CIRCUITE DE IMPARTIRE ZECIMALA

GHICA MADALINA

CUPRINS

6. Testare si validare
7. Concluzii
8. Bibliografie

CAPITOLUL 1

Introducere

1.1 Context

Un circuit de împărțire zecimală trebuie să primească două numere de intrare sub formă de numere BCD (Binary Coded Decimal) și să efectueze operația de împărțire zecimală asupra lor. Rezultatul ar trebui să fie și el un număr BCD. Pentru a face acest lucru,voi implementa un circuit care simulează divizarea zecimală și aduce rezultatul, similar cu metoda pe hârtie a împărțirii zecimale.

1.2 Solutia propusa

Soluția propusă constă în dezvoltarea unui circuit de împărțire zecimală, folosind metoda scaderilor repetate, utilizând FPGA și limbajul VHDL. Proiectul are ca scop implementarea operației de împărțire zecimală pentru numere BCD de 8/4 cifre.Soluția propusă ar consta în următoarele etape:

a. Proiectarea arhitecturii circuitului: Crearea unei arhitecturi funcționale pentru circuit, care să includă blocuri pentru conversia datelor BCD, divizorul zecimal, comparatorul și blocurile de adunare BCD.

b. Implementarea în limbajul VHDL: Dezvoltarea codului VHDL pentru fiecare bloc funcțional al circuitului, inclusiv logica de control.

c. Simulare și verificare: Utilizarea simulatorului VHDL pentru a verifica corectitudinea operației circuitului și a identifica eventuale erori sau probleme.

d. Sinteza pentru FPGA: Sinteza codului VHDL pentru a genera configurația FPGA și pentru a programa FPGA-ul.

e. Testare hardware: Testarea circuitului pe FPGA, asigurându-se că acesta îndeplinește cerințele proiectului și că rezultatele sunt corecte.

1.3 Plan de proiect

1.3.1 Studiu Bibliografic

Colectarea informațiilor despre proiectarea circuitului de împărțire zecimală, inclusiv a caracteristicilor sale principale.

1.3.2 Analiza

Selecția unui set de instrucțiuni pentru a fi implementate și disponibile pentru execuție în cadrul circuitului de împărțire zecimală.Asigurarea furnizării numărului minim de instrucțiuni necesare și includerea cel puțin unei instrucțiuni de salt.Documentarea fiecărei instrucțiuni selectate.

1.3.3 Design si implementare

Unitate de Conversie BCD

Divizor Zecimal

Comparator BCD

Adunare BCD

Registre pentru stocarea datelor

Controler logic

Multiplexoare și Demultiplexoare

Afișaj

1.3.4 Codul programului

Scrierea unui program pentru a demonstra corectitudinea designului și încorporarea acestuia în proiect.

1.3.5 Testare

Testarea individuală a fiecărei componente și rezolvarea oricăror probleme care pot apărea. Conectarea tuturor componentelor și testarea întregului circuit de împărțire zecimală prin rularea programului scris anterior.

CAPITOLUL 2

STUDIU BIBLIOGRAFIC

Împărțirea este o operație aritmetică prin care se determină de câte ori un număr poate fi cuprins în altul.

Primul operand este numit deimpartit (D), al doilea se numeste impartitor(I), iar rezultatele se numesc cat (C) si rest(R), iar deimpartitul se calculeaza astfel :

D=C*I+R , avand conditia obligatorie R<I

Impartirea este de doua feluri: impartire zecimala si binara.

Algoritmul pt impartirea zecimala este urmatorul : se alege o cifra si se scade produsul dintre aceasta cifra si impartitorul din restul partial; daca rezultatul este decat impatitorul, cifra a fost aleasa corect, in caz contrar, se alege o alta cifra si scaderea se repeta; in fiecare pas se obtine o cifra a catului.

Impartirea binara, pe de alta parte, consta in scaderi repetate ale impartitorului I din restul partial R, iar scaderile se efectueaza numai dac I<=R ->cifra catului este 1; in caz contrat cifra catului este 0.

Împărţirea zecimală este mai complexă decât împărţirea binară, deoarece cifrele câtului pot lua valori între 0 şi 9. Aceasta presupune efectuarea în fiecare etapă a unui număr variabil de adunări sau scăderi ale împărţitorului. Metodele de împărţire binară pot fi aplicate şi pentru împărţirea zecimală, deoarece nu există deosebiri de principiu între acestea. Considerând că numerele sunt reprezentate în MS, se pot utiliza următoarele metode de împărţire zecimală:

- Metoda refacerii restului parţial;
- Metoda fără refacerea restului parţial;
- Metoda celor nouă multipli ai împărţitorului;
- Metoda înjumătăţirii împărţitorului;
- Metoda Gilman.

Metoda refacerii restului parţial

Această metodă este similară cu cea utilizată pentru împărţirea binară, efectuându-se prin scăderi repetate ale împărţitorului din restul parţial. În fiecare etapă, se obţine o cifră a câtului care este valoarea cea mai mare din cele zece posibile pentru care restul parţial este încă pozitiv. Atunci când restul parţial devine negativ, se adună împărţitorul pentru a se reface restul parţial. Pentru determinarea cifrelor câtului, blocul de comandă conţine un numărător, care este iniţializat cu 0 la începutul fiecărei etape, fiind incrementat la fiecare scădere a împărţitorului din restul parţial. Scăderea împărţitorului continuă până când restul parţial devine negativ, moment în care se reface restul parţial. Conţinutul decrementat al numărătorului reprezintă cifra câtului corespunzătoare etapei curente. Dacă numărătorul se incrementează numai atunci când restul parţial devine negativ, nu mai este necesară decrementarea acestuia. Înaintea începerii operaţiei, se testează dacă apare o depăşire şi dacă rezultatul operaţiei este zero, în mod similar cu împărţirea binară. Considerăm un dispozitiv de împărţire zecimală pentru numere reprezentate în MS, cu structura similară cu cea a unui dispozitiv de împărţire binară care utilizează metoda refacerii restului parţial. Dispozitivul conţine grupurile de registre AZ, QZ, BZ, cu funcţii similare registrelor A, Q, B ale dispozitivului de împărţire binară, un sumator-scăzător zecimal elementar şi un bloc de comandă. Acest bloc conţine un numărător N care este decrementat în fiecare etapă a operaţiei şi un numărător N1 care se utilizează pentru numărarea operaţiilor de adunare sau de scădere efectuate pentru calculul unei cifre a câtului.

Exemplul 3.4

Considerăm $X = 684$, $Y = 28$. Execuția operației de împărțire este prezentată în Tabelul 3.3.

Tabelul 3.3. Execuția operației de împărțire 684:28 prin metoda refacerii restului parțial.

Pas	AZ	QZ	BZ	N	N_1	Operații
0	0 06	84	28	2	0	Inițializare
1	0 68 -	40	28	1	0	Deplasare stânga AZ_QZ
	28 0 40 -	40	28	1	1	Scădere BZ Incrementare N_1
	28 0 12 -	40	28	1	2	Scădere BZ Incrementare N_1
	28 9 84 +	40	28	1	3	Scădere BZ Incrementare N_1
	28 0 12	42	28	1	2	Adunare BZ Decrementare N_1, $QZ_0 = N_1$
2	1 24 -	20	28	0	0	Deplasare stânga AZ_QZ
	28 0 96 -	20	28	0	1	Scădere BZ Incrementare N_1
	28 0 68 -	20	28	0	2	Scădere BZ Incrementare N_1
	28 0 40 -	20	28	0	3	Scădere BZ Incrementare N_1
	28 0 12 -	20	28	0	4	Scădere BZ Incrementare N_1
	28 9 84 +	20	28	0	5	Scădere BZ Incrementare N_1
	28 0 12	24	28	0	4	Adunare BZ Decrementare N_1, $QZ_0 = N_1$

Câtul este 24, iar restul este 12.

Metoda fără refacerea restului parţial

Această metodă elimină adunarea împărţitorului la restul parţial, atunci când acesta devine ne_gativ. Algoritmul metodei fără refacerea restului parţial este următorul: 1. Se deplasează la stânga registrul combinat care conţine restul parţial şi deîmpărţitul. Se efectu_ează scăderi repetate ale împărţitorului din restul parţial, până când diferenţa devine negativă. Dacă s-au efectuat m1 scăderi, cifra câtului este m1-1. 2. Se deplasează la stânga registrul combinat care conţine restul parţial şi deîmpărţitul. Se efectu_ează adunări repetate ale împărţitorului la restul parţial, până când suma devine pozitivă. Dacă s-au efectuat m2 adunări, cifra câtului este 10-m2. 3. Se continuă cu etapele 1 şi 2, până când se obţin toate cifrele câtului. 4. Dacă după ultima etapă restul este negativ, se reface restul prin adunarea împărţitorului. Structura dispozitivului care implementează această metodă este similară cu cea a unui dispo_zitiv de împărţire zecimală care implementează metoda refacerii restului parţial.

Exemplul 3.5

Considerăm din nou $X = 684$, $Y = 28$. Execuția operației de împărțire este prezentată în Tabelul 3.4.

Tabelul 3.4. Execuția operației de împărțire 684:28 prin metoda fără refacerea restului parțial.

Pas	AZ	QZ	BZ	N	N_1	Operații
0	0 06	84	28	2	0	Inițializare
1	0 68 -	40	28	1	0	Deplasare stânga AZ_QZ
	28 0 40 -	40	28	1	1	Scădere BZ Incrementare N_1
	28 0 12 -	40	28	1	2	Scădere BZ Incrementare N_1
	28 9 84	42	28	1	3	Scădere BZ Incrementare N_1, $QZ_0 = N_1$-1
2	8 44 +	20	28	0	0	Deplasare stânga AZ_QZ
	28 8 72 +	20	28	0	1	Adunare BZ Incrementare N_1
	28 9 00 +	20	28	0	2	Adunare BZ Incrementare N_1
	28 9 28 +	20	28	0	3	Adunare BZ Incrementare N_1
	28 9 56 +	20	28	0	4	Adunare BZ Incrementare N_1
	28 9 84 +	20	28	0	5	Adunare BZ Incrementare N_1
	28 0 12	24	28	0	6	Adunare BZ Incrementare N_1, $QZ_0 = 10\text{-}N_1$

Câtul este 24, iar restul este 12.

Cu toate că metoda fără refacerea restului parţial este mai eficientă decât cea cu refacerea restului parţial, în cazul numerelor zecimale creşterea eficienţei este mai redusă comparativ cu cazul numerelor binare. În cazul metodei refacerii restului parţial, pentru obţinerea cifrei 0 a câtului sunt necesare două operaţii de scădere sau de adunare, pentru obţinerea cifrei 1 sunt necesare trei operaţii, şi în general, pentru obţinerea cifrei x sunt necesare x+2 operaţii. Considerând că frecvenţa statistică de utilizare a diferitelor cifre este aceeaşi, numărul mediu de operaţii necesare pentru obţinerea unei cifre a câtului este:

$$\frac{2+3+4+5+6+7+8+9+10+11}{10} = \frac{(2+11)\cdot 10}{2\cdot 10} = 6{,}5$$

În cazul metodei fără refacerea restului parţial, se câştigă o operaţie la obţinerea fiecărei cifre a câtului, numărul mediu de operaţii necesare fiind:

$$\frac{1+2+3+4+5+6+7+8+9+10}{10} = \frac{(1+10)\cdot 10}{2\cdot 10} = 5{,}5$$

Numărul de operaţii necesare se reduce deci cu mai puţin de 20% pentru fiecare cifră obţinută, faţă de o reducere de aproximativ 50% în cazul împărţirii binare.

Metoda celor nouă multipli ai împărţitorului

Metoda celor nouă multipli ai împărţitorului are ca avantaj reducerea timpului de execuţie a operaţiei de împărţire. Dispozitivul care implementează această metodă este asemănător cu cel de înmulţire care implementează metoda celor nouă multipli ai deînmulţitului, conţinând 9 grupe de registre care se încarcă cu multiplii împărţitorului. Sumatorul zecimal elementar este înlocuit cu un sumator- scăzător zecimal elementar, sumatorul fiind folosit în etapa de generare a multiplilor împărţitorului, iar scăzătorul în etapa de împărţire propriu-zisă. Blocul de comandă conţine un comparator, acesta având rolul de a selecta multiplul cel mai mare al împărţitorului, care prin scădere din restul parţial, determină un rezultat pozitiv. În fiecare etapă a operaţiei, se deplasează la stânga registrul combinat care conţine restul parţial şi deîmpărţitul, iar restul parţial este comparat cu multiplii împărţitorului. Se efectuează apoi scăderea multiplului selectat al împărţitorului din restul parţial. După efectuarea scăderii, factorul de multiplicare corespunzător multiplului selectat este memorat drept cifră a câtului. Dacă nu se consideră timpul nece_sar generării multiplilor împărţitorului, pentru fiecare cifră obţinută este necesară o operaţie de scădere. Deoarece păstrarea tuturor celor 9 multipli necesită un echipament costisitor, se utilizează circuite care generează numai anumiţi multipli ai împărţitorului. În particular, se pot utiliza aceleaşi circu_ite combinaţionale pentru generarea multiplilor de 2 şi de 5. În acest caz, blocul de comandă trebuie să determine în fiecare etapă cifra corespunzătoare a câtului, iar apoi multiplul care trebuie scăzut din res_tul parţial. Succesiunea operaţiilor necesare pentru determinarea cifrelor câtului este prezentată în Figura 3.3, unde s-a notat cu R restul parţial, iar cu Y împărţitorul.

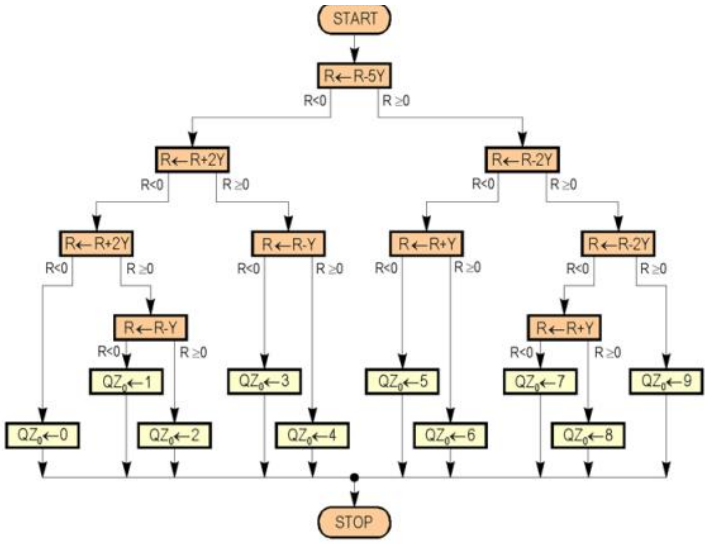


Figura 3.3. Obținerea cifrelor câtului prin metoda celor nouă multipli ai împărțitorului, atunci când sunt disponibili multiplii de 2 și de 5 ai împărțitorului

Iniţial, se scade multiplul de 5 al împărţitorului din restul parţial. Dacă rezultatul este negativ, cifra câtului este cuprinsă între 0 şi 4. În acest caz, se adună multiplul de 2 al împărţitorului la restul parţial. Dacă rezultatul este negativ, cifra câtului este cuprinsă între 0 şi 2, şi se adună din nou multiplul de 2 al împărţitorului la restul parţial. Dacă rezultatul este tot negativ, cifra câtului este 0, în caz contrar această cifră fiind 1 sau 2. În ultimul caz, se scade împărţitorul din restul parţial; dacă rezultatul este negativ, cifra câtului este 1, iar dacă rezultatul este pozitiv sau zero, cifra câtului este 2. În mod similar se poate determina cifra câtului în cazul în care după scăderea multiplului de 5 al împărţitorului din restul parţial rezultatul este pozitiv sau zero. O cifră a câtului se poate determina prin maximum 4 operaţii. Numărul de operaţii necesare pentru determinarea cifrelor câtului este indicat în Tabelul 3.5.

Tabelul 3.5. Numărul de operații necesare pentru determinarea cifrelor câtului atunci când sunt disponibili multiplii de 2 și de 5 ai împărțitorului.

Cifra	0	1	2	3	4	5	6	7	8	9
Număr de operații	3	4	4	3	3	3	3	4	4	3

Media statistică a numărului de operaţii necesare pentru determinarea unei cifre a câtului este:

$$\frac{3+4+4+3+3+3+3+4+4+3}{10}=\frac{34}{10}=3{,}4$$

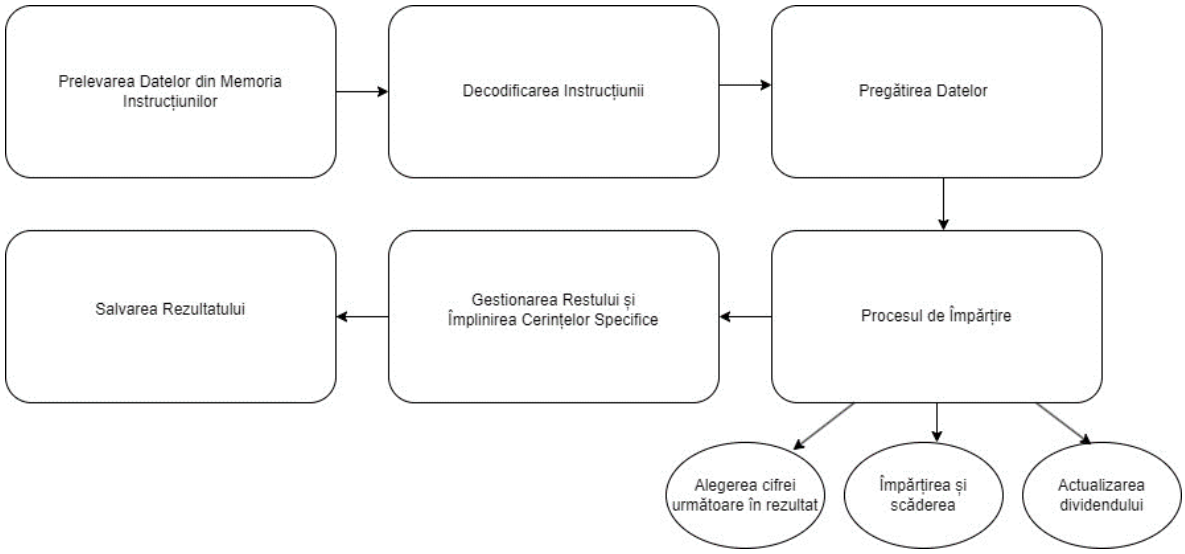
Deci, în medie sunt necesare 3,4 operaţii, comparativ cu 6,5 operaţii necesare în cazul metodei refacerii restului parţial şi 5,5 operaţii în cazul metodei fără refacerea restului parţial.

CAPITOLUL 3

ANALIZA

3.1 Functionarea circuitului de impartire zecimala

Un circuit de împărțire zecimală este conceput pentru a efectua operații de împărțire cu numere zecimale într-un sistem informatic. Procesul de funcționare a unui astfel de circuit implică următoarele etape principale:



1 .**Prelevarea Datelor din Memoria Instrucțiunilor:** Circuitul de împărțire zecimală începe prin preluarea instrucțiunilor și datelor necesare din memoria programului. Instrucțiunile includ operația de împărțire, adresele de memorie pentru datele de intrare (divizorul și dividendul), adresa pentru rezultat, precum și alte informații relevante.

2**. Decodificarea Instrucțiunii:** Instrucțiunea de împărțire este decodificată pentru a determina tipul de operație de împărțire (de exemplu, împărțire cu virgulă mobilă sau împărțire între numere întregi), precum și dimensiunea numerelor zecimale implicate (numărul de cifre zecimale).

3. **Pregătirea Datelor:** Datele de intrare, divizorul și dividendul, sunt preluate din memoria programului sau din locațiile de memorie specificate în instrucțiune. Aceste

date sunt apoi pregătite pentru a fi folosite în operația de împărțire. De exemplu, ele pot fi încărcate în registre speciale.

4. **Procesul de Împărțire:** Împărțirea în sine este efectuată într-un mod secvențial, asemănător cu împărțirea manuală. Circuitul de împărțire zecimală execută următoarele operații:

→ Alegerea cifrei următoare în rezultat: Se selectează următoarea cifră din rezultat și se realizează o estimare inițială a câtului.

→ Împărțirea și scăderea: Se împarte cifra curentă a dividendului la divizor și se obține o cifră a câtului. Apoi, se scade produsul dintre cifra curentă a divizorului și cifra curentă a câtului din dividend.

→ Actualizarea dividendului: După scăderea efectuată, se actualizează dividendul cu cifra rămasă, și procesul se repetă pentru următoarea cifră în rezultat până când toate cifrele sunt calculate.

5. **Gestionarea Restului și Împlinirea Cerințelor Specifice:** În cazul împărțirii cu virgulă mobilă, circuitul trebuie să se ocupe și de partea fracționară a numerelor. Acesta trebuie să gestioneze și resturile și să se asigure că rezultatul este format corect.

6. **Salvarea Rezultatului:** Rezultatul final al împărțirii este salvat în memoria sistemului sau în locația specificată în instrucțiune. Acest rezultat poate fi utilizat ulterior în alte operații sau instrucțiuni.

3.2 Lista de intructiuni pentru impartirea zecimala

Pentru a implementa un circuit de impartire zecimala, va trebui sa definim setul de intructiuni si operatii necesare pentru aceasta operatie.

LOAD_A: Această instrucțiune încarcă un număr BCD (Binary Coded Decimal) de 8 cifre într-un registru specific pentru operandul A

LOAD_B: Această instrucțiune încarcă un număr BCD de 8 cifre într-un alt registru specific pentru operandul B

DIVIDE: Această instrucțiune declanșează operația de împărțire zecimală între cele două numere încărcate anterior în registrele A și B. Rezultatul este stocat în alt registru

GET_QUOTIENT: Această instrucțiune transferă rezultatul împărțirii (partea întreagă a rezultatului) într-un registru specific

GET_REMAINDER: Această instrucțiune transferă restul împărțirii (partea zecimală a rezultatului) în alt registru.

CHECK_DONE: Această instrucțiune verifică dacă operația de împărțire este finalizată și setează un indicator (flag) în funcție de rezultat

RESET: Această instrucțiune resetează toate registrele și starea circuitului la starea inițială

NOP (No Operation): Oinstrucțiune "NOP" poate fi inclusă pentru a avea un ciclu fără operație, util pentru sincronizare sau temporizare.

JUMP/BRANCH: Instrucțiuni de salt/branch pot fi incluse pentru a permite programului săsărite la alte locații de memorie în funcție de condiții sau pentru a executasecvențe de instrucțiuni specifice.

3.3 Campul de conditie

- Flag de Zero (Z): Acest bit poate fi setat când rezultatul împărțirii este zero sau atunci când restul este zero. Acest lucru este util pentru a verifica dacă împărțirea a produs un rezultat exact.
- Flag de Semn (S): Acest bit poate indica semnul rezultatului. De exemplu, este setat când rezultatul este negativ sau atunci când restul este negativ.
- Flag de Depășire (O): Acest bit indică dacă rezultatul împărțirii depășește intervalul de valori pe care îl poate reprezenta. Acesta poate fi util pentru a preveni depășirea sau pentru a gestiona rezultatele în cazuri limită.
- Flag de Validitate (V): Acest bit indică dacă rezultatul împărțirii este valid și utilizabil. De exemplu, poate fi setat când operația de împărțire a fost finalizată cu succes.
- Flag de Erori (E): Acest bit poate indica apariția unor erori în timpul operației de împărțire, cum ar fi împărțirea la zero sau alte situații anormale.
- Flag de Depășire a Numărului de Cifre Zecimale (D): Acest bit poate fi folosit pentru a indica depășirea numărului de cifre zecimale disponibile în rezultat.

- Flag de Stare Internă (I): Acest bit poate fi folosit pentru a indica o stare internă a circuitului, care poate fi utilă pentru diagnosticarea și depanarea circuitului.

3.4 Aplicatii si scenarii de utilizare

Împărțirea zecimală este o operație matematică importantă în numeroase domenii și are o gamă largă de aplicații și scenarii de utilizare, cum ar fi urmatoarele:

- Finanțe și Contabilitate
- Comerțul Electronic și Plățile Online
- Calculatoare și Procesoare
- Inginerie Electronică și Proiectarea Circuitelor
- Telecomunicații
- Medicină și Diagnostic Medical
- Invățare Automată și Procesarea Imaginilor
- Modelarea Economică și Simulări
- Analiza Statistică

3.5 Aprofundarea algoritmului ales

Într-un circuit de împărțire zecimală, algoritmul de refacere a restului parțial este important pentru a asigura că operația de împărțire zecimală funcționează corect. Acest algoritm implică gestionarea și actualizarea resturilor generate în timpul operației de împărțire și poate varia în funcție de complexitatea circuitului și de arhitectura specifică.

<u>Algoritm de Refacere a Restului Parțial</u>

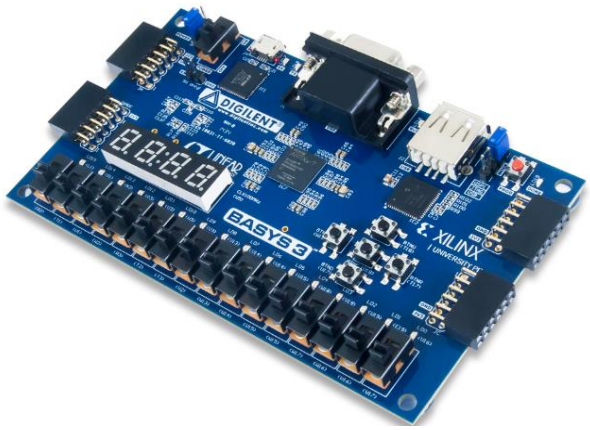
- Inițializare: La începutul operației de împărțire, restul parțial (inițial 0) este inițializat
- Luarea Cifrelor: Pe măsură ce procesul de împărțire avansează, se iau cifrele din dividend (numitorul împărțirii) pe rând, de la cifra cea mai semnificativă la cea mai puțin semnificativă. Aceste cifre sunt adăugate la restul parțial existent

- Verificarea pentru Împărțire: După adăugarea fiecărei cifre, se verifică dacă restul parțial acumulat este mai mare sau egal cu divizorul. Dacă este adevărat, se face o împărțire și se actualizează atât restul cât și rezultatul. În caz contrar, se trece la următoarea cifră
- Actualizarea Restului: După fiecare împărțire parțială, restul este actualizat prin scăderea divizorului din restul anterior
- Continuarea: Se continuă acest proces pentru toate cifrele din dividend
- Rezultatul Final și Restul Final: La finalul procesului, rezultatul final al împărțirii este stocat într-un registru corespunzător, iar restul final este, de asemenea, disponibil pentru utilizare.

CAPITOLUL 4

DESIGN

4.1 Prezentare generala



Pentru implementare proiectului am ales sa folosesc un BASYS3. Placa de dezvoltare BASYS3 este un dispozitiv FPGA versatil care poate fi utilizat pentru a implementa și

testa diverse circuite digitale, inclusiv operații de împărțire zecimală. In urmatorul box voi prezenta cateva detalii importante despre placa BASYS3:

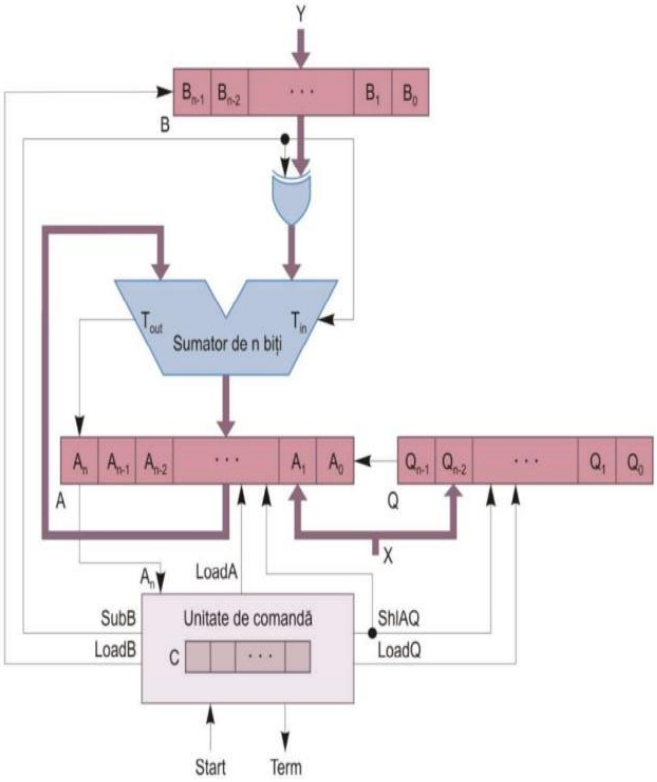
Arhitectură FPGA: Artix-7 are resurse precum blocuri de memorie RAM, blocuri de logică configurabilă (CLB - Configurable Logic Blocks), multiplexoare, interconexiuni, ceasuri programabile și alte componente care permit implementarea unor proiecte digitale complexe.
Interfețe de E/S: Placa Basys 3 dispune de o varietate de interfețe de intrare/ieșire, incluzând butoane, LED-uri, afișaje cu șapte segmente, porturi USB, conectori pentru comunicare digitală, precum și conectori pentru alimentare și alte dispozitive periferice.
Conexiuni și Extensibilitate: Placa este proiectată pentru a putea fi extinsă și conectată la alte plăci de dezvoltare, adăugând funcționalități suplimentare sau interacționând cu alte dispozitive și senzori.
Software de Dezvoltare: Pentru programarea și dezvoltarea aplicațiilor pentru placa Basys 3, se folosesc suite de instrumente software precum Vivado Design Suite sau Xilinx ISE, care permit scrierea și sinteza codului VHDL/Verilog, simulări și generarea fișierelor bitstream pentru încărcarea pe placa FPGA.

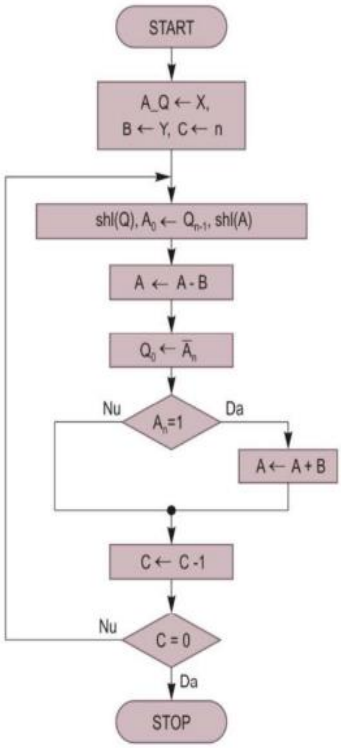
4.2 Cum vom folosi placuta pentru implementare

- cercul rosu din stanga reprezinta deimpartitul
- cercul albastru din dreapta reprezinta impartitorul
- cercul mov este pentru butonul care atunci cand va fi apasat va efectua impartirea
- cercul negru este pentru butonul care atunci cand va fi apasat va afisa pe ssd catul in binar
- cercul verde este pentru butonul care atunci cand va fi apasat va afisa pe ssd restul in binar

4.3 Scheme bloc si componente

4.3.1 Schema bloc a circuitului de împărțire, care implementează metoda refacerii restului parțial + organigrama



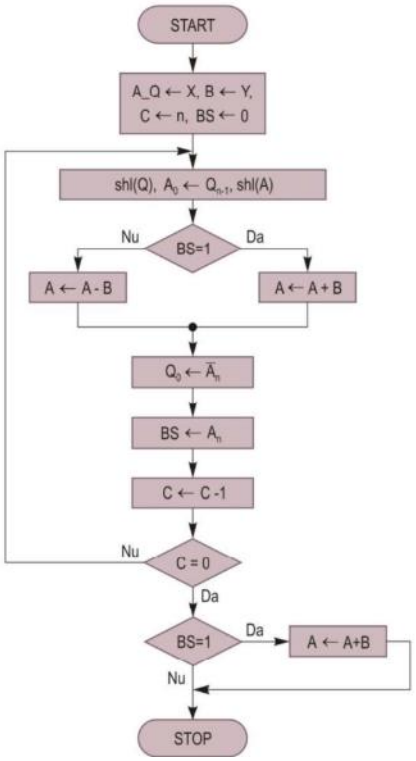


1) Schema bloc a circuitului de împărțire, care implementează metoda refacerii restului parțial pentru numerele fără semn, este descrisă astfel: Registrul B și Q sunt configurate cu n poziții fiecare. Registrul A are n+1 poziții pentru a păstra semnul în timpul operației de scădere, care are loc în fiecare etapă a împărțirii. Atât Registrul A, cât și Registrul Q pot acționa ca registre de deplasare la stânga. Intrarea serială a Registrului A permite introducerea celui mai semnificativ bit al Registrului Q în poziția A. Sumatorul folosit are n biți. Conținutul Registrului B poate fi adăugat sau scăzut din conținutul Registrului acumulator A. Scăderea, în care Registrul B conține divizorul și Registrul A conține restul parțial, se realizează prin adăugarea complementului în doi al Registrului B la Registrul A. Obținerea complementului în doi al Registrului B se realizează folosind un grup de porți logice SAU EXCLUSIV și semnalul SubB, similar cu operația executată în metoda Booth pentru înmulțire. Dacă semnalul SubB este '1', se realizează o operație de scădere, în caz contrar se efectuează o operație de adunare.

2) În organigrama prezentată nu sunt incluse teste care ar trebui efectuate după etapa de inițializare. Este esențial să se efectueze teste pentru a verifica mai multe condiții cruciale. În primul rând, trebuie verificat dacă împărțitorul este zero. Dacă acest lucru se confirmă, operația de împărțire trebuie oprită, indicând o eroare

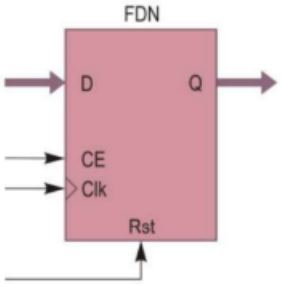
asociată împărțirii la zero. De asemenea, este important să se testeze pentru depășirea capacitatii. Aceasta apare atunci când conținutul registrului A (partea mai semnificativă a deîmpărțitului) este mai mare sau egal cu conținutul registrului B (împărțitorul). În plus, se poate efectua un test pentru a verifica dacă deîmpărțitul este mai mic decât împărțitorul; în caz afirmativ, câtul obținut va fi zero, iar restul va fi egal cu deîmpărțitul.

4.3.2 Organigrama circuitului de împărțire, care implementează metoda fara refacerea restului parțial



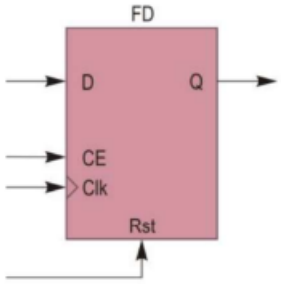
În etapa de inițializare, similară cu metoda de refacere a restului parțial, bistabilul suplimentar BS este inițializat la 0. În fiecare etapă a operației, registrele A și Q sunt deplasate la stânga cu o poziție, urmată de fie o scădere a împărțitorului din restul parțial, fie o adunare a împărțitorului la restul parțial, în funcție de starea bistabilului BS. Dacă restul parțial este pozitiv (A_{n} = 0), cifra câtului este 1. În etapa următoare se efectuează o scădere, moment în care bistabilul BS este resetat la 0.

4.3.3 Registru de n biti cu resetare sincrona



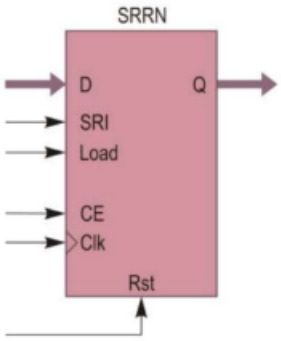
Registrul FDN utilizează semnalul Clk ca intrare de ceas și vectorul D cu n biți ca intrare de date. Vectorul O cu n biți reprezintă ieșirea de date a registrului. Atunci când semnalul Rst este la nivel logic 1, registrul este resetat sincronizat la frontul crescător al semnalului de ceas, toate ieșirile sale fiind aduse la valoarea logică 0. În absența semnalului Rst, dacă semnalul CE (Clock Enable) este la nivel logic 1, se realizează încărcarea paralelă a registrului cu datele furnizate la intrare.

4.3.4 Bistabil cu resetare sincrona



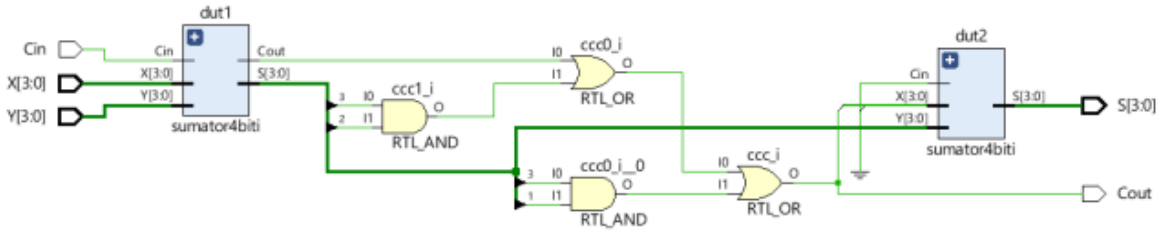
Intrarea de ceas a bistabilului FD este Clk, intrarea de date este D, iar iesirea de date este Q. Functionarea acestui bistabil este similara cu cea a registrului FDN, cu deosebire ca intrarea de date si iesirea de date sunt de cate un bit in locul unor vectori.

4.3.5 Registru de deplasare la dreapta de n biti cu resetare sincrona



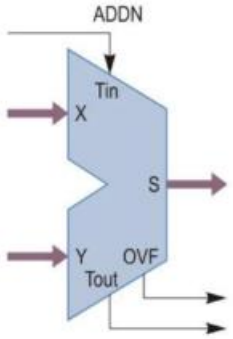
Intrarea de ceas pentru registrul de deplasare SRRN (conform figurii 5.10) este sincronizată cu semnalul Clk. Acesta primește datele în mod paralel pe vectorul D, iar datele în mod serial pe intrarea SRI (Shift Right Input). Ieșirea de date este reprezentată de vectorul O de n biți. Atunci când semnalul Rst este la nivel logic 1, registrul este resetat sincron la frontul ascendent al semnalului de ceas, ieșirile de date fiind resetate la nivel logic 0. În situația în care semnalul Load este la nivel logic 1, registrul este încărcat în mod paralel cu datele aplicate la intrare. În absența semnalului Load la nivel logic 1 și a semnalului CE (Clock Enable) la nivel logic 1, se realizează operația de deplasare a registrului cu o poziție spre dreapta. Concomitent, datele de pe intrarea serială SRI sunt încărcate în poziția "On-t". Similar, se poate defini un registru de deplasare la stânga.

4.3.6 Sumator zecimal elementar



Sumatorul zecimal integrează două sumatoare pe patru biți și o logică specifică pentru a preveni depășirea cifrelor de la 0 la 9. Atunci când se depășește limita, în cel de-al doilea sumator se adaugă valoarea binară 6, transformând numărul din baza 16 în baza 10. Pentru a genera bitul de depășire, se folosește o combinație de porți logice. Ieșirea de transport a primei etape este apoi utilizată ca intrare pentru al doilea sumator. Practic, adunarea valorii 6 se realizează doar dacă numărul depășește 9. Mai jos este prezentată structura RTL cu intrările, ieșirile și componentele interne.

4.3.7 Sumator de n biti



Intrările sumatorului ADDN sunt valorile X și Y, ambele de lungime n biți, care urmează să fie adunate, alături de intrarea de transport Tin. Ieșirile sumatorului constau din suma S formată din n biți, transportul de ieșire Tout și semnalul de depășire OVF (Overflow). Semnalul OVF este relevant doar în cazul adunării numerelor cu semn.

CAPITOLUL 5

Implementare

5.1 Prezentare generala

In acest capitol se vor prezenta diagramele RTL pentru fiecarei componenta, dar si codul aflat in entitate.

Pasii pe care ii vom urma in efectuarea circuitului:

- Inițierea operației de împărțire: Atunci când semnalul de start este activat (start = '1'), începe procesul de împărțire.

- Sumatorul pe 16 biți efectuează operații de adunare sau scădere: Folosim acest sumator pentru a efectua operațiile de adunare și scădere necesare în procesul de împărțire.

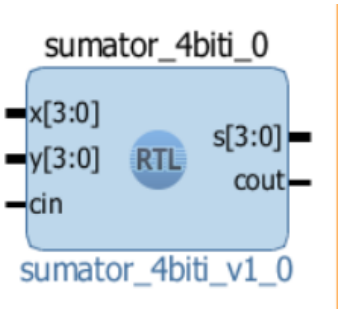
- Registrele stochează și manipulează valorile temporare: Registrele de 8 biți stochează temporar anumite valori care sunt utilizate în timpul operației de împărțire.
- Registrul de deplasare la stânga efectuează operații de deplasare: Acest registru este folosit pentru a deplasa valorile într-o direcție specifică (în acest caz, la stânga), ceea ce este necesar în procesul de împărțire.
- Logica de control coordonează și sincronizează operațiile: Există o unitate de control (UC) care coordonează operațiile între aceste componente, generând semnale de control (cum ar fi semnale de încărcare, semnale de deplasare etc.) pentru a asigura că fiecare componentă funcționează corect și în sincronizare.

Mai multe detalii despre fiecare componenta se gasesc in capitolul anterior.

5.2 Componente

5.2.1 Sumator de 4 biti

FOLOSIT IN COMPONENTA SUMATORULUI ZECIMAL.

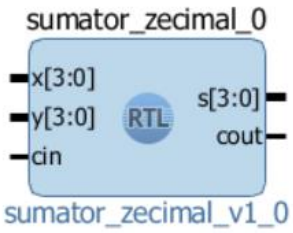


Entitate sumator
pe 4 biti

```
entity sumator_4biti is
Port ( x: in std_logic_vector(3 downto 0);
       y: instd_logic_vector(3 downto 0);
       cin: in std_logic;
       s: outstd_logic_vector(3 downto 0);
       cout: out std_logic);
end sumator_4biti;
```

5.2.2 Sumator zecimal

FOLOSIT IN COMPONENTA SUMATORULUI PE 16 BITI.

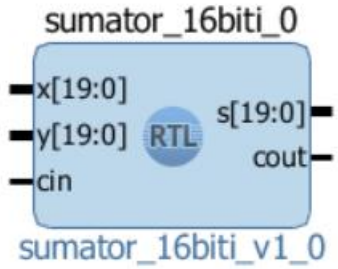


Entitate sumator zecimal

```
entity sumator_zecimal is
 Port ( x: in std_logic_vector(3 downto 0);
    y: instd_logic_vector(3 downto 0);
    cin: in std_logic;
    s: outstd_logic_vector(3 downto 0);
    cout: out std_logic);
end sumator_zecimal;
```

5.2.3 Sumator pe 16 biti

ACEST SUMATOR ESTE ELEMENTAR DEOARECE ESTE FOLOSIT LA EFECTUAREA OPERATIILOR DE ADUNARE SI SCADERE NECESARE IN PROCESUL DE IMPARTIRE.

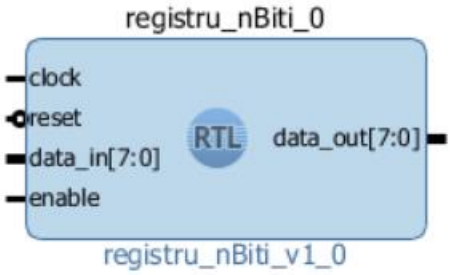


Entitate sumator
pe 16 biti

```
entity sumator_16biti is
Port ( x : in STD_LOGIC_VECTOR (19
downto 0);
y : in STD_LOGIC_VECTOR (19 downto 0);
cin : in STD_LOGIC;
s : out STD_LOGIC_VECTOR (19 downto 0);
cout : out STD_LOGIC);
end sumator_16biti;
```

5.2.4 Registru de n biti

Acest registrru il folosesc pentru a salva impartitorul in etapele de impartire

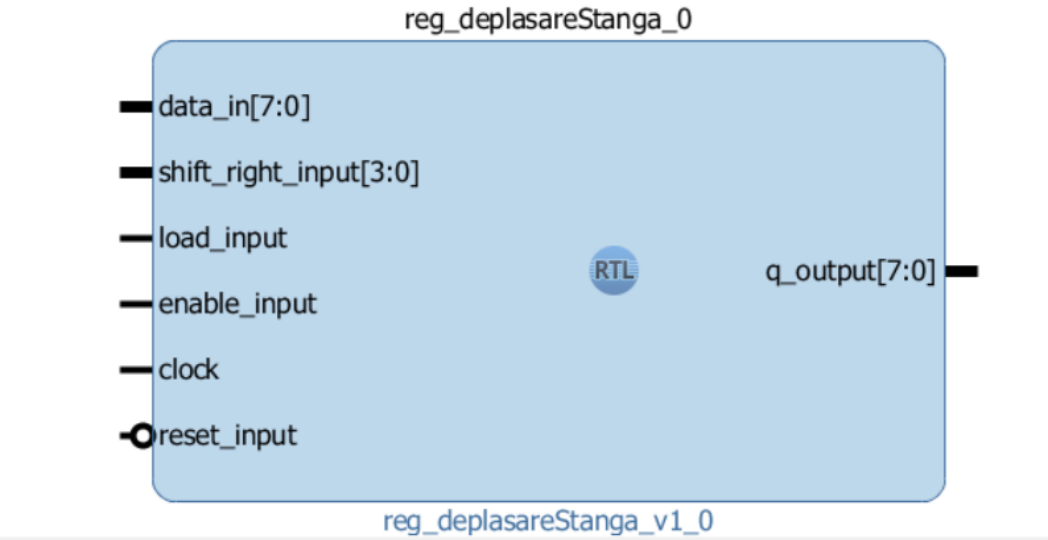


Entitate registru pe n biti

```
entity registru_nBiti is
Port (
    clock : in std_logic;
    reset : in std_logic;
    data_in : in std_logic_vector(7 downto
0);
    enable : in std_logic;
    data_out : out std_logic_vector(7
downto 0);
);
end registru_nBiti;
```

5.2.5 Registru de deplasare la stanga

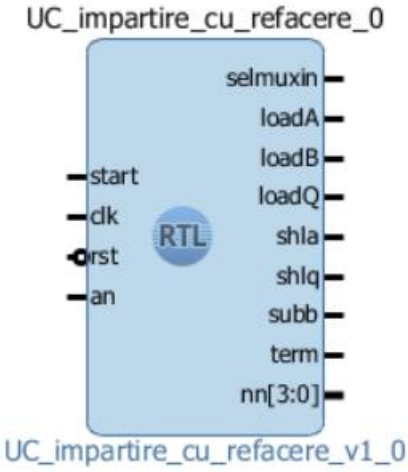
Acest registru este utilizat pentru a stoca catul si restul operatiei de impartire.



Entitate registru de deplasare la stanga

```
entity reg_deplasareStanga is
 Port (
data_in: in std_logic_vector(7 downto 0);
shift_right_input: in std_logic_vector(3
downto 0);
load_input: in std_logic;
enable_input: in std_logic;
clock: in std_logic;
reset_input: in std_logic;
q_output: out std_logic_vector(7 downto
0)
);
end reg_deplasareStanga;
```

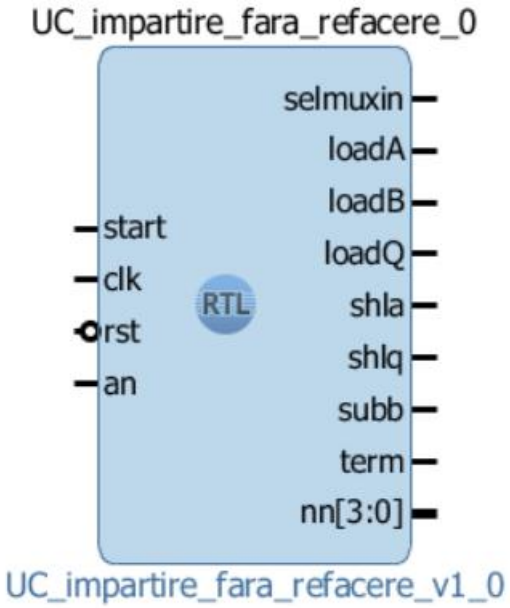
5.2.6 UC->impartire cu refacerea restului

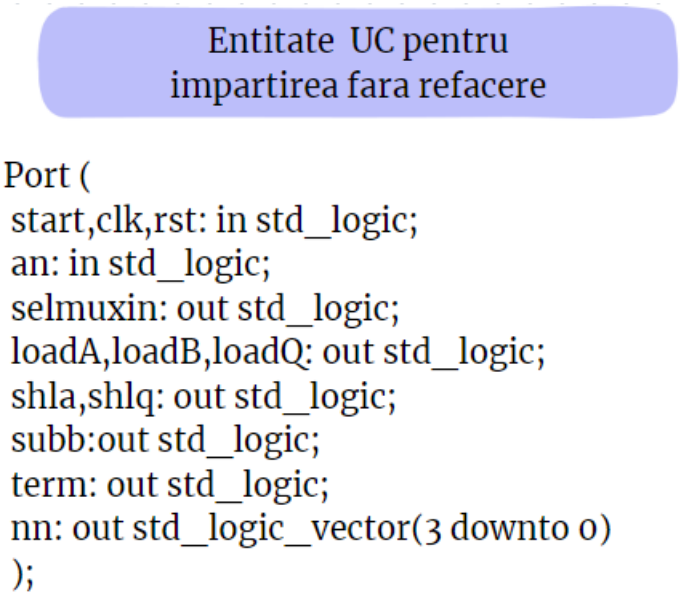


Entitate UC pentru
impartirea cu refacere

```
entity UC_impartire_cu_refacere is
 Port (
 start,clk,rst: in std_logic;
 an: in std_logic;
 selmuxin: out std_logic;
 loadA,loadB,loadQ: out std_logic;
 shla,shlq: out std_logic;
 subb:out std_logic;
 term: out std_logic;
 nn: out std_logic_vector(3 downto 0)
 );
```

5.2.7 UC->impartire fara refacerea restului





5.3 Impartirea cu refacere a restului partial

5.3.1 Descrierea modului de interconectare a componentelor

1. Sumatorul pe 16 biți:
   - Primește ca intrări numerele A și B și poate executa operații de adunare și scădere.
   - Are ieșiri pentru rezultatul sumei și semnale de transport pentru gestionarea transportului între biți în operații de adunare.
2. Registrul de deplasare la stânga:
   - Primește datele din sumatorul pe 16 biți și poate deplasa valorile la stânga cu un număr specific de poziții (în cazul împărțirii).
   - Ieșirea sa conectată la intrarea sa este controlată de semnalele generate de logica de control.
3. Registrul de stocare (de exemplu, un registru de 8 biți):
   - Folosit pentru a stoca temporar valori pe parcursul operației de împărțire.

- Ieșirea sa este legată la intrarea sumatorului și la intrarea registrelor de deplasare la stânga.

4. Logica de control (Unitatea de Control - UC):
   - Coordonarea semnalelor și a operațiilor între sumator, registre și alte componente.
   - Generează semnalele de control pentru a indica când să se efectueze operații specifice precum încărcarea, deplasarea sau oprirea.

În operația de împărțire cu refacere a restului parțial, semnalele de control sunt esențiale pentru sincronizarea și coordonarea corectă a operațiilor între aceste componente. Sumatorul pe 16 biți și registrele de stocare sunt utilizate pentru realizarea calculelor necesare în operația de împărțire, iar registrul de deplasare la stânga este utilizat pentru ajustarea și manipularea datelor în timpul operației de împărțire.

entitatea pentru impartirea cu refacerea restului

```
entity impartirea_cu_refacere is
generic(n : natural:= 20);
Port (
clk: in std_logic;
rst : in STD_LOGIC;
start : in STD_LOGIC;
x : in STD_LOGIC_VECTOR(n-1 downto 0);
y : in STD_LOGIC_VECTOR(n-1 downto 0);
a : out STD_LOGIC_VECTOR(n-1 downto 0);
q : out STD_LOGIC_VECTOR(n-1 downto 0);
Term : out STD_LOGIC);
end impartirea_cu_refacere;
```

5.4 Impartirea fara refacere a restului partial

5.4.1 Descrierea modului de interconectare a componentelor

Pentru operația de împărțire fără refacere a restului parțial, conexiunile dintre aceste componente ar putea arăta astfel:

- Sumatorul pe 16 biți primește datele A și B pentru operații de adunare și scădere.
- Rezultatul din sumator este direcționat spre registrul de 8 biți pentru stocarea temporară a rezultatului parțial.

- Registrul de deplasare la stânga primește rezultatul din registrul de 8 biți pentru a-l deplasa la stânga, conform algoritmului de împărțire.
- Semnalele de control generate de unitatea de control coordonează operațiile între aceste componente. De exemplu, aceasta poate activa/dezactiva registrele sau sumatorul în funcție de starea operației de împărțire.
- Rezultatul final este stocat în registrul de 8 biți și poate fi folosit pentru a obține rezultatul împărțirii și restul parțial pentru operația respectivă.

Aceste conexiuni sunt realizate astfel încât fiecare componentă să își execute operațiile în mod corespunzător și în sincronizare cu celelalte, astfel încât operația globală de împărțire să fie realizată corect.

entitatea pentru impartirea fara refacerea restului

```
entity impartireaFaraRefacere is
generic(n : natural:= 20);
 Port (
 clk: in std_logic;
 rst : in STD_LOGIC;
 start : in STD_LOGIC;
 x : in STD_LOGIC_VECTOR(n-1 downto 0);
 y : in STD_LOGIC_VECTOR(n-1 downto 0);
 a : out STD_LOGIC_VECTOR(n-1 downto 0);
 q : out STD_LOGIC_VECTOR(n-1 downto 0);
 Term : out STD_LOGIC);
end impartireaFaraRefacere;
```

CAPITOLUL 6

TESTARE SI VALIDARE

6.1 Prezentare generala

Pentru a simplifica procesul de identificare a problemelor, am verificat fiecare element individual. Am acoperit diverse scenarii pentru a confirma corectitudinea funcționării acestora.

În următoarea parte, voi expune codul specific pentru fiecare banc de teste, alături de o matrice ce detaliază intrările, ieșirile așteptate și cele reale. În plus, voi include și o reprezentare grafică a formei de undă pentru fiecare test efectuat.

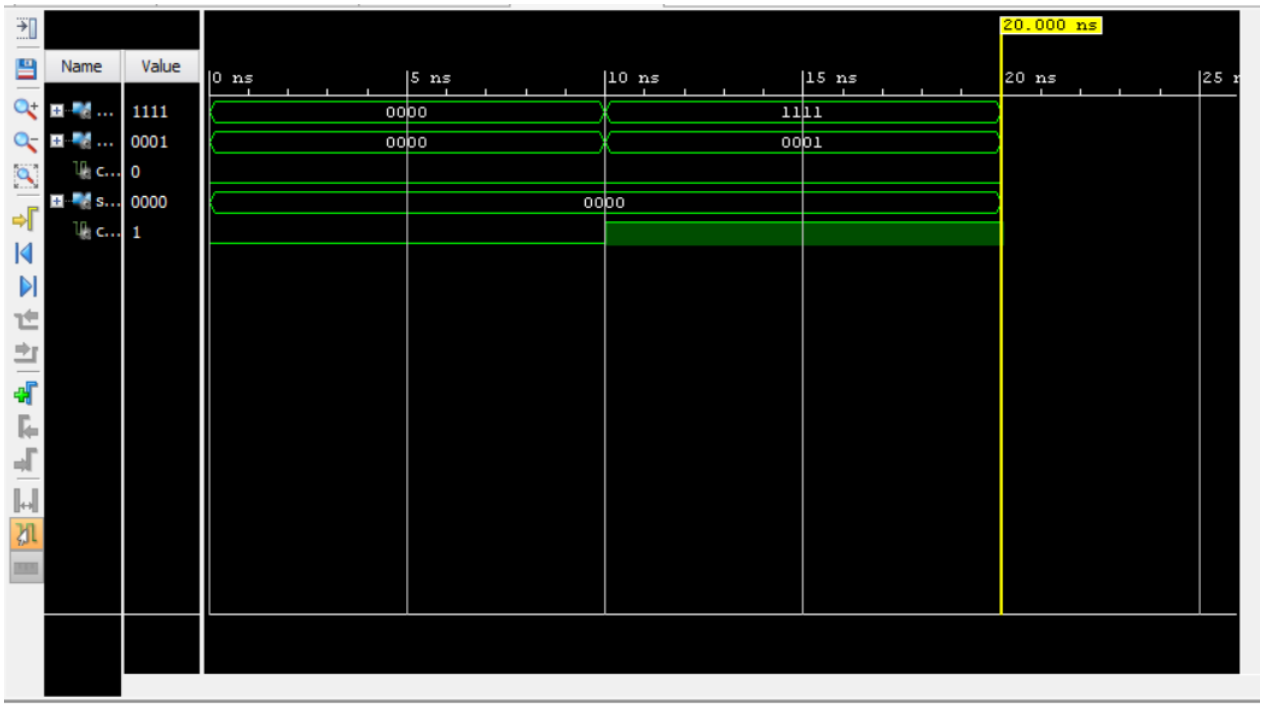
6.2 Simulare

6.2.1 Sumator pe 4 biti

tb pentru sumator pe 4 biti

```
process
 begin
  -- Test 1
  x_sig <= "0000";
  y_sig <= "0000";
  cin_sig <= '0';
  wait for 10 ns; -- Așteptare pentru propagare
  assert (s_result = "0000" and cout_result = '0') report "Test
1 failed!" severity failure;

  -- Test 2
  x_sig <= "1111";
  y_sig <= "0001";
  cin_sig <= '0';
  wait for 10 ns; -- Așteptare pentru propagare
  assert (s_result = "0000" and cout_result = '1') report "Test
2 failed!" severity failure;
  wait;
 end process;
```



1) Testul 1:

x = "0000"

y = "0000"

cin = '0'

Rezultat așteptat (s și cout) = "0000" și '0' (deci suma și carry-out-ul trebuie să fie zero)

2) Testul 2:

x = "1111"

y = "0001"

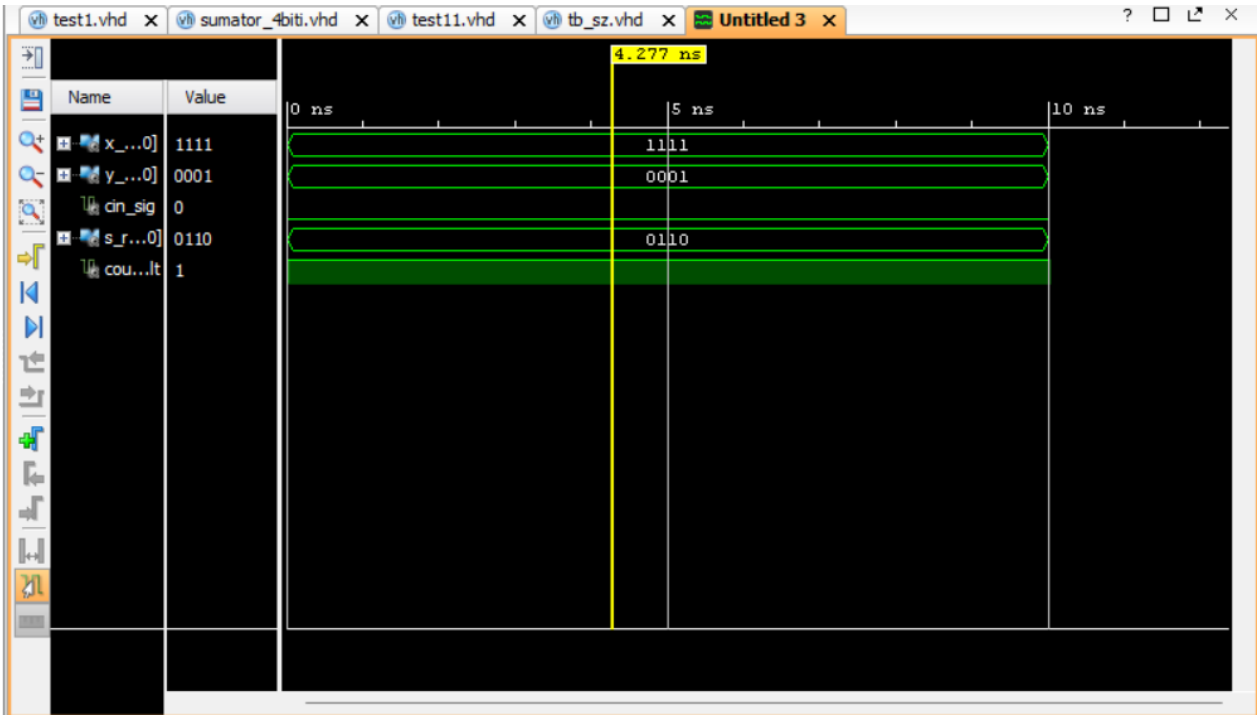
cin = '0'

Rezultat așteptat (s și cout) = "0000" și '1' (suma ar trebui să fie "0000", iar carry-out-ul sa fie '1')

6.2.2 Sumator zecimal

tb pentru sumator zecimal

```
process
begin
  x <= "1111"; -- Valoarea 2 în zecimal
  y <= "0001"; -- Valoarea 1 în zecimal
  cin <= '0'; -- Fără carry-in inițial
  wait for 10 ns;
  assert s = "0000" and cout = '1' report "Test 1 failed!"
severity error;
  wait;
end process;
```



Test:

x = "1111"

y = "0001"

cin = '0'

Rezultatul așteptat (s și cout) ar trebui să fie:

s (suma) = "0000"

cout (carry-out) = '1'

6.2.3 Sumator pe 16 biti

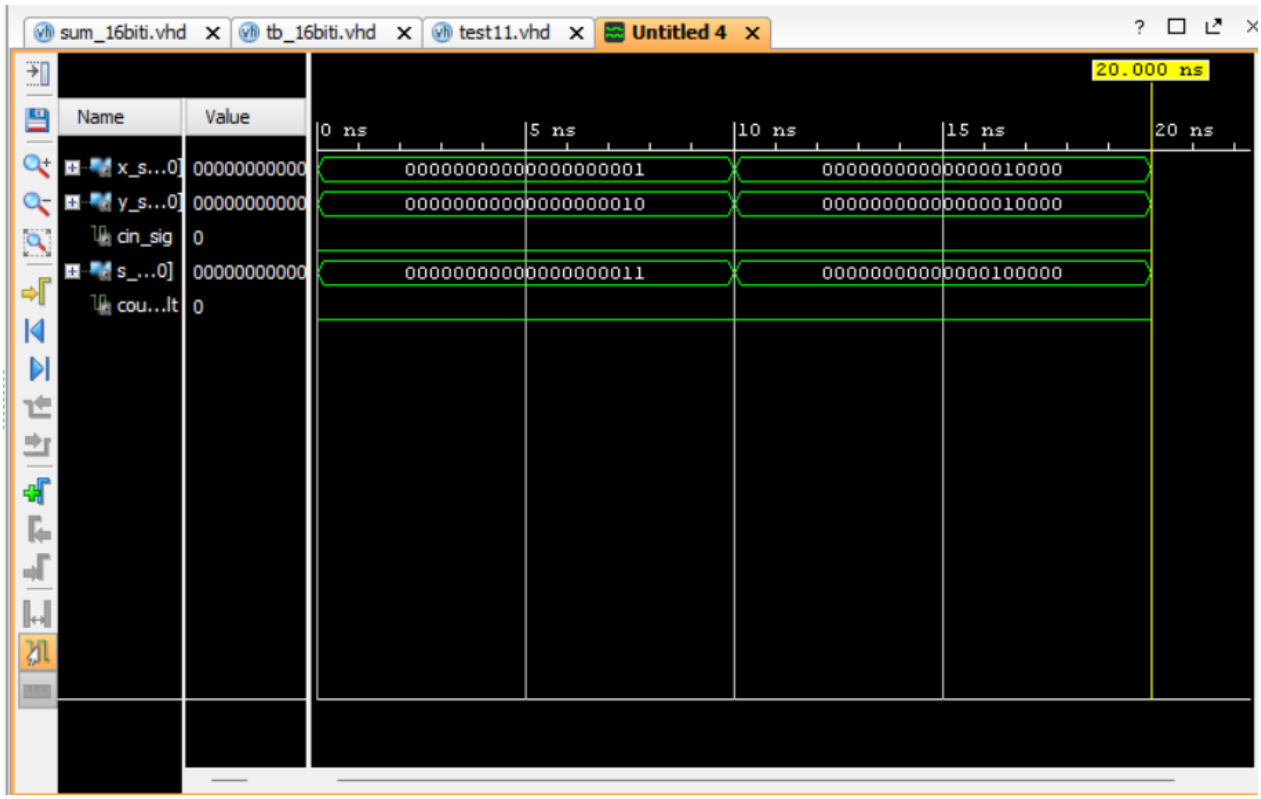
tb pentru sumator pe 16 biti

```
 process
 begin
  wait for 10 ns; -- A?teptare pentru sincronizare
 end process;

 -- Teste
 process
 begin
  -- Test 1: Adunare a două numere zecimale mici
  x_sig <= "0000000000000000001"; -- Numărul 1
  y_sig <= "0000000000000000010"; -- Numărul 2
  cin_sig <= '0';
  wait for 10 ns; -- A?teptare pentru propagare
  assert (s_result = "0000000000000000011" and
cout_result = '0') report "Test 1 failed!" severity failure;

  -- Test 2: Adunare a două numere mai mari
  x_sig <= "0000000000000010000"; -- Numărul 16
  y_sig <= "0000000000000010000"; -- Numărul 16
  cin_sig <= '0';
  wait for 10 ns; -- A?teptare pentru propagare
  assert (s_result = "0000000000000100000" and
cout_result = '0') report "Test 2 failed!" severity failure;

  wait;
 end process;
```



1) Test 1:

x = "0000000000000000001" (reprezintă numărul 1 în format zecimal)

y = "0000000000000000002" (reprezintă numărul 2 în format zecimal)

cin = '0'

Rezultat așteptat (s și cout) = "0000000000000000003" și '0' (suma ar trebui să fie 3, iar carry-out-ul să fie '0')

2) Test 2:

x = "0000000000000010000" (reprezintă numărul 16 în format zecimal)

y = "0000000000000010000" (reprezintă numărul 16 în format zecimal)

cin = '0'

Rezultat așteptat (s și cout) = "0000000000000100000" și '0' (suma ar trebui să fie 32, iar carry-out-ul să fie '0')

6.2.4 Registru de n biti

tb pentru registrul de n biti

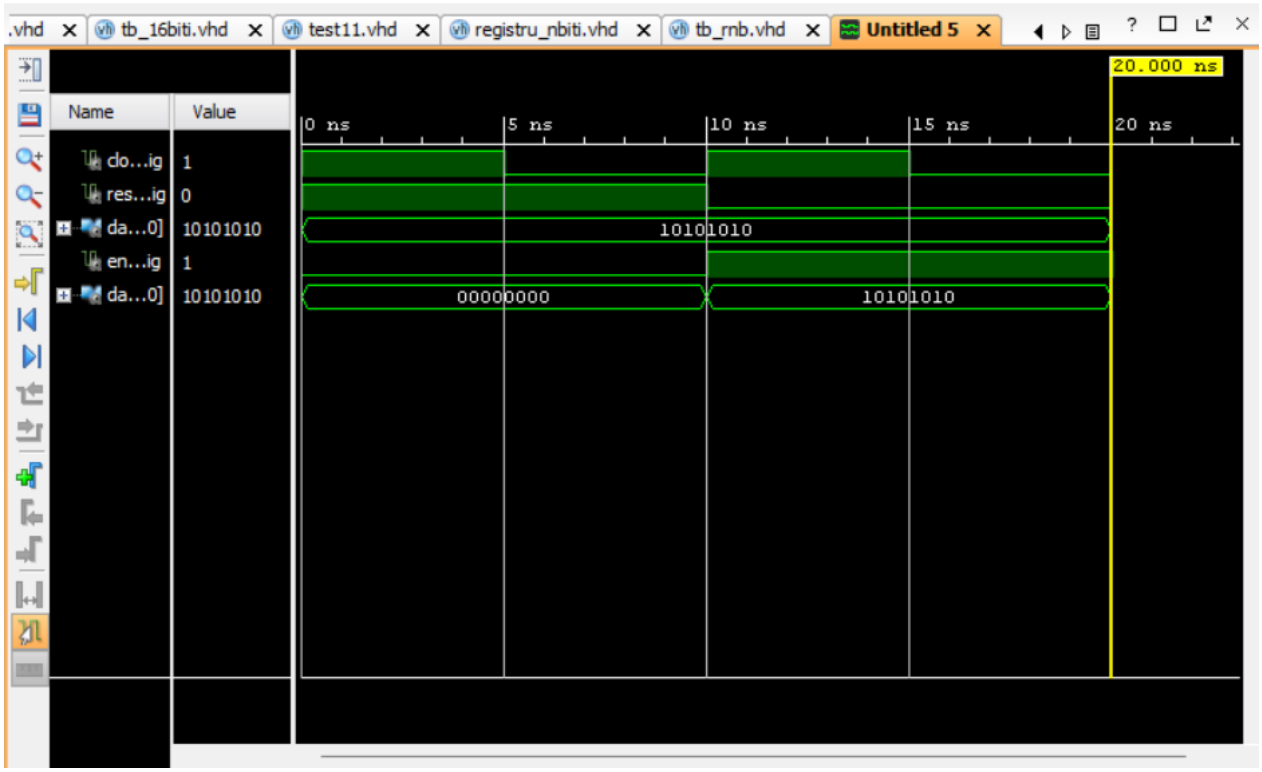
```
process
 begin
  while true loop
   clock_sig <= not clock_sig; -- Generează un semnal de ceas
alternant
    wait for 5 ns; -- A?teptare pentru a simula un ciclu de ceas
  end loop;
 end process;

 -- Testare
 process
 begin
  reset_sig <= '1'; -- Activează semnalul de resetare
  enable_sig <= '0'; -- Debifează semnalul de activare
  wait for 10 ns; -- A?teptare pentru sincronizare

  reset_sig <= '0'; -- Dezactivează semnalul de resetare
pentru a începe func?ionarea normală
  enable_sig <= '1'; -- Activează semnalul de activare
  wait for 20 ns; -- A?teptare pentru propagare


  assert (data_out_result = "10101010") report "Test failed!
Data_out is not equal to input data" severity failure;

  wait;
 end process;
```



1) Testarea funcției de reset:

reset = '1' pentru o perioadă de timp, apoi reset = '0' pentru a dezactiva resetul.

Rezultatul așteptat pentru semnalul de ieșire (data_out) ar trebui să fie valoarea implicită, adică "00000000" (zero).

2) Testarea încărcării datelor:

Setarea semnalului enable = '1' pentru a permite încărcarea datelor.

Transmiterea unei anumite valori pe linia data_in (de exemplu, "10101010").

Rezultatul așteptat pentru semnalul de ieșire (data_out) ar trebui să fie aceeași valoare ca cea transmisă prin data_in.

6.2.5 Registru de deplasare la stanga

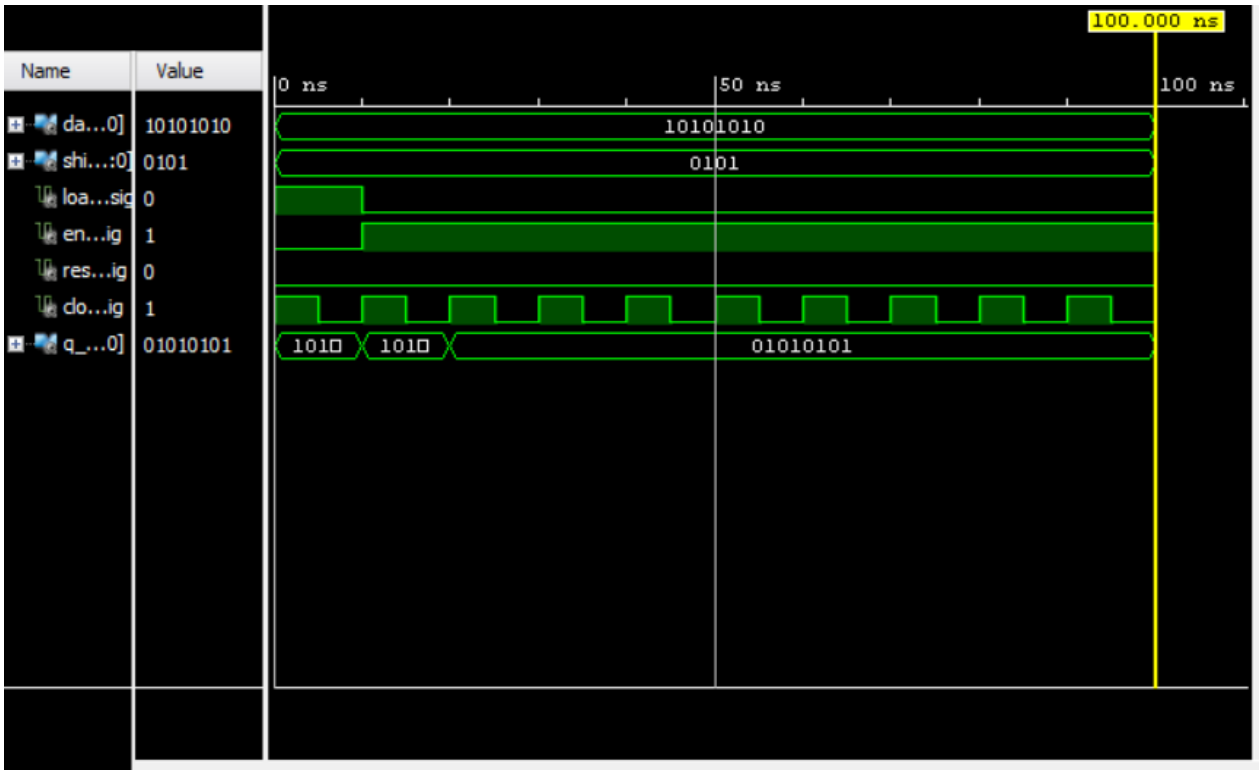
tb pentru registrul de deplasare la stanga

```
 process
begin
  while true loop
    clock_sig <= not clock_sig; -- Generează un semnal de ceas
alternant
     wait for 5 ns; -- A?teptare pentru a simula un ciclu de ceas
  end loop;
end process;

 -- Testare
 process
 begin
   load_input_sig <= '1'; -- Activează semnalul de încărcare
   enable_input_sig <= '0'; -- Debifează semnalul de activare
   reset_input_sig <= '0'; -- Debifează semnalul de resetare
   wait for 10 ns; -- A?teptare pentru sincronizare
   load_input_sig <= '0'; -- Dezactivează semnalul de
încărcare pentru a permite opera?ia normală
   enable_input_sig <= '1'; -- Activează semnalul de activare
pentru opera?ia de deplasare
   reset_input_sig <= '0'; -- Debifează semnalul de resetare
   wait for 20 ns; -- A?teptare pentru propagare

   assert (q_output_result = "01010101") report "Test failed!
Unexpected output from left shift operation" severity failure;

   wait;
 end process;
```



1) Testarea funcției de încărcare:

Setarea semnalului load_input = '1' pentru a activa încărcarea datelor în registru.

Transmiterea unei anumite valori pe linia data_in (de exemplu, "10101010").

Rezultatul așteptat pentru semnalul de ieșire (q_output) ar trebui să fie aceeași valoare ca cea transmisă prin data_in.

2) Testarea operației de deplasare la stânga:

Setarea semnalului enable_input = '1' pentru a activa operația de deplasare la stânga.

Transmiterea unei valori pe linia shift_right_input care reprezintă cantitatea și direcția de deplasare (de exemplu, "0101").

Rezultatul așteptat pentru semnalul de ieșire (q_output) ar trebui să fie rezultatul deplasării la stânga a valorii inițiale a registrului.

6.2. UC->impartire cu refacerea restului

tb pentru impartirea cu refacere

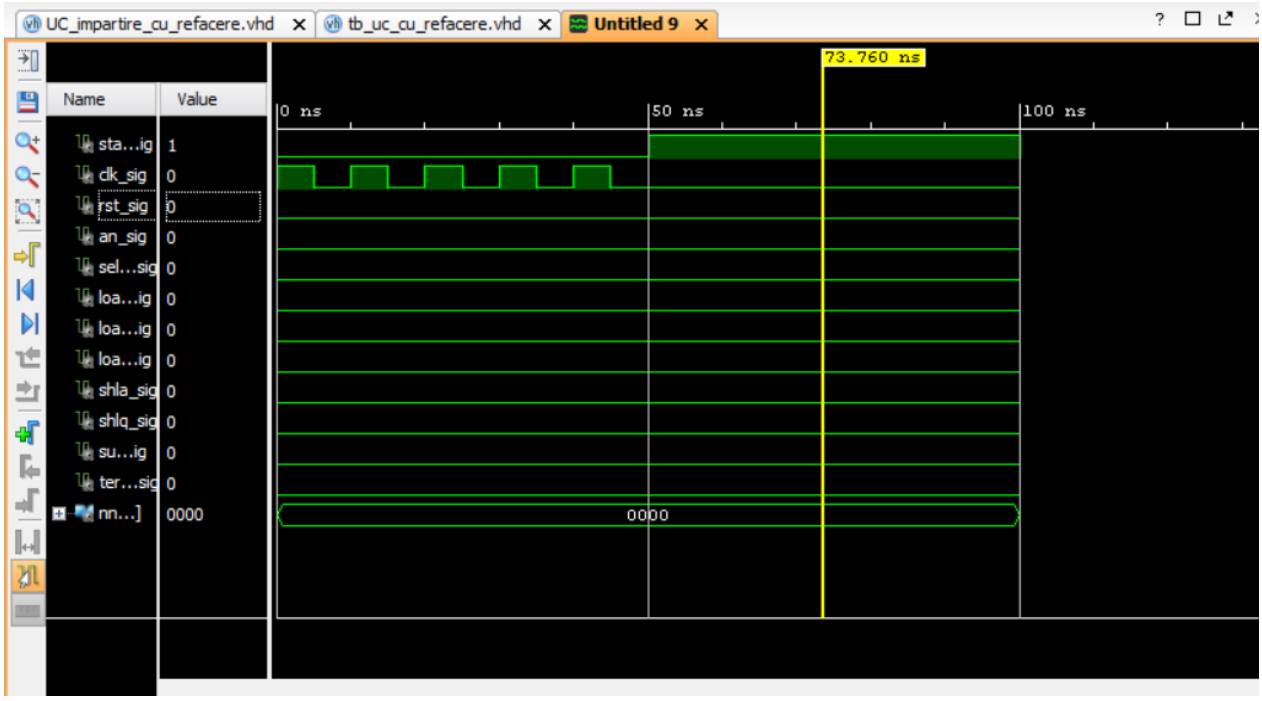
```
-- Testarea
process
begin
  -- Inițializarea semnalelor
  start_sig <= '0';
  clk_sig <= '0';
  rst_sig <= '0';
  an_sig <= '0';

  -- Ciclu de clock simulat
  for i in 1 to 10 loop
    clk_sig <= not clk_sig; -- Ciclu alternant de ceas
    wait for 5 ns; -- Așteptare pentru a simula un ciclu de ceas
  end loop;

  -- Simulare acțiuni de bază pentru test
  start_sig <= '1'; -- Activare semnal de start
  wait for 10 ns;

  -- Alte acțiuni și testare ulterioare pot fi adăugate pentru a
simula diferite scenarii de utilizare

  wait;
end process;
```



1) Starea Idle (Idle):

- Toate semnalele de control sunt dezactivate și toate ieșirile sunt la valori implicite sau zero.
- La detectarea semnalului de start, starea ar trebui să treacă la starea de initializare.

2) Starea de Inițializare (Initializare):

- Semnalele pentru încărcarea datelor (loadA, loadB, loadQ) sunt activate pentru a încărca datele în anumite registre interne.
- Se pregătește și se trece la starea de shiftareA.

3) Starea de ShiftareA (Shiftare A):

- Semnalele asociate cu shiftarea (shla, shlq) sunt activate pentru a efectua operația de shiftare.
- Se pregătește și se trece la starea de scădere.

4) Starea de Scădere (Scadere):

- Semnalul asociat operației de scădere (subb) este activat pentru a efectua operația de scădere.
- Se pregătește și se trece la starea de decizie.

5) Starea de Decizie (Decizie):

- Se iau decizii în funcție de anumite condiții (an ='1') pentru a trece în alte stări cum ar fi adunarea sau continuarea scăderii.
- Se pregătește

6.2.7 UC->impartire fara refacerea restului

tb pentru impartirea fara refacere

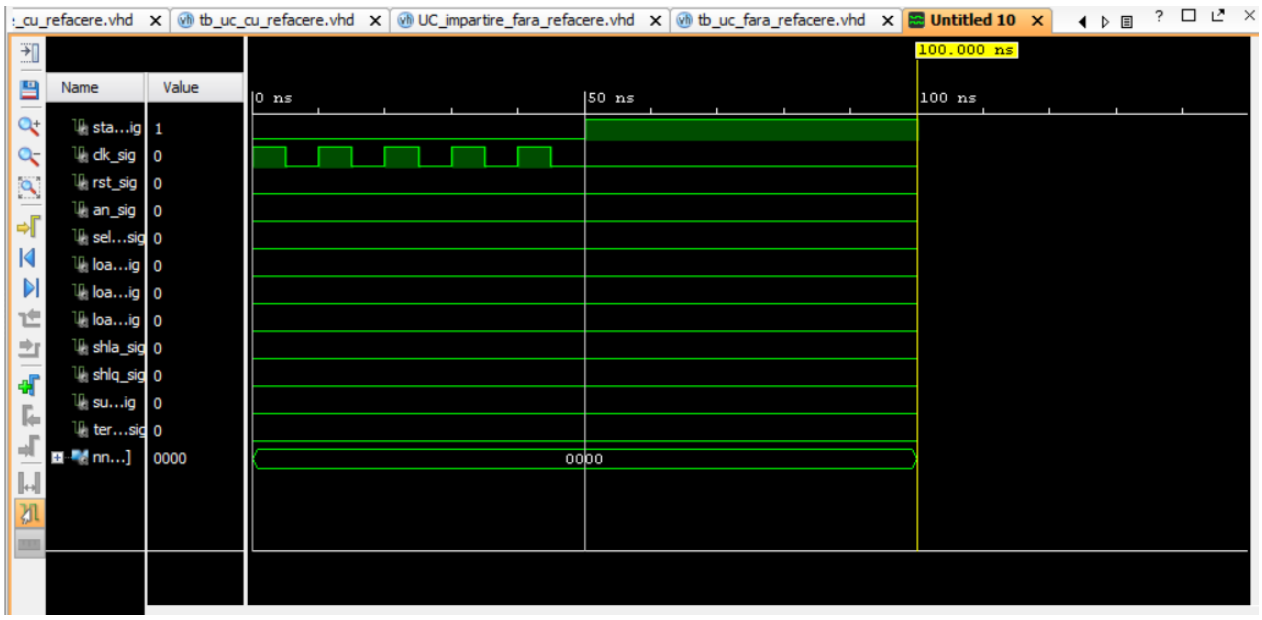
```
process
 begin
   -- Inițializarea semnalelor
   start_sig <= '0';
   clk_sig <= '0';
   rst_sig <= '0';
   an_sig <= '0';

   -- Ciclu de clock simulat
   for i in 1 to 10 loop
    clk_sig <= not clk_sig; -- Ciclu alternant
de ceas
    wait for 5 ns; -- Așteptare pentru a
simula un ciclu de ceas
   end loop;

   -- Simulare acțiuni de bază pentru test
   start_sig <= '1'; -- Activare semnal de
start
   wait for 10 ns;

   -- Alte acțiuni și testare ulterioare pot fi
adăugate pentru a simula diferite scenarii de
utilizare

   wait;
 end process;
```



6.2.8 Impartire cu refacerea restului

1) Starea Idle (Idle):
   - Toate semnalele de control sunt dezactivate.
   - Semnalele pentru încărcarea datelor (loadA, loadB, loadQ) sunt inactive.
   - Semnalele de shiftare (shla, shlq) și operaționale (subb) sunt inactive.
   - Semnalul de selectare (selmuxin) este dezactivat.
   - Semnalul 'term' indică că operația s-a încheiat și este dezactivat.

2) Starea de Inițializare (Initializare):
   - Semnalele pentru încărcarea datelor (loadA, loadB, loadQ) sunt activate pentru a încărca datele în anumite registre interne.
   - Semnalul de selectare (selmuxin) este activat.
   - Semnalele de shiftare (shla, shlq) și operaționale (subb) sunt inactive.
   - Semnalul 'term' este inactiv.

3) Starea de ShiftareA (Shiftare A):
   - Semnalele de încărcare (loadA, loadB, loadQ) sunt inactive.
   - Semnalele de shiftare (shla) sunt activate pentru a efectua operația de shiftare.
   - Semnalul de selectare (selmuxin) este dezactivat.
   - Semnalul 'term' este inactiv.

CAPITOLUL 6

CONCLUZII

În concluzie, implementarea unui circuit de împărțire zecimală pe plăcuța Basys 3 reprezintă un pas semnificativ în dezvoltarea sistemelor digitale complexe. Circuitul de împărțire zecimală este un element esențial pentru o gamă largă de aplicații, cum ar fi procesarea semnalelor, calculatoarele încorporate și aplicațiile financiare. Implementarea sa pe plăcuța Basys 3 a demonstrat funcționalități esențiale și performanțe corecte.

Principalele avantaje ale acestui circuit pe plăcuța Basys 3 include uramtoarele:

- Utilizarea resurselor FPGA pentru a realiza operații de împărțire zecimală într-un mod eficient.
- Flexibilitatea și scalabilitatea oferită de FPGA, permițând ajustarea și extinderea circuitului conform necesităților specifice.
- Implementarea acestui circuit a implicat o serie de provocări și a necesitat o înțelegere profundă a algoritmilor și a logicii împărțirii zecimale. Testarea riguroasă și verificările funcționale au jucat un rol esențial în asigurarea corectitudinii și performanței circuitului.

CAPITOLUL 7

BIBLIOGRAFIE

- Cursuri teams- cursul 5 ssc
- Laboratorul 6-7 „Circuite aritmetice secventiale”
- fisier SSC_old_lab (teams)