Міністерство освіти і науки України
Національний університет “Львівська політехніка”

Кафедра ЕОМ

Звіт

про виконання лабораторної роботи №2
з дисципліни:
“Моделювання комп’ютерних систем”

Виконав: ст. гр. КІ-202
Ключко Д.С.
Прийняв:
Старший викладач Козак Н.Б.

Львів 2023

Тема роботи. Структурний опис цифрового автомата. Перевірка роботи автомата за допомогою стенда **Elbert V2 – Spartan 3A FPGA**.

Мета роботи. На базі стенда реалізувати цифровий автомат світлових ефектів згідно вимог.

Етапи роботи:

1. Інтерфейс пристрою та функціонал реалізувати згідно отриманого варіанту завдання. Дивись розділ ***Завдання***.
2. Логіку переходів реалізувати з використанням мови опису апаратних засобів *VHDL*. Заборонено використовуючи оператори *if, switch, for, when*.
3. Логіку формування вихідних сигналів реалізувати з використанням мови опису апаратних засобів *VHDL*. Заборонено використовуючи оператори *if, switch, for, when*.
4. Згенерувати *Schematic* символи для *VHDL* описів логіки переходів та логіки формування вихідних сигналів.
5. Зінтегрувати всі компоненти (логіку переходів, логіку формування вихідних сигналів та пам'ять станів) в єдину систему за допомогою *ISE WebPACK™ Schematic Capture*. Пам'ять станів реалізувати за допомогою графічних компонентів з бібліотеки.
6. Промоделювати роботу окремих частин автомата та автомата вцілому за допомогою симулятора *ISim*.
7. Інтегрувати створений автомат зі стендом *Elbert V2 – Spartan 3A FPGA* (додати подільник частоти для вхідного тактовового сигналу, призначити фізичні виводи на *FPGA*).
8. Згенерувати *BIT* файал та перевірити роботу за допомогою стенда *Elbert V2 – Spartan 3A FPGA*.
9. Підготувати і захистити звіт.

Завдання:

Так як я (12) варіант за журнальним списком, я виконую шостий варіант завдання, а саме:

Варіант – 6:

- Пристрій повинен реалізувати 8 комбінацій вихідних сигналів згідно таблиці:

Стан#	LED_0	LED_1	LED_2	LED_3	LED_4	LED_5	LED_6	LED_7
0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
2	1	1	1	0	0	0	0	0
3	1	1	1	1	0	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	1	1	0	0
6	0	0	0	0	1	1	1	0
7	0	0	0	0	1	1	1	1

- Пристрій повинен використовувати *12MHz* тактовий сигнал від мікроконтролера *IC1* і знижувати частоту за допомогою внутрішнього подільника. Мікроконтролер *IC1* є частиною стенда *Elbert V2 – Spartan 3A FPGA*. Тактовий сигнал заведено нв вхід *LOC = P129 FPGA* (див. ***Додаток – 1***).
- Інтерфейс пристрою повинен мати вхід синхронного скидання (*RESET*).
- Інтерфейс пристрою повинен мати вхід керування режимом роботи (*MODE*):
  - Якщо *MODE=0* то стан пристрою інкрементується по зростаючому фронту тактового сигналу пам'яті станів (0->1->2->3->4->5->6->7->0…).
  - Якщо *MODE=1* то стан пристрою декрементується по зростаючому фронту тактового сигналу пам'яті станів (0->7->6->5->4->3->2->1->0…).
- Інтерфейс пристрою повинен мати однорозрядний вхід (*TEST*) для подачі логічної «1» на всі непарні виходи одночасно:
  - Якщо *TEST=0* то автомат перемикає сигнали на виходах згідно заданого алгоритму.
  - Якщо *TEST=1* то на непарних виходах (7, 5, 3, 1) повинна бути логічна «1» (непарні *LED* увімкнені).
- Для керування сигналом *MODE* використати будь який з 8 *DIP* перемикачів (див. **Додаток – 1**).
- Для керування сигналами *RESET/TEST* використати будь якІ з *PUSH BUTTON* кнопок (див. **Додаток – 1**).

Виконання завдання

Я розпочав з того, що реалізував логіку виходів згідно мого варіанту. Створивши файл OutputLogic.vhd я реалізував 8 комбінацій вхідних сигналів згідно таблиці поданої вище. Також реалізував однорозрядний вхід test згідно умови: якщо вхід test активований то на непарних виходах буде логічна одиниця, якщо вхід test неактивний то переключення відбувається згідно заданого алгоритму.

```
----------------------------------------------------------------------------------
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 out_logic_intf is
      Port ( IN_BUS : in  std_logic_vector(2 downto 0);
             TEST : in std_logic;
             OUT_BUS : out  std_logic_vector(7 downto 0)
           );
end out_logic_intf;

architecture out_logic_arch of out_logic_intf is

begin
      OUT_BUS(0) <=  not(IN_BUS(2));
      OUT_BUS(1) <= (TEST or (not(IN_BUS(2)) and (IN_BUS(1) or IN_BUS(0))));
      OUT_BUS(2) <= (not(IN_BUS(2)) and IN_BUS(1));
      OUT_BUS(3) <= (TEST or (not(IN_BUS(2)) and IN_BUS(1) and IN_BUS(0)));
      OUT_BUS(4) <=  IN_BUS(2);
      OUT_BUS(5) <= (TEST or (IN_BUS(2) and (IN_BUS(1) or IN_BUS(0))));
      OUT_BUS(6) <= (IN_BUS(2) and IN_BUS(1));
      OUT_BUS(7) <= (TEST or (IN_BUS(2) and IN_BUS(1) and IN_BUS(0)));
end out_logic_arch;
```

Рис.1. OutputLogic.vhd

Другою умовою, яку потрібно було реалізувати логіку переходів. Я реалізував її у файлі TransitionLogic.vhd. Також там я реалізував логіку роботи сигналу mode та reset. При mode=1 стан пристрою декрементується по зростаючому фронту. При mode = 0 навпаки(інкрементується)

```
entity transition_logic_intf is
      Port ( CUR_STATE : in  std_logic_vector(2 downto 0);
           MODE : in  std_logic;
           RESET : in std_logic;
           NEXT_STATE : out  std_logic_vector(2 downto 0)
           );
end transition_logic_intf;

architecture transition_logic_arch of transition_logic_intf is

begin

    NEXT_STATE(0) <= (not(RESET) and not(MODE) and not(CUR_STATE(2)) and not (CUR_STATE(1)) and not (CUR_STATE(0))) or  -- 000-->001(0)
                     (not(RESET) and not(MODE) and not(CUR_STATE(2)) and CUR_STATE(1) and not (CUR_STATE(0))) or       -- 010-->011(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and not(CUR_STATE(1)) and not (CUR_STATE(0))) or       -- 100-->101(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and CUR_STATE(1) and not (CUR_STATE(0))) or            -- 110-->111(0)
                     (not(RESET) and MODE and not(CUR_STATE(2)) and not (CUR_STATE(1)) and not (CUR_STATE(0))) or      -- 000-->111(1)
                     (not(RESET) and MODE and CUR_STATE(2) and CUR_STATE(1) and not (CUR_STATE(0))) or                 -- 110-->101(1)
                     (not(RESET) and MODE and CUR_STATE(2) and not (CUR_STATE(1)) and not (CUR_STATE(0))) or           -- 100-->011(1)
                     (not(RESET) and MODE and not(CUR_STATE(2)) and CUR_STATE(1) and not (CUR_STATE(0)));              -- 010-->001(1)

    NEXT_STATE(1) <= (not(RESET) and not(MODE) and not(CUR_STATE(2)) and not (CUR_STATE(1)) and CUR_STATE(0)) or       -- 001-->010(0)
                     (not(RESET) and not(MODE) and not(CUR_STATE(2)) and CUR_STATE(1) and not (CUR_STATE(0))) or       -- 010-->011(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and not(CUR_STATE(1)) and CUR_STATE(0)) or             -- 101-->110(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and CUR_STATE(1) and not (CUR_STATE(0))) or            -- 110-->111(0)
                     (not(RESET) and MODE and not(CUR_STATE(2)) and not (CUR_STATE(1)) and not (CUR_STATE(0))) or      -- 000-->111(1)
                     (not(RESET) and MODE and CUR_STATE(2) and CUR_STATE(1) and CUR_STATE(0)) or                       -- 111-->110(1)
                     (not(RESET) and MODE and CUR_STATE(2) and not (CUR_STATE(1)) and not(CUR_STATE(0))) or            -- 100-->011(1)
                     (not(RESET) and MODE and not(CUR_STATE(2)) and CUR_STATE(1) and CUR_STATE(0));                    -- 011-->010(1)

    NEXT_STATE(2) <= (not(RESET) and not(MODE) and not(CUR_STATE(2)) and CUR_STATE(1) and CUR_STATE(0)) or            -- 011-->100(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and not(CUR_STATE(1)) and not (CUR_STATE(0))) or      -- 100-->101(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and not(CUR_STATE(1)) and CUR_STATE(0)) or            -- 101-->110(0)
                     (not(RESET) and not(MODE) and CUR_STATE(2) and CUR_STATE(1) and not (CUR_STATE(0))) or           -- 110-->111(0)
                     (not(RESET) and MODE and not(CUR_STATE(2)) and not (CUR_STATE(1)) and not (CUR_STATE(0))) or     -- 000-->111(1)
                     (not(RESET) and MODE and CUR_STATE(2) and CUR_STATE(1) and CUR_STATE(0)) or                      -- 111-->110(1)
                     (not(RESET) and MODE and CUR_STATE(2) and CUR_STATE(1) and not (CUR_STATE(0))) or                -- 110-->101(1)
                     (not(RESET) and MODE and CUR_STATE(2) and not(CUR_STATE(1)) and CUR_STATE(0));                   -- 101-->100(1)
end transition_logic_arch;
```

Рис.2. TransitionLogic.vhd

Третім етапом моєї роботи стало генерування схематік символів для файлів TransitionLogic.vhd та файлів OutputLogic.vhd після чого сформовані символи були додані у нещодавно створений файл LightController.vhd в якому реалізував автомат світлових ефектів

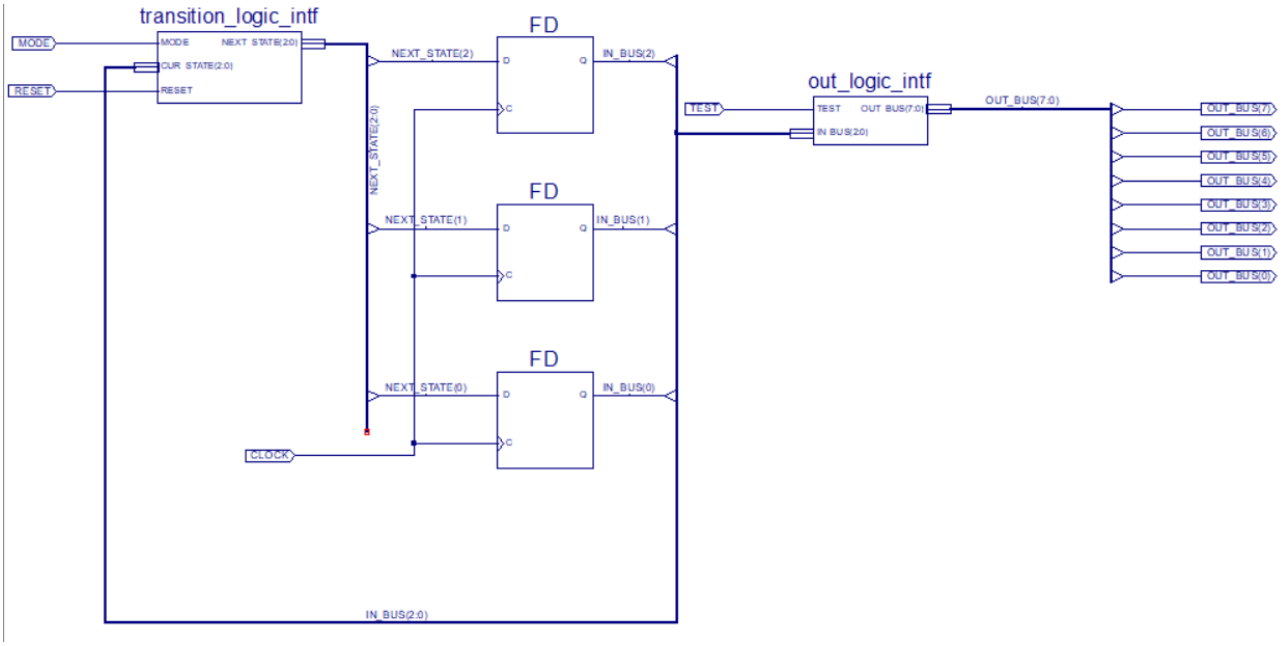

Рис.3. LightController.sch

Далі я переробив код, що реалізує підключення виводів схеми до фізичних виводів цільової FPGA.

```
#**************************************************************************************************************************#
#                                          UCF for ElbertV2 Development Board                                              #
#**************************************************************************************************************************#
CONFIG VCCAUX = "3.3" ;

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

    NET "OUT_BUS(0)"            LOC = P46   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(1)"            LOC = P47   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(2)"            LOC = P48   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(3)"            LOC = P49   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(4)"            LOC = P50   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(5)"            LOC = P51   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(6)"            LOC = P54   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "OUT_BUS(7)"            LOC = P55   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

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

    NET "TEST"          LOC = P70   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
    NET "MODE"          LOC = P69   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "DPSwitch[2]"          LOC = P68   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "DPSwitch[3]"          LOC = P64   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "DPSwitch[4]"          LOC = P63   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "DPSwitch[5]"          LOC = P60   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "DPSwitch[6]"          LOC = P59   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "DPSwitch[7]"          LOC = P58   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

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

    NET "RESET"         LOC = P80   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Switch[1]"          LOC = P79   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Switch[2]"          LOC = P78   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Switch[3]"          LOC = P77   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Switch[4]"          LOC = P76   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#    NET "Switch[5]"          LOC = P75   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

Рис.4. MyConstraints.ucf

Далі я створив схематік файл TopLevel.sch в якому реалізував подільник частоти згідно умови та з’єднав його з згенерованим схематік символом автомата світлових ефектів

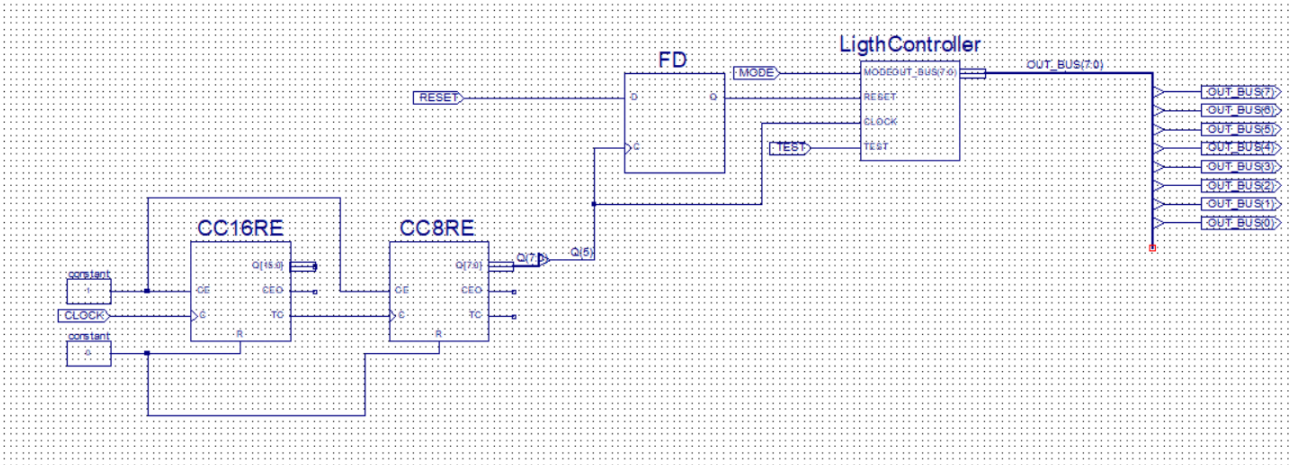

Рис.5. TopLevel.sch

Останнім кроком стало створення testbench файлу у якому я промоделював різні стани сигналів автомата.

```
    RESET <= '0';
    MODE <= '0';
    TEST <= '0';
    wait for 4000 ms;

    RESET <= '0';
    MODE <= '1';
    TEST <= '0';
    wait for 2500 ms;

    RESET <= '1';
    MODE <= '0';
    TEST <= '0';
    wait for 2000 ms;

    RESET <= '0';
    MODE <= '0';
    TEST <= '1';
    wait for 1000 ms;

    RESET <= '0';
    MODE <= '1';
    TEST <= '1';
    wait for 1000 ms;

    -- Add output check statements to verify the expected output
    -- assert OUT_BUS = "00000000" report "Test failed: Output does not match expected value" severity error;

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

  tb_clk: PROCESS
  BEGIN

    CLOCK <= not CLOCK;

    wait for 83 ns;
```

Рис.6. TestBench.vhd

Результат перевірки роботи схеми за допомогою симулятора ISim:

Висновок: Я згенерував схему автомата, який переходить у наступний стан і, взалежності від стану автомату, загоряються ті чи інші лампочки згідно варіанту, також взалежності від значення MODE міняється логіка переходу сигналів автомату, RESET скидає значення автомату в стан #0, в залежності від значення TEST загоряються LED-и згідно варіанту, як вказано у таблиці(TEST = 0) або непарні загоряються внезалежності від стану автомату, а інші LED-и міняються згідно таблиці (TEST = 1). У симуляторі ISim продемонстровано коректну роботу схеми.