Proiectarea unei unități aritmetico-logice în virgulă flotantă care efectuează operațiile de adunare și înmulțire

Mesaroș Radu-Andrei

Grupa 30233

Cuprins

Introducere

1.1 Context

Scopul acestui proiect este de a dezvolta o unitate aritmetico-logică în virgulă flotantă (ALU) care să realizeze operațiuni de adunare și înmulțire pe numere în virgulă flotantă, conform standardului IEEE 754. ALU-ul în virgulă flotantă joacă un rol esențial în performanța unui sistem de calcul modern, deoarece multe aplicații științifice și grafice necesită o manipulare eficientă a numerelor reale.

Dezvoltarea acestei unități permite implementarea operațiilor cu numere mari sau foarte mici, care nu pot fi reprezentate în format întreg. ALU-ul va fi proiectat utilizând VHDL și va fi integrat într-un proiect Xilinx Vivado, împreună cu un testbench pentru simulare. Acest design poate fi ulterior programat pe o placă FPGA pentru verificare hardware.

1.2 Obiective

Unitatea aritmetico-logică în virgulă flotantă va efectua următoarele operațiuni de bază:

-Adunarea a două numere în virgulă flotantă

-Înmulțirea a două numere în virgulă flotantă

Designul va respecta standardul IEEE 754, utilizând formatul pe 32 de biți (simplă precizie). În cadrul acestui proiect, unitatea va fi optimizată pentru a evita erorile de precizie și pentru a asigura rezultate corecte în toate cazurile de coliziune între operanzi. Unitatea va include, de asemenea, un testbench pentru a simula corectitudinea operațiilor de adunare și înmulțire.

Studiu Bibliografic

2.1 Ce este o unitate aritmetico-logică în virgulă flotantă?

O unitate aritmetico-logică în virgulă flotantă (FPU - Floating Point Unit) este un component esențial al unui procesor care efectuează operații aritmetice pe numere reale, reprezentate în format virgulă flotantă. În contrast cu aritmetica pe întregi, aritmetica în virgulă flotantă permite reprezentarea unui interval mult mai larg de valori, făcând-o ideală pentru calcule științifice, financiare și grafice. Standardul IEEE 754 definește modul în care numerele reale sunt codificate și cum se efectuează operațiile aritmetice cu acestea.

2.2 Implementarea operațiilor de adunare și înmulțire

Pentru a realiza adunarea și înmulțirea în virgulă flotantă, este necesar să se urmeze câțiva pași specifici în conformitate cu standardul IEEE 754:

Adunarea în virgulă flotantă

1. **Alinierea exponenților**: Dacă cei doi operanzi au exponenți diferiți, numărul cu exponentul mai mic va fi aliniat, adică mantisa acestuia va fi deplasată pentru a face exponenții egali.
2. **Adunarea mantiselor**: Mantisele ajustate vor fi adunate. Rezultatul poate necesita normalizare dacă mantisa depășește limitele acceptabile.
3. **Normalizarea rezultatului**: Dacă mantisa rezultată este prea mare sau prea mică, va trebui să fie normalizată, prin deplasarea mantisei și ajustarea exponentului corespunzător.
4. **Gestionarea erorilor și excepțiilor**: Trebuie tratate cazurile speciale precum NaN (Not a Number), infinit, și zero.

Înmulțirea în virgulă flotantă

1. **Adunarea exponenților**: Exponenții celor doi operanzi sunt adunați și ajustați conform regulilor IEEE 754.
2. **Înmulțirea mantiselor**: Mantisele celor doi operanzi sunt înmulțite. Mantisa rezultată poate necesita normalizare.
3. **Normalizarea rezultatului**: Ca și în cazul adunării, mantisa trebuie normalizată dacă depășește limitele. Exponentul este ajustat corespunzător.
4. **Gestionarea excepțiilor**: Și aici sunt necesare tratamente speciale pentru cazuri precum NaN, infinit sau zero.

2.3 Erori și excepții

Erorile de precizie sunt inevitabile în aritmetica în virgulă flotantă, acestea fiind:

- **Overflow**: Rezultatul operației depășește valorile maxime permise de formatul în virgulă flotantă.
- **Underflow**: Rezultatul operației este prea mic pentru a fi reprezentat corect.
- **NaN și infinit**: Trebuie tratate corespunzător pentru a preveni erori de calcul sau oprirea procesorului.

Analiza

3.1 Propunerea Proiectului

Unitatea aritmetico-logică în virgulă flotantă (ALU) dezvoltată în acest proiect va integra funcționalitățile esențiale pentru operarea cu numere reale, conform standardului IEEE 754. Acest ALU este proiectat să ofere precizie și eficiență pentru operațiile de adunare și înmulțire, abordând și gestionarea cazurilor speciale și excepțiilor.

3.1.1 Funcționalitățile unității ALU în virgulă flotantă

1. **Instrucțiuni de bază pentru operații în virgulă flotantă**
   ALU-ul include suport pentru două instrucțiuni fundamentale, adunarea și înmulțirea în virgulă flotantă, conforme cu formatul IEEE 754 pe 32 de biți.

2. **Gestionarea cazurilor de eroare și excepție**
   Implementarea tratează situațiile de **overflow** și **underflow**, precum și valorile speciale **NaN** și **infinit**, conform regulilor IEEE 754, pentru a asigura stabilitatea și corectitudinea calculului în orice context.

3. **Normalizarea și alinierea exponenților**
   În cazul adunării și înmulțirii, sunt incluse circuite dedicate pentru alinierea exponenților și normalizarea rezultatului, asigurându-se astfel că rezultatele respectă structura standardizată și optimizată a IEEE 754.

4. **Testare prin programe de simulare**
   Proiectul include dezvoltarea unui set de programe de testare (test benches) pentru a valida corectitudinea ALU-ului în diferite scenarii, inclusiv operații pe numere mari și foarte mici, pentru a verifica gestionarea corectă a valorilor excepționale și a exponenților diferiți.

Aici avem o diagrama pentru procesul de adunare

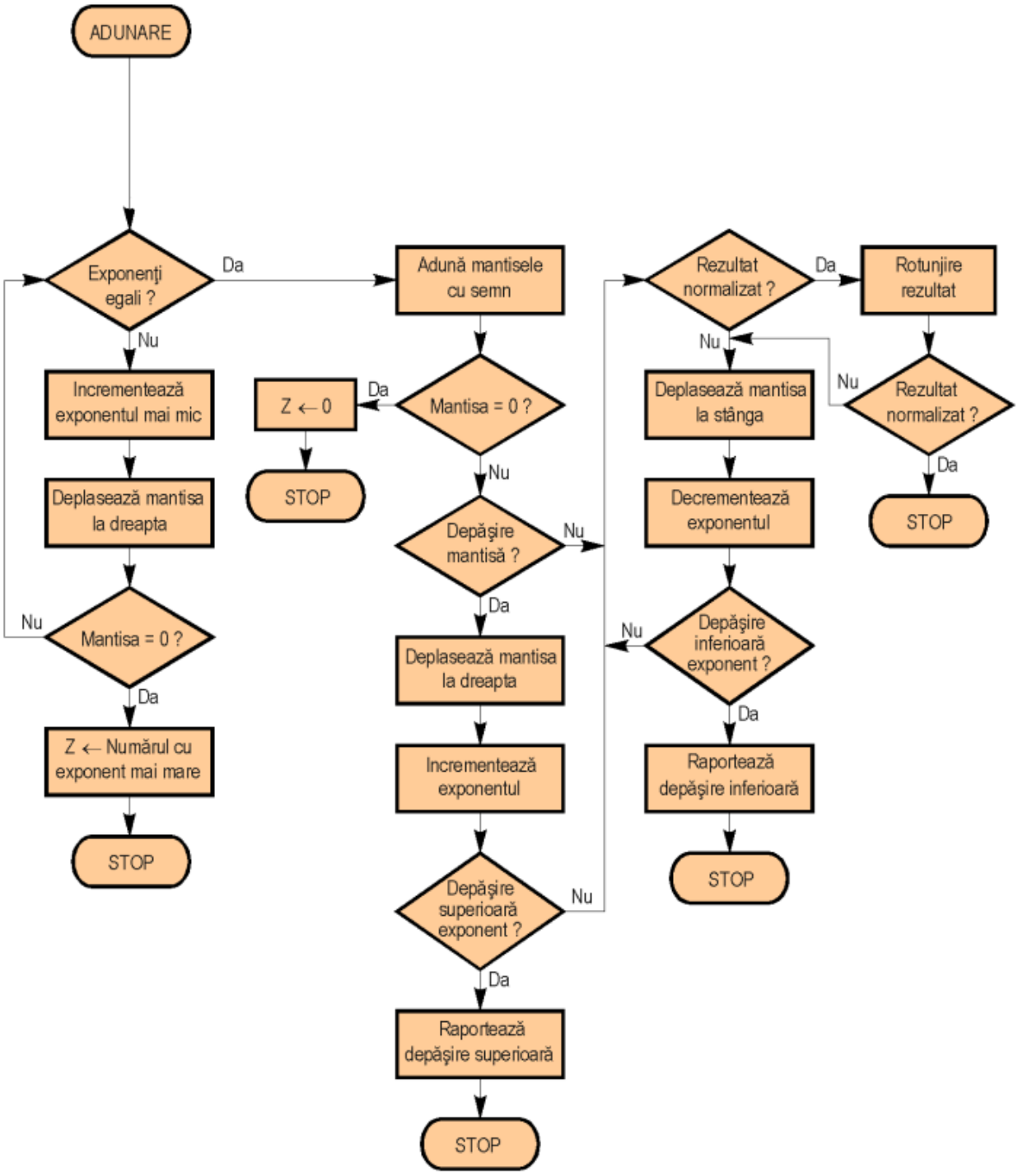


Figura 6.17 Adunarea in virgula flotanta – Baruch, Arhitectura Calculatoarelor

1. Verificarea Exponenților (Alinierea Exponenților)

- **Scop**: Este necesar ca ambii operanzi să aibă același exponent pentru a putea aduna mantisele corect.
- **Proces**:
    - Dacă exponenții celor două numere nu sunt egali, identificăm exponentul mai mic și incrementăm acest exponent pentru a-l aduce la nivelul celuilalt.
    - Pentru a păstra valoarea inițială a numărului cu exponentul mai mic, se deplasează mantisa acestui număr la dreapta, în același timp în care se incrementează exponentul.
    - Dacă exponenții sunt egali după acest pas, se trece la adunarea mantiselor.

2. Adunarea Mantiselor

- **Scop**: Se adună mantisele celor două numere pentru a obține rezultatul preliminar al adunării.
- **Proces**:
    - Mantisele ajustate (aliniate) sunt adunate împreună.
    - Dacă în urma adunării mantiselor se obține o valoare zero (adică, rezultatul adunării este zero), procesul se oprește aici, și rezultatul final este zero.
    - Dacă mantisa rezultată depășește valoarea maximă reprezentabilă, apare o "depășire de mantisă" (overflow), ceea ce necesită normalizarea rezultatului.

3. Normalizarea Rezultatului

- **Scop**: Normalizarea este necesară pentru a asigura că mantisa se încadrează în intervalul valid și că reprezentarea rezultatului este corectă.
- **Proces**:
    - Dacă mantisa depășește limitele acceptabile (adică este prea mare), aceasta este deplasată la dreapta cu un bit pentru a o aduce în intervalul valid.
    - Deplasarea mantisei la dreapta necesită creșterea exponentului cu o unitate pentru a menține valoarea numerică corectă.
    - Dacă mantisa este prea mică după adunare, se efectuează deplasarea mantisei la stânga, iar exponentul este decrementat pentru a păstra valoarea numerică.

4. Verificarea Rezultatului Normalizat

- **Scop**: După normalizare, este necesar să verificăm dacă rezultatul este într-adevăr normalizat (adică mantisa este în intervalul corect).
- **Proces**:
    - Dacă rezultatul nu este normalizat nici după prima normalizare, se repetă procesul: mantisa este deplasată la stânga și exponentul decrementat până când se obține o valoare corectă.
    - În unele cazuri, când exponentul scade prea mult, se ajunge la o situație de "suboverflow" sau "depășire inferioară de exponent".

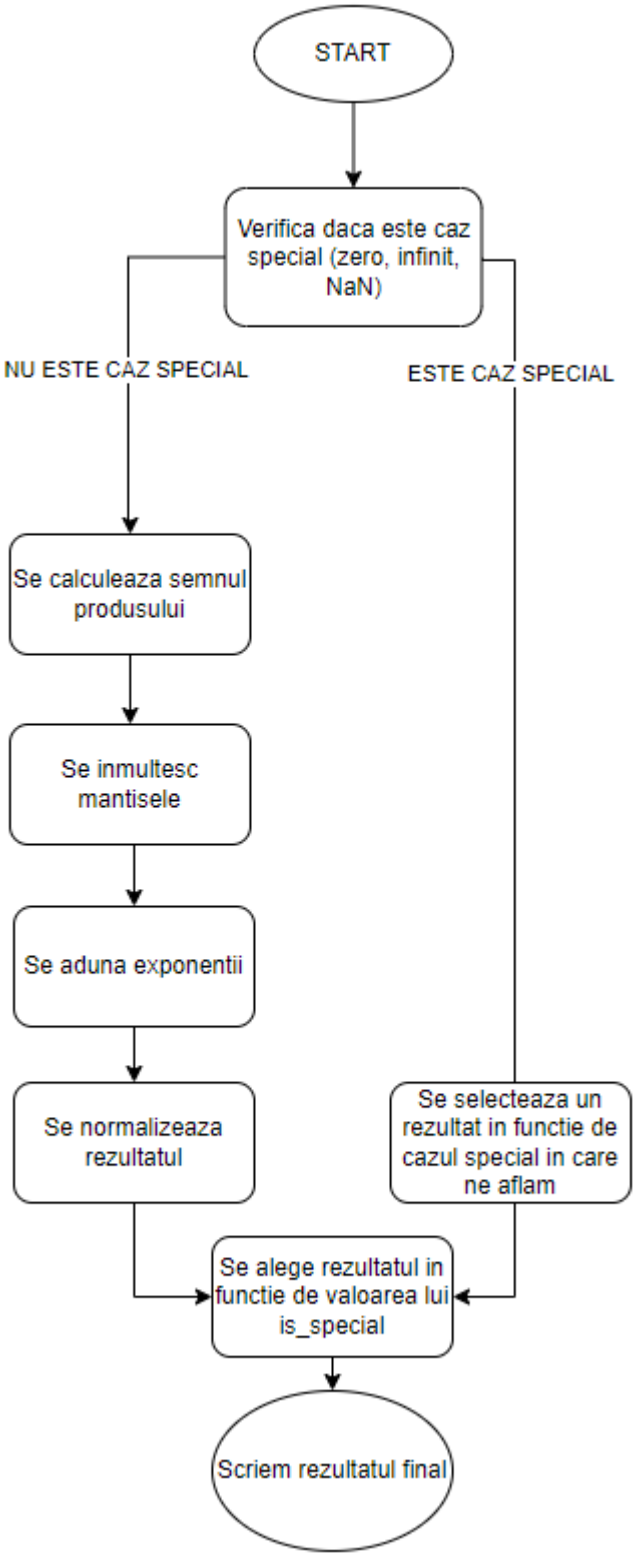
5. Gestionarea Depășirilor de Exponent

- **Scop**: În funcție de valoarea finală a exponentului, se verifică dacă acesta este în intervalul acceptabil. Dacă nu este, se raportează o eroare.
- **Tipuri de Depășire**:
    - **Depășire Superioară (Overflow)**: Dacă exponentul rezultat este prea mare, acest lucru indică o valoare prea mare pentru a fi reprezentată și este raportată ca eroare de overflow.
    - **Depășire Inferioară (Underflow)**: Dacă exponentul rezultat este prea mic, acest lucru indică o valoare prea mică pentru a fi reprezentată și este raportată ca eroare de underflow.
- **Finalizare**: În cazul în care apare o depășire de exponent (fie superioară, fie inferioară), procesul se oprește și se semnalizează eroarea respectivă.

6. Rotunjirea Rezultatului

- **Scop**: În aritmetica în virgulă flotantă, rotunjirea poate fi necesară pentru a menține precizia, deoarece există o limitare a numărului de cifre semnificative care pot fi reprezentate.
- **Proces**:
    - După ce rezultatul este normalizat, se aplică o regulă de rotunjire pentru a asigura că mantisa este reprezentată corect în limitele disponibile.
    - Dacă este necesară o rotunjire (exemplu: rotunjire la cel mai apropiat număr sau trunchiere), aceasta se aplică conform regulilor implementate.

Figura 1 – Procesul de inmultire in virgula flotanta



1. **Verificarea Inițială a Operanzilor**:

Procesul începe cu verificarea dacă unul dintre operanzi, X sau Y, este egal cu zero, infinit sau NaN. Aceasta se face pentru a optimiza operația. In functie de valoarea din X si din Y, se va scrie un rezultat cu o valoare prestabilita, iar flag-ul “is_special” va fi activ (valoarea 1).

2. **Stabilirea Semnului Rezultatului**:

Semnul rezultatului este stabilit în funcție de semnele operanzilor inițiali X și Y. Dacă cei doi operanzi au semne diferite, rezultatul va fi negativ; în caz contrar, rezultatul va fi pozitiv.

3. **Adunarea Exponenților**:

Această etapă constă în adunarea exponenților numărului X și Y și se realizează în format binar. Conform standardului IEEE 754, fiecare exponent are o valoare de **deplasament (BIAS)**, specifică pentru a reprezenta atât numere pozitive, cât și negative. Pentru formatul pe 32 de biți (simplă precizie), valoarea de deplasament este de obicei 127. Pentru a corecta valoarea obținută în urma adunării exponenților, deplasamentul este scăzut.

4. **Înmulțirea Mantiselor**:

Mantisele reprezintă partea fracționară a numărului și sunt înmulțite pentru a obține o mantisă intermediară.

Această operație de înmulțire poate produce o mantisă care are mai mulți biți decât cei care pot fi reprezentați în formatul inițial, astfel că poate necesita ajustări suplimentare. Se realizeaza cu algoritmul Booth

5. **Normalizarea Rezultatului**:

Rezultatul obținut după înmulțirea mantiselor trebuie să fie **normalizat**. Normalizarea reprezintă ajustarea mantisei astfel încât aceasta să fie cuprinsă în intervalul acceptabil pentru reprezentarea corectă, cu primul bit diferit de zero.

Normalizarea poate implica deplasarea mantisei la dreapta sau la stânga, iar pentru a păstra valoarea numerică corectă, se ajustează corespunzător și valoarea exponentului.

6. **Finalizarea**:

După toate etapele menționate, rezultatul final este generat, care include **semnul**, **exponentul** ajustat și **mantisa** normalizată. Acesta reprezintă rezultatul final al operației de înmulțire a celor două numere în virgulă flotantă, conform standardului IEEE 754.

Design si Implementare

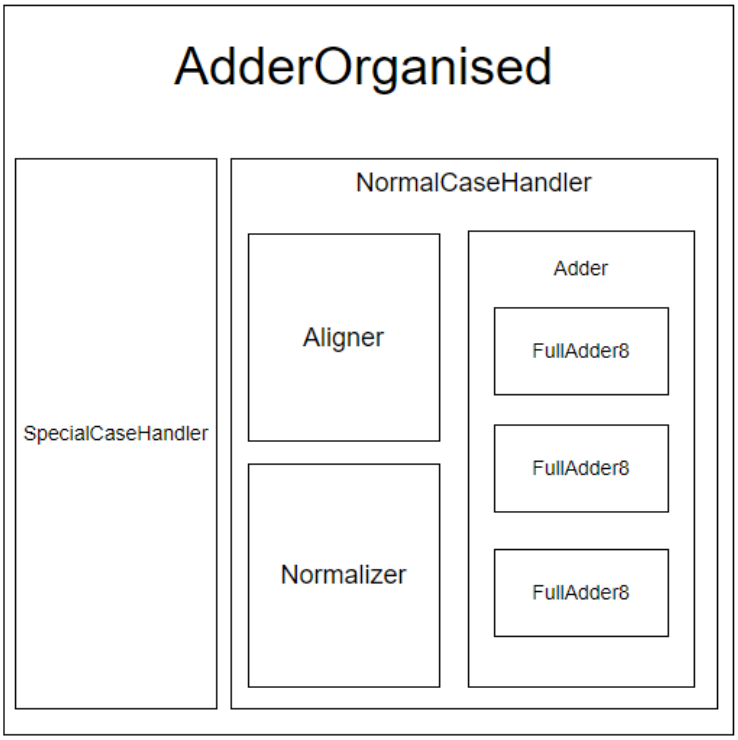


Figura 2 - Organizarea componentelor în unitatea de adunare

Unitatea de adunare se imparte in doua mari componente: **SpecialCaseHandler** si **NormalCaseHandler.**

SpecialCaseHandler:
Aceasta componenta se ocupa de cazurile “speciale” ale adunarii. Aici intra adunarea cu zero, infinit sau NaN. Se verifica fiecare caz prin mai multe structuri “if”, iar rezultatul este unul prestabilit in functie de cazul in care ne aflam. De asemenea, avem un flag “SpecialCase” care ia valoarea 1 atunci cand ne aflam in unul din cazurile speciale.

NormalCaseHandler:
Aceasta componenta trateaza cazurile in care ambele numere au valori obisnuite. Ea se imparte la randul ei in 3 alte componente: **Aligner**, **Adder** si **Normalizer**.

Aligner:

Este utilizat pentru a ajusta mantisele a două numere astfel încât acestea să fie pregătite pentru operații de adunare. Dacă exponenții celor două numere sunt diferiți, acest modul ajustează mantisa numărului cu exponentul mai mic, prin deplasarea ei la dreapta (shift right). Această operație garantează că ambele numere au același exponent și pot fi adunate corect ulterior.

Adder:

Compus din 3 FullAdder-uri pe 8 biti, aceasta componenta aduna mantisele care au rezultat in urma procesului din Aligner. Returneaza mantisa rezultata + carry-ul.

Normalizer:

Ajustează mantisa și exponentul unui număr cu virgulă flotantă. Dacă mantisa depășește intervalul normalizat, aceasta este shiftată la dreapta, iar exponentul crește. Dacă este sub-normalizată, mantisa se shift-ează la stânga, iar exponentul scade. Modulul returnează mantisa normalizată, exponentul ajustat și semnalul de overflow.

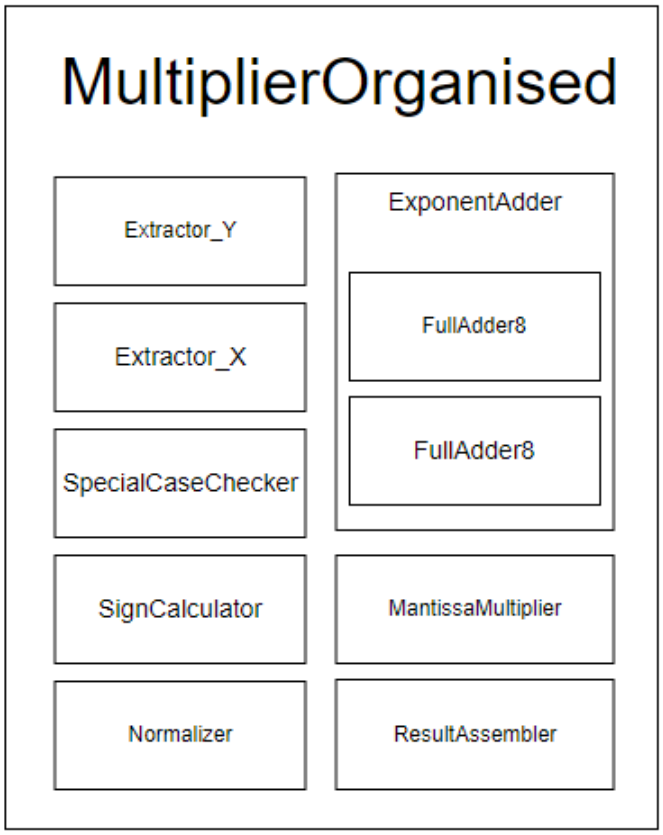


Figura 3 - Organizarea componentelor în unitatea de inmultire

Unitatea de inmultire se imparte in mai multe componente: **Extractor_X/Y, SpecialCaseChecker, SignCalculator, Normalizer, ExponentAdder, MantissaMultiplier si ResultAssembler.**

Extractor:

Aceasta componenta doar primeste ca input numarul de 32 de biti si il “imparte” in semn, exponent si mantisa. Este folosit mai mult pentru a lucra mai elegant, cu variabile cu nume mai sugestiv cum ar fi exponent_x sau sign_x.

SpecialCaseChecker:

La fel ca la adunare, am decis sa realizez o componenta care se ocupa doar de cazurile “speciale”. Se realizeaza mai multe verificari pentru a vedea daca inmultirea este cu zero, infinit sau NaN, dupa care se stabileste un rezultat predefinit si se activeaza flag ul “is_special”.

SignCalculator:

Se preiau semnele lui X si Y, iar ca output este rezultatul sign_x XOR sign_y, pentru a stabili care este semnul rezultatului.

ExponentAdder:

Modulul calculează suma celor doi exponenți, ajustând rezultatul prin eliminarea bias-ului. Inițial, exponenții de intrare sunt adunați folosind FullAdder-uri pe 8 biti. Bias-ul este inversat (operație NOT) și adăugat la suma obținută. Rezultatul final este exponentul ajustat, furnizat pe ieșirea sum_exponent.

MantissaMultiplier:

Modulul implementează un multiplicator pentru mantise pe 24 de biți folosind algoritmul **Booth**. Algoritmul efectuează o multiplicare serială bazată pe operații succesive de adunare sau scădere și o shiftare aritmetică la dreapta.

Normalizer:

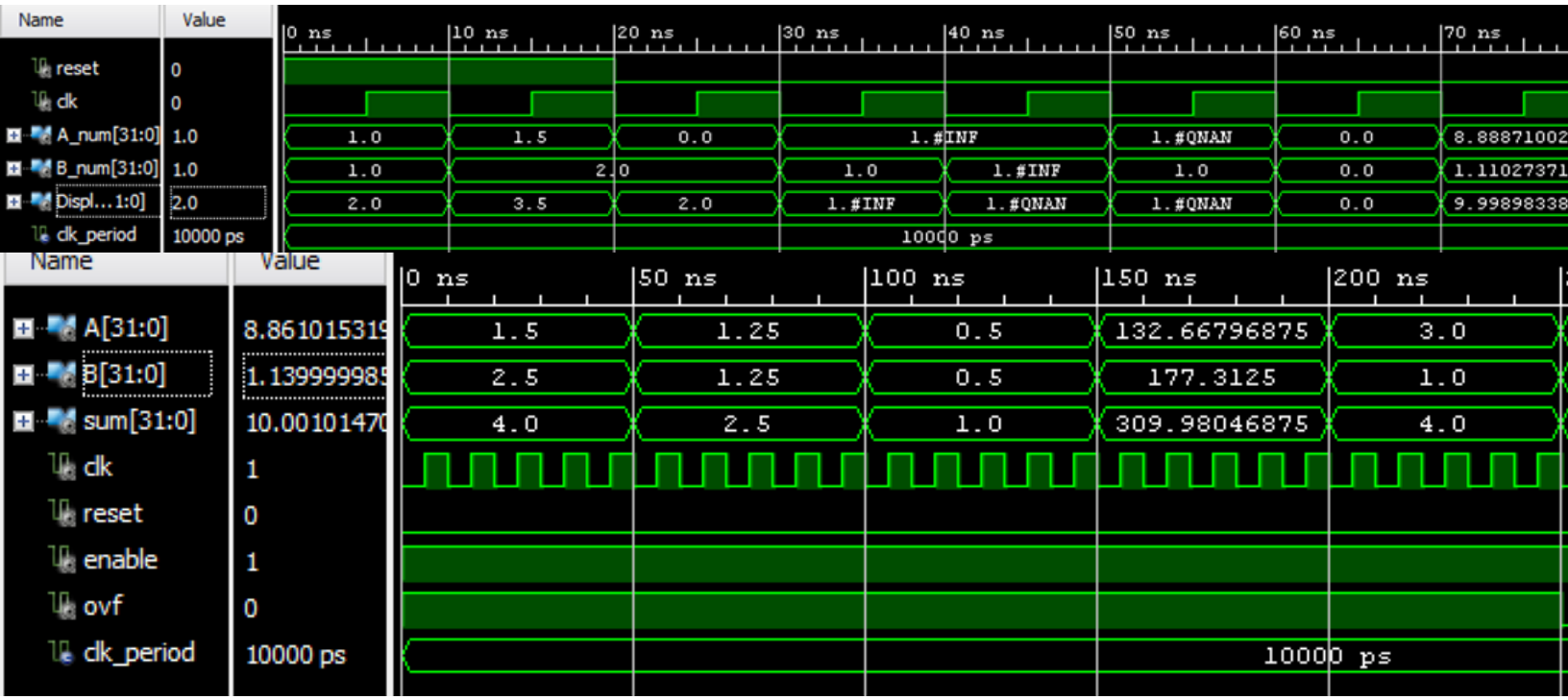
Modulul ajustează mantisa și exponentul unui produs pe 48 de biți. Dacă bitul 47 este 1, mantisa se shiftează la dreapta și exponentul crește. Altfel, mantisa și exponentul rămân nemodificate.

Assembler:

Modulul construiește rezultatul final al unui număr în virgulă flotantă pe 32 de biți. Acesta concatenează semnul (sign), exponentul (exponent) și mantisa (mantissa) într-un singur vector de 32 de biți, furnizat pe ieșirea result.

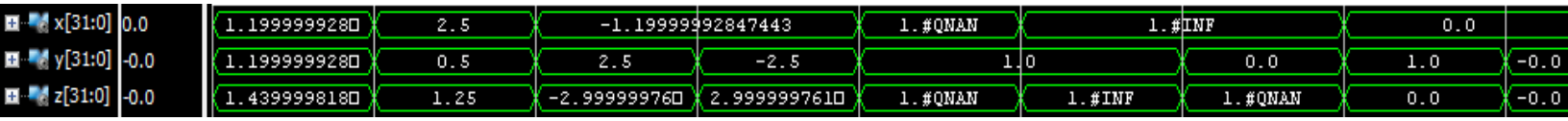
Testare

Întâi se vor prezenta rezultatele de la testele componentei de adunare



A=1	B=1	A+B=2	Rezultat primit: 2
A=1.5	B=2	A+B=3.5	Rezultat primit: 3.5
A=8.8887	B=1.1102	A+B=9.9989	Rezultat primit 9.9989
A=1.5	B=2.5	A+B=4	Rezultat primit: 4
A=1.25	B=1.25	A+B=2.5	Rezultat primit: 2.5
A=0.5	B=0.5	A+B=1	Rezultat primit: 1
A=132.66786875	B=177.3125	A+B=309.98046875	Rezultat primit: 309.98046875
A=INF	B=0	A+B=INF	Rezultat primit: INF
A=INF	B=INF	A+B=NaN	Rezultat primit: NaN
A=NaN	B=1	A+B=NaN	Rezultat primit: NaN
A=0	B=0	A+B=0	Rezultat primit: 0

Acum se vor prezenta rezultatele de la testele componentei de înmulțire



A=1.12	B=1.2	A*B=1.44	Rezultat primit: 1.44
A=2.5	B=0.5	A*B=1.25	Rezultat primit: 1.25
A=-1.12	B=2.5	A*B=-3	Rezultat primit: -3
A=-1.12	B=-2.5	A*B=3	Rezultat primit: 3
A=NaN	B=1	A*B=NaN	Rezultat primit: NaN
A=INF	B=1	A*B= INF	Rezultat primit: INF
A=INF	B=0	A*B= NaN	Rezultat primit: NaN
A=0	B=1	A*B=0	Rezultat primit: 0
A=0	B=-0	A*B=-0	Rezultat primit: -0

Concluzii

În cadrul acestui proiect, am realizat implementarea operațiilor de adunare și scădere a numerelor în virgulă mobilă. Pe parcursul acestuia, am adunat informații relevante despre Standardul IEEE 754, precum și despre modul în care se efectuează aceste operații. Simultan, am îmbunătățit și dezvoltat abilitățile mele în scrierea de cod VHDL.

Dezvoltări ulterioare

Extinderea funcționalităților ALU:

- Adăugarea altor operații aritmetice, precum scăderea, împărțirea sau operații trigonometrice (sin, cos, etc.).
- Suport pentru numere în virgulă flotantă de dublă precizie (64 de biți) conform standardului IEEE 754.

Bibliografie

IEEE Standard for Floating-Point Arithmetic, IEEE 22 July 2019

Adunare VM, Dr. Baruch Zoltan Francisc, 2000

Booth, Wikipedia

„IEEE Standard 754 for Binary Floating-Point Arithmetic”
Lectures Notes, Prof. W. Kahan, Elect. Eng. & Computer Science, University of California, 1997

„Computer Organisation and Design”
Ediția a cincea, David A. Patterson și John L. Hennessy