UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

Proiect
Calculator de buzunar
Disciplina: Proiectarea Sistemelor Numerice

Student: Strujan Florentina

Facultate: Automatică și Calculatoare

Specializare: Calculatoare și Tehnologia Informației

Grupa: 302110

Îndrumător: Văcariu Lucia

Cuprins

I. Specificația proiectului

a) Cerință

Să se proiecteze un calculator de buzunar cu operații logice fundamentale (adunare, scădere, înmulțire, împărțire). Operațiile de înmulțire și împărțire se vor implementa folosind algoritmi specifici, nu operatorii limbajului. Operanzii sunt reprezentați pe 8 biți cu semn. Operanzii și operatorii vor fi introduși secvențial, în formă zecimală. Se vor folosi afișajele cu 7 segmente de pe plăcuțele cu FPGA.

b) Funcționare

Implementarea proiectului este făcută în limbaj VHDL. Testarea si funcționalitatea acestuia sunt prezentate cu ajutorului simulatorului Active-HDL și implementarea pe plăcuța FPGA NEXYS 4.

Dispozitivul realizează, în urma unei selecții, cele 4 operații aritmetice de bază, folosind numere pe 8 biți: adunare, scădere, înmulțire și împărțire și prezintă rezultatul pe afișajul BCD-7-segment,rezultatul fiind pe 16 biți cu semn. În funcție de o a doua selecție, se vor alege semnele operanzilor

Operanzii sunt introduși prin intermediul switch-urilor și a două butoane de validare, unul pentru zeci și unul pentru unități. Pentru a valida o cifră, switch-ul corespunzător ei primește valoarea '1', iar butonul de validare zeci/validare unități este apăsat. Dacă mai multe switch-uri corespunzătoare cifrelor au valoarea '1', la apăsarea butonului de validare nu se va întâmpla nimic. Operația este introdusă de la switch-urile 14 și 13 astfel: 00 pentru adunare, 01 penru scădere, 10 pentru înmulțire și 11 pentru împărțire . Pentru selectarea operandului care trebuie introdus (primul sau al doilea), folosim switch-ul cu numărul 10. Pentru introducerea semnelor numerelor folosim switch-urile 12 și 11 astfel: 00 pentru ambele numere pozitive, 11 pentru ambele numere negative, 01 pentru primul număr pozitiv, iar al doilea negativ și 10 pentru ptimul număr negativ, iar al doilea pozitiv . Pe post de egal, folosim switch-ul cu numărul 15. Operanzii și rezultatul se afișează pe afișajul BCD-7 segment.

În cazul întâmpinării unei erori, precum împărțirea la 0 sau rezultatul mai mare de 999, se afișează mesajul "FAIL" (failure) pe afișajele cu 7 segmente ale plăcuței.

II. Descriere schemă bloc cu componente

Cutia neagră

Utilizând mediul de dezvoltare Xilinx ISE Design Suite obținem, în urma procesului de sintetizare, schema bloc a calculatorului de buzunar pe care dorim să îl implementăm.

Cutia neagră a memoriei este prezentată în figura 1, având ca și intrări principale datele introduse de la switch-uri (sw(15:0)), clock-ul intern al plăcuței, butonul de introducere a unităților (buton_unitati) și cel de introducere a zecilor (buton_zeci). Ieșirile principale sunt datele introduse, care vor fi afișate pe BCD cu 7 segmente (myanod, mycatod).

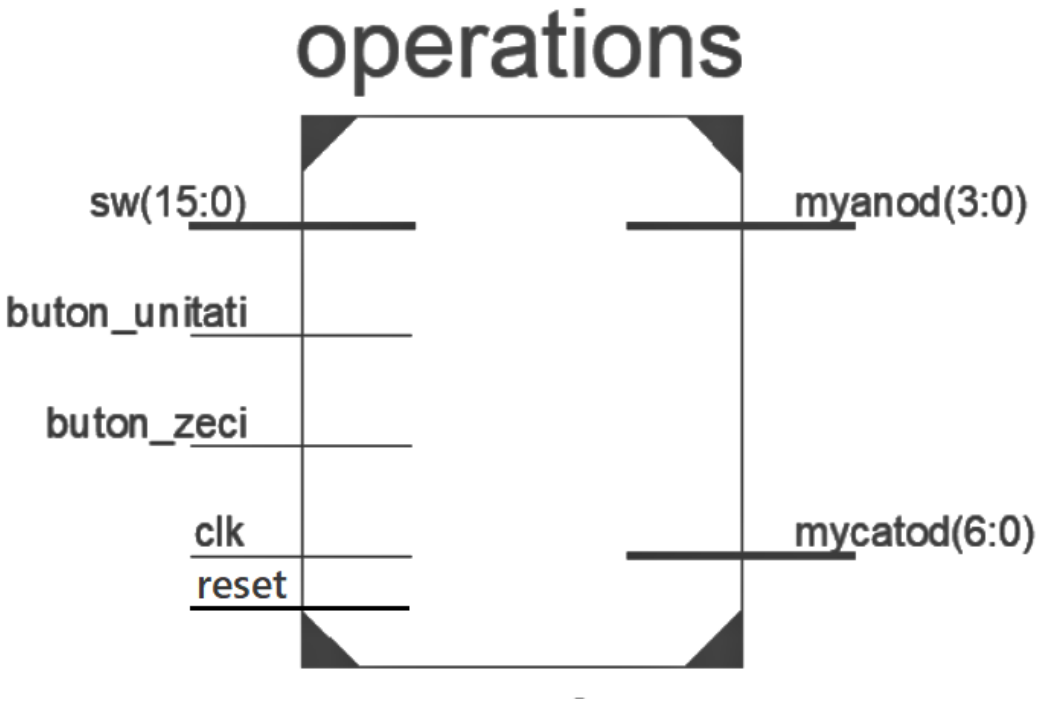

Figura 1. *Cutia neagră a calculatorului de buzunar*

Schema bloc

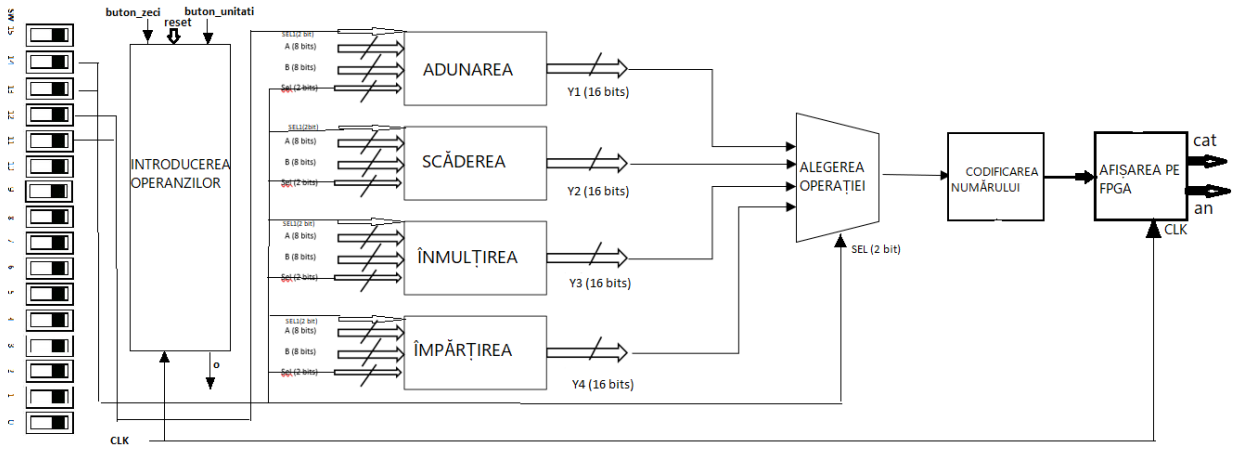

Figura 2. *Schema bloc a calculatorului de buzunar*

Prezența a patru operații implică necesitatea folosirii a 2 biți pentru prima intrare de selecție. Intrarea SEL reprezintă selecția operației efectuate:

- ➔ 00 pentru adunare
- ➔ 01 pentru scădere
- ➔ 10 pentru înmulțire
- ➔ 11 pentru împărțire

Prezența a patru combinații de semne implică necesitatea folosirii a 2 biți pentru intrarea a doua de selecție. Intrarea SEL1 reprezintă selecția semnului operanzilor:

- ➔ 00 pentru ambii operanzi pozitivi
- ➔ 01 pentru primul pozitiv, al doilea negativ
- ➔ 10 pentru primul negativ, al doilea pozitiv
- ➔ 11 pentru ambii operanzi negativi

Componente utilizate

Componentele folosite pentru realizarea proiectului sunt:

- sumator
- scăzător
- înmulțitor
- împărțitor
- multiplexor

- display

-codificator

- debouncer pentru butoane

- indata (introducerea datelor)

Toate aceste componente sunt interconectate în fișierul operatii.vhdl

Pentru operațiile de înmulțire și împărțire folosim algoritmul adunării repetate, respectiv al scăderii repetate.

S-a apelat la utilizarea unui multiplexor 4:1 care realizează selectarea output-ului corespunzător fiecărei operații, în funcție de selecția inițială.

III. Proiectare și implementare

Proiectare componente

Adunarea

- operația reprezintă adunarea dintre 2 numere pe 8 biți. Aceasta se realizează cu ajutorul unui FULL ADDER simbolizat prin semnul “+”.
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- Construim rezultatul în variabila z, pentru a face trecerea de la 8 la 16 biți, iar la final outputul y ia valoarea lui z

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

entity adunare is
    port (SEL,SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end entity;

architecture addition of adunare is
begin
    process(SEL,SEL1,a,b)
    variable z: std_logic_vector(15 downto 0);  -- variabila care schimba lungimea rezultatului
    begin                                       --semnul afisat se va determina in functie de SEL1 in momentul afisarii
    if(SEL="00") then
        if(SEL1="00") then                      --daca amandoua sunt pozitive
            z := "0000000000000000";            --initializarea vectorului cu 0
            z := z+a+b;
        end if;
        if(SEL1="11") then                      --daca amandoua sunt negative
            z := "1000000000000000";
            z := z+a+b;
        end if;
```

```
        if(SEL1="01") then                  --daca primul e pozitiv, al doulea negativ
            if(a<=b)    then
            z := "1000000000000000";        --se determina numarul cel mai mare
            z(7 downto 0) := b-a;
        else
            z := "0000000000000000";
            z(7 downto 0) := a-b;
            end if;
        end if;
        if(SEL1="10") then                 --daca primul e negativ,al doilea pozitiv
            if(a<=b)    then
            z := "0000000000000000";
            z(7 downto 0) := b-a;
        else
            z := "1000000000000000";
            z(7 downto 0) := a-b;
            end if;
        end if;
    end if;
    y <= z;
    end process;
end architecture;
```

Figura 3. *Componenta "adunare"*

Scăderea

- operația reprezintă scăderea dintre 2 numere pe 8 biți. Aceasta se realizează prin semnul "-".
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- Construim rezultatul în variabila z, pentru a face trecerea de la 8 la 16 biți, iar la final outputul y ia valoarea lui z

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

entity scadere is
    port (SEL,SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end entity;

architecture subtraction of scadere is
begin
    process(SEL,SEL1,a,b)
    variable z: std_logic_vector(15 downto 0);
    begin
    if(SEL="01") then
        if(SEL1="00")then
          if(a<=b)then
              z := "1000000000000000";
              z(7 downto 0) := b-a;
          else
              z := "0000000000000000";
              z(7 downto 0) := a-b;
          end if;
```

```
       end if;
       if(SEL1="11") then
         if(a<=b)  then
             z := "0000000000000000";
             z(7 downto 0) := b-a;
         else
             z := "1000000000000000";
             z(7 downto 0) := a-b;
         end if;
       end if;
       if (SEL1="01") then
           z := "0000000000000000";
           z :=z+a+b;
       end if;
       if (SEL1="10") then
           z := "1000000000000000";
           z :=z+a+b;
       end if;
   end if;
   y <= z;
   end process;
end architecture;
```

Figura 4. *Componenta "scădere"*

Înmulțirea

- operația reprezintă înmulțirea a 2 numere pe 8 biți
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- se realizează prin metoda adunărilor repetate: a este adunat de b ori
- Construim rezultatul în variabila z, pentru a face trecerea de la 8 la 16 biți, iar la final outputul y ia valoarea lui z

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

entity inmultire is
    port (  SEL, SEL1 : in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end entity;

architecture multiplication of inmultire is
begin
    process ( SEL , SEL1 , a , b )
variable r: std_logic_vector (7 downto 0);
variable z: std_logic_vector (15 downto 0);
variable k:integer;
 begin
     if(SEL="10") then
         if( SEL1="00") then                    --ambele numere pozitive
           if(b /= "0000000000000000") then       -- b diferit de 0
            z:= "0000000000000000";
            r := "00000000";
            for k in 1 to 128 loop
            if (r < b) then
                z := z+a;              -- se aduna a de b-1 ori
                r := r + '1';          -- se numara adunarile, apoi se compara cu b
                end if;
            end loop;
            y<= z;
            else
            y <="0000000000000000";
           end if;
         end if;
           if( SEL1="11") then                    --ambele numere negative
           if(b /= "0000000000000000") then       -- b diferit de 0
            z:= "0000000000000000";
            r := "00000000";
            for k in 1 to 128 loop
            if (r < b) then
                z := z+a;              -- se aduna a de b-1 ori
                r := r + '1';          -- se numara adunarile, apoi se compara cu b
                end if;
            end loop;
            y <= z;
            else
            y <="0000000000000000";
           end if;
           end if;
           if( SEL1="10") then                    --primul nr negativ, al doilea pozitiv
           if(b /= "0000000000000000") then       -- b diferit de 0
            z:= "1000000000000000";
            r := "00000000";
            for k in 1 to 128 loop
            if (r < b) then
                z := z+a;              -- se aduna a de b-1 ori
                r := r + '1';          -- se numara adunarile, apoi se compara cu b
                end if;
            end loop;
            y<= z;
            else
            y <="0000000000000000";
           end if;
           end if;
           if( SEL1="01") then                    --primul nr pozitiv, al doilea negativ
           if(b /= "0000000000000000") then       -- b diferit de 0
            z:= "1000000000000000";
            r := "00000000";
            for k in 1 to 128 loop
            if (r < b) then
                z := z+a;              -- se aduna a de b-1 ori
                r := r + '1';          -- se numara adunarile, apoi se compara cu b
                end if;
```

```
            end loop;
            y<= z;
            else
            y <="0000000000000000";
           end if;
        end if;
     end if;

    end process;
end architecture;
```

Figura 5. *Componenta "înmulțire"*

Împărțirea

- Operația reprezintă împărțirea a două numere pe 8 biți
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- se realizează prin metoda scăderilor repetate:
    - dacă b (împărțitorul) este diferit de 0, z ia valoarea lui a
    - scădem repetat b din z, atâta timp cât z e mai mare sau egal cu b
    - numărăm scăderile efectuate, rezultatul reprezentând câtul
- în cazul în care operandul a este mai mic decât b, câtul primește valoarea 0, în cazul unei eventuale împărțiri la 0, y primește valoarea "1111111111111111", pe display afișându-se "FAIL",(failure).

Implementarea în limbaj VHDL se poate observa în figura 6.

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

entity impartire is
    port (  SEL,SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end entity;

architecture division of impartire is
begin
    process(SEL,SEL1,a,b)
variable cat: std_logic_vector (15 downto 0);
variable z: std_logic_vector (15 downto 0);
variable k:integer;
 begin
     if(SEL="11") then
         if(SEL1="00") then
          if(b /= "0000000000000000") then        -- daca b e diferit de 0
            z:= "0000000000000000";                   -- cat si z sunt 0
            cat := "0000000000000000";
            z := z+a;
            for k in 1 to 128 loop         -- scadem z repetat cat timp z e mai mare sau egal cu b
```

```
            if (z >= b) then
                z := z-b;
                cat := cat + '1';          -- numaram scaderile
                end if;
            end loop;
            y<= cat;          -- restul este incarcat in primii 8 biti ai outputului
          else
            y <="1111111111111111";         -- afisam FAIL
        end if;
        end if;
        if(SEL1="01") then
          if(b /= "0000000000000000") then        -- daca b e diferit de 0
            z:= "0000000000000000";                  --z e 0
            cat := "1000000000000000";           -- numerele de sens opusa rezulta un cat negativ
            z := z+a;
            for k in 1 to 128 loop         -- scadem z repetat
            if (z >= b) then
                z := z-b;
                cat := cat + '1';          -- numaram scaderile
                end if;
            end loop;
            y<= cat;
          else
            y <="1111111111111111";         -- afisam FAIL
        end if;
        end if;
        if(SEL1="10") then
          if(b /= "0000000000000000") then
            z:= "0000000000000000";
            cat := "1000000000000000";
            z := z+a;
            for k in 1 to 128 loop
            if (z >= b) then
                z := z-b;
                cat := cat + '1';
                end if;
            end loop;
            y<= cat;
          else
            y <="1111111111111111";
        end if;
        end if;
        if(SEL1="11") then
          if(b /= "0000000000000000") then
            z:= "0000000000000000";
            cat := "0000000000000000";
            z := z+a;
            for k in 1 to 128 loop
            if (z >= b) then
                z := z-b;

                cat := cat + '1';
                end if;
            end loop;
            y<= cat;
          else
            y <="1111111111111111";
        end if;
        if (a<b) then                          -- daca a < b atunci catul este 0
            y<= "0000000000000000";
        end if;
        end if;
    end if;
  end process;
end architecture;
```

Figura 6. *Componenta impartire*

Multiplexorul

- transmite, in funcție de selecție, unul dintre cele 4 rezultate ale operațiilor către afișor
- Fiecare dintre cele 4 rezultate se află pe una dintre intrările multiplexorului

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

entity multiplexor is
    port ( SEL: in std_logic_vector (1 downto 0);
    x,m,z,t: in std_logic_vector (15 downto 0);
    y: out std_logic_vector (15 downto 0));
end multiplexor;

architecture mux of multiplexor is
begin
    process(SEL,x,m,z,t)
    begin
        case SEL is
            when "00" => y <= x;
            when "01" => y <= m;
            when "10" => y <= z;
            when "11" => y <= t;
            when others =>
        end case;
    end process;
end architecture;
```

Figura 7. *Componenta multiplexor*

Debouncer-ul

Apare necesitatea utilizării unui debouncer pentru butoanele „buton_zeci”, „buton_unități” și „reset”

Figura 8. *Oscilarea semnalului de la buton*

Când se apasă un buton de pe plăcuța FPGA, acesta oscilează intre stările high și low de câteva ori înainte de a se fixa pe un output. Pentru a evita această stare de „bouncing”, folosim un debouncer.

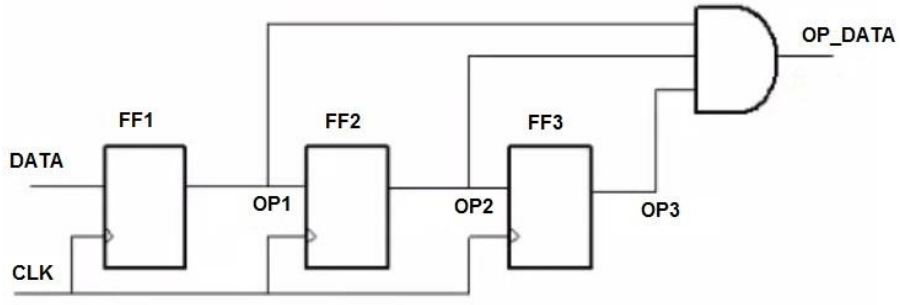

Figura 9. *Logica din spatele debouncerului*

Implementarea unui debouncer în VHDL este următoarea:

```
4   entity VHDL_Code_Debounce is
5   Port (
6   DATA: in std_logic;
7   CLK : in std_logic;
8   OP_DATA : out std_logic);
9   end VHDL_Code_Debounce ;
10
11  architecture Behavioral of VHDL_Code_Debounce is
12  Signal OP1, OP2, OP3: std_logic;
13  begin
14      Process(CLK)
15      begin
16      if rising_edge(CLK) then
17          OP1 <= DATA;
18          OP2 <= OP1;
19          OP3 <= OP2;
20  end if;
21  end process;
22      OP_DATA <= OP1 and OP2 and OP3;
23  end Behavioral;
24
```

Figura 10. *Componenta "debouncer"*

Codificator

-Componenta în cadrul căreia se codifică numele din zecimal în binar pentru afișarea pe plăcuța FPGA

```
3
4   entity conversie is
5       port(A: in integer;
6       B: out std_logic_vector(3 downto 0));
7   end conversie;
8
9   architecture binar of conversie is
10  begin
11      process(A)
12      begin
13          case A is
14              when 0 => B<="0000";
15          when 1 => B<="0001";
16          when 2 => B<="0010";
17          when 3 => B<="0011";
18          when 4 => B<="0100";
19          when 5 => B<="0101";
20          when 6 => B<="0110";
21          when 7 => B<="0111";
22          when 8 => B<="1000";
23          when 9 => B<="1001";
24          when others => B<="0000";
25      end case;
26      end process;
27  end binar;
```

Figura 11. *Componenta "conversie"*

Input data

- Componenta în cadrul căreia se fac citirile
- Pentru o introducere ușoară a datelor, am ales ca numărul maxim ce poate fi introdus să fie 99. Pentru a introduce cifra unităților, selectăm de la switch-uri cifra pe care dorim să o introducem, iar apoi apăsăm butonul „buton_unitati", care validează cifra.

Introducerea zecilor este similară, însă se apasă butonul „buton_zeci”. În cazul în care se va apăsa butonul „reset” , numărul va fi resetat. Cifra zecilor va fi înmulțită cu 10 ,la care va fi adunată cifra unităților pentru formarea corectă a numărului.

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

entity indata is
    port(sw: in std_logic_vector(15 downto 0);
    clk: in std_logic;
    buton_unitati: in std_logic;
    buton_zeci: in std_logic;
    reset: in std_logic;
    o: out std_logic_vector(7 downto 0));
end indata;

architecture intrari of indata is

component VHDL_Code_Debounce is
    Port (
    DATA: in std_logic;
    CLK : in std_logic;
    OP_DATA : out std_logic);
end component;

signal aux,aux1: std_logic_vector(3 downto 0);
signal buton_unitati_rezolvat: std_logic;
```

```
signal buton_zeci_rezolvat: std_logic;
signal reset_rezolvat: std_logic;
begin
operatie_1: VHDL_Code_Debounce  port map (buton_unitati,clk,buton_unitati_rezolvat);
operatie_2: VHDL_Code_Debounce  port map (buton_zeci,clk,buton_zeci_rezolvat);
resetul: VHDL_Code_Debounce  port map (reset,clk,reset_rezolvat);
process(buton_unitati_rezolvat,sw,reset_rezolvat)
    begin
    if rising_edge(reset_rezolvat) then
        aux1<="0000";
        end if;
    if rising_edge(buton_unitati_rezolvat) then
        case sw(9 downto 0) is
            when "1000000000" => aux1 <="0000"; -- 0
            when "0100000000" => aux1 <="0001"; -- 1
            when "0010000000" => aux1 <="0010"; -- 2
            when "0001000000" => aux1 <="0011"; -- 3
            when "0000100000" => aux1 <="0100"; -- 4
            when "0000010000" => aux1 <="0101"; -- 5
            when "0000001000" => aux1 <="0110"; -- 6
            when "0000000100" => aux1 <="0111"; -- 7
            when "0000000010" => aux1 <="1000"; -- 8
            when "0000000001" => aux1 <="1001"; -- 9
            when others =>
            end case;
```

```
    end if;
end process;

process(buton_zeci_rezolvat,sw,reset_rezolvat)
    begin
    if rising_edge(reset_rezolvat) then
        aux<="0000";
        end if;
    if rising_edge(buton_zeci_rezolvat) then
        case sw(9 downto 0) is
                when "1000000000" => aux <="0000"; -- 0
                when "0100000000" => aux <="0001"; -- 1
                when "0010000000" => aux <="0010"; -- 2
                when "0001000000" => aux <="0011"; -- 3
                when "0000100000" => aux <="0100"; -- 4
                when "0000010000" => aux <="0101"; -- 5
                when "0000001000" => aux <="0110"; -- 6
                when "0000000100" => aux <="0111"; -- 7
                when "0000000010" => aux <="1000"; -- 8
                when "0000000001" => aux <="1001"; -- 9
                when others =>
                end case;
        end if;
end process;
```

```
o<="00000000"+ aux*"1010" + aux1;

end architecture;
```

Figura 12. *Componenta „indata*

Afișajul 7 segmente

Componenta BCD (figura 15) are rolul de a afișa informația primită de la rezultat pe display-ul plăcuței. Acest lucru este posibil cu ajutorul celor 7 intrări comune a segmentelui BCD (catod), dar și cu cele 4 intrări separate a display-ului (anod), ambele fiind active pe 0. Aceste intrări sunt de fapt ieșirile blocului BCD. Introducerea informației în componentă se face prin patru semnale de 4 biți fiecare, câte unul pentru fiecare afișor dintre cele patru ale display-ului. În cadrul componentei BCD sunt folosite un numărător (pentru divizorul de frecvență) și două multiplexoare, care au ca și selecție ultimii doi biți(15 și 14) a vectorului emis de numărător.

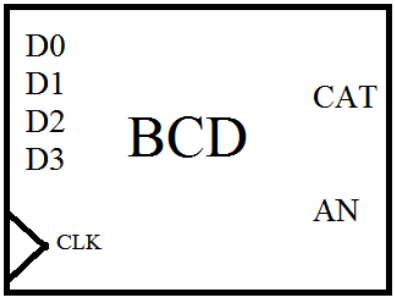

Figura 13. *Componenta BCD*

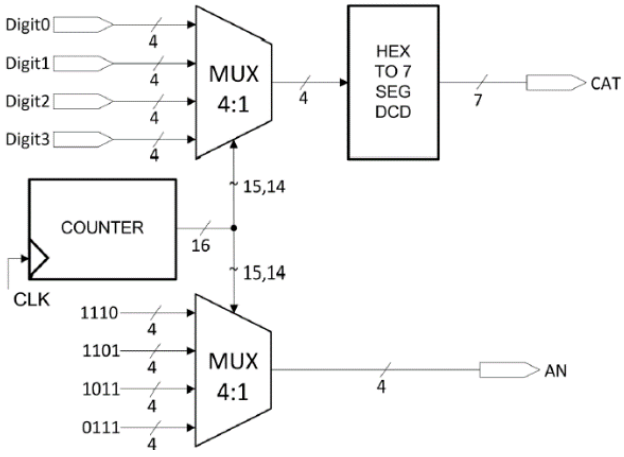

Figura 14. *Schema internă a componentei BCD*

Proiectare ansamblu

Codul VHDL din pagina principală a proiectului poate fi impărțit la rândul său în 3 părți importante: declararea componentelor folosite, declararea entității (cutia neagră - figura 1) și arhitectura (interiorul cutei negre).

Declararea entității conține porturile de intrare și ieșire ale acesteia, în cazul nostru cele 16 switch-uri de pe placa Nexys 4, clock-ul intern, butoanele de validare și afișajul 7 segmente (figura 15).

```
entity operations is
    port(   reset: in std_logic;
    sw: in std_logic_vector(15 downto 0);
    clk: in std_logic;
    buton_unitati: in std_logic;
    buton_zeci: in std_logic;
    mycatod:out std_logic_vector(6 downto 0);
    myanod:out std_logic_vector(3 downto 0));
end entity;
```

Figura 15. *Entitatea proiectului*

Arhitectura conține componentele menționare anterior. Fiecare componentă este construită separat, cu ajutorul proceselor sau în mod structural.

```
architecture operatii of operations is
component adunare is
    port (SEL, SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end component;

component scadere is
    port (SEL, SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end component;

component inmultire is
    port (SEL,SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end component;

component impartire is
    port (SEL,SEL1: in std_logic_vector (1 downto 0);
    a,b: in std_logic_vector (7 downto 0);
    y: out std_logic_vector (15 downto 0));
end component;
```

```
component multiplexor is
    port ( SEL: in std_logic_vector (1 downto 0);
    x,m,z,t: in std_logic_vector (15 downto 0);
    y: out std_logic_vector (15 downto 0));
end component;

component indata is
    port(reset: in std_logic;
    sw: in std_logic_vector (15 downto 0);
    clk: in std_logic;
    buton_unitati,buton_zeci: in std_logic;
    o: out std_logic_vector(7 downto 0));
end component;

component BCD is
    Port ( D: in  STD_LOGIC_VECTOR (15 downto 0);
           CLK: in  STD_LOGIC;
           CAT: out  STD_LOGIC_VECTOR (6 downto 0);
           AN: out  STD_LOGIC_VECTOR (3 downto 0));
end component;
```

Figura 13. *Declararea componentelor*

În arhitectură sunt legate componentele între ele cu ajutorul unor semnale. Ne folosim de 2 procese, unul pentru selecția operației, celălalt pentru egal (mai exact, switch-ul 15 afișează un operand când este în starea 0 și rezultatul când este în starea 1).

```
63  signal y,y1,y2,y3,y4: std_logic_vector (15 downto 0);
64  signal outputFromSwitch: std_logic_vector(7 downto 0);
65  signal t1: std_logic_vector (7 downto 0) := "00000000";
66  signal t2: std_logic_vector(7 downto 0):= "00000000";
67  signal afis: std_logic_vector(15 downto 0):= "0000000000000000";
68
69  begin
70
71      op_0 : indata port map (reset,sw, clk, buton_unitati,buton_zeci,outputFromSwitch);
72      op_1 : adunare port map (sw(14 downto 13),sw(12 downto 11),t1,t2,y1); --14 si 13 sunt sel
73      op_2 : scadere port map (sw(14 downto 13),sw(12 downto 11),t1,t2,y2);  --12 si 11 sunt selectia semnului, sel1
74      op_3 : inmultire port map (sw(14 downto 13),sw(12 downto 11),t1,t2,y3);
75      op_4 : impartire port map (sw(14 downto 13),sw(12 downto 11),t1,t2,y4);
76      mux: multiplexor port map(sw(14 downto 13),y1,y2,y3,y4,y);  -- outputul fiecarei operatii va deveni input pentru multiplexor,in funct
77
78      print: BCD port map(afis,clk,mycatod,myanod);
79  process (sw)
80  begin
81      case sw(10) is --cu switchul 10 selectez daca introduc numarul 1 sau 2
82          when '0' => t1<= outputFromSwitch;
83          when '1' => t2<= outputFromSwitch;
84          when others =>
85      end case;
86  end process;
87
88  process (sw)
89  begin
90      case sw(15) is --15 e egalul operatiei
91          when '0' => afis(7 downto 0) <= outputFromSwitch;
92        when '1' => afis <= y;
93          when others => afis <= (others => '0');
94      end case;
95  end process;
96
97
98  end operatii;
```

Figura 14. *Arhitectura proiectului*

IV. Lista de componente utilizate

Componentele folosite pentru realizarea proiectului sunt:

- sumator
- scăzător
- înmulțitor
- împărțitor
- multiplexor
- bcd-to-7-segment display
- debouncer pentru buton
- codificator
- indata (introducerea datelor)

Toate aceste componente sunt interconectate în fișierul operatii.vhdl

Pentru operațiile de înmulțire și împărțire folosim algoritmul adunării repetate, respectiv al scăderii repetate.

S-a apelat la utilizarea unui multiplexor 4:1 care realizează selectarea output-ului corespunzător fiecărei operații, in funcție de selecția inițială.

Adunarea

- operația reprezintă adunarea dintre 2 numere pe 8 biți. Aceasta se realizează cu ajutorul unui FULL ADDER simbolizat prin semnul “+”.
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- Construim rezultatul în variabila z, pentru a face trecerea de la 8 la 16 biți, iar la final outputul y ia valoarea lui z

Scăderea

- operația reprezintă scăderea dintre 2 numere pe 8 biți. Aceasta se realizează prin semnul “-”.
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- Construim rezultatul în variabila z, pentru a face trecerea de la 8 la 16 biți, iar la final outputul y ia valoarea lui z

Înmulțirea

- operația reprezintă înmulțirea a 2 numere pe 8 biți
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- se realizează prin metoda adunărilor repetate: a este adunat de b ori
- Construim rezultatul în variabila z, pentru a face trecerea de la 8 la 16 biți, iar la final outputul y ia valoarea lui z

Împărțirea

- Operația reprezintă împărțirea a două numere pe 8 biți
- Proces ce depinde de selecție(SEL) , selecția semnului operanzilor(SEL1) și operanzii a și b
- se realizează prin metoda scăderilor repetate:
    - dacă b (împărțitorul) este diferit de 0, z ia valoarea lui a
    - scădem repetat b din z, atâta timp cât z e mai mare sau egal cu b
    - numărăm scăderile efectuate
- în cazul în care operandul a este mai mic decât b, câtul primește valoarea 0, în cazul unei eventuale împărțiri la 0, y primește valoarea ”1111111111111111”, pe display afișându-se ”FAIL” (failure).

Multiplexorul

- transmite, in funcție de selecție, unul dintre cele 4 rezultate ale operațiilor către afișor
- Fiecare dintre cele 4 rezultate se află pe una dintre intrările multiplexorului

Debouncer-ul

Apare necesitatea utilizării unui debouncer pentru butoanele „buton_zeci”, „buton_unități” și „reset”.

Când se apasă un buton de pe plăcuța FPGA, acesta oscilează intre stările high și low de câteva ori înainte de a se fixa pe un output. Pentru a evita această stare de „bouncing”, folosim un debouncer.

Codificatorul

-Componenta în cadrul căreia se codifică numele din zecimal în binar pentru afișarea pe plăcuța FPGA

Input data

- Componenta în cadrul căreia se fac citirile
- Pentru o introducere ușoară a datelor, am ales ca numărul maxim ce poate fi introdus să fie 99. Pentru a introduce cifra unităților, selectăm de la switch-uri cifra pe care dorim să o introducem, iar apoi apăsăm butonul „buton_unitati”, care validează cifra. În cazul în care se va apăsa butonul „reset” , numărul va fi resetat. Introducerea zecilor este similară, însă se apasă butonul „buton_zeci”. Cifra zecilor va fi înmulțită cu 10 ,la care va fi adunată cifra unităților pentru formarea corectă a numărului.

Pagina principală

Codul VHDL din pagina principală a proiectului poate fi impărțit la rândul său în 3 părți importante: declararea componentelor folosite, declararea entității (cutia neagră - figura 1) și arhitectura (interiorul cutei negre).

Declararea entității conține porturile de intrare și ieșire ale acesteia, în cazul nostru cele 16 switch-uri de pe placa Nexys 4, clock-ul intern, butoanele de validare și afișajul 7 segmente.

Arhitectura conține componentele menționare anterior. Fiecare componentă este construită separat, cu ajutorul proceselor sau în mod structural.

În arhitectură sunt legate componentele între ele cu ajutorul unor semnale. Ne folosim de 2 procese, unul pentru selecția operației, celălalt pentru egal (mai exact, switch-ul 15 afișează un operand când este în starea 0 și rezultatul când este în starea 1).

Afișajul 7 segmente

Componenta BCD are rolul de a afișa informația primită de la rezultat pe display-ul plăcuței. Acest lucru este posibil cu ajutorul celor 7 intrări comune a segmentelui BCD (catod), dar și cu cele 4 intrări separate a display-ului (anod), ambele fiind active pe 0. Aceste intrări sunt de fapt ieșirile blocului BCD. Introducerea informației în componentă se face prin patru semnale de 4 biți fiecare, câte unul pentru fiecare afișor dintre cele patru ale display-ului. În cadrul componentei BCD sunt

folosite un numărător (pentru divizorul de frecvență) și două multiplexoare, care au ca și selecție ultimii doi biți(15 și 14) a vectorului emis de numărător.

V. Semnificația notațiilor I/O și a semnalelor interne

Intrările principale sunt *datele introduse de la switch-uri (sw(15:0)),*astfel:

- sw(15) -egalul
- sw(14), sw(13) – selecția operației
- sw(12) - semnul primului operand
- sw(11) – semnul celui de-al doilea operand
- sw(10) – selecția numărului ce va fi introdus: primul sau al doilea
- sw(9:0) – selecția cifrei zecilor/ unităților

, intrarea *clk* la care este asignat clock-ul intern al plăcuței, *butonul de introducere a unităților (buton_unitati)* , *butonul de introducere a zecilor (buton_zeci)* și *resetul(reset).*

Ieșirile principale sunt datele introduse, care vor fi afișate pe cele 4 afișoare BCD ale display-ului *(myanod, mycatod*).

Semnalele interne sunt:

- OP1, OP2, OP3- semnalele utilizate pentru realizarea debouncer-ului
- NR- folosit pentru divizorul de frecvență și determinarea anodului folosit
- MUX1 și MUX2- folosite pentru determinarea cărei cifre pe care anod va fi afișată
- Numar- pentru conversia din std_logic_vector în integer
- Sute, zeci, unități – cifrele sutelor, zecilor, unităților în integer
- Sute1, zeci1, unitati1 – conversia sutelor,zecilor, unităților din integer în std_logic_vector
- Aux, aux1- folosite în componenta indata pentru cifra zecilor, respectiv a unităților , cu care va fi format numărul ce urmează să fie prelucrat
- Buton_unitati_rezolvat, buton_zeci_rezolvat, reset_rezolvat – pe baza lor se face selecția zecilor/unităților (ieșiri ale debouncer-ului)
- y1, y2, y3, y4- ieșirile componentelor operațiilor
- y- rezultatul operației alese
- outputFromSwitch- numărul introdus, ce va fi prelucrat în componenta op
- t1, t2 – primul și al doilea număr, în care se va introduce outputFromSwitch în funcție de switch-ul 10
- afis- rezultatul final , ce va fi afișat prin componenta BCD

Variabilele folosite sunt:

- r- variabila ce va fi incrementată , apoi comparată cu înmulțitorul
- z- variabila folosită pentru schimbarea lungimii rezultatului
- k- „i”- ul pentru for
- cat- va fi incrementat de fiecare dată când va fi scăzut împărțitorul din deîmpărțit, câtul împărțirii

Legătura dintre codul VHDL și plăcuța FPGA se face cu ajutorul UCF-ului, care este prezentat în figura:

```
NET "sw(15)" LOC= "V10" | IOSTANDARD= "LVCMOS33";
NET "sw(15)" CLOCK_DEDICATED_ROUTE = FALSE;
NET "sw(14)" LOC= "U11" | IOSTANDARD= "LVCMOS33";
NET "sw(13)" LOC= "U12" | IOSTANDARD= "LVCMOS33";
NET "sw(12)" LOC= "H6" | IOSTANDARD= "LVCMOS33";
NET "sw(11)" LOC= "T13" | IOSTANDARD= "LVCMOS33";
NET "sw(10)" LOC= "R16" | IOSTANDARD= "LVCMOS33";
NET "sw(10)" CLOCK_DEDICATED_ROUTE = FALSE;
NET "sw(9)" LOC= "U8" | IOSTANDARD= "LVCMOS33";
NET "sw(8)" LOC= "T8" | IOSTANDARD= "LVCMOS33";
NET "sw(7)" LOC= "R13" | IOSTANDARD= "LVCMOS33";
NET "sw(6)" LOC= "U18" | IOSTANDARD= "LVCMOS33";
NET "sw(5)" LOC= "T18" | IOSTANDARD= "LVCMOS33";
NET "sw(4)" LOC= "R17" | IOSTANDARD= "LVCMOS33";
NET "sw(3)" LOC= "R15" | IOSTANDARD= "LVCMOS33";
NET "sw(2)" LOC= "M13" | IOSTANDARD= "LVCMOS33";
NET "sw(1)" LOC= "L16" | IOSTANDARD= "LVCMOS33";
NET "sw(0)" LOC= "J15" | IOSTANDARD= "LVCMOS33";

NET "mycatod<0>" LOC= "T10" | IOSTANDARD= "LVCMOS33";
NET "mycatod<1>" LOC= "R10" | IOSTANDARD= "LVCMOS33";
NET "mycatod<2>" LOC= "K16" | IOSTANDARD= "LVCMOS33";
NET "mycatod<3>" LOC= "K13" | IOSTANDARD= "LVCMOS33";
NET "mycatod<4>" LOC= "P15" | IOSTANDARD= "LVCMOS33";
NET "mycatod<5>" LOC= "T11" | IOSTANDARD= "LVCMOS33";
NET "mycatod<6>" LOC= "L18" | IOSTANDARD= "LVCMOS33";
NET "myanod<0>" LOC= "J17" | IOSTANDARD= "LVCMOS33";
NET "myanod<1>" LOC= "J18" | IOSTANDARD= "LVCMOS33";
NET "myanod<2>" LOC= "T9" | IOSTANDARD= "LVCMOS33";
NET "myanod<3>" LOC= "J14" | IOSTANDARD= "LVCMOS33";

NET "buton_unitati" LOC= "P18" | IOSTANDARD= "LVCMOS33";
NET "buton_unitati" CLOCK_DEDICATED_ROUTE = FALSE;
NET "buton_zeci" LOC= "P17" | IOSTANDARD= "LVCMOS33";
NET "buton_zeci" CLOCK_DEDICATED_ROUTE =FALSE;
NET "clk" LOC = "E3" | IOSTANDARD= "LVCMOS33";
NET "clk" CLOCK_DEDICATED_ROUTE= FALSE;
NET "reset" LOC = "N17" | IOSTANDARD= "LVCMOS33";
NET "reset"CLOCK DEDICATED ROUTE= FALSE;
```

Figura 15. *Legătura dintre entitate și plăcuța FPGA*

VI. Placa FPGA Nexys 4

Aceasta are un clock intern de o viteză de 100 MHz (pinul E3), 16 switch-uri de tip slide, 16 led-uri, 5 butoane, 2 display-uri de 4 cifre fiecare cu 7 segmente (figura 15).

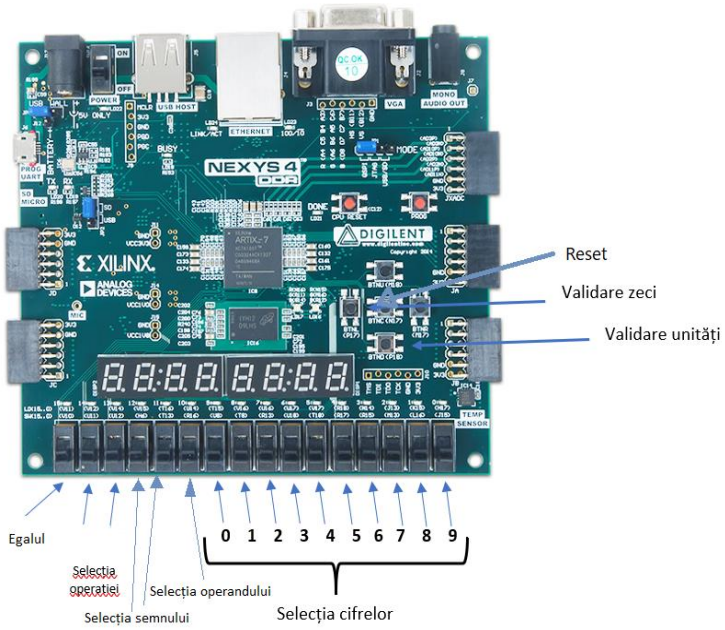

Figura 16. *Semnificația notațiilor efectuate pe placa NEXYS 4*

VII. Justificarea soluției alese

Am făcut alegerea metodei de implementare în funcție de nivelul de dificultate: am considerat-o ușoară, atât de proiectat, cât și de înțeles. Proiectul poate fi cu ușurință explicat oricărui utilizator, fie el specialist sau nu. Este un calculator compact, folosește doar operații fundamentale, însă este util studenților sau elevilor.

Calculatorul a fost realizat într-o astfel de manieră încât toate comenzile să fie cât se poate de clare, iar simularea să fie ușor de urmărit.

VIII. Utilizare și rezultate

A) Resurse necesare

Pentru utilizarea plăcuței FPGA și a calulatorului de buzunar este necesar programul ISE Design Suite, care poate fi downloadat de pe site-ul Xilinx. După instalarea programului, se începe rularea codului VHDL.

B)Descrierea utilizării

Se urmăresc următorii pași după ce FPGA-ul a fost progamat:

- Se introduce câte o cifră pe rând de la switch-uri (se observă modificarea display-ului
- Se validează cifra zecilor/unităților de la butoane
- Se modifică starea switch-ului 15 (selecția operandului), pentru introducerea celui de-al doilea operand
- Se repetă primii 2 pași
- Se selectează cu switch-urile 13 și 14 operația ce urmează să fie efectuată
- Se selectează cu switch-urile 11 și 12 semnele operanzilor
- Se modifică starea switch-ului 15 (egalul operației)
- Rezultatul este afișat pe display
- Se apasă btonul de reset pentru efectuarea unei noi operații

C)Rezultate obținute

Mediul de proiectare Active-HDL permite simularea codului scris în limbaj VHDL. Proiectantul introduce date sau taste stimulator, iar evoluția semnalelor poate fi urmărită prin intermediul unei diagrame waveform. Un exemplu poate fi observat în figura 14:

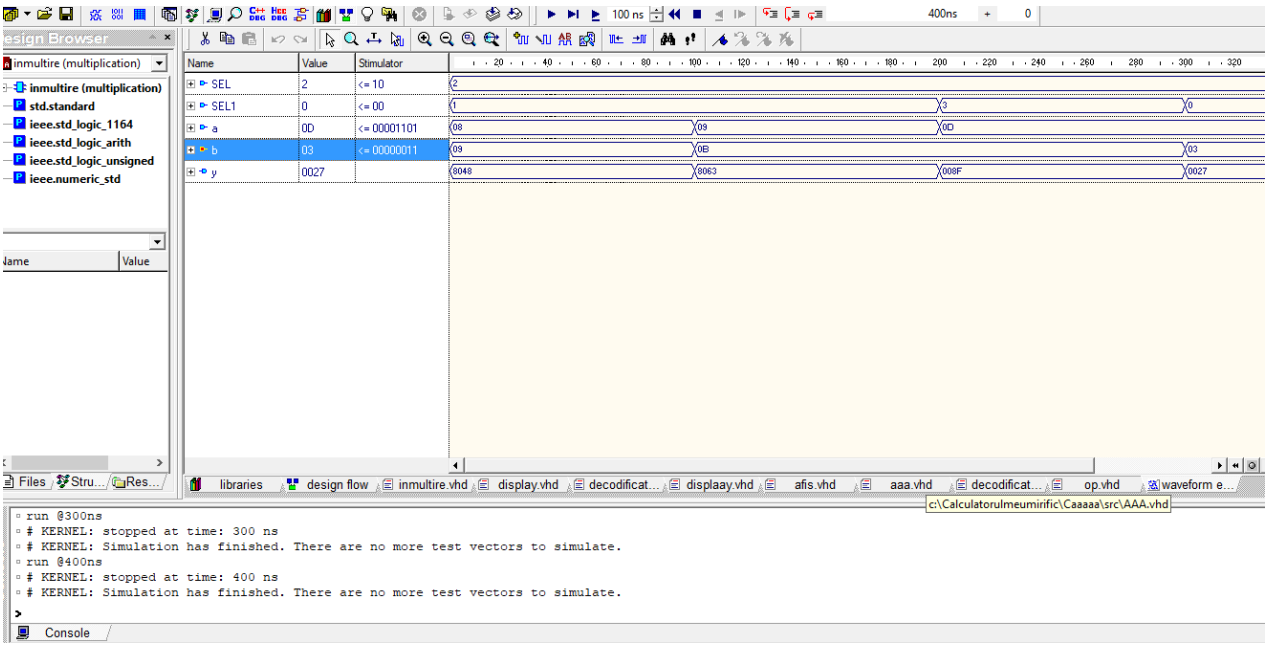

Figura 17. *Simularea proiectului în mediul de proiectare Active-HDL*

IX. Posibilități de dezvoltare ulterioară

Calculatorul ar putea fi îmbunătățit prin posibilitatea de a face un lanț de operații sau de a utiliza operanzi de dimensiuni mai mari.