**Universitatea Tehnica din Cluj-Napoca**

* **PROIECT-**

**Calculatorul de buzunar**

**Autor :**

Mates Bogdan 30215

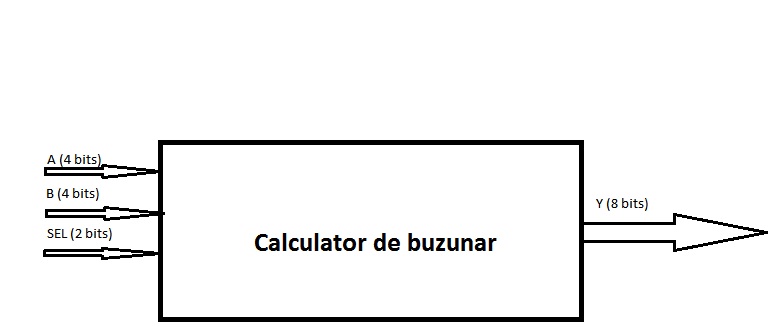
**Indrumator:**

Prof. Mihai Timar

**Calculator de buzunar**

Proiectul este implementat in totalitate in limbaj VHDL. Testarea si functionalitatea acestuia sunt prezentate cu ajutorului simulatorului Active-HDL si incarcate pe placuta FPGA . Proiectul isi propune implementarea operatiilor de adunare, scadere, inmultire, impartire cu numere pe 4 biti sub forma unui calculator de buzunar. Acesta inglobeaza cate un modul responsabil cu operatiile precedent specificate care sunt comandate de un modul UAL.

**DIAGRAMA BLOCK**



Cei doi operanzi pe 4 biti sunt A si B, iar ei vor fi introdusi in binar folosind butoanele de pe placuta.

SEL reprezinta selectia operatiei:

🡺 00 pentru adunare

**🡺** 01 pentru scadere (in modul)

🡺10 pentru inmultire

🡺11 pentru impartire

Prezenta a patru operatii implica necesitatea folosirii a 2 biti pentru aceasta intrare.

Rezultatul operatiei este reprezentat de Y. Vom folosi 8 biti deoarece inmultirea a 2 numere de lungime 4 biti implica un rezultat pe maxim 8 biti.

Astfel proiectul se imparte in 5 componente, 4 dintre ele reprezentand fiecare cate o operatie, iar ultima o reprezinta un multiplexor care va face selectia dintre rezultate si il va afisa pe cel necesar.

**COMPONENTE**

1. **Adunarea**

* operatia reprezinta adunare dintre 2 numere pe patru biti. Aceasta se realizeaza cu ajutorul unui FULL ADDER simbolizat prin semnul “+”.

**Sintaxa componentei:**

*component adunare is*

*port (SEL: in std\_logic\_vector (1 downto 0);*

*a,b: in std\_logic\_vector (3 downto 0);*

*y: out std\_logic\_vector (7 downto 0));*

*end component;*

*architecture adding of adunare is*

*begin*

*process(SEL,a,b)*

*variable z: std\_logic\_vector(7 downto 0);*

*begin*

*if(SEL="00") then*

*z := "00000000";*

*z := z+a+b;*

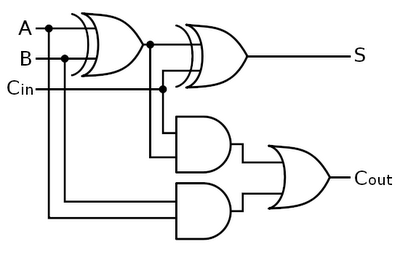
*end if;*

*y <= z;*

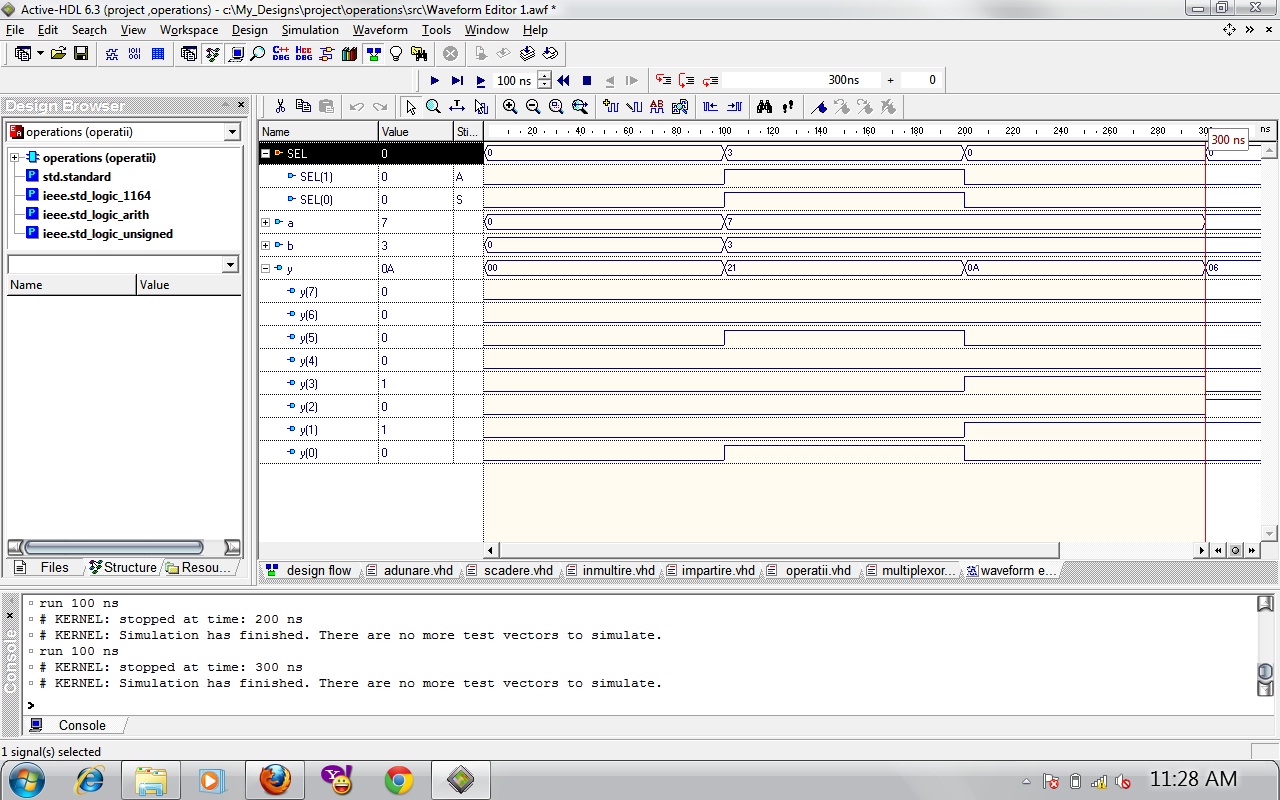
*end process;*

*end architecture;*

**Diagrama block:**



**Exemplificare waveform:**

****

**Variabile:**

* s-a folosit variabila Z pentru a face tranzitia de la 4 biti la 8 biti.

1. **Scaderea (in valoare absoluta)**

* operatia reprezinta scaderea in valoare absoluta dintre 2 numere pe patru biti. Aceasta se realizeaza cu ajutorul unui FULL SUBTRACTOR simbolizat prin semnul “-”. Operatia se va face scazand numarul mai mic din numarul mai mare, indiferent de ordinea acestora, realizand practice modulul diferentei dintre cele 2 numere.

**Sintaxa componentei:**

*component scadere is*

*port (SEL: in std\_logic\_vector (1 downto 0);*

*a,b: in std\_logic\_vector (3 downto 0);*

*y: out std\_logic\_vector (7 downto 0));*

*end component;*

*architecture substraction of scadere is*

*begin*

*process(SEL,a,b)*

*variable z: std\_logic\_vector(7 downto 0);*

*begin*

*if(SEL="01") then*

*z := "00000000";*

*if(a<=b) then*

*z(3 downto 0) := b-a;*

*else*

*z(3 downto 0) := a-b;*

*end if;*

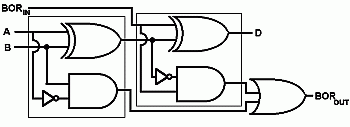
*end if;*

*y <= z;*

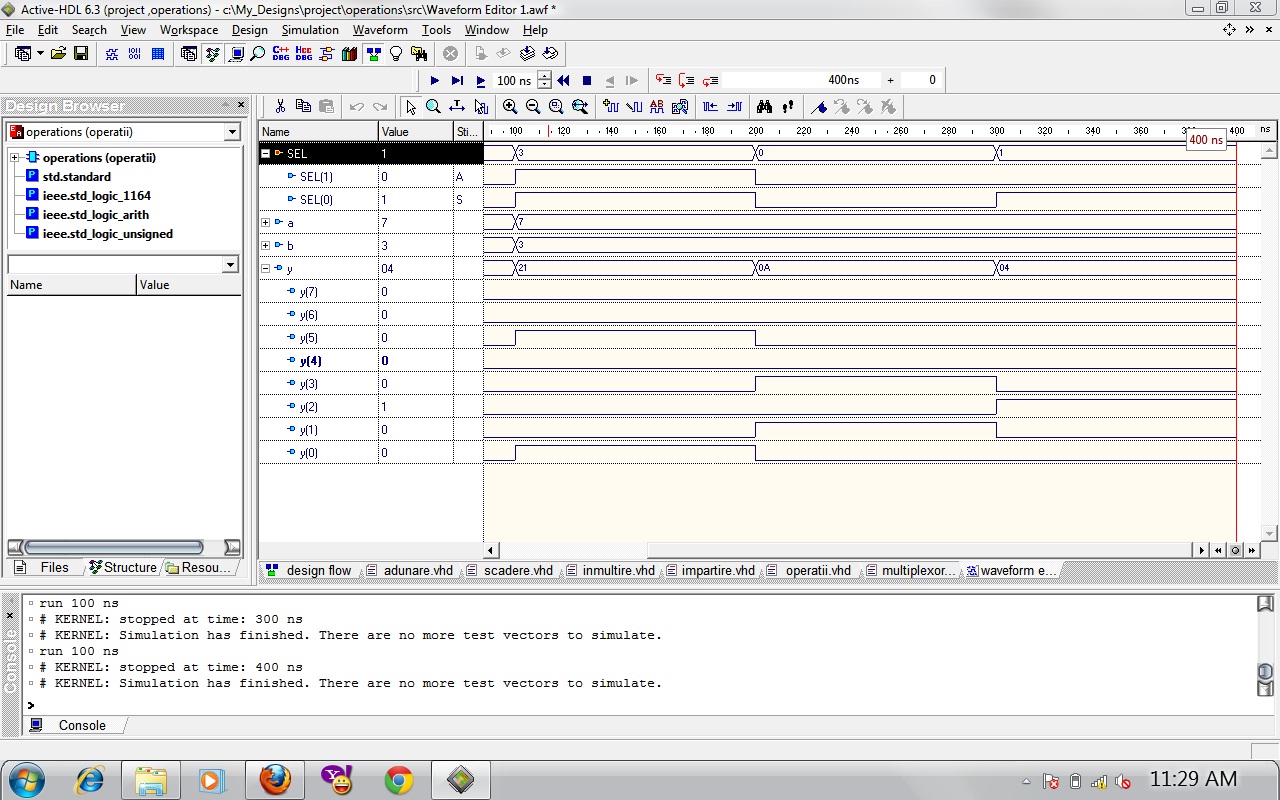
*end process;*

*end architecture;*

**Diagrama block:**



**Exemplificare waveform:**

****

**Variabile:**

* s-a folosit variabila Z pentru a face tranzitia de la 4 biti la 8 biti.

1. **Inmultirea**

* operatia reprezinta inmultirea dintre 2 numere pe patru biti. Aceasta operatie se realizeaza prin metoda adunarii repetate.

**Sintaxa componentei:**

*component inmultire is*

*port (SEL: in std\_logic\_vector (1 downto 0);*

*a,b: in std\_logic\_vector (3 downto 0);*

*y: out std\_logic\_vector (7 downto 0));*

*end component;*

*architecture multiplication of inmultire is*

*begin*

*process(SEL,a,b)*

*variable s0,s1,s2,s3: std\_logic\_vector (7 downto 0);*

*begin*

*if(SEL="10") then*

*y <= "00000000"; s0 := "00000000"; s1 := "00000000"; s2 := "00000000"; s3 := "00000000";*

*s0(0) := b(0) and a(0); s0(1) := b(0) and a(1); s0(2) := b(0) and a(2); s0(3) := b(0) and a(3);*

*s1(1) := b(1) and a(0); s1(2) := b(1) and a(1); s1(3) := b(1) and a(2); s1(4) := b(1) and a(3);*

*s2(2) := b(2) and a(0); s2(3) := b(2) and a(1); s2(4) := b(2) and a(2); s2(5) := b(2) and a(3);*

*s3(3) := b(3) and a(0); s3(4) := b(3) and a(1); s3(5) := b(3) and a(2); s3(6) := b(3) and a(3);*

*y <= s0 + s1 + s2 + s3;*

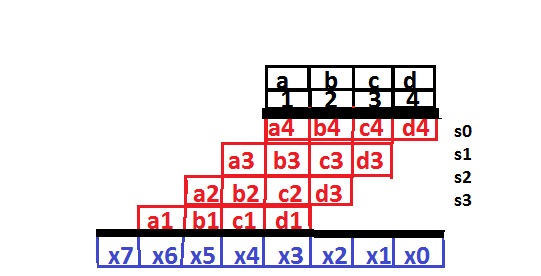
*end if;*

*end process;*

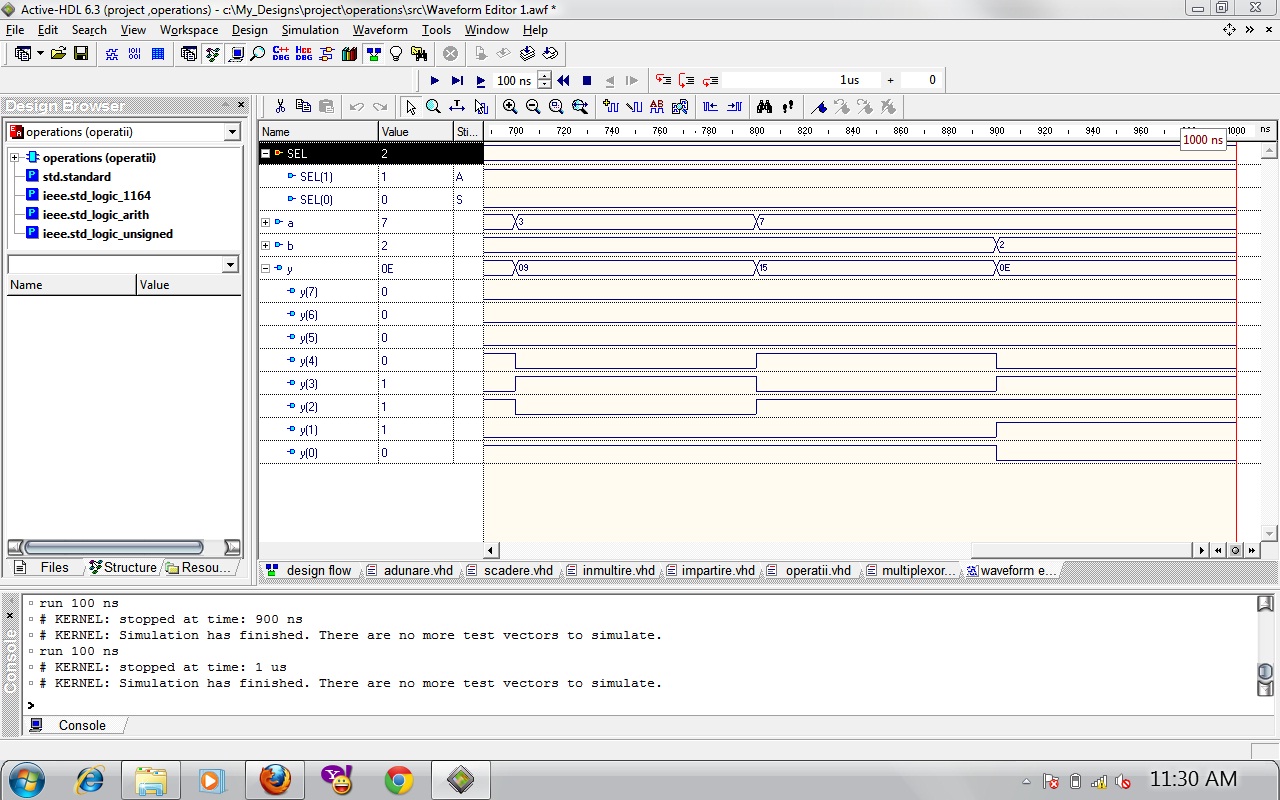
*end architecture;*

**Exemplificare prin desen:**

* Exemplu de inmultire pe 4 biti:



**Exemplificare waveform:**

****

**Variabile utilizate:**

* S-au folosit urmatoarele variabile: s0,s1,s2,s3;

**Descrierea procesului:**

* Abcd si 1234 reprezinta cele 2 numere scrise binar. Astfel in variabila s0 de tip std\_logic\_vector se vor memora in primele 4 pozitii , s0(0) … s0(3) inmultirea dintre cifra 4 a numarului 2 cu toate cifrele primului numar. In variabila s1 de tip std\_logic\_vector se vor memora in pozitiile 2,3,4,5 , s0(1) … s0(4) inmultirea dintre cifra 3 a numarului 2 cu toate cifrele primului numar. In variabila s2 de tip std\_logic\_vector se vor memora in pozitiile 3,4,5,6 , s0(2) … s0(5) inmultirea dintre cifra 2 a numarului 2 cu toate cifrele primului numar. In variabila s3 de tip std\_logic\_vector se vor memora in pozitiile 3,4,5,6 , s0(2) … s0(5) inmultirea dintre cifra 3 a numarului 1 cu toate cifrele primului numar. Iar in final cei 4 vectori vor fi adunati pentru a rezulta inmultirea dintre cele 2 numere.

1. **Impartirea**

* operatia reprezinta impartirea dintre 2 numere pe patru biti. Aceasta operatie se realizeaza prin metoda scaderii repetate. De remarcat faptul ca in cazul impartirii la 0 aceasta operatie nu este posibila iar in final toate ledurile vor fi aprinse in semn de eroare.

**Sintaxa componentei:**

*component impartire is*

*port (SEL: in std\_logic\_vector (1 downto 0);*

*a,b: in std\_logic\_vector (3 downto 0);*

*y: out std\_logic\_vector (7 downto 0));*

*end component;*

*architecture division of impartire is*

*begin*

*process(SEL,a,b)*

*variable rest,cat: std\_logic\_vector (3 downto 0);*

*variable z: std\_logic\_vector (7 downto 0);*

*variable k:integer;*

*begin*

*if(SEL="11") then*

*if(b /= "0000") then*

*z:= "00000000";*

*cat := "0000";*

*rest := "0000";*

*z := z+a;*

*for k in 1 to 16 loop*

*if (z >= b) then*

*z := z-b;*

*cat := cat + '1';*

*end if;*

*end loop;*

*rest := z(3 downto 0);*

*y(7 downto 4) <= cat;*

*y(3 downto 0) <= rest;*

*else*

*y <="11111111";*

*end if;*

*if (a<b) then*

*y(7 downto 4) <= "0000";*

*y(3 downto 0) <= a;*

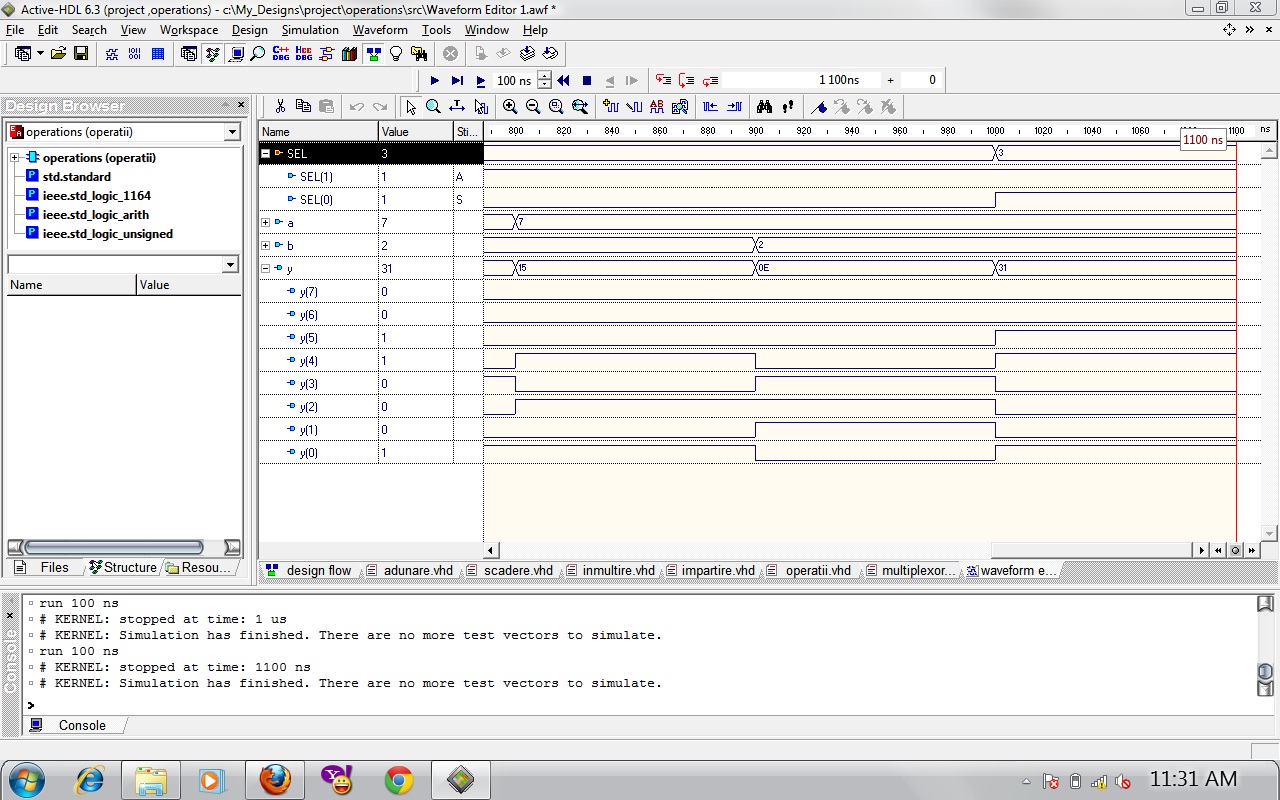
*end if;*

*end if;*

*end process;*

*end architecture;*

**Exemplficare waveform:**

****

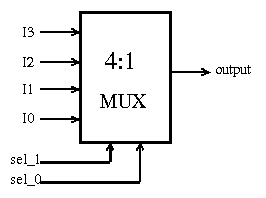
**Descrierea procesului:**

* pentru a realiza impartirea s-a utilizat metoda scaderii repetate. Aceasta scadere se realizeaza intr-o strcutura repetitive for, atat timp cat numarul impartit este mai mare sau egal cu numarul la care se imparte. Primele 3 leduri vor reprezenta restul impartirii in binar, ledul al 4- lea va reprezenta un spatiu intre partea intreaga si cea fractionara, urmand ca ultimele 4 leduri sa reprezinte catul in binar.

**Variabile utilizate:**

* S-au folosit urmatoarele variabile: rest,cat,z;
  + - **Multiplexorul**
* s-a apelat la utilizarea unui multiplexor 4:1 care realizeaza selectarea output-ului corespunzator fiecarei operatii, in functie de selectia initiala.

**Diagrama block:**



**Sintaxa componentei:**

*component multiplexor is*

*port ( SEL: in std\_logic\_vector (1 downto 0);*

*a,b,c,d: in std\_logic\_vector (7 downto 0);*

*y: out std\_logic\_vector (7 downto 0));*

*end component;*

*architecture multiplexer of multiplexor is*

*begin*

*process(SEL,a,b,c,d)*

*begin*

*case SEL is*

*when "00" => y <= a;*

*when "01" => y <= b;*

*when "10" => y <= c;*

*when "11" => y <= d;*

*when others =>*

*end case;*

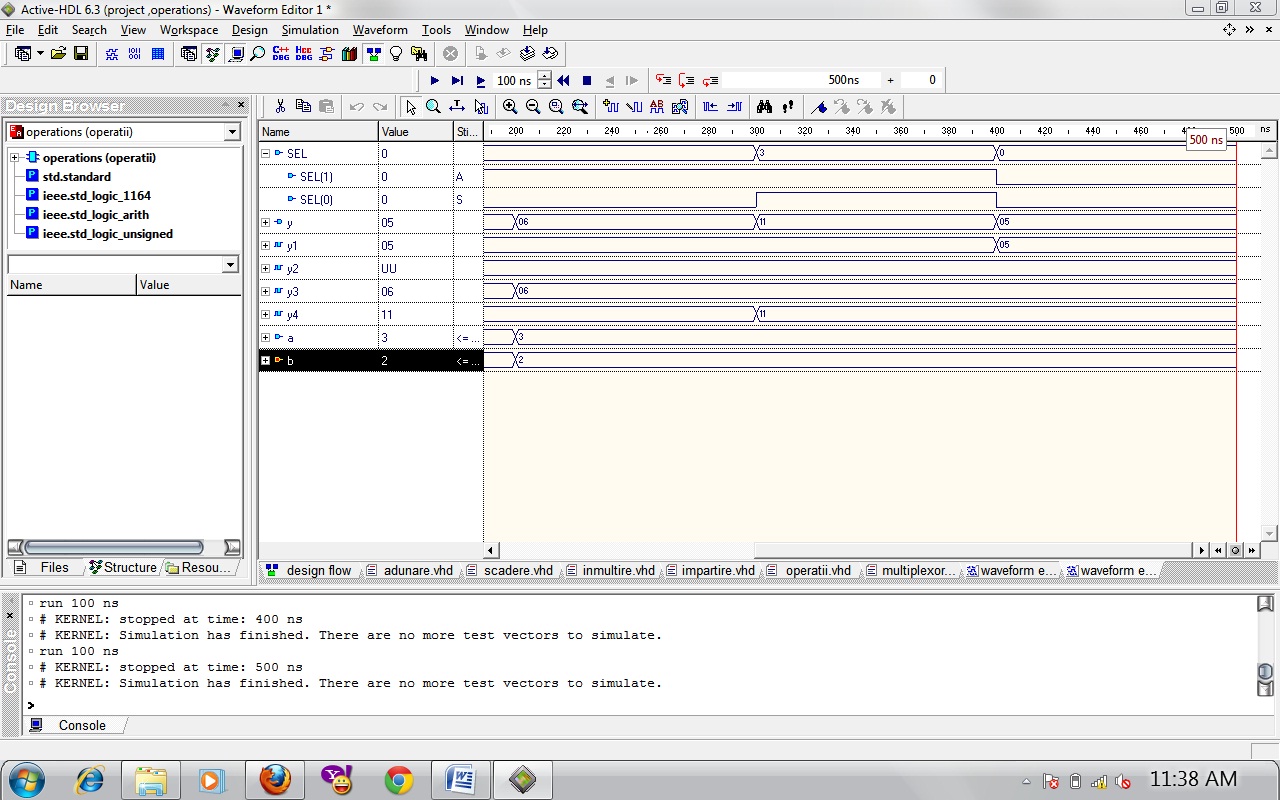
*end process;*

*end architecture;*

**Descrierea procesului:**

* s-a folosit un multiplexor de tip 4:1 cu selectie pe 2 biti intrucat avem 4 operatii, iar selectia necesara este pe 2 biti. Astfel in modul 00 multiplexorul va selecta rezultatul adunarii, in 01 rezultatul scaderii, in 10 inmultirea, iar in 11 impartirea.

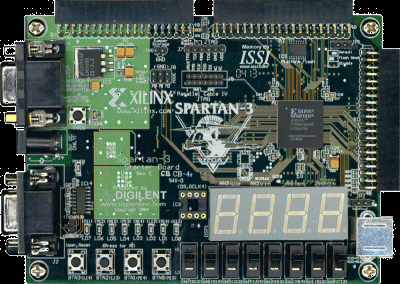
**Exemplficare waveform:**

****

**Variabile folosite:**

* S-au folosit urmatoarele variabile: y1,y2,y3,y4. Aceste variabile reprezinta rezultatul operatiilor, urmand ca in functie de selectie sa fie afisat cel dorit.

**Descrierea pinilor (family Spartan 3E)**

****

**Vectorul A:** *NET "a<0>" LOC = "P11" ; NET "a<1>" LOC = "L3" ; NET "a<2>" LOC = "K3"; NET "a<3>" LOC = "B4" ;*

**Vectorul B:** *NET "b<0>" LOC = "G3" ; NET "b<1>" LOC = "F3" ; NET "b<2>" LOC = "E2" ;*

*NET "b<3>" LOC = "N3" ;*

**Selectia:** *NET "SEL<0>" LOC = "A7" ; NET "SEL<1>" LOC = "M4" ;*

**Vectorul Y:** *NET "y<0>" LOC = "M5" ; NET "y<1>" LOC = "M11" ; NET "y<2>" LOC = "P7" ; NET "y<3>" LOC = "P6" ; NET "y<4>" LOC = "N5" ; NET "y<5>" LOC = "N4" ; NET "y<6>" LOC = "P4" ; NET "y<7>" LOC = "G1"*

**Posibilitati de dezvoltare ulterioara**

Posibilitatea de dezvoltare a calculatorului de buzunar este una vasta. Dimensiunea numerelor se poate mari la mai multi biti, iar numarul de functii poate fi de asemenea crescut. Proiectul ofera posibilitatea de a adauga oricat de multe functii, modificand doar dimensiunea selectiei si cea a multiplexorului.

**Justificarea solutiei alese**

Solutia aleasa reprezinta creare proiectului din componente elementare pentru a fi usor de inteles si de utilizat. Astfel pentru adunare si scadere s-au folosit unitatile elementare de baza a VHDL-ului pentru aceaste operatii, respective Full Adder si Full Subtractor. Pentru cele 2 operatii mai complicate am decis sa le impart deasemenea in operatii elementare (adunare si scadere). Astfel s-au evitat operatorii “\*”,”/” care nu sunt sintetizabili pe placuta FPGA. Am apelat la adunari repetate pentru inmultire si scaderi repetate pentru impartire. Multiplexorul a fost ales pentru a realize selectia operatiei intr-un mod cat mai simplu.