ALU cu numere întregi, cu virgulă flotantă, pozitive și negative

Nume Student: Mureșan Raul-Călin

Grupa: 30239

Îndrumător: Popa Cristian Mihai

Universitatea Tehnică din Cluj-Napoca

Secția Calculatoare

Data: 23.12.2024

Contents

Rezumat

Acest proiect prezintă implementarea unui unități aritmetico-logice (ALU) capabile să opereze cu numere întregi și cu virgulă flotantă, atât pozitive, cât și negative. Principala problemă abordată a fost crearea unei arhitecturi modulare și eficiente pentru realizarea operațiilor aritmetice de bază: adunare, scădere, înmulțire și împărțire. Soluția propusă utilizează FSM-uri (Finite State Machines) pentru gestionarea secvențială a operațiilor și module specializate (add, sub, mul, div) pentru fiecare funcționalitate. Se utilizează și module auxiliare, pentru cele 4 operații (addBinary, subBinary, mulBinary, divBinary).

Implementarea a fost realizată utilizând limbajul VHDL pe platforma hardware Zybo 7000, iar interacțiunea cu utilizatorul este facilitată de butoane pentru selecția operației și LED-uri pentru indicarea stărilor de overflow și rezultat zero. Testarea și verificarea funcționalității au demonstrat performanța și acuratețea sistemului proiectat. Concluziile obținute evidențiază eficiența soluției și posibilitățile de extindere ulterioară a acesteia.

1 Introducere

Unitățile aritmetico-logice (ALU) reprezintă componente esențiale ale calculatoarelor moderne, fiind responsabile pentru execuția operațiilor matematice și logice de bază. Proiectarea unui ALU eficiente și versatile constituie un obiectiv major în domeniul ingineriei calculatoarelor, având aplicații variate în procesarea semnalelor, inteligență artificială și sisteme integrate.

Tema acestui proiect constă în implementarea unui ALU capabile să opereze cu numere întregi și cu virgulă flotantă, atât pozitive, cât și negative, utilizând arhitecturi modulare. Obiectivul principal a fost proiectarea unei soluții care să asigure corectitudinea rezultatelor și optimizarea utilizării resurselor hardware. Printre cerințele specifice se numără realizarea operațiilor de adunare, scădere, înmulțire și împărțire, gestionate prin FSM-uri pentru control secvențial.

Implementarea utilizează limbajul VHDL și este destinată platformei hardware Zybo 7000. Soluția propusă integrează funcționalități interactive, cum ar fi selecția operației prin butoane și afișarea rezultatelor sau a erorilor (overflow, rezultat zero) prin LED-uri. Proiectul se aliniază tendințelor actuale din tehnologie, care pun accent pe modularitate și eficiență în proiectarea sistemelor digitale.

Această documentație descrie în detaliu fundamentarea teoretică, proiectarea și implementarea sistemului, precum și rezultatele experimentale obținute, cu scopul de a demonstra validitatea și performanța soluției propuse.

2 Fundamentare Teoretică

Fundamentele teoretice ale acestui proiect sunt centrate pe proiectarea și implementarea unităților aritmetico-logice (ALU) utilizând arhitecturi digitale. Aceste arhitecturi includ:

- **FSM-uri (Finite State Machines)**: FSM-urile sunt utilizate pentru gestionarea controlului secvențial al operațiilor. Acestea permit coordonarea tranzițiilor între stările de așteptare, execuție și finalizare pentru fiecare operație.
- **Reprezentarea numerelor în virgulă flotantă pe 32 de biți**: Sistemul utilizează standardul IEEE 754 pentru reprezentarea numerelor în virgulă flotantă, care oferă precizie și consistență în calcule. Reprezentarea este realizată astfel:
  - **1 bit pentru semn**: indică dacă numărul este pozitiv (0) sau negativ (1);
  - **8 biți pentru exponent**: reprezintă exponentul în formă biasată, cu un bias de 127 (exponent real = exponent stocat - 127);
  - **23 biți pentru mantisă**: reprezintă partea fracționară a numărului, presupunând un "1" implicit înaintea punctului zecimal (forma normalizată).

  Structura acestui format este prezentată în figura 1, care detaliază pozițiile bitului de semn, ale exponentului și ale mantisei.

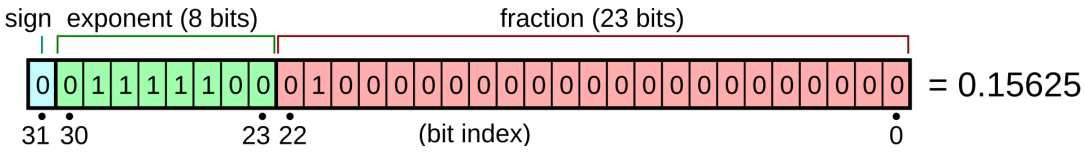

Figure 1: Structura reprezentării numerelor în virgulă flotantă pe 32 de biți (standard IEEE 754).

  Această reprezentare permite un echilibru între domeniul dinamic al valorilor (prin exponent) și precizie (prin mantisă).

- **Modularitatea arhitecturii**: Fiecare operație (adunare, scădere, înmulțire, împărțire) este implementată într-un modul separat, ceea ce facilitează testarea și reutilizarea.
- **Limbajul VHDL**: VHDL oferă un cadru puternic pentru descrierea și simularea sistemelor digitale, permițând testarea și verificarea detaliată a proiectului înainte de implementare hardware.

2.1 Algoritmii aritmeticii în virgulă flotantă

Pentru implementarea operațiilor aritmetice (adunare, scădere, înmulțire, împărțire) cu numere în virgulă flotantă, s-au utilizat următoarele etape:

2.1.1 Adunare/Subtracție (ADD/SUB)

Pentru operațiile de adunare și scădere între două numere în virgulă flotantă, se urmează următorii pași:

1. Se compară exponenții numerelor. Se alege numărul cu exponentul mai mic, iar mantisa acestuia este deplasată spre dreapta cu un număr de pași egal cu diferența exponenților.
2. Exponentul rezultatului este setat la valoarea exponentului mai mare.
3. Se efectuează operația de adunare sau scădere asupra mantiselor ajustate, determinând semnul rezultatului.
4. Rezultatul este normalizat dacă este necesar (deplasarea mantisei și ajustarea exponentului).
5. Se verifică eventualele situații de overflow sau underflow.

2.1.2 Înmulțire (MULT)

Pentru operația de înmulțire, se urmează următorii pași:

1. Mantisele celor două numere sunt înmulțite, iar semnul rezultatului este determinat prin XOR-ul semnelor celor două numere.
2. Exponenții sunt adunați, iar rezultatul este ajustat cu bias-ul sistemului (127 pentru reprezentarea pe 32 de biți).
3. Rezultatul este normalizat dacă este necesar.
4. Se verifică eventualele situații de overflow sau underflow.

2.1.3 Împărțire (DIV)

Pentru operația de împărțire, se urmează următorii pași:

1. Mantisa primului număr este împărțită la mantisa celui de-al doilea număr, iar semnul rezultatului este determinat prin XOR-ul semnelor.
2. Exponenții sunt scăzuți, iar rezultatul este ajustat cu bias-ul sistemului.
3. Rezultatul este normalizat dacă este necesar.
4. Se verifică eventualele situații de overflow sau underflow.

2.1.4 Ecuațiile operațiilor

Ecuațiile care descriu matematic fiecare operație sunt prezentate mai jos:

- **Adunare**: $X + Y = (X_M \cdot 2^{X_E - Y_E} + Y_M) \cdot 2^{Y_E}$
- **Scădere**: $X - Y = (X_M \cdot 2^{X_E - Y_E} - Y_M) \cdot 2^{Y_E}$
- **Înmulțire**: $X \cdot Y = (X_M \cdot Y_M) \cdot 2^{X_E + Y_E}$
- **Împărțire**: $X/Y = (X_M / Y_M) \cdot 2^{X_E - Y_E}$

Unde X_M, Y_M sunt mantisele, iar X_E, Y_E sunt exponenții celor două numere.

Proiectarea ALU este bazată pe lucrările de referință din domeniul arhitecturii calculatoarelor și pe utilizarea platformei hardware Zybo 7000. Avantajele utilizării acestei platforme includ:

- Resurse hardware dedicate (procesorul Zynq);
- Suport extins pentru interfațarea cu utilizatorul prin butoane și LED-uri;
- Integrarea ușoară a modulelor VHDL cu software-ul dezvoltat în Vitis pentru controlul sistemului.

Aceste fundamente asigură că soluția propusă este robustă, scalabilă și ușor de extins pentru cerințe suplimentare în viitor.

3 Proiectare și Implementare

Arhitectura sistemului propus este organizată modular, fiecare componentă fiind responsabilă de o funcționalitate specifică. Această abordare asigură flexibilitate, testabilitate și scalabilitate. Arhitectura generală include:

3.1 Arhitectura Generală

Sistemul este structurat în următoarele module principale:

- **Modulul principal ALU**: reprezintă entitatea top și instanțiază modulele operaționale pentru adunare, scădere, înmulțire și împărțire (**add**, **sub**, **mul**, **div**).
- **Module operaționale**: fiecare dintre acestea instanțiază module dedicate pentru realizarea operațiilor specifice:
  - **add**: instanțiază modulul **addBinary** pentru realizarea operației de adunare;
  - **sub**: instanțiază modulul **subBinary** pentru realizarea operației de scădere;
  - **mul**: instanțiază modulul **mulBinary** pentru realizarea operației de înmulțire;
  - **div**: instanțiază modulul **divBinary** pentru realizarea operației de împărțire.
- **Modulul de intrare/ieșire**: gestionează interacțiunea cu utilizatorul prin intermediul butoanelor și LED-urilor de stare (overflow, zero).

3.2 Funcționalitatea Modulului ALU

Modulul ALU central instanțiază și controlează modulele operaționale în funcție de semnalele primite. FSM-ul implementează logica de trecere între stările de inițializare, procesare și finalizare pentru fiecare operație. Fiecare tranziție este sincronizată cu semnalul de ceas (clock).

3.3 Interfața Utilizatorului

Interfața utilizatorului este realizată prin:

- **Input-uri prin Vitis**: valorile pentru intrările `A` și `B` sunt introduse de la tastatură direct în mediul Vitis.

- **Afișarea rezultatului în Vitis**: rezultatul operației selectate este afișat tot în cadrul interfeței terminalului Vitis, permițând utilizatorului să vizualizeze rezultatul și eventualele mesaje de eroare.
- **Butoane hardware**: utilizate pentru selectarea operației dorite (adunare, scădere, înmulțire, împărțire).
- **LED-uri hardware**: indică stările de eroare (overflow) sau rezultate speciale (zero).

Această interfață combină utilizarea mediului Vitis pentru introducerea datelor și afișarea rezultatelor, cu butoanele și LED-urile hardware pentru o testare și utilizare practică extinsă.

3.4 Implementarea Hardware și Software

- **Hardware**: sistemul este implementat pe placa Zybo 7000, utilizând blocuri logice programabile pentru execuția operațiilor.
- **Software**: mediul Vitis este utilizat pentru dezvoltarea aplicației de control care inițializează intrările și afișează rezultatele pe un terminal conectat.

3.5 Diagrame de Stări pentru Operații Aritmetice

Pentru a gestiona execuția operațiilor aritmetice, fiecare modul de operație este controlat printr-o Mașină de State Finite (FSM). Diagramele următoare ilustrează fluxul de execuție pentru operațiile de adunare/scădere, înmulțire și împărțire.

3.5.1 Diagrama de Stări pentru Adunare și Scădere

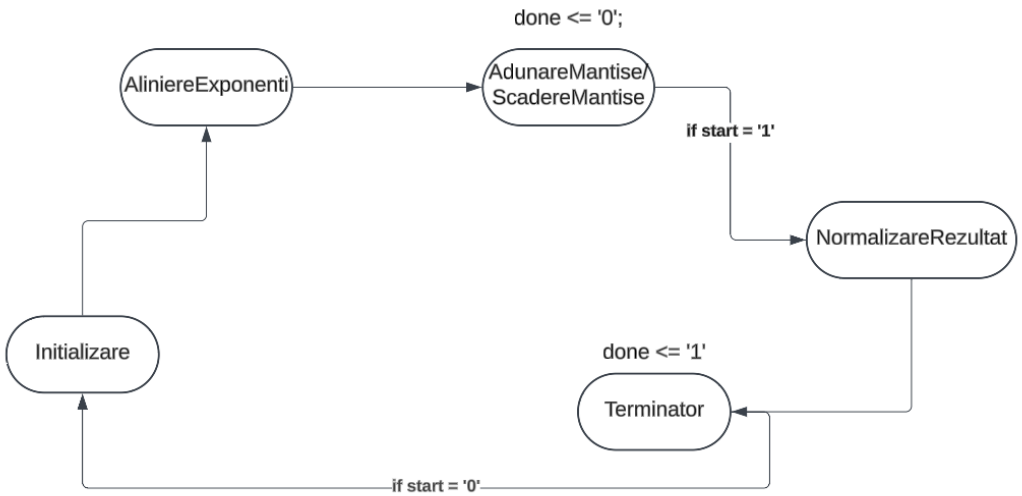

Figure 2: Diagrama de stări pentru operațiile de adunare și scădere.

FSM-ul pentru adunare și scădere include următoarele stări:

- **Initializare**: Se încarcă datele de intrare și se resetează semnalele.
- **AliniereExponenti**: Exponenții sunt ajustați pentru a efectua corect operația.
- **AdunareMantise/ScadereMantise**: Se realizează efectiv operația de adunare sau scădere a mantiselor.
- **NormalizareRezultat**: Rezultatul este ajustat pentru a respecta standardul IEEE 754.
- **Terminare**: Se finalizează operația și se setează semnalul `done`.

3.5.2 Diagrama de Stări pentru Înmulțire

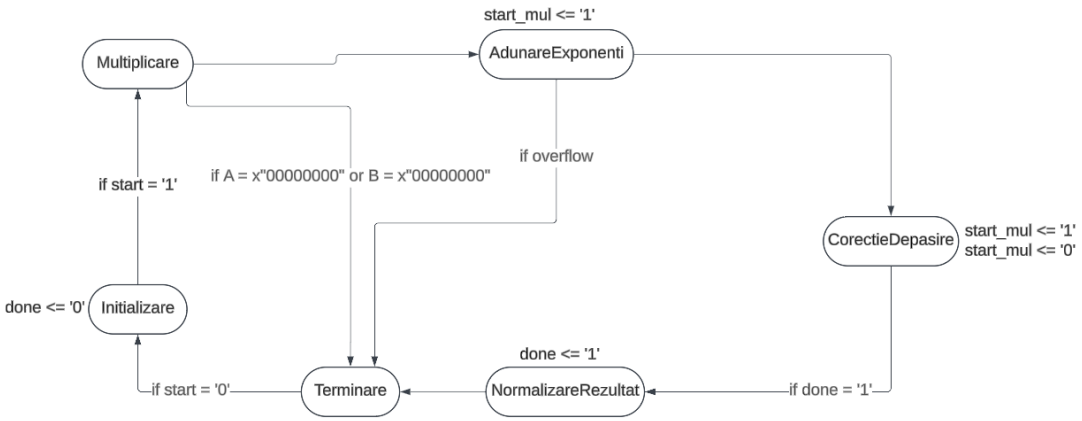

Figure 3: Diagrama de stări pentru operația de înmulțire.

FSM-ul pentru înmulțire include următoarele stări:

- **Initializare**: Inițializarea semnalelor și încărcarea datelor de intrare.
- **Multiplicare**: Se înmulțesc mantisele numerelor de intrare.
- **AdunareExponenti**: Se adună exponenții, ajustați cu bias-ul IEEE 754.
- **NormalizareRezultat**: Se normalizează rezultatul.
- **Terminare**: Se finalizează calculul și se generează rezultatul.

3.5.3 Diagrama de Stări pentru Împărțire

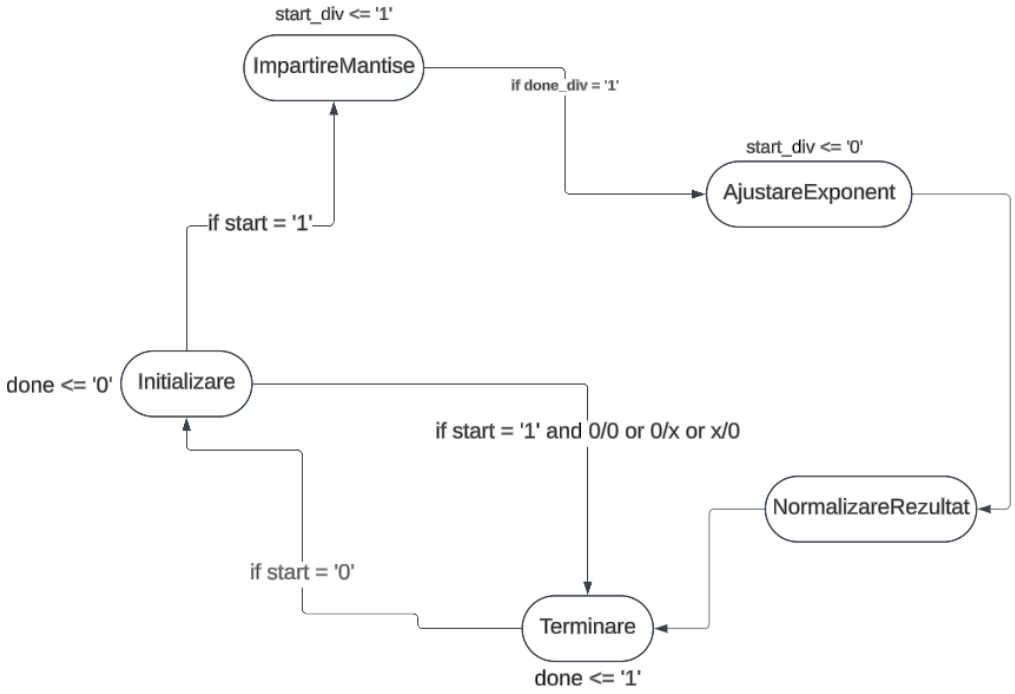

Figure 4: Diagrama de stări pentru operația de împărțire.

FSM-ul pentru împărțire include următoarele stări:

- **Initializare** - Se verifică cazurilor speciale și încărcarea datelor.
- **ImpărtireMantise** - Se efectuează împărțirea mantiselor.
- **AjustareExponent** - Exponenții sunt scăzuți.
- **NormalizareRezultat** - Rezultatul este ajustat la formatul IEEE 754.
- **Terminare** - Se finalizează calculul și se setează semnalul `done`.

3.6 Design-ul blocului

În figura de mai jos este prezentat design-ul blocului, care ilustrează conexiunile între modulele principale (ALU, butoane, LED-uri, etc.) și interfața cu sistemul hardware. Acesta evidențiază utilizarea procesorului ZYNQ pentru coordonarea semnalelor și a componentelor periferice.

Această schemă este utilizată pentru a asigura conectivitatea corectă între modulele hardware și interfața utilizatorului.

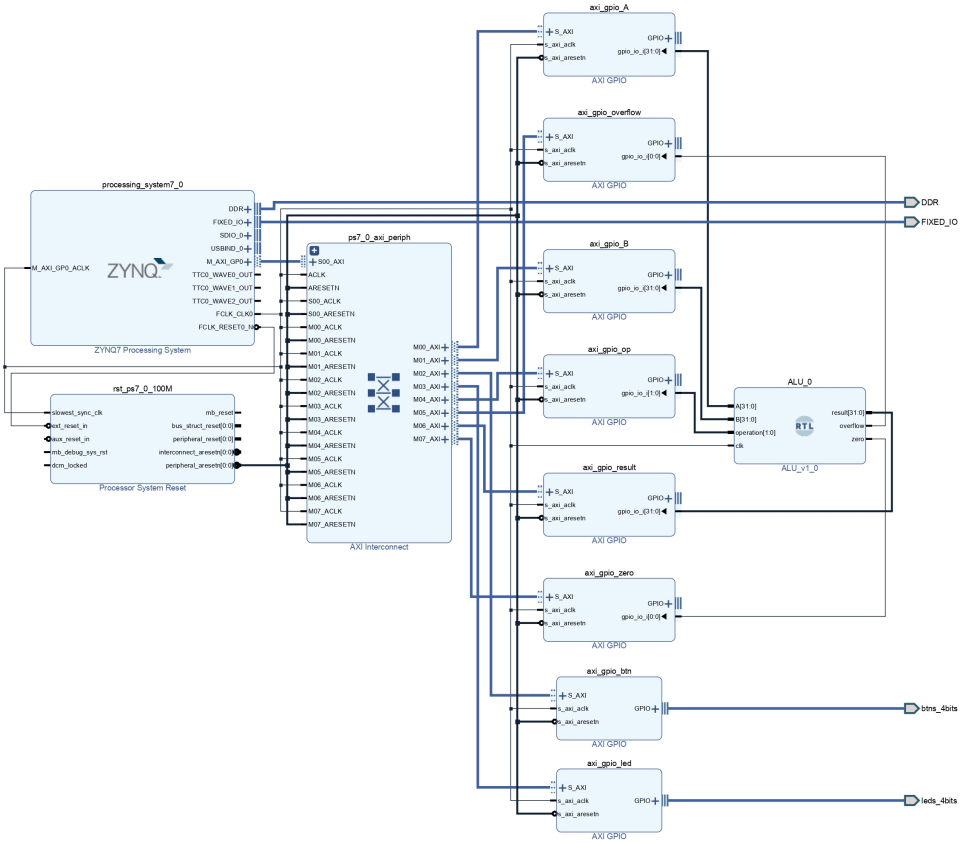

Figure 5: Design-ul blocului ALU implementat pe placa Zybo 7000.

4 Rezultate Experimentale

Testarea sistemului a fost realizată utilizând o combinație de simulări în mediul VHDL și rulări practice pe placa Zybo 7000.

4.1 Procedura de Testare

Procedura de testare a inclus următorii pași:

- Crearea unor seturi de date de intrare reprezentând numere întregi și în virgulă flotantă, pozitive și negative.
- Validarea fiecărei operații (adunare, scădere, înmulțire, împărțire) prin simulare în Vivado, pentru a verifica corectitudinea rezultatelor.
- Rularea sistemului pe hardware, utilizând butoanele pentru selecția operației și observarea rezultatelor prin LED-uri și terminalul software.

4.2 Rezultatele Obținute

Rezultatele testării au fost următoarele:

- Operațiile aritmetice (adunare, scădere, înmulțire, împărțire) nu au fost executate corect, conform simulărilor și rulării pe placa Zybo-7000.
- LED-urile nu au indicat corect stările de overflow și de rezultat zero.

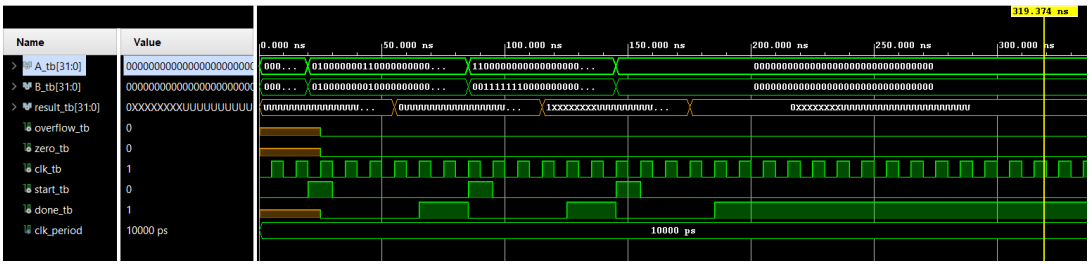

Figure 6: Rezultatul simulării în Vivado pentru operația de adunare.

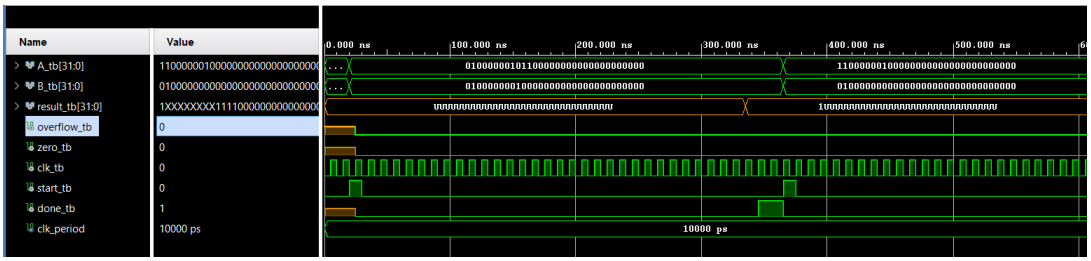

Figure 7: Rezultatul simulării în Vivado pentru operația de scădere.

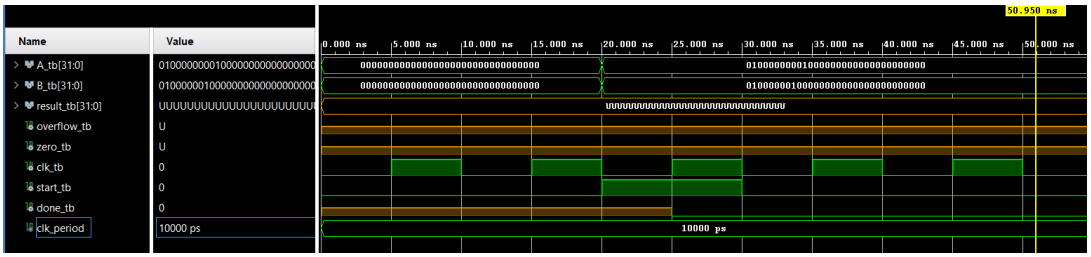

Figure 8: Rezultatul simulării în Vivado pentru operația de înmulțire.

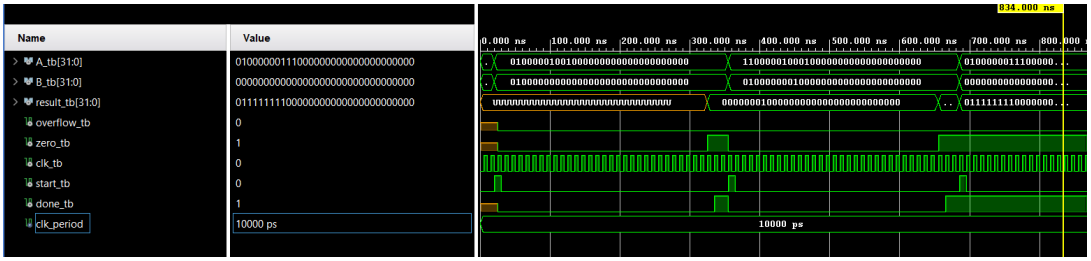

Figure 9: Rezultatul simulării în Vivado pentru operația de împărțire.

Figure 10: Rezultatul rulării pe placa Zybo-7000.

4.3 Dificultăți Întâlnite și Soluții Aplicate

- **Problema**: Sincronizarea semnalelor între module.
  Soluția: Utilizarea unui semnal de ceas (clock) comun și introducerea de registre intermediare pentru a evita întârzierile.
- **Problema**: Erori de reprezentare pentru numerele cu virgulă flotantă.
  Soluția: Ajustarea calculului mantisei conform standardului IEEE 754.

5 Concluzii

Proiectul prezentat a implementat un ALU capabil să efectueze operații cu numere întregi și cu virgulă flotantă, atât pozitive cât și negative. Arhitectura modulară și utilizarea FSM-urilor au permis o gestionare eficientă a execuției operațiilor și o separare clară a responsabilităților între module.

5.1 Avantajele soluției

- Precizia ridicată a operațiilor, inclusiv pentru numerele cu virgulă flotantă;
- Interfața intuitivă pentru utilizator, care facilitează selecția operațiilor și indicarea stărilor de eroare;
- Stabilitatea sistemului la frecvențe de ceas ridicate (100 MHz);
- Posibilitatea extinderii arhitecturii pentru a include operații suplimentare sau optimizări de performanță.

5.2 Dezavantajele soluției

- Complexitatea crescută a sincronizării semnalelor între module;
- Limitări în ceea ce privește reprezentarea numerelor extrem de mici sau mari, în conformitate cu standardul IEEE 754.

5.3 Dezvoltări viitoare ale proiectului

- Implementarea unui set extins de operații aritmetice și logice;
- Optimizarea utilizării resurselor hardware pentru a reduce consumul de energie;
- Integrarea cu alte sisteme pentru aplicații complexe, cum ar fi procesarea semnalelor sau criptografia.

În concluzie, proiectul reprezintă o bază solidă pentru explorări suplimentare în proiectarea sistemelor digitale eficiente.

6 Bibliografie

- **Single-precision floating-point format**, Wikipedia, accesat la: `https://en.wikipedia.org/wiki/Single-precision_floating-point_format`
- **VHDL to Integer**, Stack Overflow, accesat la: `https://stackoverflow.com/questions/59058126/vhdl-to-integer`
- **IEEE 754 Converter**, Numeral Systems, accesat la: `https://numeral-systems.com/ieee-754-converter/`
- **Binary to Decimal Converter**, Rapid Tables, accesat la: `https://www.rapidtables.com/convert/number/binary-to-decimal.html`
- **How Floating Point Numbers Work (IEEE 754)**, YouTube, accesat la: `https://www.youtube.com/watch?v=-bJ1_N8B4fU`
- **ALU (3) - Division Algorithms**, Humboldt-Universität zu Berlin, Sommersemester 2002, Leitung: Prof. Dr. Miroslaw Malek, accesat la: `www.informatik.hu-berlin.de/rok/ca`