

**Calculator de buzunar cu operații aritmetice fundamentale**

**Facultatea de Automatică și Calculatoare**

**Specializare: Calculatoare si Tehnologia Informației**

**Disciplina: Proiectarea Sistemelor Numerice**

**Profesor coordonator: Dr. Ing. Lișman Florin Dragoș**

**Student: Pop Crina-Maria**

**Grupa: 30218**

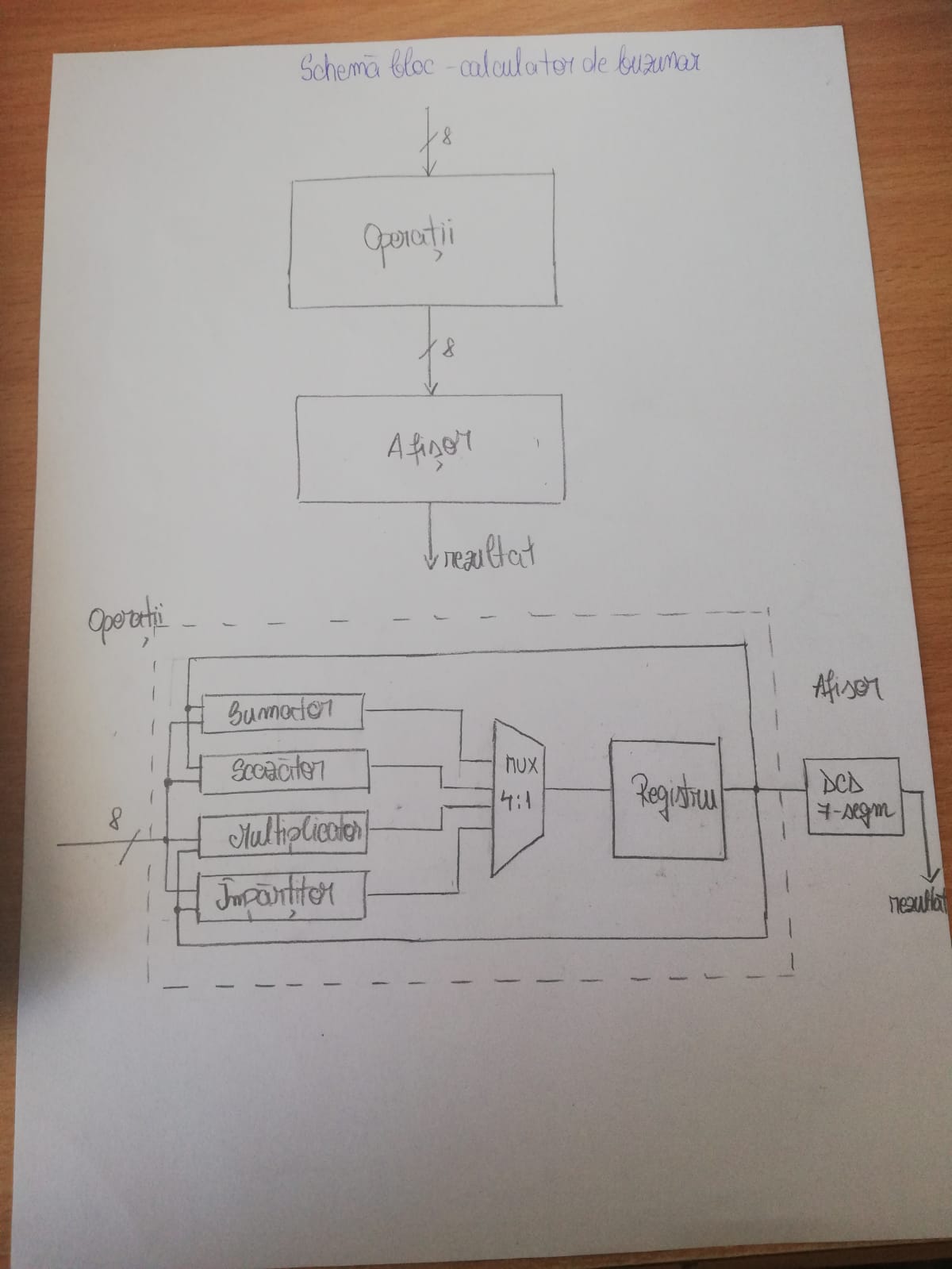
**Cuprins**

1. Specificație proiect...................................................................................2
2. Schema bloc..............................................................................................2
3. Explicații........................................................................................3
4. Proiectare și implementare......................................................................3
   1. Schema de detaliu........................................................................3
   2. Proiectarea componentelor în VHDL...........................................4
   3. Simularea componentelor...........................................................14
   4. Fișierul de constrângeri și placa FPGA........................................19
5. Lista componentelor utilizate.................................................................21
6. Justificarea soluției alese........................................................................21
7. Utilizare și rezultate................................................................................22
8. Posibilități de dezvoltare ulterioară......................................................23

**1.Specificație proiect**

Să se proiecteze un calculator de buzunar cu operații aritmetice fundamentale (adunare, scădere, înmulțire, împărțire). Operațiile de înmulțire și îmărțire 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.

**2.Schema bloc**



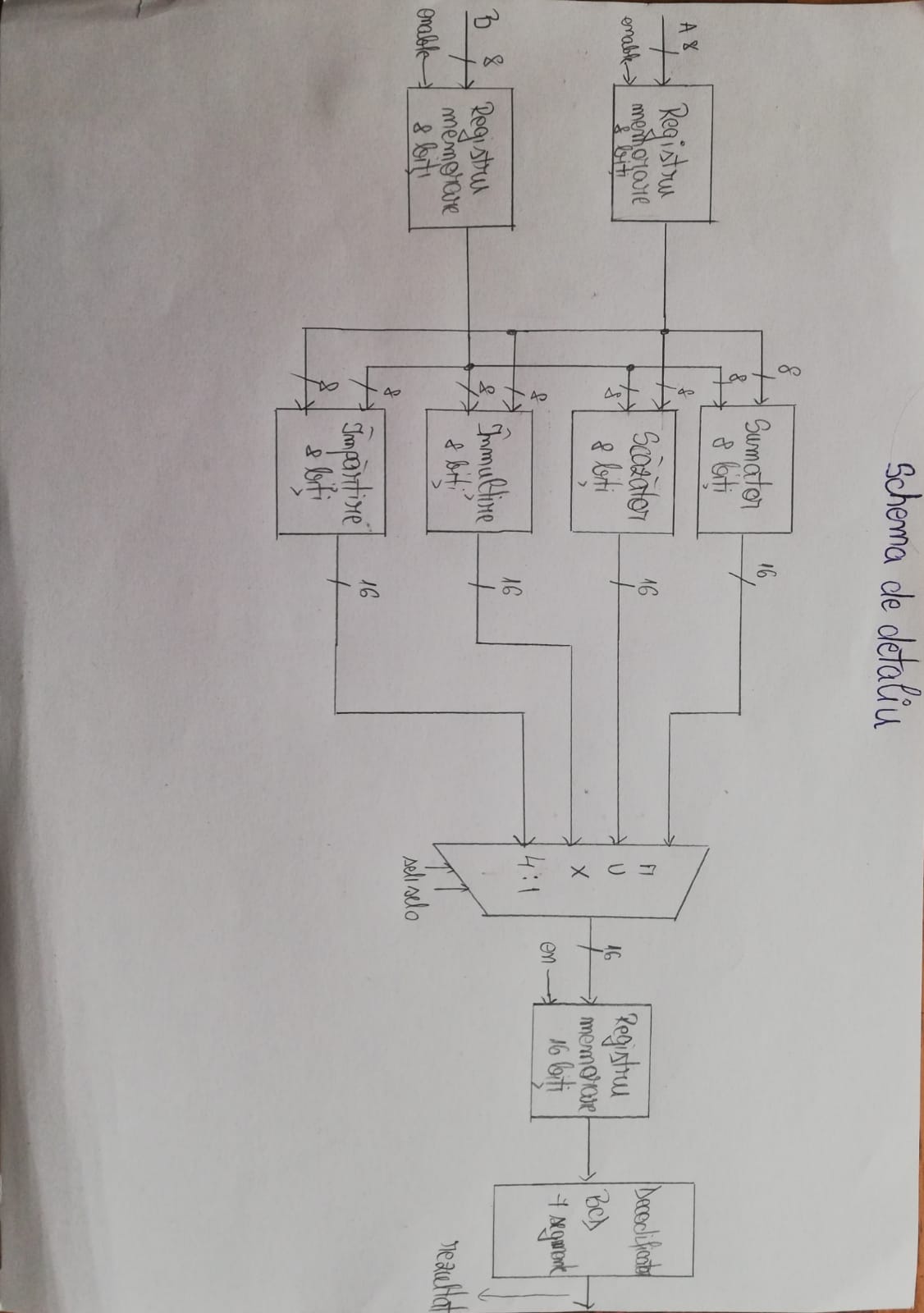
**2.1 Explicații**

Schema bloc a calculatorului de buzunar prezintă în partea de sus a paginii 2 cutii negre: cutia de operații în care se vor realiza operațiile fundamentale și afișorul care va arăta rezultatele.

În partea de jos a paginii am dezvoltat puțin schema și am aratat faptul că înăuntrul cutiei de operații avem nevoie de alte 4 cutii care realizează cele 4 operații fundamentale (adunare, scădere, înmulțire și împărțire) ale căror rezultate vor trece într-un multiplexor, iar în funcție de selecțiile acestuia, unul dintre rezultate va fi memorat într-un registru pentru ca mai apoi să fie afișat.

**3.Proiectare și implementare**

**3.1 Schema de detaliu**

****

**Descrierea schemei de detaliu**

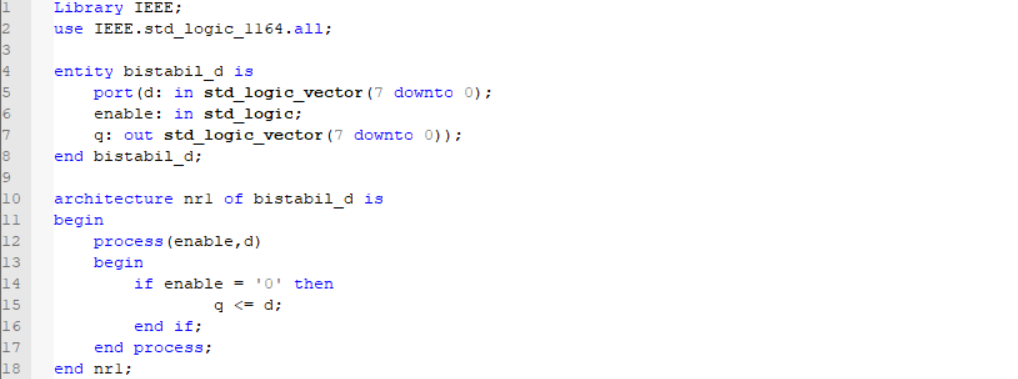
Am dezvoltat schema de detaliu a proiectului meu după schema bloc, încercând să păstrez aceeași idee de proiectare. După cum se poate observa în imagine, numerele asupra cărora vrem să realizăm operații sunt introduse în două registre de memorare (reprezentate de niște bistabile de tip D) pe 8 biți, iar în funcție de enable (corespondentul clock-ului) care va fi activ pe 0 pentru primul registru și activ pe 1 pentru al doilea, numerele vor fi scoase și introduse pe rând în componentele pentru realizarea operațiilor. Pentru adunare am ales un sumator complet pe 8 biți, respectiv pentru scădere un scăzător complet pe 8 biți. Referitor la înmulțire și împărțire, așa cum se specifică și în cerința proiectului, ele nu pot fi implementate cu operatorii ”\*” și ”/”. De aceea, am realizat două componente pentru aceste operații înăuntrul cărora se realizează algoritmii acestora (pentru înmulțire adunări repetate, iar pentru împărțire scăderi repetate). Rezultatele de la toate operațiile vor fi reprezentate pe 16 biți, întrucat m-am gândit că la înmulțirea a doua numere pe 8 biți va rezulta unul de mărime dublă, adică 16 biți și pentru a lucra cu aceeași dimensiune a numerelor am ales să fie toate rezultatele pe 16 biți pentru a putea fi introduse în multiplexor.

În multiplexorul 4:1 sunt introduse rezultatele pe 16 biți, iar cele 2 selecții vor decide care dintre rezultate să treacă mai departe. Rezultatul selectat va fi introdus într-un alt registru de memorare, de această dată pe 16 biți, și acesta va fi afișat pe afișoarele decodificatorului BCD-7 segmente daca en este 1.

* 1. **Proiectarea componentelor în VHDL**

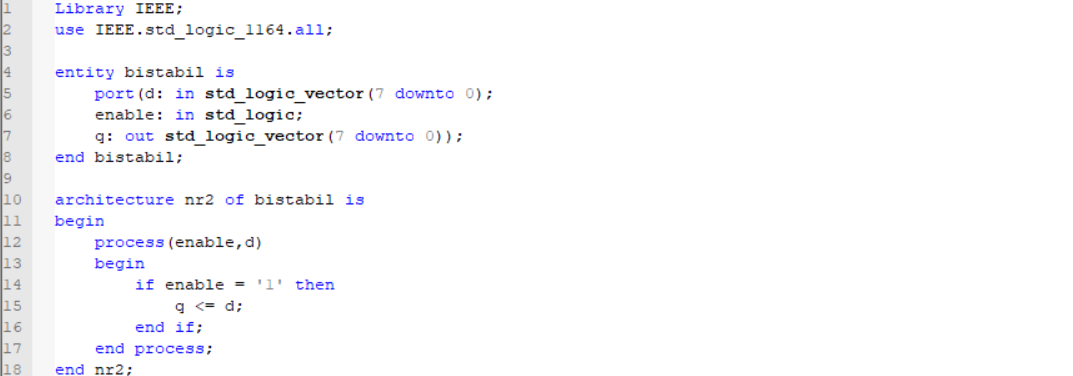
**Codul componentelor folosite**

1. *Registru memorare pentru primul număr*



În entitatea registrului am declarat intrările: d (numărul pe 8 biți care intră în registru), enable (în loc de clk; dacă enable are valoarea ‘0’, numărul va apărea la ieșire) și iesirea q (acest semnal primește valoarea intrării d).

1. *Registru memorare pentru al doilea număr*



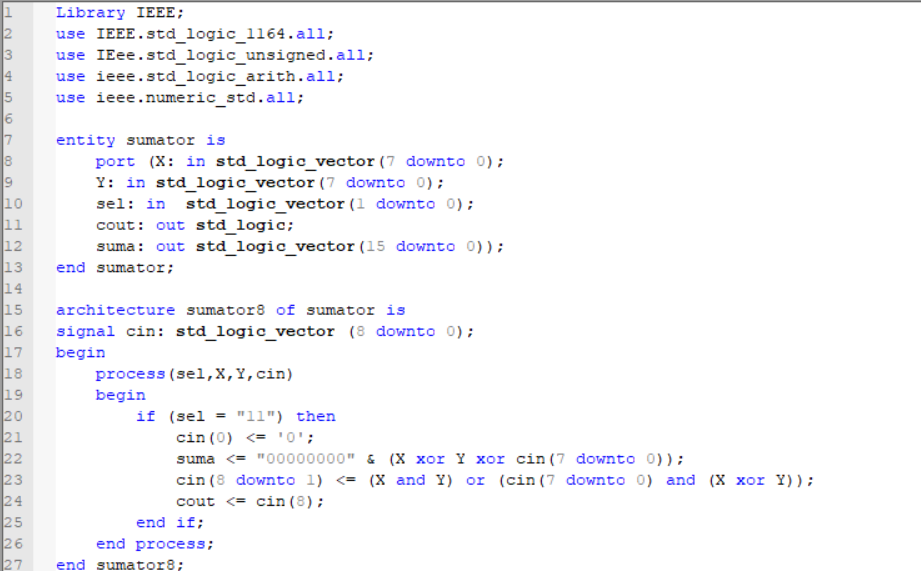
La fel ca primul registru, acesta are aceleași intrări și iesiri, deosebirea putând fi observată la nivelul arhitecturii în procesul care depinde de enable. Ieșirea q va primi valoarea intrării d doar dacă enable este ‘1’.

1. *Registru memorare pentru rezultat*



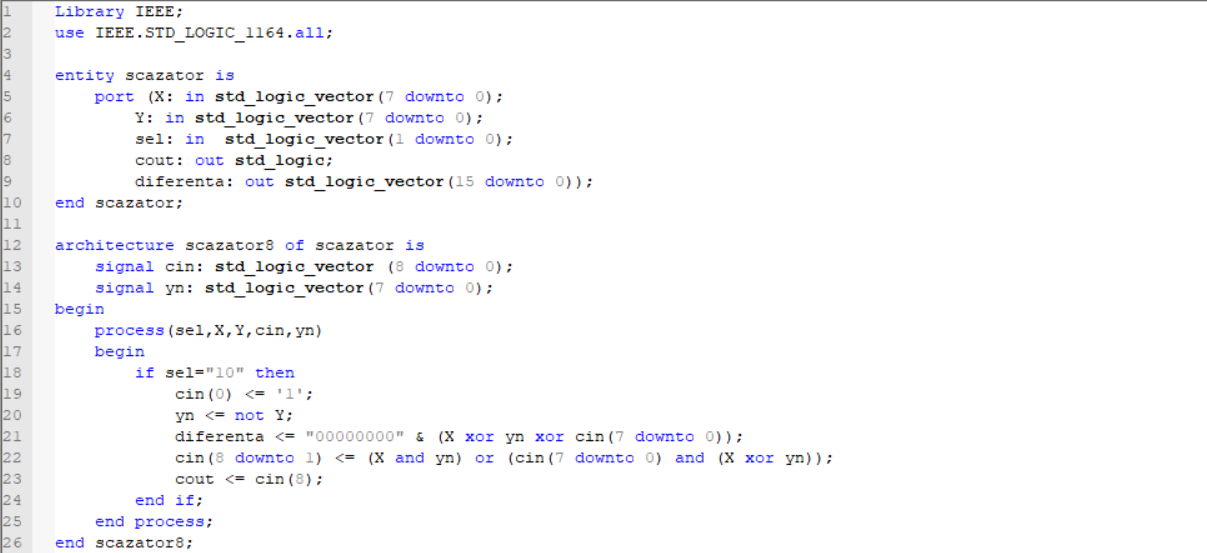
Acest registru este similar cu cele prezentate mai sus, diferența constând în faptul că acesta este un registru de memorare pe 16 biți în care vor fi ținute rezultatele (intrarea d este declarată în entitate corespunzător cu acest lucru), iar enable-ul de aici are altă denumire, en, fiind un semnal diferit. Rezultatul va fi scos din registru doar când en este ‘1’.

1. *Sumator complet 8 biți*



În entitatea sumatorului am decalarat următoarele intrări: X și Y- cele două numere pe 8 biți, sel- selecția pe 2 biți a multiplexorului, cout-carry out-ul sumatorului complet, reprezentând depășirea care poate apărea la cel mai semnificativ bit, și ieșirea suma-reprezentată pe 16 biți unde va fi reținut rezultatul adunării. În arhitectură am declarat un semnal intermediar cin care este un vector de 9 biți și reprezintă transportul de la rangul anterior către următorul rang (dacă la poziția *i* se adună cei doi biți ai numerelor X și Y și se genereză un transport, suma biților de la rangul *i+1* este egală cu suma biților de la acest rang ai numerelor X și Y + bitul de la rangul anterior). Procesul sumatorului depinde de intrări si de acest vector de biți de transport, iar dacă selecția este “11” (așa am ales să fie codificată adunarea) se realizează operația pe baza ecuațiilor logice ale unui sumator complet. Ultimul bit al vectorului cin este de fapt transportul care ne arată o depășire și i se atribuie ieșirii cout. Faptul că am concatentat la partea mai semnificativă a sumei 8 biți de 0 nu influnțează cu nimic rezultatul (el va fi tot pe 8 biți).

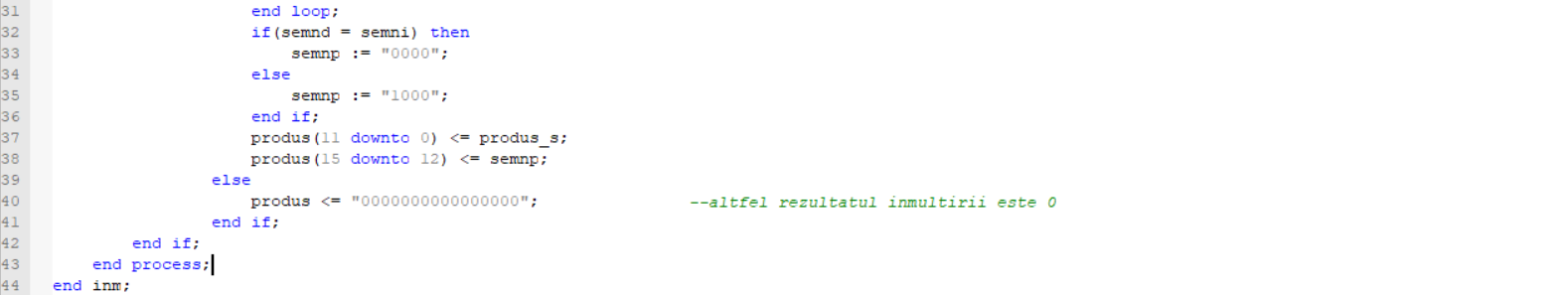
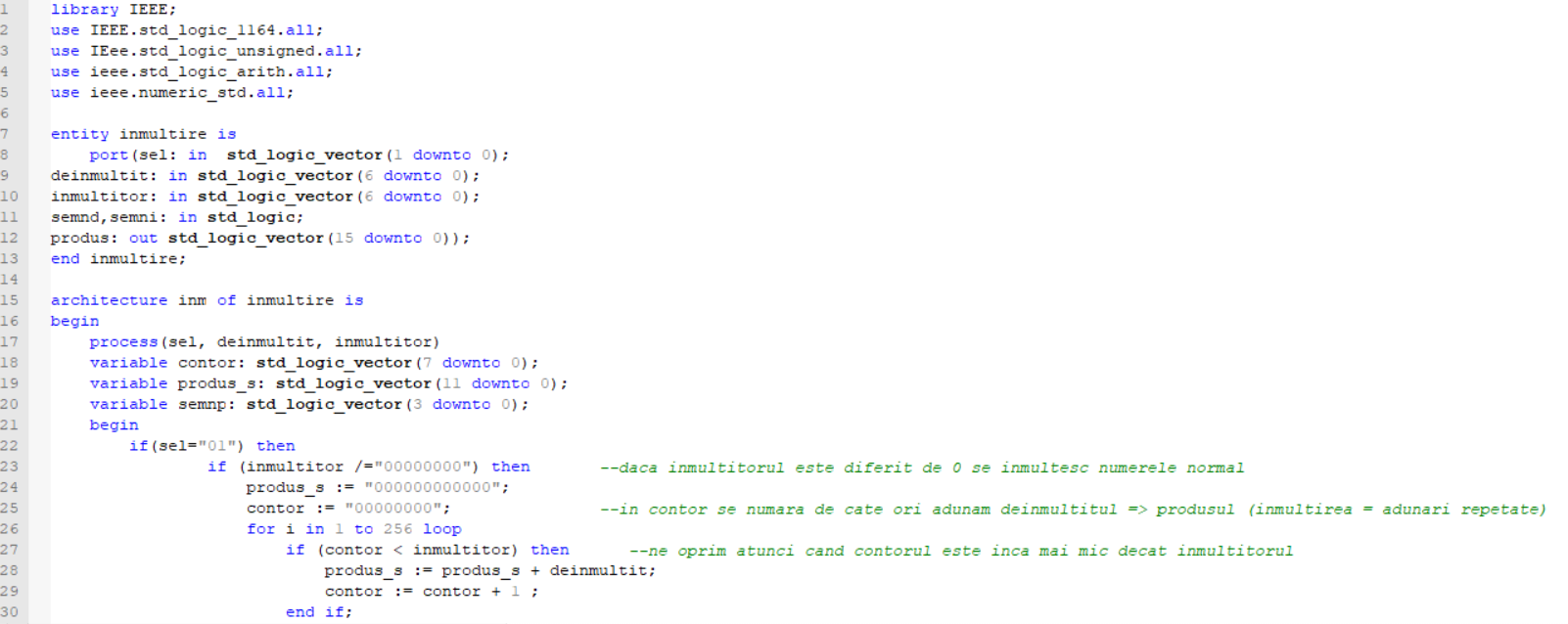
1. *Scăzător complet 8 biți*



Similar cu entitatea sumatorului, cea a scăzătorului are aceleași intrări și ieșiri, doar că semnalul de ieșire nu se mai numește suma, ci diferența, care este tot pe 16 biți. În schimb, arhitectura scăzătorului diferă puțin, întrucăt scăderea înseamnă adunarea cu complementul față de 2. Pentru a obține complementul față de 2 al celui de-al doilea număr, luăm un semnal intermediar yn, care va lua valoarea negată a lui Y. Însă complementul față de 2 înseamnă negarea numărului și adunarea unui 1 la numărul negat. De aceea, primul bit al vectorului de cin (ca și la sumator, acesta va reține împrumuturile) este inițializat cu ‘1’. Ecuațiile logice sunt aceleași cu ale sumatorului, doar că lucrăm cu complementul față de 2 al celui de-al doilea număr. Selecția pentru scădere am ales-o să fie ”10”. La fel ca la sumator, concatenarea celor 8 de 0 nu influențează cu nimic rezultatul.

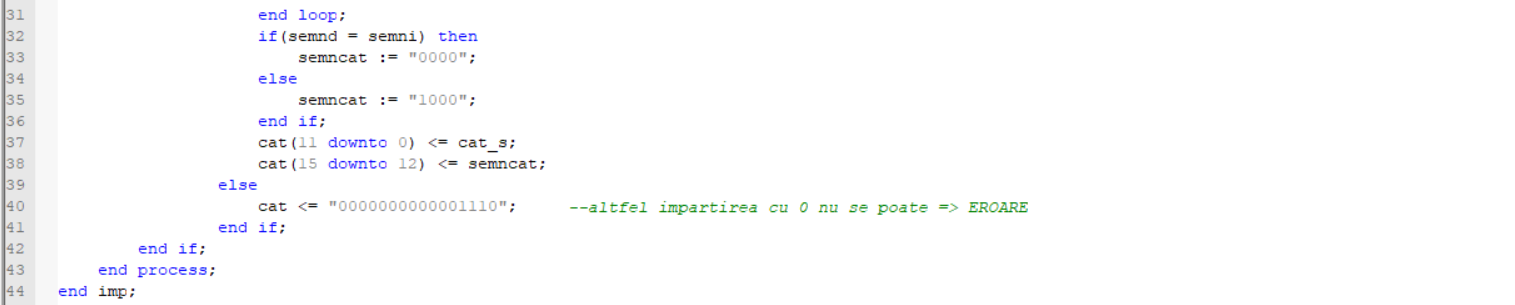
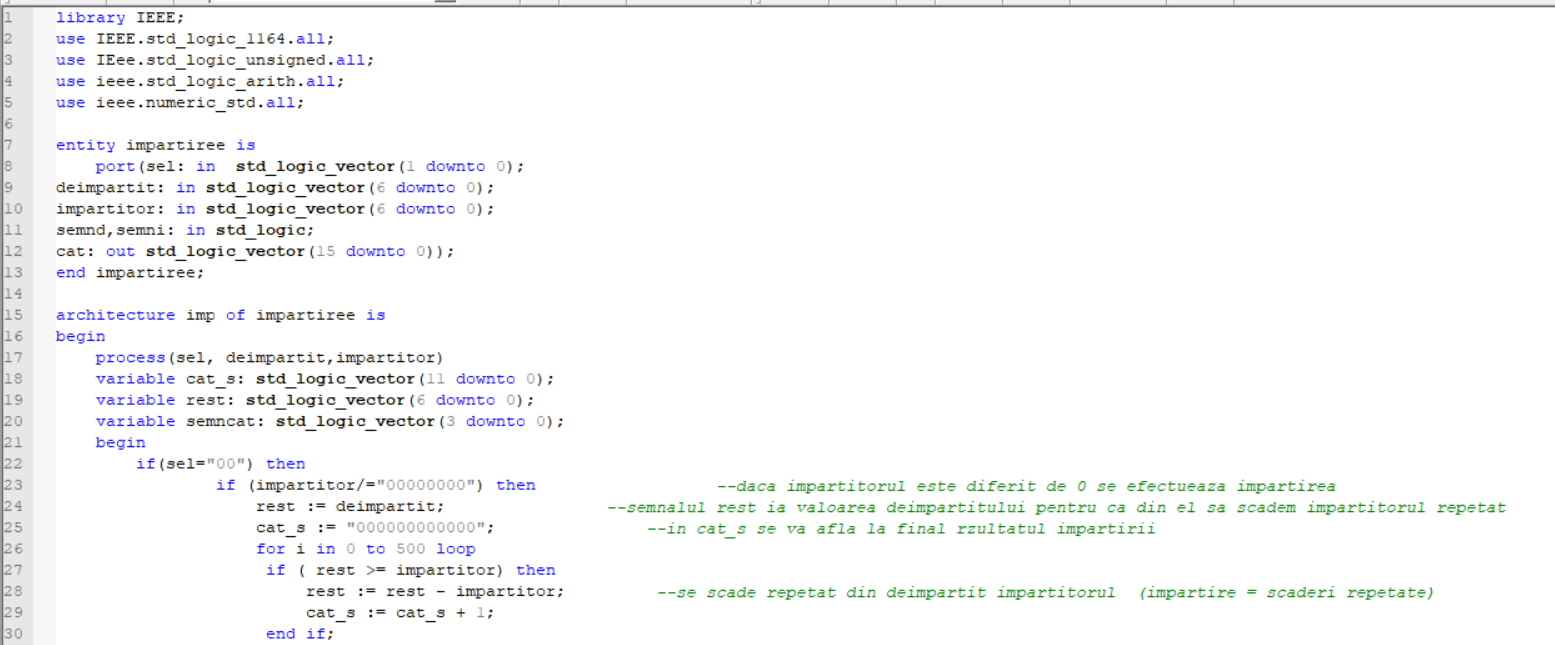
1. *Înmulțire*

Componenta pentru înmulțire are declarate în entitate intrările: sel-pentru a selecta înmulțirea, deinmultitul și inmultitorul- cele 2 numere pe 7 biți+semnd și semni- semnele celor 2 numere și ieșirea produs- rezultatul înmulțirii. Procesul din arhiectură depinde de selecție, deinmultit si inmultitor (intrările din entitate). Am declarat trei variabile intermediare contor, produs\_s și semnp pentru a putea calcula produsul și rezultatul acestuia. Acest lucru se face prin adunări repetate. Astfel, dacă selecția sel este ”01” și dacă inmultitorul este diferit de 0, atunci inițializam variabila produs cu valoarea 0 și variabila contor tot cu 0 (va fi un contor care va număra câte adunări s-au făcut pentru a ne putea opri când contorul este mai mic decât inmultitor). Am ales instrucțiunea repetitivă for și am pus ca valoare de oprire numărul 256 (2^8 este 255, dar noi am pornit de la 1 și se vor face atâtea adunări cât este necesar. În variabila produs se va afla la finalul buclei rezultatul înmulțirii noastre. Dacă inmultitorul este 0 se va asigna direct valoarea 0 prdusului Semnul rezultatului se calculează cu o instrucțiune if (dacă semnele sunt egale, semncat va fi ”0000”, iar în cazul contrar va fi ”1000”, așa cum se întâmplă și în realitate).

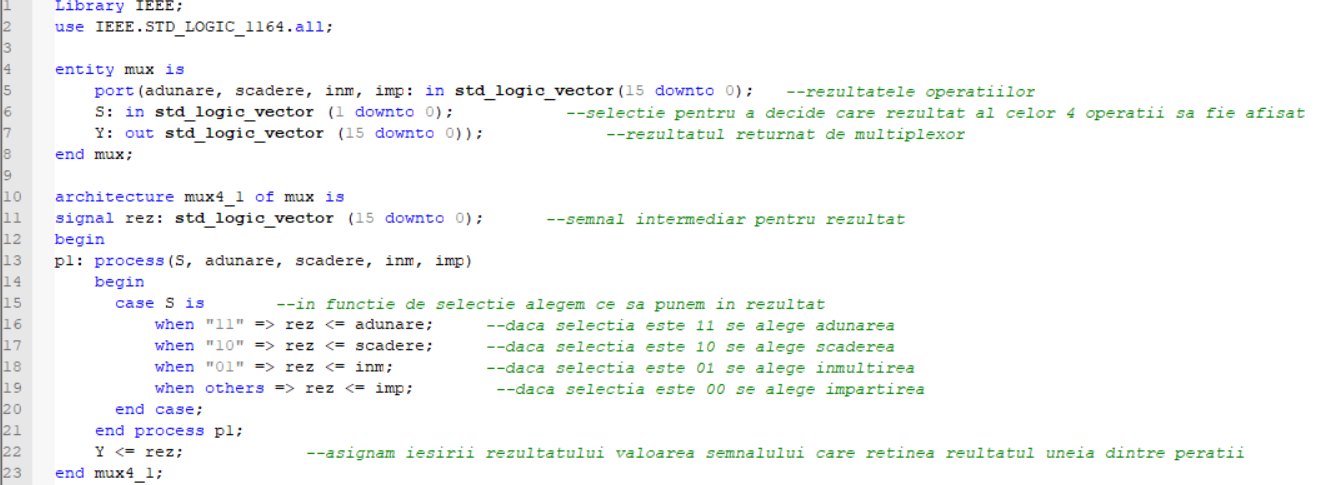


1. *Împărțire*

Componenta pentru împărțire are declarate în entitate intrările: sel-pentru a selecta împărțirea, deimpartitul și impartitorul- cele 2 numere pe 7 biți+ semnd și semni(8 biți în total)- semnele celor 2 numere și ieșirea cat- rezultatul împărțirii. Procesul din arhiectură depinde de selecție, deimpartit și impartitor (intrările din entitate). Am declarat trei variabile intermediare cat\_s, rest\_s, semncat pentru a putea calcula câtul, restul și semnul rezultatului. Acest lucru se face prin scăderi repetate, așa cum am mai spus. Astfel, dacă selecția sel este ”00” și dacă impartitorul este diferit de 0, atunci inițializam variabila rest cu valoarea deimpartitului și variabila cat\_s cu 0 (va fi ca un contor care va număra câte scăderi s-au făcut și acela va fi rezultatul împărțirii). Am ales instrucțiunea repetitivă for și am pus ca valoare de oprire un număr foarte mare, ca să fiu sigură că realizează toate scăderile, iar dacă restul în care am pus valoarea deimpartitului este încă mai mare decât impartitorul se efectuează scăderea. Variabila rest nu a fost întâmplător denumită așa, în ea va rămâne la final restul împărțirii, dar noi nu vom avea nevoie de el, pentru că lucrăm cu întregi. Dacă impartitorul este 0 se va semnala o eroare, întrucât împărțirea cu 0 nu se poate. Semnul rezultatului se calculează cu o instrucțiune if (dacă semnele sunt egale, semncat va fi ”0000”, iar în cazul contrar va fi ”1000”, așa cum se întâmplă și în realitate).

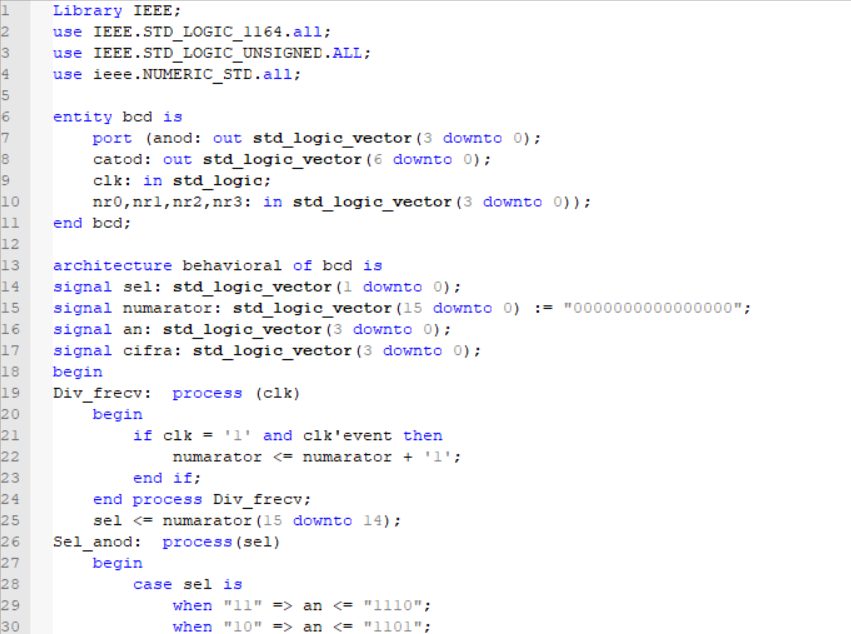


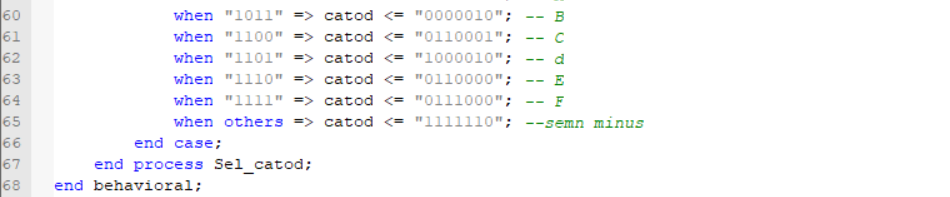
1. *Multiplexor 4:1*



Multiplexorul are ca intrări rezultatele operațiilor rezlizate, care în funcție de selecția sel vor fi lăsate să treacă sau nu. Fiecare operație este codificată și anume: pentru adunare alegem ”11”, pentru scădere alegem ”10”, pentru înmulțire alegem ”01” și pentru împărțire ”00”.

1. *Decodificator BCD-7 segmente (afișorul)*

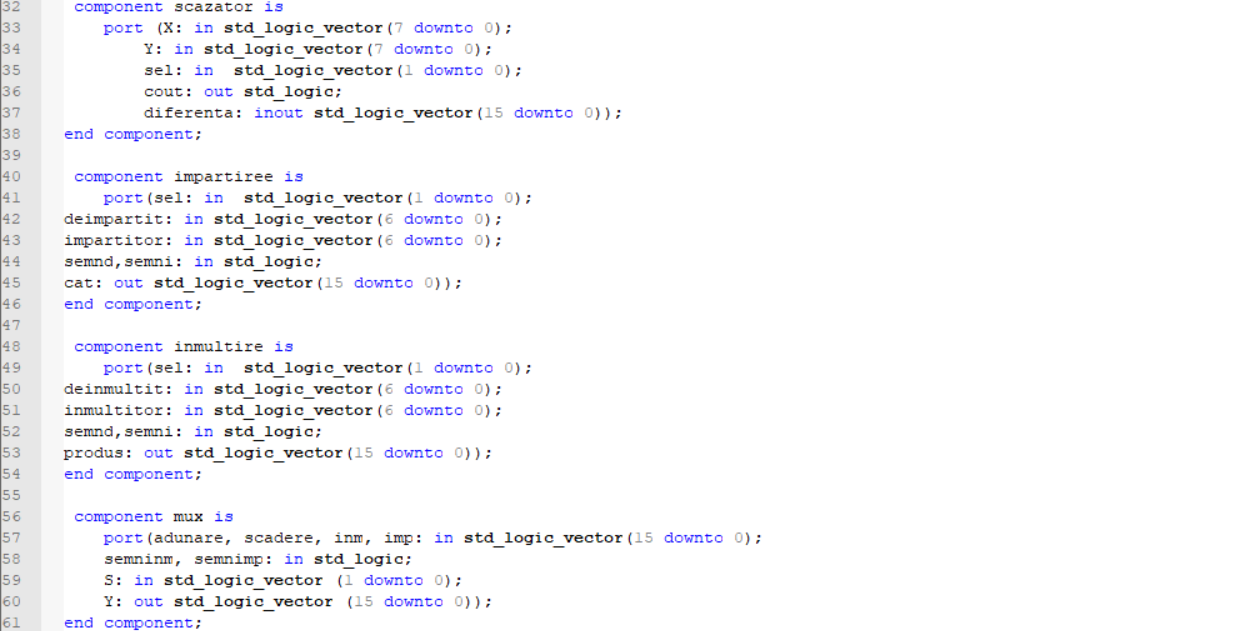


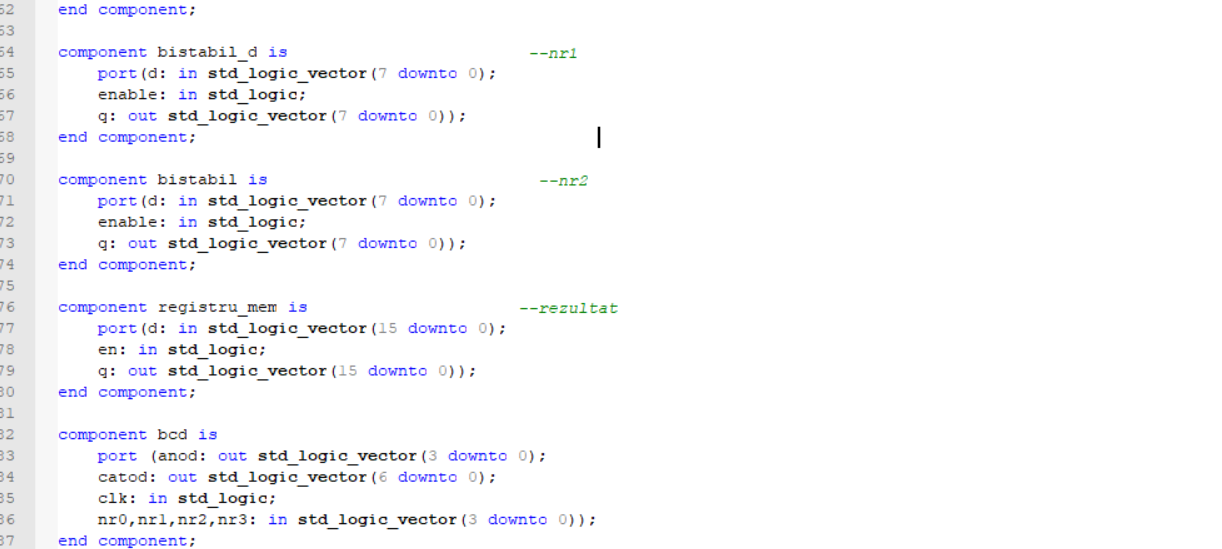


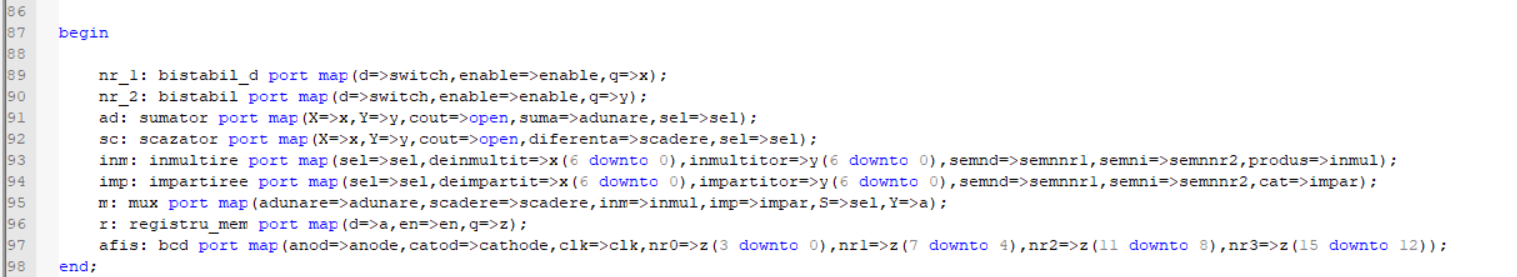
Entitate afișorului are ca semnale de intrare clk, nr0, nr1, nr2, nr3 (aceste nr0-3 vor fi de fapt cifrele de afișat ale rezultatului pe cele 4 ecrane). Clk reprezintă semnalul de tact al plăcii FPGA Nexys-4 și are frecvența de 100 MHz. Întrucât imaginea de pe aceste afișoare ale plăcii este afișată foarte rapid și pentru a vedea informația de pe ecran se realizează divizarea clock-ului. În arhitectură am declarat semnal pentru selecție-sel, pentru numărător-care va diviza frecvența clock-ului, pentru anod-an și pentru cifra care va fi afișată. Sel va lua valoarea ultimilor cei mai semnificanți biți ai numărătorului și cu ajutorul acestuia vom selecta anodul pe care se afișează o cifră. Anodul și catodul sun active pe 0. Astfel, pentru anod, cand primul bit este 0 se va activa primul afișor, când al doilea bit este 0 se va activa al doilea afișor etc. Procesul care depinde de semnalul an are scopul de a alege ca primii 4 biți ai rezultatului să meargă pe primul afișor când acesta este activ, următorii 4 biți vor fi afișați pe al doilea afișor, următorii pe al treilea afișor și ultimii 4 biți vor fi afișați pe ultimul afișor. Procesul care depinde de cei 4 biți separați ai rezultatului vor activa catodul și vor apărea pe afișoare numere de la 0 la 9 și litere de la A la F (rezultatele operațiilor sunt in hexazecimal).

1. *Calculatorul de buzunar-componenta finală*









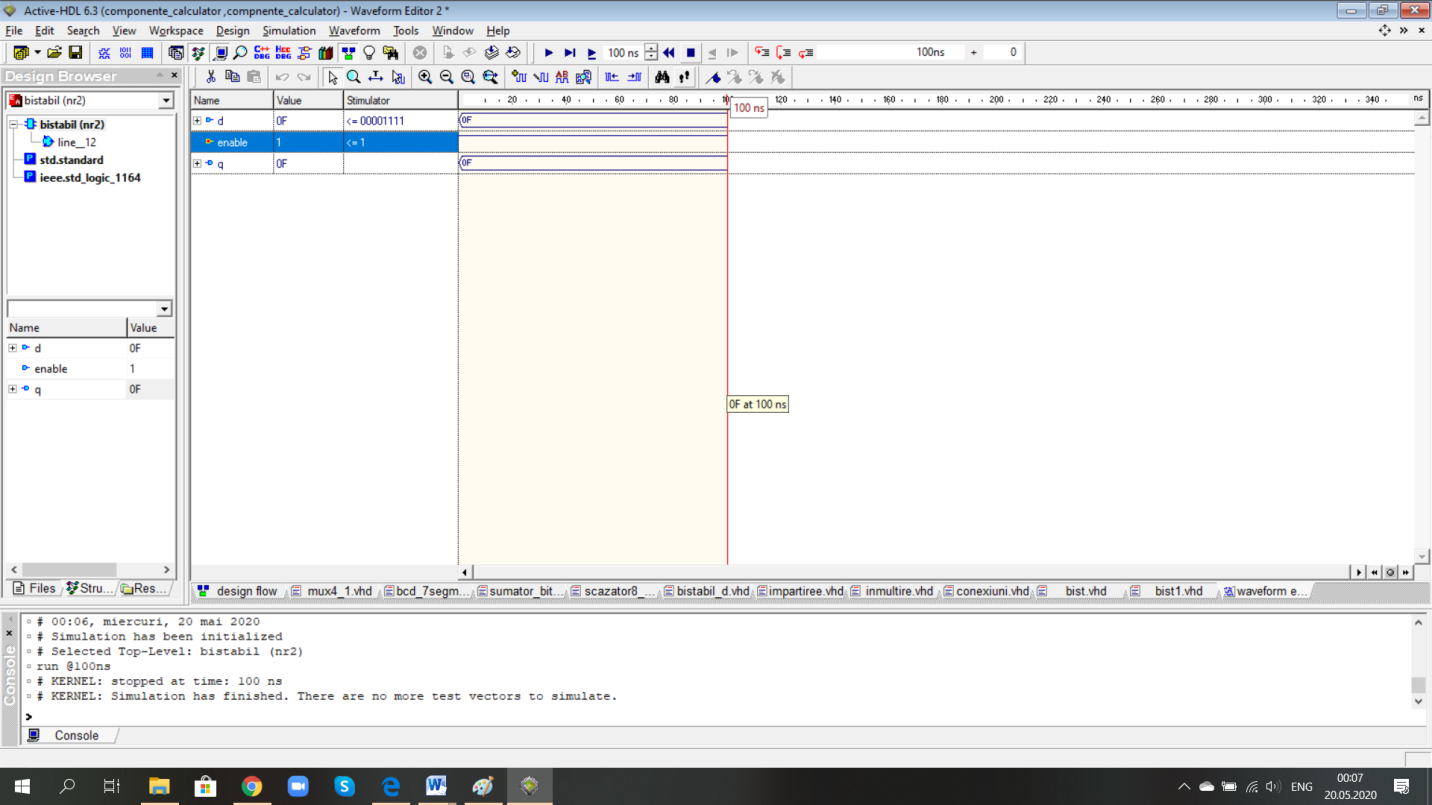
Entitatea proiectului final are declarate următoarele semnale de intrare: switch-reprezintă switch-urile de pe placa FPGA, sel-selecția care ne va da alegerea operației dorite, clk-clockul plăcii folosite, en-enable-ul necesar pentru a scoate rezultatul, enable-celălalt semnal de enable pentru a scoate numărul dorit din registru, semnnr1 și semnnr2-semnele numerelor valabile doar pentru înmulțire și împărțire și ieșirile anode și cathode pentru a controla afișoarele.

În arhitectura proiectului am declarant 2 semnale a și z pentru a prelua rezultatele finale, 2 semnale x și y pentru numerele introduse de la switch-uri, 2 semnale pentru semnele de la înmulțire și împărțire și 4 semnale adunare, scadere,impar,inmul pentru rezultatele .Înainte de begin-ul arhitecturii se declară toate componentele pe care le-am descris mai sus. Instanțierea componentelor se face prin cuvintele cheie component....end component, iar între ele se scrie tot ce este in entitatea componentei respective. Apoi, după scrierea cuvântului begin trebuie realizate conexiunile între componente prin semnalele declarate. Așa cum se poate observa în cod, la intrarea registrelor pentru numere ”legăm” switchurile astfel încat în registre să fie la final numerele (ieșirile acestora sunt legate la semnalele x, respectiv y). La toate componentele pentru operații, ca intrări vom avea numerele care acum se află în x și y, iar rezultatele de la acestea se vor pune în semnalele adunare, scadere, inmul, impar. Operațiile de inmulțire și împărțire vor avea în plus semnele numerelor ca al 8-lea bit, și partea cea mai semnificativă a rezultatului e va arăta semnul final. În multiplexor vor intra cele 4 rezultate pe 16 biți (așa cum am arătat și în schema detaliată) și în funcție de selecția sel la ieșirea multiplexorului va fi unul dintre rezultate (pe care îl punem în semnalul a. În registrul de memorare pe 16 biți a rezultatului va intra semnlul care are rezultatul de la multiplexor, a, și rezultatul final care va merge spre afișoare va fi în semnalul z. La afișor se conectează anode și cathode pentru a controla ledurile și afișoarele, clk-ul și numărul care trebuie afișat împărțit în bucăți de câte 4 biți corespunzătoare unui afișor.

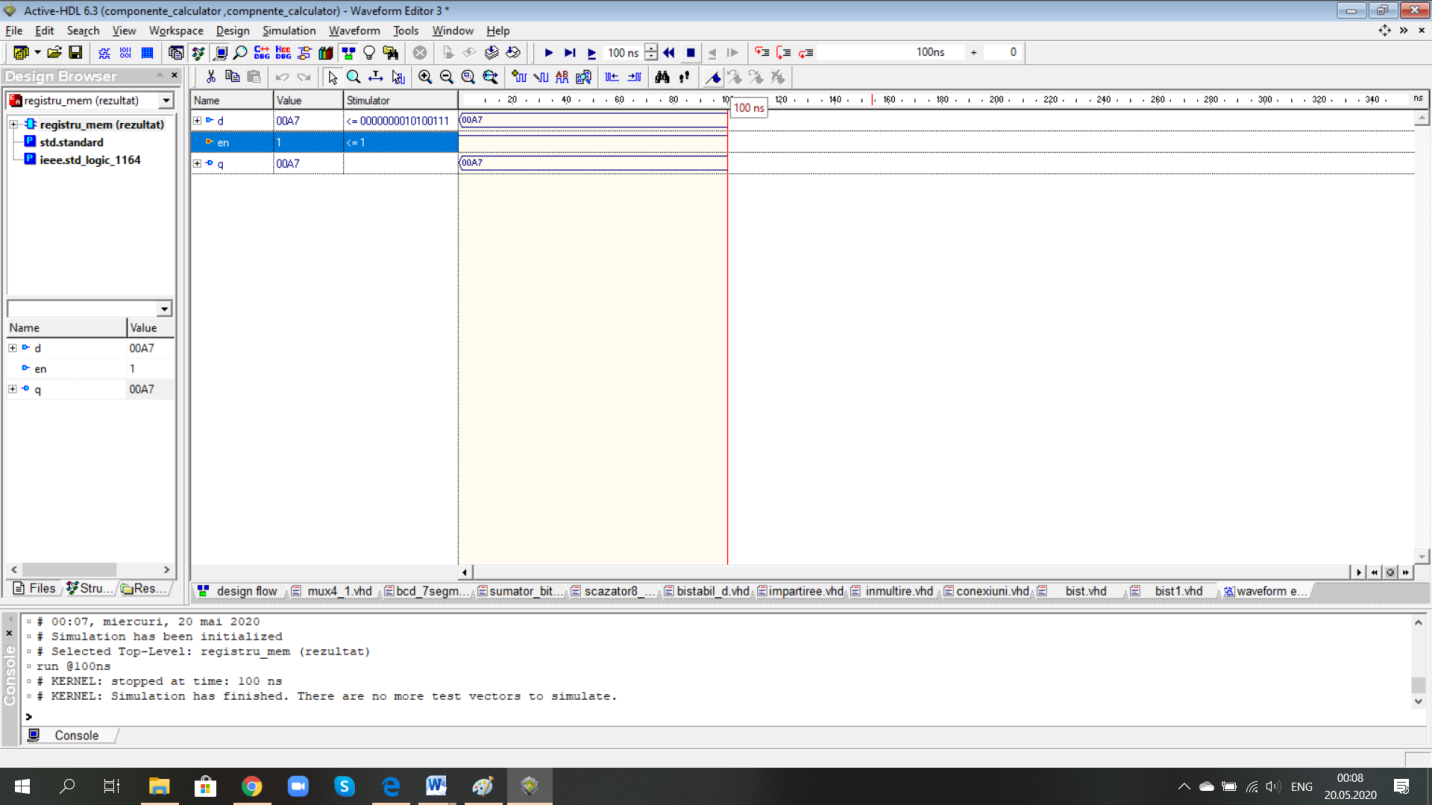
* 1. **Simularea componentelor**

****

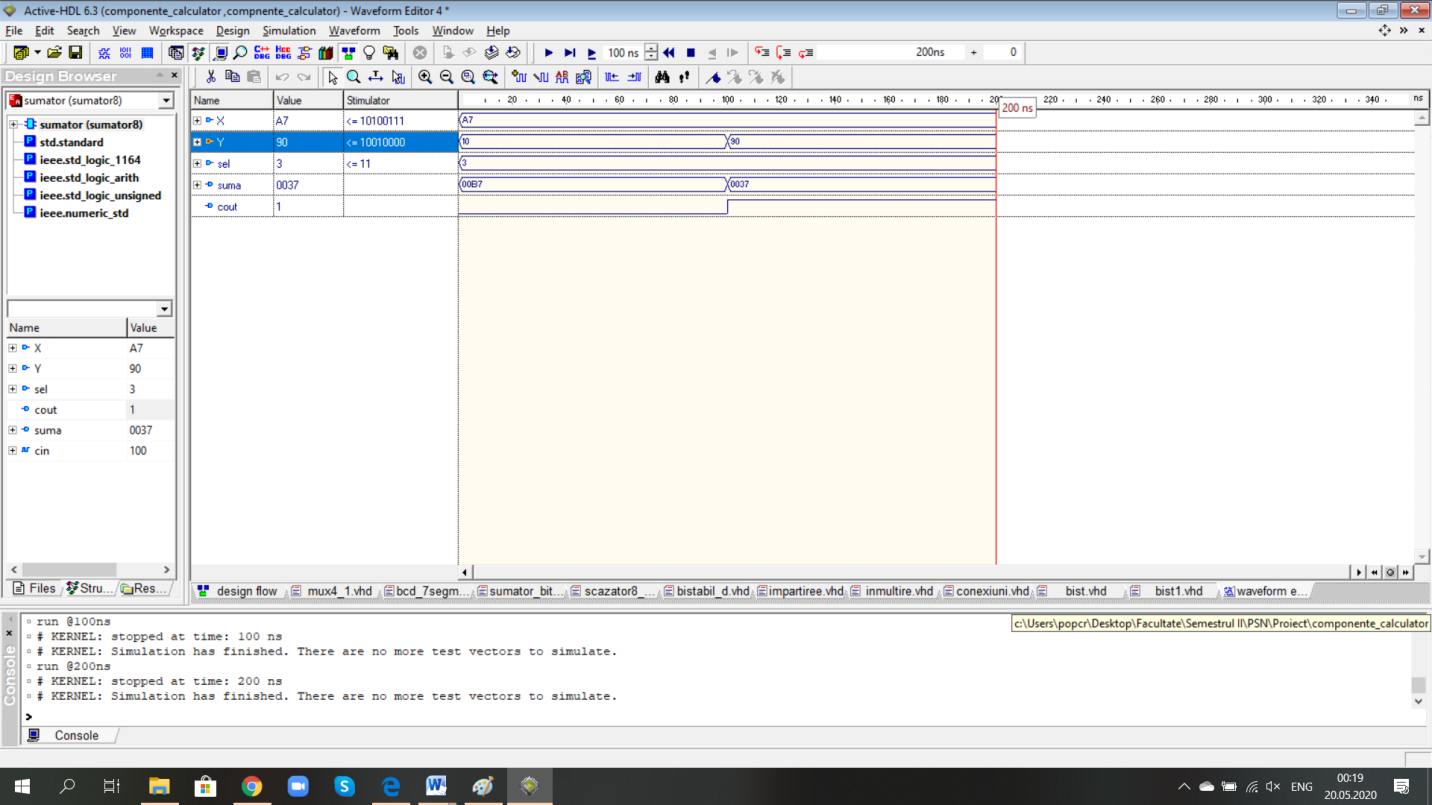
Se poate observa că registrul nostru arată la ieșire ceea ce este la intrare dacă enable este ‘0’.



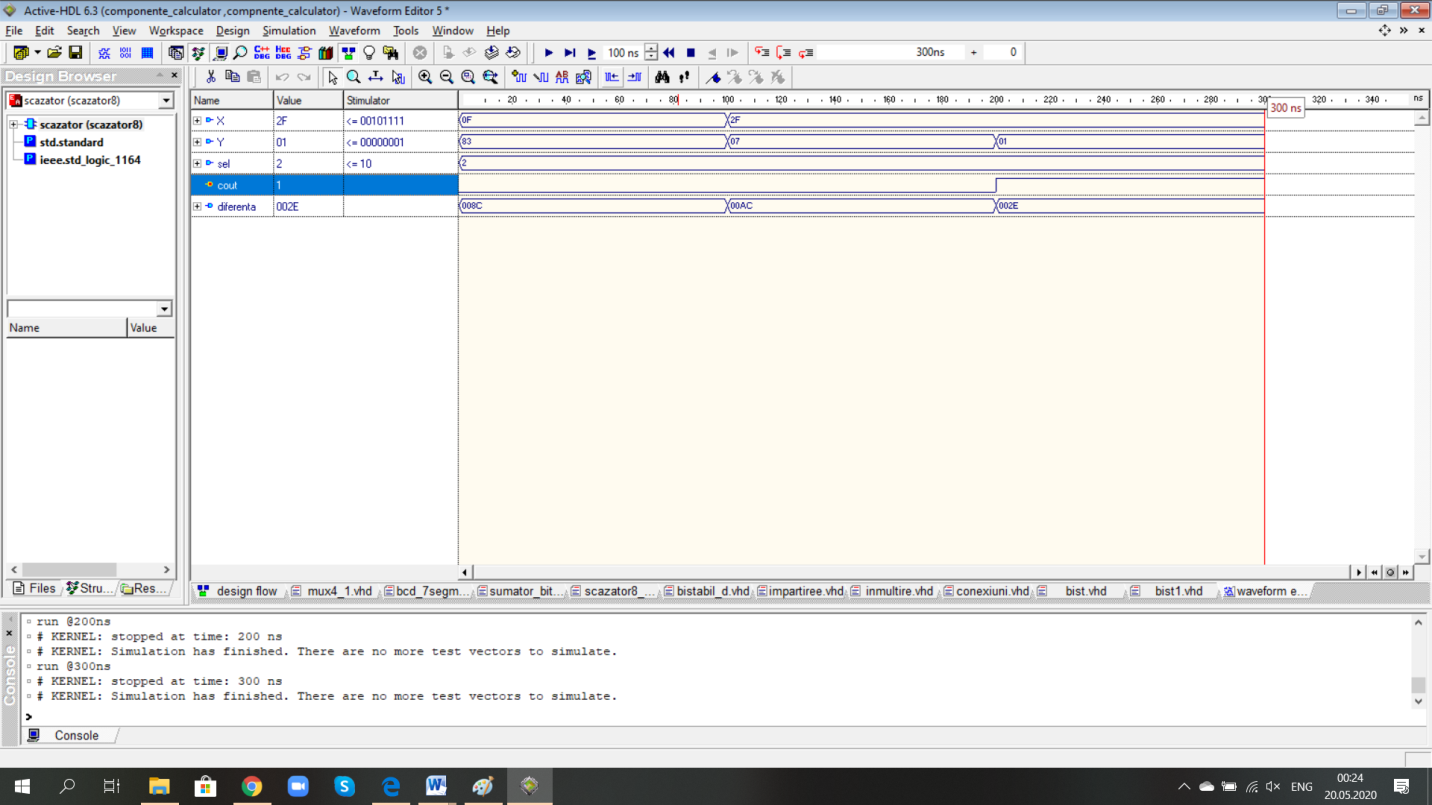
La fel și registrul pentru al doilea număr, doar când enable este ‘1’.



Registrul de memorare al rezultatului pe 16 biți funcționează de asemenea.



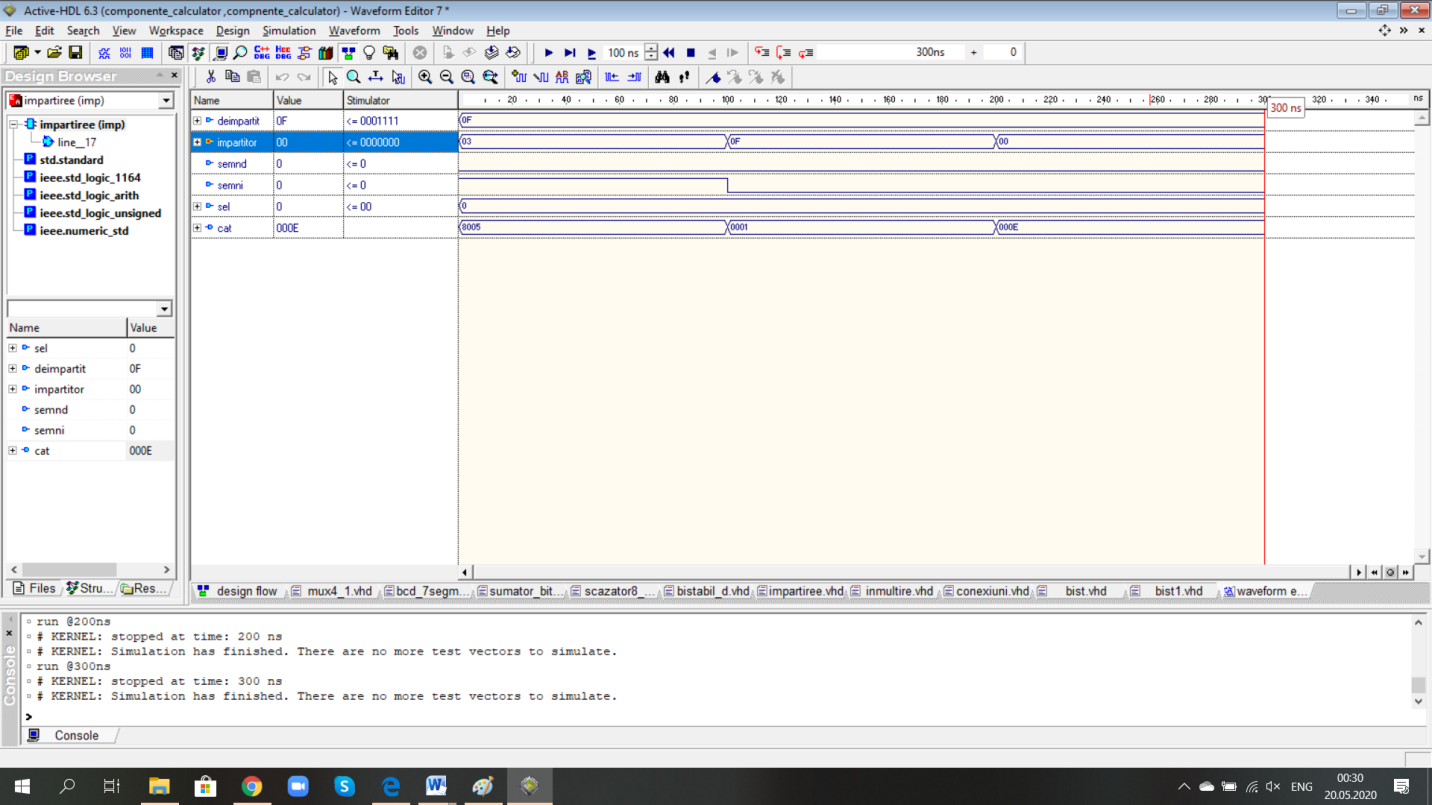
În simularea sumatorului se poate observa că adunarea este corectă, iar dacă adunăm 2 numere care depășesc valoarea limită cout este 1.



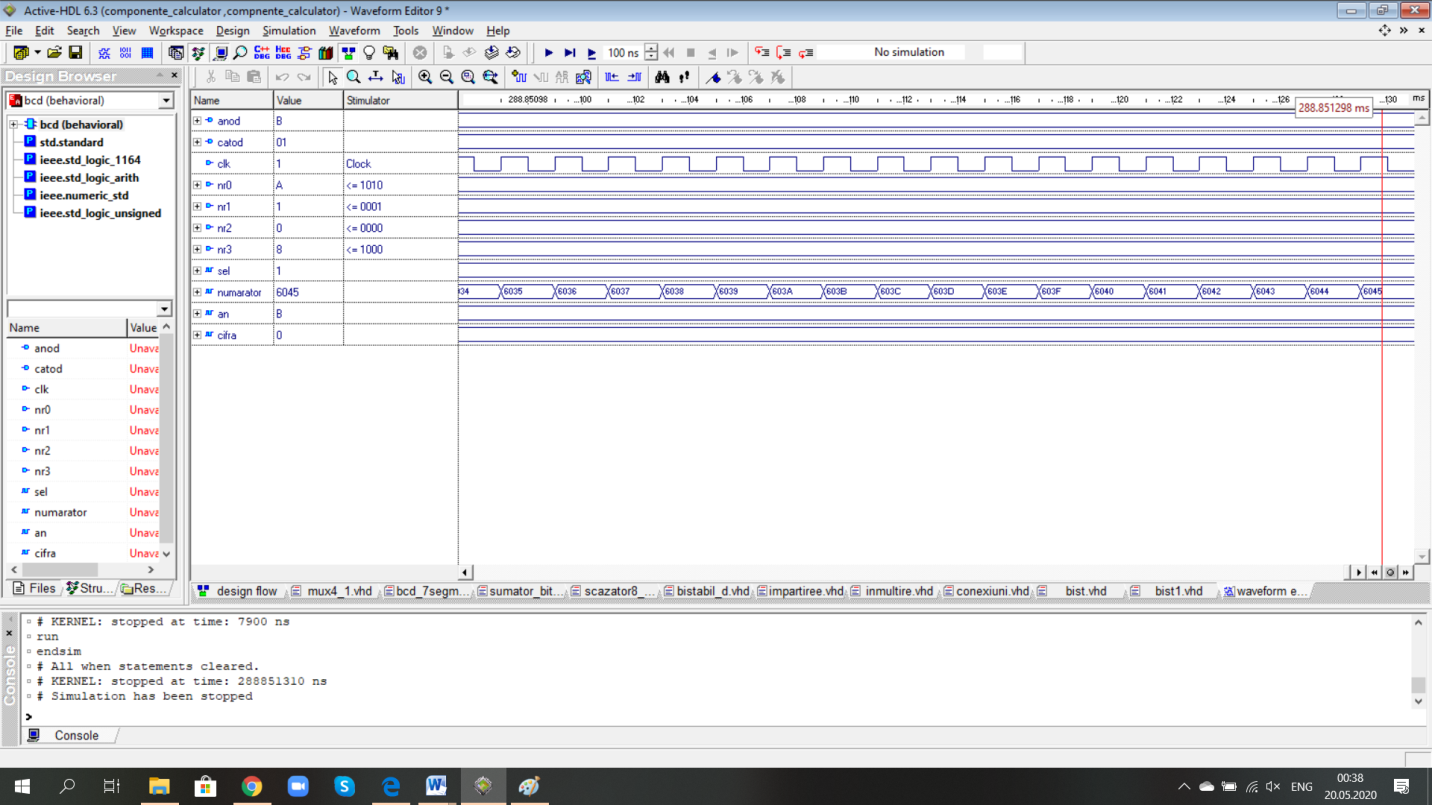
Scăzătorul nostru funcționează corect și se poate observa că scăderea se face în complement față de 2.

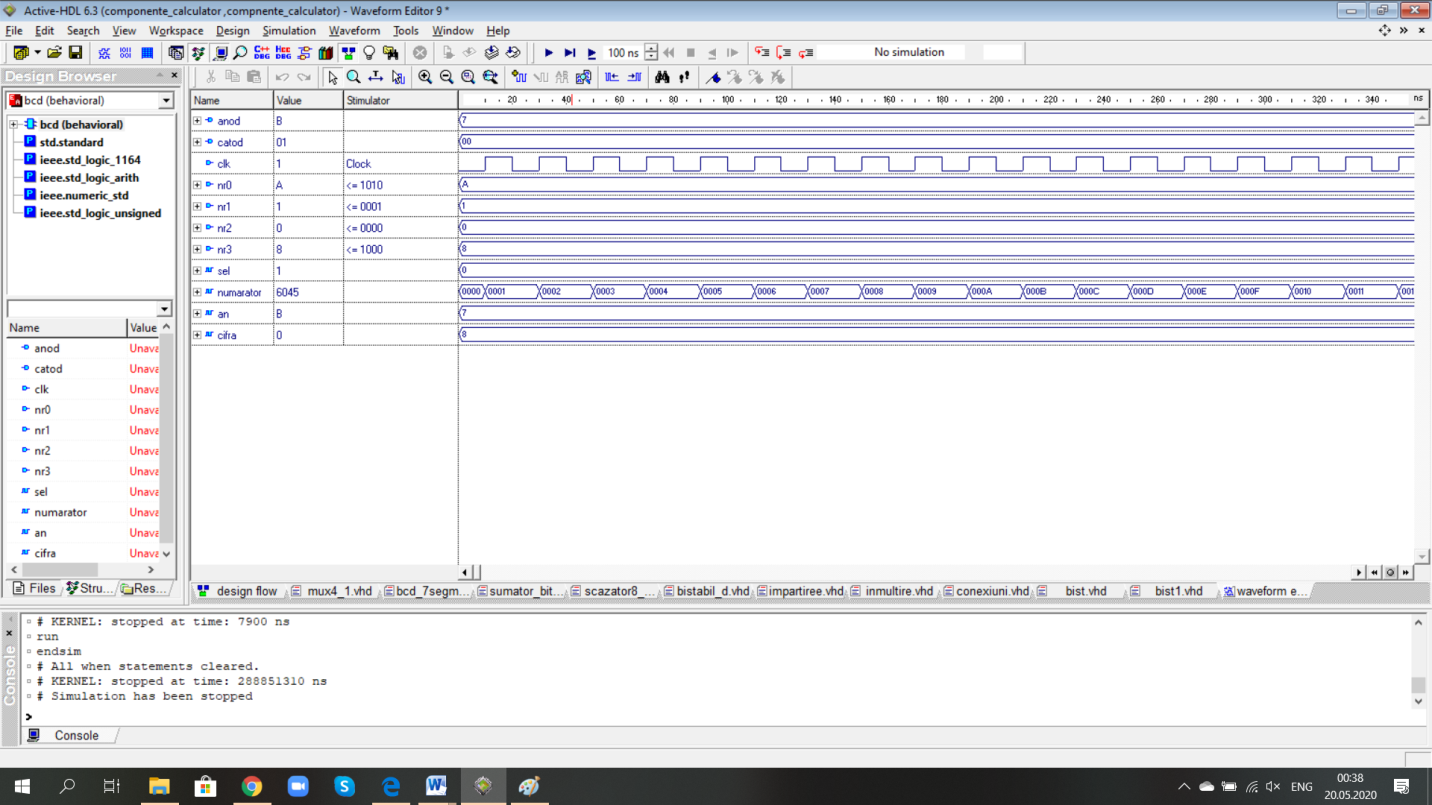


Înmulțirea funcționează atât pentru valori diferite de 0, cât și atunci când înmulțitorul este 0. Semnul rezultatului este corect atunci când înmulțim două numere pozitive sau un număr pozitiv și unul negativ.

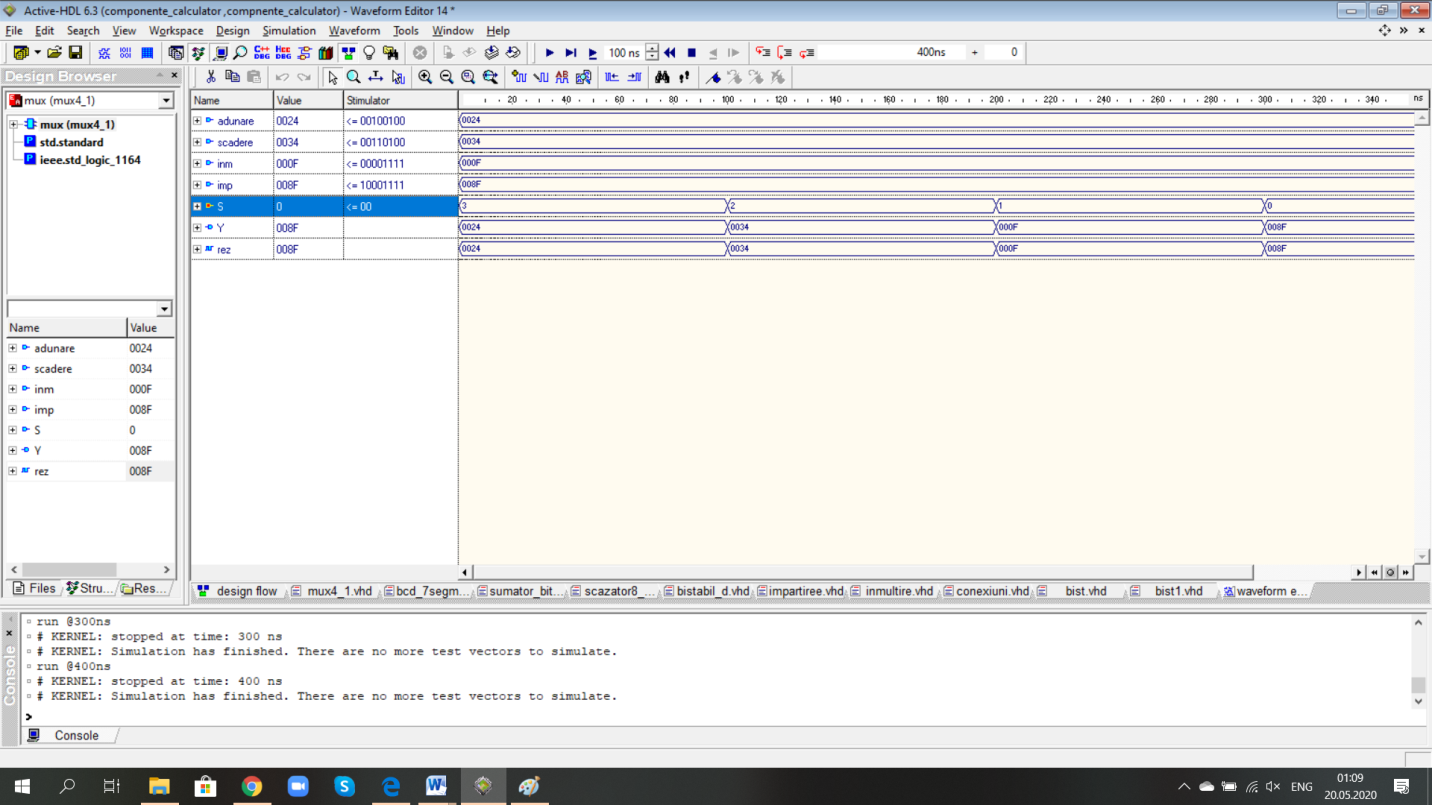


La împărțire putem observa că atunci când vrem să realizăm o împarțire cu 0 ne va da eroare și o împărțire normală este corectă de asemenea, iar semnele se asignează corect pentru rezultate.

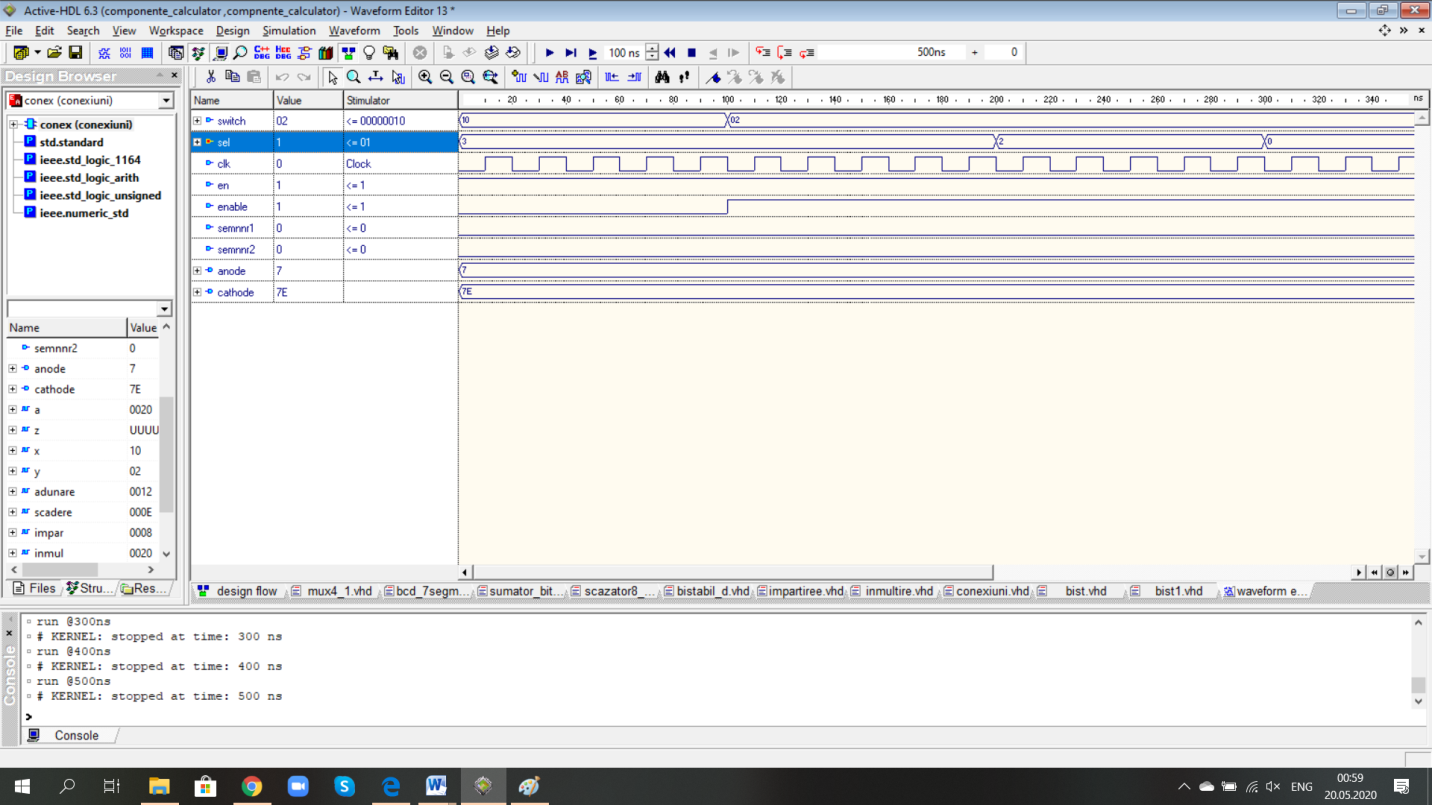


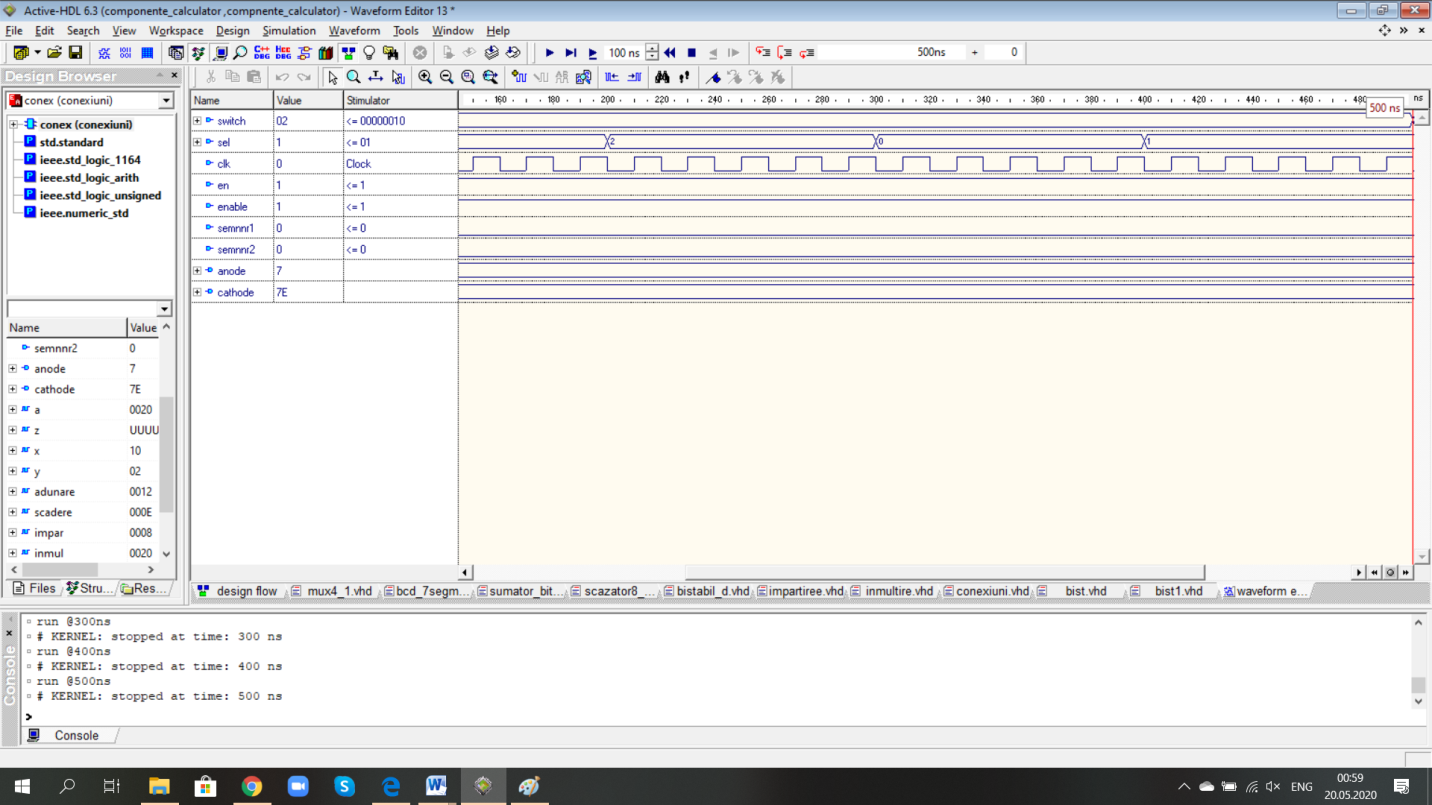


Componenta pentru afișor este de asemenea funcțională.

****

Multiplexorul lasă să treacă rezultatul unei operații în funcție de selecțiile aplicate.

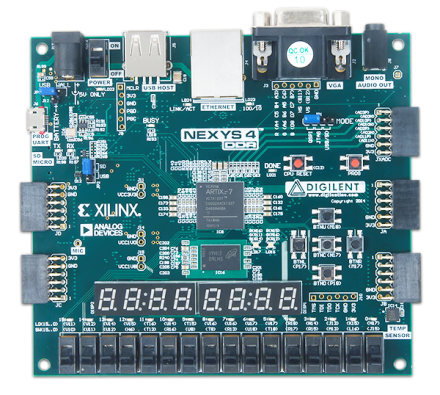




După cum se poate observa întregul sistem funcționează. Toate calulele sunt corecte.

* 1. **Fișierul de constrângeri și placa FPGA**

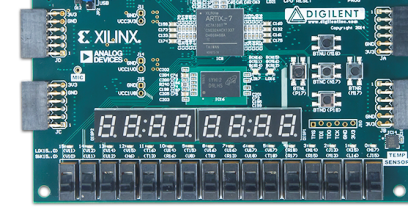
Înainte de toate voi începe cu o scurtă prezentare a plăcii NEXYS-4 pe care am ales-o pentru implementare.



Ca și componente, placa NEXYS-4 cuprinde cele 8 afișoare 7 segmente, cele 16 comutatoare cu 2 stări, cele 5 comutatoare de tip push-button și 16 leduri după cum sunt exemplificate mai jos:

8 afișoare 7 segmente

16 LED-uri



5 push-buttons

16 switch-uri

Placa cuprinde un oscillator intern cu frecvența de 100 MHz, acesta fiind clock-ul intern al plăcii. Semnalele de tact de la oscilator se conectează la pinii de tact global. Astfel se poate realiza divizarea de frecvență.

#Clock

NET "clk" LOC=E3 | IOSTANDARD=LVCMOS33;

NET "clk" CLOCK\_DEDICATED\_ROUTE=FALSE;

# Switches

NET "switch[0]" LOC=J15 | IOSTANDARD=LVCMOS33;

NET "switch[1]" LOC=L16 | IOSTANDARD=LVCMOS33;

NET "switch[2]" LOC=M13 | IOSTANDARD=LVCMOS33;

NET "switch[3]" LOC=R15 | IOSTANDARD=LVCMOS33;

NET "switch[4]" LOC=R17 | IOSTANDARD=LVCMOS33;

NET "switch[5]" LOC=T18 | IOSTANDARD=LVCMOS33;

NET "switch[6]" LOC=U18 | IOSTANDARD=LVCMOS33;

NET "switch[7]" LOC=R13 | IOSTANDARD=LVCMOS33;

NET "semnnr1" LOC=T8 | IOSTANDARD=LVCMOS33;

NET "semnnr2" LOC=U8 | IOSTANDARD=LVCMOS33;

#7 catment display

NET "cathode[0]" LOC=L18 | IOSTANDARD=LVCMOS33;

NET "cathode[1]" LOC=T11 | IOSTANDARD=LVCMOS33;

NET "cathode[2]" LOC=P15 | IOSTANDARD=LVCMOS33;

NET "cathode[3]" LOC=K13 | IOSTANDARD=LVCMOS33;

NET "cathode[4]" LOC=K16 | IOSTANDARD=LVCMOS33;

NET "cathode[5]" LOC=R10 | IOSTANDARD=LVCMOS33;

NET "cathode[6]" LOC=T10 | IOSTANDARD=LVCMOS33;

NET "anode[0]" LOC=J17 | IOSTANDARD=LVCMOS33;

NET "anode[1]" LOC=J18 | IOSTANDARD=LVCMOS33;

NET "anode[2]" LOC=T9 | IOSTANDARD=LVCMOS33;

NET "anode[3]" LOC=J14 | IOSTANDARD=LVCMOS33;

#Buttons

NET "sel[0]" LOC=P18 | IOSTANDARD=LVCMOS33;

NET "sel[1]" LOC=N17 | IOSTANDARD=LVCMOS33;

NET "enable" LOC=M18 | IOSTANDARD=LVCMOS33;

NET "en" LOC=M17 | IOSTANDARD=LVCMOS33;

**4.Lista componentelor utilizate**

1. 2 Registre de memorie pe 8 biți
2. Registru de memorie pe 16 biți
3. Sumator pe 8 biți
4. Scăzător pe 8 biți
5. Înmulțire pe 8 biți
6. Împărțire pe 8 biți
7. Multiplexor 4:1
8. Decodificator BCD-7 segmente

Semnificația semnalelor de I/O și a semnalelor interne a fost arătată la descrierea componentelor.

**5.Justificarea soluției alese**

La reazlizarea acestui proiect am optat pentru obținerea unui cod lizibil și am utilizat o metodă simplă și anume împărțirea pe componente. Fiecare componentă utilizată are câte o funcție diferită în funcționarea calculatorului de buzunar. Am utilizat de asemenea nume semnificative pentru semnale, astfel încât să se înțeleagă utilitatea lor, iar unele părți ale codului au fost comentate. Legarea componentelor în fișierul final am făcut-o printr-o descriere structurală, aceasta fiind foarte utilă în cazul în care dorim să folosim din nou acele componente.

Alegerea făcută pentru implementare mi s-a părut cea mai ușoară și logică.

**6. Utilizare și rezultate**

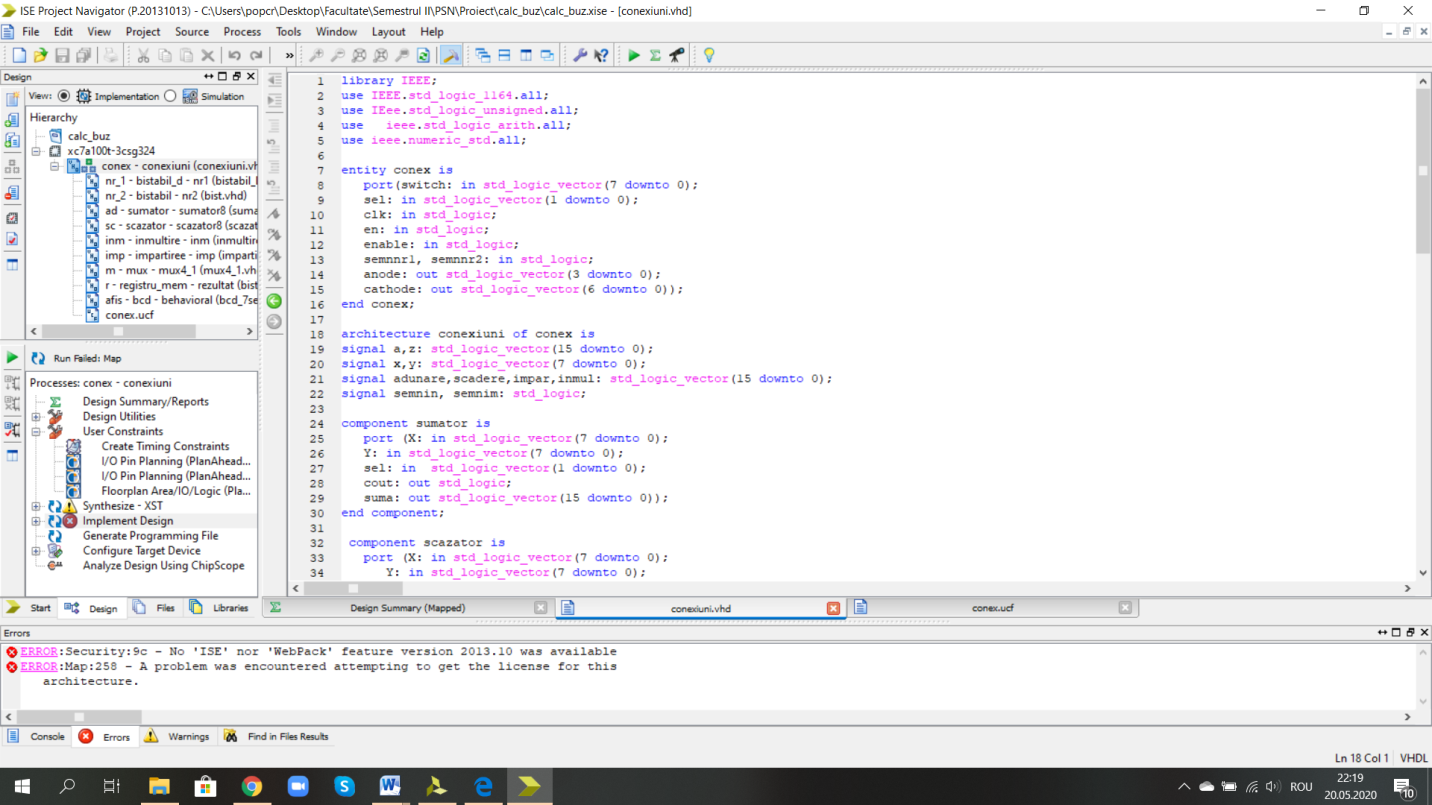
După pornirea programului Active-Hdl, se selectează fișierul ”conexiuni.vhd” de la File>Open. În primul rând se compilează codul vhdl, prin apăsarea lui ”Compile” de la Design, de lângă File. Apoi se selectează New Waveform, alegem semnalele de care avem nevoie (le putem adăuga pe toate). După aceasta, în loc de Add Signals, dăm click dreapta pe semnale și selectăm Stimulators, apoi atribuim valori semnalelor după cum urmează: switch-urile primesc valoarea primului număr, semnalul de enable va avea valoarea ‘0’, semnalul de clk va avea valoare de clock, iar dacă alegem ca selecție înmulțirea și împărțirea vom da valori și semnalelor semnnr1 și semnnr2. Se apasă butonul Run For, apoi trebuie să modificăm valoarea switch-urilor cu un alt număr, semnalul enable va avea acum valoarea ‘1’ pentru a scoate din registru al doilea număr și semnalul en valoarea ‘1’. Se va observa că după apăsarea butonului Run For încă o data se va afișa rezultatul operației selectate.

Pentru utilizarea proiectului pe o placă FPGA se deschide programul XILINX ISE, în care se face un workspace. Aici, la User Constraints-Edit Constraints se pot declara legăturile semnalelor. După asignarea pinilor, se va apăsa dublu-click pe Generate Programming files. Dacă programul nu va genera nici o eroare, putem continua prin testarea proiectului pe placa FPGA. În continuare, se selectează Digilent->Adept->Export din meniul START. Se pornește plăcuța, apoi se apasă butonul Initialize Chain. Introducem fișierul cu extenia .bit și apăsăm butonul Program Chain. În acest moment, plăcuța este pregătită pentru utilizare.

Am utilizat 10 switch-uri pentru introducerea numerelor, 4 butoane pentru selecții și enable și de asemenea 4 afișoare 7 segmente.

**Funcția fiecărui buton și switch este prezentată mai jos:**

* J15- switch pentru primul bit
* L16- switch pentru al doilea bit
* M13- switch pentru al treilea bit
* R15- switch pentru al patrulea bit
* R17- switch pentru al cincilea bit
* T18- switch pentru al șaselea bit
* U18- switch pentru al șaptelea bit
* R13- switch pentru al optulea bit
* T8- switch pentru semnul primului numar
* U8- switch pentru semnul celui de-al doilea numar
* P18-buton pentru sel[0]
* N17-buton pentru sel[1]
* M18-buton pentru enable-ul care va scoate primul sau al doilea număr
* M17-buton pentru enable-ul care scoate rezultatul
* Anozii, catozii sunt utilizați pentru afișarea rezultatelor pe cele 4 ecrane.

****

Așa cum se observă în fereastra de Design a proiectului nostru, generarea fișierului cu extensie .bit nu a avut loc, întrucât nu am reușit să scap de erorile și warning-urile apărute.

Nu am dispus de o placă FPGA pentru a putea testa proiectul cu fiecare etapă de proiectare, însă acesta funcționează în mediul Active-HDL.

**7.Posibilități de dezvoltare ulterioară**

Mereu va fi loc pentru orice îmbunătățire, așa că eu m-am gândit că proiectul meu ar putea fi dezvoltat ca numerele să fie introduse în zecimal, nu în binar, întrucât acest sistem de numerație nu este cunoscut oricui, iar rezultatele să nu fie afișate în hexazecimal, ci în zecimal.

O altă îmbunătățire ar putea fi calculul altor operații, pentru că atunci când cumperi un calculator de buzunar te așepți să aibă cât mai multe funcționlități, nu doar operațiile de bază.

De asemenea, s-ar putea extinde domeniul de calcul, de la 8 biți la oricât este nevoie.