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”

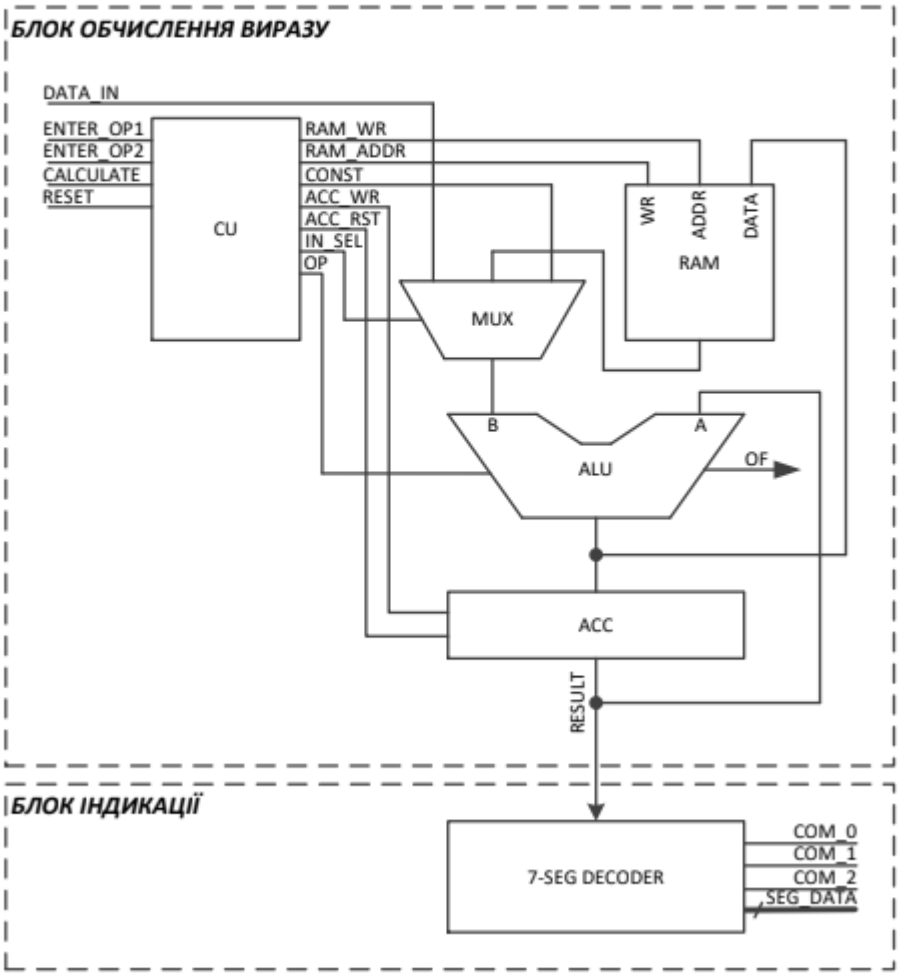
Варіант – 08

Виконав: ст.гр. КІ-202
Жук Р.Р.
Перевірив:
асистент
Козак Н.Б.

Львів 2023

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 *(ALU)* повинен виконувати за один такт одну операцію), та реалізованим згідно наступної структурної схеми (*Малюнок 1*):



Малюнок 1 - Структурна схема автомата.

Завдання

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

Виконання роботи:

CU.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity CU_INTF 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);
		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 CU_INTF;

architecture CU_ARCH of 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
	clk : 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 clk;

	next_state : process (cu_cur_state, ENTER_OP1, ENTER_OP2,
CALCULATE)
	begin
		cu_next_state <= cu_cur_state;
```

```
	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 next_state;

output : 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 <= "01";
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 <= "01";
RAM_ADDR_BUS <= "01";
RAM_WR <= '0';
ACC_RST <= '0';
ACC_WR <= '1';
when cu_run_calc2 =>
IN_SEL <= "01";
OP_CODE_BUS <= "10";
RAM_ADDR_BUS <= "01";
RAM_WR <= '0';
ACC_RST <= '0';
ACC_WR <= '1';
when cu_run_calc3 =>
IN_SEL <= "01";
OP_CODE_BUS <= "11";
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 output;

end CU_ARCH;
```

RAM.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity RAM_INTF is
        port (
                CLOCK : in std_logic;
                WR : in std_logic;
                DATA_IN_BUS : in std_logic_vector (7 downto 0);
                ADDRESS_BUS : in std_logic_vector (1 downto 0);
                DATA_OUT : out std_logic_vector (7 downto 0)
        );
end RAM_INTF;

architecture RAM_ARCH of RAM_INTF is
        type ram_type is array (3 downto 0) of std_logic_vector(7 downto 0);
        signal ram_unit : ram_type;
begin
        ram : process (CLOCK, ADDRESS_BUS, DATA_IN_BUS)
        begin
                if (rising_edge(CLOCK)) then
                        if (WR = '1') then
                                ram_unit(conv_integer(ADDRESS_BUS)) <=
DATA_IN_BUS;
                        end if;
                end if;
                DATA_OUT <= ram_unit(conv_integer(ADDRESS_BUS));
        end process ram;
```

```
end RAM_ARCH;
```

MUX.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX_INTF is
	port (
		SEL_IN_BUS : in std_logic_vector (1 downto 0);
		RAM_DATA_BUS : in std_logic_vector (7 downto 0);
		DATA_INPUT_BUS : in std_logic_vector (7 downto 0);
		DATA_OUT : out std_logic_vector (7 downto 0)
	);
end MUX_INTF;

architecture MUX_ARCH of MUX_INTF is
	signal const : std_logic_vector(7 downto 0);
begin
	const <= "00000000";
	mux : process (SEL_IN_BUS, DATA_INPUT_BUS, RAM_DATA_BUS)
	begin
		case (SEL_IN_BUS) is
			when "00" => DATA_OUT <= DATA_INPUT_BUS;
			when "01" => DATA_OUT <= RAM_DATA_BUS;
			when others => DATA_OUT <= const;
		end case;
	end process;
end MUX_ARCH;
```

ALU.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity ALU_INTF is
	port (
		OP_CODE_BUS : in std_logic_vector(1 downto 0);
		MUX_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 ALU_INTF;

architecture ALU_ARCH of ALU_INTF is

begin
	alu : process (OP_CODE_BUS, MUX_OUT_BUS, ACC_DATA_OUT_BUS)
```

```
		variable a : unsigned(7 downto 0);
		variable b : unsigned(7 downto 0);
		variable temp_mul : unsigned (15 downto 0);
	begin
		a := unsigned(ACC_DATA_OUT_BUS);
		b := unsigned(MUX_OUT_BUS);

		case(OP_CODE_BUS) is
			when "00" => ACC_DATA_IN_BUS <= std_logic_vector(b);
			when "01" => temp_mul := (a * b);
			ACC_DATA_IN_BUS <= std_logic_vector(temp_mul(7 downto
0));
			when "10" => ACC_DATA_IN_BUS <= std_logic_vector(a srl
1);
			when "11" => ACC_DATA_IN_BUS <= std_logic_vector(a + b);
			when others => ACC_DATA_IN_BUS <= "00000000";
		end case;
	end process alu;

end ALU_ARCH;
```

ALU.vhd

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

entity ACC_INTF is
port (
	CLOCK: in std_logic;
	DATA_IN_BUS: in std_logic_vector(7 downto 0);
	WR: in std_logic;
	RST: in std_logic;
	DATA_OUT_BUS: out std_logic_vector(7 downto 0));
end ACC_INTF;

architecture ACC_ARCH of ACC_INTF is
	signal DATA: std_logic_vector(7 downto 0);
begin
	ACC : process(CLOCK, DATA)
	 begin
		if (rising_edge(CLOCK)) then
			if(RST = '1') then
				DATA <= "00000000";
			elsif (WR = '1') then
				DATA <= DATA_IN_BUS;
			end if;
		end if;
```

```
            DATA_OUT_BUS <= DATA;
        end process ACC;

end ACC_ARCH;
```

DECODER.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity 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 DECODER_INTF;

architecture DECODER_ARCH of 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);
            hex_src := hex_src(hex_src'left - 1 downto hex_src'right) & '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
```

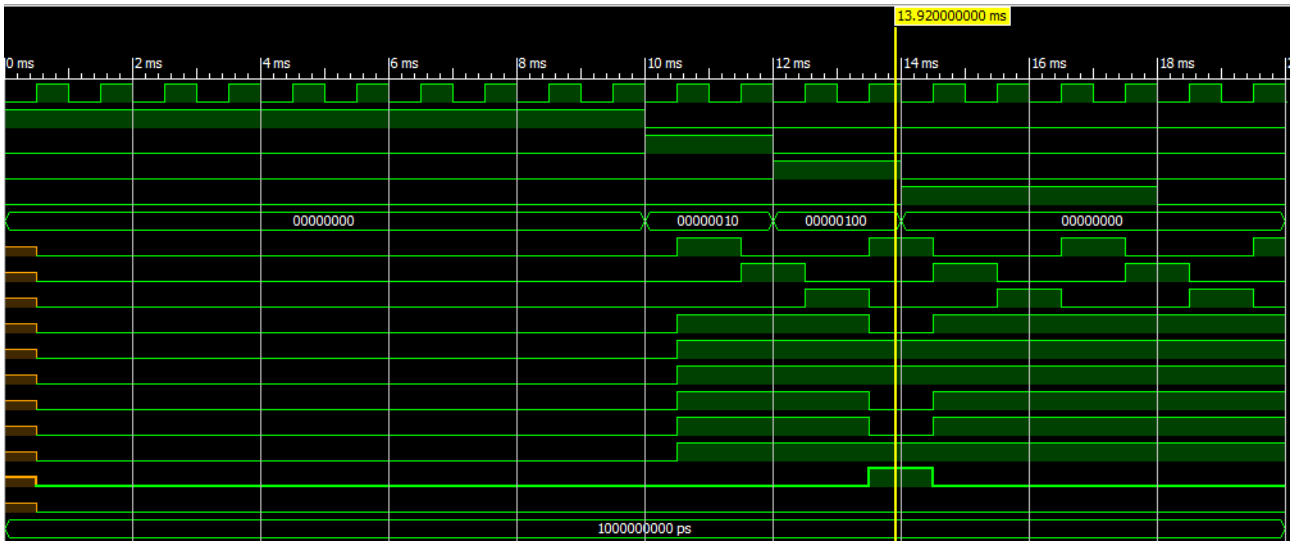
```
                        when "0000" => digit_ctrl := "1111110";
                        when "0001" => digit_ctrl := "0110000";
                        when "0010" => digit_ctrl := "1101101";
                        when "0011" => digit_ctrl := "1111001";
                        when "0100" => digit_ctrl := "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 DECODER_ARCH;
```



Результат роботи

Висновок: Під час даної лабораторної роботи, я на базі стенда Elbert V2 – Spartan 3A FPGA, реалізував цифровий автомат для обчислення значення заданого виразу.