Universitatea Tehnică Cluj-Napoca
Facultatea de Automatică și Calculatoare
Departamentul Calculatoare

Proiect la disciplina
Structura Sistemelor de Calcul

Proiectarea unei unități aritmetico-logice în virgulă flotantă Adunare + Scădere

Student: Jișa Diana-Maria
Grupa: 30234
An academic: 2023–2024

Cuprins

1 Introducere 3

1.1 Context.....
1.2 Specificații.....
1.2 Obiective.....

2 Studiu bibliografic-tehnic .....

3 Analiză .....

3.1 Adunare.....
3.2 Scădere.....

4 Design.....

5 Detalii de implementare .....

6 Testare și Validare.....

7 Concluzii. Limitări și dezvoltări ulterioare.....

8 Bibliografie .....

1. Introducere

1.1 Context

Scopul principal al acestui proiect constă în dezvoltarea și implementarea unei Unități Aritmetico-Logice (ALU) specializate pentru a efectua operații de adunare și scădere pe numere în virgulă flotantă pe 32 de biți. Această ALU va păstra conformitatea cu standardul IEEE, dar va fi ajustată pentru a oferi o precizie simplă, potrivită pentru aplicații mai puțin complexe.

Această unitate reprezintă un element crucial în calculul numeric și are aplicații extinse în domenii precum știința, ingineria, grafica computerizată și procesarea semnalelor. Prin oferirea unui grad ridicat de flexibilitate în ceea ce privește implementarea sa, această soluție se poate integra cu ușurință într-o varietate de dispozitive, inclusiv procesoare sau microprocesoare.

1.2 Specificații

Dispozitivul va fi supus unui proces riguros de testare în mediul de dezvoltare oferit de Vivado, asigurându-ne că unitatea aritmetico-logică funcționează corespunzător. Pentru a garanta această funcționalitate, datele introduse de la tastatură vor fi inițial în format binar. În etapa de procesare, aceste date vor fi convertite în formatul adecvat pentru operațiile în virgulă mobilă conform standardului IEEE-754 Single Precision, care definește numerele pe 32 de biți împărțiți în trei componente: mantisă, exponent și bitul de semn, cu lungimi de 23, 8 și 1 biți, respectiv. Ulterior desfășurării acestor operații, rezultatul va fi retransformat în format binar. Această conversie bidirecțională facilitează introducerea și interpretarea datelor, contribuind la eficiența și accesibilitatea întregului proces.

1.3 Obiective

Principalele obiective ale acestui proiect constau în:

1. Proiectarea unei unități aritmetico-logice eficiente și precise, capabilă să efectueze operații de adunare și scădere în conformitate cu standardele pentru calculele în virgulă flotantă. Unitatea va primi numere în format binar la intrare, le va normaliza și va extrage toate componentele necesare reprezentării în virgula flotantă. Apoi, în funcție de opțiunea utilizatorului, va efectua operația de adunare sau de scădere. Rezultatele acestor operații vor fi disponibile pentru o vizualizare imediată și clară în simulator, furnizând utilizatorului informații privind rezultatele adunării sau scăderii.
2. Transpunerea unității aritmetico-logice în limbajul de descriere hardware, folosind mediul de dezvoltare Xilinx, pentru a asigura funcționarea corespunzătoare și conformitatea cu cerințele de precizie și performanță.
3. Documentarea completă a proiectului, furnizând informații detaliate despre design, implementare, testare și rezultatele obținute, pentru a putea fi utilizată ca resursă de referință.

2. Studiu bibliographic - tehnic

Standardul IEEE reprezintă numerele în virgulă mobilă cu precizie simplă folosind 1 bit pentru semnul numărului, 8 biți pentru exponent și 23 de biți pentru mantisă. Pe baza acestora, numărul este format utilizând formula următoare:

$$(-1)^{sign} \times 2^{exponent-127} \times 1.mantissa$$

Operațiile de adunare și scădere în virgulă flotantă prezintă o complexitate superioară față de operațiile de înmulțire și împărțire. Această complexitate derivă din necesitatea de a alinia exponenții pentru a efectua corect adunarea sau scăderea.

Procesul însumează patru etape principale:

1. **Alinierea mantiselor**: În prima etapă, se compară exponenții celor două numere și se aliniază mantisele în funcție de exponentul mai mic.
2. **Adunarea sau scăderea mantiselor**: Etapa următoare constă în adunarea sau scăderea mantiselor. În cazul scăderii, semnul celui de-al doilea operand este inversat.
3. **Normalizarea rezultatului**: După operația de adunare sau scădere, rezultatul trebuie normalizat pentru a se potrivi formatului zecimal mobil. Această normalizare implică ajustarea mantisei și exponenței pentru a obține un rezultat corect.
4. **Rotunjirea rezultatului**: În unele situații, rezultatul poate necesita rotunjire pentru a se încadra corespunzător în formatul zecimal mobil. Această etapă se ocupă de gestionarea cazurilor în care poate apărea o depășire a exponenței.

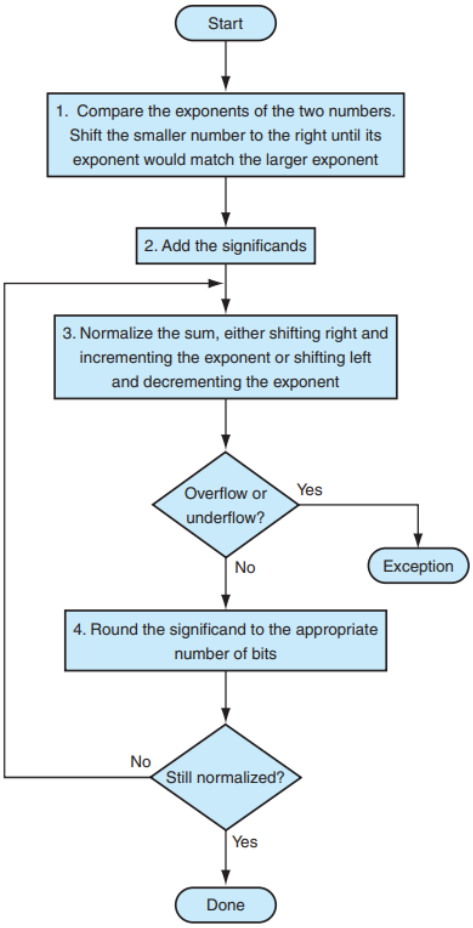
Pentru a executa aceste operații, cei doi operanzi trebuie transferați inițial în registrele destinate Unității Aritmetice și Logice (UAL). Dacă formatul de reprezentare în virgulă mobilă include un bit implicit pentru mantisă, acesta trebuie să fie explicit luat în considerare în cadrul operației. În general, exponenții și mantisele sunt menținuți separat în registre distincte și ulterior sunt reunite pentru a genera rezultatul final.

Procesul de aliniere a mantiselor presupune compararea exponenților și alinierea mantisei numărului cu exponent mai mic. Aceasta se realizează prin efectuarea repetată a deplasărilor la dreapta ale mantisei și prin incrementarea exponentului până când aceștia devin egali. În cazul în care mantisa care a fost deplasată devine zero, se raportează rezultatul ca fiind celălalt număr. De aceea, atunci când exponenții celor două numere sunt semnificativ diferiți, numărul cu exponent mai mic este ignorat.

Ulterior, mantisele sunt adunate, ținând cont de semnele lor. Din cauza diferențelor de semne, rezultatul poate fi zero. Există, de asemenea, posibilitatea ca mantisa rezultatului să depășească cu un bit formatul zecimal mobil. În astfel de cazuri, mantisa rezultatului este deplasată la dreapta cu o poziție, iar exponentul este incrementată. Dacă această deplasare duce la o depășire a exponenței, se semnalează o depășire, iar operația este întreruptă.

3. Analiză

3.1 Adunare



Imaginea prezentată mai sus ilustrează algoritmul utilizat pentru adunarea numerelor în format virgulă mobilă. În primii doi pași, mantisa numărului cu exponentul mai mic este ajustată, urmând ca apoi să se adune cele două mantise. Al treilea pas implică normalizarea rezultatului, ceea ce necesită verificarea pentru posibile cazuri de supraîncărcare sau subîncărcare. Această verificare din pasul 3 este influențată de precizia operanzilor implicați. Este important de menționat că în reprezentarea în virgulă mobilă, un model cu toți biții exponentului pe zero este utilizat pentru a semnifica numărul zero. De asemenea, un model cu toți biții exponentului pe unu este rezervat pentru a indica valori sau situații care depășesc limita numerelor normale în virgulă mobilă. Astfel, pentru precizia unică (single precision), exponentul maxim posibil este 127, iar cel minim este -126.

3.2 Scădere

Datorită naturii lor similare, implementarea adunării și scăderii într-o unitate logică aritmetică este remarcabil de asemănătoare. Principala diferență între cele două operații, în contextul acestui tip de unitate, constă în manipularea semnului operandului secundar: în timp ce adunarea păstrează semnul original al ambilor operanzi, scăderea se realizează prin simpla inversare a semnului operandului scăzut. Această abordare eficientizează designul și funcționalitatea unității aritmetice, permițând o tratare elegantă și eficientă a ambelor operații.

4. Design

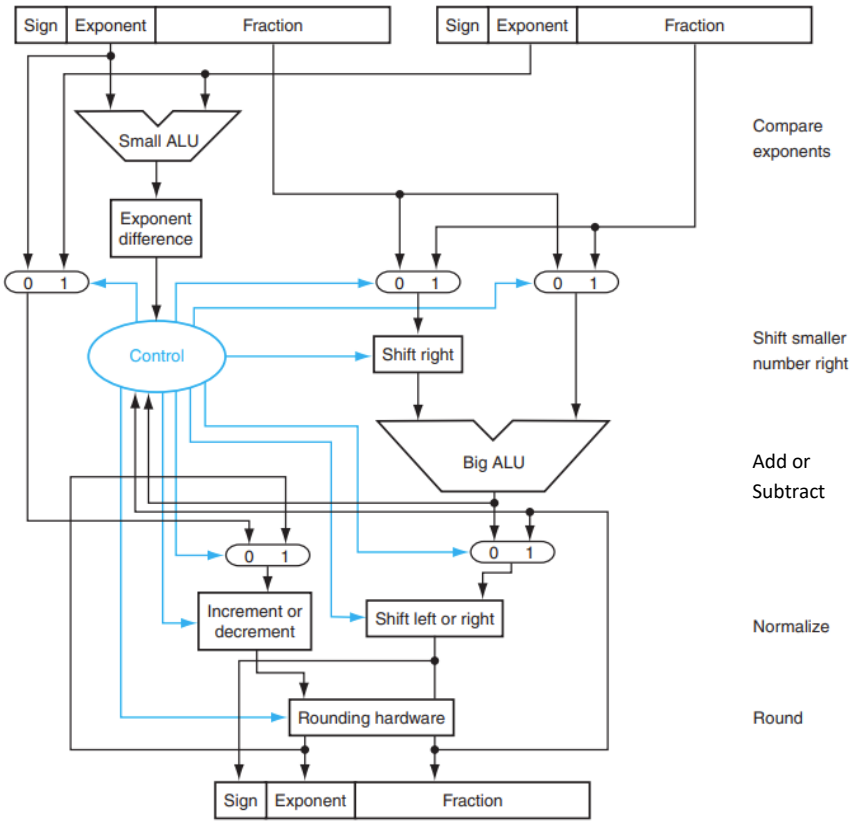


Diagrama bloc a unei unități aritmetice dedicate adunării și scăderii în virgulă mobilă ilustrează componentele și procesele esențiale pentru realizarea acestor operații. Începând cu partea superioară a diagramei, pașii sunt dispusi în ordine logică. Inițial, se compară exponenții celor doi operanzi folosind o ALU de dimensiuni reduse pentru a stabili care este mai mare și diferența dintre ei. Această diferență este esențială pentru controlul celor trei multiplexoare următoare, care, de la stânga la dreapta, selectează exponentul mai mare, mantisa numărului cu exponentul mai mic (partea semnificativă) și mantisa numărului cu exponentul mai mare. Mantisa

numărului cu exponentul mai mic este apoi deplasată spre dreapta pentru aliniere, urmând ca mantisele să fie adunate sau scăzute, în funcție de operația necesară, cu ajutorul unei ALU de dimensiuni mai mari. Următorul pas, cel de normalizare, implică deplasarea la stânga sau la dreapta a sumei sau diferenței obținute, cu ajustări corespunzătoare ale exponentului. În final, procesul de rotunjire determină formarea rezultatului final, care poate necesita o nouă normalizare pentru a se conforma standardelor de virgulă mobilă

5. Detalii de implementare

Modulul WaitState

Modulul WaitState este responsabil pentru inițializarea și pregătirea datelor de intrare pentru procesarea ulterioară. Acesta așteaptă un semnal de start și, odată primit, extrage și formatează semnul (bitul cel mai semnificativ), exponentul (următorii 8 biți) și mantisa (restul biților) numerelor de intrare în formatul IEEE 754.

Modulul AlignState

AlignState aliniază mantisele celor două numere de intrare pe baza exponenților lor. Scopul este de a pregăti numerele pentru operații aritmetice ulterioare asigurându-se că acestea au același exponent.Modulul calculează diferența dintre exponenți și deplasează mantisa numărului cu exponentul cel mai mic pentru a se alinia cu cealalt număr. Rezultatul este stocat în aligned_nr1_mantissa și aligned_nr2_mantissa, iar exponentul comun este salvat în res_exp.

```
if unsigned(nr1_exp) > unsigned(nr2_exp) then
    diff := signed(nr1_exp) - signed(nr2_exp);
    shift_amount := to_integer(diff);
    res_exp <= nr1_exp;
    if (shift_amount < 24) then
        aligned_nr1_mantissa <= nr1_mantissa;
        aligned_nr2_mantissa <= std_logic_vector(unsigned(nr2_mantissa) srl shift_amount);
    end if;
elsif unsigned(nr1_exp) < unsigned(nr2_exp) then
    diff := signed(nr2_exp) - signed(nr1_exp);
    shift_amount := to_integer(diff);
    res_exp <= nr2_exp;
    if (shift_amount < 24) then
        aligned_nr1_mantissa <= std_logic_vector(unsigned(nr1_mantissa) srl shift_amount);
        aligned_nr2_mantissa <= nr2_mantissa;
    end if;
else
    res_exp <= nr1_exp;
    aligned_nr1_mantissa <= nr1_mantissa;
    aligned_nr2_mantissa <= nr2_mantissa;
end if;
next_state <= OPERATION_STATE;
```

Modulul OperationState

Modulul determină operația pe baza semnalelor de intrare nr1_sgn, nr2_sgn și op. În funcție de aceste semnale, se realizează adunarea sau scăderea mantiselor. Semnul rezultatului este stabilit în funcție de semnele numerelor de intrare și rezultatul operației.

```
          if op = '0' then
             if (nr1_sgn = nr2_sgn) then
                 res_mantissa <= std_logic_vector((unsigned(nr1_mantissa) + unsigned(nr2_mantissa)));
                 res_sgn <= nr1_sgn;
             elsif unsigned(nr1_mantissa) >= unsigned(nr2_mantissa) then
                 res_mantissa <= std_logic_vector((unsigned(nr1_mantissa) - unsigned(nr2_mantissa)));
                 res_sgn <= nr1_sgn;
             else
                 res_mantissa <= std_logic_vector((unsigned(nr2_mantissa) - unsigned(nr1_mantissa)));
                 res_sgn <= nr2_sgn;
             end if;
          else
             if (nr1_sgn /= nr2_sgn) then
                 res_mantissa <= std_logic_vector((unsigned(nr1_mantissa) + unsigned(nr2_mantissa)));
                 res_sgn <= nr1_sgn;
             elsif unsigned(nr1_mantissa) >= unsigned(nr2_mantissa) then
                  res_mantissa <= std_logic_vector((unsigned(nr1_mantissa) - unsigned(nr2_mantissa)));
                  res_sgn <= nr1_sgn;
             else
                  res_mantissa <= std_logic_vector((unsigned(nr2_mantissa) - unsigned(nr1_mantissa)));
                  res_sgn <= not nr1_sgn;
             end if;
          end if;
          next_state <= NORMALIZE_STATE;
       end if;
    end process;
end Behavioral;
```

Modulul NormalizeState

NormalizeState normalizează rezultatul operației pentru a se conforma standardului IEEE 754. Modulul verifică dacă mantisa rezultatului necesită normalizare (deplasarea la stânga sau la dreapta) și ajustează exponentul corespunzător. Procesul asigură că rezultatul este în formatul corect IEEE 754.

```
architecture Behavioral of NormalizeState is
begin
    process(clk, rst)
    begin
        if rst = '1' then
            -- Resetarea ieșirilor
            res_mantissa_out <= (others => '0');
            res_exp_out <= (others => '0');
            next_state <= WAIT_STATE;
        elsif rising_edge(clk) then
            -- Logica de normalizare
            if unsigned(res_mantissa_in) = TO_UNSIGNED(0, 25) then
                res_mantissa_out <= (others => '0');
                res_exp_out <= (others => '0');
                next_state <= ROUNDING_STATE;
            elsif(res_mantissa_in(24) = '1') then
                res_mantissa_out <= '0' & res_mantissa_in(24 downto 1);
                res_exp_out  <= std_logic_vector(unsigned(res_exp_in) + 1);
                next_state <= ROUNDING_STATE;
            elsif(res_mantissa_in(23) = '0') then
                res_mantissa_out <= res_mantissa_in(23 downto 0) & '0';
                res_exp_out <= std_logic_vector(unsigned(res_exp_in) - 1);
                next_state<= NORMALIZE_STATE;
            else
                res_mantissa_out <= res_mantissa_in;
                res_exp_out <= res_exp_in;
                next_state <= ROUNDING_STATE;
            end if;
        end if;
    end process;
end Behavioral;
```

Modulul RoundingState

RoundingState finalizează procesul de calcul prin rotunjirea mantisei.Acest modul efectuează rotunjirea mantisei rezultatului, setează exponentul și semnul, și formează numărul final în format IEEE 754. De asemenea, semnalul done este setat pentru a indica finalizarea procesului de calcul.

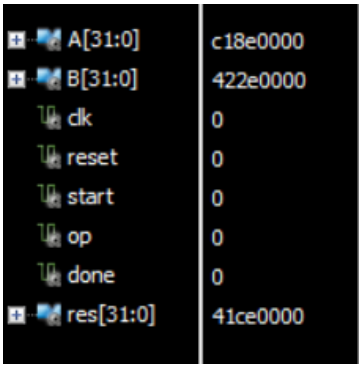
Modulul FloatingPointUnit

FloatingPointUnit coordonează toate modulele anterioare pentru a realiza operații cu numere în virgulă mobilă.

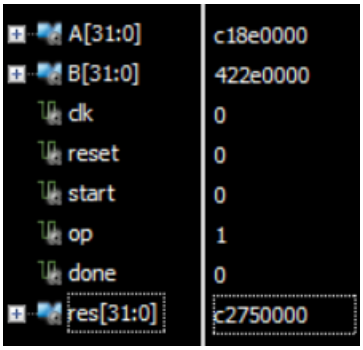
6. Testare și Validare

Exemplele de mai jos au fost folosite pentru a testa funcționalitatea corectă a unității aritmetico-logice utilizând numere în reprezentarea cu virgulă mobile:

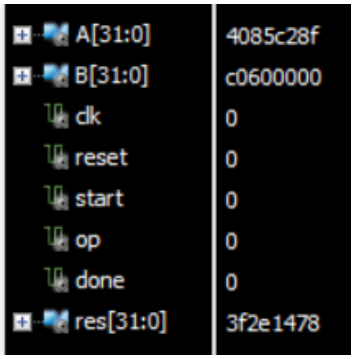
- $A = -17.75_{10} = 1.10000011.00011100000000000000000_2 = c18e0000_{16}$
  $B = 43.5_{10} = 0.10000100.01011100000000000000000_2 = 422e0000_{16}$
  $R = A + B = 25.75_{10} = 0.10000011.10011100000000000000000_2 = 41ce0000_{16}$



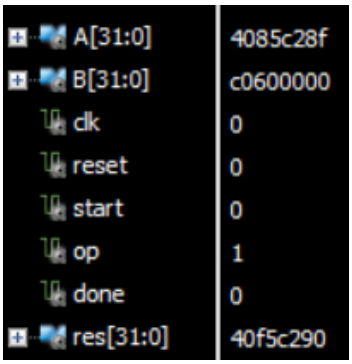
$R = A - B = -61.25_{10} = 1.10000100.11101010000000000000000_2 = c2750000_{16}$



- $A = 4.17999982834_{10} = 0.10000001.00001011100001010001111_2 = 4085c28f_{16}$
  $B = -3.5_{10} = 1.10000000.11000000000000000000000_2 = c0600000_{16}$
  $R = A + B = 0.679999828339_{10} = 0.01111110.01011100001010001111000_2 = 3f2e1478_{16}$



  $R = A - B = 7.68000030518_{10}= 0.10000001.11101011100001010010000_2 = 40f5c290_{16}$



7. Concluzii. Limitări și dezvoltări ulterioare

În concluzie, în cadrul acestei elaborări pentru proiectarea unei unități aritmetico-logice în virgulă mobilă, cu accent pe operațiile de adunare și scădere, am îndeplinit cu succes obiectivele propuse, implementând aceste operații conform standardului IEEE pentru formatul pe 32 de biți. Procesul de proiectare a fost complex, cu provocări semnificative, iar una dintre sarcinile cele mai complexe a fost proiectarea și implementarea etapei de aliniere, dar si a celei de normalizare. Aceaste etape au implicat o analiză detaliată a resurselor și soluțiilor existente, culminând într-o abordare personalizată și eficientă. Realizarea acestui proiect a contribuit la o înțelegere mai profundă a mașinilor pe care le programăm și a modului în care operațiile de bază sunt realizate la nivel hardware.

Pentru a îmbunătăți proiectul în viitor, o direcție posibilă ar putea fi extinderea funcționalității pentru a include operațiile de înmulțire și împărțire, aducând o complexitate suplimentară și explorând noi aspecte ale aritmeticii în virgulă mobilă. De asemenea, optimizările privind performanța și eficiența unității aritmetico-logice ar putea facilita integrarea acesteia în sisteme mai ample.

În concluzie, deși proiectul a atins cu succes obiectivele inițiale, identificarea limitărilor și direcțiile posibile de dezvoltare ulterioară subliniază potențialul continuu de creștere și îmbunătățire a acestei unități aritmetico-logice în contextul virgulei mobile.

8. Bibliografie

1. https://users.utcluj.ro/~baruch/book_ac/AC-Adunare-VM.pdf
2. https://www.h-schmidt.net/FloatConverter/IEEE754.html
3. https://www.ijert.org/research/an-fpga-based-double-precision-floating-point-arithmetic-unit-using-verilog-IJERTV2IS100200.pdf
4. http://www.ijvdcs.org/uploads/143625IJVDCS17341-06.pdf
5. http://www.csjournals.com/IJEE/PDF%202-1/40.pdf