Proiectarea unei unități aritmetico-logice (UAL)

Cuprins

1. Introducere

1.1 Context

Scopul acestui proiect este de a proiecta o unitate aritmetico-logică unde se implementează și testează operațiile de adunare și scădere, înmulțire și împărțire , incrementare/decrementare, operațiile folosite în logica binară, negarea și cele două operații de rotație. O diferență ar fi la operațiile de adunare/scădere care vor fi reprezentate în formatul C2.

Această unitate poate fi folosită cu ușurință de către oricine, iar operațiile se pot testa la fel de simplu ca și un calculator cu operații. Poate fi integrat într-un calculator mai mic sau poate fi adaptat la o unitate și mai mare având operații mult mai multe și mai folositoare și așa se poate dezvolta.

1.2 Specificații

UAL va fi simulat în Vivado, iar ulterior se va adăuga și placa Basys 3. UAL își încarcă datele în regiștrii de intrare, unitate externă îi spune la UAL ce operație să execute, iar la final se salvează rezultatele de la operații în regiștrii de ieșire. Mai există câteva registre pentru diferite cazuri , ce vor fi folositoare la câteva operații și anume: carry in/carry out.

1.3 Obiective

Obiectivul principal la această unitate ar fi să se implementeze o unitate de control care să se ceară ce operație se dorește să fie implementată, să se execute atâtea operații câte dorește persoana respectivă, iar la final, printr-o comandă, să încheie tot procesul, ca să se creeze alt proces după și tot așa. Să se implementeze toate operațiile cât să fie corecte și să dea rezultatele care trebuie.

2.Studiul bibliografic

În această secțiune se va vorbi despre operații și cum se vor implementa.

Complement față de doi : este metoda de reprezentare binară a numerelor întregi în calculator. Această metodă simplifică cele două operații de adunare și scădere [3]. Ca și orice altă metodă de reprezentare a numerelor pe biți, bitul cel mai semnificativ este salvat pentru a arăta semnul numărului (0 pentru + și 1 pentru -) . La scădere, în această metodă, se va face de fapt adunarea descăzutului cu complementul față de doi al scăzătorului.

a) **Adunarea și scăderea**
   Atât la adunare, cât și la scădere, a două numere, se adună bit cu bit/se scade bit cu bit, iar noul rezultat se păstrează într-un output. La fiecare operație poate apărea carry(adunare) sau borrow(scădere), unde: la adunare se vor aduna cei 2 operanzi împreuna cu carry in, iar suma va fi într-un output și carry out, iar la scădere, se scad cei 2 împreună cu borrow, și scăderea va fi într-un output și carry out.[1][2]
b) **Operațiile ȘI, SAU, NU**

   Avem 2 input-uri în cazul operațiilor ȘI, SAU, unde pentru cele 2 input-uri A și B care pot lua valorile 0 și 1, se face operația propriu-zisă pe A și B, iar rezultatul va fi stocat într-un output Y. La aceste două operații se verifică bit cu bit și în funcție de operație, cu tabelul de adevăr, se returnează rezultatul.[1]

   În cazul operației NOT, se primește un input A și toți biții se neagă, urmând ca noua valoare să fie stocată în Y.[1]

c) **Rotațiile stânga și dreapta**
   Operandul este tratat ca un buffer circular de biți, astfel încât biți cei mai semnificativi și mai puțin semnificativi sunt adiacenți.[1]

d) **Incrementare și decrementare**
   Valoarea stocată într-un input A, crește sau scade cu 1, iar rezultatul obținut în urma acestor operații se va salva într-un output Y.[1]

e) **Înmulțire și împărțire**
   În orice operație de împărțire, avem 4 operanzi și anume: deîmpărțit, împărțitor, cât și rest. Algoritmul este următorul.[2]
   1.Se alege o cifră, se scade produsul dintre aceasta și împărțitor din rest parțial.
   2.Dacă rezultatul este mai mic ca împărțitorul, s-a alea cifra corect

3.Dacă nu, se alege o altă cifră și se repetă.

La înmulțire, există mai multe metode pentru a se înmulți 2 numere pe 32 de biți și anume: Shift and Add Multiplication, Booth's Techinque, Higher-Radix Multiplication, Array Multiple și Wallace Tree.

3.Analiză și 4.Design

a) Adunare

Adunarea numerelor în complement față de doi nu este foarte diferită față de celălalte metode, deoarece se adună bit cu bit, doar că aici există cazul de overflow(de obicei când se adună 2 numere pozitive și iese rezultat negativ sau invers), și atunci trebuie verificare și atenție mare asupra acestei operații. În figura 1, sunt câteva exemple de adunare cu sau fără overflow.

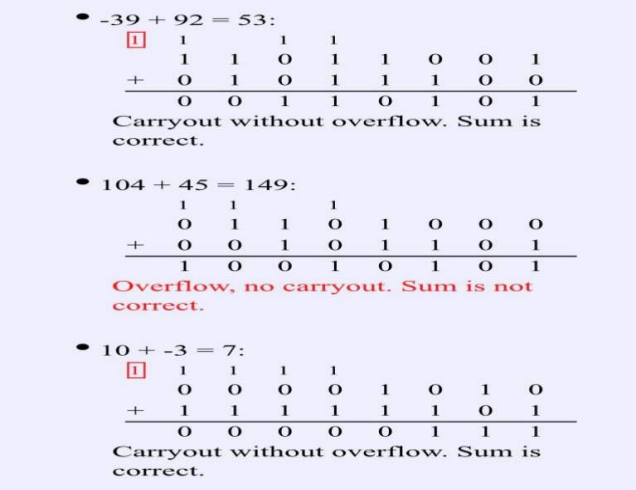


Figura 1. Adunarea numerelor cu sau fără overflow [5]

Pentru implementarea acestei operații, se vor folosi 32 de full adders, ca în figura 2 [4]

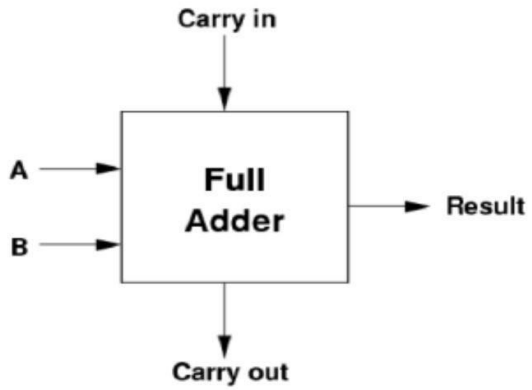


Figura 2. Schema unui full adder [4]

Acestea vor fi legate după cum urmează în figura 3

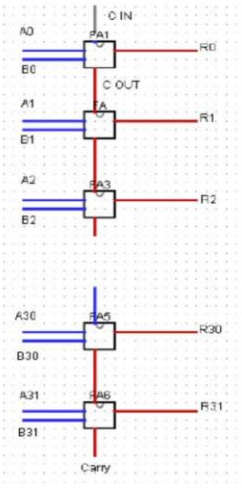


Figura 3. Schema pentru cele 32 full addere [4]

Această componentă va fi implementată cu Ripple Carry Adder, după cum se poate vedea în imagine, fiecare carry out e legat la următorul și tot așa. [4]

b) Scădere

Se întâmplă același lucru ca la adunare, doar că trebuie să dăm atenție la borrow. O altă modalitate de implementare a acestei operații este cu operația de adunare, ne putem folosi de aceasta, și să transformăm scăderea într-o adunare după cum se vede în figura 4. [4]

$$A - B = A + (-B) = A + \sim B + 1$$

Figura 4. Transformare scădere în adunare [4]

Pentru a nega unul dintre numerele date la input, se scriu în ordine inversă biții și se adaugă valoarea 1 la adunarea dintre cele două numere.

c) Incrementare

Este tot un fel de operație de adunare, doar că este un operand și valoarea 1. Cum avem operația de adunare, ne vom folosi de aceasta, iar operația de incrementare se va folosi de componenta de Ripple Carry Adder implementată la operația propriu-zisă de adunare

d) Decrementare

Această operație este asemănătoare cu scăderea, și cum la scădere am propus 2 variante, la fel se poate și aici, să facem o adunare cu valoarea -1 și să folosim doar componenta Ripple Carry Adder.

e) Operațiile AND, OR, NOT

Pentru aceste operații, procedeul este similar, se face operația bit cu bit

La AND, se vor folosi 32 de porți AND [4], care vor fi legate cum indică figura 5.

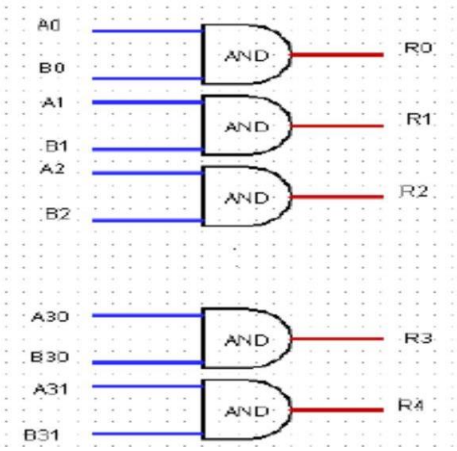
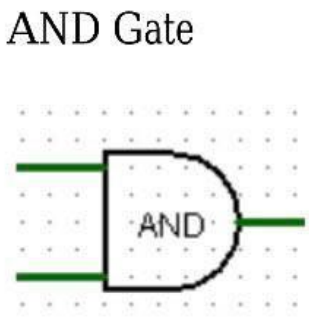


Figura 5. Schema pentru cele 32 de porți AND [4]

Și are tabelul de adevăr (figura 6) :



a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

Figura 6. Tabelul de adevăr pentru poarta AND [4]

La OR, la fel: 32 de porți OR (figura 7) [4]

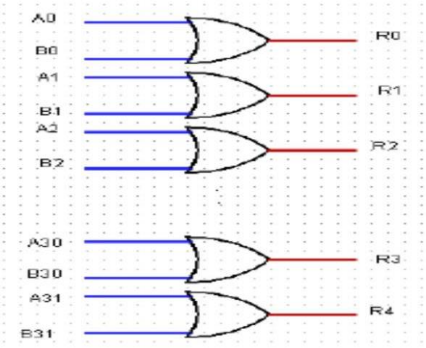
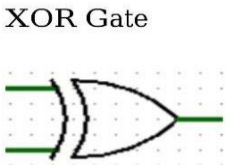


Figura 7.Schema pentru cele 32 de porți OR [4]

Cu tabelul de adevăr (figura 8) :



a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

Figura 8. Tabelul de adevăr pentru poarta OR [4]

La NOT, la fel ca și la cele două de mai sus, fiecărui bit în parte, i se aplică operația de NOT.

f) Împărțire

În orice operație de împărțire, avem 4 operanzi și anume: deîmpărțit, împărțitor, cât și rest. Algoritmul este următorul [2] (figura 9) :

1.Se alege o cifră, se scade produsul dintre aceasta și împărțitor din rest parțial.

2.Dacă rezultatul este mai mic ca împărțitorul, s-a alea cifra corect

3.Dacă nu, se alege o altă cifră și se repetă.

Algoritmul se bazează pe scăderi repetate, dar s-ar putea optimiza, prima oară să se facă shiftare și după scădere.

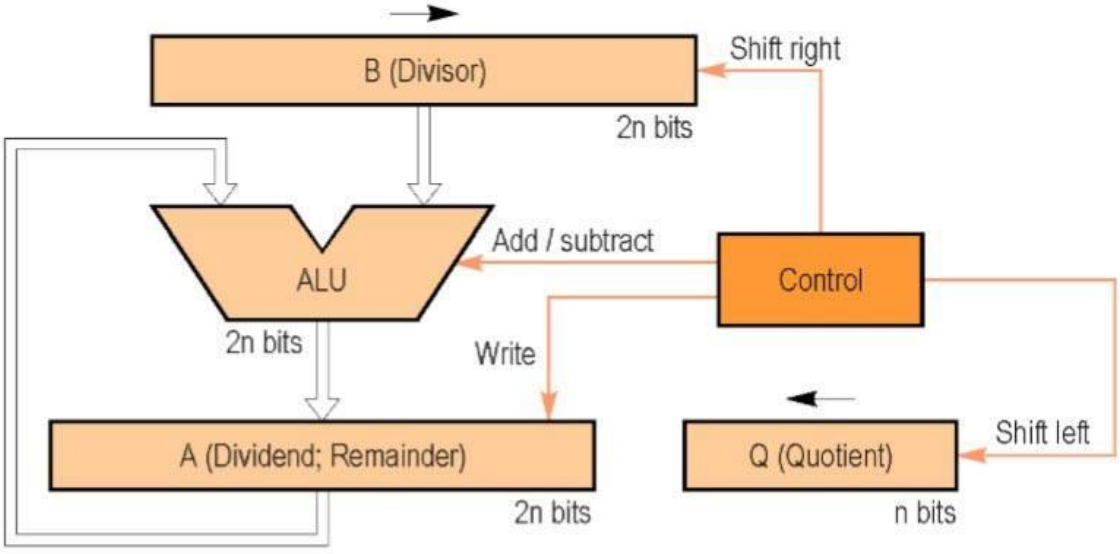


Figura 9. Schema pentru operația de împărțire explicată mai sus [2]

g)Înmulțire

Această operație se face ca și la numerele normale. Un algoritm ușor de implementat pentru ca numerele pe 32 de biți să poată fi înmulțite ar fi : Shift and Add Multiplication [2] (figura 10) : Se adună numărul A(primul număr) de B (al doilea număr) ori. Fiecare bit se așază la locul său, către stânga cu o poziție la fiecare linie, unul sub celălalt. La final, toate rezultatele sunt adunate fiecare bit cu bit și așa reiese rezultatul înmulțirii a două numere.

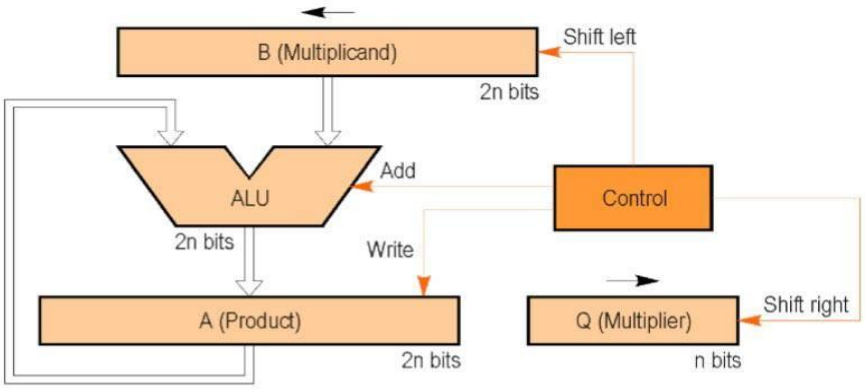


Figura 10. Schema pentru operația de înmulțire explicată mai sus [2]

h)Rotații

•) la stânga : se mută toți biții cu o căsuță la stânga, iar primul bit de dinainte să se facă rotația trece la final, devenit ultimul bit (figura 11)

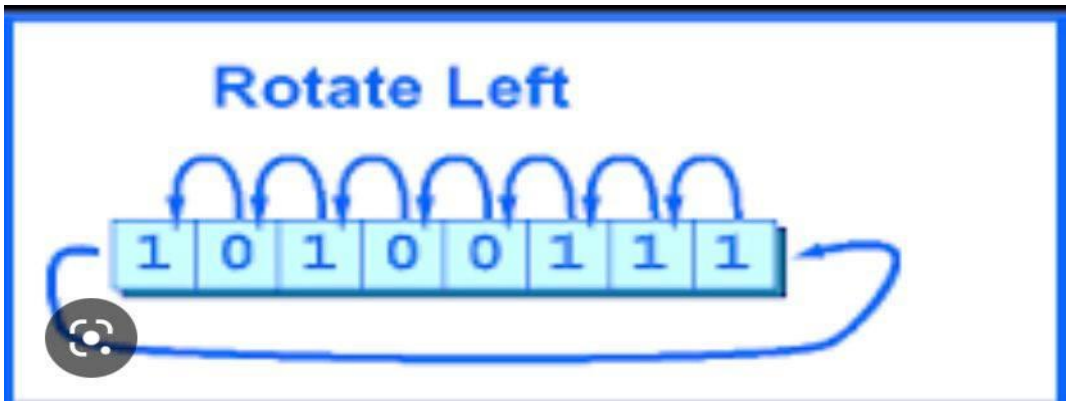


Figura 11. Operația de rotire spre stânga [7]

•) la dreapta : același procedeu ca la rotația la stânga, doar că de data aceasta ultimul bit, va trece în locul primului bit, iar restul se mută cu o căsuță mai la dreapta (figura 12)

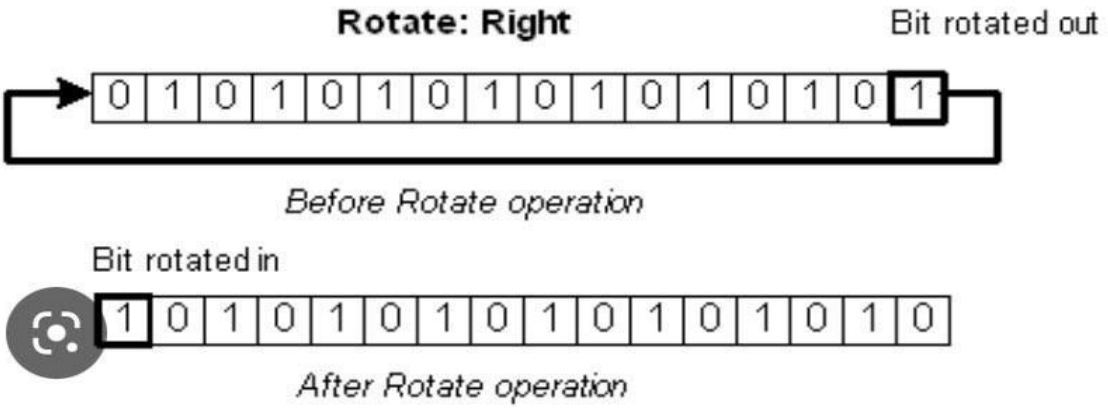
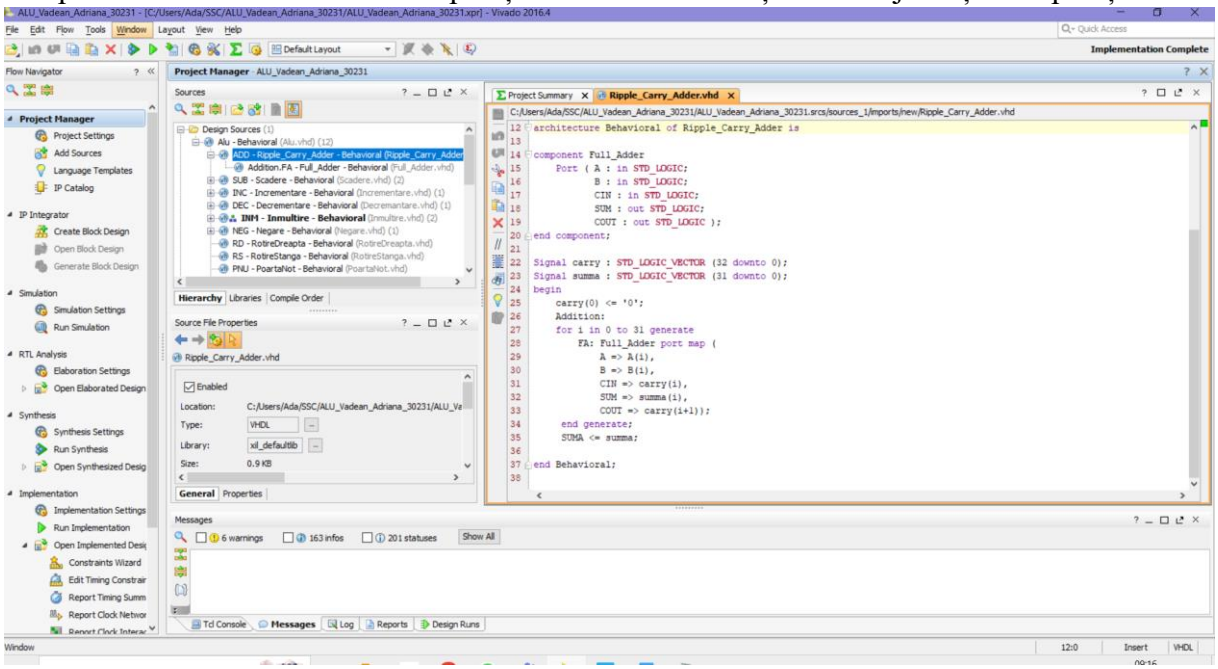


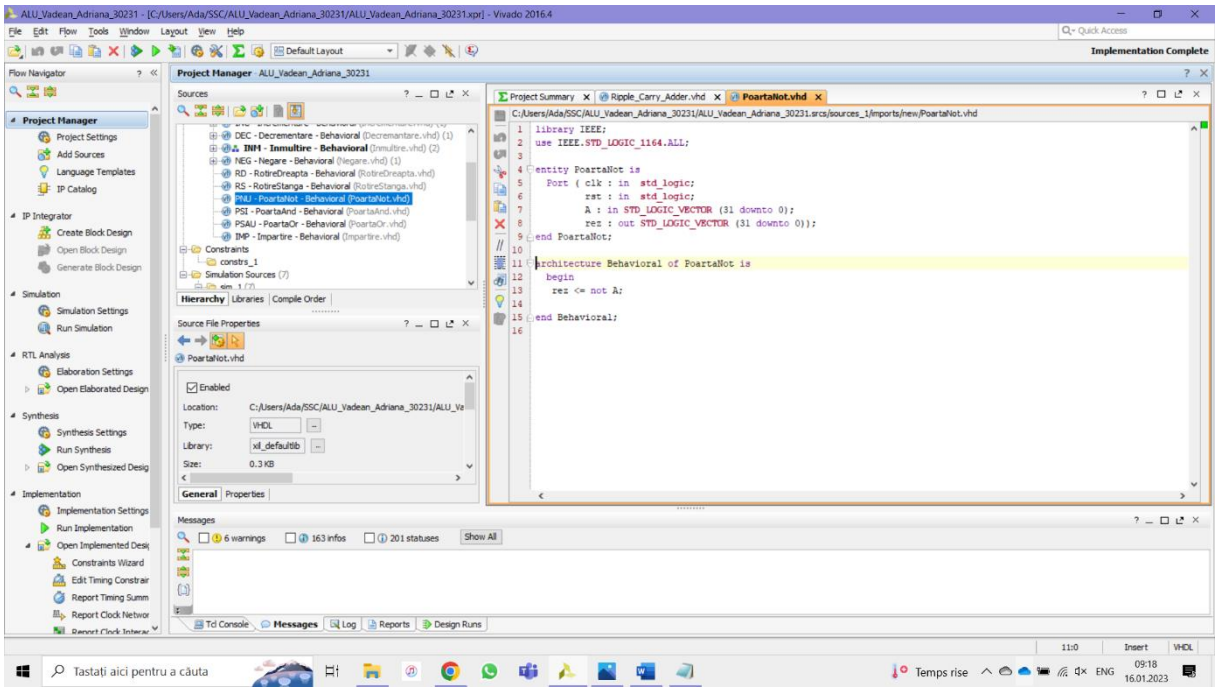
Figura 12. Operația de rotire spre dreapta [6]

5.Implementare

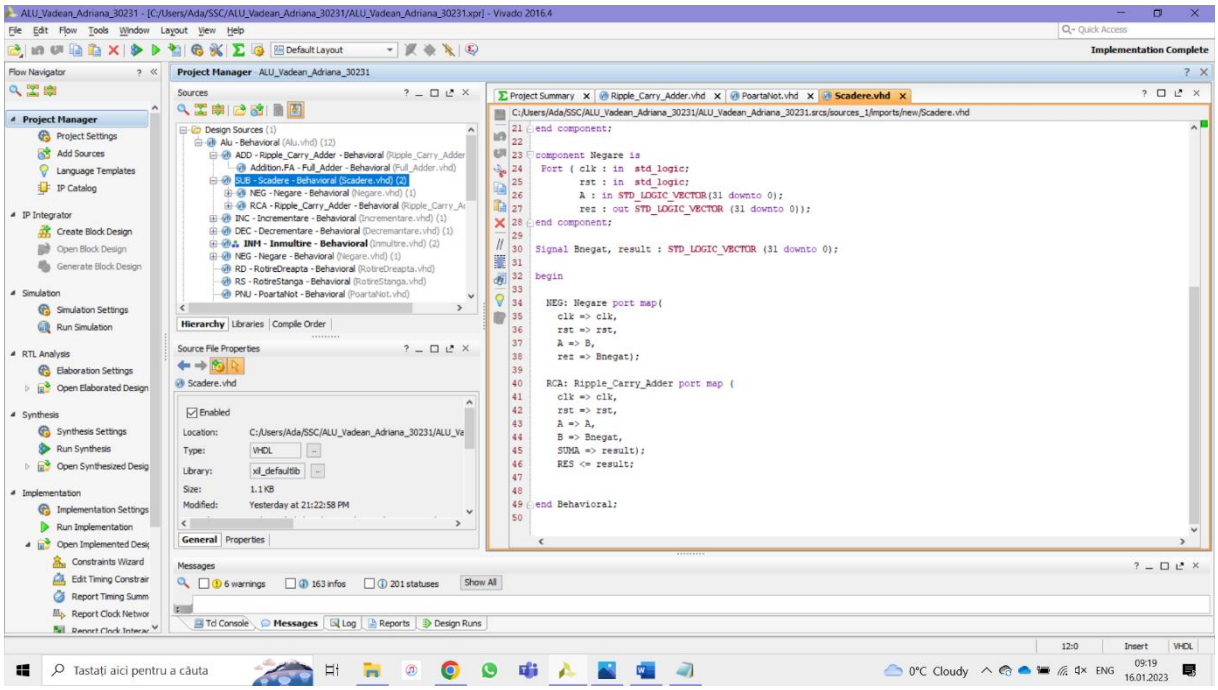
Implementarea operației de adunare cu ajutorul lui Ripple Carry Adder, unde a fost folosita componenta Full Adder. Această operație a contribuit la obținerea majorității de operații.



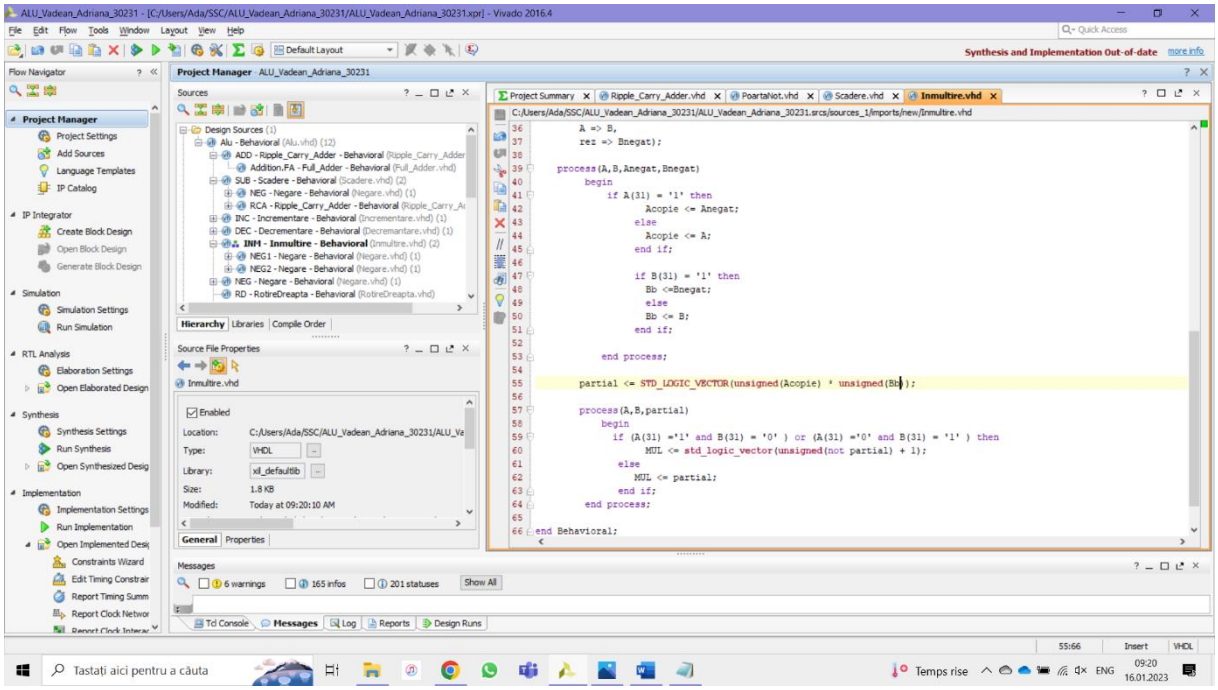
Implementarea porții Not. La fel au fost implementate și And și Or.



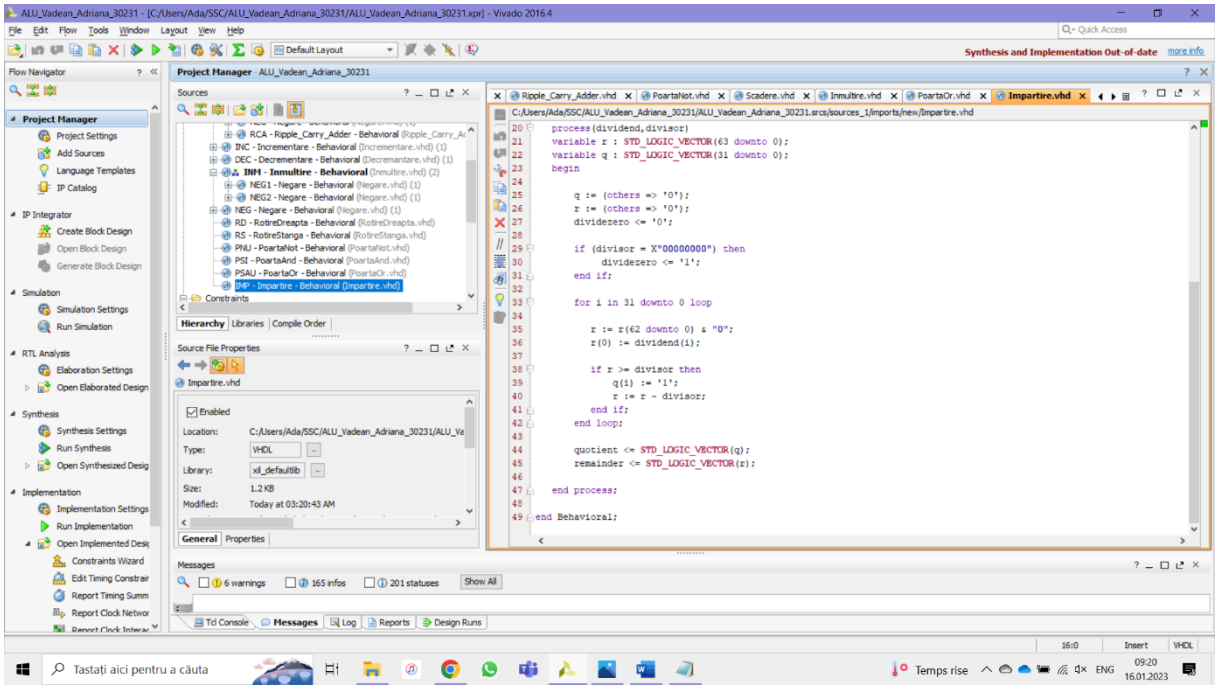
Implementarea operației de scădere, unde ne-am folosit de operația de negare și de cea de adunare, negând al doilea număr și astfel efectuând operație de adunare



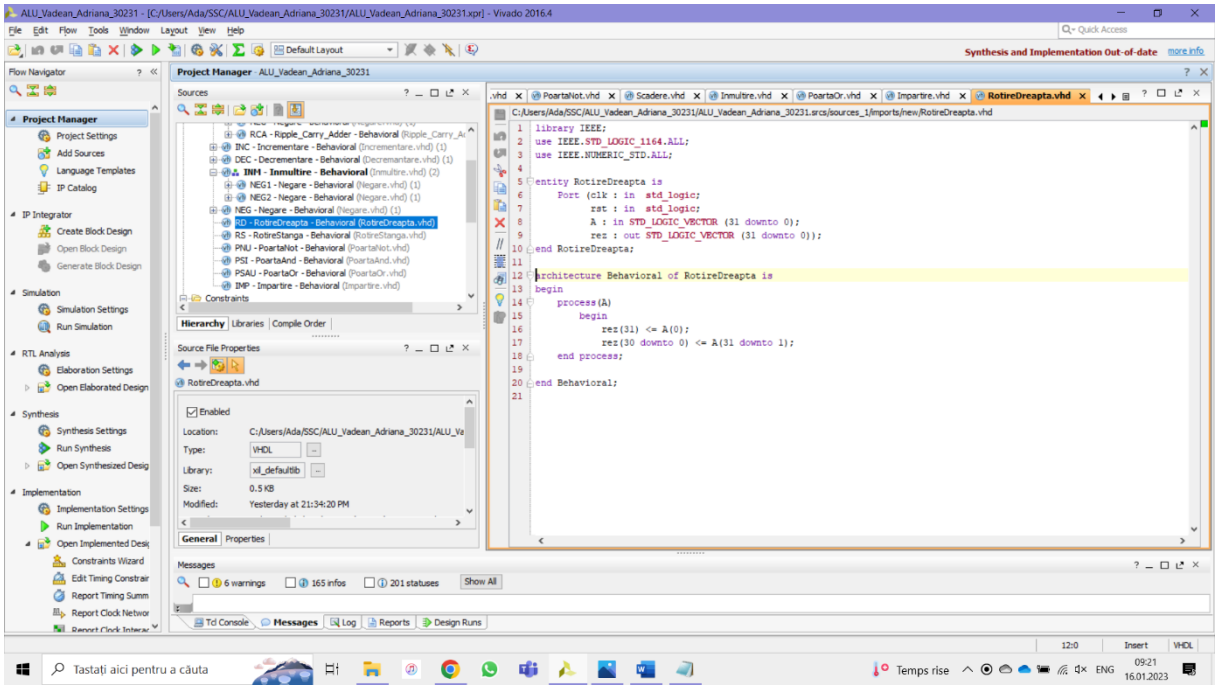
Implementarea operației de înmulțire



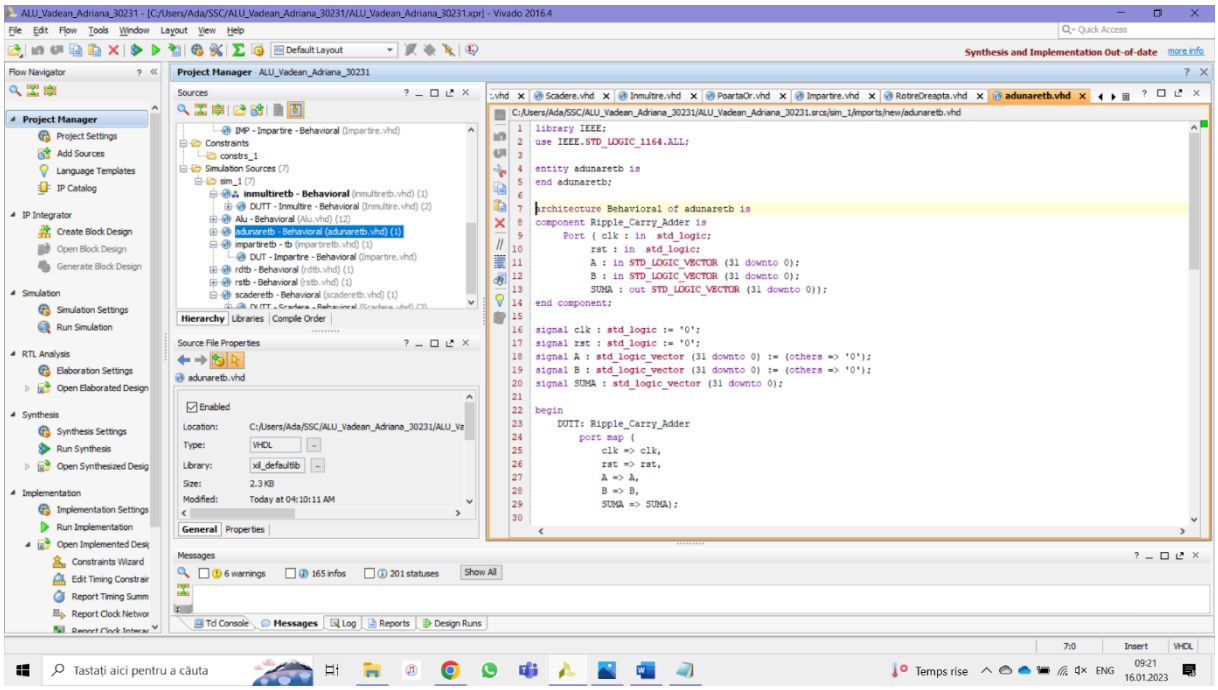
Implementarea operației de împărțire

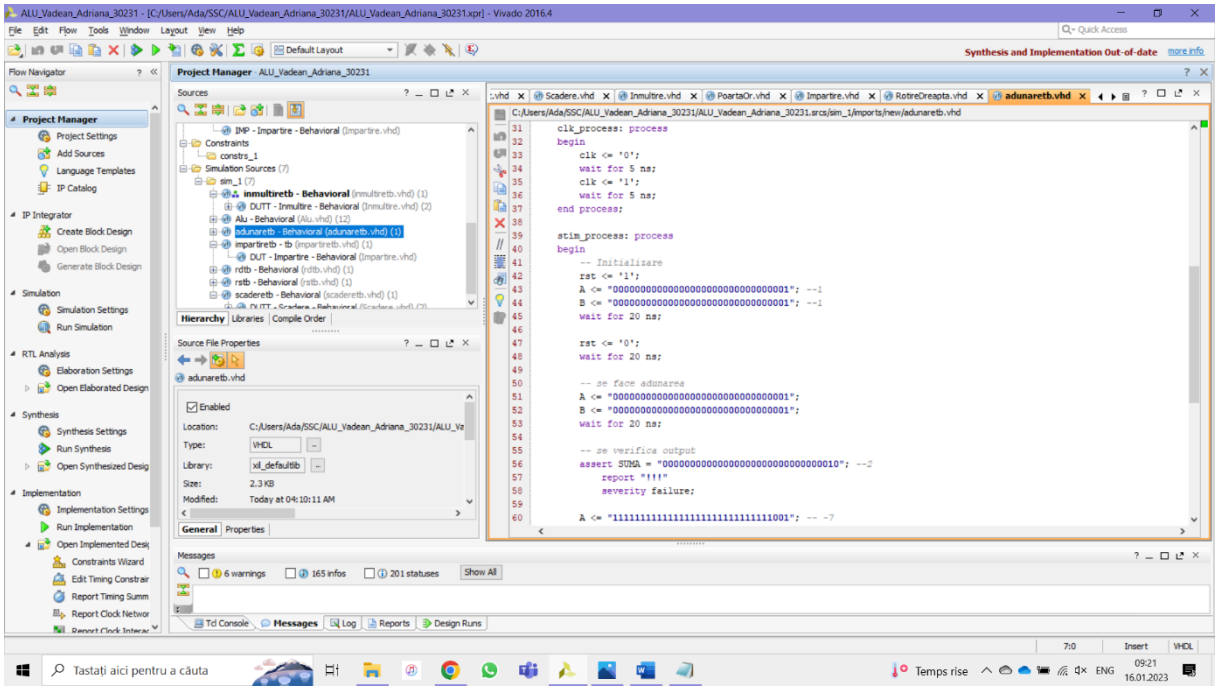


Implementarea operației de rotire dreapta, la fel fiind efectuată și pentru stânga



Implementarea testbench-ului de adunare, asemenea fiind efectuate și cele pentru înmulțire, împărțire, scădere și rotații.

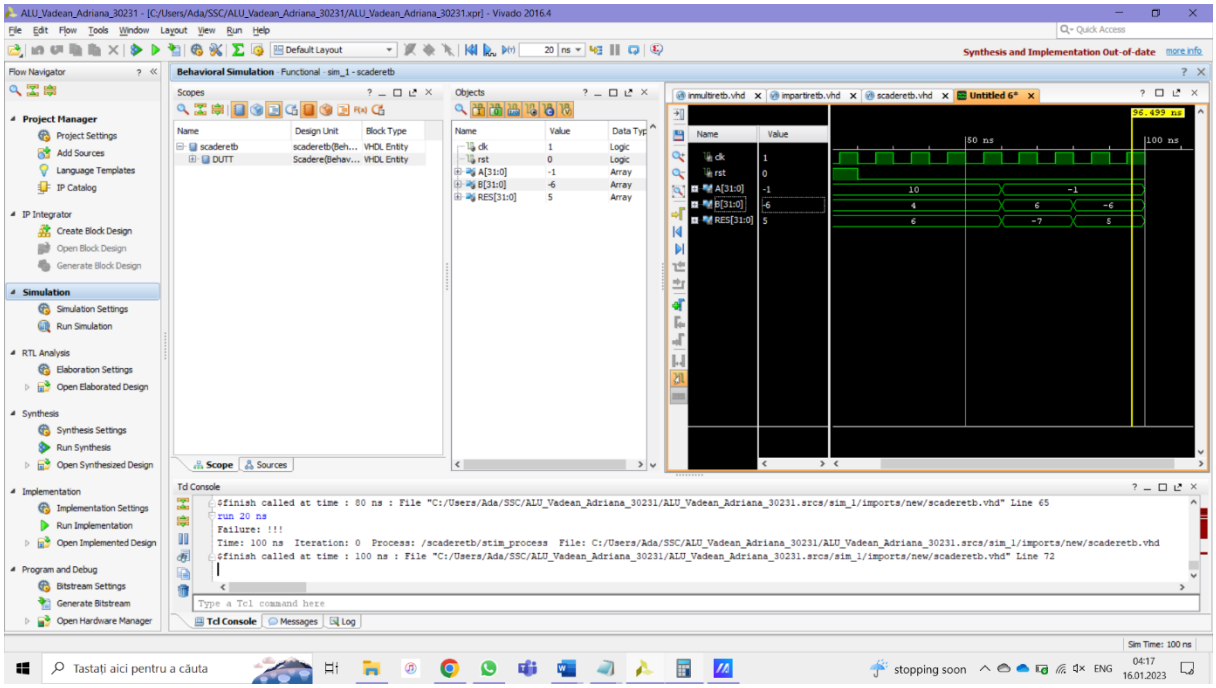




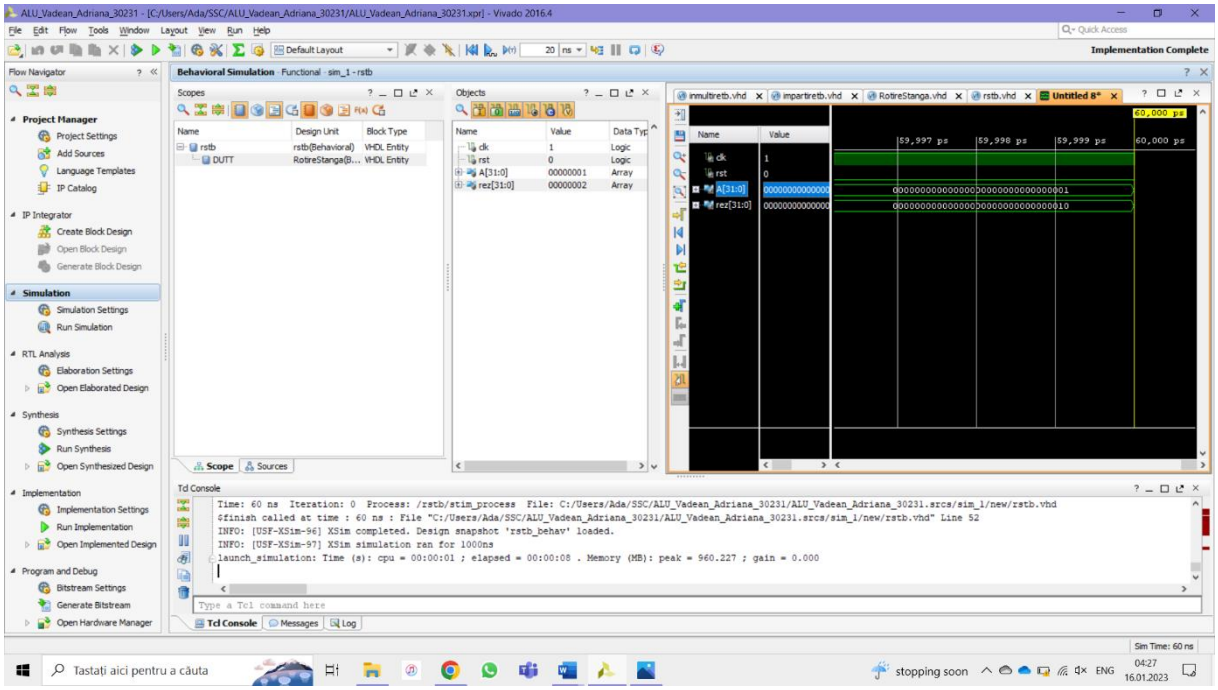
6.Testare și validare

Au fost efectuate testbench-uri pe operațiile de adunare, scădere, înmulțire, împărțire și cele de rotire.

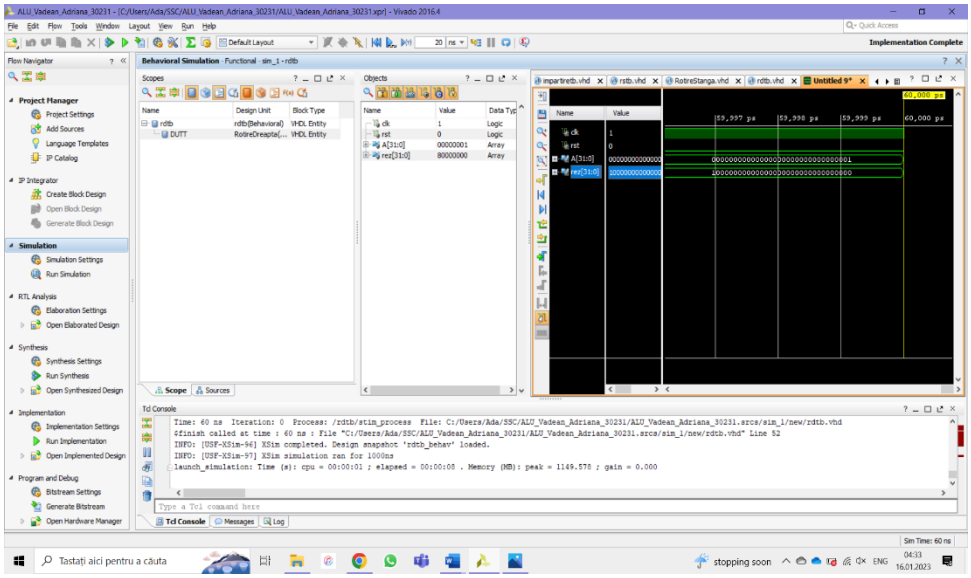
Validarea operației de scădere



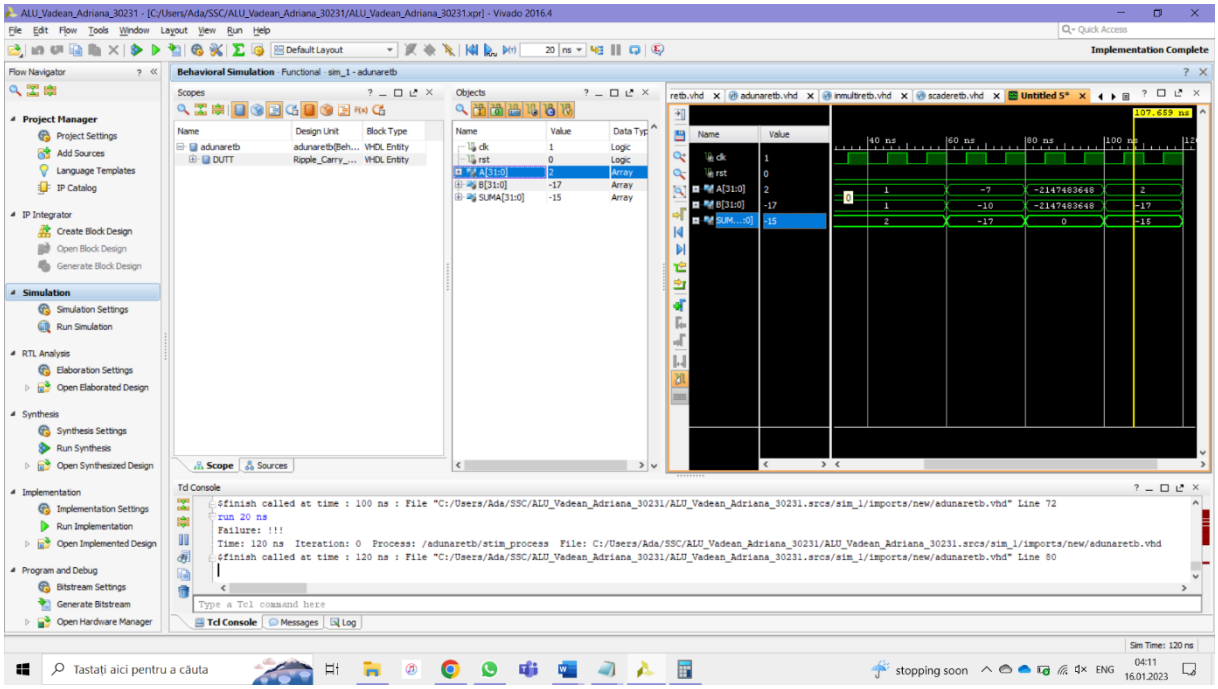
Validarea operației de rotire stânga



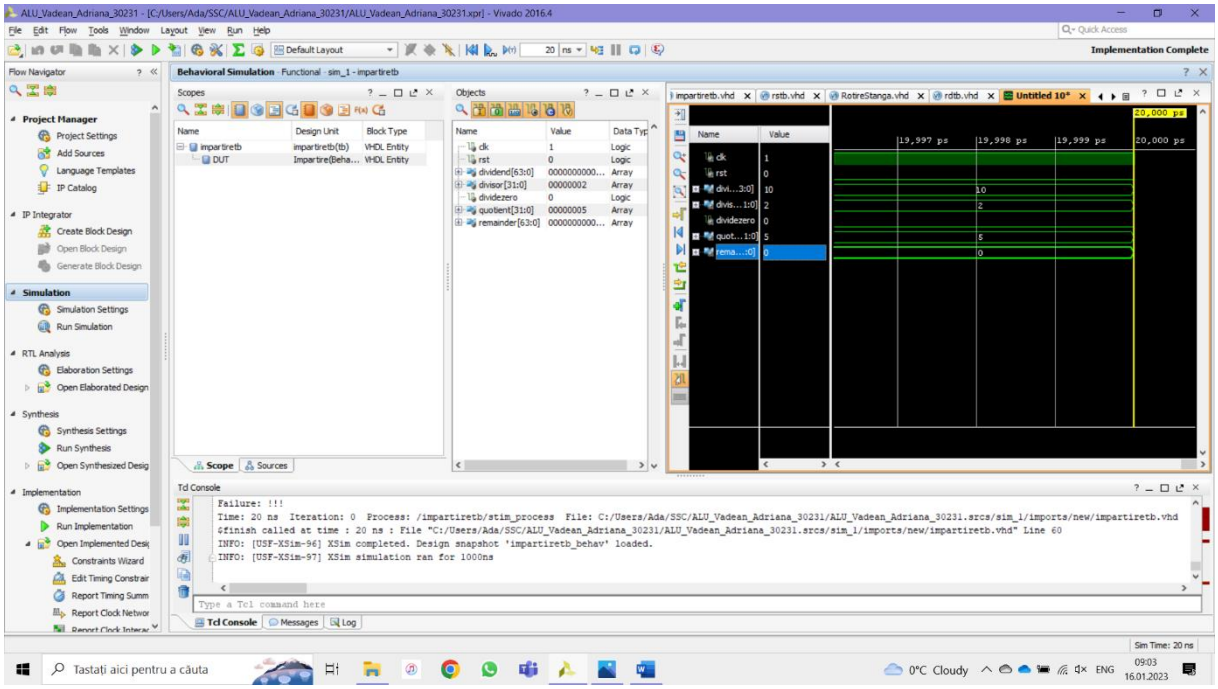
Validarea operației de rotire dreapta



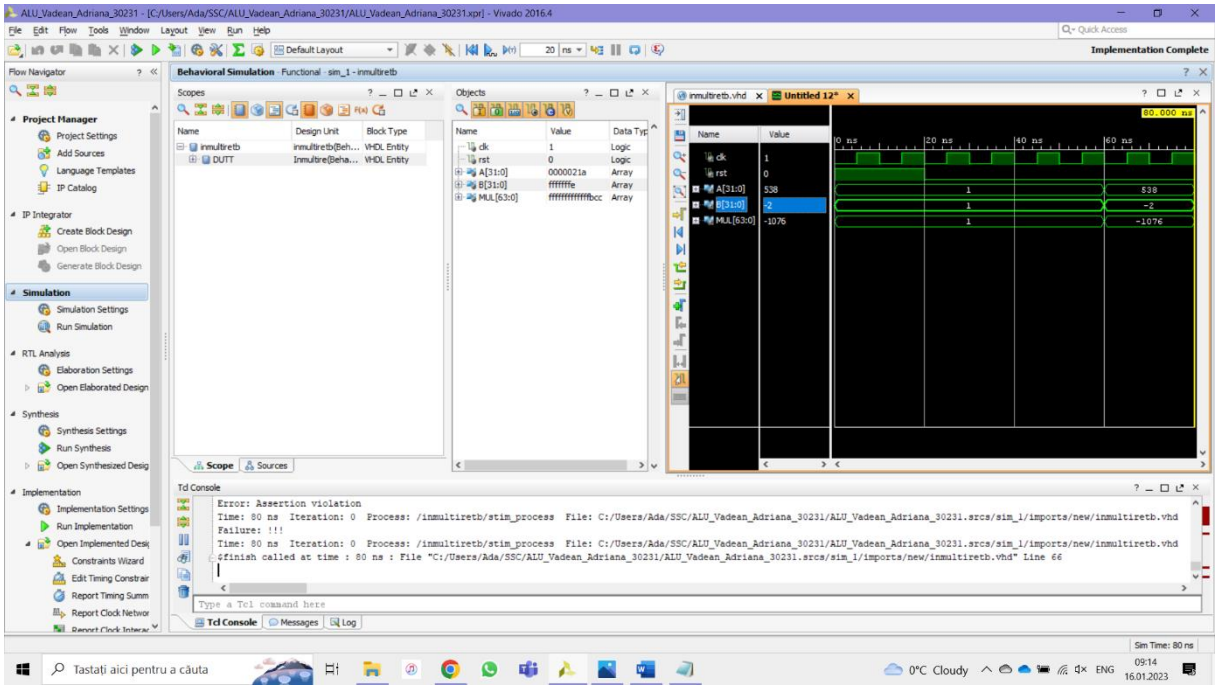
Validarea operației de adunare



Validarea operației de împărțire



Validarea operației de înmulțire



7.Concluzii

Această aplicație poate să se dezvolte, ca ,de exemplu, să aibă o interfață, să se implementeze operații mai complexe cum ar fi derivare sau integrare și implementarea acesteia pe o placuță compatibilă pentru a se afișa prin intermediul ei rezultatele într-un mod mai interesant.

8.Bibliografie

[1]"Arithmetic logic unit" https://en.m.wikipedia.org/wiki/Arithmetic_logic_unit

[2] Lab 4 Design of ALU components

[3] "Complement față de doi" https://ro.wikipedia.org/wiki/Complement_fa%C8%9B%C4%83_de_doi

[4] "Design and Development of a 32-bit ALU"

https://www.researchgate.net/publication/341055790_Design_and_Development_of_a_32-bit_ALU

[5] http://sandbox.mc.edu/~bennet/cs110/tc/add.html

[6]"Rotate"https://www.unitronicsplc.com/Download/SoftwareHelp/VisiLogic_Knowledgebase/Ladder/Functions/Logic_Functions/Rotate.htm

[7] http://programmedlessons.org/AssemblyTutorial/Chapter-23/ass23_14.html