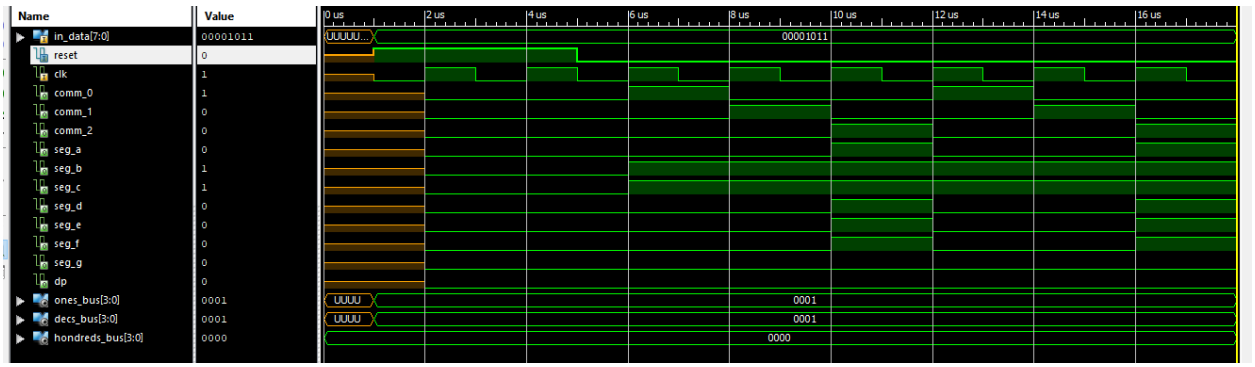
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity SEVEN_SEG_DECODER is
port(
        IN_DATA         : IN  STD_LOGIC_VECTOR(7 downto 0);
        RESET                   : IN  STD_LOGIC;
        CLK             : IN  STD_LOGIC;

        COMM_0          : OUT STD_LOGIC;
        COMM_1          : OUT STD_LOGIC;
        COMM_2          : 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 SEVEN_SEG_DECODER;

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

      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(CLK)
              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";
```

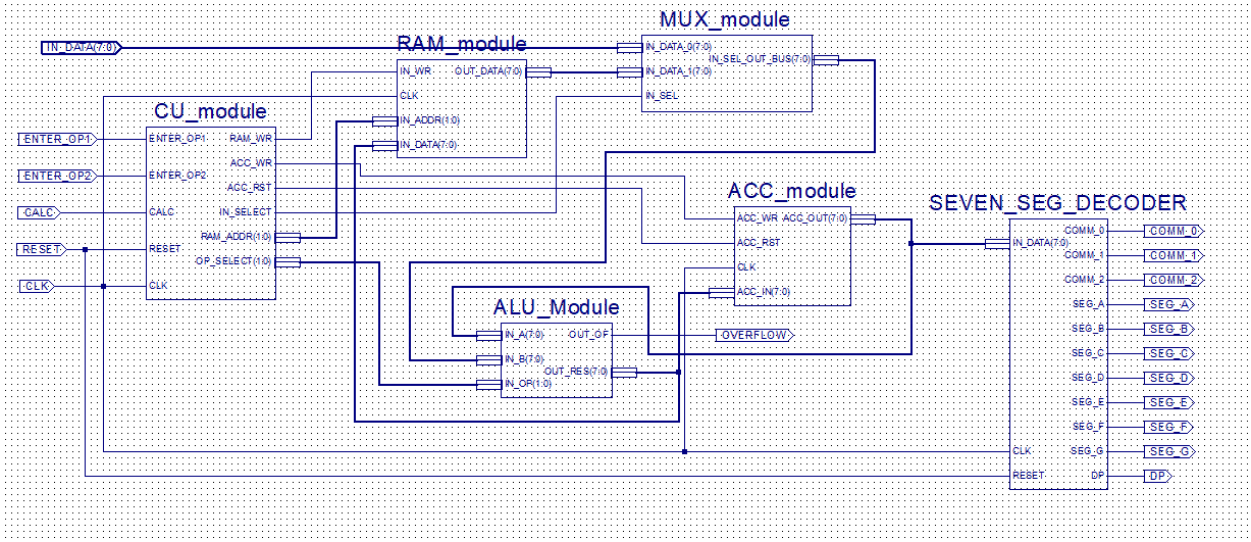
```
begin
	if (rising_edge(CLK)) then
		if(RESET = '0') then
			case CUR_DIGIT is
				when ONES =>
					DIGIT_VAL := ONES_BUS;
					CUR_DIGIT := DECS;
					COMMONS_CTRL := "001";
				when DECS =>
					DIGIT_VAL := DECS_BUS;
					CUR_DIGIT := HUNDREDS;
					COMMONS_CTRL := "010";
				when HUNDREDS =>
					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_0    <= COMMONS_CTRL(0);
                                COMM_1    <= COMMONS_CTRL(1);
                                COMM_2    <= 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;
```

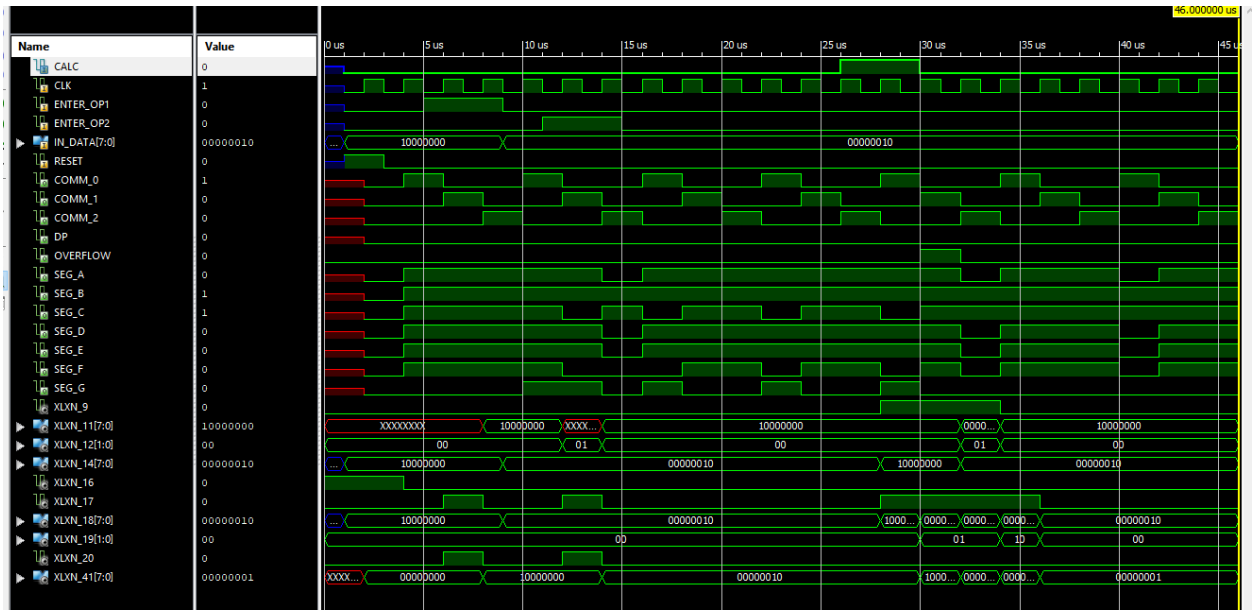
12. Модуляція роботи



13. Створення файлу для кінцевої схеми та реалізація автомата



14. Модуляція роботи кінцевої схеми



15. Створення файлу конфігурації

```
CONFIG VCCAUX = "3.3" ;

 # Clock 12 MHz
  NET "CLK"                   LOC = P129   | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;

##############################################################################################################
#                                  Seven Segment Display
##############################################################################################################

    NET "DP"        LOC = P117   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_G"     LOC = P116   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_F"     LOC = P115   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_E"     LOC = P113   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_D"     LOC = P112   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_C"     LOC = P111   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_B"     LOC = P110   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "SEG_A"     LOC = P114   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

    NET "COMM_2"    LOC = P124   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "COMM_1"    LOC = P121   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "COMM_0"    LOC = P120   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

##############################################################################################################
#                                        LED
##############################################################################################################

    NET "OVERFLOW"              LOC = P55    | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

##############################################################################################################
#                                    DP Switches
##############################################################################################################

    NET "IN_DATA(7)"         LOC = P70    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(6)"         LOC = P69    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(5)"         LOC = P68    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(4)"         LOC = P64    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(3)"         LOC = P63    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(2)"         LOC = P60    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(1)"         LOC = P59    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "IN_DATA(0)"         LOC = P58    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

##############################################################################################################
#                                    Switches
##############################################################################################################

    NET "ENTER_OP1"           LOC = P80    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "ENTER_OP2"           LOC = P79    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "CALC"                LOC = P78    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "RESET"               LOC = P75    | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

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