UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Circuite integrate digitale

Proiectarea unui automat de stare

Realizat de: Mahalean Andra Gelia
Grupa: 2124/sg 3

Facultatea de Electronica,Telecomunicatii
si Tehnologia Informatiei

16 ianuarie 2024

Cuprins

1 Descrierea si implementarea circuitului combinational

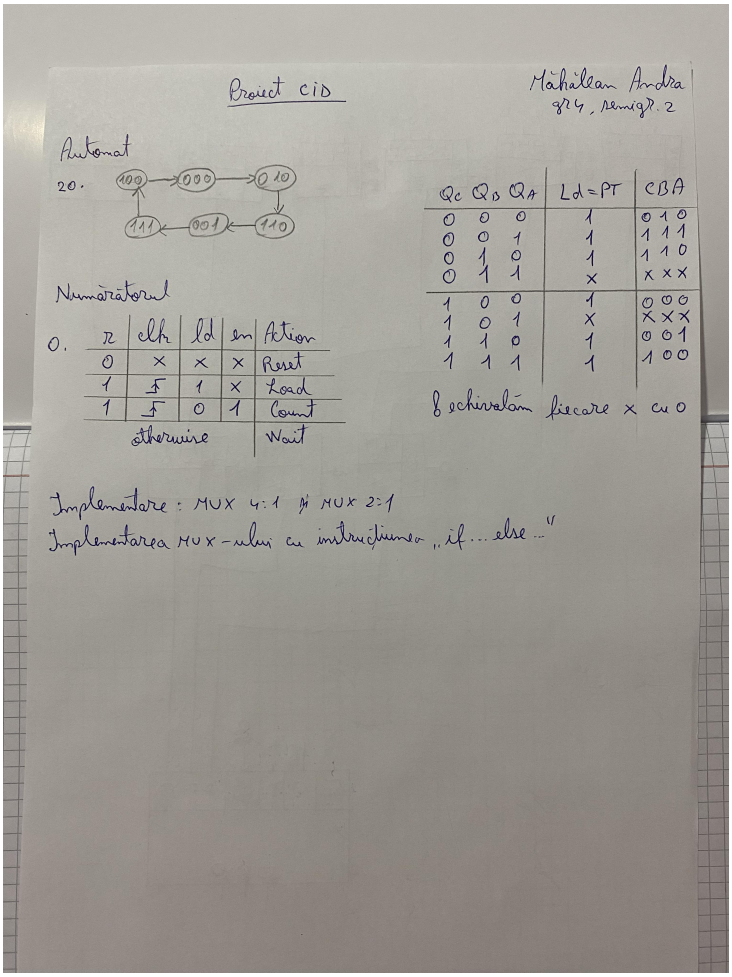

Figura 1: Rezolvarea circuitului pe hartie

Pentru realizarea automatului de stare ca și circuit combinațional am folosit un *multiplexor 4:1 si un multiplexor 2:1*, implementate cu instructiunea *if...else*

.

1.0.1 Implementarea codului

MUX 4:1:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_4_1 is
    Port ( i0 : in STD_LOGIC;
           i1 : in STD_LOGIC;
           i2 : in STD_LOGIC;
           i3 : in STD_LOGIC;
           a0 : in STD_LOGIC;
           a1 : in STD_LOGIC;
           y : out STD_LOGIC);
```

```
11 end mux_4_1;
12
13 architecture Behavioral of mux_4_1 is
14
15 signal a : std_logic_vector (1 downto 0);
16
17 begin
18
19     a <= a1 & a0;
20
21     process(i0,i1,i2,i3,a)
22     begin
23         if a="00" then
24
25          y <= i0;
26         elsif a="01" then
27         y <= i1;
28         elsif a="10" then
29         y <= i2;
30         elsif a="11" then
31          y <= i3;
32
33         end if;
34     end process;
35
36 end Behavioral;
```

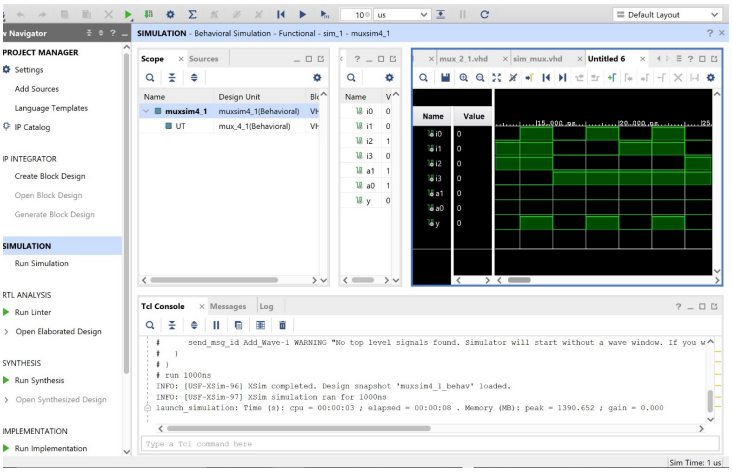

Figura 2: Simulare MUX 4:1

MUX 2:1:

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

entity mux_2_1 is
    Port (
        i0 : in STD_LOGIC;
        i1 : in STD_LOGIC;
        a0 : in STD_LOGIC;
        y : out STD_LOGIC
    );
end mux_2_1;

architecture Behavioral of mux_2_1 is
begin

    process(i0, i1, a0)
    begin
        if a0='0' then

                y <= i0;
            elsif a0='1' then
                y <= i1;

        end if;
    end process;

end Behavioral;
```

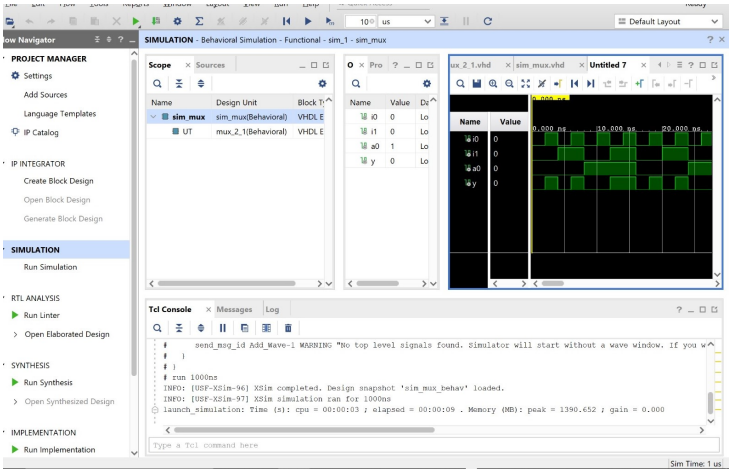

Figura 3: Simulare MUX 2:1

In codul de mai sus am descris algoritmul de implementare comportamentală al unui multiplexor 4:1 si un multiplexor 2:1 prin: declararea porturilor de intrare*(4 respectiv 2)*, adreselor*(2 respectiv 1)* si ieșirile acestora.

2 Descrierea si implementarea circuitului secvential

Pentru circuitul secvențial am folosit numaratorul N care are "load" activ pe '1' și "reset" activ pe '0'.

2.0.1 Implementarea codului

Numaratorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;--pentru numarare
entity cnt is
    Port ( d : in STD_LOGIC_VECTOR (2 downto 0);
           clk : in STD_LOGIC;
           r : in STD_LOGIC;
           en : in STD_LOGIC;
           ld : in STD_LOGIC;
           cy : out STD_LOGIC;
           q : out STD_LOGIC_VECTOR (2 downto 0));
end cnt;

architecture Behavioral of cnt is
signal qint: std_logic_vector(2 downto 0);
begin

    counter:process(r, clk)
 begin
        if r = '0' then
            qint <= "000";
        elsif (rising_edge(clk)) then
            if (ld = '1') then
                qint <= d;
            elsif (ld = '0' and en = '1') then
                qint <= qint+1;
            else
                qint <= qint;
            end if;
        end if;

    end process;

    q <= qint;
    cy <= '1' when (qint="111" and en='1') else '0';
```

```
37
38 end Behavioral;
```

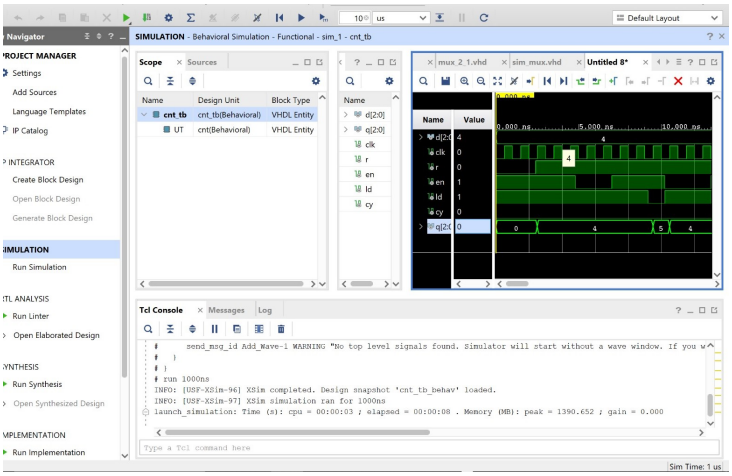

Figura 4: Simularea numărătorului

3 Implementarea finala a codului

Am realizat automatul secvențial cu secvențele *100 -> 000 -> 010 ->110 ->001 ->111.*

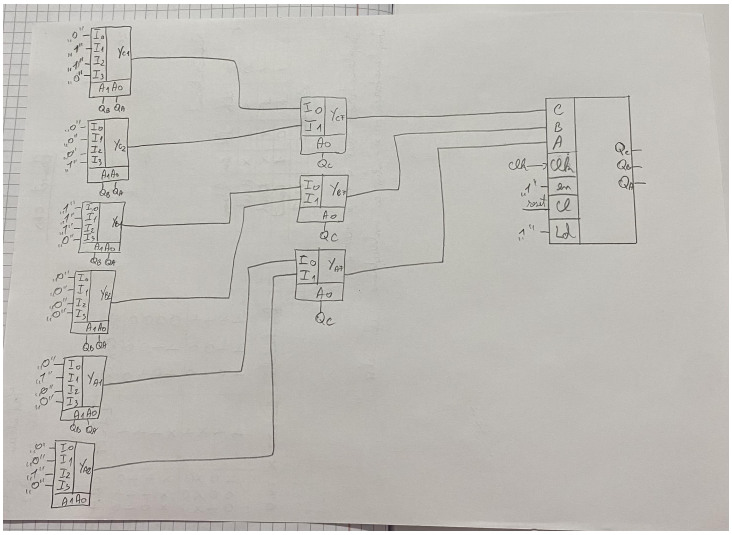

Figura 5: Desenul

Codul:

```
entity gen is
    Port (
        r : in STD_LOGIC;
        clk : in STD_LOGIC;
        s : out STD_LOGIC_VECTOR (2 downto 0));
end gen;

architecture Behavioral of gen is
    component mux4_1 is
        Port (
            i0 : in STD_LOGIC_VECTOR(2 downto 0);
            i1 : in STD_LOGIC_VECTOR(2 downto 0);
            i2 : in STD_LOGIC_VECTOR(2 downto 0);
            i3 : in STD_LOGIC_VECTOR(2 downto 0);
            a0 : in STD_LOGIC_VECTOR;
            a1 : in STD_LOGIC_VECTOR;
            y : out STD_LOGIC_VECTOR(2 downto 0));
    end component;

    component mux2_1 is
        Port (
            i0 : in STD_LOGIC_VECTOR(2 downto 0);
            i1 : in STD_LOGIC_VECTOR(2 downto 0);
            a0 : in STD_LOGIC;
            y : out STD_LOGIC_VECTOR(2 downto 0));
    end component;

```

```
28      component cnt is
29          Port (
30              d : in STD_LOGIC_VECTOR (2 downto 0);
31              clk : in STD_LOGIC;
32              r : in STD_LOGIC;
33              en : in STD_LOGIC;
34              ld : in STD_LOGIC;
35              cy : out STD_LOGIC;
36              q : out STD_LOGIC_VECTOR (2 downto 0));
37      end component;
38
39      signal en: std_logic := '1';
40      signal ld: std_logic := '0';
41      signal q : std_logic_vector (2 downto 0);
42      signal ym: std_logic_vector(2 downto 0);
43
44  begin
45      en <= '0' when ld = '1' else '1';
46      ld <= '1' when ym = "101" else '0';
47
48      U_cnt: cnt port map(
49          clk => clk,
50          r => r,
51          en => en,
52          ld => ld,
53          d => "000",
54          q => q
55      );
56
57      U_mux_4_1: mux_4_1 port map(
58          i0 => "100",
59          i1 => "110",
60          i2 => "111",
61          i3 => "101",
62          a0 => q(0),
63          a1 => q(1),
64          y => ym(2)
65      );
66
67      U_mux2_1: mux2_1 port map(
68          i0 => "001",
69          i1 => "000",
70          a0 => q(2),
71          y => s(2)
72      );
73       s(0) <= ym(0);
74       s(1) <= ym(1);
75
```

```
76 end Behavioral;
```

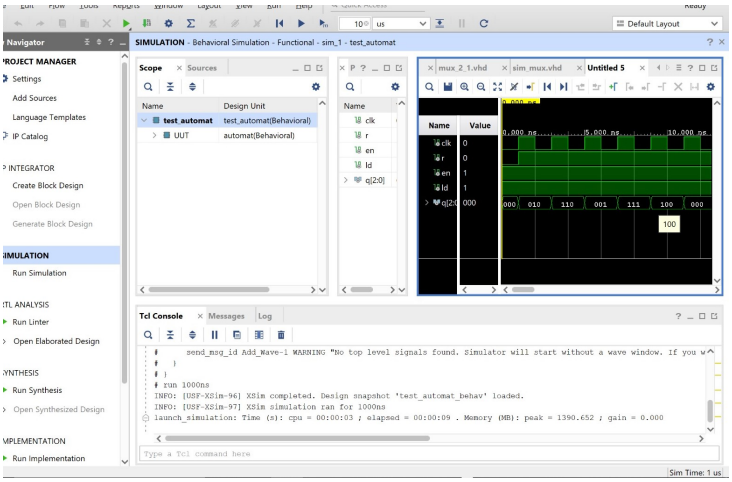

Figura 6: Simularea automatului

Descriere:

Inițial am setat semnalul *en* pe ’1’ pentru a număra o dată.

Am pus conditia ca valoarea semnalului *en* sa fie actualizata in functie de valoarea semnalului *ld*. Dacă semnalul *ld* este ’1’, atunci semnalul *en* devine ’0’, altfel devine ’1’.

Apoi am pus conditia ca dacă *ym* este "101", atunci semnalul *ld* devine ’1’, altfel devine ’0’.

Adresele multiplexoarelor sunt legate la ieșirea numărătorului.