Universitatea Tehnică Cluj-Napoca

Facultatea de Automatică şi Calculatoare

Secṭia: Calculatoare-română

Disciplina: Proiectarea Sistemelor Numerice



**Animație text**

*Proiect realizat de Ștef Paula-Elena*

*Grupa 30211*

*Profesor Indrumător*

*Sl. Dr. Ing Lucia Văcariu*

**Cuprins**

1. Specificaţie proiect…………………………pagina 3

2. Descriere schemă bloc cu componente…….pagina 4

3. Proiectare şi implementare

-schema de detaliu…………………………….pagina 5

- proiectare componente………………………pagina 6

- proiectare ansamblu …………………………pagina 20

- simulare în ActiveHDL………………………pagina 26

- fișierul de constrângeri din ISE………………pagina 30

4. Lista de componente utilizate……………….pagina 32

5. Semnificaţia notaţiilor de I/O şi a semnalelor interne………………………………………….pagina 33

6. Justificarea soluţiei alese……………………pagina 34

7. Posibilităţi de dezvoltare ulterioară…………pagina 35

8. Utilizare şi rezultate…………………………pagina 35

- resurse necesare

- descrierea utilizării

1. ***Specificație proiect***

*Proiectul nr.9 animația text*

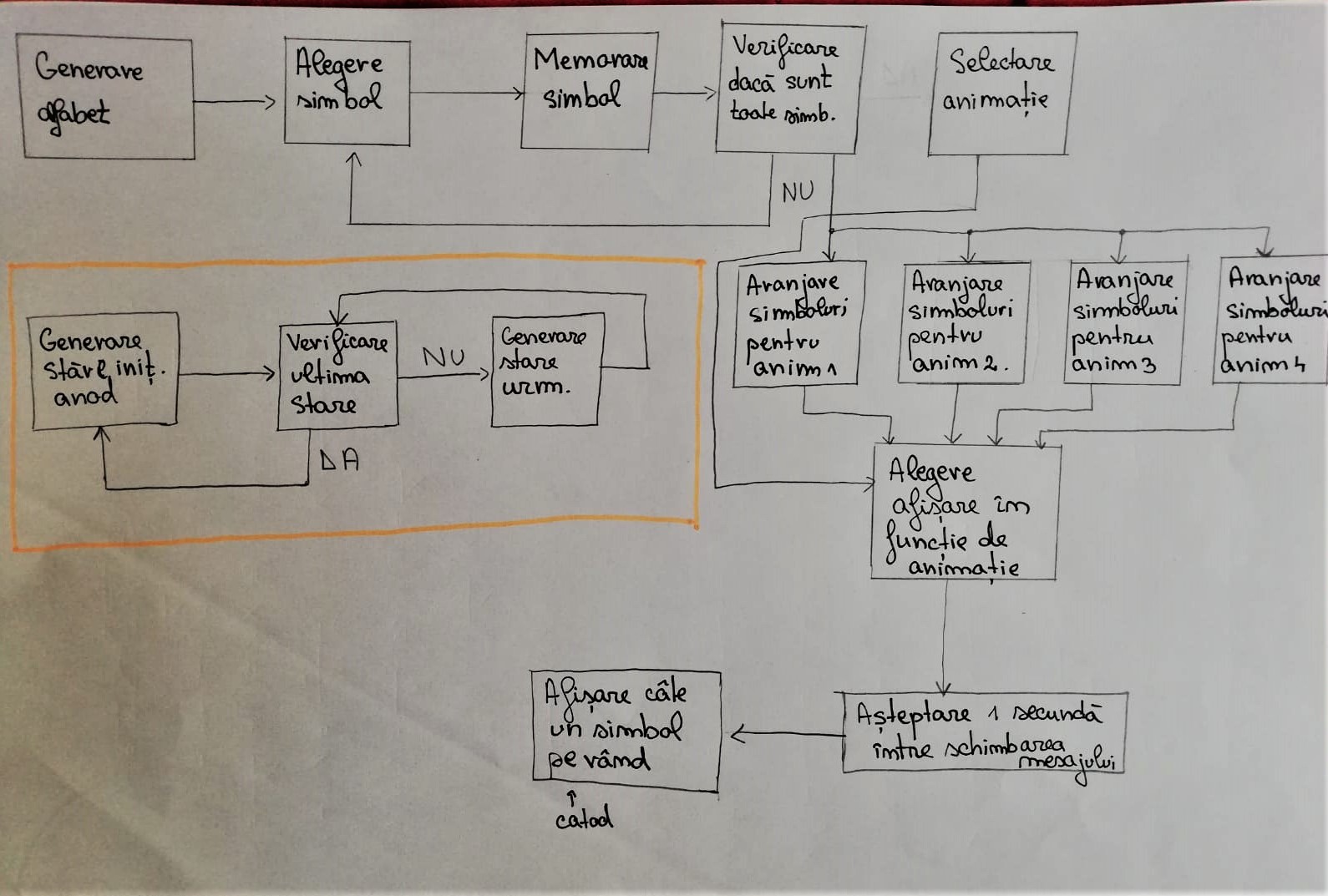
Să se proiecteze o reclamă publicitară cu animații multiple. Se vor folosi cele 4 afișaje cu 7 segmente de pe plăcuțele Digilent Nexys-2 și Starter. Textul de afișat va fi hard-codat (în momentul sintezei proiectuluise știe textul reclamei, care va fi încorporat în arhitectura hardwer). Reclama va avea mai multe regimuri de funcționare (minimum 4) ce vor putea fi selectate de către utilizator, de la comutatoarele plăcuței FPGA. Se vor folosi oscilatorul de cuarț încorporat în plăcuța FPGA (semnalul de clock va trebui sa fie desigur divizat). În ceea ce privește regimurile de funcționare, se vor folosi ca sursă de inspirație PPT sau reclamele existente (vezi farmacii). Exemple de regimuri de funcționare: -curgerea- scrisului de la dreapta spre stânga, pâlpâire, afișaj litera cu litera sau simbol cu simbol).

Deoarece pe un afișaj cu 7 segmente nu se pot reprezenta toate literele, se va crea un alfabet maximal și mesajele vorfi compuse din simbolurile aceluia.

Animațiile implementate vor fi:

1. Pâlpâire text – comutatoarele fiind ”00”
2. Curgere text de la stânga la dreapta – comutatoarele fiind ” 01”
3. Afișare cate o literă pe rând – comutatoarele fiind ” 10”
4. Adăugare cate o literă pe rând – comutatoarele fiind ” 11”

2. ***Schema bloc cu componente***

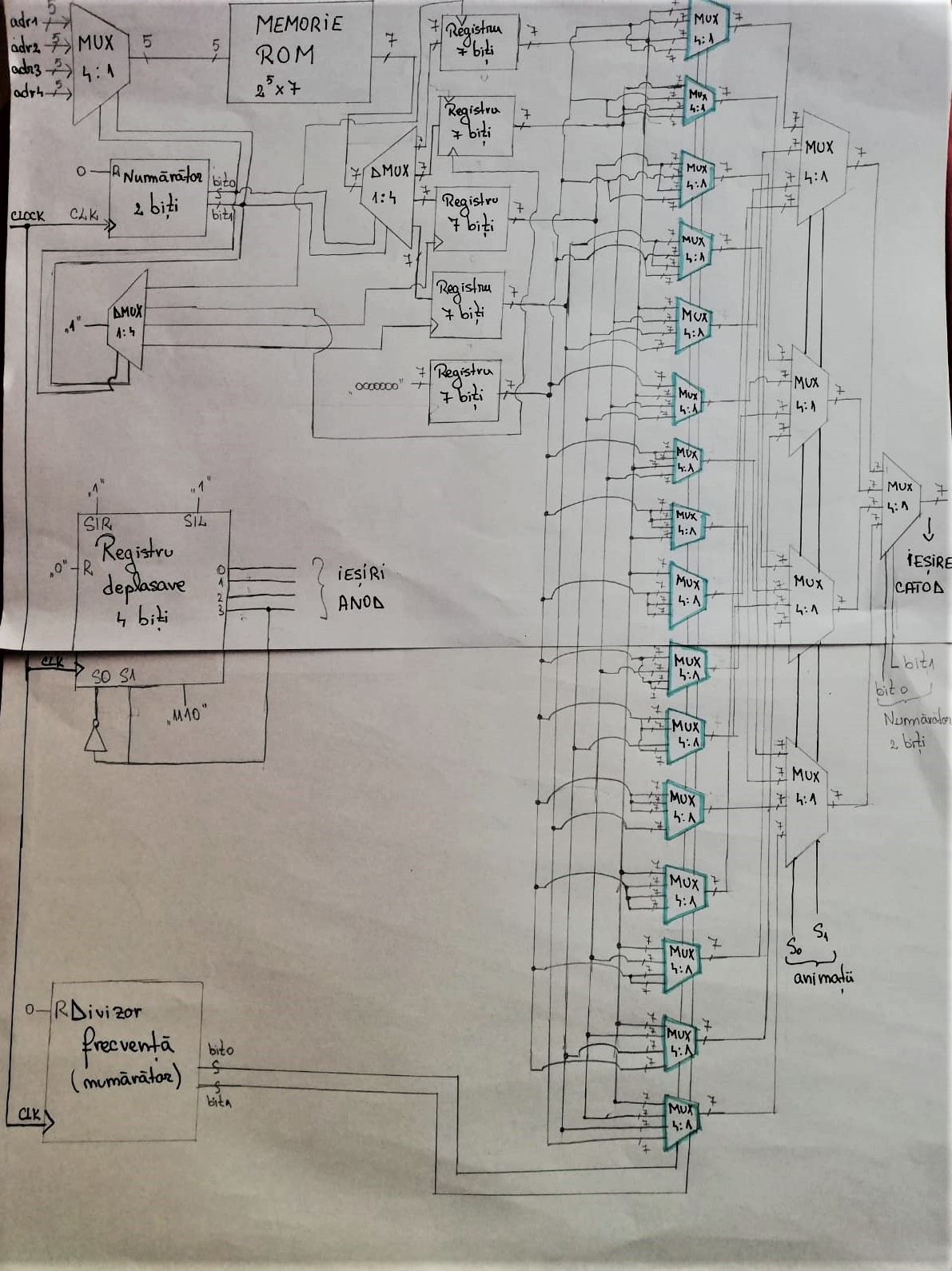
******

Vom genera noul alfabet cu 26 de simboluri care vor corespunde celor 26 de litere din alfabet ( ne vom baza pe alfabetul englez- fără ”ă, î, â, ș, ț”) și le vom păstra în memorie pentru a putea fi accesate. Apoi vom alege pe rând câte un simbol din alfabet, si îl vom memora. După ce vom avea toate cele 4 simboluri necesare vom crea o ordine de afișare a acestora pentru a realiza astfel animația dorită. Apoi în funcție de animația selectată vom afișa pe rând mesajul dorit (în modul în care a fost creat pentru animația respectivă), așteptând o secundă intre schimbarea mesajului.

De asemenea pentru anoduri vom genera cele 4 stări necesare pornind de la o stare inițială, apoi generând următoarele stări corespunzătoare. Vom verfica dacă nu s-a ajuns in ultima stare posibilă,iar in caz afirmativ vom relua procesul de la starea inițială.

3. ***Proiectare şi implementare***

* *Schemă de detaliu*

**

* *Proiectare componente*

Proiectul este alcătuit din mai multe componente care au fost legate împreună intr-o componentă principală, pentru a putea realiza cele 4 animații. In funcție de ce animație este aleasa de către utilizator, simbolurile corespunzătoare vor fi afișate. Componentele folosite sunt:

1. *Memorie ROM 2^5x7*

* Aici am reținut codul pentru cele 27 de simboluri ale alfabetului realizat pentru a putea fi afișat

-------Memoria ROM 2^5x7----

entity ROMmem is

port (

addr : in bit\_vector(4 downto 0); --are ca intrare o adresa pe 5 biti prin care se acceseaza

--literele; fiecarei litere ii corespunde o adresa particulara

Mem : out bit\_vector (6 downto 0) ); --iesirea reprezinta codul specific fiecarui simbol din alfabet

end ROMmem;

architecture behave of ROMmem is --Am declarant codul corespunzator simbolurilor ca si constante

constant data0 : bit\_vector (6 downto 0) :="0001000"; --A

constant data1 : bit\_vector (6 downto 0) :="1100000"; --B

constant data2 : bit\_vector (6 downto 0) :="0110001"; --C

constant data3 : bit\_vector (6 downto 0) :="1000010"; --D

constant data4 : bit\_vector (6 downto 0) :="0110000"; --E

constant data5 : bit\_vector (6 downto 0) :="0111000"; --F

constant data6 : bit\_vector (6 downto 0) :="0100000"; --G

constant data7 : bit\_vector (6 downto 0) :="1001000"; --H

constant data8 : bit\_vector (6 downto 0) :="1001111"; --I

constant data9 : bit\_vector (6 downto 0) :="1000011"; --J

constant data10 : bit\_vector (6 downto 0) :="1011000"; --K

constant data11 : bit\_vector (6 downto 0) :="1110001"; --L

constant data12 : bit\_vector (6 downto 0) :="1110001"; --M

constant data13 : bit\_vector (6 downto 0) :="1101000"; --N

constant data14 : bit\_vector (6 downto 0) :="0000001"; --O

constant data15 : bit\_vector (6 downto 0) :="0011000"; --P

constant data16 : bit\_vector (6 downto 0) :="0010000"; --Q

constant data17 : bit\_vector (6 downto 0) :="1101110"; --R

constant data18 : bit\_vector (6 downto 0) :="0100100"; --S

constant data19 : bit\_vector (6 downto 0) :="0001111"; --T

constant data20 : bit\_vector (6 downto 0) :="1000001"; --U

constant data21 : bit\_vector (6 downto 0) :="1100011"; --V

constant data22 : bit\_vector (6 downto 0) :="1000000"; --W

constant data23 : bit\_vector (6 downto 0) :="1101100"; --X

constant data24 : bit\_vector (6 downto 0) :="1001100"; --Y

constant data25 : bit\_vector (6 downto 0) :="0010010"; --Z

type rom\_array is array(0 to 25) of bit\_vector(6 downto 0);

constant rom : rom\_array := ( data0,data1,data2,data3,data4,data5,data6,data7,data8,data9,data10,data11,data12,data13,data14,data15,

data16,data17,data18,data19,data20,data21,data22,data23,data24,data25);

begin

process(addr)

begin

case addr is

when "00000" => Mem<=rom(0);

when "00001" => Mem<=rom(1);

when "00010" => Mem<=rom(2);

when "00011" => Mem<=rom(3);

when "00100" => Mem<=rom(4);

when "00101" => Mem<=rom(5);

when "00110" => Mem<=rom(6);

when "00111" => Mem<=rom(7);

when "01000" => Mem<=rom(8);

when "01001" => Mem<=rom(9);

when "01010" => Mem<=rom(10);

when "01011" => Mem<=rom(11);

when "01100" => Mem<=rom(12);

when "01101" => Mem<=rom(13);

when "01110" => Mem<=rom(14);

when "01111" => Mem<=rom(15);

when "10000" => Mem<=rom(16);

when "10001" => Mem<=rom(17);

when "10010" => Mem<=rom(18);

when "10011" => Mem<=rom(19);

when "10100" => Mem<=rom(20);

when "10101" => Mem<=rom(21);

when "10110" => Mem<=rom(22);

when "10111" => Mem<=rom(23);

when "11000" => Mem<=rom(24);

when "11001" => Mem<=rom(25);

when others => NULL;

end case;

end process;

end behave;

constant data12 : bit\_vector (6 downto 0) :="1110001"; --M

constant data13 : bit\_vector (6 downto 0) :="1101000"; --N

constant data14 : bit\_vector (6 downto 0) :="0000001"; --O

constant data15 : bit\_vector (6 downto 0) :="0011000"; --P

constant data16 : bit\_vector (6 downto 0) :="0010000"; --Q

constant data17 : bit\_vector (6 downto 0) :="1101110"; --R

constant data18 : bit\_vector (6 downto 0) :="0100100"; --S

constant data19 : bit\_vector (6 downto 0) :="0001111"; --T

constant data20 : bit\_vector (6 downto 0) :="1000001"; --U

constant data21 : bit\_vector (6 downto 0) :="1100011"; --V

constant data22 : bit\_vector (6 downto 0) :="1000000"; --W

constant data23 : bit\_vector (6 downto 0) :="1101100"; --X

constant data24 : bit\_vector (6 downto 0) :="1001100"; --Y

constant data25 : bit\_vector (6 downto 0) :="0010010"; --Z

type rom\_array is array(0 to 25) of bit\_vector(6 downto 0);

constant rom : rom\_array := ( data0, data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, data11, data12, data13 ,data14, data15, data17, data18, data19, data21, data22, data23, data24, data25); -- am adaugat toate simbolurile intr o constanta de tip “rom\_array” (un tip nou definit -- --de noi)

begin

process(addr) -- in functie de adresa cu care este accesata memoria,aceasta va returna codul simbolului

begin -- corespunzator adresei

case addr is

when "00000" => Mem<=rom(0);

when "00001" => Mem<=rom(1);

when "00010" => Mem<=rom(2);

when "00011" => Mem<=rom(3);

when "00100" => Mem<=rom(4);

when "00101" => Mem<=rom(5);

when "00110" => Mem<=rom(6);

when "00111" => Mem<=rom(7);

when "01000" => Mem<=rom(8);

when "01001" => Mem<=rom(9);

when "01010" => Mem<=rom(10);

when "01011" => Mem<=rom(11);

when "01100" => Mem<=rom(12);

when "01101" => Mem<=rom(13);

when "01110" => Mem<=rom(14);

when "01111" => Mem<=rom(15);

when "10000" => Mem<=rom(16);

when "10001" => Mem<=rom(17);

when "10010" => Mem<=rom(18);

when "10011" => Mem<=rom(19);

when "10100" => Mem<=rom(20);

when "10101" => Mem<=rom(21);

when "10110" => Mem<=rom(22);

when "10111" => Mem<=rom(23);

when "11000" => Mem<=rom(24);

when "11001" => Mem<=rom(25);

when others => NULL;

end case;

end process;

end behave;

when "01100" => Mem<=rom(12);

when "01101" => Mem<=rom(13);

when "01110" => Mem<=rom(14);

when "01111" => Mem<=rom(15);

when "10000" => Mem<=rom(16);

when "10001" => Mem<=rom(17);

when "10010" => Mem<=rom(18);

when "10011" => Mem<=rom(19);

when "10100" => Mem<=rom(20);

when "10101" => Mem<=rom(21);

when "10110" => Mem<=rom(22);

when "10111" => Mem<=rom(23);

when "11000" => Mem<=rom(24);

when "11001" => Mem<=rom(25);

when others => NULL;

end case;

end process;

end behave;

* Reprezentarea se face ținând cont de faptul că **ledurile** segmentelor sunt **active pe 0 logic**. Utilizatorul va putea alege textul dorit făcând modificări in codul **vhdl** care va fi prezentat la sfârșitul documentației. Modificările necesare sunt schimbarea adreselor(adr1,adr2,adr3,adr4) prin care se acceseza memoria din codul vhdl

1. *Divizorul de frecvență*

Divizorul de frecvență

entity Clock\_divider is

port (

Clk : in BIT; --Clock-ul plăcii

Reset : in BIT;

sout : out Bit\_vector(1 downto 0) --iesirile care vor determina schimbarea simbolurilor de pe afisor

);

end Clock\_divider;

architecture behave of Clock\_divider is

begin

process

variable a:integer:=0; -- o variabila prin intermediul careia vom numara timp de o secunda

variable b:bit\_vector (1 downto 0):="00"; -- o variabila in care retinem starea iesirilor pentru a le putea

--modifica in ordine

begin

if(Reset='1') then

a:=0; b:="00";

elsif clk='1' and clk'event then

a:=a+1;

if a=10 and b="00" then

b:="01"; a:=0;

elsif a=10 and b="01" then

b:="10"; a:=0;

elsif a=10 and b="10" then

b:="11"; a:=0;

elsif a=10 and b="11" then

b:="00"; a:=0;

end if;

end if;

sout <=b;

wait on clk;

end process;

end behave;

* Rolul divizorului de frecventa este unul foarte important deoarece fără el utilizatorul n-ar avea posibilitatea de a vedea animația respectivă. Clock-ul plăcii **NEXYS-3** funcționează pe **100MHz**, de aceea avem nevoie de o modalitate de a incetini timpul de execuție. Acest divizor funcționeaza ca un numărător care va număra până la o secunda astfel încât utilizatorul să poată observa cu ochiul liber schimbările efectuate pe afișoarele folosite. Deoarece  O frecvență de 1 Hz corespunde unei perioade de repetare de o secundă ,iar Clock-ul plăcii functioneaza la o frecventa de 100 MHZ ( 100 x 106 Hz ) numărătorul nostru va număra până la **100.000.000 ,** după care iși va schimba ieșirile(ieșirile divizorului de frecvență vor determina schimbarea simbolurilor mesajului de pe afișor) si va reseta divizorul.

1. *Multiplexor 4:1 pe 5 biți*

entity mux\_4to1 is

port(

A,B,C,D : in bit\_vector (4 downto 0);

S0,S1: in bit;

Z: out bit\_vector (4 downto 0)

);

end mux\_4to1;

architecture bhv of mux\_4to1 is

begin

Z(0) <=( A(0) and (not S0) and (not S1)) or ( B(0) and S0 and (not S1)) or (C(0) and (not S0) and S1) or ( D(0) and s0 and s1);

Z(1) <=( A(1) and (not S0) and (not S1)) or ( B(1) and S0 and (not S1)) or (C(1) and (not S0) and S1) or ( D(1) and s0 and s1);

Z(2) <=( A(2) and (not S0) and (not S1)) or ( B(2) and S0 and (not S1)) or (C(2) and (not S0) and S1) or ( D(2) and s0 and s1);

Z(3) <=( A(3) and (not S0) and (not S1)) or ( B(3) and S0 and (not S1)) or (C(3) and (not S0) and S1) or ( D(3) and s0 and s1);

Z(4) <=( A(4) and (not S0) and (not S1)) or ( B(4) and S0 and (not S1)) or (C(4) and (not S0) and S1) or ( D(4) and s0 and s1);

end bhv;

* Vom folosi acest multiplexor pe 5 biți pentru a alege pe rând din cele 4 adrese (adr1,adr2,adr3,adr4) prin intermediul căreia vom accesa memoria ROM. Fiecare adresă corespunde unui anumit simbol din alfabetul creat.

1. *Numărător pe 2 biți*

entity Counter2 is

port(

Clk : in BIT;

CLR : in BIT;

Q : out bit\_vector (1 downto 0) );

end Counter2;

architecture behave of Counter2 is

Signal b : bit\_vector (1 downto 0);

begin

process(Clk)

variable a : integer:=1;

begin

if Clk'event and Clk='1' then

a:= a+1;

if a=1 then

b <="00";

elsif a=2 then

b <="01";

elsif a=3 then

b<="10";

elsif a=4 then

b<="11"; a:=0;

end if;

end if;

Q <= b;

end process;

end behave;

* Ieșirile numaratorului vor reprezenta selecțiile multiplexorului care va alege adresa pentru memoria ROM , dar si selecțiile demultiplexoarele care vor decide in care registru va fi memorat codul simbolurilor.

1. *Demultiplexor 1:4 pe un bit*

entity DMUX\_anim is

port(

X : in BIT;

s0 : in BIT;

s1 : in BIT;

y1 : out BIT;

y2 : out BIT;

y3 : out BIT;

y4 : out BIT

);

end DMUX\_anim;

architecture behave of DMUX\_anim is

begin

y4 <= X and s0 and s1;

y3 <= X and s1 and (not s0);

y2 <= X and (not s1) and s0;

y1 <= X and (not s1) and (not s0);

end behave;

* Acest demultiplexor are ca intrare ”1” logic , iar ca selecții iesirile de la numărătorul pe 2 biti , astfel de fiecare dată când un simbol este extras din memorie si va fi plasat intr-un registru, ieșirile acestui demultiplexor se vor comporta ca un clock pentru registru ,astfel incât simbolul să poată fii memorat.

1. *Demultiplexor 1:4 pe 7 biți*

entity DMUX1\_4 is

port(

a : in BIT\_VECTOR(6 downto 0);

s0 : in BIT;

s1 : in BIT;

y1 : out BIT\_VECTOR(6 downto 0);

y2 : out BIT\_VECTOR(6 downto 0);

y3 : out BIT\_VECTOR(6 downto 0);

y4 : out BIT\_VECTOR(6 downto 0)

);

end DMUX1\_4;

architecture behave of DMUX1\_4 is

component DMUX\_anim

port(

X : in BIT;

s0 : in BIT;

s1 : in BIT;

y1 : out BIT;

y2 : out BIT;

y3 : out BIT;

y4 : out BIT

);

end component;

begin

OUT1 : DMUX\_anim port map(a(0),s0,s1,y1(0),y2(0),y3(0),y4(0));

OUT2 : DMUX\_anim port map(a(1),s0,s1,y1(1),y2(1),y3(1),y4(1));

OUT3 : DMUX\_anim port map(a(2),s0,s1,y1(2),y2(2),y3(2),y4(2));

OUT4 : DMUX\_anim port map(a(3),s0,s1,y1(3),y2(3),y3(3),y4(3));

OUT5 : DMUX\_anim port map(a(4),s0,s1,y1(4),y2(4),y3(4),y4(4));

OUT6 : DMUX\_anim port map(a(5),s0,s1,y1(5),y2(5),y3(5),y4(5));

OUT7 : DMUX\_anim port map(a(6),s0,s1,y1(6),y2(6),y3(6),y4(6));

end behave;

1. *Registru pe 7 biți*

------ Bistabil D --------

entity Flip\_flop\_D is

port(

D : in BIT;

Clk : in BIT;

Q : out BIT

);

end Flip\_flop\_D;

architecture behave of Flip\_Flop\_D is

begin

process(Clk)

variable q\_temp : BIT;

begin

if(Clk'EVENT and Clk='1') then

q\_temp := D;

end if;

Q <= q\_temp;

end process;

end behave;

------- Mux 4:1 un bit----

entity MUX41 is

port(

i3 : in BIT;

i2 : in BIT;

i1 : in BIT;

i0 : in BIT;

s0,s1 : in BIT ;

O : out BIT

);

end MUX41;

architecture behave of MUX41 is

begin

O <= (i0 and s0 and s1) or (i1 and (not s0) and s1 ) or ( i2 and s0 and (not s1)) or ( i3 and (not s0) and (not s1));

end behave;

---------- Registru 7 biti litere ----

entity Registru7\_biti is

port(

sin : in BIT\_VECTOR(6 downto 0);

Clk : in BIT;

sout : out bit\_vector(6 downto 0)

);

end Registru7\_biti;

architecture behave of Registru7\_biti is

component Flip\_flop\_D

port(

D : in BIT;

Clk : in BIT;

Q : out bit

);

end component;

begin

D0: Flip\_flop\_D port map(sin(0),Clk,sout(0));

D1: Flip\_flop\_D port map(sin(1),Clk,sout(1));

D2: Flip\_flop\_D port map(sin(2),Clk,sout(2));

D3: Flip\_flop\_D port map(sin(3),Clk,sout(3));

D4: Flip\_flop\_D port map(sin(4),Clk,sout(4));

D5: Flip\_flop\_D port map(sin(5),Clk,sout(5));

D6: Flip\_flop\_D port map(sin(6),Clk,sout(6));

end behave;

* Vom avea 4 regiștri pe 7 biți in care vom memora cele 4 coduri ale simbolurilor care urmează sa fie afișate in funcție de animația selectată

---------- Registru 7 biti litere ----

entity Registru7\_biti is

port(

sin : in BIT\_VECTOR(6 downto 0);

Clk : in BIT;

sout : out bit\_vector(6 downto 0)

);

end Registru7\_biti;

architecture behave of Registru7\_biti is

component Flip\_flop\_D

port(

D : in BIT;

Clk : in BIT;

Q : out bit

);

end component;

begin

D0: Flip\_flop\_D port map(sin(0),Clk,sout(0));

D1: Flip\_flop\_D port map(sin(1),Clk,sout(1));

D2: Flip\_flop\_D port map(sin(2),Clk,sout(2));

D3: Flip\_flop\_D port map(sin(3),Clk,sout(3));

D4: Flip\_flop\_D port map(sin(4),Clk,sout(4));

D5: Flip\_flop\_D port map(sin(5),Clk,sout(5));

D6: Flip\_flop\_D port map(sin(6),Clk,sout(6));

end behave;

1. *Multiplexor 4:1 pe 7 biți*

---------- MUX 2:1 ------

entity MUX2\_1 is

Port( a,b : in bit\_vector(6 downto 0);

s : in bit;

y: out bit\_vector(6 downto 0) );

end MUX2\_1;

architecture Behavioral of MUX2\_1 is

begin

y(0) <= (s and b(0) ) or ( (not s ) and a(0) );

y(1) <= (s and b(1) ) or ( (not s ) and a(1) );

y(2) <= (s and b(2) ) or ( (not s ) and a(2) );

y(3) <= (s and b(3) ) or ( (not s ) and a(3) );

y(4) <= (s and b(4) ) or ( (not s ) and a(4) );

y(5) <= (s and b(5) ) or ( (not s ) and a(5) );

y(6) <= (s and b(6) ) or ( (not s ) and a(6) );

end Behavioral;

------ MUX 4:1 -----

entity MUX4\_1 is

Port( a,b,c,d : in bit\_vector(6 downto 0);

s0,s1 : in bit;

y: out bit\_vector(6 downto 0) );

end MUX4\_1;

architecture Behavioral of MUX4\_1 is

component MUX2\_1 is

Port ( a,b : in bit\_vector (6 downto 0);

s : in bit ;

y: out bit\_vector (6 downto 0));

end component;

Signal Y1,Y2 : bit\_vector (6 downto 0);

begin

test0 : MUX2\_1 port map(a,b,s0,y1);

test1 : MUX2\_1 port map(c,d,s0,y2);

test2 : MUX2\_1 port map(y1,y2,s1,y);

end Behavioral;

*Vom folosi mai multe multiplexoare 4:1 pe 7 biți pentru a selecta codul simbolurilor din alfabet in așa fel încât să putem crea animațiile dorite.*

* *Fiecărei animații ii corespund 4 multiplexoare care vor selecta simbolurile pentru animație astfel: ca primă intrare pe fiecare multiplexor vor fi simbolurile care se vor afișa prima data (pentru animația respectivă) ,pe a doua intrare de la cele 4 multiplexoare se vor afla simbolurile ce se vor afișa pentru a doua oară și tot așa.De exemplu pentru animatia 1 (pâlpâire) pe primele intrări vor fi cele 4 simboluri ce apartin cuvântului,pe următoarele intrări vom avea ”spații„ ,apoi din nou cele 4 simboluri si apoi spațiile. Selecțiile acestor multiplexoare vor fi ieșiriile de la divizorul de frecvență astfel încât aceste schimbări să se modifice la interval de o secundă.*
* *Astfel de la fiecare animație vor rezulta câte 4 ieșiri pe 7 biți,deci in total 16 ieșiri.Pe aceste ieșiri o sa le punem in alte 4 multiplexoare sub aceași idee. Astfel ieșirile ce corespund primei animații vor fi primele intrări ale noilor multiplexoare, cele de la a doua animație vor fi pe următoarele intrări si tot așa. Pentru aceste 4 multiplexoare vom avea ca selecții, intrările ce determină animația ( „00”, „01”, „10”, „11”).*
* *Ca ultim pas vom avea nevoie de încă un multiplexor care să ne aleagă pe rând litera care va fi afișată in funcție de anodul care este aprins,iar acest multiplexor iși va schimba ieșirea la fiecare impuls, la fel ca și anodurile.*

1. *Registru de deplasare pe 4 biți*

------ Bistabil D ------

ENTITY DFF IS

PORT(d, clk,CL : IN BIT;

q : OUT BIT);

END DFF;

Architecture behave OF DFF IS

begin

process(clk)

variable q\_temp : BIT;

begin

IF(CL= '1') then

q\_temp :='0';

elsif(clk'EVENT and clk='1') then

q\_temp := d;

end if;

q <= q\_temp;

end process;

end behave;

-------Registru deplasare 4 biti----------

entity Registru\_deplasare is

port(

clk : in BIT;

SL : in BIT;

SR : in BIT;

CL : in BIT;

S0 : in BIT;

S1 : in BIT;

i : in BIT\_VECTOR(3 downto 0);

q : out bit\_vector (3 downto 0)

);

end Registru\_deplasare;

architecture behave of Registru\_deplasare is

component DFF

port(

d : in BIT;

clk : in BIT;

CL : in BIT;

q : out bit

);

end component;

component MUX41

port(

i0 : in BIT;

i1 : in BIT;

i2 : in BIT;

i3 : in BIT;

s0,s1 : in BIT;

o : out BIT

);

end component;

SIGNAL o,qt: BIT\_VECTOR(3 DOWNTO 0);

begin

U1:MUX41 PORT MAP(SL, qt(2), i(3), qt(3), s0,s1, o(3));

U2:MUX41 PORT MAP(qt(3), qt(1), i(2), qt(2), s0,s1, o(2));

U3:MUX41 PORT MAP(qt(2), qt(0), i(1), qt(1), s0,s1, o(1));

U4:MUX41 PORT MAP(qt(1), SR, i(0), qt(0), s0,s1, o(0));

U5:DFF PORT MAP(o(3), clk, CL, qt(3));

U6:DFF PORT MAP(o(2), clk, CL, qt(2));

U7:DFF PORT MAP(o(1), clk, CL, qt(1));

U8:DFF PORT MAP(o(0), clk, CL, qt(0));

Q <= qt;

end behave;

);

end component;

component MUX41

port(

i0 : in BIT;

i1 : in BIT;

i2 : in BIT;

i3 : in BIT;

s0,s1 : in BIT;

o : out BIT

);

end component;

SIGNAL o,qt: BIT\_VECTOR(3 DOWNTO 0);

begin

U1:MUX41 PORT MAP(SL, qt(2), i(3), qt(3), s0,s1, o(3));

U2:MUX41 PORT MAP(qt(3), qt(1), i(2), qt(2), s0,s1, o(2));

U3:MUX41 PORT MAP(qt(2), qt(0), i(1), qt(1), s0,s1, o(1));

U4:MUX41 PORT MAP(qt(1), SR, i(0), qt(0), s0,s1, o(0));

U5:DFF PORT MAP(o(3), clk, CL, qt(3));

U6:DFF PORT MAP(o(2), clk, CL, qt(2));

U7:DFF PORT MAP(o(1), clk, CL, qt(1));

U8:DFF PORT MAP(o(0), clk, CL, qt(0));

Q <= qt;

end behave;

* Vom folosi acest registru pentru anodurile afișorului care sunt reprezentate de un vector pe 4 biți. Șirul de 4 biți care reprezintă anodul “decide” care din cele patru segmente sa fie activ (anodurile sunt active pe 0 logic). Acestea se aprind secvential, însă cu frecvența de tact corespunzătoare ( adică folosind frecvența oscilatorului de 100Mhz) vom crea iluzia că toate sunt aprinse simultan deoarece schimbarea lor nu va putea fi observată. Registrul de deplasare are 4 moduri de lucru (Resetare, Încărcare paralelă,Deplasare dreapta-stânga si stânga-dreapta).Vom incărca parallel registrul cu ”1110” ,după care vom schimba modul de lucru pe deplasare dreapta-stânga iar pe serial input vom incărca valoarea ”1” logic. Asfel, anodurile vor trece prin toate stările ( 1110, 1101, 1011, 0111 ), iar cand se ajunge la ultima stare vom schimba din nou modul delucru pe incărcare paralelă si o vom lua de la capăt.
* *Proiectare ansamblu*

Am creat entitatea pentru componenta principală cu intrările si ieșirile corespunzătoare:

”Anim”- vector pe 2 biți care prin care se alege ce animație se va afișa; Va fi legat ca și intrare la intrerupătoare

”Clock”- Clock-ul cu o frecvență de 100 Mhz, care va fi reprezentat de oscilatorul de quartz interior al plăcii

”Anod”- Vectorul de 4 biți ce va fi conectat la anodurile plăcii

”segment”- vectorul de 7 biți care va fi conectat la ledurile afișoarelor pentru a afișa mesajul dorit

*--------------- COD PROIECT ANIMATIE-----------*

*entity Animatie\_text is*

*port (*

*Anim : in bit\_vector (1 downto 0);*

*Clock : in bit;*

*Anod : out bit\_vector (3 downto 0);*

*segment : out bit\_vector (6 downto 0)*

*);*

*end Animatie\_text;*

*architecture Behavioral of Animatie\_text is*

*component MUX4\_1 is*

*Port( a,b,c,d : in bit\_vector(6 downto 0);*

*s0,s1 : in bit;*

*y: out bit\_vector(6 downto 0) );*

*end component;*

*component mux\_4to1 is*

*port(*

*A,B,C,D : in bit\_vector (4 downto 0);*

*S0,S1: in bit;*

*Z: out bit\_vector (4 downto 0) );*

*end component;*

*component DMUX1\_4 is*

*port(*

*a : in BIT\_VECTOR(6 downto 0);*

*s0 : in BIT;*

*s1 : in BIT;*

*y1 : out BIT\_VECTOR(6 downto 0);*

*y2 : out BIT\_VECTOR(6 downto 0);*

*y3 : out BIT\_VECTOR(6 downto 0);*

*y4 : out BIT\_VECTOR(6 downto 0)*

*);*

*end component;*

*component DMUX\_anim*

*port(*

*X : in BIT;*

*s0 : in BIT;*

*s1 : in BIT;*

*y1 : out BIT;*

*y2 : out BIT;*

*y3 : out BIT;*

*y4 : out BIT*

*);*

*end component;*

*component Registru7\_biti is*

*port(*

*sin : in BIT\_VECTOR(6 downto 0);*

*Clk : in BIT;*

*sout : out bit\_vector(6 downto 0)*

*);*

*end component;*

*component Registru\_deplasare is*

*port(*

*clk : in BIT;*

*SL : in BIT;*

*SR : in BIT;*

*CL : in BIT;*

*S0 : in BIT;*

*S1 : in BIT;*

*i : in BIT\_VECTOR(3 downto 0);*

*q : out bit\_vector (3 downto 0)*

*);*

*end component;*

*component Counter2 is*

*port(*

*Clk : in BIT;*

*CLR : in BIT;*

*Q : out bit\_vector (1 downto 0)*

*);*

*end component;*

*component Clock\_divider is*

*port (*

*Clk : in BIT;*

*Reset : in BIT;*

*sout : out Bit\_vector(1 downto 0)*

*);*

*end Component;*

*component ROMmem is*

*port (*

*addr : in bit\_vector(4 downto 0);*

*Mem : out bit\_vector (6 downto 0)*

*);*

*end component;*

*constant adr1 : bit\_vector (4 downto 0) :="10100"; --U*

*constant adr2 : bit\_vector (4 downto 0) :="10011"; --T*

*constant adr3 : bit\_vector (4 downto 0) :="00010"; --C*

*constant adr4 : bit\_vector (4 downto 0) :="01101"; --N*

*Signal empty : bit\_vector (6 downto 0) :="0000000"; --- gol*

*Signal ADR\_c : bit\_vector (4 downto 0);*

*SIGNAL LITERA,Dm\_A,Dm\_B,Dm\_C,Dm\_D,A,B,C,D,Lit1,Lit2,Lit3,Lit4 : bit\_vector (6 downto 0);*

*SIGNAL S,Q ,Sel\_adr : bit\_vector (1 downto 0) ;*

*SIGNAL A1\_1,A1\_2,A1\_3,A1\_4,A2\_1,A2\_2,A2\_3,A2\_4,A3\_1,A3\_2,A3\_3,A3\_4,A4\_1,A4\_2,A4\_3,A4\_4 : bit\_vector (6 downto 0);*

*SIGNAL Enable,anod\_out : bit\_vector (3 downto 0);*

*SIGNAL H,L,E,C\_A,C\_B,C\_C,C\_D,Nex,S0,S1 : bit ;*

*begin*

*Adr : mux\_4to1 port map(adr1,adr2,adr3,adr4,Sel\_adr(0),Sel\_adr(1),ADR\_c); --Se alege adresa folosind un mux cu selectiile de la numarator*

*Num : Counter2 port map(Clock,'0',Sel\_adr); --Numaratorul care va da selectiile*

*Mem : ROMmem port map(ADR\_c,Litera); --Se elege litera din memorie*

*Dmux : DMUX1\_4 port map(Litera,Sel\_adr(0),Sel\_adr(1),Dm\_A,Dm\_B,Dm\_C,Dm\_D); --Se alege registrul in care se pastreaza*

*Dmux\_clk : DMUX\_anim port map('1',Sel\_adr(0),Sel\_adr(1),C\_A,C\_B,C\_C,C\_D); --Un dmux care v-a avea ca iesiri clock-ul pentru registre*

*REG1 : Registru7\_biti port map(Dm\_A,C\_A,A);*

*REG2 : Registru7\_biti port map(Dm\_B,C\_B,B);*

*REG3 : Registru7\_biti port map(Dm\_C,C\_C,C);*

*REG4 : Registru7\_biti port map(Dm\_D,C\_D,D);*

*---Prima anim*

*Mux1\_1 : MUX4\_1 port map(A,empty,A,empty,S(0),S(1),A1\_1);*

*Mux2\_1 : MUX4\_1 port map(B,empty,B,empty,S(0),S(1),A1\_2);*

*Mux3\_1 : MUX4\_1 port map(C,empty,C,empty,S(0),S(1),A1\_3);*

*Mux4A2 : MUX4\_1 port map(D,empty,D,empty,S(0),S(1),A1\_4);*

*---A doua anim*

*Mux1A2 : MUX4\_1 port map(A,empty,empty,empty,S(0),S(1),A2\_1);*

*Mux2\_2 : MUX4\_1 port map(B,A,empty,empty,S(0),S(1),A2\_2);*

*Mux3\_2 : MUX4\_1 port map(C,B,A,empty,S(0),S(1),A2\_3);*

*Mux4\_2 : MUX4\_1 port map(D,C,B,A,S(0),S(1),A2\_4);*

*---A treia anim*

*--Mux1\_3 : MUX4\_1 port map(A,empty,empty,empty,S(0),S(1),A3\_1); --Putem folosi A2\_1*

*Mux2\_3 : MUX4\_1 port map(empty,B,empty,empty,S(0),S(1),A3\_2);*

*Mux3\_3 : MUX4\_1 port map(empty,empty,C,empty,S(0),S(1),A3\_3);*

*Mux4\_3 : MUX4\_1 port map(empty,empty,empty,D,S(0),S(1),A3\_4);*

*---A patra anim*

*Mux1\_4 : MUX4\_1 port map(A,A,A,A,S(0),S(1),A4\_1);*

*Mux2\_4 : MUX4\_1 port map(empty,B,B,B,S(0),S(1),A4\_2);*

*Mux3\_4 : MUX4\_1 port map(empty,empty,C,C,S(0),S(1),A4\_3);*

*--Mux4\_4 : MUX4\_1 port map(empty,empty,empty,D,S(0),S(1),A4\_4); --Putem folosim A3\_4*

*---Divizor de frecv*

*Divider : Clock\_divider port map(Clock,'0',S);*

*--alegere anod*

*AN : Registru\_deplasare port map(Clock,'1','1','0',S0,S1,"1110",anod\_out);*

*S0 <= not anod\_out(3);*

*S1 <= not S0;*

*Anod <= anod\_out;*

*--Alegere animatii*

*Mux\_an1 : MUX4\_1 port map(A1\_1,A2\_1,A2\_1,A4\_1,Anim(0),Anim(1),Lit1);*

*Mux\_an2 : MUX4\_1 port map(A1\_2,A2\_2,A3\_2,A4\_2,Anim(0),Anim(1),Lit2);*

*Mux\_an3 : MUX4\_1 port map(A1\_3,A2\_3,A3\_3,A4\_3,Anim(0),Anim(1),Lit3);*

*Mux\_an4 : MUX4\_1 port map(A1\_4,A2\_4,A3\_4,A3\_4,Anim(0),Anim(1),Lit4);*

*--Afisare*

*Mux\_afis : MUX4\_1 port map(Lit1,Lit2,Lit3,Lit4,Sel\_adr(0),Sel\_adr(1),segment);*

*end Behavioral;*

Apoi am continuat prin a declara toate componentele pe care le-am văzut mai sus, pentru a putea fi folosite in componenta principală

*component DMUX\_anim*

*port(*

*X : in BIT;*

*s0 : in BIT;*

*s1 : in BIT;*

*y1 : out BIT;*

*y2 : out BIT;*

*y3 : out BIT;*

*y4 : out BIT*

*);*

*end component;*

*component Registru7\_biti is*

*port(*

*sin : in BIT\_VECTOR(6 downto 0);*

*Clk : in BIT;*

*sout : out bit\_vector(6 downto 0)*

*);*

*end component;*

*component Registru\_deplasare is*

*port(*

*clk : in BIT;*

*SL : in BIT;*

*SR : in BIT;*

*CL : in BIT;*

*S0 : in BIT;*

*S1 : in BIT;*

*i : in BIT\_VECTOR(3 downto 0);*

*q : out bit\_vector (3 downto 0)*

*);*

*end component;*

*component Counter2 is*

*port(*

*Clk : in BIT;*

*CLR : in BIT;*

*Q : out bit\_vector (1 downto 0)*

*);*

*end component;*

*component Clock\_divider is*

*port (*

*Clk : in BIT;*

*Reset : in BIT;*

*sout : out Bit\_vector(1 downto 0)*

*);*

*end Component;*

*component ROMmem is*

*port (*

*addr : in bit\_vector(4 downto 0);*

*Mem : out bit\_vector (6 downto 0)*

*);*

*end component;*

Apoi am declarat semnalele si constantele necesare

*component ROMmem is*

*port (*

*addr : in bit\_vector(4 downto 0);*

*Mem : out bit\_vector (6 downto 0)*

*);*

*end component;*

Cele 4 adrese care corespund literelor pe care dorim sa le afișăm

Un semnal care va reprezenta „spațiul” ,și pe care il vom folosi cand dorim să nu afișăm nimic

*constant adr1 : bit\_vector (4 downto 0) :="10100"; --U*

*constant adr2 : bit\_vector (4 downto 0) :="10011"; --T*

*constant adr3 : bit\_vector (4 downto 0) :="00010"; --C*

*constant adr4 : bit\_vector (4 downto 0) :="01101"; --N*

*Signal empty : bit\_vector (6 downto 0) :="0000000"; --- gol*

*Signal ADR\_c : bit\_vector (4 downto 0);*

*SIGNAL LITERA, Dm\_A, Dm\_B, Dm\_C, Dm\_D, A, B, C, D, Lit1, Lit2, Lit3, Lit4 : bit\_vector (6 downto 0);*

*SIGNAL S, Q , Sel\_adr : bit\_vector (1 downto 0) ;*

*SIGNAL A1\_1, A1\_2, A1\_3, A1\_4, A2\_1, A2\_2, A2\_3, A2\_4, A3\_1, A3\_2, A3\_3, A3\_4, A4\_1, A4\_2, A4\_3, A4\_4 : bit\_vector (6 downto 0);*

*SIGNAL Enable,anod\_out : bit\_vector (3 downto 0);*

*SIGNAL H, L, E, C\_A, C\_B, C\_C, C\_D, Nex, S0, S1: bit ;*

**d)**

**b)**

**a)**

*begin*

*Adr : mux\_4to1 port map(adr1,adr2,adr3,adr4,Sel\_adr(0),Sel\_adr(1),ADR\_c); --Se alege adresa folosind un multiplexor cu selectiile de la numarator*

*Num : Counter2 port map(Clock,'0',Sel\_adr); --Numaratorul care va da selectiile*

*Mem : ROMmem port map(ADR\_c,Litera); --Se elege litera din memorie*

*Dmux : DMUX1\_4 port map(Litera,Sel\_adr(0),Sel\_adr(1),Dm\_A,Dm\_B,Dm\_C,Dm\_D); --Se alege registrul in care se pastreaza*

*Dmux\_clk : DMUX\_anim port map('1',Sel\_adr(0),Sel\_adr(1),C\_A,C\_B,C\_C,C\_D); --Un dmux care v-a avea ca iesiri clock-ul pentru registre*

*REG1 : Registru7\_biti port map(Dm\_A,C\_A,A);*

*REG2 : Registru7\_biti port map(Dm\_B,C\_B,B);*

*REG3 : Registru7\_biti port map(Dm\_C,C\_C,C);*

*REG4 : Registru7\_biti port map(Dm\_D,C\_D,D);*

*---Prima anim*

*Mux1\_1 : MUX4\_1 port map(A,empty,A,empty,S(0),S(1),A1\_1);*

*Mux2\_1 : MUX4\_1 port map(B,empty,B,empty,S(0),S(1),A1\_2);*

*Mux3\_1 : MUX4\_1 port map(C,empty,C,empty,S(0),S(1),A1\_3);*

*Mux4A2 : MUX4\_1 port map(D,empty,D,empty,S(0),S(1),A1\_4);*

*---A doua anim*

*Mux1A2 : MUX4\_1 port map(A,empty,empty,empty,S(0),S(1),A2\_1);*

*Mux2\_2 : MUX4\_1 port map(B,A,empty,empty,S(0),S(1),A2\_2);*

*Mux3\_2 : MUX4\_1 port map(C,B,A,empty,S(0),S(1),A2\_3);*

*Mux4\_2 : MUX4\_1 port map(D,C,B,A,S(0),S(1),A2\_4);*

*---A treia anim*

*--Mux1\_3 : MUX4\_1 port map(A,empty,empty,empty,S(0),S(1),A3\_1); --Putem folosi A2\_1*

*Mux2\_3 : MUX4\_1 port map(empty,B,empty,empty,S(0),S(1),A3\_2);*

*Mux3\_3 : MUX4\_1 port map(empty,empty,C,empty,S(0),S(1),A3\_3);*

*Mux4\_3 : MUX4\_1 port map(empty,empty,empty,D,S(0),S(1),A3\_4);*

*---A patra anim*

*Mux1\_4 : MUX4\_1 port map(A,A,A,A,S(0),S(1),A4\_1);*

*Mux2\_4 : MUX4\_1 port map(empty,B,B,B,S(0),S(1),A4\_2);*

*Mux3\_4 : MUX4\_1 port map(empty,empty,C,C,S(0),S(1),A4\_3);*

*--Mux4\_4 : MUX4\_1 port map(empty,empty,empty,D,S(0),S(1),A4\_4); --Putem folosim A3\_4*

*---Divizor de frecv*

*Divider : Clock\_divider port map(Clock,'0',S);*

*--alegere anod*

*AN : Registru\_deplasare port map(Clock,'1','1','0',S0,S1,"1110",anod\_out);*

*S0 <= not anod\_out(3);*

*S1 <= not S0;*

*Anod <= anod\_out;*

*--Alegere animatii*

*Mux\_an1 : MUX4\_1 port map(A1\_1,A2\_1,A2\_1,A4\_1,Anim(0),Anim(1),Lit1);*

*Mux\_an2 : MUX4\_1 port map(A1\_2,A2\_2,A3\_2,A4\_2,Anim(0),Anim(1),Lit2);*

*Mux\_an3 : MUX4\_1 port map(A1\_3,A2\_3,A3\_3,A4\_3,Anim(0),Anim(1),Lit3);*

*Mux\_an4 : MUX4\_1 port map(A1\_4,A2\_4,A3\_4,A3\_4,Anim(0),Anim(1),Lit4);*

*--Afisare*

*Mux\_afis : MUX4\_1 port map(Lit1,Lit2,Lit3,Lit4,Sel\_adr(0),Sel\_adr(1),segment);*

*end Behavioral;*

1. Inițial se alege una dintre adresele adr1, adr2, adr3, adr4, care corespund simbolurilor pe care dorim să le afișăm folosind un multiplexor care are ca și selecții ieșirile numărătorului Counter2. Pentru a memora codul simbolurilor in registrele pe 7 biți folosim un demultiplexor pentru a alege registrul in care se va păstra codul și un alt demultiplexor pe 1 bit care va reprezenta ”clock-ul” registrelor. Cum si aceste demultiplexoare au ca selecții ieșirile numărătorului Counter2 in momentul in care pe una dintre ieșirile demultiplexorului de 7 biți va fi codul unui simbol,se va aprinde și ieșire corespunzătoare de la demultiplexorul pe 1 bit,astfel ralizandu-se memorarea codului in registru.

**c)**

*--Alegere animatii*

*Mux\_an1 : MUX4\_1 port map(A1\_1,A2\_1,A2\_1,A4\_1,Anim(0),Anim(1),Lit1);*

*Mux\_an2 : MUX4\_1 port map(A1\_2,A2\_2,A3\_2,A4\_2,Anim(0),Anim(1),Lit2);*

*Mux\_an3 : MUX4\_1 port map(A1\_3,A2\_3,A3\_3,A4\_3,Anim(0),Anim(1),Lit3);*

*Mux\_an4 : MUX4\_1 port map(A1\_4,A2\_4,A3\_4,A3\_4,Anim(0),Anim(1),Lit4);*

*--Afisare*

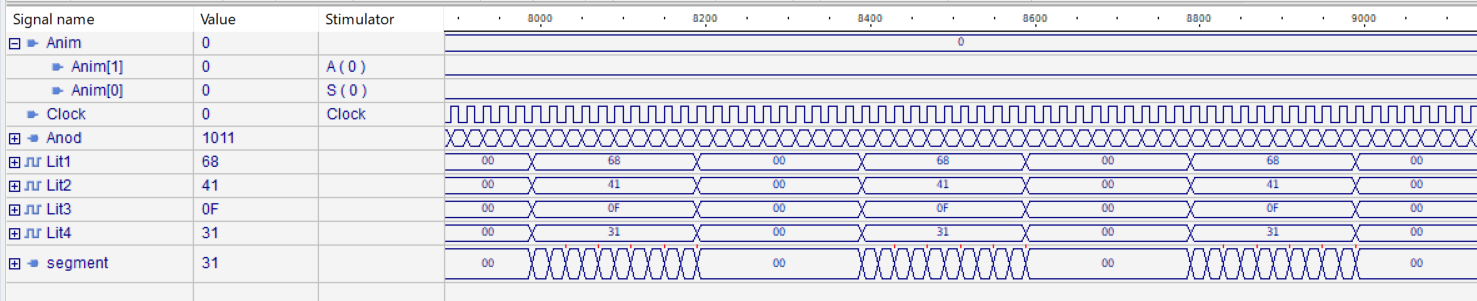
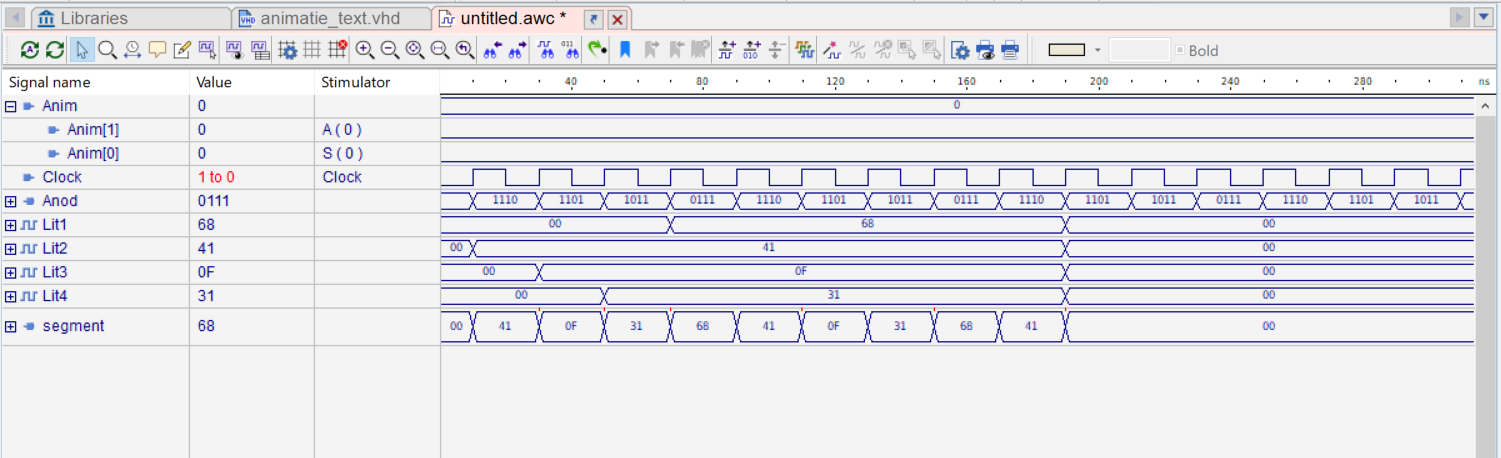
*Mux\_afis : MUX4\_1 port map(Lit1,Lit2,Lit3,Lit4,Sel\_adr(0),Sel\_adr(1),segment);*

*end Behavioral;*

1. După ce vom avea toate cele 4 coduri în registre vom folosi multiplexoarele pentru a crea animațiile dorite.După cum am spus mai sus fiecărei animații ii corespund 4 multiplexoare care au ca și selecții ieșirile divizorului de frecvență.Astfel la fiecare interval de aproximativ o secundă aceste multiplexoare iși vor schimba ieșirile astfel incât să se formeze pe afișor animația dorită.
2. Insă pentru a afișa doar animația selectată de utilizator cele 16 ieșiri ale acestor multiplexoare vor fi introduse ca intrări la alte 4 multiplexoare in aceași idee (primele intrări ale multiplexoarelor vor reprezenta mesajul primei animații etc.),dar acestea vor avea ca selecții intrările de la intrerupătoarele plăcuței (ANIM). Astfel dacă utilizator alege prima animație,adică pe intrerupătoare se va pune ”00” aceste 4 multiplexoare vor alege primele intrări. Acestea din urmă le vom mai introduce intr-un ultim multiplexor pentru a putea afișa literele pe rând prin intermediul ieșirii ”segment”. Deoarece litera afișată trebuie să se schimbe in aclași timp cu anodurile , vom lega selecțiile multiplexorului de ieșirile numărătorului pe 2 biți. Acesta folosind clock-ul plăcuței ,iși va schimab valoarea la fiecare impuls detact,deci și ieșirile multiplexorului se vor schimab la fiecare impuls de tact , și prin urmare de fiecare dată când se schimbă anodurile.
3. Ieșirea care va fi legată la anodurile afișorului este reprezentată de vectorul de 4 biți ANOD. Cum acestea sunt active pe ”0” logic și se aprind secvențial nu vom putea avea decât câte un afișor aprins pe rând. Însă folosindu-ne de oscilatorul plăcuței, care are o frecventă de 100 Mhz vom trece vectorul prin cele 4 stări posibile (1110, 1101, 1011, 0111) la frecvența respectivă si astfel vom crea iluzia că toate afișoarele sunt aprinse in același timp,deoarece schimbarea lor nu poate fi observată de utilizator datorită frecvenței. Pentru a putea trece vectorul prin cele 4 stări vom folosi un registru de deplasare pe 4 biți. Inițtial acesta va fi incărcat paralel cu ”1110”, apoi vom deplasa dreapta-stânga biții vectorului iar pe intrarea Serial Input vom avea valoarea ”1” logic.Astfel vectorul va deveni ”1101” – ”1011” – ”0111”. După ce se ajunge in uultima stare registrul va fi incărcat paralel din nou cu valoarea inițială. Pentru a modifica modul de lucru al registrului vom folosi selecțiile S1 și S0 astfel: S0 va primi valoarea negată a bitului 3 (cel mai semmnificativ) iar S1 va primi valoarea negată a lui S0. Astfel cât timp bitul 3 va fi pe 1 vom avea pe selecții ”01” ,adică modul de funcționare este deplasare dreapta-stânga; cand registrul ajunge in starea ”0111” selecțiile vor primi valoarea ”10” ,adica modul de incărcare paralel si se va relua astfel tot procesul.

* *Simulare în ActiveHDL*

Pentru a face mai vizibile schimbarile la nivelul anodurilor si catodurilor in simulari am schimbat pana la cat va Numara divizorul de frecventa,insa pentru placile de fpga acesta trebuie sa fie setat cum am exprimat mai sus

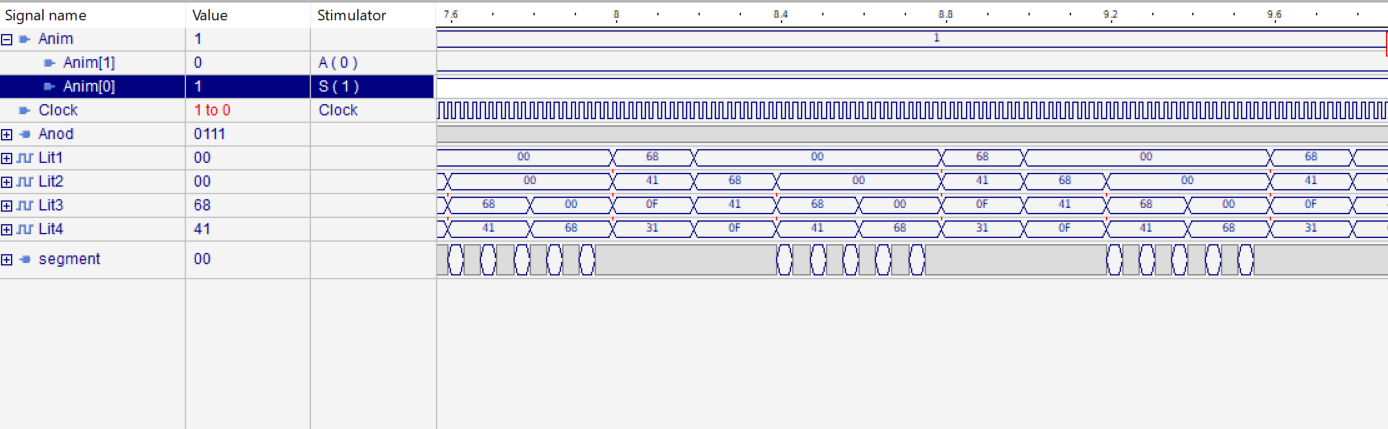
1. *Animația 1 - Pâlpâire text*

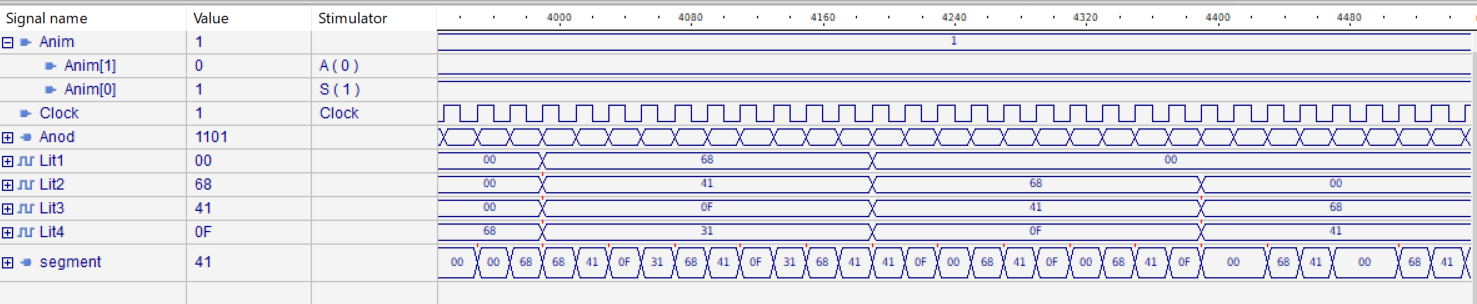
* Această animație dă impresia de pâlpâire a textului prin alternarea dintre afișarea textului propriu-zis si afișarea unor spații

U-T-C-N 🡪 \_-\_- \_- \_ 🡪 U-T-C-N 🡪 \_-\_-\_-

* Pentru a se accesa această animație, de pe intrerupătoarele plăcuței utilizatorul va da valoarea ”00” pentru intrarea Anim

1. *Animație 2 – Deplasare cuvinte de la stânga la dreapta*



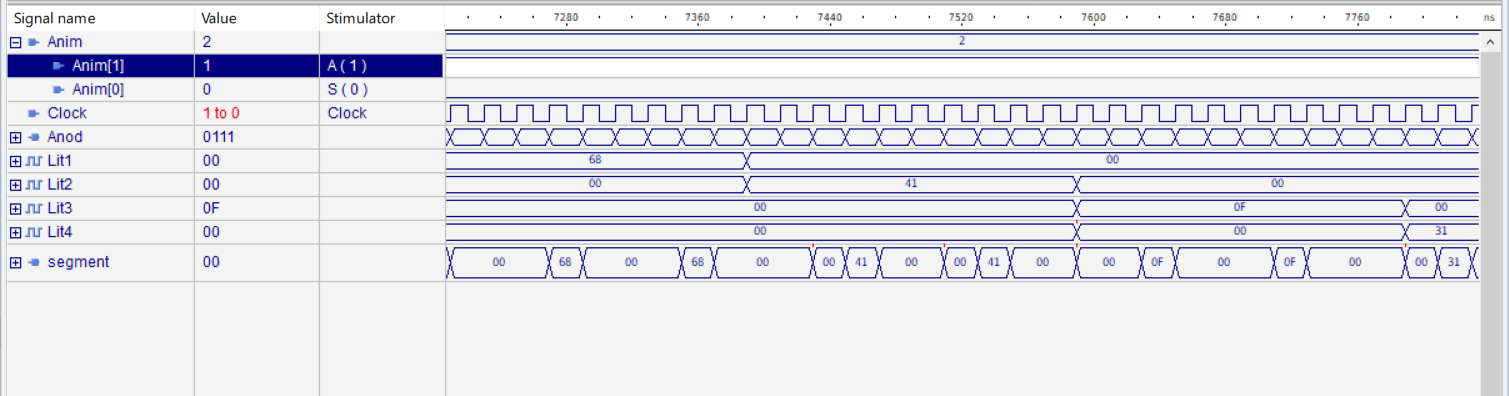
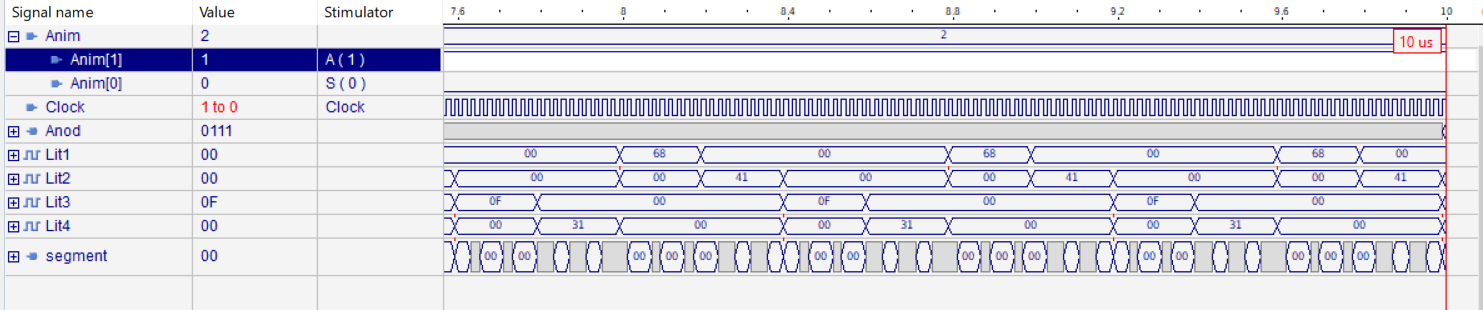
**

* Modul de ”curgere” al textului se realizeaza prin afișarea celor 4 simboluri , insă la fiecare secunda ,simbolurile vor fi mutate cu un afișor

U-T-C-N 🡪 \_-U-T-C 🡪 \_-\_-U-T 🡪 \_-\_-\_-U

* Pentru a se accesa această animație, de pe intrerupătoarele plăcuței utilizatorul va da valoarea ”01” pentru intrarea Anim

1. *Animația 3 – Afișare câte o literă pe rând*

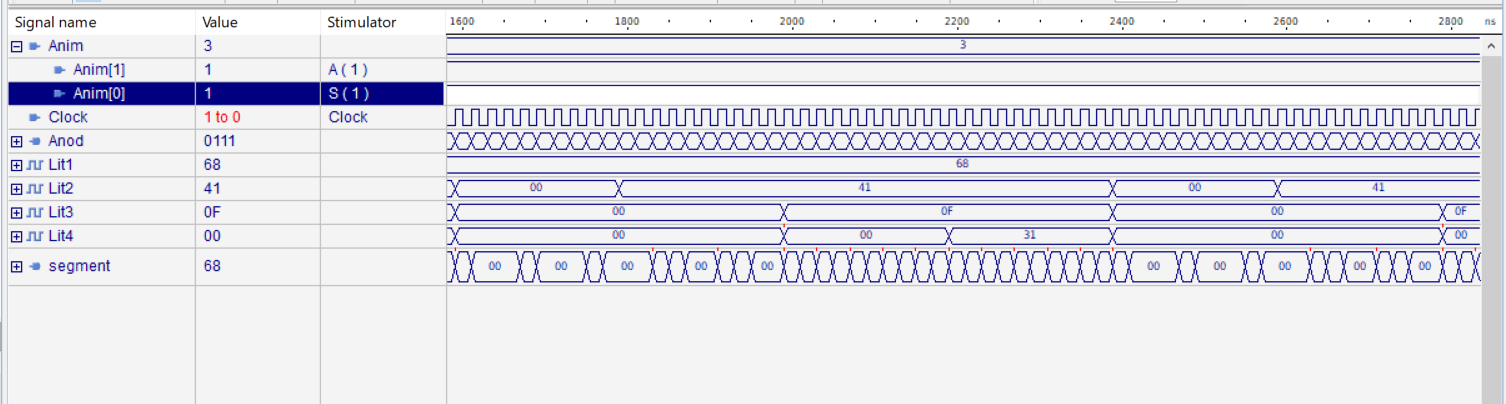
**

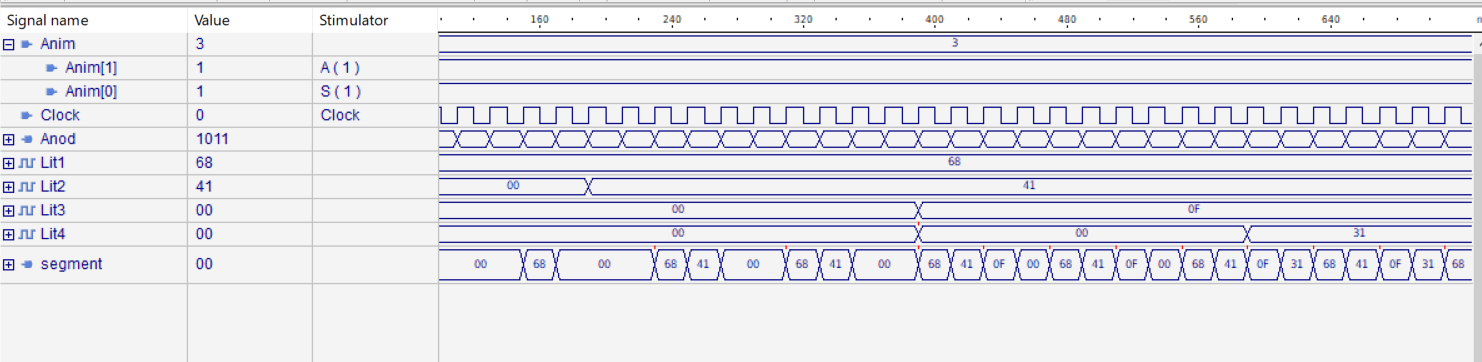
* Din nou pentru a realiza această animație am avut nevoie de 4 ”stări” ale mesajului,si anume am afișat câte o singură literă ,restul afișoarelor lăsându-le goale

*U\_-\_-\_ 🡪 \_-T-\_-\_ 🡪 \_-\_-C-\_ 🡪 \_-\_-\_-N*

* Pentru a se accesa această animație, de pe intrerupătoarele plăcuței utilizatorul va da valoarea ”10” pentru intrarea Anim

1. *Animația 4 – Adăugare câte o literă pe rând*



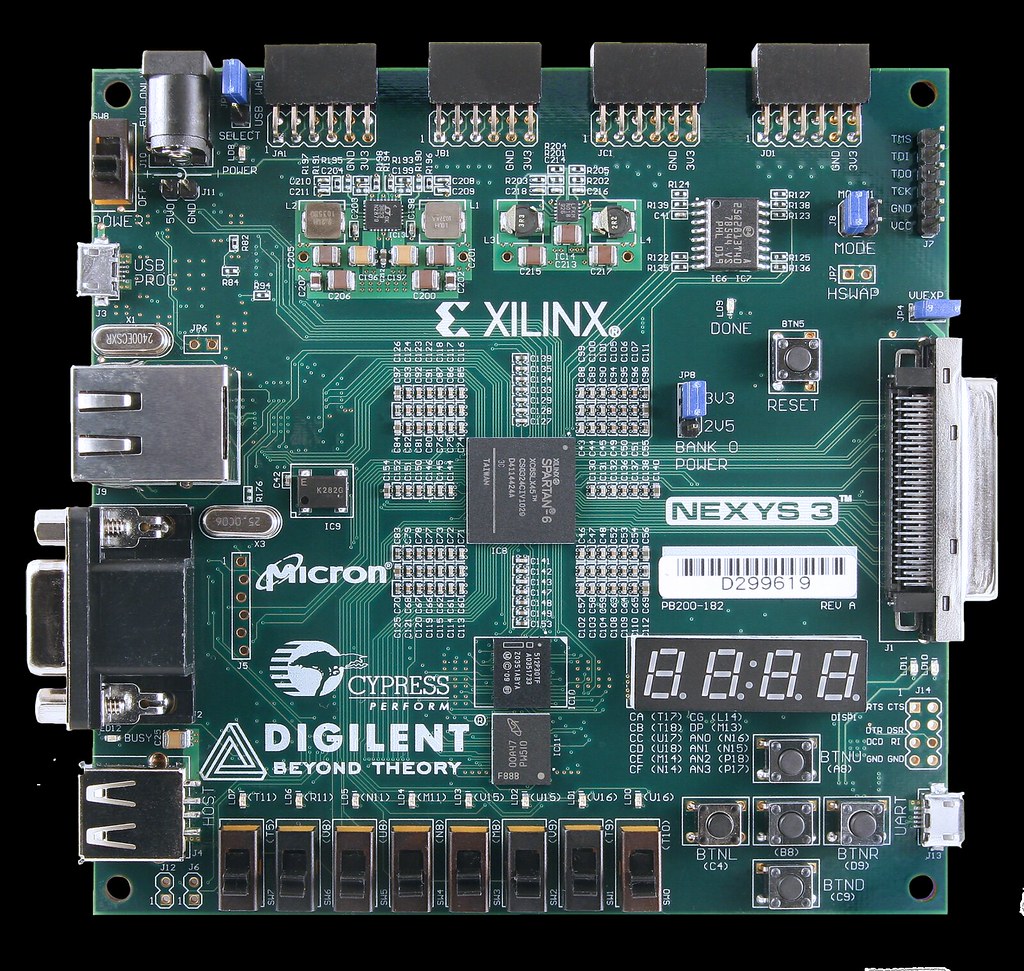
**

* Pentru acastă animație am făcut oarecum invers decât la animația 2; dacă acolo am extras câte o lieră pe rând,aici vom adăuga.

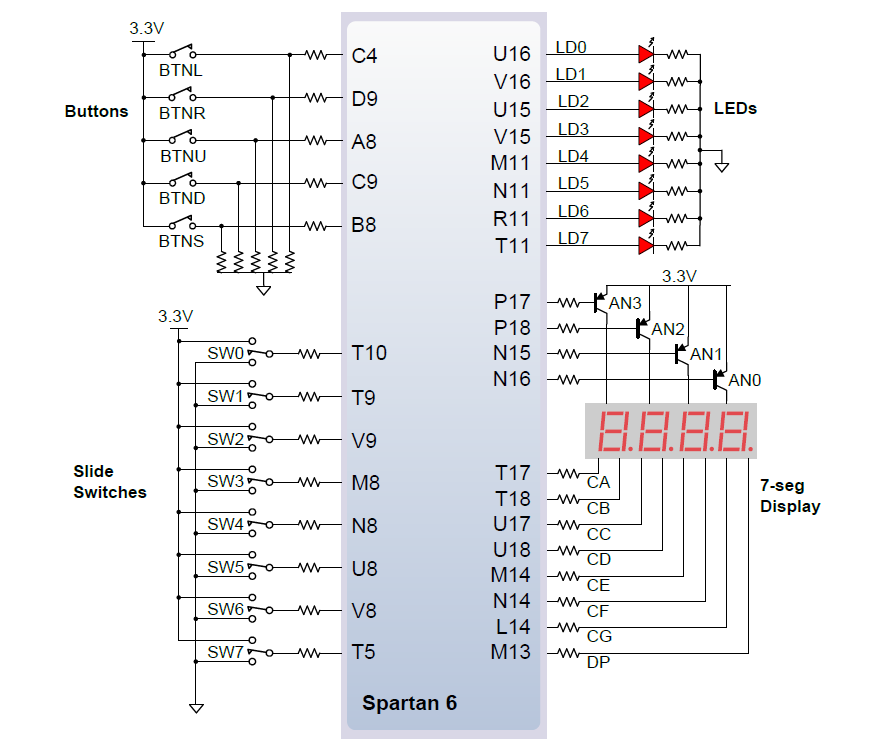
U\_-\_-\_ 🡪 U-T-\_-\_ 🡪 U-T-C-\_ 🡪 U-T-C-N

* Pentru a se accesa această animație, de pe intrerupătoarele plăcuței utilizatorul va da valoarea ”11” pentru intrarea Anim
* *Fișierul de constrângeri din ISE pentru placa cu FPGA aleasă*

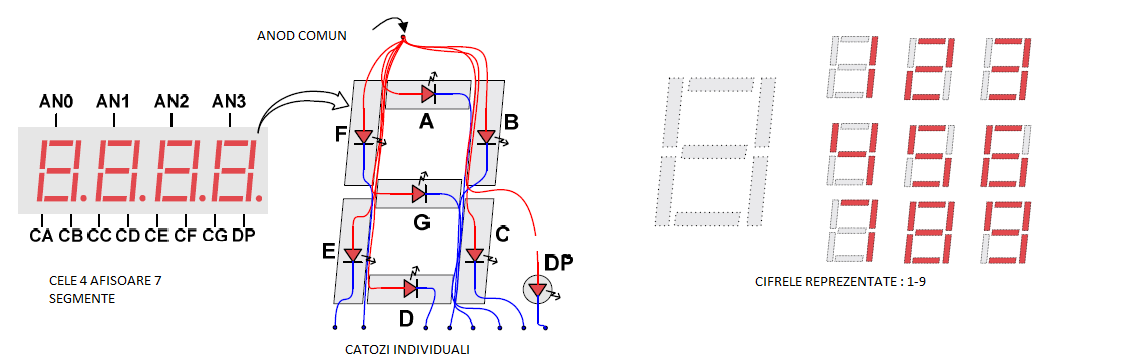
Pentru această parte am folosit placa **Nexys3**.   
Nexys3 este un sistem complet, gata de utilizare,o platformă de dezvoltare a circuitului digital bazat pe FPGA Xilinx Spartan-6 LX16



Starter Kit: **cele 4 afișoare pe 7 segmente**, **cele 8 comutatoare cu 2 stări**, **cele 5 comutătoare de tip push-button** și **LED-urile.**



Cele 4 afișoare pe 7 segmente:



-după cum se observă din imagine, afișoarele au cele 7 leduri reprezentând segmentele, si un simbol **dp** reprezentând punctul decimal

-fiecare are un anod activ pe 0 (funcționează numai atunci când sunt setate pe 0 logic)

În programul nostru am folosit 2 switch-uri, oscilatorul de quartz interior de **100MHz** pe post de Clock și afișorul pe 7 segmente. Adresele fiecărui switch si ale anodurilor si catodurilor afișorului utilizat

Catoadele (ledurile) afișorului

Anodurile afișorului(vectorul pe 4 biți)

Întrerupătoarele folosite pentru alegerea animației

*NET "Clock" LOC = "V10";*

*NET "Anim(0)" LOC = "T10";*

*NET "Anim(1)" LOC = "T9";*

*NET "Anod(0)" LOC = "P17";*

*NET "Anod(1)" LOC = "P18";*

*NET "Anod(2)" LOC = "N15";*

*NET "Anod(3)" LOC = "N16";*

*NET "segment(0)" LOC = "T17";*

*NET "segment(1)" LOC = "T18";*

*NET "segment(2)" LOC = "U17";*

*NET "segment(3)" LOC = "U18";*

*NET "segment(4)" LOC = "M14";*

*NET "segment(5)" LOC = "N14";*

*NET "segment(6)" LOC = "L14";*

*4. Lista de componente utilizate*

-Memorie ROM -Registru pe 7 biți

-Divizor de fracvență -Registru de deplasare pe 4 biți

-Multiplexor 4:1 pe 5 biți

-Demultiplexor 1:4 pe 7 biți

-Demultiplexor 1:4 pe 1 bit

-Numărător pe 2 biți

-Multiplexor pe 7 biți

1. *Semnificaţia notaţiilor de I/O şi a semnalelor interne*

* Anod : Vectorul de 4 biți care reprezintă cele 4 anoduri
* Segmente : ieșirile pentru afișoarele 7 segmente (vector de 7 componente)
* Clk : clock-ul de la placuță
* Anim : intrările de la switch-uri care permit utilizatorului să aleagă animația dorită
* Sel\_adr sunt ieșirile numărătorului pe 2 biți care au rolul selecțiilor pentru Dmux-urile si mux-ul care realizează memorarea simbolurilor in register
* S : ieșirile divizorului de frecvență care vor lua rolul selecțiilor pentru multiplexoarele ce realizează animațiile
* Adr1,adr2,adr3,adr4 – constante care reprezintă adresele la care se află *simbolurile ce vor fi afișate*
* Adr\_c : este adresa curentă cu care se accesează memoria (una din cele 4 adrese de mai sus)
* Empty – un vector de 7 biți ”gol” care va reprezenta spațiul/afișorul gol

1. *Justificarea soluţiei alese*

* Am ales acest mod de realizare a proiectului pentru ca acesta să fie cât mai simplu și practic de implementat,dar și eficient. Am dat nume cât mai sugestive intrărilor si ieșirilor,cât si semnalelor si constantelor.Prin implementarea cu multiplexoare am incercat o abordare cât mai practică pentru alegerea fiecărui simbol ce va fi afișat ,acestea oferind o modificare si personalizare foarte ușoară , iar selecțiile or au avut avantajul de a rezolva problema ”divizării de frecvență” cât și a alegerii animației.

1. *Posibilităţi de dezvoltare ulterioară*

* Posibile dezvoltări ale proiectului ar fi adăugarea altor tipuri de animații din care ar putea alegere utilizatorul,o extindere a alfabetului folosit și o mai ușoară schimbare a mesajului pe care utilizatorul dorește să îl afișeze.

1. *Utilizare și rezultate*

Pentru realizarea acestui proiect am folosit programul ISE-Design Suite pentru scrierea codului in limbaj VHDL și realizarea simulărilor si placa Nexys3( bazată pe Xilinx Spartan-6 LX16 FPGA )

Pentru a simula proiectul pe o placa FPGA, se deschide programul Xilinx ISE. Aici se da click pe New Project, si se selecteaza placuta cu care lucram. Mai apoi se adauga fisierele .vhd din folderul unde am lucrat in Active VHDL cu Project -> Add Source. Pasul urmator ar fi sintetizarea sursei, acest lucru fiind posibil apasand de doua ori pe Synthesize. Dupa aceasta se da click dreapta pe proiect in hierarhie: Source -> Implementation Constraints File, in care se vor specifica care buton ce rol va avea in timpul simularii. Mai apoi se da click de doua ori pe “Implement design”, “Generate Programming File” si “Manage Configuration Projekt (iMPACT)”. Aici se va deschide o noua ferestrea unde va trebui adaugat fisierul nostru .bit (“Boundary Scan” -> “Add Xilinx Device”). Ultimul pas ar fi sa dam click dreapta pe figura care apare pe ecran, si apasand pe “Program”, placa FPGA este gata de testare.

În programul acesta am folosit 2 switch-uri pentru a alege animația dorită , oscilatorul de quartz al plăcii pentru Clock , si catoadele si anoadele afișorului pe 7 segmente pentru a ne afișa textul.