PROIECT

Titlu: ***Împărțirea în virgulă flotantă***

Nume studentă: ***Ciochină Cătălina-Andreea***

Disciplina: ***Structura Sistemelor de Calcul***

Universitate: ***Universitatea Tehnică din Cluj-Napoca, Facultatea de Automatică și Calculatoare***

Număr grupa: ***30238***

Nume îndrumător de proiect: ***Lișman Dragoș Florin***

Data: ***08.01.2024***

Cuprins

1. Introducere

Proiectul abordează o problemă de importanță semnificativă în domeniul structurilor sistemelor de calcul: implementarea divizării în virgulă flotantă în limbajul VHDL. împărțirea în virgulă flotantă este o operație complexă, iar implementarea ei corectă și eficientă reprezintă un punct central în dezvoltarea structurilor sistemelor de calcul moderne. Alegerea limbajului VHDL pentru această implementare subliniază importanța de a avea o descriere hardware precisă și eficientă, adecvată pentru integrarea în dispozitive electronice și circuite personalizate.

Contextul temei proiectului se desfășoară într-un peisaj tehnologic în continuă evoluție, caracterizat de o cerere crescândă pentru prelucrarea numerică intensivă într-o varietate de aplicații, precum simulările științifice, inteligența artificială și calculul de înaltă performanță. În acest context, operațiile în virgulă flotantă reprezintă un aspect esențial, deoarece oferă o modalitate flexibilă și eficientă de a reprezenta și manipula numere reale.

Problema de rezolvat constă în proiectarea și implementarea unei soluții în VHDL care să ofere o împărțire precisă și eficientă în virgulă flotantă, cu atenție la gestionarea cazurilor speciale și la optimizarea resurselor. Soluția propusă se distinge prin abordarea riguroasă a cerințelor IEEE 754 și prin implementarea unui algoritm eficient.

Aritmetica în virgulă flotantă este o metodă de reprezentare a numerelor reale într-un sistem binar și implică manevrarea virgulei pentru a acoperi un interval larg de valori și precizii. În contextul prelucrării numerice și aritmeticii în virgulă flotantă, terminologia de bază cuprinde concepte esențiale legate de reprezentarea numerelor reale. Un număr în virgulă flotantă constă în mantisă (partea semnificativă), exponent (puterea la care este ridicată baza) și semn (indicând pozitiv sau negativ). Standardul IEEE 754 stabilește formatele și operațiile pentru aritmetica în virgulă flotantă, asigurând interoperabilitatea. Mantisa reprezintă coeficienții importanți, exponentul influențează scala numerelor, iar semnul indică polaritatea. Operația de diviziune în virgulă flotantă, fundamentală în acest context, implică manipularea mantisei și a exponentului conform standardului. Acești termeni sunt esențiali în înțelegerea reprezentării și manipulării numerelor reale în sistemele de calcul, precum și în implementarea operațiilor matematice complexe.

Împărțirea în virgulă flotantă presupune divizarea a două numere reprezentate în formatul virgulă flotantă, având în vedere mantisele, exponenții și semnele corespunzătoare.

Aceasta este o problemă complexă din cauza diversității operațiilor implicate și a gestionării corecte a cazurilor speciale, precum împărțirea la zero sau rezultatele infinite.

Soluția propusă în acest proiect adoptă o abordare bazată pe algoritmul Booth pentru implementarea împărțirii în virgulă flotantă în limbajul VHDL. Algoritmul Booth este cunoscut pentru eficiența sa în operațiile de multiplicare și împărțire, și în acest context, este adaptat pentru a asigura o divizie precisă și eficientă în virgulă flotantă.

Soluția adoptă un design modular cu componente distincte. Această metodă oferă eficiență resurselor și flexibilitate în integrare. Spre deosebire de alte soluții, implementarea pe componente aduce beneficii semnificative în adaptabilitate și eficiența resurselor, asigurând totodată precizie în operațiile de divizare în virgulă flotantă.

Pe parcursul documentației, secțiunile ulterioare vor detalia fundamentele teoretice, vor explora metodologia de implementare și vor oferi analize comparative, în timp ce concluziile vor sintetiza contribuțiile și impactul proiectului în contextul structurilor sistemelor de calcul.

2. Fundamentare teoretică

În cadrul fundamentării teoretice a proiectului, am consultat surse variate care au contribuit la construirea unei baze solide pentru implementarea diviziei în virgulă flotantă în limbajul VHDL.

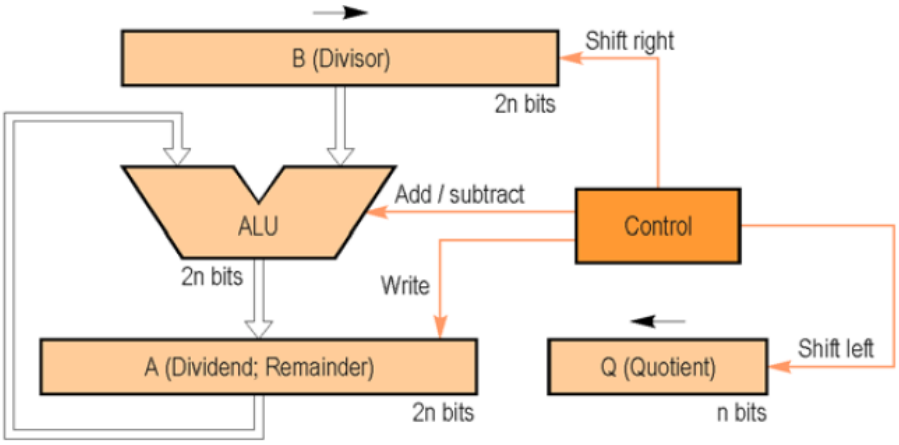
În primul rând, standardul IEEE pentru aritmetică în virgulă flotantă oferă cadrul normativ esențial pentru înțelegerea reprezentării și operațiilor numerelor reale în acest context. Am consultat diverse videoclipuri de pe internet pentru a înțelege algoritmii utilizați și pentru a ajunge la un grad înalt de înțelegere a unor aspecte care m-au ajutat la implementarea cerințelor sistemului de împărțire.

O resursă cheie în această fundamentare este lucrarea "A Floating Point Divider for RC Systems" de R. Jennell Rouse, care furnizează perspectiva detaliată a unui divizor în virgulă flotantă specific pentru sistemele RC. De asemenea, lucrarea "Design of a Single Precision Floating Point Divider and Multiplier with Pipelined Architecture" de Mayuresh Vijay Keni aduce contribuții semnificative, explorând arhitectura pipeline pentru împărțitor și înmulțitor în virgulă flotantă.

În contextul metodelor și modelelor, standardul IEEE 754 continuă să reprezinte o referință esențială pentru operațiile în virgulă flotantă. În paralel, cursurile și laboratoarele

aferente materiei, Structura Sistemelor de Calcul, au furnizat cunoștințe fundamentale și îndrumări practice pentru proiect.

Metoda aleasă pentru implementarea diviziei în virgulă flotantă în limbajul VHDL se bazează pe algoritmul de împărțire cu restaurarea restului. Circuitul implementat, prezentat în figura de mai jos reflectă această abordare:



În fiecare etapă a algoritmului, divizorul este deplasat cu o poziție la dreapta, în timp ce câtul este deplasat cu o poziție la stânga. În stadiul inițial, deîmpărțitul este încărcat în jumătatea dreaptă a registrului A de 2n biți, iar împărțitorul este încărcat în jumătatea stângă a registrului B de 2n biți. Câtul (Q), un registru de n biți, este setat la 0, și contorul N este inițializat cu n+1.

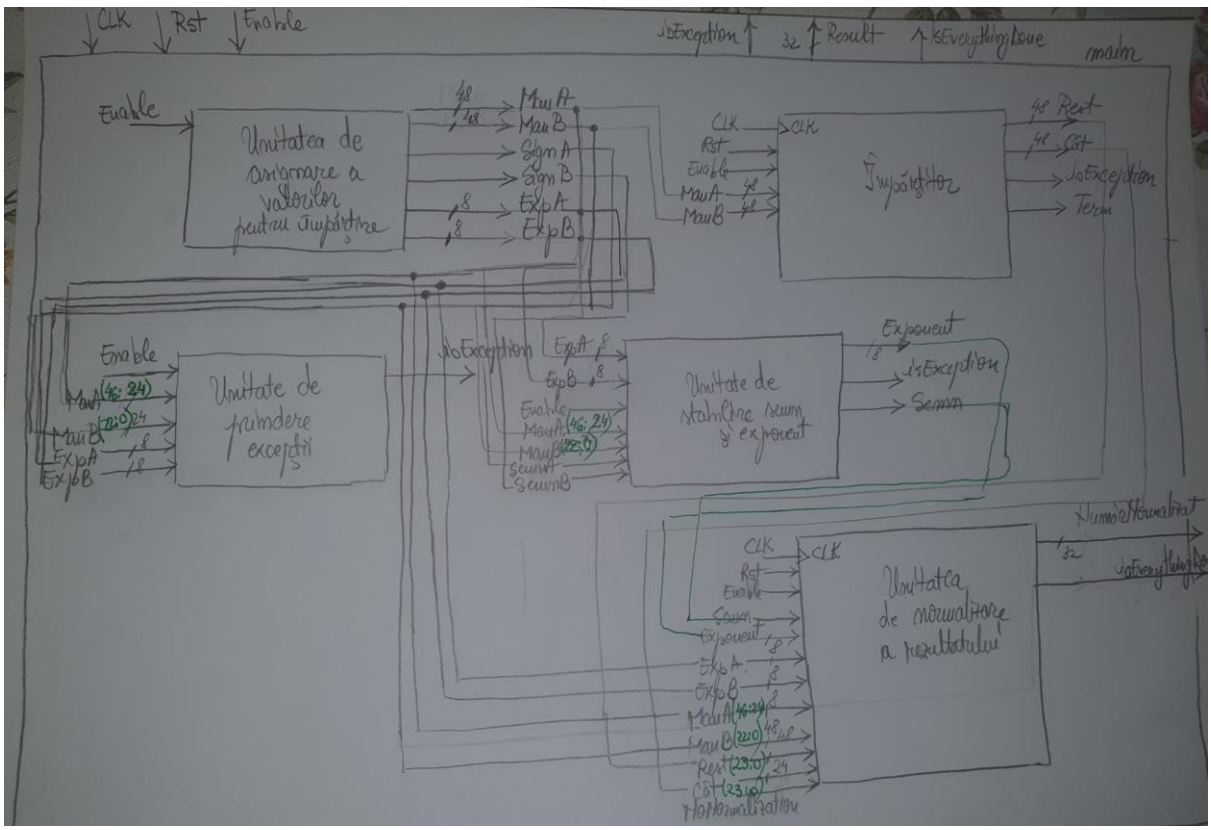
Pentru a determina dacă divizorul este mai mic decât restul parțial, se realizează o operație de scădere a divizorului din rest. În cazul în care rezultatul este negativ, se restaurează valoarea anterioară prin adăugarea divizorului la rest, generând un 0 în poziția Q0 a registrului câtului. Dacă rezultatul este pozitiv, se generează un 1 în poziția Q0. Apoi are loc o shiftare dreapta a registrului B. Acest proces se repetă în fiecare iterație, iar la final, rezultatul final este obținut în registrul A- restul și în Q- câtul.

Proiectul propus își aduce contribuția prin adoptarea unei abordări modulare și eficiente în VHDL, integrând elemente cheie din literatura existentă și aducând inovații în optimizarea resurselor și adaptabilitatea la diverse arhitecturi de sisteme digitale.

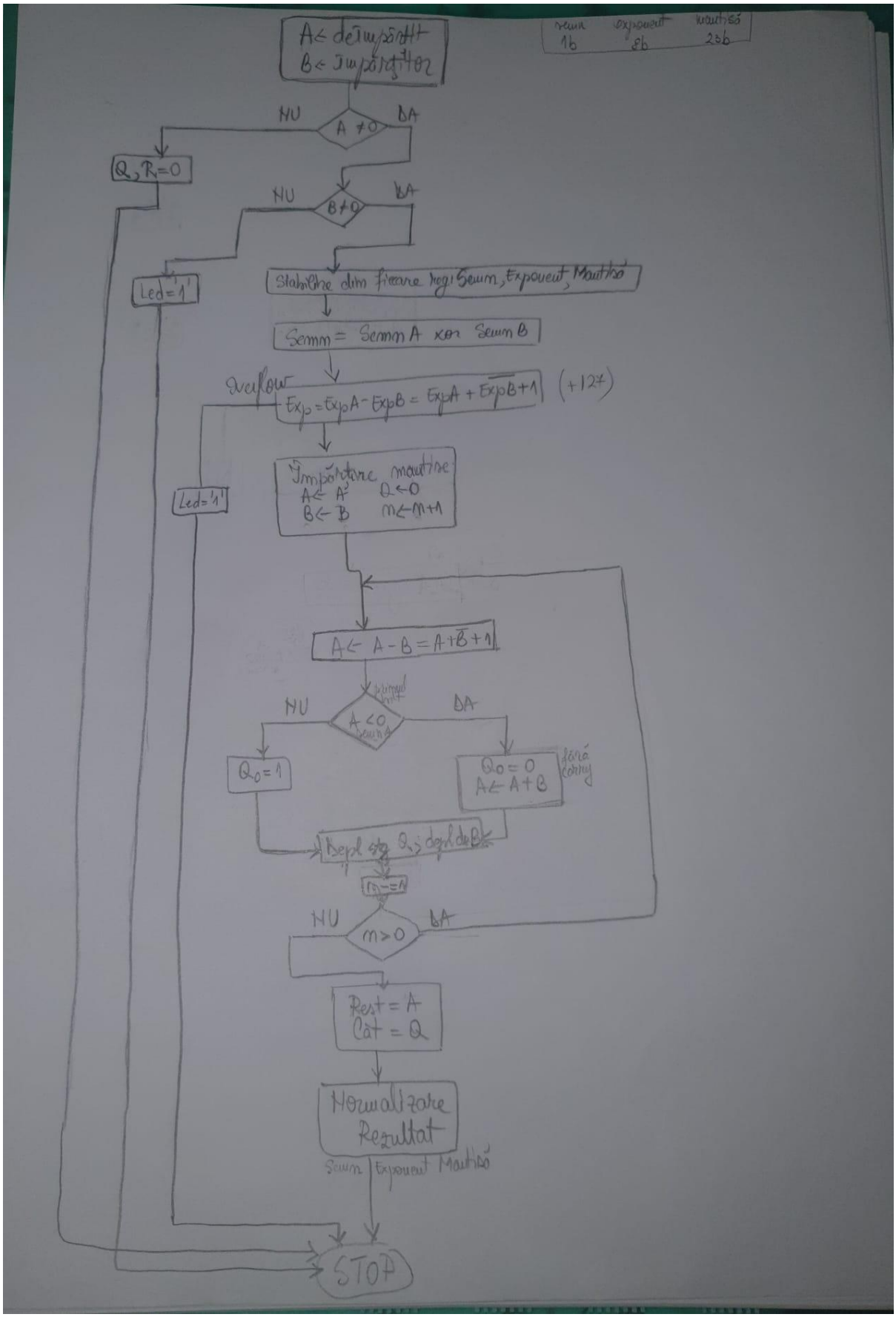
3. Proiectare și implementare

Proiectul a adoptat o abordare de implementare hardware în limbajul VHDL pentru realizarea operației de divizare în virgulă flotantă. Alegerea acestei metode experimentale se bazează pe necesitatea unei soluții eficiente și integrate în hardware, în concordanță cu obiectivele de performanță și adaptabilitate. După cum am prezentat și în secțiunile anterioare, am optat pentru o abordare modulară, bazată pe componente distincte, pentru a asigura eficiența resurselor și ușurința de integrare în diverse arhitecturi. Sistemul de divizare în virgulă flotantă implementat beneficiază de o structură eficientă, evidențiată de utilizarea a trei unități de control distincte. Această abordare modulară include o unitate de control dedicată operației de divizare și una operației de normalizare, optimizând astfel procesul și facilitând monitorizarea și ajustarea parametrilor specifici divizorului și normalizării rezultatului. În paralel, o a treia unitate de control supraveghează întregul sistem, asigurând coordonarea și sincronizarea corectă între diferitele module și etape ale algoritmului. Această diviziune a funcțiilor de control aduce beneficii semnificative în ceea ce privește flexibilitatea sistemului și posibilitatea de a ajusta parametrii în mod independent pentru fiecare componentă.

Schema bloc a sistemului cu componenetele principale

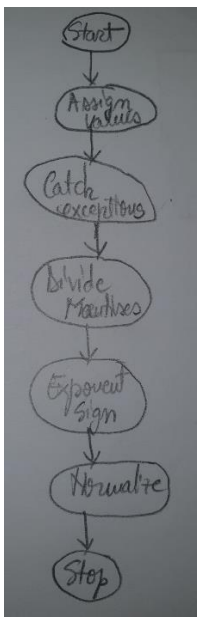


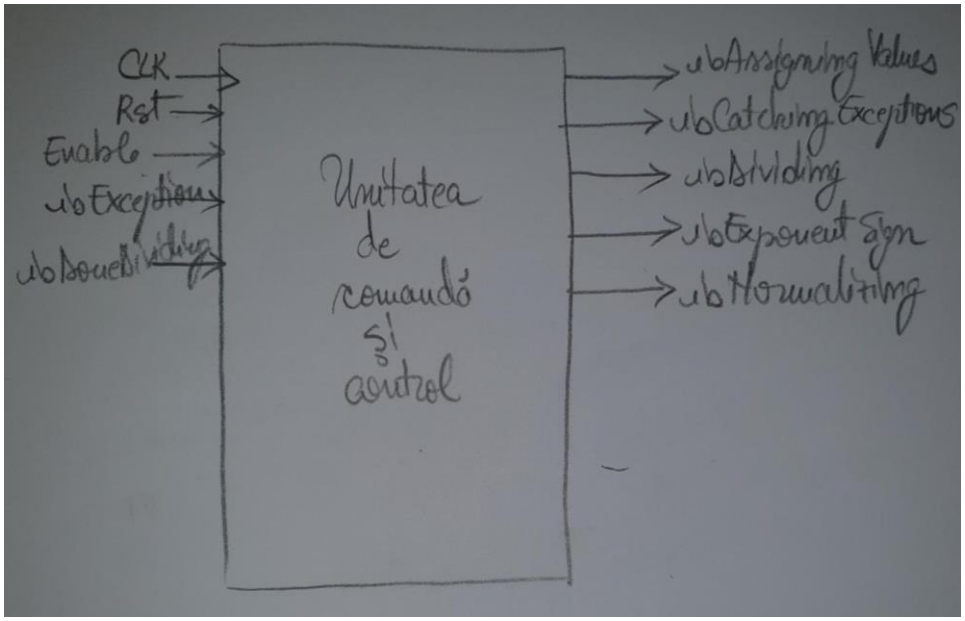
Organigrama



Unitatea principală de comandă și control

Această unitate are rolul de a controla ordinea de desfășurare a operațiilor. Aceasta utilizează un automat de stare pentru a gestiona și coordona diferitele etape ale procesului de divizare în virgulă flotantă. Mai jos este prezentată schema bloc și diagrama de flux a automatului. În starea de început are loc încărcarea valorilor reprezentate deja in virgulă mobilă, am folosit pentru testare site-ul http://weitz.de/ieee/. Apoi se trece în starea de CatchExceptions, în care se analizează comportamentul valorilor de mantisăB si exponentB, iar în caz de excepție se va aprinde un led și următoarea stare va fi Stop. În caz contrar, următoarea stare este DivideMantises, în care este pornită componenta de divizare a celor două mantise. După divizare se trece în starea de stabilire a exponentului și a semnului, ale căror valori împreună cu rezultatul obținut în starea anterioară sunt transmise unității de normalizare în următoarea stare activă. Din starea de normalizare este obținut rezultatul dorit.

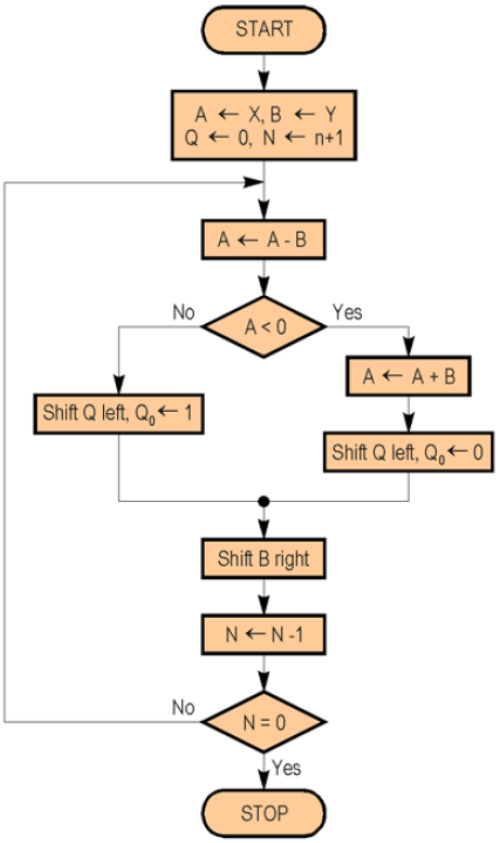




Unitatea de împărțire (divider)

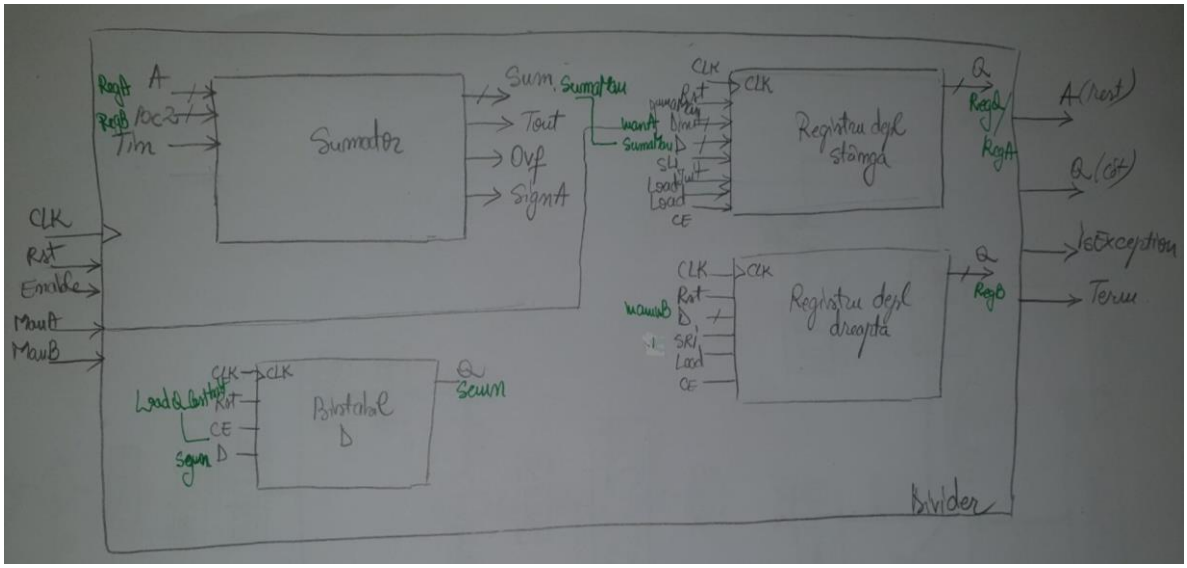
După cum am menționat anterior, metodologia adoptată pentru implementarea operației de divizare în virgulă flotantă în limbajul VHDL se fundamentează pe algoritmul de împărțire cu restaurare. În prima parte a proiectului, am dezvoltat acest modul utilizând o abordare inspirată de modelul Booth pentru împărțire, structurat pe componente individuale. În cadrul acestei implementări, am inclus o unitate de comandă și control dedicată, care supervizează și coordonează procesul de divizare, asigurându-se că fiecare etapă a algoritmului este executată în mod corect și sincronizat.

La implementarea acestui modul am urmat organigrama de mai jos, care prezintă algoritmul de împărțire cu restaurarea restului.

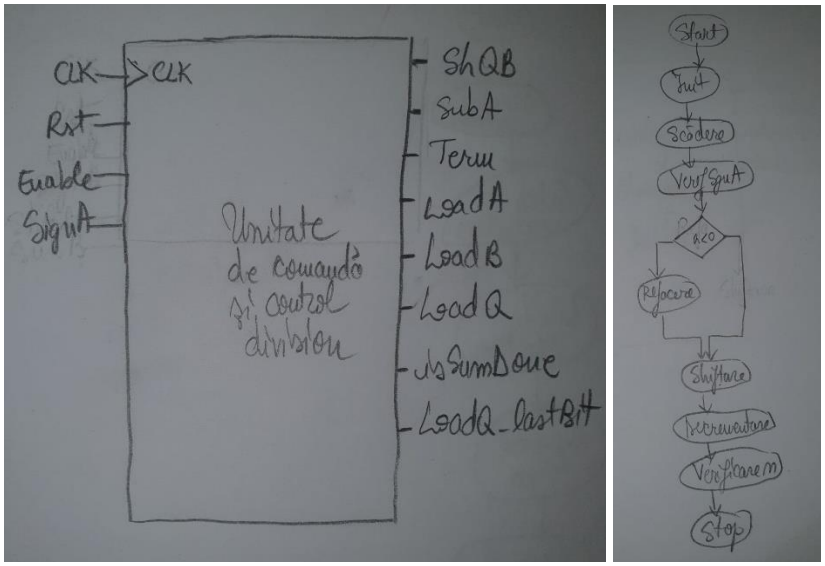


În primă fază, deîmpărțitul este încărcat în partea dreaptă a registrului A de 2n biți, iar împărțitorul este încărcat în partea stângă a registrului B de 2n biți. Registrul câtului Q, având n biți, este inițializat la 0, iar contorul N este setat la n+1. Pentru a determina dacă împărțitorul este mai mic decât restul parțial, se efectuează o scădere a registrului împărțitor(B) din registrul deîmpărțit(A). Dacă rezultatul este negativ, etapa următoare constă în restaurarea valorii anterioare prin adăugarea împărțitorul la deîmpărțit, generând un 0 în poziția Q0 a registrului câtului. Acesta este motivul pentru care această metodă este denumită divizare cu restaurare. În cazul în care rezultatul este pozitiv, se generează un 1 în poziția Q0 a registrului câtului. În etapa următoare, împărțitorul este deplasat spre dreapta, aliniindu-l cu deîmpărțitul pentru iterarea ulterioară. Algoritmul se repetă de n+1 ori, iar rezultatul se află: in registrul A - restul și în registrul Q - câtul.

Pentru a realiza acest algoritm m-am folosit de următoarea schemă bloc:



La sincronizarea stărilor prin care trebuie să treacă automatul am folosit o unitate de comandă și control, cu schema bloc și diagrama de flux a stărilor:



Prima stare prin care trece automatul este de Init în care are loc încărcarea valorilor mantiselor în registrele A,B și valorii 0 în registrul Q. Următoarea stare este Scăderea celor două registre A și B, pe care am realizat-o ca o adunare A+B negat +1 (reprezentarea lui B în complement față de 2, am folosit un ripple carry adder). În următoarea stare se verifică semnul scăderii anterioare, în cazul în care este negativ, urmează starea de refacere a registrului A. Următoarea stare este Shiftarea, care presupune shift dreapta registrul B și shift stânga a registrului Q cu adăugarea pe ultima poziție Q0 semnul produs de scăderea

anterioară. În starea de Verificare, se verifică dacă mai există iterații de făcut, în caz afirmativ, algoritmul se reia, iar în contrar, se oprește.

Cele două registre de deplasare sunt comune, cel de stânga are 2 tipuri de load, pentru ca să poată fi folosit atât la încărcarea valorii în starea Init, cât și valorii de sumă. În caz de Load activ, se pune valorare din D/Dinit în registrul de output. În caz de enable la shiftare pe bitul care rămâne neocupat se pune valoarea SLI/SRI (care este semnul rezultatului scăderii).

Bistabilul de tip D are rolul de păstrare pentru starea de shiftare a bitului de semn stabilit în starea de Scădere. Sumatorul este format prin stabilirea fiecărui bit de adunare cu transportul de la pasul anterior. În final, semnul operației este stabilit ca fiind primul bit din rezultatul sumei. De asemenea se stabilește și setarea bitului de overflow.

Unitatea de asignare a valorilor (assign_values)

În cadrul acestui modul, se desfășoară procesul delicat de încărcare a valorilor asociate Semnului, Mantisei și Exponentului pentru cele două numere. Acest pas esențial are loc atunci când semnalul de activare este pe 1 logic, asigurând o pregătire atentă și precisă a datelor pentru etapele ulterioare.

Unitatea de prindere a excepțiilor (catch_exceptions)

În această unitate, se gestionează situațiile excepționale cum ar fi valoarea 0 pe mantisa și exponentul împărțitorului. În caz de excepție identificată se setează bitul de execepție pe 1 logic.

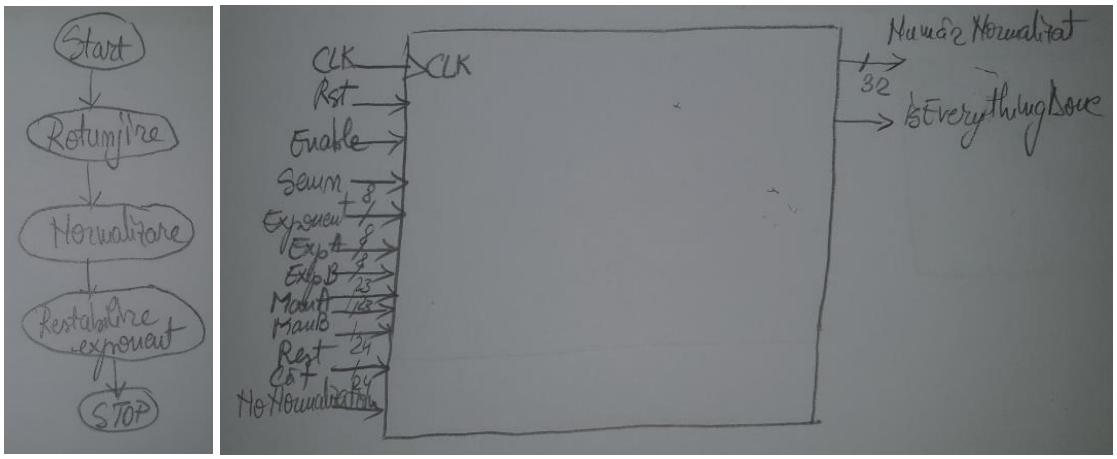
Unitatea de stabilirea a semnului și exponentului

În cazul semnului se face operația xor între cele două semne, iar la exponentul este calculat prin scăderea celor doi exponenți (am făcut scăderea pe signed și nu a mai fost nevoie să adun 127). De asemenea, stabilesc și anumite cazuri de prindere a unor excepții de underflow/overflow.

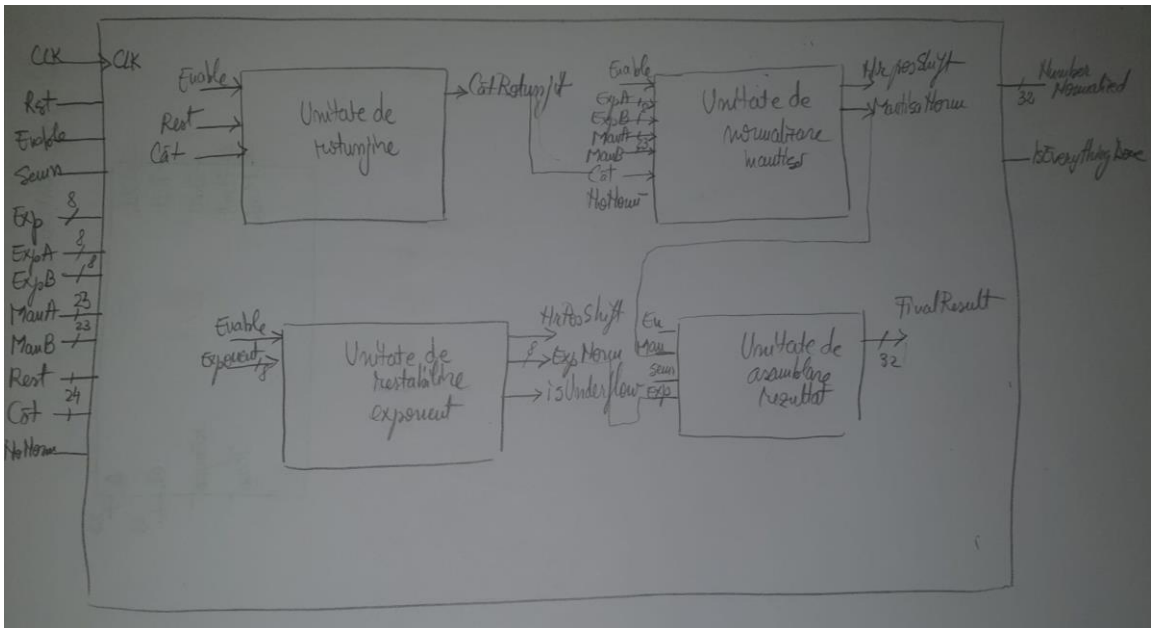
Unitatea de normalizare

Unitatea de normalizare reprezintă un element crucial în procesul de divizare în virgulă flotantă, aducând coerență și eficiență operațiilor desfășurate. Rostul său esențial constă în ajustarea rezultatului divizării, astfel încât să respecte forma standard a numerelor în virgulă flotantă.

Și în cazul unității de normalizare am abordat aceeași metodă ca la divider, în care am sincronizat stările prin intermediul unui automat de stare. Schema bloc și diagrama de stări:



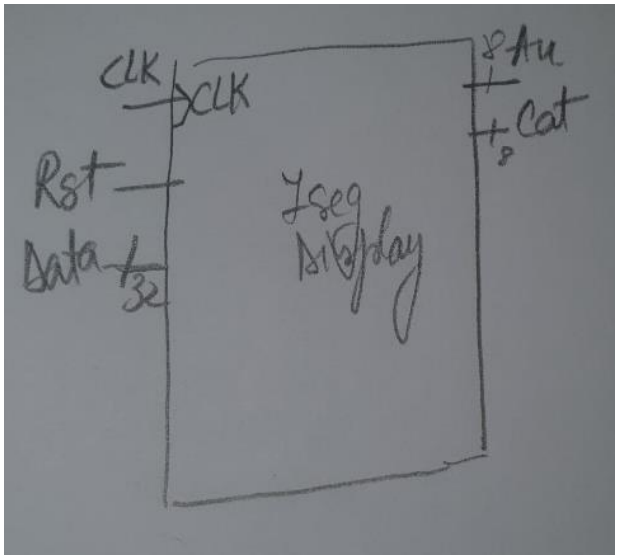
Această unitate are misiunea de a corecta exponentul și mantisa rezultatului, astfel încât să obținem o reprezentare normalizată, eliminând zerourile neesențiale din mantisă și asigurând acuratețea și coerența datelor rezultate. Schema bloc a acestei unități este următoarea:



Prima stare prin care trece automatul este cea de Rotunjire, în care am implementat un algoritm simplu de rotunjire a câtului în funcție de valoarea restului, se adaugă 1 la cât dacă restul trece peste jumătate. Următoarea stare este de normalizare, se shiftează la stânga mantisa-cât cu x poziții până la întâlinirea primului bit de 1, păstrând numărul de biți de 0 eliminați. Aici am mai tratat și anumite cazuri de excepție(0/0 sau nr/0), întrucât era punctul în care aveam majoritatea valorilor necesare calculate în pașii anteriori. Următorul pas este restabilirea exponentului, prin scăderea din acesta a valorii primite de la unitatea de normalizare a mantisei, numărul de poziții shiftate. Ultima stare este cea de asamblare a rezultatului, prin care se constituie vectorul de 32biți din bitul de semn, cei 8 biți de exponent și 23 biți de mantisă.

Unitatea de afișare 7 seg

Modulul display7seg este proiectat pentru a facilita afișarea informațiilor pe un display cu 7 segmente. Funcția principală este de a traduce valorile hexazecimale conținute în vectorul Data în secvențe specifice de aprindere a segmentelor, permițând astfel o reprezentare vizuală corespunzătoare pe display. Acest modul utilizează un contor și selecția ciclică a display-urilor pentru a asigura afișarea corectă a fiecărei cifre hexazecimale. Prin intermediul semnalelor An și Seg, modulul coordonează procesul de afișare, aducând astfel o contribuție esențială la reprezentarea vizuală a datelor hexazecimale în cadrul proiectului. Cu ajutorul acestui modul am realizat afișarea rezultatului pe plăcuța Nexis4. În vectorul Data este încărcată valoarea rezultată după normalizare. Schema bloc:



4. Rezultate experimentale

Pentru evaluarea performanțelor sistemului de divizare în virgulă flotantă implementat în limbajul VHDL, am utilizat o procedură de testare cuprinzătoare. Procesul de testare a implicat asignarea de valori semnificative atât pentru deîmpărțit, cât și pentru împărțitor în cadrul modulului assign_values. Aceste valori au inclus semnul, mantisa și exponentul pentru ambele numere.

Am dezvoltat bancuri de test elaborate pentru a verifica corectitudinea operațiilor sistemului în diverse scenarii și condiții. Utilitarele de conversie au jucat un rol crucial în verificarea preciziei rezultatelor obținute în urma divizării. Aceste programe au facilitat conversia datelor în formatele corespunzătoare pentru a asigura că sistemul interpretează și procesează datele în mod adecvat (http://weitz.de/ieee/).

În cadrul simulărilor, am inițiat funcționarea sistemului prin activarea semnalului Enable la nivel logic '1'. Astfel, am obținut rezultatele normalizate, iar acestea au fost comparate cu valorile așteptate pentru a valida corectitudinea implementării.

Procedura de testare s-a dovedit esențială pentru evidențierea performanțelor și funcționării corespunzătoare a sistemului. Acest demers a inclus depășirea unor scenarii critice și a facilitat o evaluare detaliată a modului în care sistemul gestionază și procesează datele în cadrul operațiilor de divizare în virgulă flotantă.

Pe parcursul implementării, am întâmpinat provocări tehnice, precum gestionarea eficientă a resurselor hardware și sincronizarea adecvată a modulelor. Cu toate acestea, aceste obstacole au fost abordate cu succes, contribuind la obținerea unor rezultate valide și la funcționarea corespunzătoare a sistemului în ansamblu.

În procesul extins de testare al sistemului, am beneficiat de resurse online, în special de site-ul http://weitz.de/ieee/. Acest site oferă o interfață intuitivă care permite introducerea de valori în format zecimal, urmată de conversia acestora în reprezentarea în virgulă mobilă 32biți (floating-point) conform standardului IEEE 754. Această platformă a constituit o resursă valoroasă pentru generarea deîmpărțiților și împărțitorilor diversificați și complexi, având posibilitatea de a explora scenarii variate de testare. După introducerea datelor, am avut opțiunea de a alege operația de divizare, iar rezultatele obținute au fost comparate cu rezultatele simulărilor sistemului nostru implementat în limbajul VHDL. Această metodă a contribuit la validarea performanțelor și preciziei algoritmului de divizare în contexte realiste și diverse.

În cadrul testarii am luat în calcul următoarele cazuri, unele intră în calculul normal, altele în cazurile de excepție:

1

mantisaA	0
0	0

Inf => mantisa=0...; exp=1..

2

0	exponentA
0	0

Inf => mantisa=0...; exp=1..

3

0	0
0	exponentB

0 => mantisa=0...; exp=0...

4

0	0
mantisaB	0

0 => mantisa=0...; exp=0...

5

0	0
0	0

Nan=> mantisa=1...; exp=1..

6

mantisaA	exponentA
0	0

Inf => mantisa=0..; exp=1..

10

mantisaA	0
mantisaB	exponentB

9

0	exponentA
mantisaB	exponentB

12

mantisaA	0
mantisaB	0

13

0	exponentA
0	exponentB

14

mantisaA	0
0	exponentB

15

0	exponentA
mantisaB	0

7

0	0
mantisaB	exponentB

0 => mantisa=0..; exp=0...

8

mantisaA	exponentA
mantisaB	0

Inf => mantisa=0..; exp=1..

9

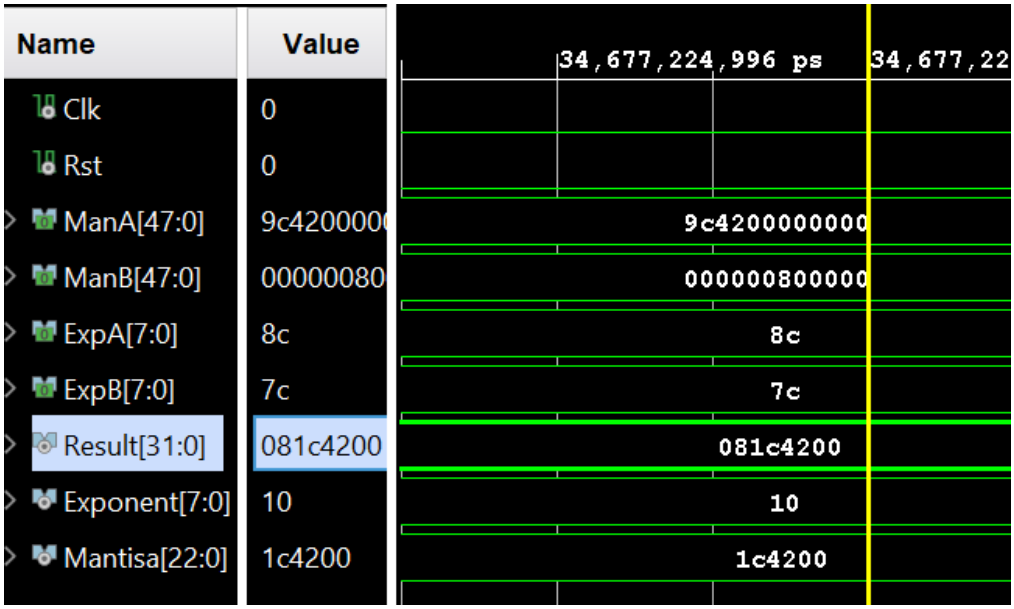
mantisaA	exponentA
0	exponentB

16

mantisaA	exponentA
mantisaB	exponentB

În secțiunea următoare, se vor prezenta capturi de ecran ilustrative, evidențiind simulările realizate pentru sistemul implementat. Fiecare imagine va ilustra scenarii specifice de testare, oferind astfel o perspectivă vizuală asupra performanțelor și rezultatelor obținute în timpul simulărilor. Valorile din simulator sunt în hexa, voi atașa și o conversie la mantise a valorii binare în hexazecimal. Exemple:

1.



0001 1100 0100 0010 0000 0000

HEX 1C 4200

	Sign	Significand	Exponent
487.5	0	1 . 11100111100000000000000	10000111
	+	1.9042969	+8

0x43F3C000

0b01000011111100111100000000000000

2.

	Sign	Significand	Exponent
0.125	0	1 . 00000000000000000000000	01111100
	+	1.0	-3

0x3E000000

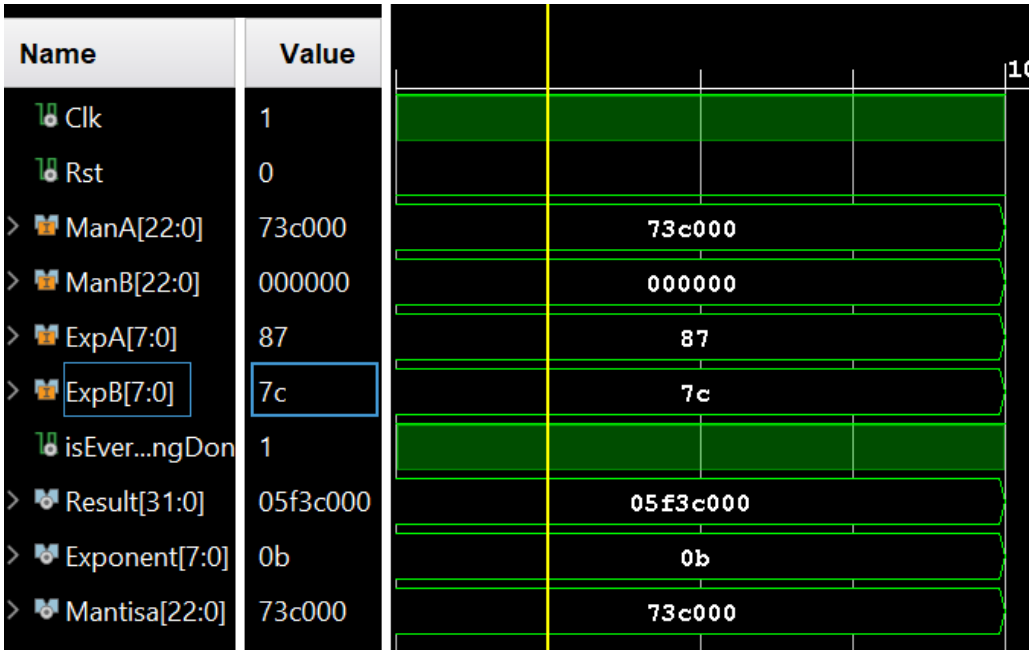
0b00111111000000000000000000000000

+ - × /

	Sign	Significand	Exponent
3900.0	0	1 . 11100111100000000000000	10001010
	+	1.9042969	+11

0x4573C000

0b01000101011100111100000000000000



0111 0011 1100 0000 0000 0000

HEX 73 C000

	Sign	Significand	Exponent
0.5	0	1 . 00000000000000000000000	01111110
	+	1.0	-1

0x3F000000

0b00111111000000000000000000000000

	Sign	Significand	Exponent
0.125	0	1 . 00000000000000000000000	01111100
	+	1.0	-3

0x3E000000

0b00111110000000000000000000000000

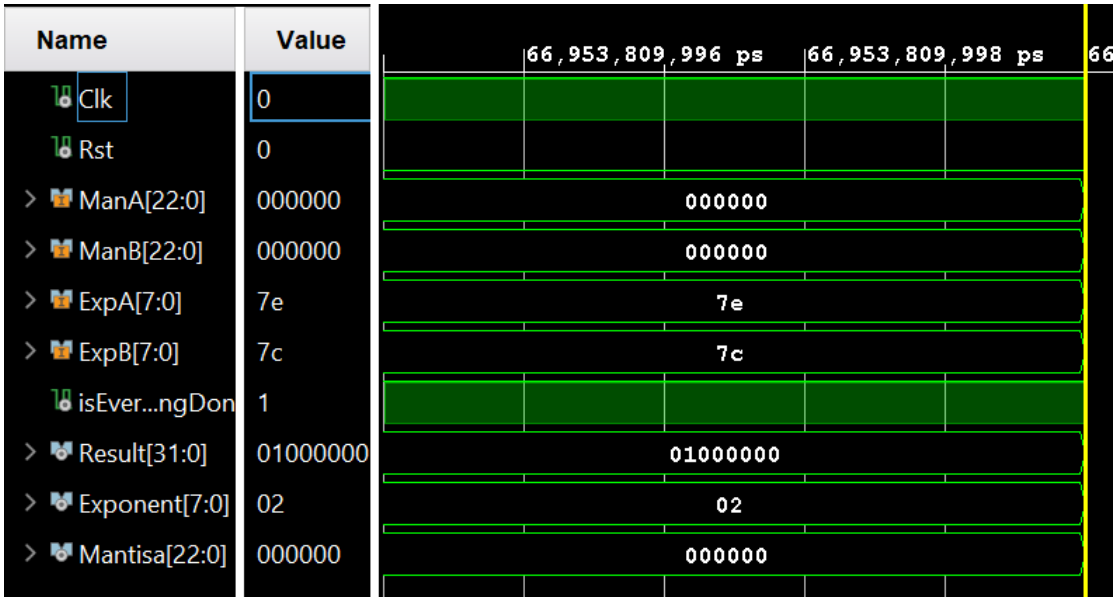
3.

+ - × /

	Sign	Significand	Exponent
4.0	0	1 . 00000000000000000000000	10000001
	+	1.0	+2

0x40800000

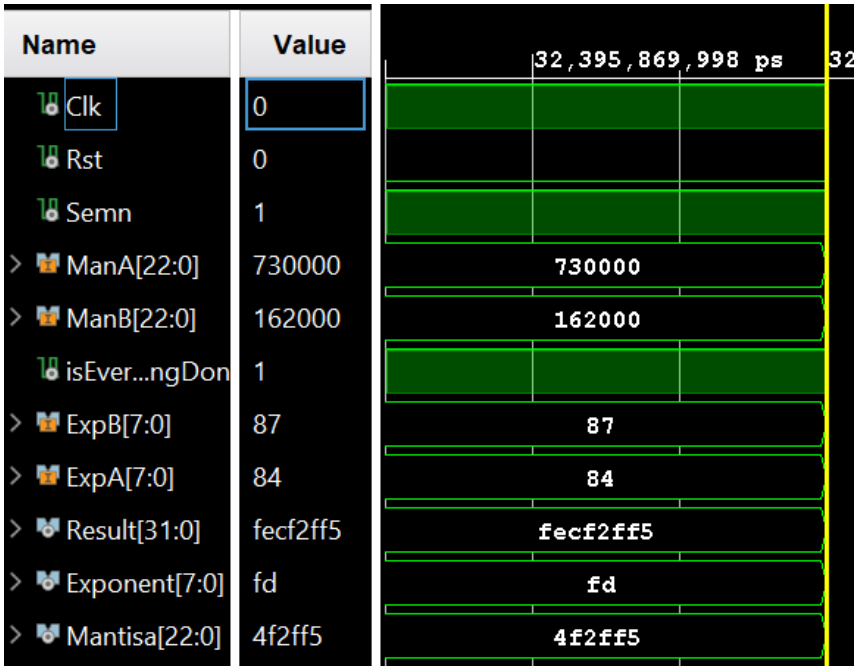
0b01000000100000000000000000000000

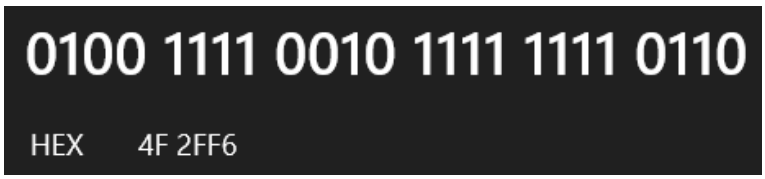


	Sign	Significand	Exponent
-60.75	1	1 . 11100110000000000000000	10000100
	-	1.8984375	+5
		0xC2730000	
		0b11000010011100110000000000000000	
300.25	0	1 . 00101100010000000000000	10000111
	+	1.1728516	+8
		0x43962000	
		0b01000011100101100010000000000000	

+ - × /

-0.2023314	1	1 . 10011110010111111110110	01111100
	-	1.6186512	-3
		0xBE4F2FF6	
		0b10111110010011110010111111110110	

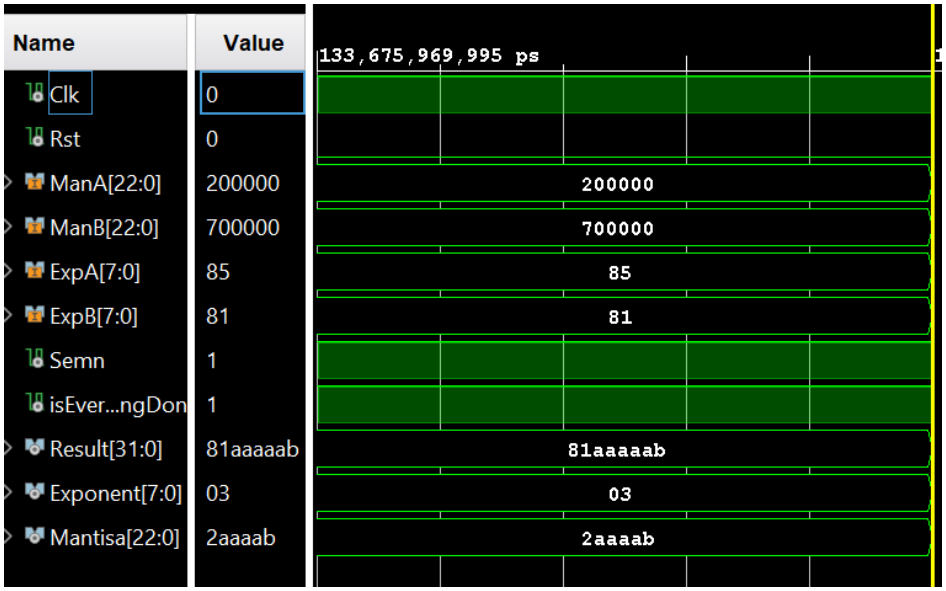




5

	Sign	Significand	Exponent
80.0	0	1 . 01000000000000000000	10000101
	+	1.25	+6
	0x42A00000		
	0b01000010101000000000000000000000		
7.5	0	1 . 11100000000000000000	10000001
	+	1.875	+2
	0x40F00000		
	0b01000000111100000000000000000000		
10.666667	0	1 . 01010101010101010101	10000010
	+	1.3333334	+3
	0x412AAAAB		
	0b01000001001010101010101010101011		

+ - × /



	Sign	Significand	Exponent
80.0	0	1 . 01000000000000000000000	10000101
	+	1.25	+6

0x42A00000
0b01000010101000000000000000000000

	Sign	Significand	Exponent
0.0	0	0 . 00000000000000000000000	00000000
	+	0.0	+0

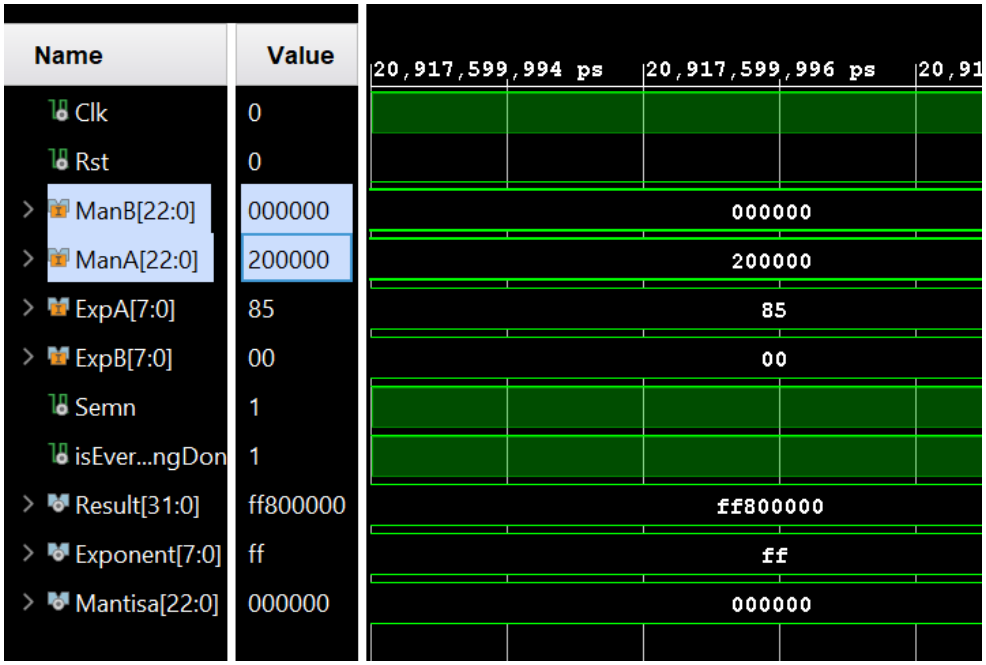
0x00000000
0b00000000000000000000000000000000

6

+ - × /

	Sign	Significand	Exponent
Inf	0	. 00000000000000000000000	11111111
	+	Inf	

0x7F800000
0b01111111100000000000000000000000



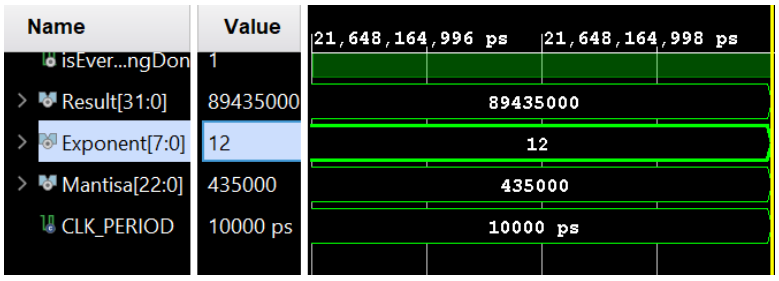
	Sign	Significand	Exponent
100000.0	0	1 . 10000110101000000000000	10001111
	+	1.5258789	+16
		0x47C35000	
		0b01000111111000011010100000000000	

7

	Sign	Significand	Exponent
0.25	0	1 . 00000000000000000000000	01111101
	+	1.0	-2
		0x3E800000	
		0b00111111010000000000000000000000	

+ - × /

	Sign	Significand	Exponent
400000.0	0	1 . 10000110101000000000000	10010001
	+	1.5258789	+18
		0x48C35000	
		0b01001000110000110101000000000000	



0100 0011 0101 0000 0000 0000

HEX 43 5000

5. Rezumat

Proiectul propune o soluție eficientă pentru operații de divizare în virgulă flotantă în limbajul VHDL, cu un accent deosebit pe implementarea modulară și gestionarea resurselor. Metoda adoptată se bazează pe o variantă a algoritmului de împărțire cu restaurare, dezvoltată într-o abordare Booth. Componentele individuale sunt integrate printr-o unitate de comandă și control (cele principale având la rândul lor câte o unitate- divider si normalizare), asigurând o coordonare precisă a operațiilor sistemului. Pentru validarea performanțelor, am efectuat simulări extinse, utilizând date generate atent pe site-ul http://weitz.de/ieee/. Rezultatele obținute au fost comparate cu simulările pentru a confirma corectitudinea și precizia soluției. Proiectul reprezintă o realizare semnificativă în dezvoltarea unui modul de divizare eficient, furnizând baza pentru extinderi sau optimizări ulterioare.

6. Concluzii

În final, proiectul a reușit să ofere o soluție eficientă și precisă pentru operațiile de divizare în virgulă flotantă în limbajul VHDL. Prin implementarea unei abordări bazate pe algoritmul de împărțire cu restaurare, adaptată într-un cadru Booth, am obținut rezultatele dorite, confirmate prin simulări și teste extinse. Avantajele proiectului includ precizia operațiilor de divizare, adaptabilitatea la diverse seturi de date și o implementare modulară ușor de extins. Desigur, există întotdeauna loc pentru îmbunătățiri. O posibilă direcție de dezvoltare viitoare ar fi optimizarea performanțelor, luând în considerare cerințele specifice ale anumitor aplicații. Totodată, o extindere a proiectului pentru a acoperi și alte operații aritmetice complexe ar aduce un aport semnificativ în domeniul structurilor sistemelor de calcul. În concluzie, proiectul nu doar a îndeplinit obiectivele propuse, dar și a deschis perspective interesante pentru viitorul dezvoltărilor în acest domeniu.

7. Bibliografie/ Webografie

https://users.utcluj.ro/~baruch/media/ssc/curs/SSC-Impartire.pdf

https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=10862&context=theses

https://klabs.org/DEI/Arithmetic/division/fp_divider.pdf

https://iremi.univ-reunion.fr/IMG/pdf/ieee-754-2008.pdf

https://users.utcluj.ro/~baruch/media/ssc/curs/SSC-VM.pdf

http://weitz.de/ieee/