

Міністерство освіти і науки України  
Національний університет «Львівська політехніка»

Кафедра ЕОМ



Звіт

до 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»

Варіант №7

Виконав:  
ст. гр. КІ-201  
Глинський В.А.  
Прийняв:  
ст. викладач  
каф. ЕОМ  
Козак Н. Б.

Львів 2024

**Мета роботи:** На базі стенда реалізувати цифровий автомат для обчислення значення виразів.

### Виконання роботи:

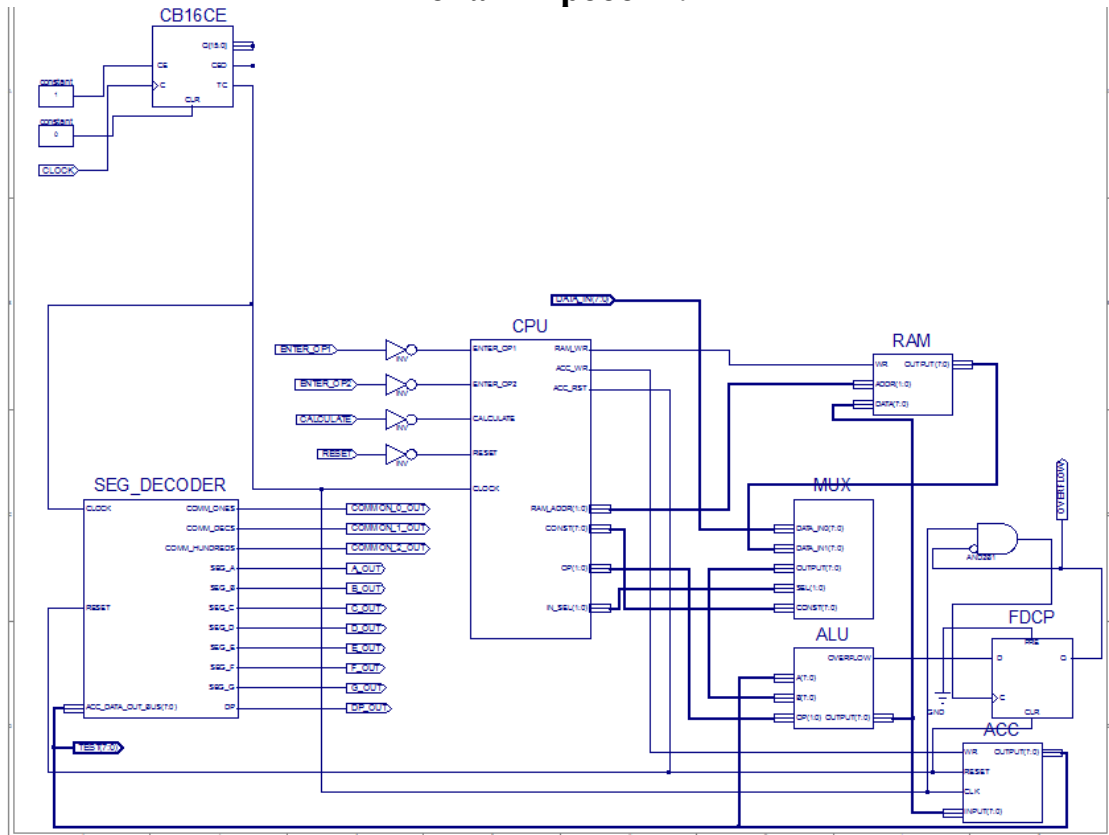


Рис. 1 – Top Level

Файл ACC.vhd

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

entity ACC is
    Port ( WR : in STD_LOGIC;
          RESET : in STD_LOGIC;
          CLK : in STD_LOGIC;
          INPUT : in STD_LOGIC_VECTOR (7 downto 0);
          OUTPUT : 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 RESET = '1' then
      DATA <= (others => '0');
    elsif WR = '1' then
      DATA <= INPUT;
    end if;
  end if;
end process;

OUTPUT <= DATA;

end ACC_arch;

```

```

Файл ALU.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
  Port ( A : in  STD_LOGIC_VECTOR(7 downto 0);
        B : in  STD_LOGIC_VECTOR(7 downto 0);
        OP : in  STD_LOGIC_VECTOR(1 downto 0);
        OUTPUT : out STD_LOGIC_VECTOR(7 downto 0);
        OVERFLOW: out STD_LOGIC);
end ALU;

architecture ALU_Behavioral of ALU is
  signal ALUR: STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
  signal Carry: STD_LOGIC := '0';
begin
  process(A, B, OP)
  begin
    case (OP) is
      when "01" => ALUR <= ("00000000" & A) +
("00000000" & B);
      when "10" => ALUR <= ("00000000" & A) +
("11111111" & not B) + "0000000000000001";
      when "11" => case(B) is

```

```

                                when x"00"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 0);
                                when x"01"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 1);
                                when x"02"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 2);
                                when x"03"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 3);
                                when x"04"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 4);
                                when x"05"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 5);
                                when x"06"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 6);
                                when x"07"      => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 7);
                                when others => ALUR <=
std_logic_vector(unsigned(("00000000" & A)) sll 0);
                                end case;
                                ALUR(15 downto 8) <= "00000000";
                                when others => ALUR <= ("00000000" & B);
                                end case;
                                end process;
                                OUTPUT <= ALUR(7 downto 0);
                                OVERFLOW <= ALUR(8) OR ALUR(9) OR ALUR(10) OR
ALUR(11) OR ALUR(12) OR ALUR(13) OR ALUR(14) OR ALUR(15);
                                end ALU_Behavioral;

```

Файл CPU.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity CPU 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 : OUT STD\_LOGIC\_VECTOR(1

DOWNT0 0);

CONST : OUT STD\_LOGIC\_VECTOR(7 DOWNT0 0);

```

        ACC_WR : OUT STD_LOGIC;
        ACC_RST : OUT STD_LOGIC;
        IN_SEL : OUT STD_LOGIC_VECTOR(1 downto 0);
        OP : OUT STD_LOGIC_VECTOR(1 DOWNT0 0));
end CPU;

architecture CPU_arch of CPU 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
    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 (CLOCK, ENTER_OP1,
ENTER_OP2, CALCULATE)
    begin
        NEXT_STATE <= CUR_STATE;

        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 =>

RAM\_WR <= '0';

RAM\_ADDR <= "00";

CONST <= "00000000";

ACC\_WR <= '0';

ACC\_RST <= '1';

IN\_SEL <= "00";

OP <= "00";

when LOAD\_OP1 =>

RAM\_WR <= '1';

RAM\_ADDR <= "00";

CONST <= "00000000";

ACC\_WR <= '0';

ACC\_RST <= '1';

IN\_SEL <= "00";

OP <= "00";

when LOAD\_OP2 =>

```

        RAM_WR <= '1';
        RAM_ADDR <= "01";
        CONST <= "00000000";
        ACC_WR <= '0';
        ACC_RST <= '1';
        IN_SEL <= "00";
        OP <= "00";
when RUN_CALC0 =>
    RAM_WR <= '0';
    RAM_ADDR <= "00";
    CONST <= "00000000";
    ACC_WR <= '1';
    ACC_RST <= '0';
    IN_SEL <= "01";
    OP <= "00";
when RUN_CALC1 =>
    RAM_WR <= '0';
    RAM_ADDR <= "01";
    CONST <= "00000010";
    ACC_WR <= '1';
    ACC_RST <= '0';
    IN_SEL <= "10";
    OP <= "11";
when RUN_CALC2 =>
    RAM_WR <= '0';
    RAM_ADDR <= "01";
    CONST <= "00000000";
    ACC_WR <= '1';
    ACC_RST <= '0';
    IN_SEL <= "01";
    OP <= "10";
when RUN_CALC3 =>
    RAM_WR <= '0';
    RAM_ADDR <= "01";
    CONST <= "00000000";
    ACC_WR <= '1';
    ACC_RST <= '0';
    IN_SEL <= "01";
    OP <= "01";
when RUN_CALC4 =>
    RAM_WR <= '0';
    RAM_ADDR <= "00";
    CONST <= "00001010";

```

```

        ACC_WR <= '1';
        ACC_RST <= '0';
        IN_SEL <= "10";
        OP <= "01";
    when IDLE =>
        RAM_WR <= '0';
        RAM_ADDR <= "00";
        CONST <= "00000000";
        ACC_WR <= '0';
        ACC_RST <= '0';
        IN_SEL <= "00";
        OP <= "00";
    when others =>
        RAM_WR <= '0';
        RAM_ADDR <= "00";
        CONST <= "00000000";
        ACC_WR <= '0';
        ACC_RST <= '0';
        IN_SEL <= "00";
        OP <= "00";
    end case;
end process;
end CPU_arch;

```

```

Файл RAM.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RAM is
    port(
        WR : IN STD_LOGIC;
        ADDR : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        DATA : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        CLOCK: IN STD_LOGIC;
        OUTPUT : 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(ADDR, CLOCK, UNIT)
    begin
        if(rising_edge(CLOCK)) then
            if (WR = '1') then
                UNIT(conv_integer(ADDR)) <= DATA;
            end if;
        end if;
        OUTPUT <= UNIT(conv_integer(ADDR));
    end process;
end RAM_arch;

```

```

Файл SEG_DECODER.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SEG_DECODER is
    port( CLOCK : IN STD_LOGIC;
          RESET : IN STD_LOGIC;
          ACC_DATA_OUT_BUS : IN STD_LOGIC_VECTOR(7
DOWNT0 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 SEG_DECODER;

architecture Behavioral of 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 (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";
            end case;
        end if;
    end if;
end;

```

```

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    <= not COMMONS_CTRL(0);
COMM_DECS    <= not COMMONS_CTRL(1);
COMM_HUNDREDS <= not
COMMONS_CTRL(2);

SEG_A <= not DIGIT_CTRL(6);
SEG_B <= not DIGIT_CTRL(5);
SEG_C <= not DIGIT_CTRL(4);
SEG_D <= not DIGIT_CTRL(3);
SEG_E <= not DIGIT_CTRL(2);
SEG_F <= not DIGIT_CTRL(1);
SEG_G <= not DIGIT_CTRL(0);
DP    <= '1';

end if;
end process INDICATE;

end Behavioral;

```

```

Файл TopLevelTest.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY TopLevel_TopLevel_sch_tb IS

```

```
END TopLevel_TopLevel_sch_tb;  
ARCHITECTURE behavioral OF TopLevel_TopLevel_sch_tb IS
```

```
    COMPONENT TopLevel
```

```
    PORT( CLOCK :      IN    STD_LOGIC;  
          RESET  :      IN    STD_LOGIC;  
          ENTER_OP1 :      IN    STD_LOGIC;  
          ENTER_OP2 :      IN    STD_LOGIC;  
          CALCULATE :      IN    STD_LOGIC;  
          DATA_IN :      IN    STD_LOGIC_VECTOR (7 DOWNT0 0);  
          COMMON_0_OUT :      OUT STD_LOGIC;  
          COMMON_1_OUT :      OUT STD_LOGIC;  
          COMMON_2_OUT :      OUT STD_LOGIC;  
          TEST: OUT STD_LOGIC_VECTOR(7 downto 0);  
          A_OUT :      OUT STD_LOGIC;  
          B_OUT :      OUT STD_LOGIC;  
          C_OUT :      OUT STD_LOGIC;  
          D_OUT :      OUT STD_LOGIC;  
          E_OUT :      OUT STD_LOGIC;  
          F_OUT :      OUT STD_LOGIC;  
          G_OUT :      OUT STD_LOGIC;  
          DP_OUT :      OUT STD_LOGIC;  
          OVERFLOW :      OUT STD_LOGIC);  
    END COMPONENT;
```

```
    SIGNAL CLOCK      :      STD_LOGIC := '0';  
    SIGNAL RESET :      STD_LOGIC;  
    SIGNAL ENTER_OP1 :      STD_LOGIC;  
    SIGNAL ENTER_OP2 :      STD_LOGIC;  
    SIGNAL CALCULATE :      STD_LOGIC;  
    SIGNAL DATA_IN   :      STD_LOGIC_VECTOR (7 DOWNT0 0);  
    SIGNAL COMMON_0_OUT :      STD_LOGIC;  
    SIGNAL COMMON_1_OUT :      STD_LOGIC;  
    SIGNAL COMMON_2_OUT :      STD_LOGIC;  
    SIGNAL A_OUT:      STD_LOGIC;  
    SIGNAL B_OUT:      STD_LOGIC;  
    SIGNAL C_OUT:      STD_LOGIC;  
    SIGNAL D_OUT :      STD_LOGIC;  
    SIGNAL E_OUT:      STD_LOGIC;  
    SIGNAL F_OUT:      STD_LOGIC;  
    SIGNAL G_OUT :      STD_LOGIC;  
    SIGNAL DP_OUT :      STD_LOGIC;  
    SIGNAL OVERFLOW :      STD_LOGIC;
```

```

    SIGNAL TEST: STD_LOGIC_VECTOR(7 downto 0);
    SIGNAL TEST1: STD_LOGIC_VECTOR(7 downto 0);
    SIGNAL TEST2: STD_LOGIC_VECTOR(7 downto 0);

--    constant CLOCK_period : time := 166ns;
    constant CLKP: time := 24ms;--48ms;

BEGIN

    UUT: TopLevel PORT MAP(
        CLOCK => CLOCK,
        RESET => RESET,
        ENTER_OP1 => ENTER_OP1,
        ENTER_OP2 => ENTER_OP2,
        CALCULATE => CALCULATE,
        DATA_IN => DATA_IN,
        COMMON_0_OUT => COMMON_0_OUT,
        COMMON_1_OUT => COMMON_1_OUT,
        COMMON_2_OUT => COMMON_2_OUT,
        A_OUT => A_OUT,
        B_OUT => B_OUT,
        C_OUT => C_OUT,
        D_OUT => D_OUT,
        E_OUT => E_OUT,
        F_OUT => F_OUT,
        G_OUT => G_OUT,
        DP_OUT => DP_OUT,
        OVERFLOW => OVERFLOW,
        TEST => TEST
    );

    CLOCK_process: process
    begin
        CLOCK <= '0';
        wait for 83ns;
        CLOCK <= '1';
        wait for 83ns;
    end process;

-- *** Test Bench - User Defined Section ***
    tb : PROCESS
    BEGIN
        --ENTER_OP1 <= '0';

```

```

--ENTER_OP2 <= '0';
--CALCULATE <= '0';
--DATA_IN <= (others => '0');
--RESET <= '1';
--wait for CLKP * 4;
--RESET <= '0';
--DATA_IN <= "00000000"; -- A
--ENTER_OP1 <= '1';
--wait for CLKP * 4;
--ENTER_OP1 <= '0';
--wait for CLKP * 4;
--DATA_IN <= "00000001"; -- B
--ENTER_OP2 <= '1';
--wait for CLKP * 4;
--ENTER_OP2 <= '0';
--wait for CLKP * 4;
--CALCULATE <= '1'; -- START CALCULATION

```

*lp1: for i in 1 to 1 loop*

*lp2: for j in 2 to 2 loop*

```

    TEST1 <= std_logic_vector(to_unsigned(i, 8)) nor
std_logic_vector(to_unsigned(j, 8));

```

```

    TEST2 <=

```

```

std_logic_vector(to_unsigned(to_integer(signed(std_logic_vector(to_signed(i,
8)) nor std_logic_vector(to_unsigned(j, 8))))) + j + 7, 8));

```

```

    ENTER_OP1 <= '1';

```

```

    ENTER_OP2 <= '1';

```

```

    CALCULATE <= '1';

```

```

    DATA_IN <= (others => '0');

```

```

    RESET <= '0';

```

```

    wait for CLKP;

```

```

    RESET <= '1';

```

```

    wait for CLKP;

```

```

    DATA_IN <= std_logic_vector(to_unsigned(i, 8)); --

```

*A*

```

    ENTER_OP1 <= '0';

```

```

    wait for CLKP;

```

```

    ENTER_OP1 <= '1';

```

```

    wait for CLKP;

```

```

    DATA_IN <= std_logic_vector(to_unsigned(j, 8)); --

```

*B*

```

    ENTER_OP2 <= '0';

```

```

    wait for CLKP;

```

```

ENTER_OP2 <= '1';
wait for CLKP;
CALCULATE <= '0'; -- START CALCULATION
wait for CLKP* 7;
assert TEST = TEST2 severity FAILURE;
wait for CLKP;
end loop;
end loop;

WAIT; -- will wait forever
END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;

```

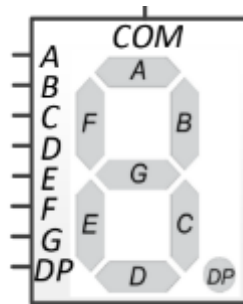


Рис.8 – 7-сегментний індикатор

Переглянемо часову діаграму 7seg decoder.

Бачимо, що для вхідного числа  $00001111(2) = 15(10)$

Ми отримуємо такі значення(якщо значення не вказане, вважаємо його за одиницю):

COMM\_ONES = 0: A,C,D, F, G = 0, що відповідає числу 5

COMM\_DECS = 0: B,C = 0, що відповідає числу 1

COMM\_HUNDREDS = 0: A,B,C,D,E,F = 0, що відповідає числу 0

Отримуємо значення 015, що збігається з даним йому.

Рис.9 – Успішна прошивка

**Висновок:** Виконуючи дану лабораторну роботу я навчився реалізовувати цифровий автомат для обчислення значення виразів використовуючи засоби VHDL.