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

Unitate de calcul în virgulă mobilă -împărțirea-

Matiș Oana-Antonia

Grupa: 30238

Îndrumător: Lișman Dragoș Florin

Cuprins

1. Rezumat

Proiectul vizează implementarea și testarea operației de împărțire pe numere în virgulă mobilă cu precizie simplă. Acest unitate de calcul se adresează persoanelor care necesită un calculator capabil să efectueze această operație și poate fi, de asemenea, integrată într-un alt proiect, cum ar fi un procesor, care o va utiliza pentru a-și îndeplini sarcinile.

Reprezentarea în virgulă mobilă a unui număr este utilizată în cazul numerelor extrem de mari sau mici. Pentru a reprezenta un număr în virgulă mobilă, sunt necesare trei componente: semnul, exponentul (magnitudinea numărului) și mantisa. Important de menționat este faptul că această reprezentare nu corespunde reprezentării numerelor reale din matematică, ci este adaptată pentru a facilita operațiile calculatorului. Cu toate acestea, pot apărea dificultăți în utilizarea acestei reprezentări, deoarece multe numere nu au o reprezentare precisă în virgulă mobilă (de exemplu, numărul 0.3), iar rezultatul va fi, în consecință, unul aproximativ.

Pentru testarea proiectului am create un test bench în care am aplicat operația de împărțire asupra mai multor perechi de numere, pentru a se observa funcționalitatea unității. În waveform am adăugat stări intermediare pentru a se evidenția procesul de calculare a rezultatului.

Proiectul funcționează în concordanță cu scopul initial, deîmpărțitul, împărțitorul și rezultatul fiind vizibile în format hexadecimal pe placa Basys3.

2. Introducere

Operația de împărțire reprezintă una dintre cele mai complexe operații în virgulă mobilă, deosebit de diferită de alte operații uzuale.

Există mai multe modalități de implementare a operației de împărțire, iar fiecare abordare are propriile avantaje și dezavantaje, influențate de viteza de execuție a operațiilor și de resursele hardware necesare. Detalii suplimentare referitoare la aceste aspecte vor fi explicate în secțiunea de Fundamentare Teoretică.

În cadrul capitolului de Proiectare și Implementare, am detaliat modul în care am implementat proiectul și scopurile pentru care am folosit fiecare entitate. În secțiunea de

Rezultate Experimentale, am generat exemple pentru a testa operațiile pe diferite valori. Concluziile legate de acest proiect sunt prezentate în capitolul de Concluzii, iar în capitolul de Bibliografie sunt menționate sursele care au stat la baza elaborării secțiunii de Fundamentare Teoretică.

3. Fundamentare teoretică

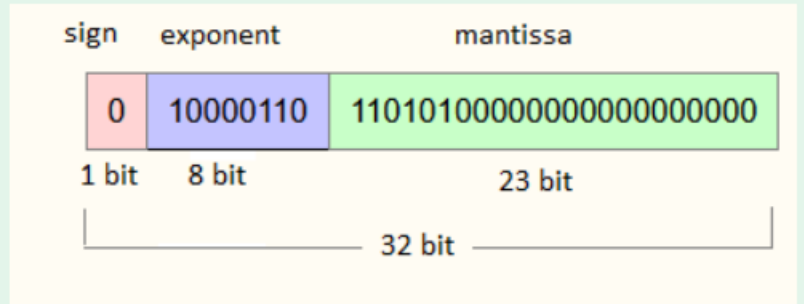
3.1. Reprezentarea numerelor în virgulă mobilă

În general, un număr N se poate reprezenta în virgulă mobilă (VM) în forma următoare:

$$N = \pm M \cdot B^{\pm E}$$

Un număr reprezentat în VM are două componente. Prima componentă este mantisa (M), care indică valoarea exactă a numărului într-un anumit domeniu, fiind reprezentată de obicei ca un număr fracţionar cu semn. A doua componentă este exponentul (E), care indică ordinul de mărime al numărului. În expresia de sus, B este baza exponentului.

Această reprezentare poate fi memorată într-un cuvânt binar cu trei câmpuri: semnul, mantisa şi exponentul. De exemplu, presupunând un cuvânt de 32 de biţi, o asignare posibilă a biţilor la fiecare câmp poate fi următoarea:



Aceasta este o reprezentare în mărime şi semn, deoarece semnul are un câmp separat faţă de restul numărului. Câmpul de semn constă dintr-un bit care indică semnul numărului, 0 pentru un număr pozitiv şi 1 pentru un număr negativ. Nu există un câmp rezervat pentru baza B, deoarece această bază este implicită şi ea nu trebuie memorată, fiind aceeaşi pentru toate numerele.

De obicei, câmpul rezervat exponentului nu conţine exponentul real, ci o valoare numită caracteristică, care se obţine prin adunarea unui deplasament la exponent,

astfel încât să rezulte întotdeauna o valoare pozitivă. Astfel, nu este necesar să se rezerve un câmp separat pentru semnul exponentului. Caracteristica C este deci exponentul deplasat:

$$C = E + deplasament$$

Valoarea reală a exponentului se poate afla prin scăderea deplasamentului din caracteristica numărului. De exemplu, dacă pentru caracteristică se rezervă un câmp de 8 biţi, valorile caracteristicii pot fi cuprinse între 0 şi 255. Considerând un deplasament de 128 (80h), exponentul real poate lua valori între –128 şi +127, fiind negativ dacă C < 128, pozitiv dacă C > 128, şi zero dacă C = 128. Exponentul este deci reprezentat în exces 128.

Unul din avantajele utilizării exponentului deplasat constă în simplificarea operaţiilor executate cu exponentul, datorită lipsei exponenţilor negativi. Al doilea avantaj se referă la modul de reprezentare al numărului zero. Mantisa numărului zero are cifre de 0 în toate poziţiile. Exponentul numărului zero poate avea, teoretic, orice valoare, rezultatul fiind tot zero. La unele calculatoare, dacă un rezultat are mantisa zero, exponentul rămâne la valoarea pe care o are în momentul respectiv, rezultând un "zero impur".

În reprezentarea ilustrată anterior, mantisa constă din 23 de biţi. Deşi virgula binară nu este reprezentată, se presupune că ea este aşezată înaintea bitului c.m.s. al mantisei. De exemplu, dacă B = 2, numărul 1,75 poate fi reprezentat sub mai multe forme:

$+0{,}111 \cdot 2^1$	0	1000 0001	1110 0000 0000 0000 0000 000
$+0{,}00111 \cdot 2^3$	0	1000 0011	0011 1000 0000 0000 0000 000

Pentru simplificarea operaţiilor cu numere în VM şi pentru creşterea preciziei acestora, se utilizează reprezentarea sub forma normalizată. Un număr în VM este normalizat dacă bitul c.m.s. al mantisei este 1. Din cele două reprezentări ale numărului 1,75 ilustrate anterior, prima este cea normalizată.

Deoarece bitul c.m.s. al unui număr normalizat în VM este întotdeauna 1, acest bit nu este de obicei memorat, fiind un bit ascuns la dreapta virgulei binare. Aceasta permite ca mantisa să aibă un bit semnificativ în plus. Astfel, câmpul de 23 de biţi este utilizat pentru memorarea unei mantise de 24 de biţi cu valori cuprinse între 0,5 şi 1,0.

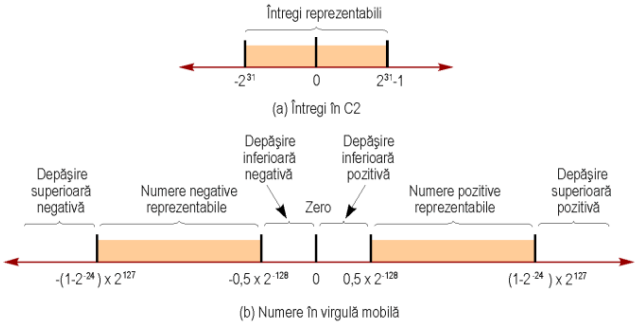


Figura 2.1. Numere care pot fi reprezentate în formate tipice de 32 de biţi.

3.2. Standardul IEEE 754

Standardul IEEE 754 defineşte următoarele formate sau precizii: precizie simplă, precizie simplă extinsă, precizie dublă şi precizie dublă extinsă. Parametrii principali ai acestor formate sunt prezentaţi în Tabelul 2.7. Standardul nu precizează ca obligatorie implementarea tuturor formatelor, dar recomandă implementarea combinaţiei formatelor cu precizie simplă şi precizie simplă extinsă, sau a formatelor cu precizie simplă, precizie dublă şi precizie dublă extinsă.

Tabelul 2.7. Parametrii formatelor definite de standardul IEEE 754.

	Precizie simplă	**Precizie simplă extinsă**	**Precizie dublă**	**Precizie dublă extinsă**
Biți ai mantisei	24	≥ 32	53	≥ 64
Exponent real maxim	127	≥ 1023	1023	≥ 16383
Exponent real minim	-126	≤ -1022	-1022	≤ -16382
Deplasament exponent	127	Nespecificat	1023	Nespecificat

Pentru toate formatele, baza implicită este 2. Formatele cu precizie simplă, precizie dublă şi precizie dublă extinsă sunt prezentate în Figura 2.2. Coprocesoarele matematice şi unităţile de calcul în virgulă mobilă ale procesoarelor implementează de obicei aceste formate.

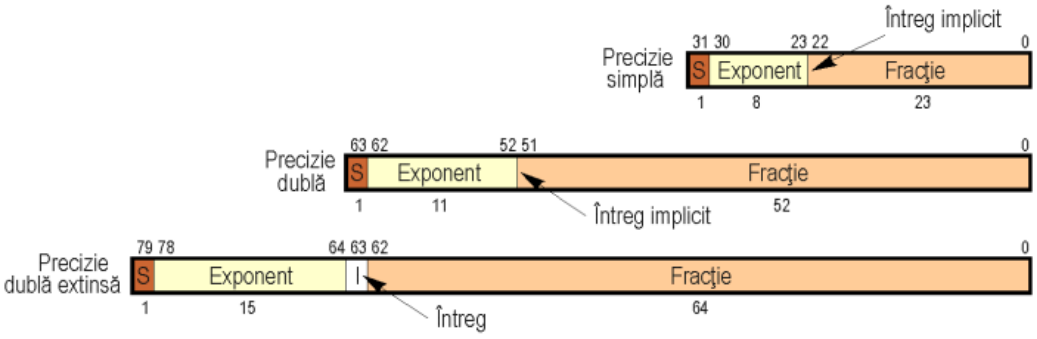


Figura 2.2. Formatele cu precizie simplă, precizie dublă şi precizie dublă extinsă definite de standardul IEEE 754.

3.3. Împărțirea în virgulă mobilă

Pașii implementării operației de împărțire

- Efectuarea operației logice XOR pe biții de semn
- Dacă divizorul este 0, rezultatul este infinit (sau Nan), iar în test bench sau pe afișajul cu 7 segmente va apărea valoarea x"7ff80000"
- Scăderea exponenților și adunarea bias-ului (bias = 127)
- Efectuarea operației de împărțire asupra mantiselor
- Normalizarea rezultatului dacă este necesar

Algoritmul de împărțire este cel fără refacerea restului parțial. Începe prin scăderea exponenților, după care se împart mantisele și se face XOR biților de semn,

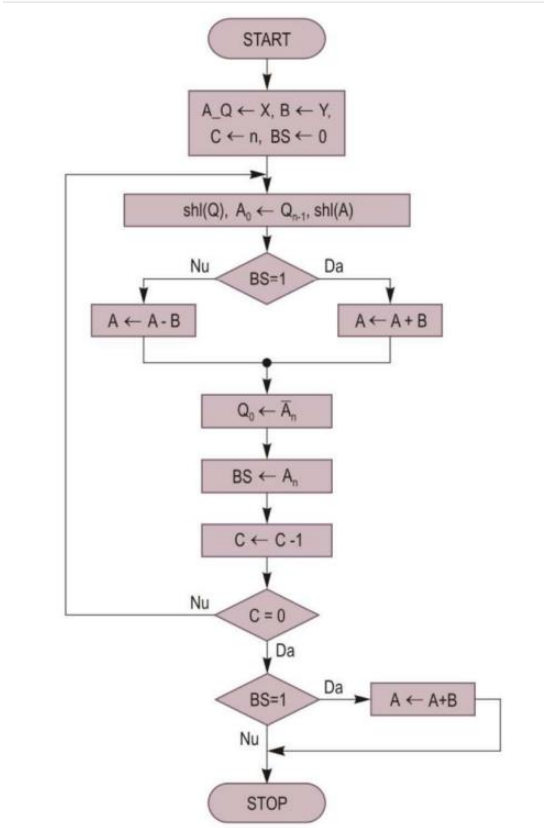
Pentru scăderea, vom utiliza un scăzător, iar pentru împărțirea mantiselor, vom folosi un divizor care va urma algoritmul de împărțire așa cum este pe hârtie (verificând dacă numitorul se potrivește în numărător, dacă da, deplasăm un '1' în rezultat și scădem numărătorul din numitor; altfel, deplasăm un '0' în rezultat; în următorul pas, vom lua în considerare încă un bit pentru numărător, până când ajungem la precizia semnalelor noastre, care este de 23 de biți).

Dezavantajele acestuia sunt overflow (când împărțim un număr foarte mare la unul foarte mic, rezultatul scăderii exponentelor ar putea duce la o depășire și ar arăta un număr foarte mic în loc de unul mai mare) și underflow (când împărțim un număr foarte mic la unul foarte mare, rezultatul scăderii exponenților ar putea duce la underflow și ar furniza un număr foarte mare în loc), dar și împărțirea la zero, care va fi rezolvată afișând un mesaj x"7ff80000" (infinit) pe afișajul cu 7 segmente.

4. Proiectare și implementare

4.1. Proiectare

Schema pe baza căreia am implementat algoritmul de împărțire al mantiselor este următoarea:

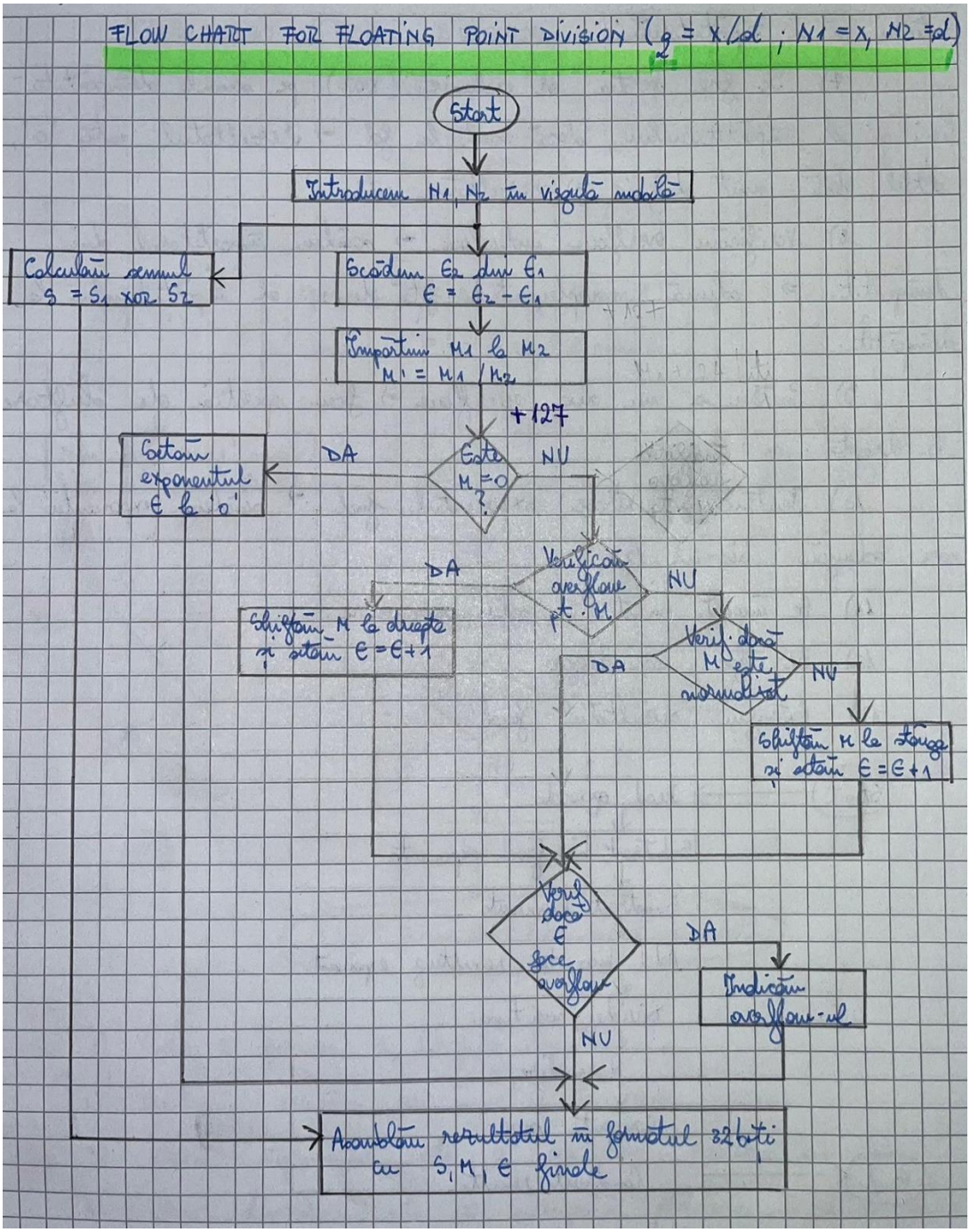


De asemenea, am folosit exemplul de împărțire fără refacerea restului parțial din laboratorul de SSC:

Pas	A	Q	B	Q_0	BS	C	Operații
0	0 0000	1110	0011	0	0	4	Inițializare
1	0 0001 + 1 1101 1 1110 +	1100 1100	0011	0	1	3	Deplasare A_Q la stânga Scădere împărțitor
2	1 1101 + 0 0011 0 0000 +	1000 1001	0011	1	0	2	Deplasare A_Q la stânga Adunare împărțitor
3	0 0001 + 1 1101 1 1110	0010 0010	0011	0	1	1	Deplasare A_Q la stânga Scădere împărțitor
4	1 1100 + 0 0011 1 1111 +	0100 0100	0011	0	1	0	Deplasare A_Q la stânga Adunare împărțitor
5	0 0011 0 0010	0100	0011	-	0	0	Adunare împărțitor

Câtul obținut în registrul Q este $0100_2 = 4$, iar restul obținut în registrul A este $0\ 0010_2 = 2$.

Organigrama unității de împărțire în virgulă mobilă:



4.2. Implementare

Implementarea unității de împărțire în virgulă mobilă a fost realizată prin împărțirea proiectului în următoarele entități de bază:

- **Divider_Unit**

- Această entitate reprezintă un modul de împărțire implementat în limbajul VHDL. Modulul primește două numere pe 24 de biți (A și B), un semnal de ceas (Clk), un semnal de start (Start) și furnizează rezultatul împărțirii (Result) pe 25 de biți și un semnal de finalizare (Finish).
- **Descrierea funcționalității:**

La activarea semnalului de start (Start), modulul inițializează variabilele interne pentru a începe operația de împărțire.

```
if(rising_edge(Clk)) then
      if (Start = '1') then
        A_Sig(47 downto 24) <= (others => '0');
        A_Sig(23 downto 0) <= A;
        B_Sig(47 downto 24) <= (others => '0');
        B_Sig(23 downto 0) <= B;
        Res_Sig <= (others => '0');
        Count <= "101111";
        Finish_Signal <= '0';
        Dividend(47 downto 24) <= (others => '0');
        Dividend(23 downto 0) <= A;
      else
        if (Finish_Signal = '0') then
         --Continue the division until the Step become "00000"
            if (Dividend >= B_Sig) then
            -- Set result bit to '1', subtract divisor from the current Divident
                Res_Sig(conv_integer(Count)) <= '1';
                Dividend <= (Dividend(46 downto 0) - B_Sig(46 downto 0)) & '0';
            else
            -- Set result bit to '0', shift the current Divident
                Res_Sig(conv_integer(Count)) <= '0';
                Dividend <= Dividend(46 downto 0) & '0';
            end if;
            -- Update Step value
            if (Count = "00000") then
                Finish_Signal <= '1';
            else
                Count <= Count - '1';
            end if;
```

Modulul folosește un proces sincronizat cu semnalul de ceas (Clk) pentru a gestiona stările și operațiile împărțirii.

Dividentul (A) este extins la 48 de biți, iar divizorul (B) este de asemenea extins pentru operațiile ulterioare.

În fiecare ciclu de ceas, modulul compară dividentul cu divizorul pentru a determina dacă se va seta un bit în rezultat.

Dacă dividentul este mai mare sau egal cu divizorul, modulul setează un bit în rezultat, scade divizorul din divident și continuă cu următorul ciclu de ceas.

Dacă dividentul este mai mic decât divizorul, se setează un alt bit în rezultat, iar dividentul este doar deplasat la stânga.

Procesul continuă până când toate ciclurile de împărțire sunt parcurse, moment în care semnalul de finalizare (Finish) este activat.

Rezultatul final este ajustat în funcție de un semnal de aproximare și furnizat pe ieșirea Result.

```
process(Res_Sig, Approximate_Sig)
begin
        Approximate_Sig <= '0';
        for i in 22 downto 0 loop --Check each bit of the result
            if (Res_Sig(i) = '1') then
                Approximate_Sig <= '1';
            end if;
         end loop;

            if (Approximate_Sig = '1') then --based on Approximate_Sig assign the Result
                Result <= Res_Sig(47 downto 23) + 1;
            else
                Result <= Res_Sig(47 downto 23);
            end if;
end process;
```

- **FPU_Division**

- **Descrierea funcționalității:**

Inițializarea semnalelor și a variabilelor interne înaintea începerii operației.

Configurarea semnului rezultatului (Res_Sign), semnului mantiselor numerelor de intrare (A_Sign și B_Sign).

Configurarea semnalelor pentru exponenți (A_Exp, B_Exp, Res_Exp) și mantise (A_Mantissa, B_Mantissa, Res_Mantissa).

Construirea mantiselor cu un bit 1 adăugat la început (A_Mantissa_With_One și B_Mantissa_With_One). Acest pas este necesar pentru a asigura corectitudinea operației de diviziune a mantiselor.

Calcularea semnului rezultatului (Res_Sign) utilizând operația de XOR între semnele numerelor de intrare (A_Sign și B_Sign).

Calcularea semnalului de normalizare (Normalization) pe baza unui bit din rezultatul operației de diviziune a mantiselor.

Calculul exponenților rezultatului (Res_Exp) pe baza exponenților numerelor de intrare (A_Exp și B_Exp) și a semnalului de normalizare.

Determinarea excepțiilor care pot apărea în operația de diviziune. Excepții precum depășirea sau împărțirea la zero sunt identificate și semnalizate.

Invocarea submodulului Divider pentru a efectua operația de divizare a mantiselor numerelor de intrare (A_Mantissa_With_One și B_Mantissa_With_One).

Submodulul primește semnalele necesare, inclusiv semnalul de ceas (Clk) și semnalul de start (Start).

Calcularea mantisei rezultatului (Res_Mantissa) pe baza semnalului de normalizare și a rezultatului operației de divizare a mantiselor.

Asamblarea rezultatului final (Result) pe baza semnelor, exponenților și mantisei calculate.

În caz de excepție (depășire sau împărțire la zero), se utilizează o valoare constantă reprezentând "Not a Number" (NaN_Value).

Semnalizarea finalizării operației prin semnalul Finish_Signal.

Semnalizarea excepțiilor prin activarea semnalului Exceptions_Enable și furnizarea rezultatului excepțiilor (Exceptions_Result) submodulului Exceptions.

Furnizarea rezultatului final (Result) către componenta superioară.

Semnalizarea terminării operației prin semnalul Finish.

Finalizarea operației și alocarea resurselor folosite.

```
-- Compute the sign
Res_Sign <= A_Sign xor B_Sign;


A_Mantissa_With_One <= '1' & A_Mantissa;
B_Mantissa_With_One <= '1' & B_Mantissa;


Normalization <= not Division_Signal(24);
Res_Exp <= (A_Exp - B_Exp) + 127 - Normalization;
-- Check for exceptions
Overflow_Exception <= '1' when (A_Exp + B_Exp > 254) else '0';
Divide_By_Zero_Exception <= '1' when (B(31) = '0' and B(30 downto 0) = "00000000000000000000000000000") or
                                     (B(31) = '1' and B(30 downto 0) = "11111111111111111111111111111") else '0';
-- Calculate the mantissa
Res_Mantissa <= Division_Signal(23 downto 1) when Normalization = '0'
                            else Division_Signal(22 downto 0);


Exceptions_Enable <=  Overflow_Exception or Divide_By_Zero_Exception;
Result <= Nan_Value when Divide_By_Zero_Exception = '1' else
          Res_Sign & Res_Exp & Res_Mantissa;
```

- **Exceptions**

-Descrierea funcționalității

În cadrul procesului principal, se calculează exponenții numerelor de intrare A și B prin interpretarea câmpurilor de exponenți.

Se verifică mai multe condiții pentru a identifica tipul de excepție:

Împărțirea la zero: condiția când B este zero sau negativ zero.

Overflow: condiția când suma exponenților depășește limita validă.

Underflow: condiția când suma exponenților este prea mică.

În funcție de condiții, se atribuie unei variabile auxiliare aux o valoare corespunzătoare:

```
if B = X"00000000" then
  -- Division by zero condition: B is zero
  aux <= NaN_Value;
elsif B = X"80000000" then
  -- Division by zero condition: B is negative zero
  aux <= NaN_Value;
elsif exp_A + exp_B > 254 then
  -- Overflow condition: sum of exponents exceeds valid range
  aux <= X"7F800000";  -- Positive infinity
elsif exp_A + exp_B < 1 then
  -- Underflow condition: sum of exponents is too small
  aux <= X"00000000";  -- Zero
else
  aux <= X"00000001";
end if;
```

Pentru împărțirea la zero, se atribuie o reprezentare a "Not a Number" (NaN).

Pentru overflow, se atribuie o reprezentare a infinitului pozitiv.

Pentru underflow, se atribuie o reprezentare a zero-ului.

Semnalizarea excepțiilor:

enable primește valoarea '1' atunci când condițiile pentru împărțire la zero sau depășire sunt îndeplinite, indicând astfel activarea excepțiilor.

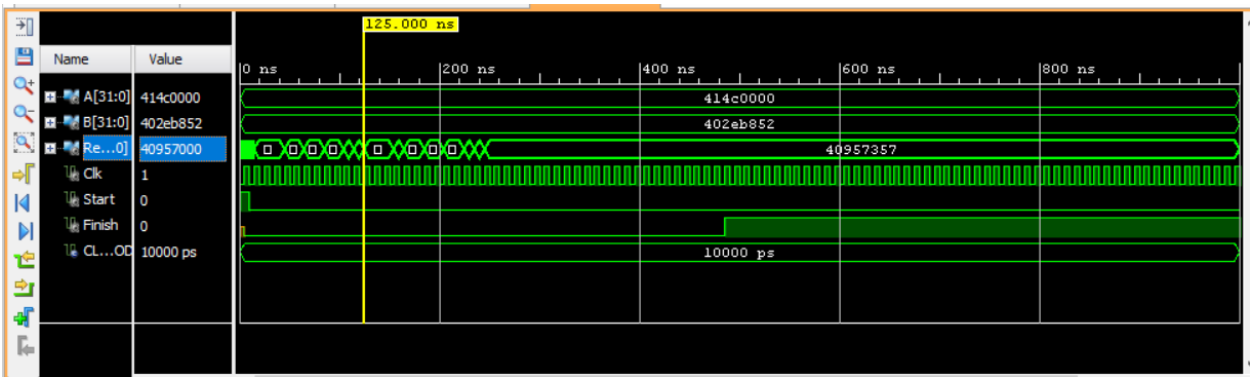
Result primește valoarea specifică excepțiilor calculate anterior.

Rezultatul final și semnalul de activare a excepțiilor sunt furnizate către componenta superioară pentru gestionarea acestora în cadrul operației de diviziune a FPU-ului.

5. Rezultate experimentale

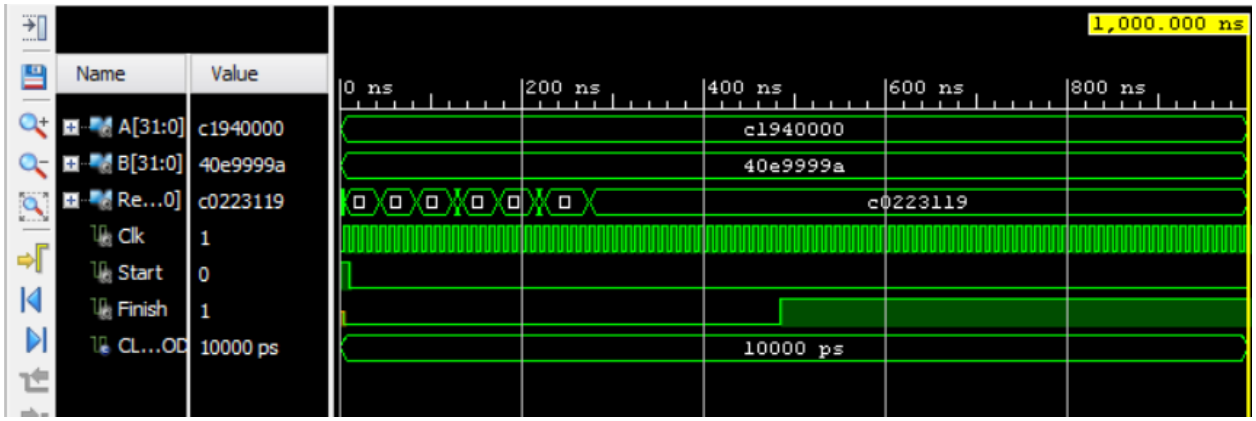
Pentru testarea aplicației am creat un modul de Test Bench al modulului care realizează împărțirea, mai exact FPU_Division. Rezultatele se pot observa în imaginile de mai jos.

```
-- Test case 1 - Normal division
A <= x"414c0000"; -- Example: 12.75
B <= x"402eb852"; -- Example: 2.73000001907
Start <= '1';
wait for 10 ns;
Start <= '0';
wait until Finish = '1';
if Result = x"40957357" then -- Expected result: 4.67
  report "Test case 1 passed: Correct result"
    severity note;
else
  report "Test case 1 failed: Incorrect result"
    severity error;
end if;
```



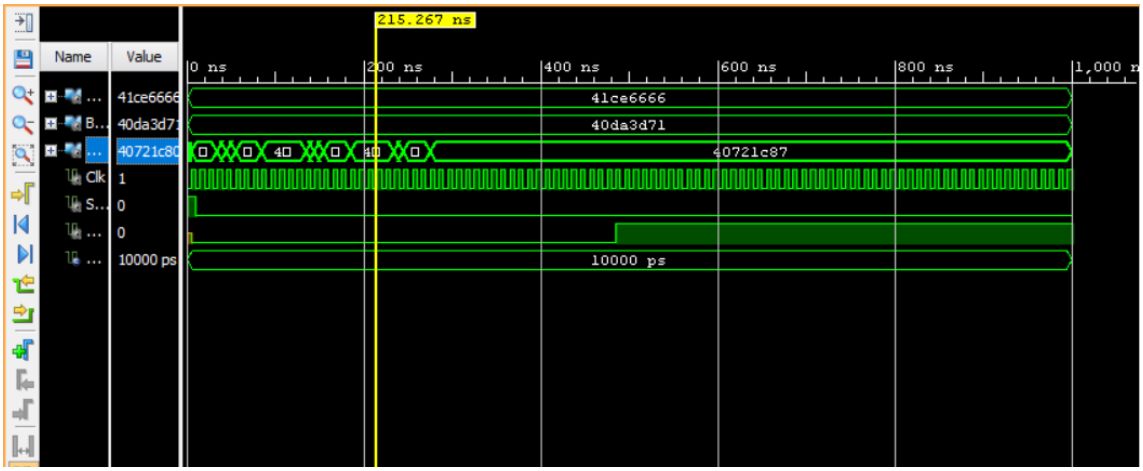

```
# run 1000ns
Note: Test case 1 passed: Correct result
```

```
A <= x"c1940000"; -- -18.5
B <= x"40e9999a"; -- 7.3
Start <= '1';
wait for 10 ns;
Start <= '0';
wait until Finish = '1';
if Result = x"c0223119" then -- Expected result: -2.53424658
  report "Test case 2 passed: Correct result"
    severity note;
else
  report "Test case 2 failed: Incorrect result"
    severity error;
end if;
```



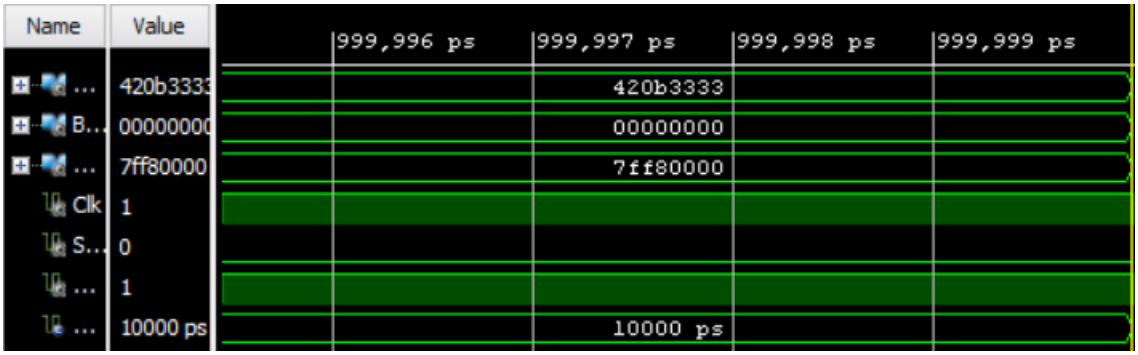

```
# run 1000ns
Note: Test case 2 passed: Correct result
```

```
--   Test case 3 - Large positive division
  A <= x"41ce6666"; -- 25.8
  B <= x"40da3d71"; -- 6.82
  Start <= '1';
  wait for 10 ns;
  Start <= '0';
  wait until Finish = '1';
  if Result = x"40721c87" then -- Expected result: 3.782991202346041
    report "Test case 3 passed: Correct result"
      severity note;
  else
    report "Test case 3 failed: Incorrect result"
      severity error;
  end if;
```




```
# run 1000ns
Note: Test case 3 passed: Correct result
```

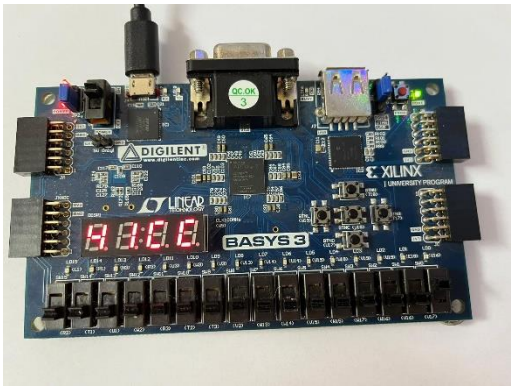
```
A <= x"420b3333"; -- 34.8
B <= x"00000000"; -- 0 (division by zero)
Start <= '1';
wait for 10 ns;
Start <= '0';
wait until Finish = '1'; -- Add a timeout condition
if Result = X"7ff80000" then -- Expected result: Positive infinity (division by zero exception)
  report "Test case 4 passed: Correct result (Division by zero)"
    severity note;
else
  report "Test case 4 failed: Incorrect result (Division by zero)"
    severity error;
end if;
```

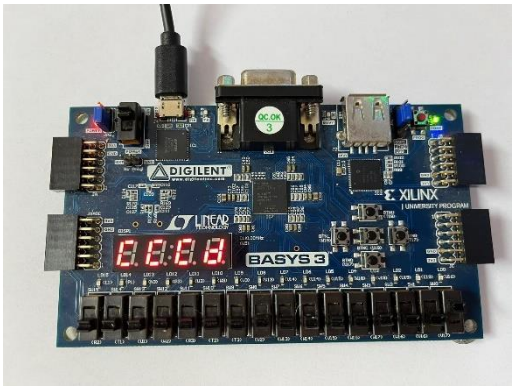



```
# run 1000ns
Note: Test case 4 passed: Correct result (Division by zero)
```

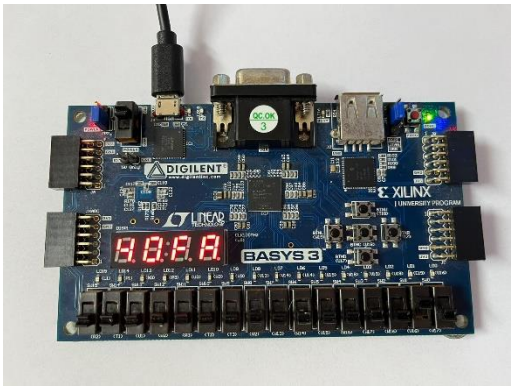
Proiectul a fost pus pe placă și numerele sunt afișate pe baza primelor trei switch-uri, prima jumătate a primului număr apare la ridicarea switch-ului sw(0), iar prin apăsarea butonului BTNR, apare a doua jumătate a numărului. La fel funcționează și pentru al doilea număr (sw(1), BTNL), dar și pentru rezulta(sw(2),BTNC).

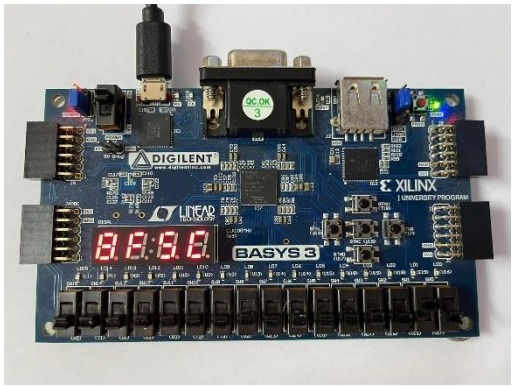
--Primul număr



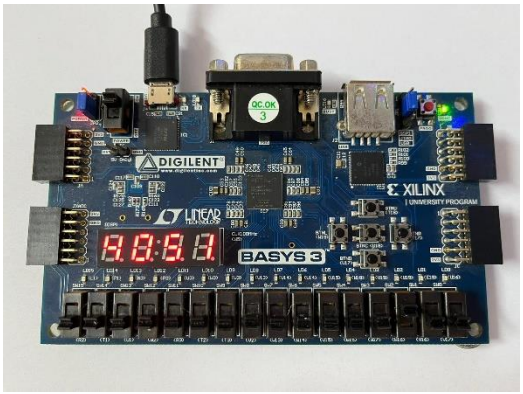


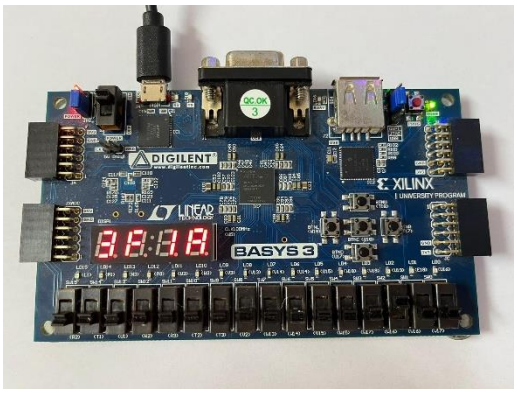
--Al doilea număr





--Rezultatul





6. Concluzii

În cadrul acestui proiect, am implementat operația de împărțire a numerelor în virgulă mobilă. Am colectat informații relevante despre Standardul IEEE 754 și am studiat modul în care această operație este efectuată, exersând în același timp abilitățile de programare în limbajul VHDL.

Ca perspectivă pentru dezvoltări ulterioare, proiectul ar putea fi integrat într-o aplicație mai complexă, unde operația de împărțire este esențială. De exemplu, acest proiect ar putea fi parte a unui calculator în care utilizatorul introduce numere în format binar și efectuează operații de împărțire cu acestea.

7. Bibliografie

- https://www.cs.umd.edu/~meesh/411/CA-online/chapter/floating-point-arithmetic-unit/index.html
- https://users.utcluj.ro/~baruch/book_ac/AC-Reprez-VM.pdf
- https://www.rfwireless-world.com/Tutorials/floating-point-tutorial.html
- https://en.wikipedia.org/wiki/Single-precision_floating-point_format
- https://d1wqtxts1xzle7.cloudfront.net/68678306/D1108031824-libre.pdf?1628576407=&response-content-disposition=inline%3B+filename%3DDesign_of_Single_Precision_Floating_Poin.pdf&Expires=1697928791&Signature=ZvJGqCI2Jv95K-QL8NS5EplwMQYcM3Mr0PZ~Be~~lj6tro5IFezQMVpFsqkPgHu5Jri6TVeYEXjM5VzhKi7s6xDyhoCw6Qogf0Pwh91OiVnSL-o6vIVMF~UfV6adscqQlqP-e1ZQN2551qT8hdYn~aNsB~6hY77xSP7o7CPeNLG14kRy4E1s~MC7o61FyaGqBSzWiuSPoqYzt9pwF3Nx1zKXEfJaFNXclBJfxq6dEuRFR5JRYa~Wi~31IBwxXprXitZMHfMX-sKY0~DfOF4f5IoUynmcbQfVylmLqnTR~fl1gL8mYknGmjVkcl9zje~FFPZH~pMRM3N9G-veY3h7BA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
- https://d1wqtxts1xzle7.cloudfront.net/35954995/IJIEEB-V6-N1-1-libre.pdf?1418751482=&response-content-disposition=inline%3B+filename%3DDesign_of_FPGA_based_32_bit_Floating_Poi.pdf&Expires=1697929016&Signature=ZHX541t-jNxE-D4p6KhXyTii~BAOqxmYiPvZKdqyWBdebxj7Y9dI2rO0EjNn7E7MOgHOVOW0b55Ny~pNUgloT2yo107turPX0W6YRNCWO20UD~y2kcF12V9toHzlMmQypmuANiuzcJ8xvczI44kysWrTwWVohdO7QTuPxgQkvk~XnVP6m8Q1dAw71jmo6J~MFN0d6ax05VsCvwZFhAXNGzz5C2odb4fKgyZ2rEbdhc2ChZcIOJrQD7q~Isb4pSNQFuq

ihqiMv1bBIdCK5ylgQEWgBNJG9j2FEmL5kPGf00nxxO3ag~vyJpYlLy~11vfhrPQAkZl1yCXImmX64Ww4GQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

- IEEE-754 Floating Point Converter (h-schmidt.net)
- L6_7 - Aritm-Secventiala.pdf