**Universitatea Tehnică din Cluj-Napoca**

**Facultatea de Automatică și Calculatoare**

**Departamentul de Calculatoare**

**Proiect pentru Structura Sistemelor de Calcul**

**Circuite de împărțire pentru numere întregi**

**Student: Olar Paula**

**Grupa: 30235**

**Prof. Coord.: Butiri Alexandru**

**An școlar: 2020/2021**

**Cuprins**

1. **Rezumat**
2. **Introducere**
3. **Fundamentare teoretică**
4. **Proiectare și implementare**
5. **Rezultate experimentale**
6. **Concluzii**
7. **Bibliografie**
8. **Introducere**

În cadrul acestui proiect se va realiza descrierea algoritmică și implementarea funcțională a două metode de împărțire a numerelor întregi în VHDL și anume împărțirea numerelor cu refacerea restului parțial și împărțirea numerelor fără refacerea restlui parțial.

Obiectivele principale sunt realizarea diagramelor de stări ale algoritmilor, descrierea și parcurgerea celor două metode de împărțire în exemple construite clar și precis, descrierea unei scheme bloc a componentelor utilizate și legătura dintre acestea, proiectarea structurală în limbaj VHDL pentru componentelor alese, testarea funcționalității proiectului în testbench-uri și în final, implementarea pe o placă de dezvoltare FPGA.

Un algoritm de diviziune este un algoritm care, având în vedere două numere întregi, N și D, calculează coeficientul și restul, rezultatele diviziunii. Acest algoritm este adesea folosit în proiecte de circuite digitale și software, sub diverse metode. Algorimii de diviziune se împart în două categorii principale: diviziune lentă și diviziune rapidă. Algoritmii de divizare lentă produc o cifră din coeficientul final pe iterație. Exemple de diviziune lentă includ restaurarea, restaurarea neperformantă, nerestabilirea și divizarea SRT. Metodele de divizare rapidă încep cu o apropiere apropiată de coeficientul final și produc de două ori mai multe cifre din coeficientul final pe fiecare iterație. Algoritmii Newton – Raphson și Goldschmidt se încadrează în această categorie. Variantele acestor algoritmi permit utilizarea algoritmilor de multiplicare rapidă. Rezultă că, pentru numere întregi mari, timpul necesar calculatorului pentru o diviziune este același, până la un factor constant, ca timpul necesar unei înmulțiri, oricare ar fi algoritmul de multiplicare utilizat.

Astfel, s-au constatat a fi două tipuri de algoritmi: algoritmi lenți și algoritmi rapizi. În categoria algoritmilor lenți se încadrează algoritmul cu refacerea restului, algoritmul fără refacerea restului parțial, algoritmul cu refacerea restlui parțial, algoritmul SRT (Sweeney, Robertson, and Tocher), iar în categoria celor rapizi, algoritmii Newton-Raphson și GoldSchmidt.

În continuare, se vor prezenta particularități pentru doi algoritmi de divizare lentă și anume: divizarea cu refacerea restului și divizarea fără refacerea restului.

În cazul împărțirii cu refacerea restului parțial, în fiecare etapă, se efectuează deplasarea la stînga a restului parțial și scăderea împărțitorului din restul parțial. Dacă se obține un rezultat negativ, restul parțial trebuie refăcut.

Pentru împărțirea fără refacerea restului parțial, după deplasarea la stânga a restului parțial, operația depinde de rezultatul din pasul precedent. Dacă restul parțial a fost pozitiv, se efectuează o scădere, iar dacă a fost negativ, se efectuează o adunare.

1. **Fundamentare teoretică**

Se vor prezenta în această secțiune specificațiile si logica algoritmilor propuși, diagrama de stare și o descriere asupra superiorității si avantajelor alegerii de proiectare a acestor metode de divizare a numerelor întregi.

Deoarece în metoda de divizare cu refacerea restului este posibil să fie necesară o corecție de adiție la fiecare pas, pot fi nevoie de până la 2n-1 pași pentru n cifre, în comparație cu metoda fără refacerea restului, în care sunt necesari maxim n + 1 pași, fiind mai rapidă.

Algoritmul fără refacerea restului parțial este mai complex, dar are avantajul, atunci când este implementat în hardware, că există o singură decizie și adunare / scădere pe bitul cotient; nu există nicio etapă de restabilire după scădere, care poate reduce numărul de operații cu până la jumătate și permite executarea mai rapidă.

În continuare, se va prezenta descrierea în detaliu a algoritmilor, împreuna cu un exemplu de divizare.

* Algoritmul de împărțire cu refacerea restului parțial

În algoritmul cu refacerea restului parțial, registrul A stochează restul, iar registrul Q, câtul. În primul pas al algoritmului, se realizează inițializarea regiștrilor și anume, A se inițializează cu 0, inclusiv bitul de semn, registrul B se inițializează cu împărțitorul, registrul Q cu deîmpărțitul, iar N, cu numărul de biți ai registrului Q. În continuare, în fiecare iterație se realizează o shiftare la stanga a registrului A cu biții registrului Q, în acest mod deplasându-se la stânga ambii regiștrii cu un bit. În continuare, se face o scădere între registrul A și B. Dacă semnul registrului A este negativ, bitul 0 al registrului Q devine 1, altfel, se reface restul parțial printr-o adunare, A = A + B, și bitul 0 al registrului Q devine 0. N scade cu 1, iar dacă este egal cu 0, algoritmul se oprește, altfel, se reîncepe o nouă iterație urmând toți pasșii prezentați anterior. Algoritmul cu refacerea restului parțial, față de cel fără, restochează la fiecare iterație valoarea registrului A.

În cazul împărțirii cu refacerea restului parțial, dacă notăm cu B împărțitorul și cu A restul, avem:

A = A – B

A = A – B + B

A = 2 \* A

A = 2 \* A – B

În tabelul de mai jos, se ilustrează o trasare a algorimului de împărțire pentru deîmpărțit = 14 și împărțitor = 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pas | A | Q(X) | B(Y) | Operații |
| 0 | 0 0000 | 1110 | 0100 |  |
| 1 | 0 0001  0 0001 + 1 1100 = 1 1101  1 1101 + 0100 = 0 0001 | 1100  1100 | 0100 | Shl (A,Q)  A = A – B  A < 0 => A = A + B, Q0 = 0 |
| 2 | 0 0011  0 0011 + 1 1100 = 1 1111  1 1111 + 0100 = 0 0011 | 1000  1000 | 0100 | Shl (A, Q)  A = A – B  A < 0 => A = A + B, Q0 = 0 |
| 3 | 0 0111  0 0111 + 1 1100 = 0 0011 | 0000  0001 | 0100 | Shl (A, Q)  A = A – B  A > 0 => Q0 = 1 |
| 4 | 0 0110  0 0110 + 1 1100 = 0 0010 | 0010  0011 | 0100 | Shl (A, Q)  A = A – B  A > 0 => Q0 = 1 |

* Algoritmul de împărțire fără refacerea restului partial

În cazul împărțirii fără refacerea restului parțial, timpul de execuție al operațiilor scade considerabil. Inițializarea regiștrilor este identică cu cea prezentată în algoritmul anterior, iar în continuare sunt mici diferențe între operațiile din cadrul iterațiilor.

Registrele A si Q se deplasează la stânga simultan, biții registrului Q deplasându-se in A, iar dacă în pasul precedent restul parțial a fost pozitiv, se scade împărțitorul din restul parțial A, în caz contrar, adunându-se. La fiecare sfârșit de iterație, se scade numărul N cu 1. În ultimul pas, dacă N este egal cu 0, dacă restul parțial este negativ, se adună împărțitorul încă o dată, algoritmul luând astfel sfârșit. Încă o diferență considerabilă între algoritmii cu și fără refacerea restului parțial este că, aici, se adaugă un bistabil suplimentar care indică operația care trebuie efectuată. Dacă valoarea bistabilului este 0, se efectuează scădere, restul parțial fiind pozitiv, iar dacă valoarea acestuia este 1, se realizează adunare, restul parțial fiind înainte negativ. În etapa de inițializare, bistabilul primește valoarea 0, efectuându-se în primul pas o scădere.

În cazul împărțirii fără refacerea restului parțial, dacă notăm cu B împărțitorul și cu A restul, avem:

A = A – B

A = 2 \* A – 2 \* B

A = 2 \* A – 2 \* B + B

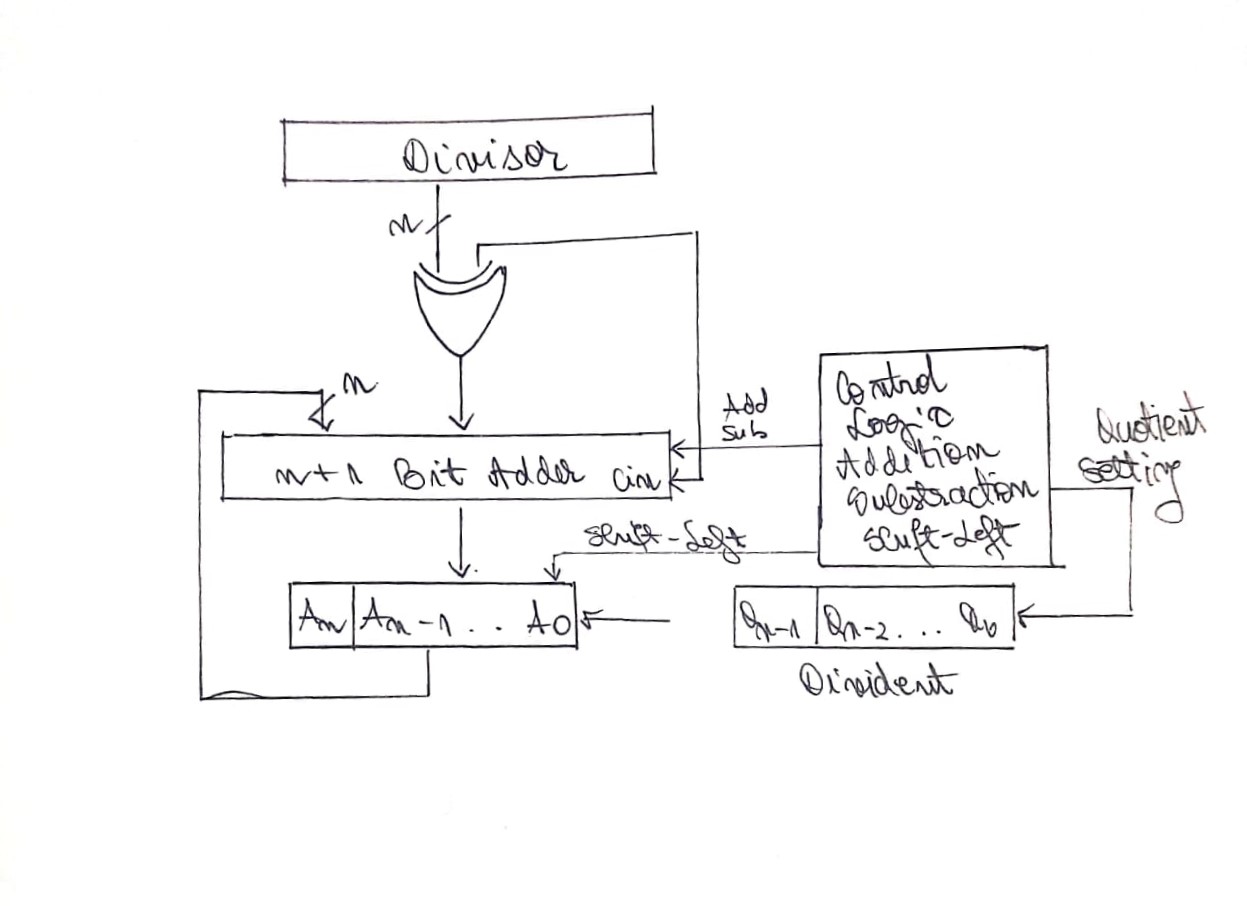
În tabelul de mai jos, se ilustrează o trasare a algorimului de împărțire pentru deîmpărțit = 14 și împărțitor = 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pas | A | Q(X) | B(Y) | BS | Operatii |
| 0 | 0 0000 | 1110 | 0100 | 0 |  |
| 1 | 0 0001  0 0001 + 1 1100 = 1 1101 | 1100  1100 | 0100 | 1 | Shl (A, Q)  BS != 1 => A = A – B  A < 0 => Q0 = 0, BS = 1 |
| 2 | 1 1011  1 1011 + 0100 = 1 1111 | 1000  1000 | 0100 | 1 | Shl (A, Q)  BS = 1 => A = A + B  A < 0 => Q0 = 0, BS = 1 |
| 3 | 1 1111  1 1111 + 0100 = 0 0011 | 0000  0001 | 0100 | 0 | Shl (A, Q)  BS = 1 => A = A + B  A > 0 => Q0 = 1, BS = 0 |
| 4 | 0 0110  0 0110 + 1 1100 = 0 0010 | 0010  0011 | 0100 | 0 | Shl (A, Q)  BS != 1 => A = A – B  A > 0 => Q0 = 1, BS = 0 |
| 5 | 0 0010 | 0011 | 0100 | 0 | BS != 1 => STOP |

1. **Proiectare și implementare**

În această secțiune se vor prezenta schema bloc a circuitului de împărțire, descrierea componentelor, organigramele corespunzătoare. Componentele si schemele bloc impreuna cu semnalele implicate vor fi de asemenea analizate in acest capitol.

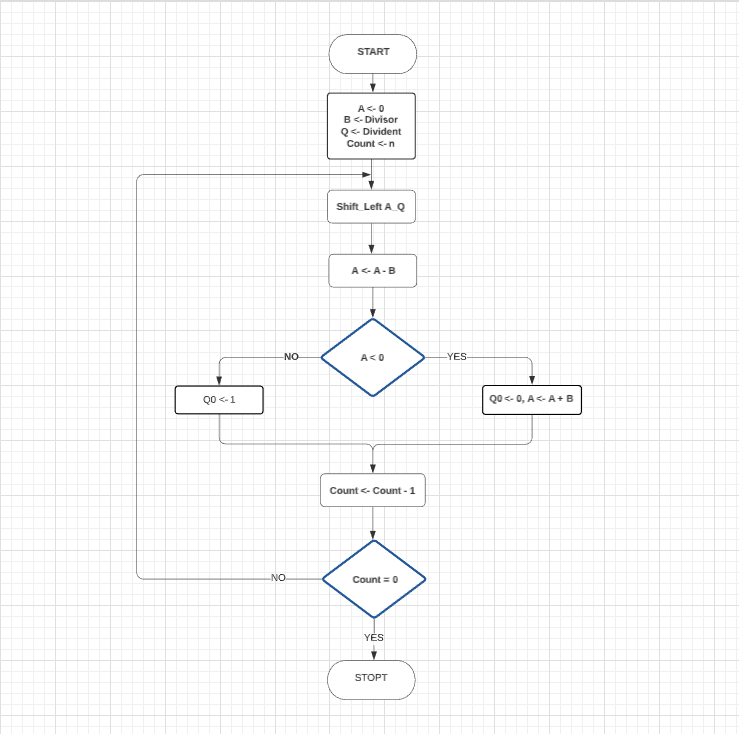
* Schema bloc principala a metodei de divizare a numerelor intregi



In continuare sunt prezentate componentele utilizate in realizarea implementarii vhdl a metodelor de divizare. Sunt prezentate scehemele bloc impreuna cu functionalitatile individuale si semnalele necesare si RTL Schematic.

**1. Metoda de divizare a numerelor intregi prin refacerea restului partial**

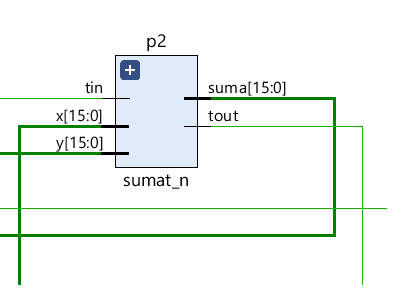
* Organigrama algoritmului de divizare cu refacerea restului parțial



Componentele utilizate in aceasta metoda sunt construite cu ajutprul genericelor, astfel incat acestea pot fi utilizate pentru oirce numar n de biti. Aceste sunt urmatoarele: sumator de n biti, registru de n biti cu resetare sincrona, registru de deplasare la dreapta de n biti cu resetare sincrona, registru de deplasare la dreapta de n biti cu resetare sincrona si cu updatarea vectorului de date, unitate centrala pentru metoda de divizare cu refacerea restului partial si afisorul cu 7 segmente pentru placa de dezvoltare basys3.

* Sumator de n biti

Aceasta componenta este realizata utilizand generice si un sumator de 1 bit complet. Acesta aduna 2 numere pe n biti, tinand cont de carry in si genereaza pe iesire suma rezultata tot pe n biti si un carry out.



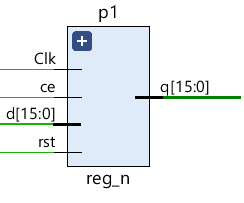
Semnale de intrare:

* Tin: semnal carry in
* X: vector de date de n biti
* Y: vector de date de n biti

Semnale de iesire:

* Suma: vector de date de n biti (suma x+y)
* Tout: semnal carry out
* Registru de n biti cu resetare sincrona

Registru declarat preia la intrare un vector de date pe n biti, iar in functie de semnalele de clk, reset si clk enable, il genereaza pe iesirile acestuia. Transpunerea corecta a datelor de la intrare la iesire tine cont de validarea semnalului de ceas conditionata de semnalul clock enable. Daca reset este activat pe ‚1’, atunci pe iesirea registrului se va genera vectorul de date ‚0’.



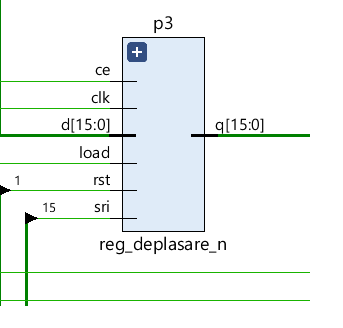
Semnale de intrare:

* Clk: semnal de ceas
* Ce: semnal de enable
* D: vector de date de n biti
* Rst: semnal de reset

Semnale de iesire:

* Q: vector de date de n biti (vectorul d este transpus pe iesire)
* Registru de deplasare la dreapta de n biti cu resetare sincrona

Cu ajutorul acestui registru si a functionalitatii sale, putem shifta la dreapta un vector de n biti, cu aplicarea pe pozitia n-1 a unui serial input. Resetarea este sincrona, astfel incat pentru activare, clk si rst trebuie sa fie setate pe ‚1’. Altfel, incarcarea se face in mod paralel, iar shiftarea este conditionata de clock enable.



Semnale de intrare:

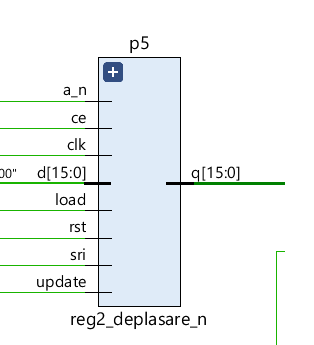
* Ce: semnal de clock enable
* Clk: semnal de ceas
* D: vector de date de n biti
* Load: semnal semnal de incarcare paralela a registrului cu datele aplicate la intrare
* Rst: semnal de reset
* Sri: intrarea seriala (shift right input)

Semnale de iesire:

* Q: vector de n biti (vectorul D se shifteaza la dreapta, iar pe prima pozitie se aplica sri)

• Registru de deplasare la dreapta de n biti cu resetare sincrona si cu updatarea vectorului de date

Acest registru are aceleasi functionalitati ca si cel prezentat anterior si are in plus un flag pentru update-ul bitului 0 al semnalului care este disponibil pe iesirile registrului. Acesta va avea valoarea conditionata de flag-ul update, care controleaza activarea updateului, iar valoarea acestuia va fi controlata de valoarea bitului de intrare a\_n.



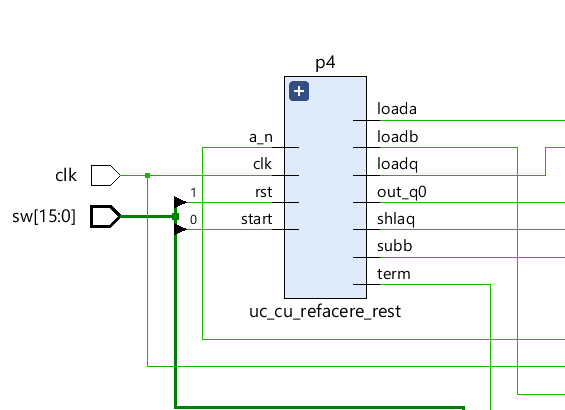
Semnale de intrare:

* Ce: semnal de clock enable
* Clk: semnal de ceas
* D: vector de date de n biti
* Load: semnal semnal de incarcare paralela a registrului cu datele aplicate la intrare
* Rst: semnal de reset
* Sri: intrarea seriala (shift right input)
* Update: semnal pentru update-ul bitului n-1 al vectorului de date de n biti

Semnale de iesire:

* Q: vector de n biti (vectorul D se shifteaza la dreapta, iar pe prima pozitie se aplica sri)
* Unitate centrala pentru metoda de divizare cu refacerea restului partial

Unitatea centrala este componenta care controleaza valorile semnalelor implicate in activarea incarcarii anumitor valori sau schimbarea valorilor anumitor semnale necesare in intregul proces de impartire. In UC este declarat un vectori de stari si anume, starile prezentate in organigrama metodei de impartire pentru fluxul controlului de date si a deciziilor posibile: idle, init, check, scadere control, conditie, adunare, scadere, shiftare, stop, setare\_q\_0 setare\_q\_1. Daca reset este activat pe ‚1’, atunci procesul se afla in starea idle. Altfel, din idle, prin activarea semnalului start, procesul trece in starea init, urmata de shiftare, scadere si conditie. In starea de conditie, in functie de validarea semnalului a\_n, procesul poate trece in starile setare\_q\_0 si setare\_q\_1. Din starea setare\_q\_0, procesul trece in starea de adunare, dupa care in scadere\_contol, iar din starea setare\_q\_1, procesul trece direct in starea scadere\_contor. Scadere contor reprezinta scaderea numarului de biti n cu 1, la fiecare pas in care procesul atinge starea respectiva. Din starea respectiva, se trece in check, iar daca acest contor atinge valoarea 0, atunci starea trece in stop si algoritmul se termina, altfel, in shiftare. Intr-un alt proces, in functie de valoarea starii, semnalelor le sunt asignate valorile corespunzatoare.



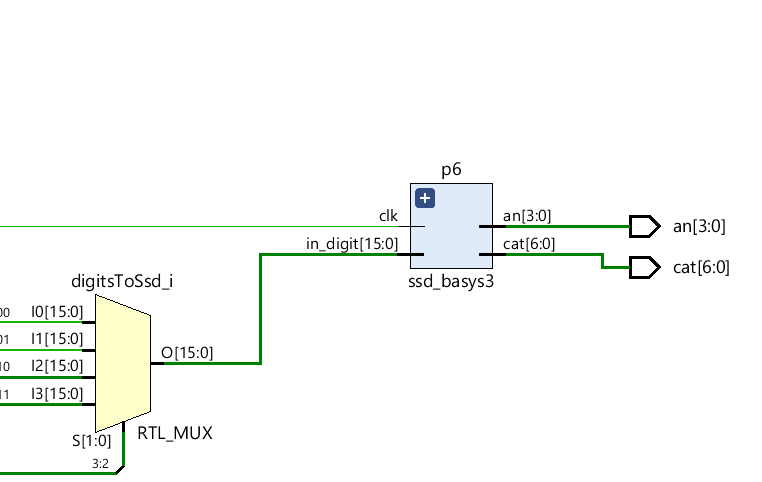
Semnale de intrare:

* A\_n: semnal care indica operatia care trebuie efectuata (adunare/scadere)
* Clk: semnal de ceas
* Rst: semnal de reset
* Start: semnal de start

Semnale de iesire:

* Loada: semnal pentru incarcarea vectorului de date a in registru pentru o operatie de adunare/scadere
* Loadb: semnal pentru incarcarea vectorului de date b in registru pentru o operatie de adunare/scadere
* Out\_q0: pozitia 0 a vectorului de date q
* Shlaq: Semnal pentru validarea activarii shitarii la dreapta a vectorilor de date a si q
* Subb: semnal pentru validarea unei operatii de scadere
* Term: semnal de validare a efectuarii complete a algoritmului cu obtinerea unui cat si a unui rest
* Afisor 7 segmente

Afisorul cu 7 segmente este utilizat in transpunerea deimpartitului, impartitorului, restului si catului pe placa basys3. Numarul incarcat va fi ales in functie de o combinatie a switch-urilor de pe placa.



Semnale de intrare:

* Clk: semnal de ceas
* In\_digit: vectorul de date care este afisat pe placuta

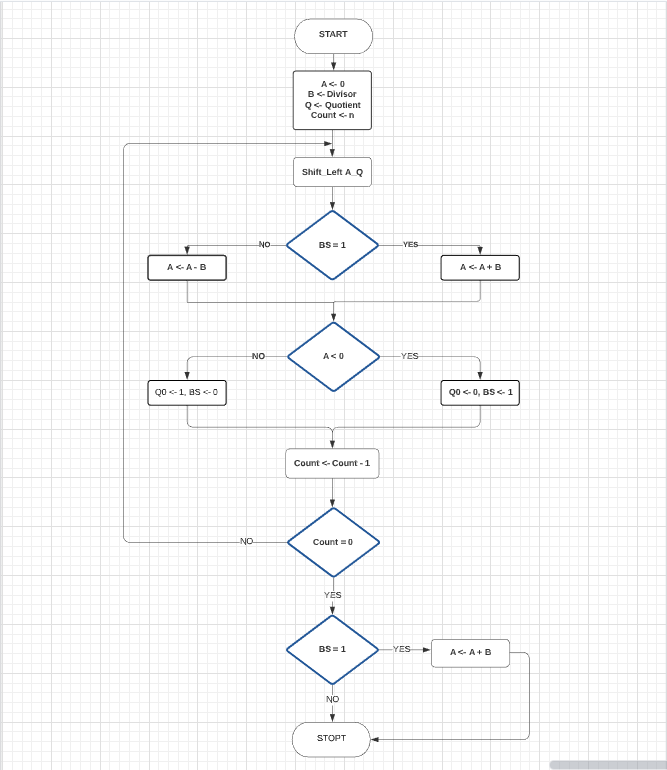
Semnale de iesire:

* An: semnal anod
* Cat: semnal catod

1. **Metoda de divizare a numerelor intregi fara refacerea restului partial**

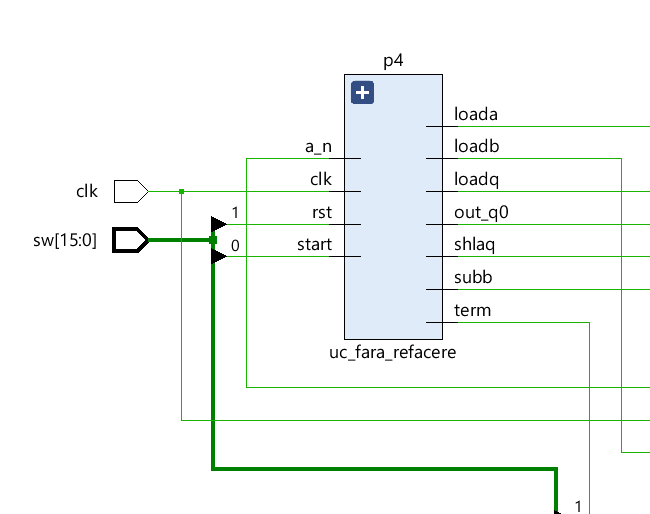
Componentele utilizate in aceasta metoda au fost prezentate mai sus, cu exceptia unitatii centrale, care are o structura diferita pentru implementarea divizarii.

• Organigrama algoritmului de divizare fără refacerea restului partial

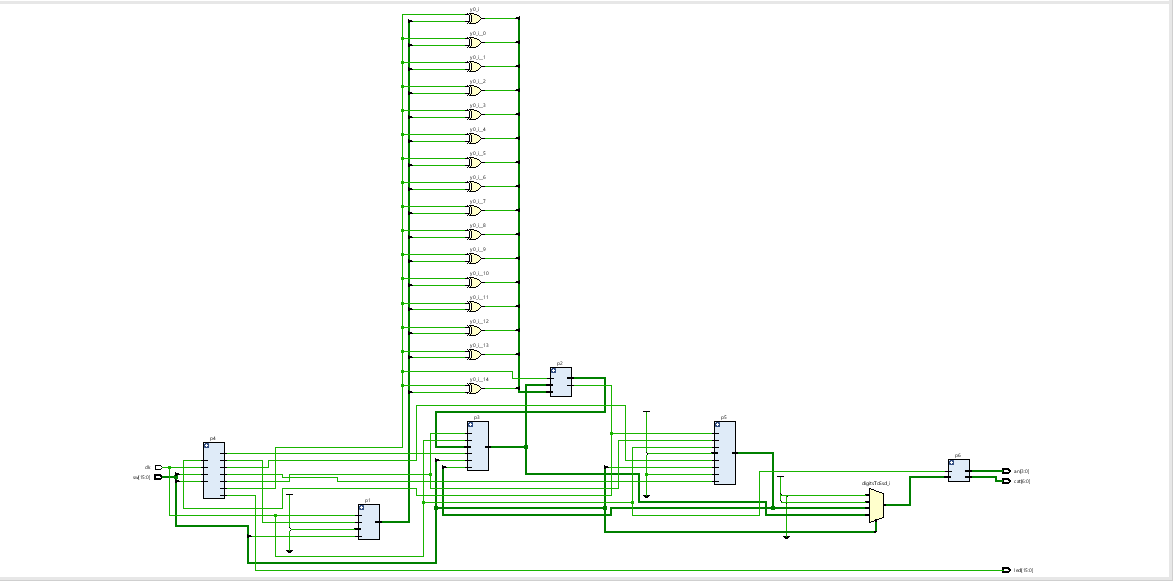
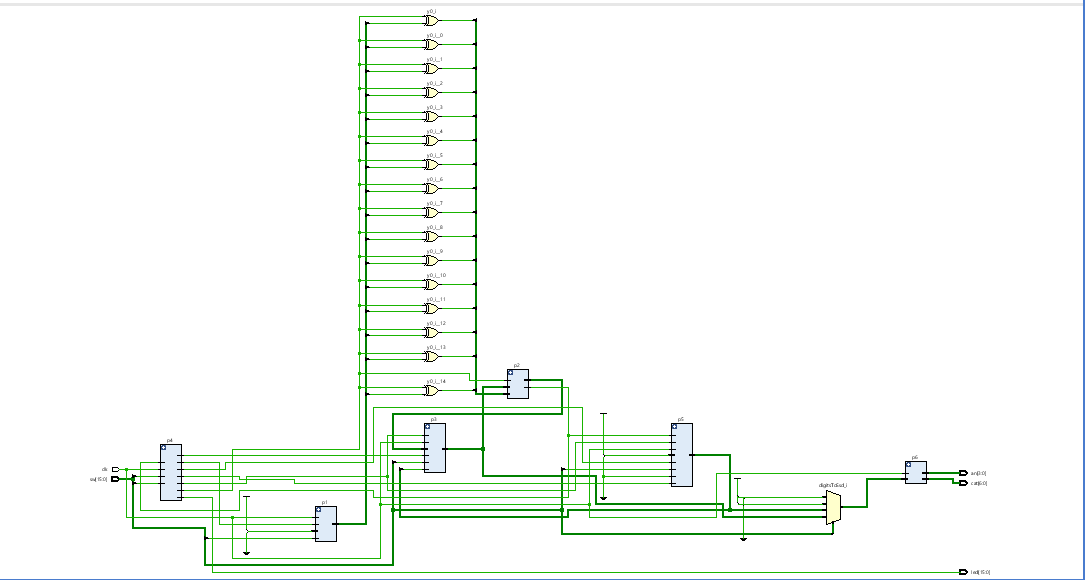


* Unitatea centrala pentru metoda de divizare fara refacerea restului partial

Prin cee ace se difera aceasta unitate fata de cea prezentata anterior este fluxul starilor de control. Acestea sunt: idle, init, last\_add, conditie\_bs\_last, setare\_bs\_1, setare\_bs\_0, conditie\_bs, conditie\_q0, check, scadere\_contor, conditie, adunare, scadere, shiftare, stop, setare\_q\_0, setare\_q\_1. Daca reset este activat pe ‚1’, atunci procesul se afla in starea idle. Altfel, din idle, prin activarea semnalului start, procesul trece in starea init, urmata de shiftare si conditie. Din starea de conditie, in functie de semnalil bistabilului bs, procesul trece in starea conditie\_q0 sau conditie\_q1. In starea conditie\_q0, in functie de semnalul de intrare a\_n, starea procesului trece in setare\_q\_0 sau setare\_q\_1. Din aceste 2 ultime stari, se trece in conditie\_bs. Din conditie\_bs, in functie de starea semnalului a\_n, starea trece in setare\_bs\_1 sau setare\_bs\_0. Din aceste 2 urmeaza o trecere in starea scadere\_contor, iar apoi in check. In check, daca numarul de biti care e scazut in starea anterioara atinge valoarea 0, atunci starea trece in conditie\_bs\_last, altfel, in shiftare. Din conditie\_bs\_last, daca semnalul de bistabil are valoarea ‘1’ atunci starea trece in last\_Add, alfel procesul se termina. Din last\_add, starea procesului trece de asemenea in stop.



Fisierele RTL Schematic obtinute in simulatorul Vivado sunt prezentate mai jos, pentru cele 2 metode de divizare.

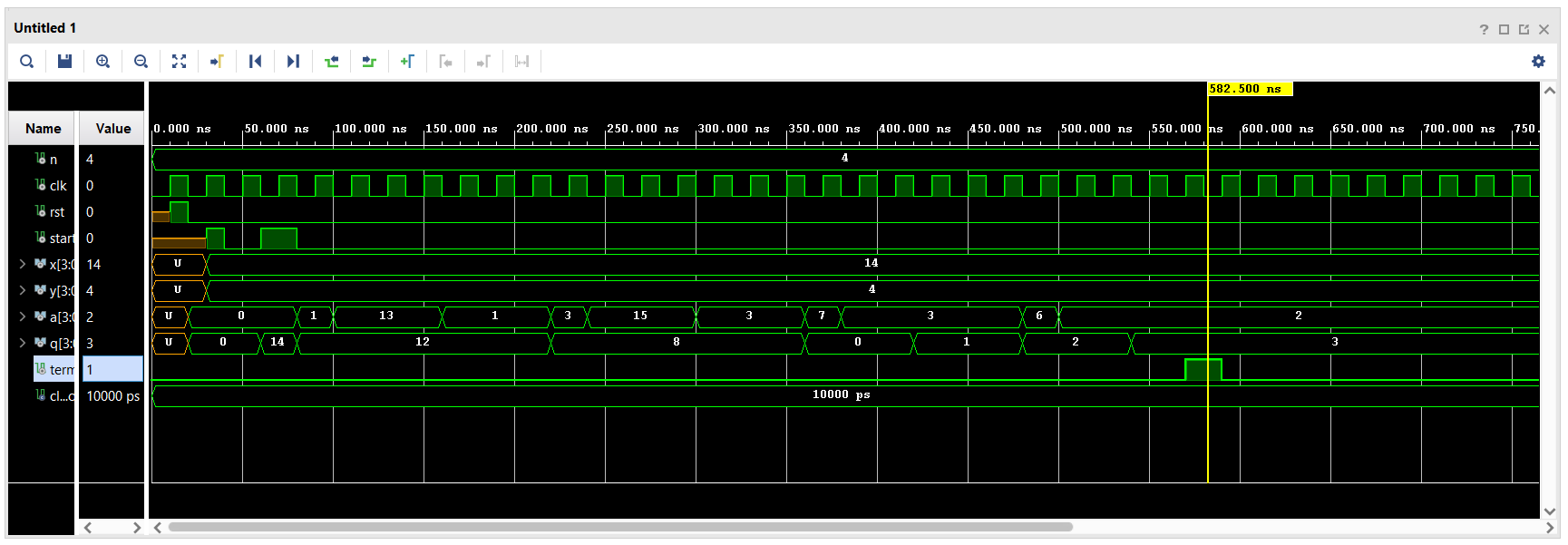
* RTL Schematic pentru divizarea numerelor intregi cu refacerea restului partial
* RTL Schematic pentru divizarea numerelor intregi fara refacerea restului partial

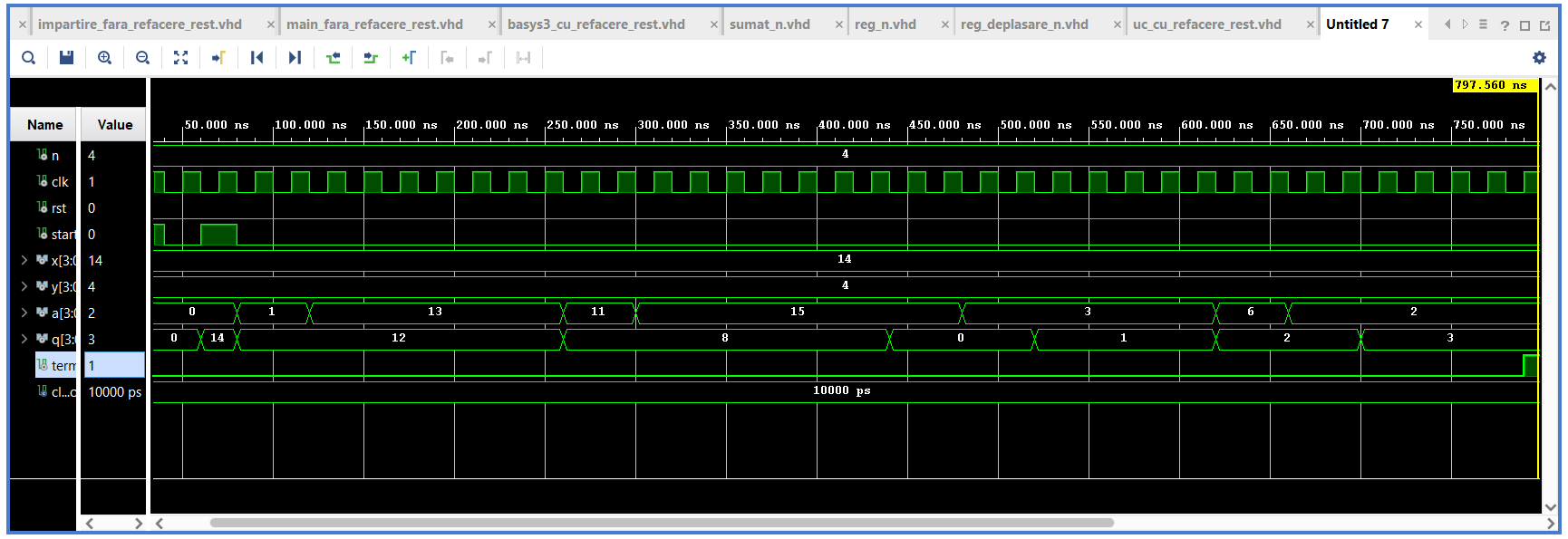
1. **Rezultate experimentale**

In aceasta sectiune vor fi analizate rezultatele experiementale obtinute in urma aplicarii metodelor de divizare a unui set de numere intregi, atat in simulator, cat si transpunerea proiectului pe o placa de dezvoltare basys3. Se vor prezenta test-bench-uri efectuate in simulatorul vivado a componentelor utilizate si trasari ale rezultatelor obtinute. Pentru exemplificarea rezultatelor obtinute pe basys3 se vor insera fotografii cu aplicarea algoritmilor, care sa corespunda deimpartitului, impartitorului, catului si restului.

In continuare, se vor prezenta si analiza rezultatele obtinute in urma implementarii algoritmilor in mediul de dezoltare Vivado, construit special pentru dezvoltarea circuitelor utilizand limbajul de programare vhdl. Bentru bancurile de test, s-au generat module de simulare cu date corespunzatoare celor prezentate in capitolul anterior, si anume, impartirea numarului 14 la 4, pentru realizarea unei trasari usoare.

In testul de simulare sunt initializate valoarea generica a lui n (numarul de biti al numerelor implicate in diviziune) si valorile deimpartitului si a impartitorului. Se realizeaza generic map-ul si port map-ul fisierelor main utilizand semnalele declarate in prealabil in fisier. Se va crea un proces in cadrul caruia se realizeaza crearea frecventei semnalului de ceas si un proces in cadrul caruia se instantiaza semnalele cu valorile corespunzatoare. Initial, se asigneaza pentru reset valoarea ‚0’, urmata de ‚1’, iar pentru start valoarea ‚1’ pentru inceperea depanarii algoritmului.

* ****Impartirea cu refacerea restului partial a numarului 14 la numarul 4
* Impartirea fara refacerea restului partial a numarului 14 la numarul 4

****

1. **Concluzii**

Acest proiect a aavut ca si scop aprofundarea metodelor de impartire a numerelor intregi, si anume impartirea numerelor intregi fara refacerea restului partial si impartirea numerelor intregi cu refacerea restului partial.

Un avantaj al utilizarii diviziunii fara refacerea restului partial fata de diviziunea cu refacerea restului partial este faptul ca se elimnia necesitatea unei scaderi in timpul testului, bitul de semn determinand daca se foloseste o adunare sau o scadere. Dezavantajul este ca trebuie mentinut un bit suplimentar in restul partial pentru a urmari semnul.

1. **Bibliografie**

<https://en.wikipedia.org/wiki/Division_algorithm>

<https://www.researchgate.net/publication/3043862_Division_algorithms_and_implementations>