UNITATE ARITMETICO-LOGICĂ ÎN VIRGULĂ FLOTANTĂ (ADUNARE, SCĂDERE)

Georgian Andreea

Universitatea Tehnică din Cluj-Napoca

CUPRINS

1. INTRODUCERE

i) CONTEXT

Acest proiect are scopul de a implementa două operații care se efectuează cu numere reprezentate în virgulă mobilă pe 32 de biți. În domeniul tehnologiei informației, virgula flotantă este unul din sistemele utilizate pentru a reprezenta numerele pe șiruri de biți. Această denumire se referă la faptul că virgula care separă partea întreagă de partea fracționară a numărului se poate muta, adică poate fi plasată oriunde in raport cu cifrele semnificative ale acestuia.

Avantajul reprezentării în virgulă mobilă față de cea în virgulă fixă este gama mai largă de valori care se pot reprezenta. În timp ce în virgulă fixă se pot reprezenta numere cu un format fix în raport cu poziția virgulei, virgula flotantă permite sacrificarea preciziei (numărului cunoscut de zecimale) pentru reprezentarea unor numere mari, și invers pentru o reprezentare mai precisă a numerelor mici.

ii) SPECIFICAȚII

Aplicația va fi simulată in Vivado și programată pe o plăcuță basys3. Va putea reprezenta intern numărul în reprezentarea IEEE cu precizie simplă (32 biți), va putea să facă conversia din complement față de doi în reprezentarea în virgulă mobilă, să realizeze operațiile pe numerele în virgulă mobilă, apoi să faca conversia în complement față de doi pentru ca rezultatul să fie într-un format mai usor de citit de utilizator.

iii) OBIECTIVE

Se va proiecta și implementa o aplicație care va stoca in virgulă mobilă numerele primite în complement față de doi. Aceasta va calcula rezultatul operației alese de către utilizator și îl va arăta rezultatul pe afișorul cu 7 segmente al plăcuței basys3.

2. STUDIU BIBLIOGRAFIC

Standardul IEEE reprezintă numerele în virgulă mobilă cu precizie simplă pe 32 de biți, care sunt distribuiți după cum urmează:

Figura 1 Reprezentarea pe 32 de biți a numerelor in virgulă mobilă

Astfel, primul bit reprezintă semnul numărului, următorii 8 biți reprezintă exponentul, iar restul de 23 de biți reprezintă mantisa.

VALOAREA ZERO: -pentru reprezentarea aceste valori, exponentul și mantisa sunt 0…000
-00000000000000000000000000000000

VALOAREA INFINIT: -pentru reprezentarea aceste valori, exponentul este 111…111, iar mantisa este 000…000
-01111111100000000000000000000000 (+ infinit)
-11111111100000000000000000000000 (- infinit)

VALOAREA NaN: -atunci când mantisa are o valoare diferită de zero și exponentul este 111…111, prin convenție, numărul în virgulă flotantă are valoarea NaN (Not a Number)

-se utilizează pentru prevenirea unor erori generate de împărțirea la zero

Deoarece adunarea și scăderea sunt identice cu excepția semnului diferit al celui de-al doilea operand, în cazul operației de scădere se schimbă semnul scăzătorului. Algoritmul pentru adunare are patru etape principale:

1. Alinierea mantiselor
2. Adunarea mantiselor
3. Normalizarea rezultatului
4. Rotunjirea rezultatului

Adunarea și scăderea în virgulă mobilă sunt operații complexe, deoarece trebuie să se realizeze egalizarea exponenților acestora. Acest fapt implică compararea exponenților și apoi alinierea mantisei numărului cu exponentul mai mic.

Alinierea se realizează prin deplasarea in mod repetat a mantisei cu o poziție la dreapta și creșterea exponentului pâna cand cei doi exponenți sunt egali. Dacă în urma acestui proces, mantisa deplasată devine 0, atunci rezultatul va fi egal cu celălalt număr. Deci, dacă cele două numere au exponenți semnificativ diferiți, numărul mai mic va fi ignorat.

Apoi, se adună mantisele, luând în considerare semnele lor. Rezultatul poate fi 0, deoarece semnele pot fi diferite. Mantisa rezultatului poate avea, de asemenea, o depășire de un bit. În acest caz, mantisa rezultatului este deplasată cu o poziție la dreapta și exponentul se incrementează. În urma deplasării, poate apărea o depășire a exponentului. În acest caz, depășirea va fi raportată și operația se va opri.

3. ANALIZA

Organigrama aplicației este reprezentată în Figura 2:

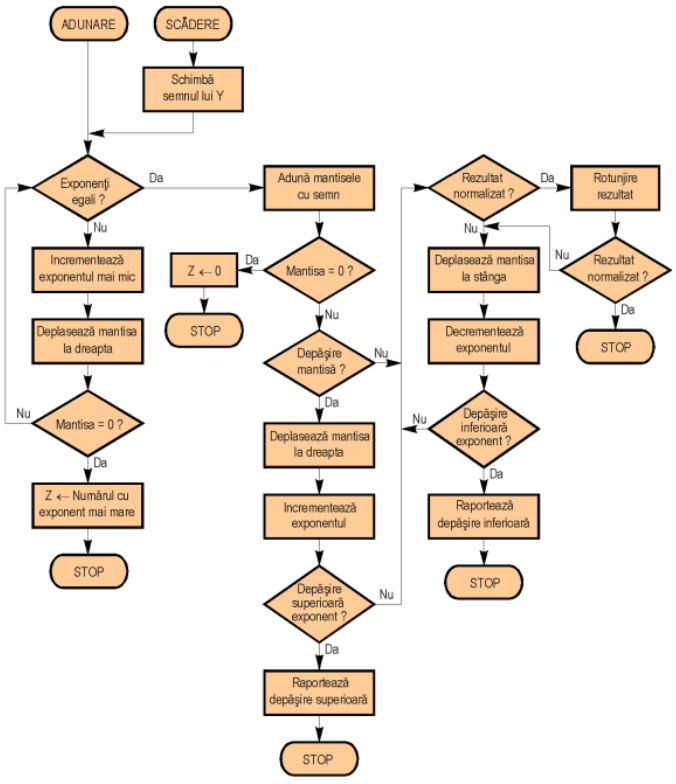


Figura 2 Organigrama [1]

După cum se vede în organigramă, pașii pentru suma numerelor în virgulă mobile sunt:

1. În cazul în care operația cerută este scăderea, atunci se inversează bitul de semn al celui de-al doilea operand

2. Se compară exponenții

3. Se deplasează la dreapta mantisa numărului mai mic și se incrementează exponentul acestuia până când exponentii celor două numere ajung egali

4. Dacă, în urma deplasării mantisei, aceasta devine 0, atunci algoritmul se va opri, iar rezultatul va fi egal cu numărul cu exponentul mai mare

4. Se adună mantisele

5. Deoarece operanzii pot avea semne diferite, mantisa poate devein 0 după adunare. În cazul în care este 0, rezultatul devine automat 0.

6. În cazul în care mantisa nu este 0, se verifică dacă a apărut o depășire în urma adunării.

7. Dacă există depășire, se deplasează mantisa cu o poziție la dreapta și se incrementează exponentul sumei

8. Dacă există și în cazul incrementării exponentului o depășire superioară, aceasta se raportează și algoritmul se oprește

9. În cazul în care nu a apărut o depășire în urma adunării mantiselor sau in urma incrementării exponentului, atunci se verifică daca rezultatul este normalizat

10. Dacă da, atunci algoritmul se oprește.

11. Dacă nu, atunci se deplasează mantisa la stânga și se decrementează exponentul până cand primul bit devine egal cu 1

12. În cazul în care, în urma decrementării, apare o depășire inferioară la exponent, atunci aceasta se raportează și algoritmul se va opri

4. DESIGN

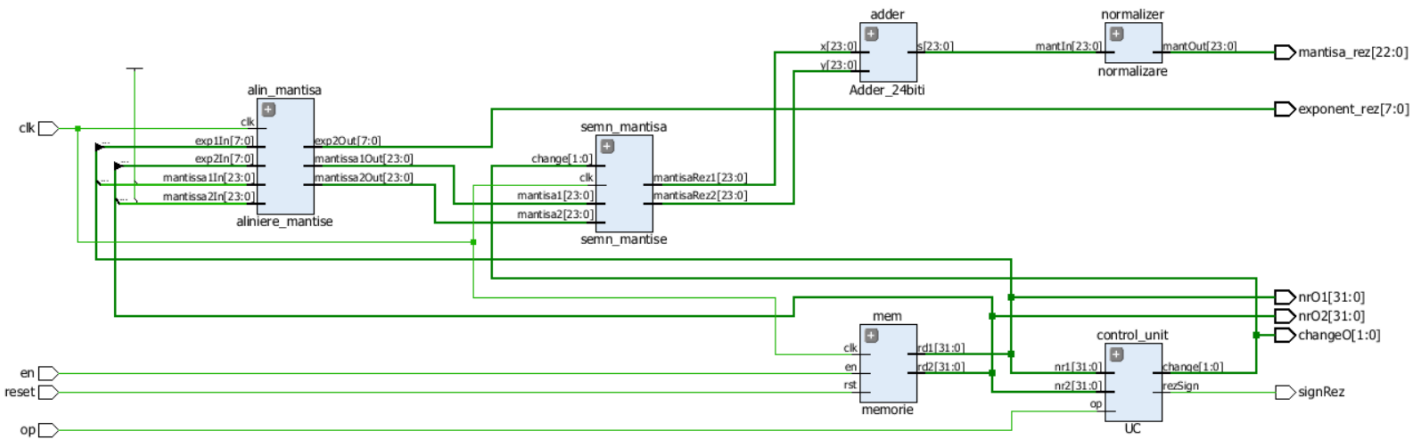


Figura 3 Schema aplicației

Memoria

În memorie sunt stocate 7 numere, între care se va face operația de adunare sau scădere, după caz. Aceasta trimite celorlalte componente numerele de pe 2 poziții consecutive, iar cand semnalul de enable este 1, atunci adresele curente sunt incrementate.

Alinierea mantiselor

Alinierea mantiselor este realizată cu ajutorul unei componente sincrone. Această componentă compară exponenții celor două numere, incrementează exponentul mai mic până ajunge egal cu celălalt exponent, iar cu ajutorul unui Barrel shifter mută mantisa care trebuie aliniată la dreapta cu atâtea poziții cât este diferența exponenților inaintea incrementarilor.

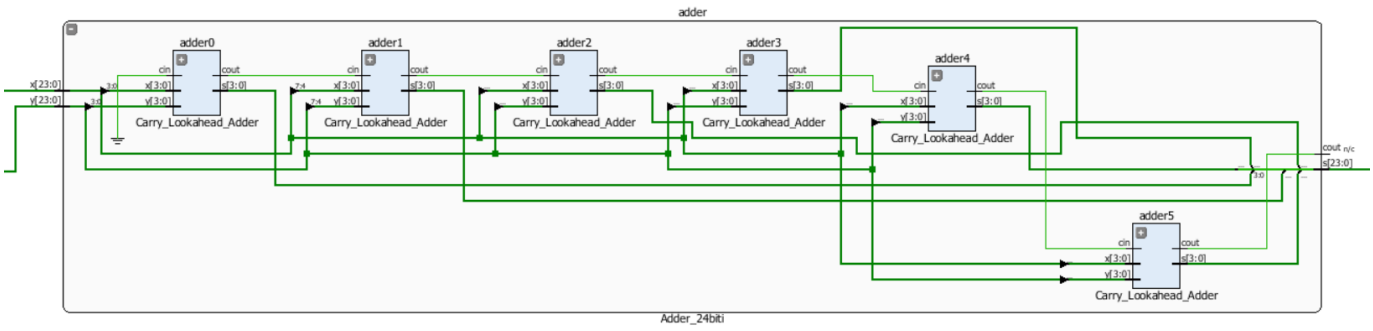
Un Barrel shifter este un circuit care poate deplasa la stânga sau la dreapta un număr cu un număr specificat de biți într-un singur ciclu de ceas. Acesta este util deoarece este mai rapid decât un circuit de shiftare obișnuit.

Semn mantisă

Această componentă transformă una dintre mantise în complement față de doi, dacă este cazul. Această operație este necesară pentru numerele negative sau pentru operația de scădere, care este văzută ca o adunare cu semnul schimbat al celui de-al doilea număr.

Adunarea mantiselor

După ce se face transformarea în complement față de doi dacă este cazul, se adună cele două mantise. Această componentă realizează adunarea a două numere de 24 de biți, prin concatenarea a 6 sumatoare cu anticiparea transportului, fiecare pe 4 biți.



Normalizarea

Această componentă preia rezultatul adunării celor două mantise și numără, începând de la pozitia 22, toți biții de 0 existenți până la întâlnirea primului bit de 1. Apoi , se realizează deplasarea la stânga a mantisei cu un număr de biți egal cu rezultatul numărării anterioare.

Unitatea de control

Unitatea de control stabilește semnul rezultatului, în funcție de semnele celor două numere și de operația care se va realiza. De asemenea, unitatea de control are ca ieșire un semnal care stabilește dacă e nevoie ca o mantisă să fie transformată în complement față de doi, iar în caz afirmativ specifică, de asemenea, care dintre cele două trebuie modificate.

5. IMPLEMENTARE

Aplicația a fost realizată în mod structular, așa cum este prezentat în figura 3.

Modulul “Barrel Shifter” are ca intrări un număr pe 24 de biți care urmează să fie deplasat, un semnal “rightShift”, care este 1 dacă deplasarea se va face la dreapta, sau este 0 dacă deplasarea se va face la stânga și un număr “shift_amount” care reprezintă numărul de biți cu care se va face deplasarea. Ca semnal de ieșire, “nrOut” reprezintă mantisa după ce va fi deplasată.

Se ia pe rând fiecare bit din contor, iar dacă acel bit este 0 se va genera un semnal intermediar egal cu semnalul precedent. În caz contrar, se va genera un semnal intermediar care va stoca semnalul precedent, deplasat la dreapta sau la stânga, după caz, cu un număr de biți egal cu 2 la puterea poziției bitului curent.

La final, ieșirea shifter-ului va primi valoarea ultimului semnal intermediar generat.

```
36 entity Barrel_shifter is
37 port(nrIn : in std_logic_vector(23 downto 0);
38      rightShift : in std_logic;
39      shift_amount : in std_logic_vector(4 downto 0);
40      nrOut : out std_logic_vector(23 downto 0));
41 end Barrel_shifter;
42
43 architecture Behavioral of Barrel_shifter is
44
45 type intermediate_results is array (0 to 5) of std_logic_vector(31 downto 0);
46 signal ir : intermediate_results;
47
48 begin
49
50     ir(0) <= x"00" & nrIn;
51     nrOut <= ir(5)(23 downto 0);
52
53     gen : for i in 0 to 4 generate
54         process(ir(i), rightShift, shift_amount)
55         begin
56             if rightShift = '1' then
57                 if shift_amount(i) = '0' then
58                     ir(i + 1) <= ir(i);
59                 else
60                     ir(i + 1) <= ((2**i - 1) downto 0 => '0') & ir(i)(31 downto 2**i);
61                 end if;
62             else
63                 if shift_amount(i) = '0' then
64                     ir(i + 1) <= ir(i);
65                 else
66                     ir(i + 1) <= ir(i)((31 - 2**i) downto 0) & ((2**i - 1) downto 0 => '0');
67                 end if;
68             end if;
69         end process;
70     end generate;
71
72 end Behavioral;
```

Figura 4 Barrel Shifter

6. TESTARE ȘI VALIDARE

În figura 5 și figura 6 vor fi prezentate două dintre rezultatele experimentale.

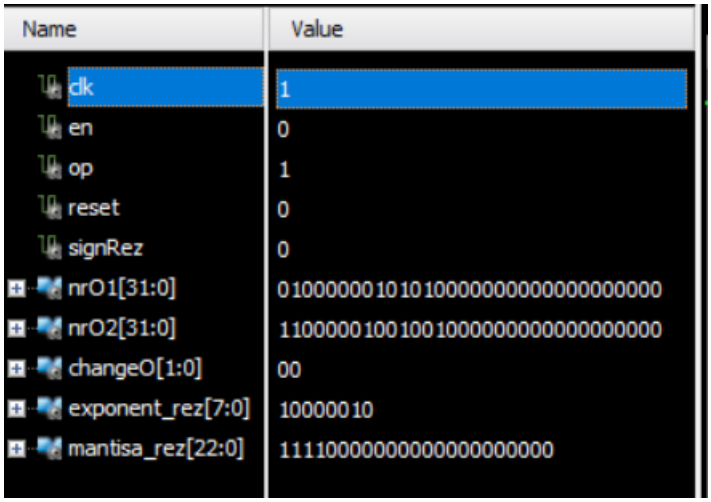


Figura 5 Operația de scădere

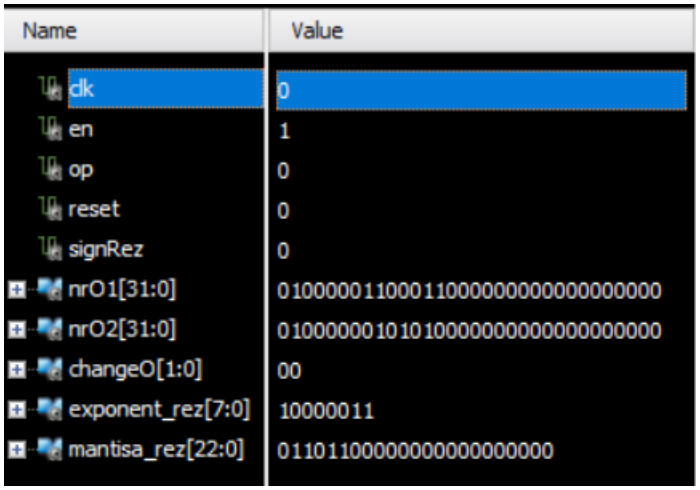


Figura 6 Operația de adunare

7. CONCLUZII

Scopul acestui proiect a fost proiectarea, implementarea și testarea a două operații cu numere reprezentate în virgulă flotantă, cu simpla precizie, pe 32 de biți, conform standardului IEEE.

Un aspect cheie al acestui proiect este faptul că scăderea este tratată la fel ca adunarea, singura diferență fiind semnul schimbat al celui de-al doilea operand. Acest fapt ușurează și reduce volumul de muncă al aplicației.

Pentru rezolvarea celor două operații, a fost necesară implementarea pe mai multe nivele, acestea fiind:

- Alinierea mantiselor
- Adunarea mantiselor
- Normalizarea rezultatului
- Rotunjirea rezultatului

Acest proiect poate fi îmbunătățit pe viitor prin adăugarea unor noi operații cu acest tip de numere, cum ar fi inmulțirea sau împărțirea.

BIBLIOGRAFIE

[1] https://users.utcluj.ro/~baruch/book_ac/AC-Adunare-VM.pdf

[2] https://www.utgjiu.ro/math/mbuneci/book/mn2007/c04.pdf

[3] https://www.princeton.edu/~rblee/ELE572Papers/Fall04Readings/Shifter_Schulte.pdf

[4] https://en.wikipedia.org/wiki/Barrel_shifter

[5] https://www.doc.ic.ac.uk/~eedwards/compsys/float/

[6] https://digitalsystemdesign.in/floating-point-addition-and-subtraction/