MINISTERUL EDUCAŢIEI ŞI CERCETĂRII ŞTIINŢIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

CIRCUITE DE IMPARTIRE ZECIMALA

Indrumator de proiect: Dr. Ing. Cristi Mocan

Student: Viman Andrei-Liviu

Data: 20.10.2021

Grupa: 30236

CUPRINS

Rezumat 4
Introducere 5
Contextul temei 5
Domeniul de studiu 6
Fundamentare teoretica 7
Metoda refacerii restului partial 7
Metoda fara refacerea restului partial 8
Metoda celor noua multiplii ai impartitorului 11
Placa folosita si detalii despre aceasta 13
Proiectare si implementare 16
Arhitectura 16
Schemae bloc 17
Sumator pe un bit 17
Sumator zecimal elementar 18
Sumator zecimal 16 biti 18
Registru n biti cu resetare sincrona 19
Registru de deplasare la stanga de n biti 20
Unitatea de control pentru impartirea cu refacerea restului 21
Unitatea de control pentru impartirea fara refacerea restului 22
Registru n biti pentru impartrea cu 9 multiplii 23
Unitatea de control pentru impartirea cu cei 9 multiplii 24
Impartirea cu refacerea restului partial 26
Impartirea fara refacerea restului partial 27
Impartirea cu cei 9 multiplii ai impartitorului 28
Rezultate si experimente 29
Sumator pe un bit 29
Sumatorul zecimal elementar 29
Sumator zecimal 16 biti 30
Registru n biti cu resetare sincrona 31

Registru de deplasare la stanga de n biti 32
Registru n biti pentru impartrea cu 9 multiplii 33
Impartirea cu refacerea restului partial 34
Impartirea fara refacerea restului partial 35
Impartirea cu cei 9 multiplii ai impartitorului 36
Mod de utilizare 37
Concluzii 38
Aspecte generale 38
Dezvoltari ulterioare 38
Bibliografie 39
Anexa 40
Sumator1bit 40
Sumator4biti 41
SumatorZecimal 42
sumator_16biti 43
nbit_register 45
slrn_registru_de_deplasare_stanga 46
unitateaDeControl_impartireCuRefacere 48
impartireaCuRefacere 54
unitateaDeControl_impartireFaraRefacere 58
impartireFaraRefacere 65
unitateaDeControl_nouaMultiplii 68
impartireNouaMultiplii 87
generator_multiplii 91
nbit_register_9multiplii 92
decodificator7seg 94
displ7seg 96
filtru_buton 99
implementare_pe_placuta 102

REZUMAT

Acest proiect ne ajuta pentru a intelege mai bine metodele de impartire zecimale si cum functioneaza circuitele zecimale. Tot aici putem spune ca acest proiect ne ajuta pentru a putea intelege cum snt reprezentate numerele in calculator si cum face unitatea aritmetica a procesorului operatiile. In acelasi timp ne ajuta sa ne familiarizam mult mai bine cu placuta de dezvoltare Nexys4 si cu perifericele ei. Putem spune ca acest mini proiect ne ajuta sa folosim mult mai usor FSM (Finite-State Machine) atunci cand implementam o unitate de control. Codul sursa este scris in limbajul VHDL, acesta fiind un limbaj de descriere hardware iar mediul de dezvoltare folosit este Vivado Xilinx, cu ajutorul caruia codul surasa este sintetizat.

INTRODUCERE

Contextul temei

Impartirea zecimala este una dintre cele mai utilizate operatii aritmetice alaturi de adunare, scadere si inmultire. Fiind dat deimpartitul X si impartitorul Y , operatia de impartire consta in determinare acatului Q si a restului R astfel incat sa fie satisfacuta relatia: X = Q * Y + R . Algoritmul de impartire zecimala incearca sa determine cifrele catului prin alegerea unei cifre si scaderea produsului dintre aceasta cifra si impartitor sin restul partial (care este o parte a deimpatitului). Daca rezultatul scaderii este un numar pozitiv mai mic decat impartitorul, cifra aleasa este corecta. In caz contrar, se alege o alta cifra si operatia se repeta. In fiecare etapa a operatiei se obtine o cifra a catului. Pentru operatiile de impartire nu sunt valabile proprietatile de comutativitate și asociativitate. La impartirea binara, lucrurile stau mult mai simplu deoarece sunt numai cifre de 0 sau 1 astfel alegerea nu mai este necesara operatia reducandu-se la o serie de scaderi ale impartitorului din restul partial, care se efectueaza numai daca restul partial este mai mare decat impartitorul.[1]

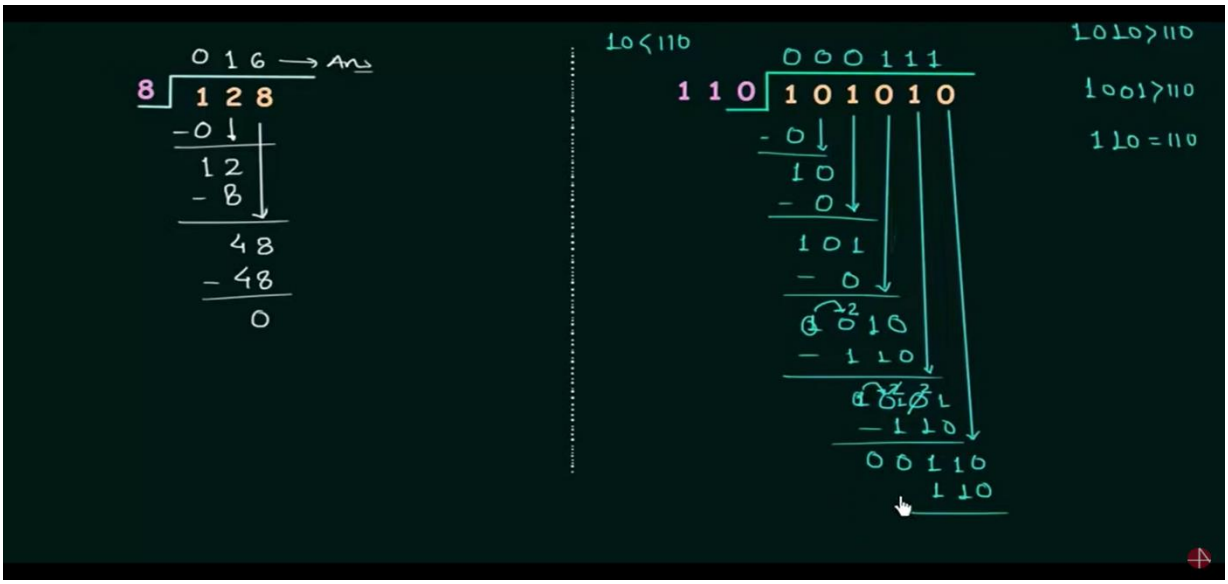
In matematica exista doua metode de a realiza impartirea, acestea fiind impartirea cu refacerea restului partial si cea fara refacerea restului partial. Împărțirea cu refacerea restului parțial: in fiecare etapă, se efectuează deplasarea la stânga a restului partial și scaderea împartitorului din restul parțial. Daca se obtine un rezultat negativ, restul partial trebuie refacut. Impartirea fara refacerea restului parțial: după deplasarea la stânga a restului partial, operația depinde de rezultatul din pasul precedent. Daca restul partial a fost pozitiv se efectueaza scadere iar daca restul partial a fost negativ se efectueaza o adunare.[2]

Pentru a putea implementa acest algoritm am ales sa folosesc un limbaj de descriere hardware deoarece imi ofera posibilitatea unei descrieri functionale a sistemelor, ea fiind o descriere de nivel inalt, fara a mai fi nevoit sa detaliem structura la nivelul portilor elementare, asfel timpul necesar descrierii sistemelor complexe se reduce semnificativ; dar si posibilitatea modificarii mai simple a descrierii hdl, datorita faptului ca descrierea reprezinta si o documentare a proiectului. Descrierile hdl sunt independente fata de diferite tipuri de circuit. Schemele logice sunt realizate cu componente de biblioteca specifice unei anumite familii de circuite programabile, pe cand descrierile hdl sunt complet independente fata de un anumit circuit.[3]

Domeniul de studiu

Proiectul de față urmărește implementarea eficientă a unor circuite de împărțire a numerelor zecimale si descrierea acestora în limbaj VHDL. În acest context, vom avea în vedere reprezentarea binară a numerelor întregi, precum și metodele de împărțire zecimala a numerelor. Un circuit este un traseu închis (format din fire sau din dispozitive bune conducătoare de electricitate) prin care poate circula curentul. Pe orice plăcuță FPGA se pot programa astfel de trasee închise cu ajutorul limbajelor de descriere hardware (cum este VHDL). Acest proiect are în vedere descrierea și încărcarea pe o plăcuță a unor astfel de circuite care doresc să înfăptuiască împărțirea numerelor zecimale reprezentate în binar.

Împărțirea efectuată cu ajutorul circuitelor și cu ajutorul reprezentării binare a numerelor zecimale îmbină domeniul matematic cu cel al electronicii și cu domeniul proiectării logice. Este importantă intrepatrunderea acestor domenii întrucât calculatoarele au fost special concepute pentru a realiza calcule și operații mai rapid decât mintea umană le poate efectua. Proiectul curent va urmări implementarea a mai multor metode de împărțire a numerelor zecimale pe care le vom descrie în secțiunea de fundamentare teoretică.



FUNDAMENTARE TEORETICA

Impartirea zecimala este mai complexa decat impartirea binara, deoarece zifrele catului pot lua valori intre 0 si 9. Aceasta presupune efectuarea in fiecare etapa a unui numar variabil de adunari si scaderi ale impartitorului. Metodele impartirii binare pot fi aplicate si pentru impartirea zecimala deoarece nu exista deosebiri de principiu intre acestea. Considerand ca numerele pot fi reprezentate dupa marime si semn, atunci vom putea utiliza urmatoarle metode de impartire: metoda refacerii restului partial, metoda fara refacerea restului partial, metoda celor noua multiplii ai impartitorului, metoda injumatatirii impartitorului.

Metoda refacerii restului partial

Aceasta metoda este similara cu cea utilizata pentru impartirea binara, efectuandu-se prin scaderi repetate ale impartitorului din restul partial. In fiecare etapa, se obtine o cifra a catului care este valoarea cea mai mare din cele zece posibile pentru care rstul partial este inca pozitiv. Atunci cand restul partial devine negativ, se aduna impartitorul pentru a reface restul partial.

Pentru a determina cifrele catului, blocul de comanda contine un numarator, care este initializat cu zero la inceputul fiecarei etape si este incremntat la fiecare scadere a impartitorului din restul partial. Scaderea impartitorului continua pana cand restul partial devine negativ, moment in care se reface restul partial , iar continutul decrementat al cifrei numaratorului reprezinta cifra catului corespunzator etapei curente. Daca numaratorul se incrementeaza atunci cand restul partial devine negativ, nu mai este necesara decrementarea acestuia . Inaintea inceperii operatiei se testeaza daca apare o depasire si daca rezultatul operatiei este zero, in mod similar cu impartirea binara.

In continuare voi pune un tabel in care sa exemplific operatia de impartire pas cu pas pentru a se intelege mai bine:

Pas	Az	Qz	Bz	N	N1	Operatii
0	0 06	84	28	2	0	Initializare
1	0 68	40	28	1	0	Deplasare stanga az_qz
	0 68- 28 0 40	40	28	1	1	Scadere bz incrementare N1
	0 40 – 28 0 12		28	1	2	Scadere bz incrementare N1
	0 12- 28 9 84	40	28	1	3	Scadere bz incrementare N1
	9 84+ 28 0 12	42	28	1	2	Abunare bz Decrementare n1 Qz0 = N1
2	1 24	20	28	0	0	Deplasare stanga az_qz
	1 24 – 28 0 96	20	28	0	1	Scadere bz incrementare N1
	0 96 – 28 0 68	20	28	0	2	Scadere bz incrementare N1
	0 68 – 28 0 40	20	28	0	3	Scadere bz incrementare N1
	0 40 – 28 0 12	20	28	0	4	Scadere bz incrementare N1
	0 12 – 28 9 84	20	28	0	5	Scadere bz incrementare N1
	9 84 + 28 0 12	24	28	0	4	Abunare bz Decrementare n1 Qz0 = N1

Metoda fara refacerea restului partial

Aceasta metoda elimina adunarea impartitoruli la restul partial, atunci cand acesta devine negativ. Considerand ca restul partial initial este 0: $r_0 = 0$ primul rest

partial r_1, este obtinut prin deplasarea restului partial initial la stanga cu o pozitie zecimala, la care se adauga cifara cea mai semnificativa a deimpartitului, x_{n-1}. Din aceasta valoare se scade impartitorul de m_1 ori (unde m_1 poate avea o valoare cuprinsa intre 1 si 10), pana cad diferenta devine negativa: $r_1 = 10\ r_0 + x_{n-1}\ Y = x_{n-1}-m_1Y$. Deoarece r_1 este prima diferenta negativa, prima cifra a catului este $q_{n-1}=m_1-1$.

Restul partial r_1 se deplaseaza la stanga cu o pozitie zecimala. Daca r_1 a fost negativ, al doilea rest partial r_2, este obtinut prin adunarea de m_2 ori a impartitorului la r_1 deplasat la stanga, pana cand suma devine pozitiva: $r_2 = 10r_1+x_{n-2}+m_2Y = 10\ (x_{n-1} – q_{n-1}Y) + x_{n-2}-10Y+m_2Y = 10\ (x_{n-1} – q_{n-1}Y) + x_{n-2} –(10-m_2)Y$. A doua cifra a catului va fi: $q_{n-2} =10 - m_2$. Al doilea rest partial devine: $r_2 = 10\ (x_{n-1} – q_{n-1}Y) + x_{n-2} –q_{n-2}Y= 10r_1+x_{n-2} +(10 – q_{n-2})Y$. Deci, daca primul rest partial r_1, a fost negativ, acesta a fost refacut. Valoarea $10 – q_{n-2}$ ete complementul fata de zece a cifrei q_{n-2}. Daca primul rest partial a fost pozitiv, al doilea rest partial este obtinut prin scaderea fr m_2 ori a impartitorului din restul partial r_1 deplasat la stanga, pana cand diferenta devine negativa: $r_2=10\ r_1 + x_{n-2} – m_2Y = 10\ (x_{n-1} – q_{n-1}Y) + x_{n-2} –m_2Y$ iar a doua cifra a catului va fi $q_{n-2} = m_2$. Al doilea rest partial va fi: $r_2=10r_1+x_{n-2}-q_{n-2}Y$.

Aceste operatii se continua pana cand se obtin toate cifrele catului, sau pana cand restul partial devine zero. Rezulta urmatorul algoritm al metodei fara refacerea restului partial:

1. Se deplaseaza la stanga registrul combinat care contine restul partial si deimpartitul. Restul initial fiind pozitiv, se efectueaza scaderi repetate ale impartitorului din restup partial, pana cand diferenta devine negativa. Daca s-au exectuat m_1 scaderi, cifra catului este m_1-1.
2. Se deplaseaza la stanga registrul combinat care contine restul partial si deimpartitul. Se efectueaza adunari repetate ale impartitorului la restul partial, pana cand suma devine pozitiva. Daca s-au efectuat m_2 adunari, cifra catului este $10-m_2$.
3. Se continua cu etapele 1 si 2, pana cand se obtin toate cifrele catuli.
4. Daca dupa ultima etapa restul este negativ, se reface restul partial prin adunarea impartitorului.

Structura dispozitivului care face acest lucru este similara cu cea a unui dispozitiv de impartire care utilizeaza netoda refacerii restului partial. In continuare voi pune un tabel in care sa exemplific operatia de impartire pas cu pas pentru a se intelege mai bine:

Pas	Az	Qz	Bz	N	N1	Operatii
0	0 06	84	28	2	0	Initializare
1	0 68	40	28	1	0	Deplasare stanga az_qz
	0 68 – 28 0 40	40	28	1	1	Scadere bz incrementare N1
	0 40 – 28 0 12	40	28	1	2	Scadere bz incrementare N1
	0 12 – 28 9 84	42	28	1	3	Scadere bz incrementare N1 Qz0 = N1-1
2	8 44	20	28	0	0	Deplasare stanga az_qz
	8 44 + 28 8 72	20	28	0	1	Adunare bz incrementare N1
	8 72 + 28 9 00	20	28	0	2	Adunare bz incrementare N1
	9 00 + 28 9 28	20	28	0	3	Adunare bz incrementare N1
	9 28 + 28 9 56	20	28	0	4	Adunare bz incrementare N1
	9 56 + 28 9 84	20	28	0	5	Adunare bz incrementare N1
	9 84 + 28 0 12	24	28	0	6	Adunare bz incrementare N1 Qz0 = 10 - N1

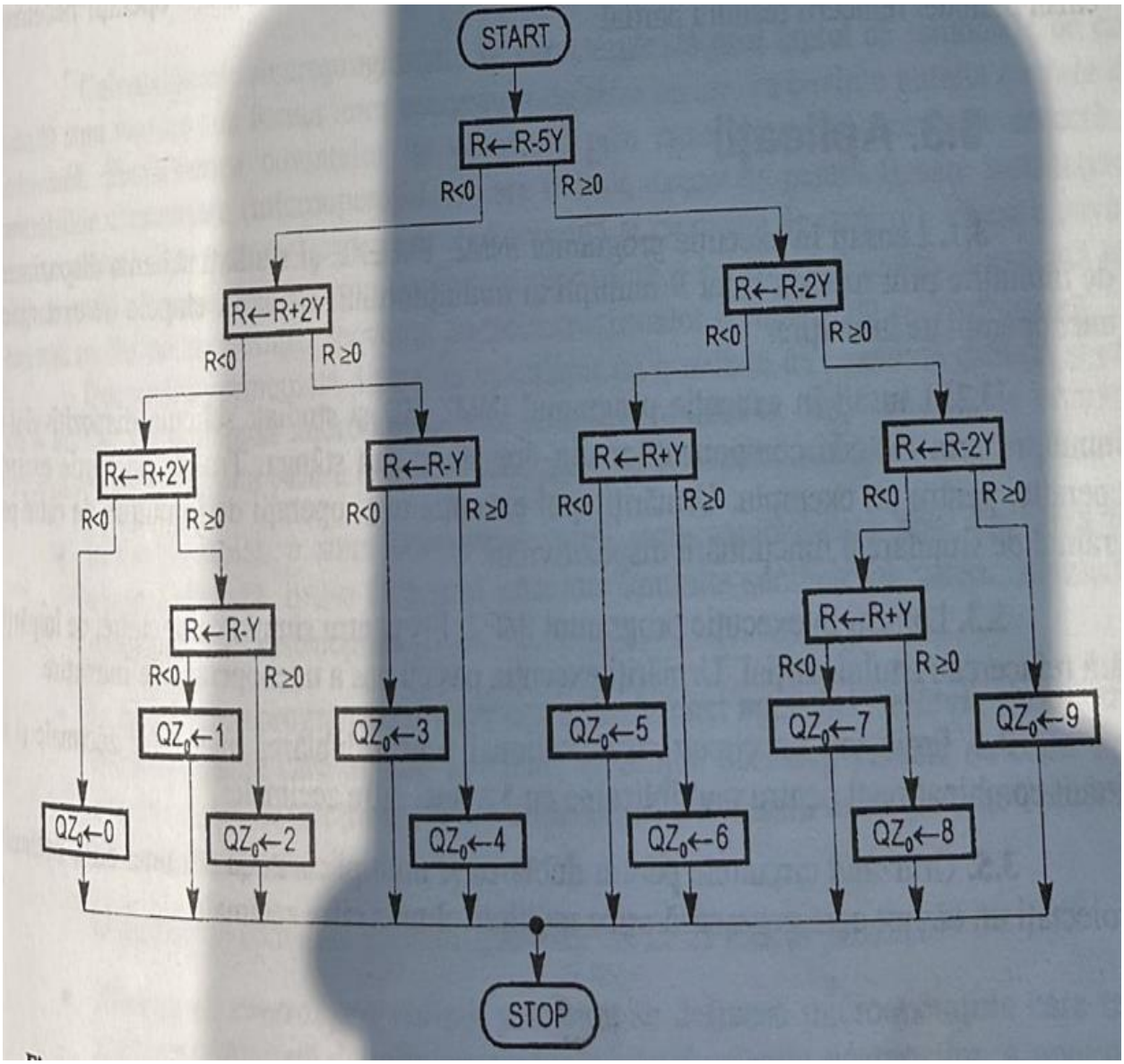
Metoda celor noua multiplii ai impartitorului

Metoda celor noua multiplii ai impartitorului are ca avantaj reducerea timpului de executrie a operatiei de impartire. Dispozitivul care utilizeaza aceasta metoda este asemanator cu cel de inmiltire prin metoda celor noua multiplii ai deinmultitului, continand noua grupe de registre care se incarca cu multiplii impartitorului. Sumatorul zecimal elementar este inlocuit cu un sumator-scazator zecimal elementar, sumatorul fiind folosit in etapa de generare a multipliloor impartitorului, iar scazatorul in etapa de impartire proprriu-zisa. Blocul de comanda contine un comparator, avand rolul de a selecta multiplul cel mai mare la impartitorului care, prin scadere din restul partial, determina un rezultat pozitiv.

In fiecare etapa a operatiei, se deplaseaza la stanga registrul combinat care contine restul partial si deimpartitul, si se compara restul partial cu multiplii impartitorului. Se efectueaza apoi scaderea multiplului selectat al impartitorului din restul partial. Dupa efectuarea scaderii, factorul de multiplicare corespunzator multiplului selectat este memorat drept cifra a catului. Daca nu se considera timpul necesar gnerarii multiplilor impartitorului, pentru fiecare cifra este necesara o operatie de scadere.

Deoarece pastrarea tuturor celor noua multiplii necesita un echipament costisitor, se utilizeaza circuite care genereaza numai anumiti multiplii ai impartitorului. In particular, se pot utiliza aceleasi circuite combinationale pentru generarea multiplilor de doi si de cinci. On acest caz, blocul de comanda trebuie sa determine in fiecare etapa cifra corespunzatoare a catului, iar apoi multiplul care trebuie scazut din restul partial. Succesiunea operatiilor necesare pentru determinarea catului partial este prezentata in figura de mai jos unde s-a notat cu R restul partial, iar cu Y impartitorul.

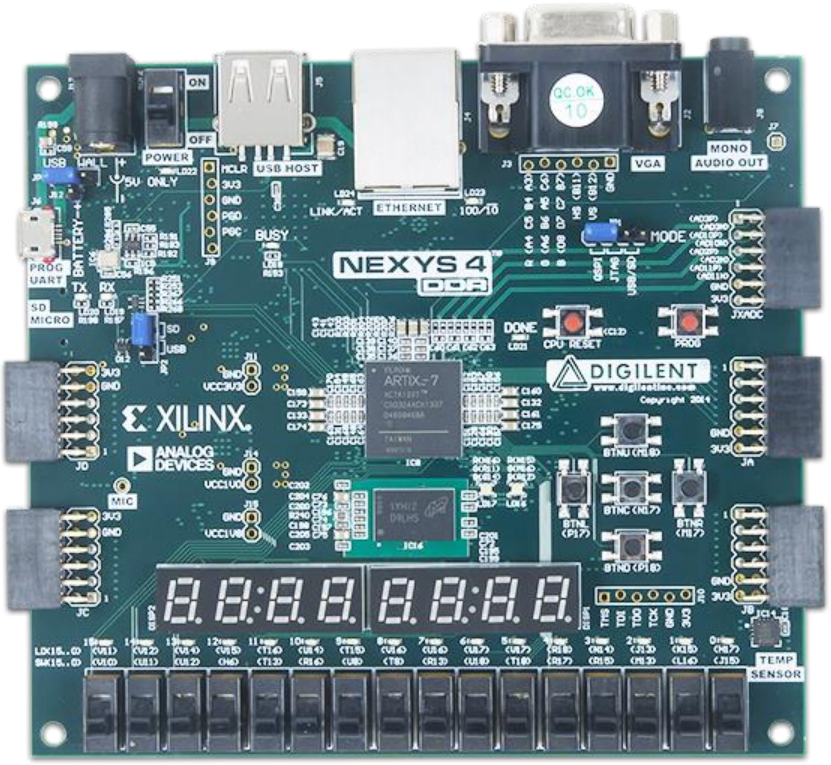
Initial, se scade multiplul de cinci al impartitorului din restul partial. Daca rezultatul este negativ, cifra catului este cuprinsa intre zero si patru. In acest caz se aduna multiplul de doi al impartitorului la restul partial. Daca rezultatul este negativ, cifra catului este cuprisa intre zero si doi, si se aduna din nou multiplul de 2 al impartitorului la restul partial. Daca rezultatul este tot negativ, cifra catului este zero, in caz contrar cifra fiind unu sau doi. In ultimul caz, se scade impartitorul din restul



partial; daca rezultatul este negativ, cifar catului este 1 iar daca rezultatul este pozitiv sau zero atunci cifra catului este 2. In mod similar se poate determina cifra catului in cazul in care dupa scaderea multiplului de 5 al impartitorului din restul partial este pozitiv sau zero.

Placa folosita si detalii despre aceasta

Placa DDR Nexys 4 este o platforma completa, gata de utilizare pentru dezvoltarea circuitelor digitale, bazata pe cea mai recenta matrice de porti programabile (FPGA) Artix-7™ proiectata de Xilinx. Cu o multime de functionalitati, cu o mare capacitate dde folosire in vaste domenii (Xilinx cu numarul de catalog XC7A100T-1CSG324C), amintim aici porturile sa le extrem de numerose si de versatile: USB, Ethernet, si multe alte porturi, DDR Nexys4 poate fi cumparata si utilizata pentru diverse lucruri, modelele variind de la circuite combinationale introductive la procesoare incorporate puternice. Mai multe periferice încorporate, inclusiv un accelerometru, un senzor de temperatură, un microfon digital MEMS, un amplificator de difuzor, Și mai multe dispozitive I/O permit utilizarea DDR-ului Nexys4 pentru o gamă largă de proiecte, fără a fi nevoie de alte componente.



Placa DDR Nexys4 poate primi energie de la portul USB-JTAG Digilent (J6) sau de la o sursă de alimentare externă. Jumper JP3 (lângă mufa de alimentare) determină ce sursă este utilizată. Toate sursele de alimentare Nexys4 DDR pot fi pornite și oprite de un singur comutator de alimentare la nivel logic (SW16). Un LED bun de putere (LD22), acționat de ieșirea "bună de alimentare" a sursei ADP2118, indică faptul că sursele sunt pornite și funcționează normal. O

prezentare generală a circuitului de alimentare DDR Nexys4 este prezentată în schema de mai jos:

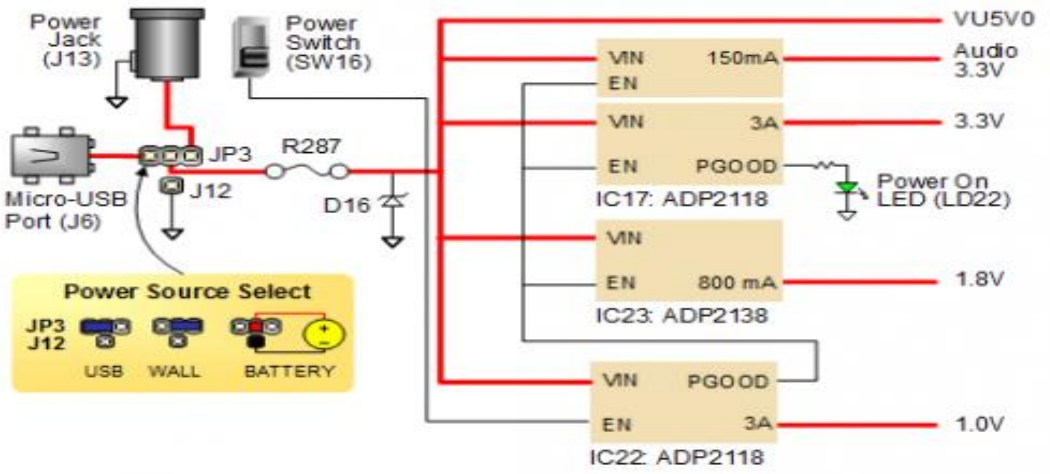


Figure 2. Nexys4 DDR power circuit.

Portul USB poate oferi suficientă energie pentru marea majoritate a design-urilor. Demo-ul nostru out-of-box atrage ~ 400mA de curent de la șina de intrare 5V. Câteva aplicații solicitante, inclusiv orice care conduce mai multe plăci periferice, ar putea necesita mai multă putere decât poate oferi portul USB. De asemenea, este posibil ca unele aplicații să trebuiști să ruleze fără a fi conectate la portul USB al unui PC. În aceste cazuri, se poate folosi o sursă de alimentare externă sau un pachet de baterii.

O sursă de alimentare externă poate fi utilizată prin conectarea la mufa de alimentare (JP3) și setarea jumperului J13 la "perete". Sursa de alimentare trebuie să utilizeze un coaxial, centru-pozitiv 2.1mm plug cu diametru intern, și să livreze 4.5VDC la 5.5VDC și cel puțin 1A de curent (adică, cel puțin 5W de putere). Multe consumabile adecvate pot fi achiziționate de la Digilent, prin Digi-Key sau alți furnizori de catalog.

Un pachet de baterii externe poate fi utilizat prin conectarea terminalului pozitiv al bateriei la pinul central al JP3 și terminalul negativ la pinul etichetat J12, direct sub JP3. Deoarece regulatorul principal de pe Nexys4 DDR nu poate găzdui tensiuni de intrare de peste 5.5VDC, un pachet de baterii externe trebuie să fie limitat la 5.5VDC. Tensiunea minimă a pachetului de baterii depinde de aplicație: dacă se utilizează funcția gazdă USB (J5), trebuie furnizată cel puțin 4.6V. În alte cazuri, tensiunea minimă este de 3,6V.

Placa DDR Nexys4 include un singur oscilator de cristal de 100 MHz conectat la pinul E3 (E3 este o intrare MRCC pe banca 35). Ceasul de intrare poate conduce MMCM-uri sau PLL-uri pentru a genera ceasuri de diferite frecvențe și cu relații de fază cunoscute care pot fi necesare pe parcursul unui design. Unele reguli restricționează MMM-urile și PLL-urile care pot fi conduse de ceasul de intrare de 100 MHz. Pentru o descriere completă a acestor reguli și a capacităților resurselor de pontaj Artix-7, consultați "Ghidul utilizatorului pentru resursele de pontaj FPGA din seria 7" disponibil de la Xilinx.

Xilinx oferă nucleul IP Clocking Wizard pentru a ajuta utilizatorii să genereze diferitele ceasuri necesare pentru un anumit design. Acest expert va instantia în mod corespunzător MMCM-urile și PLL-urile necesare pe baza frecvențelor dorite și a relațiilor de fază specificate de utilizator. Expertul va scoate apoi o componentă de înveliș ușor de utilizat în jurul acestor resurse de pontaj care pot fi introduse în designul utilizatorului. Expertul de pontaj poate fi accesat din cadrul instrumentelor Project Navigator sau Core Generator.

Placa DDR Nexys4 include două LED-uri tri-color, șaisprezece comutatoare de diapozitive, șase butoane, șaisprezece LED-uri individuale și un afișaj cu șapte segmente din opt cifre, așa cum se arată în Figura 16. Butoanele și comutatoarele de alunecare sunt conectate la FPGA prin rezistoare de serie pentru a preveni deteriorarea scurtcircuitelor accidentale (un scurtcircuit ar putea apărea dacă un pin FPGA atribuit unui buton de împingere sau unui comutator de alunecare a fost definit din greșeală ca o ieșire). Cele cinci butoane dispuse într-o configurație plus-sign sunt comutatoare "momentane" care generează în mod normal o ieșire scăzută atunci când sunt în repaus și o ieșire ridicată numai atunci când sunt presate. Butonul roșu etichetat "RESETARE CPU", pe de altă parte, generează o ieșire mare atunci când este în repaus și o ieșire scăzută atunci când este apăsată. Butonul RESET AL PROCESORULUI este destinat să fie utilizat în design-urile EDK pentru a reseta procesorul, dar îl puteți folosi și ca buton de uz general. Comutatoarele de diapozitive generează intrări constante ridicate sau scăzute, în funcție de poziția lor.

Placa mai include saisprezece LED-uri individuale de înaltă eficiență sunt conectate anod la FPGA prin rezistoare de 330 ohmi, astfel încât acestea se vor activa atunci când se aplică o tensiune logică înaltă pinului lor I/O respectiv. LED-urile suplimentare care nu sunt accesibile utilizatorului indică pornirea, starea programării FPGA și starea porturilor USB și Ethernet.

PROIECTARE SI IMPLEMENTARE

Arhitectura

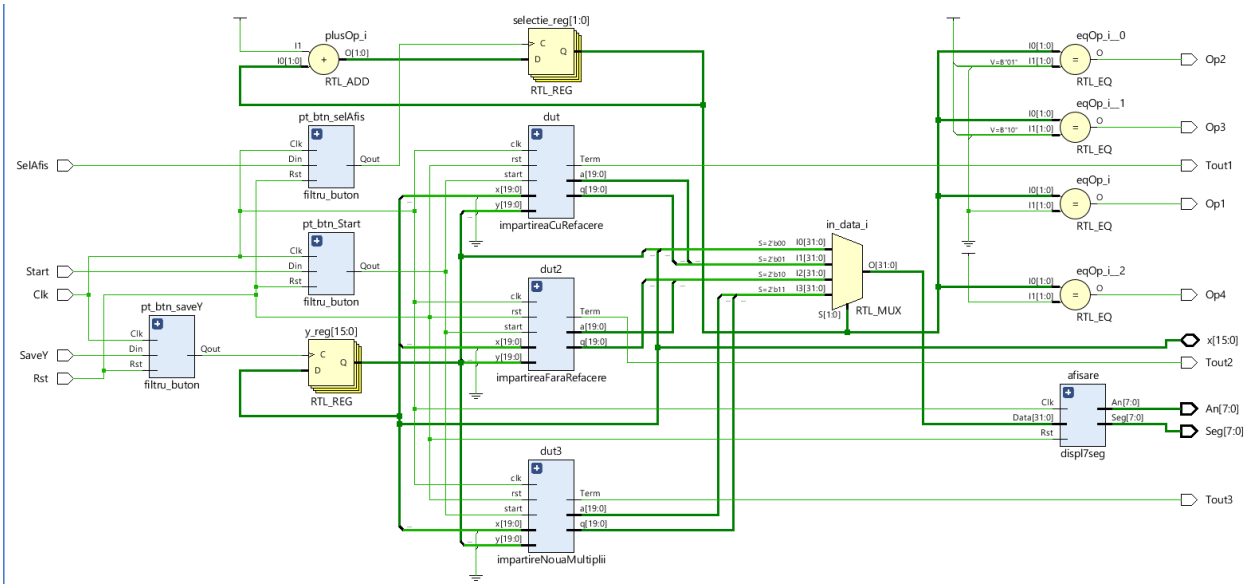
Cand ne aflam in fata faptului implinit, uitandu-ne la cerintele problemei dar si la scenariile de utilizare, se poate observa o conturare puternica a imaginii de ansamblu, adica avand 3 metode de impartire care trebuie afisate pe placuta, este clar ca avem nevoie de un controller pentru cele 2 seturi de afisoare bcd 7 segmente, avand nevoie sa le controlam pentru a putea afisa deoarece ele sunt catod comun si pentru ca ochiul nostru sa nu sesizeze doar o singura cifra avem nevoie de un divizor de fregventa si de o logica de transformare din binar in coduri pentru afisoare. De acest lucru displ7seg.

Deoarece nu putem introduce 2 numere deodata ne duce la concluzia ca avem nevoie de mai multe butoane pe care pentru a le putea folosi avem nevoie de debouncere pentru a le putea folosi, deoarece placa lucreaza la un clock foarte ridicat si dorim in momentul in care apasam butonul sa se apese doar o singura data nu de mai multe ori. In total avem 3 astfel de debouncer deoarece avem nevoie pentru butonul de start, incarcarea impartitorului, dar si de selectarea operatiei pe care dorim sa o afisam. Prntru butonul de reset nu avem nevoie de debounce deoarece din moment ce dorim sa resetam sistemul nu ne influenteaza daca el este apasat de mai multe ori.

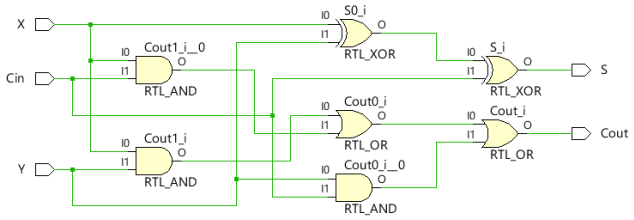
Pentru a putea stii in ce operatie ne aflam am ales sa aprind 3 leduri, cate unul pentru fiecare operatie, asfel dupa terminarea fiecarei operatii se va aprinde un led. In acelasi timp pentru a stii ce afisam pe ecran am ales sa aprind tot niste leduri, astfel atunci can afisam cele 2 numere se va aprinde primul led, cand afisam rezultatul primei operatii se va aprinde al doilea led si asa in continuare pentru fiecare operatie. Asadar am folosit in total 7 leduri.

Arhitectura in general este una destul de usoara deoarece operatiile nu sunt foarte complica, ele fiind 3 la numar. Tot in arhitectura mai exista un controlar pentru segmente si 3 debouncere pentru butoane. De aceea putem spune ca arhitectura nu este atat de complicata. Aceasta incepe sa se complice otata cu folosirea mai multor periferice ale placii.

Schemae bloc



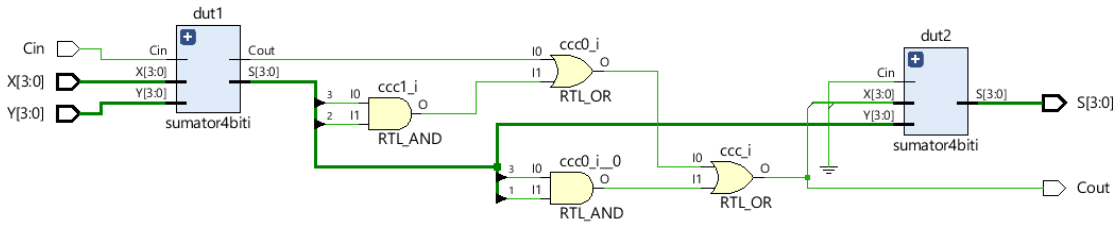
Sumator pe un bit



Sumatorul pe un bit este cea mai usoara componenta realizabila deoarece contine numai niste porti sau-exclusiv si cateva porti si si sau. Folosesc aceasta componenta pentru a putea alcatui un sumator pe patru biti care mai apoi imi va fi de folos pentru a putea face un sumator zecimal elementar deoarece proiectul se bazeaza pe o impartire zecimala asa ca am zis sa fac si sumatoarele si scazatoarele tot zecimal. Mai jos se poate observa structura RTL ci intrari si iesiri dar si cu componentele interne.

Sumator zecimal elementar

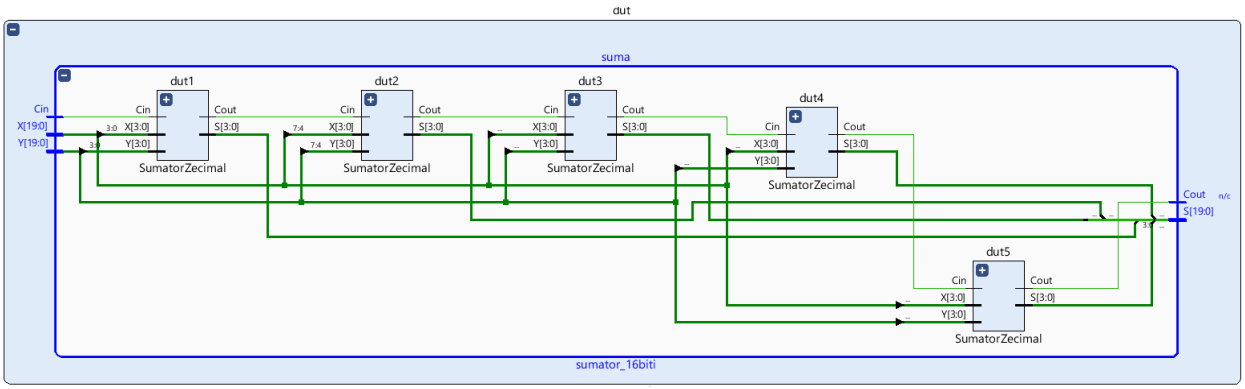
Sumatorul zecimal elementar este un sumator special deoarece contine doua sumatoare pe patru biti si o logica necesara astfel incat sa nu existe o depasire a cifrelor de la 0 la 9 in momentul in care una este depasita, in cel de-al doilea sumator se va aduna valoarea 6 in binar care va face ca numarul sa se transforme din baza 16 in baza 10. Pentru generarara bitului de depasire se foloseste o logica combinationala de mai multe porti logice urmand ca mai apoi iesiria de transport sa fie intrare pentru al toilea sumator, altfel spus se va aduna un 6 doar daca numarul este mai mare de 9. Mai jos se poate observa structura RTL ci intrari si iesiri dar si cu componentele interne.



Sumator zecimal 16 biti

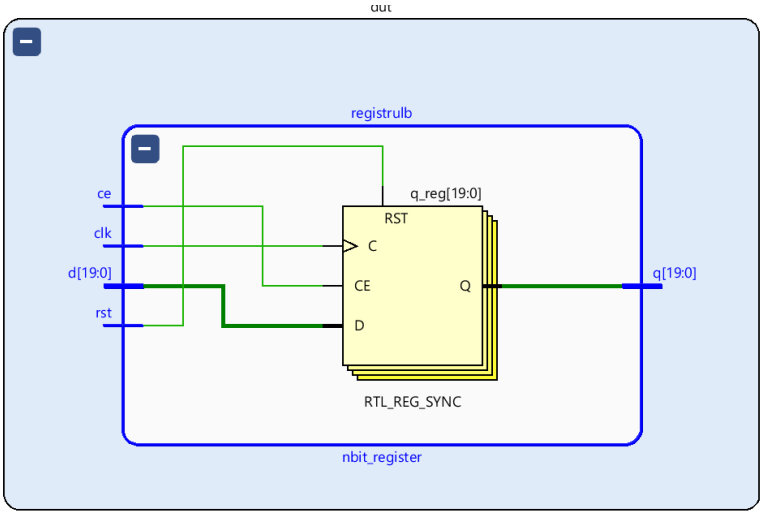
Sumatorul zecimal pe 16 biti este un sumator special deoarece contine cinci sumatoare zecimale elementare si o logica necesara astfel incat sa nu existe o depasire a cifrelor de la 0 la 9 in momentul in care una este depasita, in cel de-al doilea sumator se va aduna valoarea 6 in binar care va face ca numarul sa se transforme din baza 16 in baza 10. Pentru generarara bitului de depasire se foloseste o logica combinationala de mai multe porti logice urmand ca mai apoi iesiria de transport sa fie intrare pentru al toilea sumator, altfel spus se va aduna un 6 doar daca

numarul este mai mare de 9. Mai jos se poate observa structura RTL ci intrari si iesiri dar si cu componentele interne.



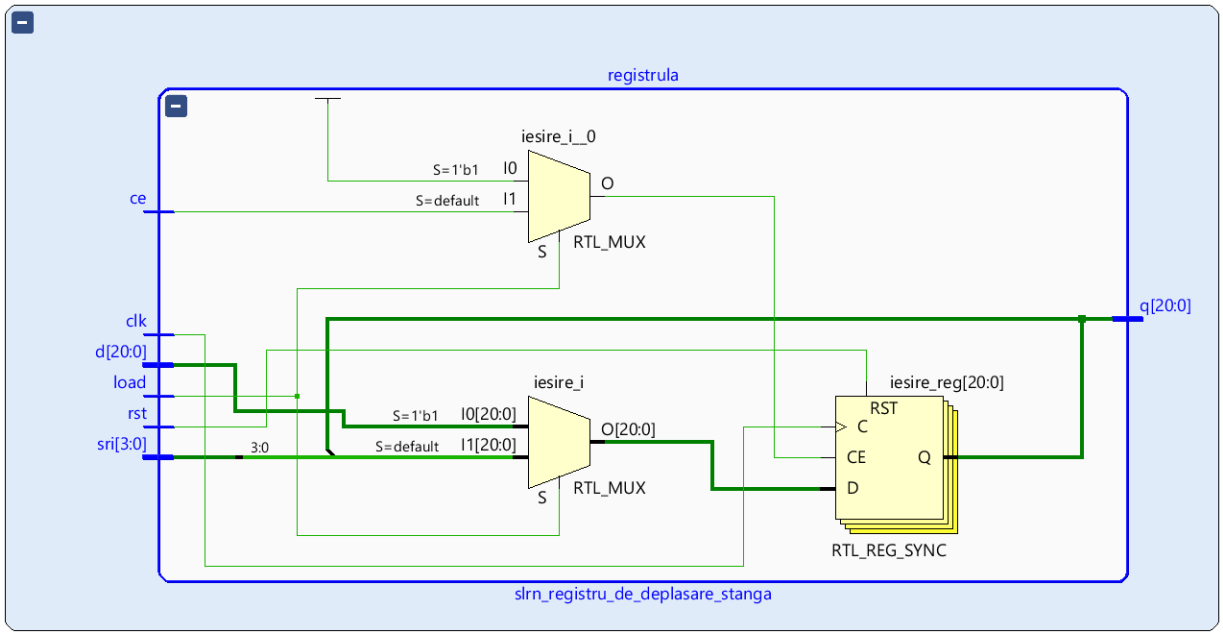
Registru n biti cu resetare sincrona

Intrarea de ceas a acestui registru este clk, intrarea de date este d, ea fiind un vector pe n biti, iar iesirea q tot un vector pe n biti. Daca semnalul rst este setat pe 1 logic, registrul este resetat in mod sincron pe frontul crescator al semnalului de ceas, iesirea fiind resetata la 0 logic, in caz contrar, daca rst nu este 1 iar semnalul ce este 1 logic se realizeaza incarcarea paralela a registrului cu datele aplicate la intrare, iar daca ce este 0 atunci iesirea se va pastra neschimbata. Acest registrru il folosesc pentru a salva impartitorul in etapele de impartire.



Registru de deplasare la stanga de n biti

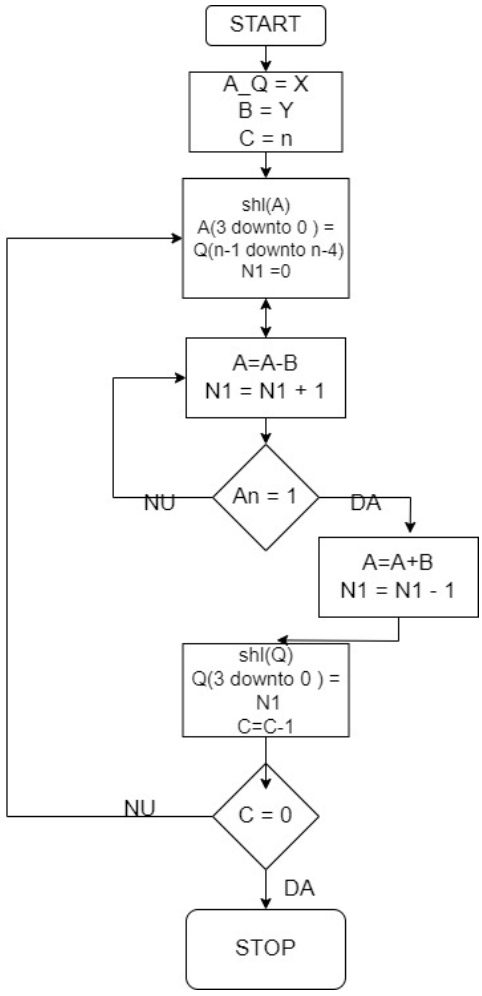
Intrarea de ceas a registrului de deplasare este clk, intrarea paralela de date este vectorului de n biti d, iar intrarea de date sri este utilizata ca si intrare aproape seriala, deoarece incarca 4 biti odata nu unul singur. Iesirea vectorului q este tot pe n biti. Daca semnalul rst este 1 logic, registrul este resetat in mod sincron pe frontul crescator a semnalului de ceas, iesirea de date fiind resetata la 0 logic. In caz contrar daca intrarea load este pe 1 logic acesta se va incarca paralel, datele de intrare fiind aplicate iesirii. Daca semnalul load nu este 1 logic, iar semnalul ce este 1 logic, se realizeaza deplasarea la stanga cu 4 pozitii si incarcarea lui paralela cu intrarea seriala sri. Deplasam cu 4 pozitii doarece atatea poziti poate ocupa o cifra in baza 10. Acest registru este utilizat pentru a stoca catul si restul operatiei de impartire.



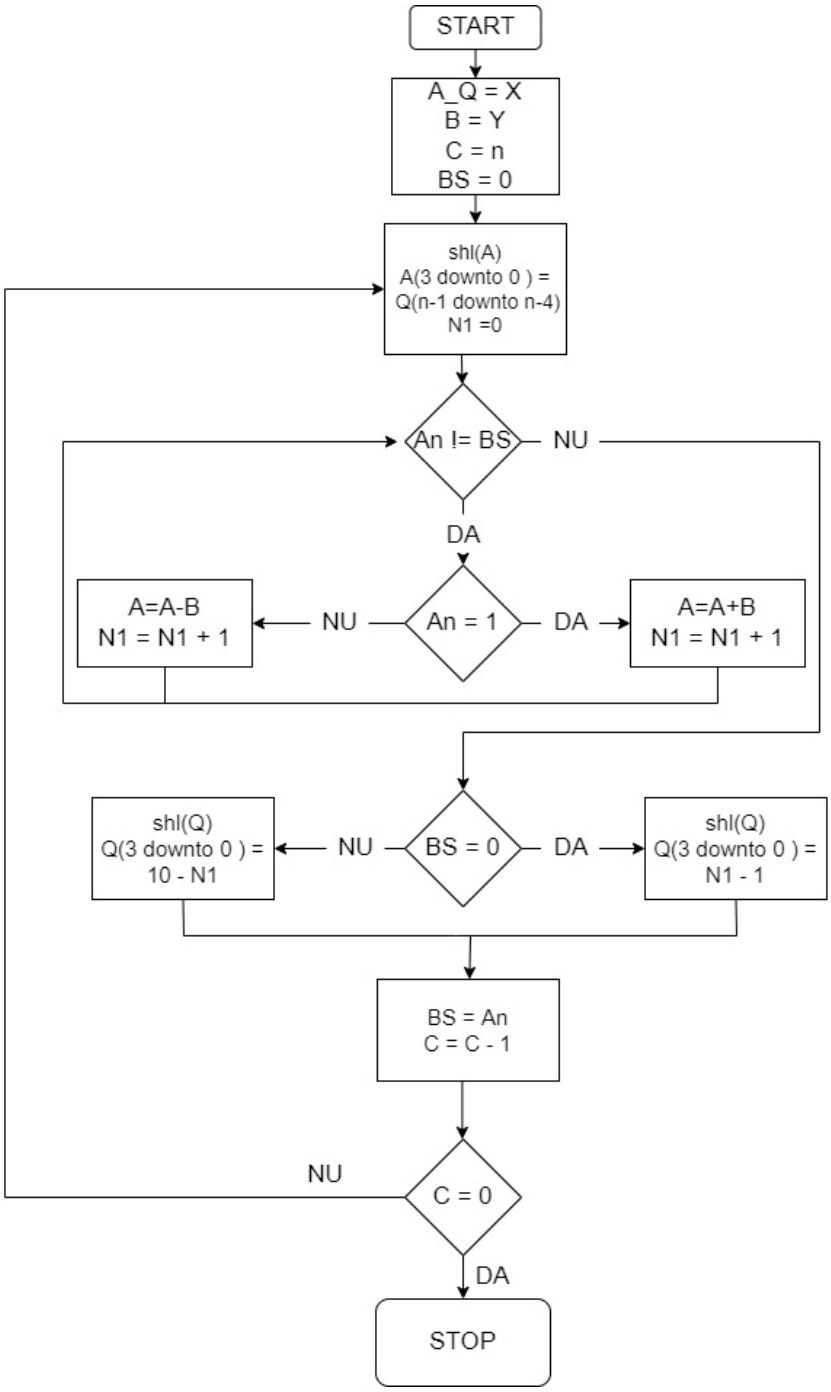
Am gandit acest registru in acest mod deoarece in unitatea de control cand calculez cifra pe care trebuie sa o introduc sa imi fie mult mai usor si sa nu pierd mult timp la incarcarea lui paralela, deoarece am vrut ca impartirea sa se realizeze mult mai repede, deoarece in ziua de azi este foarte importanta viteza de lcru a unui procesor.

Unitatea de control pentru impartirea cu refacerea restului

Aceasta componenta a fost proiectata pe baza metodologiei traditionale, construind mai intai o diagrama de stare, pe baza careia am implementat-o mai apoi in vhdl cu ajutorul unui case. Am pus aceasta diagrama in partea stanga iar acum vreau sa explic ce am facut in acest fisier vhdl. Pentru inceput exista o stare de idle in care se intra chiar dupa reset ca sa se poata sincroniza toate componentele, aici se initializeaza numarul de iteratii pe care trebuie sa il faca programul pentru a realiza impartirea. Urmeaza mai apoi o stare de initializare unde se incarca parlel registrele de deplasare si registru de incarcare paralela, iegistrele de incarcare paralela cu impartitorul iar registrele de incarcare paralela cu deimpartitul. Mai departe urmeaza starea in care regstrul a se shifteaza cu 4 poziti pentru a face loc cifrei cele mai semnificative in el. In etapa ce urmeaza se face o scadere repetata pana cand registrul a devine negativ. In tot acest timp o variabila se incrementeaza pentru a observa cifra care trebuie adaugata la cat. Cand registrul a devine negativ se realizeaza o adunare pentru a-l aduce inapui la zero si se decrementeaza variabila de cifra deorece cifar care trebuie inserata este cu o valoare mai mica. Dupa aceasta etapa se realizeaza deplasarea si incarcarea noii cifre in registrul q pentru a putea construii catul; mai apoi urmand ca cifra care contine numarul de iteratii sa se decrementeze. Cand aceasta cifra ajunge la 0 inseamna ca algoritmul s-a terminat si impartirea s-a reallizat cu succes. Aceasta metoda de impartire a fost cea mai usor de implementat unitatea logica fiind foarte usor de scris deoarece nu sunt un multe stari de tranzitie. Chiar daca a fost cel mai usor de implementat nu inseamna ca este si cea mai eficienta deoarece ea face foarte multe operatii de adunare inutile pentru a readuce restul partial inapoi la 0. De aici ii vine numele de impartire cu refaccerea restului partial, deoarece ea tot face scaderi repetate pana cad restul partial devine negativ, iar mai apoi aduna pentru a-l aduce inapoi la 0.



Unitatea de control pentru impartirea fara refacerea restului

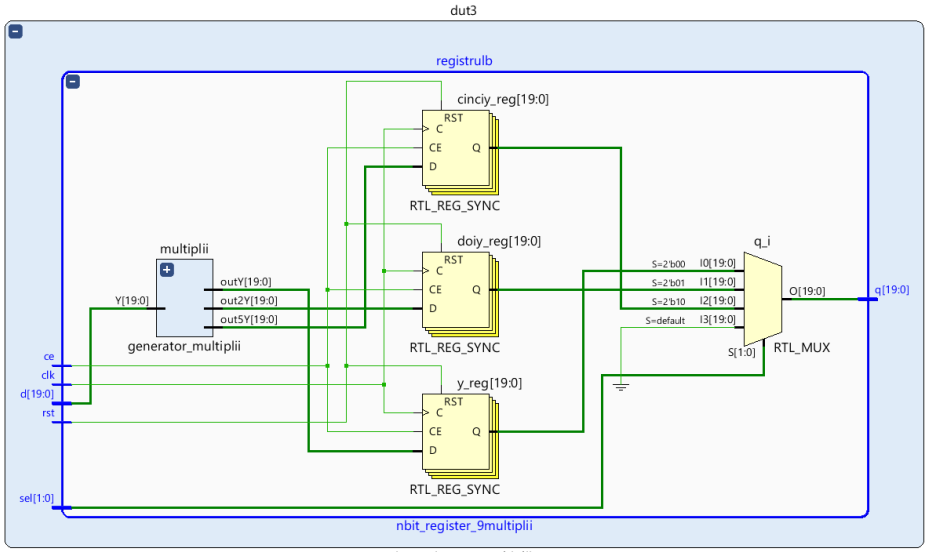


Se poate observa mai sus diagrama de stare a acestei metode care este putin mai eficienta ca si metoda cu refacerea restului partial. Si aceasta componenta a fost proiectata pe baza metodologiei traditionale, construind mai intai o diagrama de stare, pe baza careia am implementat-o mai apoi in vhdl cu ajutorul unui case.

Prima stare a acestui automat moore este o stare idle in care se asteapta apasarea butonului de start. Dupa apasarea acestui buton, automatul intra in starea de initializare unde se incarca parlel registrele de deplasare si registru de incarcare paralela, iegistrele de incarcare paralela cu impartitorul iar registrele de incarcare paralela cu deimpartitul. Tot in aceasta stare se initializeaza numarul de iteratii pe care trebuie il faca programul. Mai departe urmeaza starea in care regstrul a se shifteaza cu 4 poziti pentru a face loc cifrei cele mai semnificative in el. Tot aici se initializeaza cifra care trebuie introdusa in cat.

Urmatoarea stare verifica daca semnul anterior a lui a este diferit de semnul actual a lui a iar in functie de asta se fac anumite lucruri. Mai apoi verificam daca registrul a este negativ, atunci facem o inmultire repetata, daca nu o scadere repetata. Toate acestea se repeta atata timp cat semnu lui a si semnul anterior a lui a sunt la fel. In functie de bs verificam daca in numar trebuie sa introducem cifra minus 1 sau 10 minu cifra calculata. Toata aceasta logica se datoreaza faptului ca nu vrem sa calculam un rest partial degeaba. Mai apui se trece in starea de atribuire cat unde se shifteaza registrul q si se introdce in el o noua cifra a catului. Toate aceste operatii se executa atata timp numarul care reprzinta numarul de iteratii este diferit de 0. Daca este zero va intra intr-o stare se stop.

Registru n biti pentru impartrea cu 9 multiplii



Acesta functioneaza pe acelasi principiu cu registru normal atata tot ca mai are o intrare de selectie pentru a putea selecta multiplii pe care vrem sa ii afisam. In alte cuvinte atunci cand il incarcam paralel, deneram si un multiplu de 2 si unul de 5 de care avem nevoie la imartirea cu 9 multiplii, deoarece ca sa salvam memorie nu generam toti multiplii ci doar 3. Cand selectia este 00 pe iesire o sa apara multiplu de 1, cand este 01 o sa apara multiplul de 2 iar cnd ete 10 o sa apara multiplul de 5. Nu in ultimul larnd ca sa nu am cateva probleme la functionare atunci cand selectam 11 pe iesire o sa apara valoarea 0 pe n biti pe care o folosesc atunci cand nu trebuie sa fac o operatie de adunare sau de scadere. In componenta lui sunt 3 sumatoare care genereaza acesti multiplii.

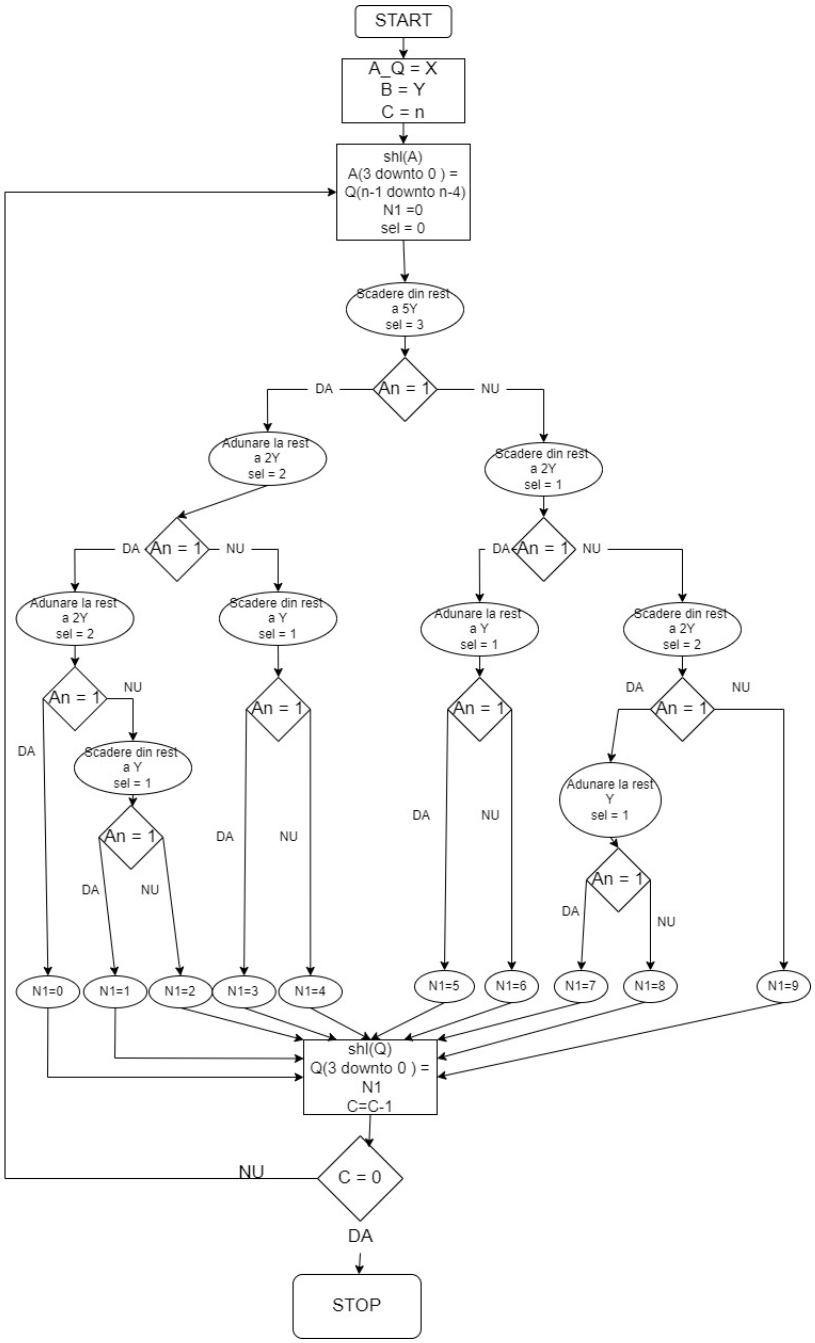
Unitatea de control pentru impartirea cu cei 9 multiplii

Aceasta unitate de control a fost cea mai greu de implementat doarece are cele mai multe stari dar are un mare avantaj el fiind ca parcurge repetitiv doar o singura data pentru fiecare iteratie starile nu de mai multe ori sa si in etapele anterioare. Si aceasta componenta a fost proiectata pe baza metodologiei traditionale, construind mai intai o diagrama de stare, pe baza careia am implementat-o mai apoi in vhdl cu ajutorul unui case. Prima stare a acestui automat moore este o stare idle in care se asteapta apasarea butonului de start. Dupa apasarea acestui buton, automatul intra in starea de initializare unde se incarca parlel registrele de deplasare si registru de incarcare paralela, iegistrele de incarcare paralela cu impartitorul iar registrele de incarcare paralela cu deimpartitul. Tot in aceasta stare se initializeaza numarul de iteratii pe care trebuie il faca programul. Mai departe urmeaza starea in care regstrul a se shifteaza cu 4 poziti pentru a face loc cifrei cele mai semnificative in el. Tot aici se initializeaza cifra care trebuie introdusa in cat. Sel va fi in tot acest timp pe 11.

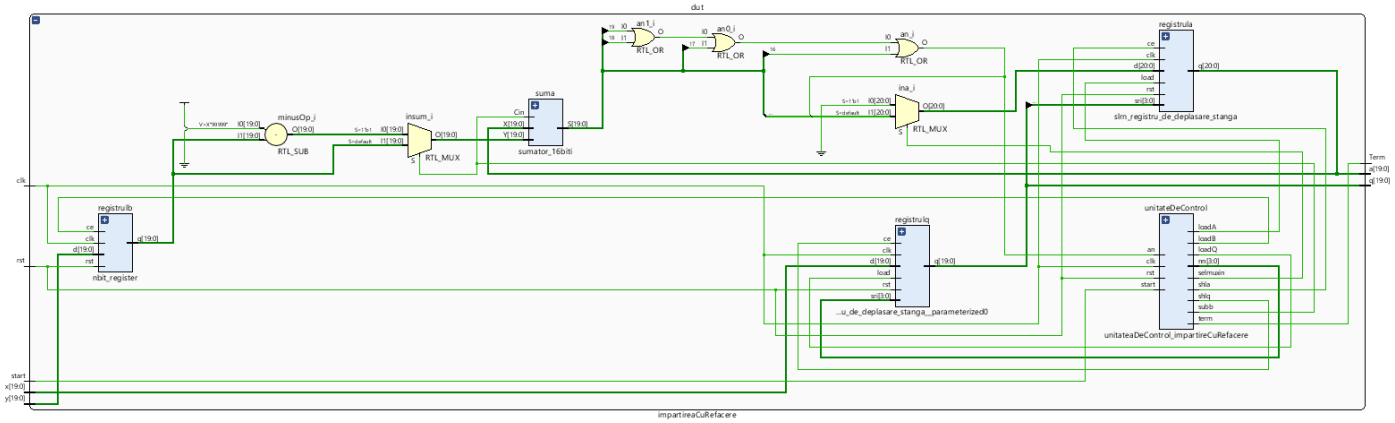
Mai departe urmeaza sa setam selul pe 10 pentru a putea sa scadem multiplul de 5 din a pentru a ne da sseama daca cifra care trebuie atribuita catului este mai mare sau mai mica decat 5. Mai departe daca cifra este mai mica se va efectua o adunare cu multiplul de 2 iar daca nu se va face tot o scadere cu multiplul de 2. Dupa adunarea multiplului de 2 daca in continuare a este negativ se mai aduna inca o data acest multiplu de 2. Daca inca este negativ inseamna ca cifra este 0 iar daca este pozitiv efectuam o scadere cu multiplu de unu pentru a putea alege intre cifar 1 si 2. Daca dupa adunarea primului multiplu de 2 a este pozitiv, atunci se efectueaza o

scadere cu multiplul de unu pentru a apege intre cifra 3 sau 4.Alegerea celorlate cifre sun asemanatoare cu etapele descrise anterior.

In funcie de ramura pe care am luat-o in automatul moore alegem cifra pe care urmeaza sa oa adaugam registrului q care este pe post de cat. Mai apui urmeaza sa verificam daca numarul de iteratii a asuns la 0, daca da inseamna ca algoritmul s-a terminat si impartirea s-a reallizat cu succes, daca nu inseamna ca inca impartirea nu este gata si ca mai trebuie parcurse cateva etape.

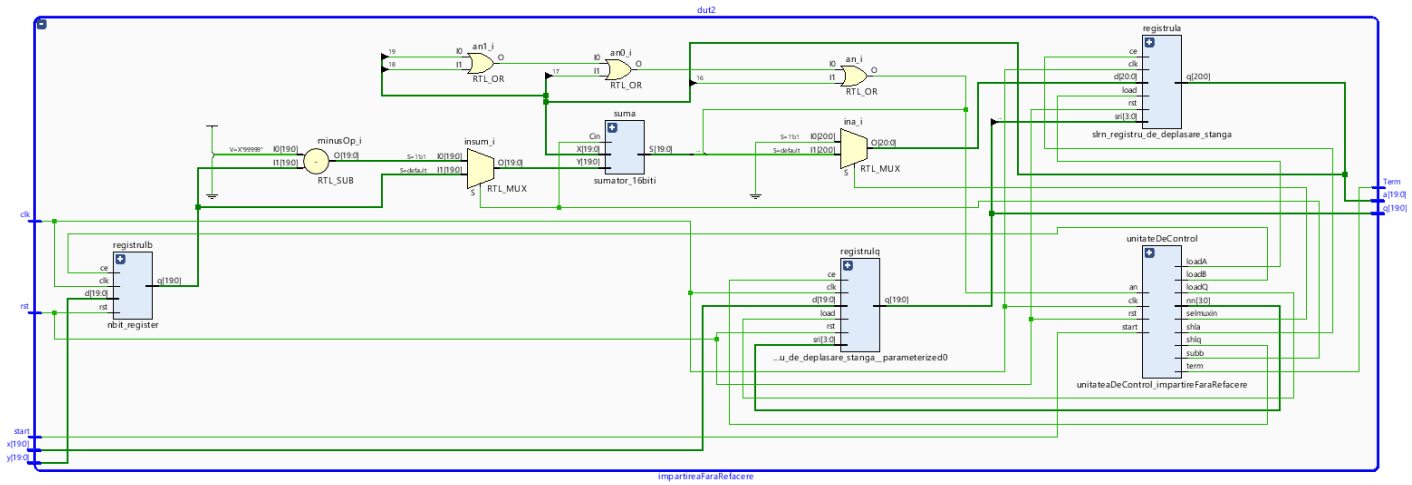


Impartirea cu refacerea restului partial



Se poate observa ca aici sunt legate toate componentele necesare acestei metode, adica un sumator pe 16 biti un registru cu incarcare paralela, doaua registre de deplasare la stanga dar si unitatea logica care opereaza aceasta impartire. Impartitorul se incarca in reistrul b iar deimpartitul in registrul q concatenat cu a. Pentru inceput in a va fi 0 iar in q va fi deimpartitul. Cei mai semnificativi biti a lui q sunt legati ca intrare sri pentruregistrul a iar ca intraer sri pt registrul q este legata cifra generata de unitatea aritmetica si logica. Dupa parcurgerea tuturor iteratiilor, in registrul q se va afla catul iar in registrul a se va afla restul partial. Ca sa sa pot incarca la inceput registrele am adaugat multiplexoare ca sa aleg intre numerele generate in urma operatiei de impartire si x si y primit de la intrare. Toate aceste fnctionalitati se realizeaza cu azutorul unitatii de control. Pentru a folosi doar o un sumator nu si un scazator, am ales sa neg numarul in baza 10 scazand din maximul care poate fi reprezentat pe cinci cifre si acel numar, iar pe cin am pus 1 logic. Am realizat toate aceste lucruri datorita proprietaii scaderi care este o adunare cu al doilea operand negat si 1 pe intrarea de transport. Deoarece numerele sunt pe 2 de biti, am folosit niste porti logice pentru a determina momentul in care in registrul a se afla un numar negativ, adica o poarta sau cu 4 intrari in care am bagat cei mai semnificativi biti ai acestui registru, reusind astfel sa creez intrarea an pentru unitatea de control. Aceasta intrare este foarte importanta deoarece in functie de ea se vor face adunarile si scaderile necesare acestei metode de impartire.

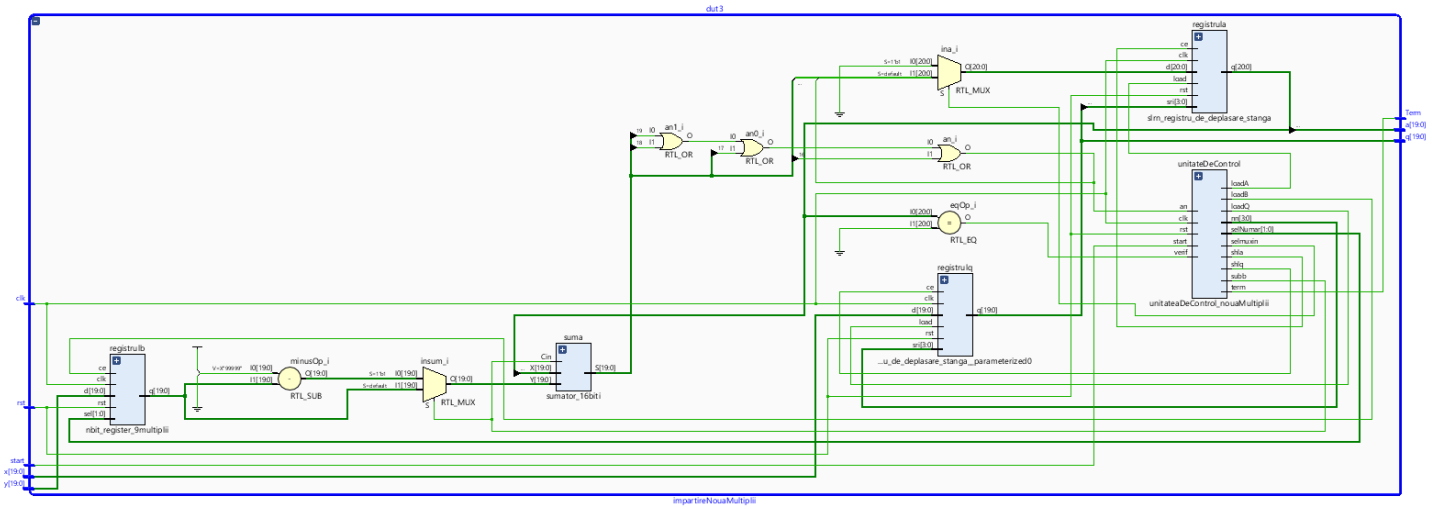
Impartirea fara refacerea restului partial



Dupa cum se poate observa, strructura este asemanatoare cu impartirea cu refacere atata tot difera unitatea de control care este putin mai complexa. Se poate observa ca aici sunt legate toate componentele necesare acestei metode, adica un sumator pe 16 biti un registru cu incarcare paralela, doaua registre de deplasare la stanga dar si unitatea logica care opereaza aceasta impartire. Impartitorul se incarca in reistrul b iar deimpartitul in registrul q concatenat cu a. Pentru inceput in a va fi 0 iar in q va fi deimpartitul. Cei mai semnificativi biti a lui q sunt legati ca intrare sri pentruregistrul a iar ca intraer sri pt registrul q este legata cifra generata de unitatea aritmetica si logica. Dupa parcurgerea tuturor iteratiilor, in registrul q se va afla catul iar in registrul a se va afla restul partial.

Ca sa sa pot incarca la inceput registrele am adaugat multiplexoare ca sa aleg intre numerele generate in urma operatiei de impartire si x si y primit de la intrare. Toate aceste fnctionalitati se realizeaza cu azutorul unitatii de control. Pentru a folosi doar o un sumator nu si un scazator, am ales sa neg numarul in baza 10 scazand din maximul care poate fi reprezentat pe cinci cifre si acel numar, iar pe cin am pus 1 logic Deoarece numerele sunt pe 20 de biti, am folosit niste porti logice pentru a determina momentul in care in registrul a se afla un numar negativ, adica o poarta sau cu 4 intrari in care am bagat cei mai semnificativi biti ai acestui registru, reusind astfel sa creez intrarea an pentru unitatea de control.

Impartirea cu cei 9 multiplii ai impartitorului



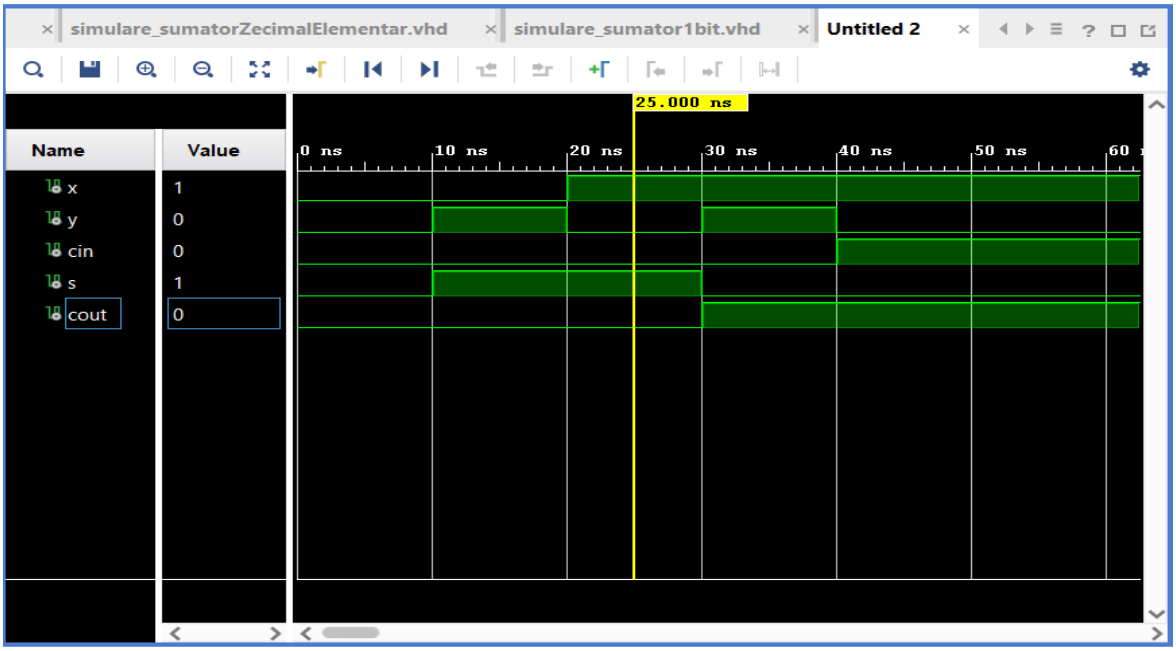
In cele ce urmeaza o sa descri cea mai eficienta metoda de impartire dintre cele implementate. Tot ce difera fata de celelalte este unitatea de control dar si faptul ca in loc de un registru cu incarcare paralela simplu, am folosit unul special, adica acel registru care are in plus o intrae de selectie care selecteaza unul dintre cei 3 multiplii necesari acestei metode.

Si aici ca sa sa pot incarca la inceput registrele am adaugat multiplexoare ca sa aleg intre numerele generate in urma operatiei de impartire si x si y primit de la intrare. Toate aceste fnctionalitati se realizeaza cu azutorul unitatii de control. Pentru a folosi doar o un sumator nu si un scazator, am ales sa neg numarul in baza 10 scazand din maximul care poate fi reprezentat pe cinci cifre si acel numar, iar pe cin am pus 1 logic Deoarece numerele sunt pe 20 de biti, am folosit niste porti logice pentru a determina momentul in care in registrul a se afla un numar negativ, adica o poarta sau cu 4 intrari in care am bagat cei mai semnificativi biti ai acestui registru, reusind astfel sa creez intrarea an pentru unitatea de control.

REZULTATE SI EXPERIMENTE

Sumator pe un bit

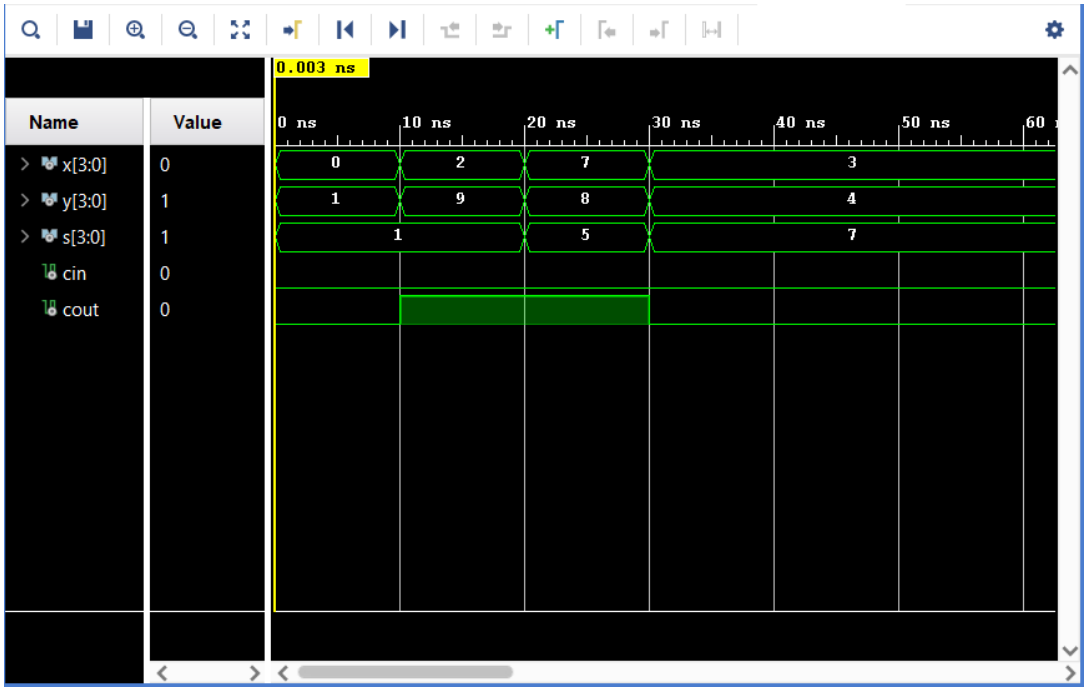
Se poate vedea mai jos ca acest sumator functioneaza corect din toate punctele de vedere doarece unu adunat cu zero este unu iar unu adunat cu unu este doi, doar ca pe desen se poate observa faptul ca se genereaza un transport iar bitul de suma este setat pe 1. Tota acelasi luceu se intampla si daca bitul de trnsport care intra in sumator este setat pe unu, o adunare elementara.



Sumatorul zecimal elementar

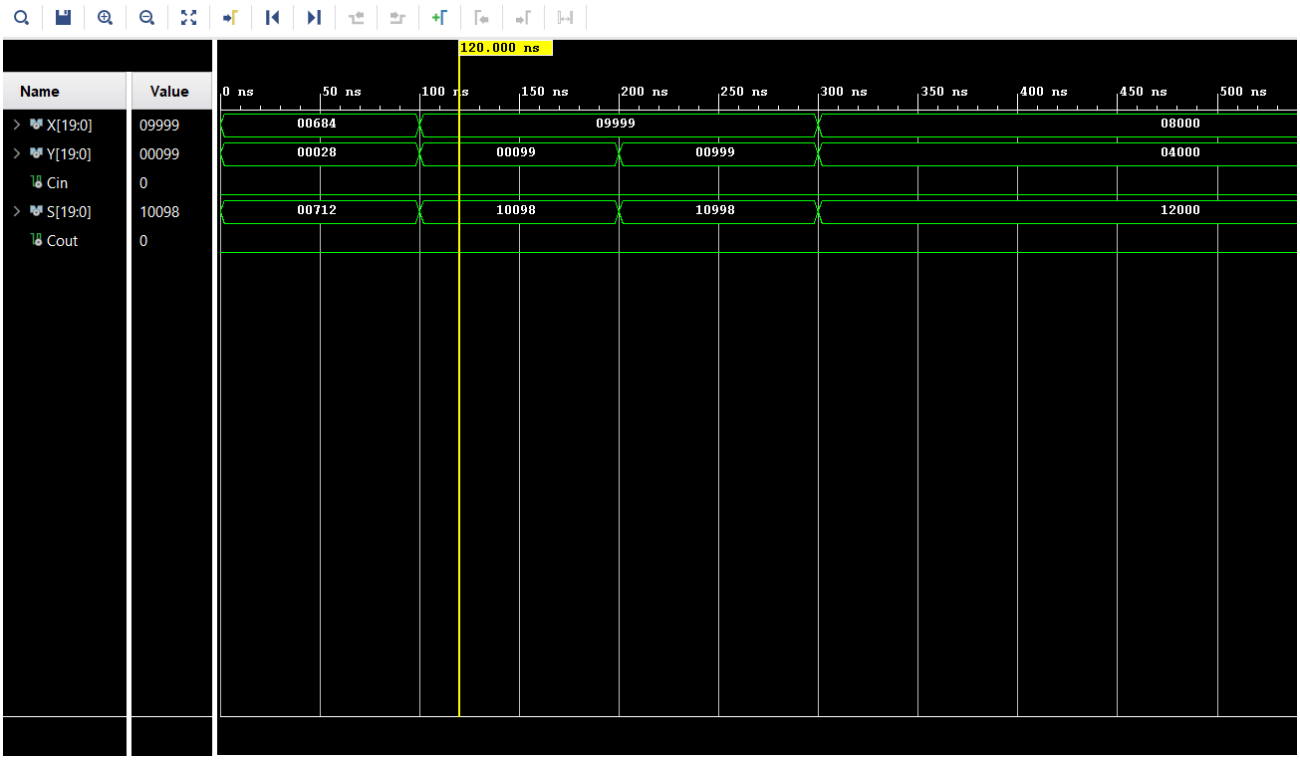
Se poate oberva ca acest sumator zecimal care chiar daca este compus din 2 sumatoare binare lucreaza in baza zece, adica spre exemplu daca adunam doi cu noua in loc sa ne dea B va da 1 si bitul de transport va fi setat la 1 ceea ce ne duce la

gandul ca rezultatul este 11, daca alipim cei bitii respectivi. Mai multe detalii se pot oberva si pe simularea generata mai jos.



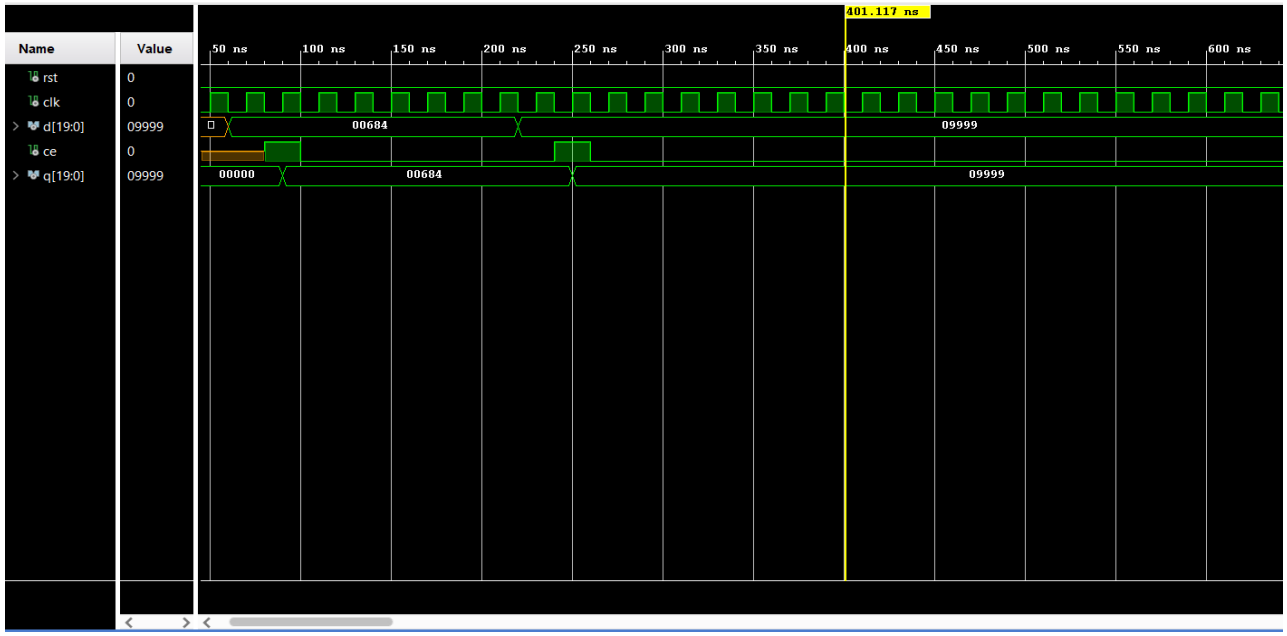
Sumator zecimal 16 biti

Mai departe putem spune ca acest sumator functioneaza bine din toate punctele de vedere chiar daca este compus din 5 sumatoare zecimale elementare. Putem observa ca depasirea se salveaza lipit de numarul pe care vrem sa il adunam, adica 4 seturi de cate 4 biti este suma iar primul set de 4 biti o folosesc pe post de depasire in calcularea sumei finale, am facut acest lucru deoarece mi s-a parut mult mai usor de implementat impartirea in acest mod.



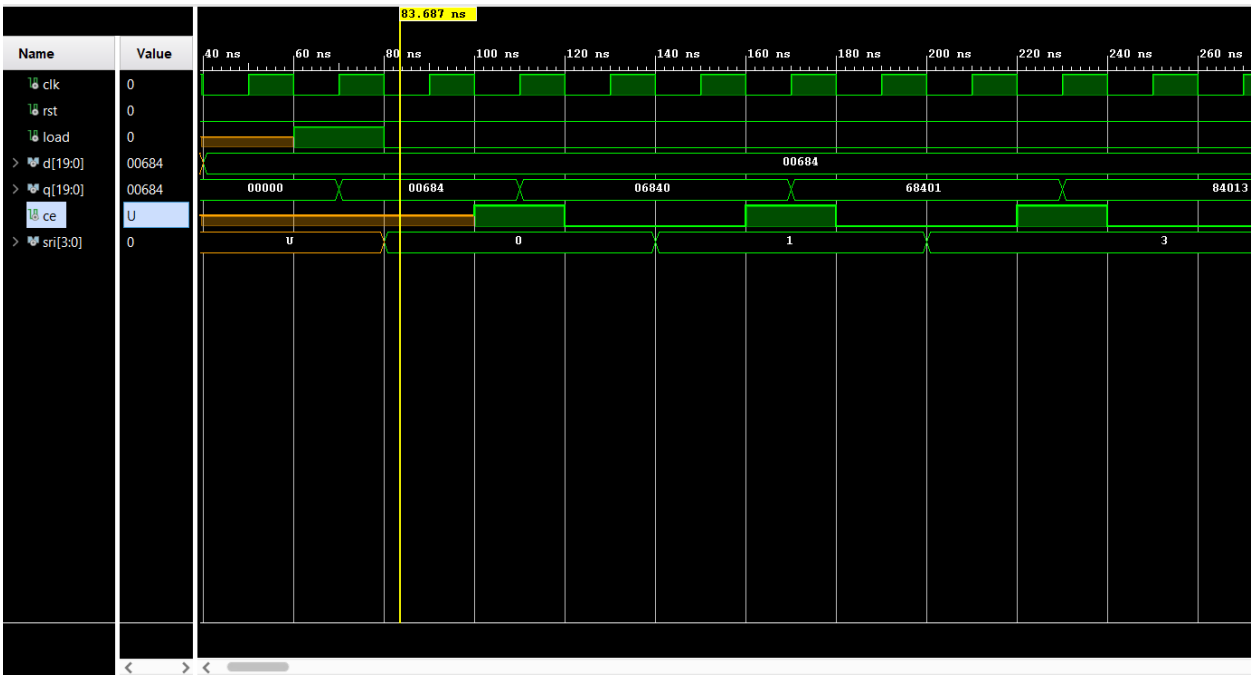
Registru n biti cu resetare sincrona

In figura de mai jos se poate observa functionalitatea de incarcare paralela a acestui registru atunci cand ce este adus de la 0 logic la 1 logic, si se poate observa ca iesirea este pastrata mai multe perioade de ceas, atata timp cat nu se fac modificari la intrarile rst si ce. De exemplu, o sa putem sa punem ce vrem noi pe intrarea d atata timp cat nu facem modificari pe ce sau pe rst, nu o sa se vada vreo modificare a iesirii q.

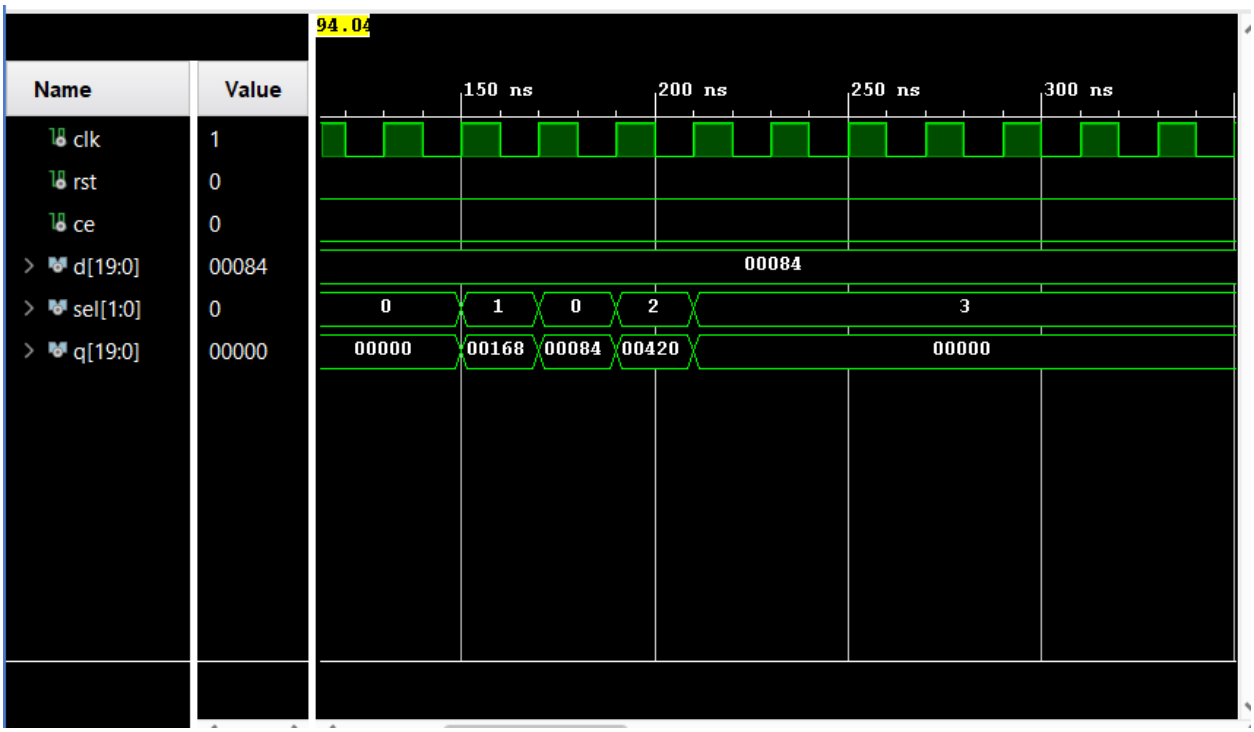


Registru de deplasare la stanga de n biti

In urmatoarea simulare se poate observa toate functionalitatile acestui registru de deplasare la stanga cu 4 pozitii, si incarcarea lui paralela cu intrarea sri, asfel pentru inceput incarcam 684 in el iar ai apoi il deplasam si incarcam valoarea 0, apoi 1, apoi 3, urmand ca numarul la final sa fie 84013. Putem observa ca chiar daca intrarea lui este 684, aceasta nu influenteaza iesirea lui atata timp cat load sau rst nu sunt pe 1 logic. Pe acelasi principiu functioneaza si ce, sri se poate modifica oricand si oricat, atata timp cat ce nu este pe 1 logic, acest registru nici nu se va deplasa nici nu se va incarca paralel cu intrarea seriala sri. In aceasta simulare nu am exemplificat cum functioneaza intrarea de reset, am ales sa exemplific in cuvinte pentru a nu incarca imaginea si pentru a fi mult mai usor de citit.

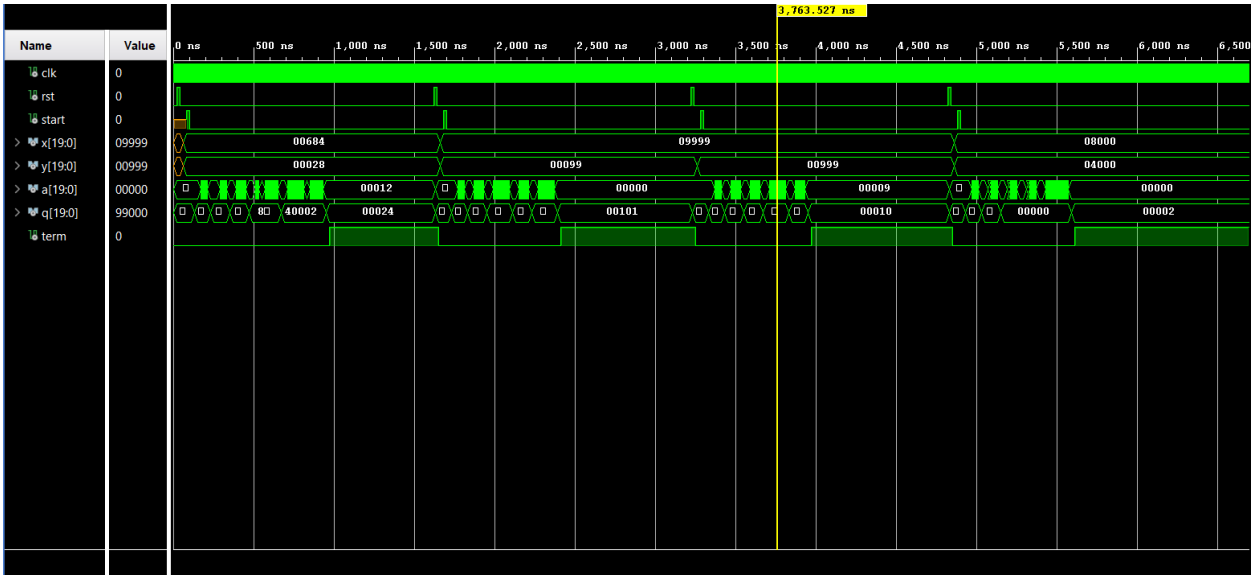


Registru n biti pentru impartrea cu 9 multiplii



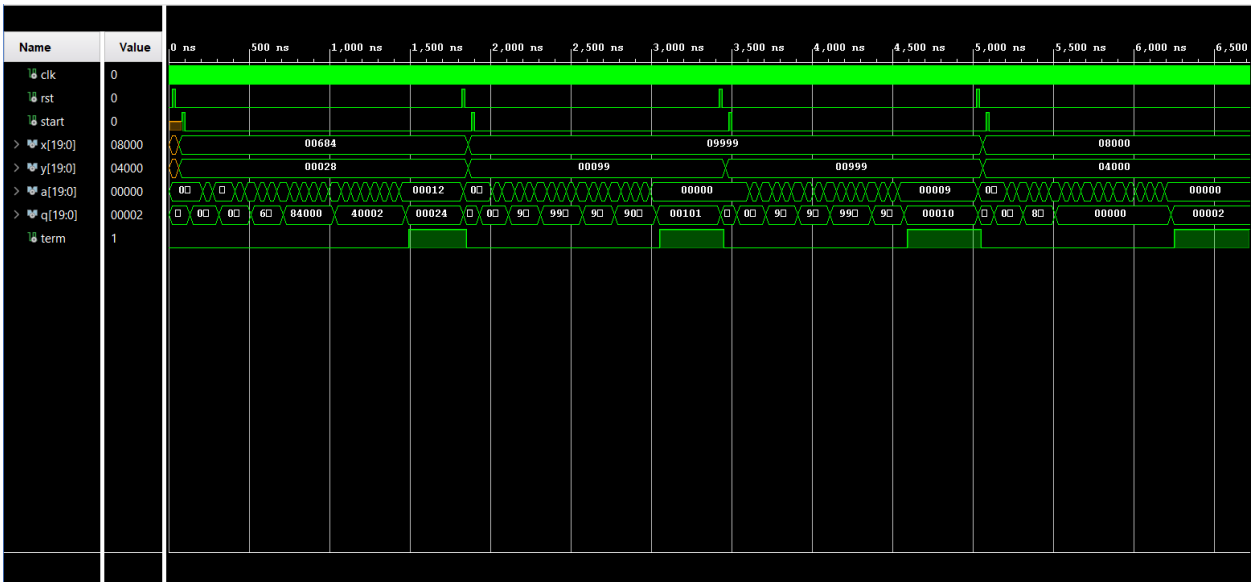
In poaza de mai sus se poate observa toate starile prin care trece acest registru cu incarcare paralela si reset sincron. Dupa cum se intelege de mai sus, acest registru este incarcat paralel cu 84 pe frontul crescator al semnalului de ceas atunci cand ce este adus la 1 logic, iar in functie de selectia pe care o facem o sa apara pe iesire ori numarul in sine ori un multiplu de 2 care in cazul nostru este 168 ori un multiplu de 5 care in cazul nostru est 420, iar daca selectia este 3 atunci o sa apara valoarea 0.

Impartirea cu refacerea restului partial



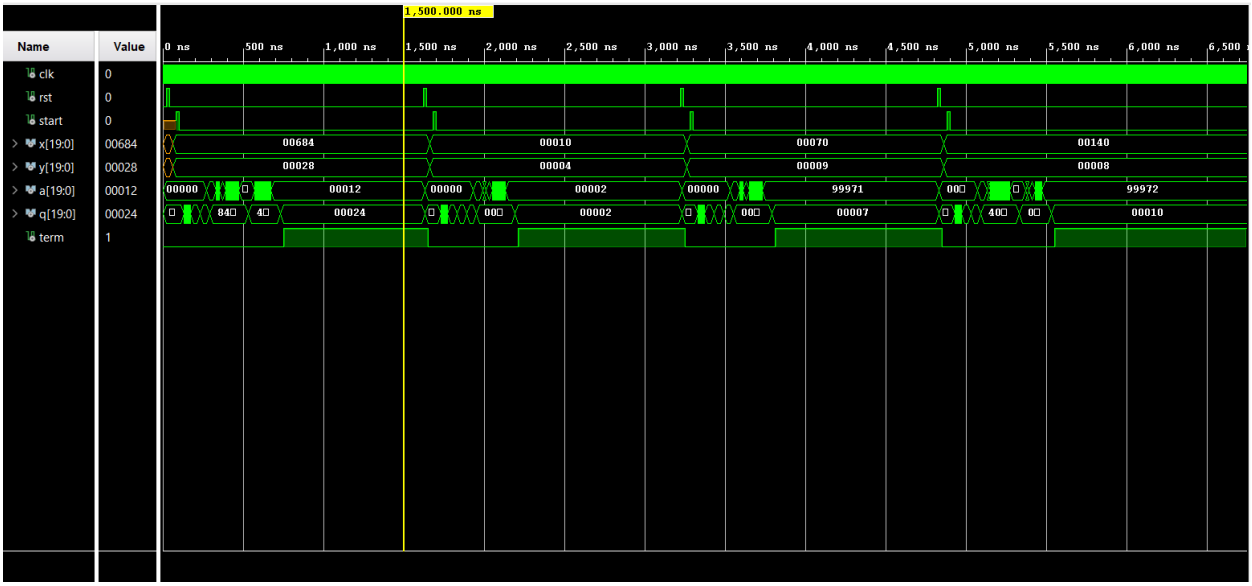
Dupa cum se poate observa metoda aceasta de impartire functioneaza corect in totalitate dar se poate observa ca trece printr-o multime de stari intermediare pana sa genereze rezultatul final ceea ce o face sa fie o inmultire nu asa de eficienta dar usor de implementat. Dupa cum se poate observa restul se afla in registrul a iar catul in registrul q, astfel 648 impartit la 28 obervam ca da restul 12 si catul 24. Pentru aceast rezultat metoda parcurge o multime de stari ale automatului. Din motive de implementare, chiar daca aceasta metoda reface restul de fiecare data, este putin mai rapida ca si metoda fara refacere dar in reealitaie nu este asa.

Impartirea fara refacerea restului partial



Aceasta metoda de impartire este mai eficienta ca si metoda cu refacere, si dupa cum se poate observa imparte corect numerele genernd tot in registrele a si q restul respectiv catul operatie de impartire. Se poate oberva ca resetul este sincron dar si startul, operatia pornind odata cu apasarea butonului de start dar si odata cu venirea unui front crescator de ceas. Iesirea term este 1 atat timp cat operatia este terminata si nu apar numere noi pe intrare, acest lucru il folosesc pentru a demonstra ca operatia a avut loc cu succes. Dupa cum se poate observa restul se afla in registrul a iar catul in registrul q, astfel 648 impartit la 28 obervam ca da restul 12 si catul 24. Pentru aceast rezultat metoda parcurge o multime de stari ale automatului. Din motive de implementare, chiar daca aceasta metoda nu reface restul partial de fiecare data, este putin mai lenta ca si metoda cu refacerea restului partial dar in realitaie nu este asa. Aceasta metoda a fost destul de greu de implementat deoarece am avut nevoie si de semnul actual a lui a si de semnul anterior a lui a pentru a putea iesi din iteratia respectiva dar si pentru a putea stii ce cifra trebuie sa inseram in q. Dupa cum se poate observa functioneaza si pe numere mai mari spre exemplu 8000 si la final genereaza tot un rezultat corect

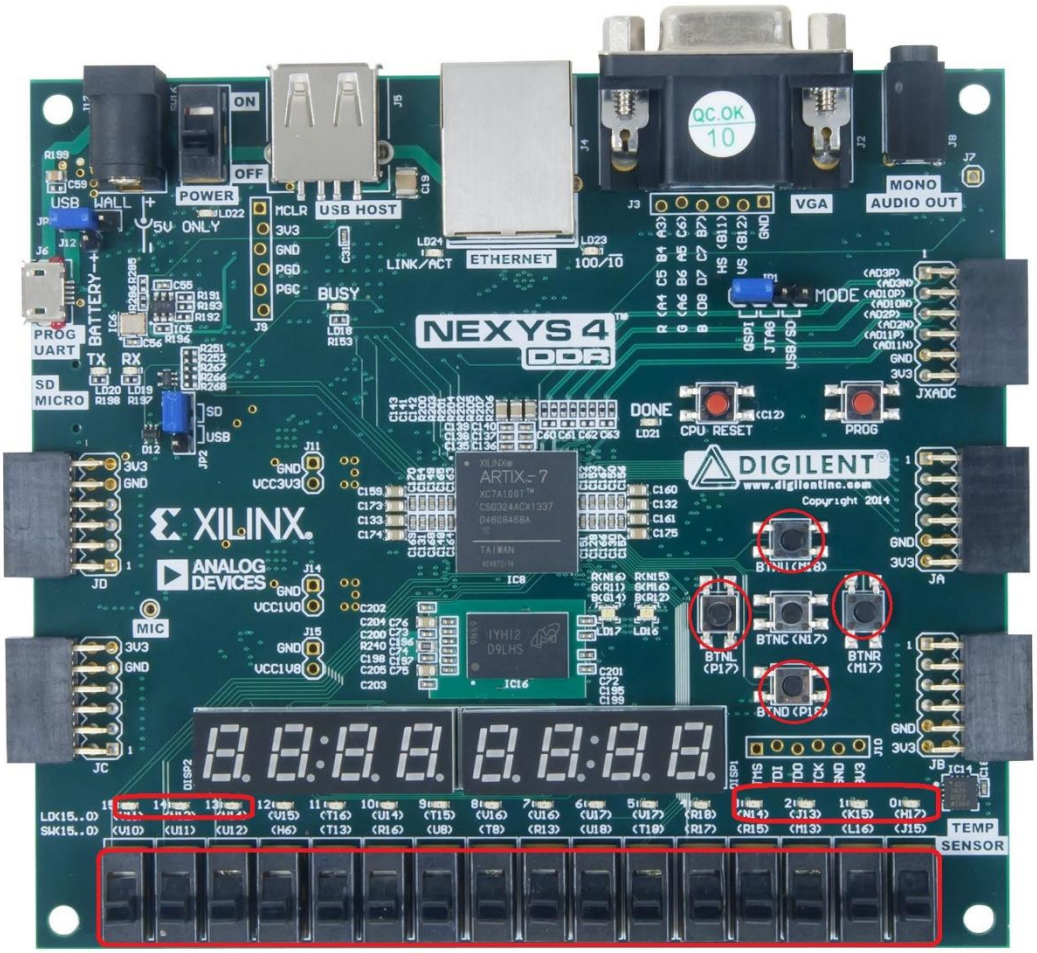
Impartirea cu cei 9 multiplii ai impartitorului



Aceasta metoda este o metoda foarte rapida dupa cum se vede si din simulare, si este si eficienta din punct de vedere a memoriei, deoarece nu am generat toti multiplii lui y ci doar 2 dintre ei, deoarece prin combinarea lor se pot forma restul de multiplii. A fost cea mai grea impartire de implementat, deoarece are o unitate de control destul de complicata. Am incercat foarte mult sa o fac sa mearga corect in totalitate dar nu am reusit, dupa cum se poate observa si din simulare pe numere mici functioneaza corect, dar numerele mari ii dau mari batai de cap. Cu toate aste am reusit sa demonstrez ca aceasta este cea mai rapida metoda de impartire dintre toate metodele implementate de mine.

MOD DE UTILIZARE

Modul de utilizare al metodelor de impartire este destul de intuitiv, utilizatorii au posibilitatea de a introduce cele 2 numere de la placa cu ajutorul celor 16 intrerupatoare. Deoarece avem 16 intrerupatoare pe care putem introduce un singur numar de 4 cifre, am ales urmatoarea implementare pentru a putea introduce ambele numere: in momentul in ne jucam cu switch-urile vom vedea ca numarul va fi afisat doar pt x, in momentul in care apasam BTNL, doar atunci se va memora in y valoarea de pe acestea, apoi putem modifica din nou switch-urile pentru primul numar, apoi putem apasa BTND pentru a reseta componentele, dupa care se poate apasa BTNU pentru a porni impartirile. De la BTNR putem schimba metoda de impartire pe care vrem sa o afisam, se va aprinde cate un led ce reprezinta in ce operatie ne situam. Ma jos se poate vedea o poza cu ce butoane trebuie folosite pentru a putea testa acest proiect pe placa Nexys4.



CONCLUZII

Aspecte generale

Solutia propusa si implementata rezolva majoritatea situatiilor de inmultire zecimala care pot fi implementate pe o placuta Nexys4 . Deoarece resursele placii sunt destul de limitate a fost destul de greu sa gandesc o metoda de a implementa introducerea numerelor pentru operatii. Ar fi mult mai usor daca as incarca din memorie numerele pentru a nu trebui sa le introduc eu de la switchuri.

Rezultatul nu este in totalitate ce am vrut eu deoarece nu am reusit sa fac o unitate de control destul de buna pentru impartirea cu noua multiplii. Poate daca aveam mai mult timp reuseam in totalitate sa fac ceea ce mi am propus.

Dezvoltari ulterioare

In randurile ulterioare as vrea sa enumar cateva idei de dezvoltare ulterioare ale sistemului:

Conectarea a unei tastaturi cu Pmode.

Salvarea numerelor si a rezultatelor in memorie.

Afisarea pe un monitor a rezultatelor cu ajutorul unui controller VGA.

Stocarea rezultatelor in memorie.

BIBLIOGRAFIE

https://users.utcluj.ro/~baruch/ro/pages/publicatii/carti-si-manuale/sscae--cuprins.php

https://users.utcluj.ro/~baruch/ssc/labor/Automate-Stare.pdf

https://users.utcluj.ro/~baruch/book_ssce/SSCE-Basic-Division.pdf

https://users.utcluj.ro/~baruch/ssc/labor/Aritm-Secventiala.pdf

https://docs.google.com/presentation/d/1uk3-Y6hyoLVcqfiI8F0eMsN1N0YeaOhG/edit#slide=id.p19

https://users.utcluj.ro/~baruch/media/ssc/curs/SSC-Impartire.pdf

https://ro.wikipedia.org/wiki/Sistem_binar#%C3%8Emp%C4%83r%C8%9Birea_%C3%AEn_binar

http://people.ee.duke.edu/~sorin/ece152/lectures/3.3-arith.pdf

https://media.digikey.com/pdf/Reference%20Design/Digi-Key%20DDS%20group/DKAN0003A_BCD_Division.pdf

https://patents.google.com/patent/US4599702A/en

https://www.allaboutcircuits.com/technical-articles/basic-binary-division-the-algorithm-and-the-vhdl-code/

https://www.cs.utah.edu/~rajeev/cs3810/slides/3810-08.pdf

https://uomustansiriyah.edu.iq/media/lectures/5/5_2018_12_17!07_25_39_PM.pdf

https://www.hzu.edu.in/csit/Computer%20Arithmetic%20%20computer%20fundamentals.pdf

https://www.cise.ufl.edu/~mssz/CompOrg/CDA-arith.html

https://www.youtube.com/watch?v=Iw5J7UsA_kc&list=PL7kkolCtIBKLukrBsEDwKRTE64JvaJDhM&index=58&ab_channel=LBEbook

https://www.youtube.com/watch?v=jcUIETi-VtM&list=PL7kkolCtIBKLukrBsEDwKRTE64JvaJDhM&index=59&ab_channel=LBEbooks

ANEXA

Sumator1bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity sumator1bit is
    Port ( X : in STD_LOGIC;
           Y : in STD_LOGIC;
           Cin : in STD_LOGIC;
           S : out STD_LOGIC;
           Cout : out STD_LOGIC);
end sumator1bit;

architecture Behavioral of sumator1bit is

begin
S <= X xor Y xor Cin;
Cout <=  (X and Y) or (X and Cin) or (Y and cin);

end Behavioral;
```

Sumator4biti

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity sumator4biti is
    Port ( X : in STD_LOGIC_VECTOR (3 downto 0);
           Y : in STD_LOGIC_VECTOR (3 downto 0);
           Cin : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           Cout : out STD_LOGIC);
end sumator4biti;

architecture Behavioral of sumator4biti is
signal carry: std_logic_vector(3 downto 0):= "0000";
begin
GEN_ADD: for i in 0 to 3 generate
    LOWER_BIT: if I=0 generate
      U0: entity WORK.sumator1bit port map(
         x(i),y(i),cin,s(i),carry(i));
    end generate LOWER_BIT;

    UPPER_BITS: if I > 0 generate
      UX: entity WORK.sumator1bit port map(
         x(I),y(I),carry(i-1),s(i),carry(i));
    end generate UPPER_BITS;

end generate GEN_ADD;
```

```
Cout<= Carry(3);
end Behavioral;
```

SumatorZecimal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity SumatorZecimal is
 Port ( X : in STD_LOGIC_VECTOR (3 downto 0);
        Y : in STD_LOGIC_VECTOR (3 downto 0);
        Cin : in STD_LOGIC;
        S : out STD_LOGIC_VECTOR (3 downto 0);
        Cout : out STD_LOGIC);
end SumatorZecimal;

architecture Behavioral of SumatorZecimal is
signal sum,sum1 : std_logic_vector(3 downto 0) := "0000";
signal c,cc,ccc: std_logic:='0';
begin
dut1: entity WORK.sumator4biti port map(x,y,cin,sum,c);
ccc <= c or (sum(3) and sum(2)) or (sum(3) and sum(1));
sum1 <= '0'& ccc & ccc & '0';

dut2: entity WORK.sumator4biti port map(sum1,sum,'0',s,cc);
```

```
cout<=ccc;
```

sumator_16biti

```
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



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;

architecture Behavioral of sumator_16biti is
signal carry: std_logic_vector(4 downto 0):="00000";
signal sum : std_logic_vector(19 downto 0):=x"00000";

begin
dut1: entity WORK.SumatorZecimal port map(x=>x(3 downto 0),y=>y(3 downto
0),
```

```
cin=>cin,s=>sum(3 downto 0),cout=>carry(0));

dut2: entity WORK.SumatorZecimal port map(x=>x(7 downto 4),y=>y(7 downto 4),
cin=>carry(0),s=>sum(7 downto 4),cout=>carry(1));

dut3: entity WORK.SumatorZecimal port map(x=>x(11 downto 8),y=>y(11 downto 8),
cin=>carry(1),s=>sum(11 downto 8),cout=>carry(2));

dut4: entity WORK.SumatorZecimal port map(x=>x(15 downto 12),y=>y(15 downto 12),
cin=>carry(2),s=>sum(15 downto 12),cout=>carry(3));

dut5: entity WORK.SumatorZecimal port map(x=>x(19 downto 16),y=>y(19 downto 16),
cin=>carry(3),s=>sum(19 downto 16),cout=>carry(4));

s<=sum;
cout<=carry(4);



end Behavioral;
```

nbit_register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity nbit_register is
  GENERIC ( n : natural ) ;
  Port (
    d: in std_logic_vector(n-1 downto 0);
    ce: in std_logic;
    clk: in std_logic;
    rst: in std_logic;
    q: out std_logic_vector(n-1 downto 0)
   );
end nbit_register;

architecture Behavioral of nbit_register is

begin
registru: process(d,ce,rst,clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
```

```
            q <= (others => '0');
         elsif ce = '1' then
            q <= d;
        end if;

    end if;
end process registru;


end Behavioral;
```

slrn_registru_de_deplasare_stanga

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity slrn_registru_de_deplasare_stanga is
GENERIC ( n : natural ) ;
  Port (
    d: in std_logic_vector(n-1 downto 0);
```

```
    sri: in std_logic_vector(3 downto 0);
    load: in std_logic;
    ce: in std_logic;
    clk: in std_logic;
    rst: in std_logic;
    q: out std_logic_vector(n-1 downto 0)
   );
end slrn_registru_de_deplasare_stanga;

architecture Behavioral of slrn_registru_de_deplasare_stanga is
signal iesire: std_logic_vector(n-1 downto 0):=(others => '0');
begin
registru: process(d,sri,load,ce,rst,clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            iesire <= (others => '0');
         elsif load = '1' then
            iesire <= d;
         elsif ce = '1' then
          iesire <=  iesire(n-5 downto 0) & sri;
        end if;

    end if;
end process registru;
```

```
q <= iesire;

end Behavioral;
```

unitateaDeControl_impartireCuRefacere

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;



entity unitateaDeControl_impartireCuRefacere is
generic(n: natural);
  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)
  );
```

```
end unitateaDeControl_impartireCuRefacere;

architecture Behavioral of unitateaDeControl_impartireCuRefacere is

type tip_stare is (idle,initializare,shiftareA,scadere,decizie,adunare,atribuirecat,comparare,stop);
signal stare: tip_stare;
signal cn: natural:=5;
signal t: std_logic:='0';
signal n1 : std_logic_vector(3 downto 0):="0000";
begin


process(clk,rst,start,an)
begin
   if rising_edge(clk) then
      if rst = '1' then
         stare<=idle;
       end if;

       case stare is
          when idle =>
                              t <= '0';
                              loada <= '0';
                              loadb <= '0';
                              loadq <= '0';
```

```
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';
                n1<=(others =>'0');
                nn<="0000";
                cn <= 5;

                if start = '1'  then
                   stare <= initializare;
                else
                   stare <= idle;
                end if;
when initializare =>
                loada <= '1';
                loadb <= '1';
                loadq <= '1';
                selmuxin <= '1';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                n1<="0000";
                stare <= shiftareA;
when shiftareA =>
                loada <= '0';
                loadb <= '0';
```

```
                loadq <= '0';
                shla <= '1';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';
                n1<="0000";
                stare <= scadere;

when scadere =>
                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '1';
                selmuxin <= '0';

                n1<= n1 + 1;

                stare <= decizie;
when decizie =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
```

```
                subb <=  '0';
                selmuxin <= '0';

                   if(an ='1') then
                      stare <= adunare;
                   else
                      stare <= scadere;
                   end if;
when adunare =>
                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                n1<= n1 - 1;

                stare <= atribuirecat;
when atribuirecat =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '1';
                subb <=  '0';
                nn<=n1;
```

```
                selmuxin <= '0';

                cn <= cn - 1;

                stare <= comparare;
when comparare =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';

                if cn = 0 then
                      stare <= stop;
                else
                       stare <= shiftareA;
                end if;
when stop =>
                t <= '1';
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
```

```
                    subb <=  '0';
                    selmuxin <= '0';
        when others => stare <= idle;


    end case;

  end if;
end process;
term <= t;
end Behavioral;
```

impartireaCuRefacere

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;


entity impartireaCuRefacere 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 impartireaCuRefacere;

architecture Behavioral of impartireaCuRefacere is


signal loada,loadb,loadq,shla,shlq,subb,ovf,selmuxin:  std_logic :='0';
signal an,outsuman: std_logic:='1';
signal outb,insum,outsum,outq: std_logic_vector(n-1 downto 0);
signal outa,ina: std_logic_vector(n downto 0);
signal nn: std_logic_vector(3 downto 0):=(others =>'0');

begin

unitateDeControl: entity work.unitateaDeControl_impartireCuRefacere generic
map ( n => n) port map (
          start=>start,clk=>clk,rst=>rst,
          an=>an,selmuxin=>selmuxin,loada=>loada,loadb=>loadb,
          loadq=>loadq,shla=>shla,shlq=>shlq,subb=>subb,
          term=>term,nn=>nn);
```

```
registrulb: entity work.nbit_register generic map(n => n) port map(
    d=>y,
    ce=>loadb,
    clk=>clk,
    rst=>rst,
    q=>outb);


insum <= x"99999" - outB when subb = '1' else outB;

suma: entity WORK.sumator_16biti
            port map(
            x=>outa(n-1 downto 0),
            y=>insum,
            cin=>subb,
            s=>outsum,
            cout=>outsuman);

an<=outsum(n-1) or outsum(n-2) or outsum(n-3) or outsum(n-4) ;
ina <= '0' & x"00000" when selmuxin = '1' else an & outsum;

registrula: entity WORK.slrn_registru_de_deplasare_stanga
      generic map(n => n+1) port map(
            d => ina,
            sri=> outq(n-1 downto n-4),
            load=>loada,
```

```
            ce=>shla,
            clk=>clk,
            rst=>rst,
            q=>outa);


registrulq: entity WORK.slrn_registru_de_deplasare_stanga
      generic map(n => n) port map(
            d => x,
            sri=> nn,
            load=>loadq,
            ce=>shlq,
            clk=>clk,
            rst=>rst,
            q=>outq);

q<=outq;
a<= outa(n-1 downto 0);



end Behavioral;
```

unitateaDeControl_impartireFaraRefacere

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;


entity unitateaDeControl_impartireFaraRefacere is
generic(n: natural);
  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)
  );
end unitateaDeControl_impartireFaraRefacere;


architecture Behavioral of unitateaDeControl_impartireFaraRefacere is
type tip_stare is (idle,initializare,shiftareA,decizie1,decizie2,scadere,adunare,atribuirecat,comparare,stop);
signal stare: tip_stare;
signal cn: natural:=5;
```

```
signal t: std_logic:='0';
signal n1 : std_logic_vector(3 downto 0):="0000";
signal BS : std_logic := '1';
begin
process(clk,rst,start,an)
begin
   if rising_edge(clk) then
      if rst = '1' then
         stare<=idle;
       end if;

       case stare is
          when idle =>
                              t <= '0';
                              loada <= '0';
                              loadb <= '0';
                              loadq <= '0';
                              shla <= '0';
                              shlq <= '0';
                              subb <=  '0';
                              selmuxin <= '0';
                              n1<=(others =>'0');
                              nn<="0000";
                              cn <= 4;

                              if start = '1'  then
```

```
                stare <= initializare;
            else
                stare <= idle;
            end if;
when initializare =>
            loada <= '1';
            loadb <= '1';
            loadq <= '1';
            selmuxin <= '1';
            shla <= '0';
            shlq <= '0';
            subb <=  '0';
            n1<="0000";
            BS<='0';
            stare <= shiftareA;
when shiftareA =>
            loada <= '0';
            loadb <= '0';
            loadq <= '0';
            shla <= '1';
            shlq <= '0';
            subb <=  '0';
            selmuxin <= '0';
            n1<="0000";
            stare <= decizie1;
```

```
when decizie1 =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';

                if an = BS then
                   stare <= decizie2;
                else
                   stare <= atribuirecat;
                end if;

when decizie2 =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';

                n1<= n1 + 1;
```

```
                if(an ='1') then
                   stare <= adunare;
                else
                   stare <= scadere;
                end if;
 when scadere =>
                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '1';
                selmuxin <= '0';

                stare <= decizie1;
when adunare =>
                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';

                stare <= decizie1;

when atribuirecat =>
```

```
                if BS = '0' then
                   nn<= n1-2;
                else
                   nn <= "1010" - n1;
                end if;
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '1';
                subb <=  '0';
                selmuxin <= '0';



                stare <= comparare;
when comparare =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';

                cn <= cn - 1;
```

```
                    BS <= an;

                    if cn = 0 then
                        stare <= stop;
                    else
                        stare <= shiftareA;
                    end if;
        when stop =>
                    t <= '1';
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
        when others => stare <= idle;
      end case;
   end if;
end process;
term <= t;

end Behavioral;
```

impartireFaraRefacere

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;


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;

architecture Behavioral of impartireaFaraRefacere is
signal loada,loadb,loadq,shla,shlq,subb,ovf,selmuxin:  std_logic :='0';
signal an,outsuman: std_logic:='1';
signal outb,insum,outsum,outq: std_logic_vector(n-1 downto 0);
```

```
signal outa,ina: std_logic_vector(n downto 0);
signal nn: std_logic_vector(3 downto 0):=(others =>'0');

begin

unitateDeControl: entity work.unitateaDeControl_impartireFaraRefacere generic map ( n => n) port map (
        start=>start,clk=>clk,rst=>rst,
        an=>an,selmuxin=>selmuxin,loada=>loada,loadb=>loadb,
        loadq=>loadq,shla=>shla,shlq=>shlq,subb=>subb,
        term=>term,nn=>nn);

registrulb: entity work.nbit_register generic map(n => n) port map(
   d=>y,
   ce=>loadb,
   clk=>clk,
   rst=>rst,
   q=>outb);


insum <= x"99999" - outB when subb = '1' else outB;

suma: entity WORK.sumator_16biti
            port map(
            x=>outa(n-1 downto 0),
            y=>insum,
```

```
            cin=>subb,
            s=>outsum,
            cout=>outsuman);

an<=outa(n-1) or outa(n-2) or outa(n-3) or outa(n-4) ;
ina <= '0' & x"00000" when selmuxin = '1' else an & outsum;

registrula: entity WORK.slrn_registru_de_deplasare_stanga
      generic map(n => n+1) port map(
            d => ina,
            sri=> outq(n-1 downto n-4),
            load=>loada,
            ce=>shla,
            clk=>clk,
            rst=>rst,
            q=>outa);


registrulq: entity WORK.slrn_registru_de_deplasare_stanga
      generic map(n => n) port map(
            d => x,
            sri=> nn,
            load=>loadq,
            ce=>shlq,
            clk=>clk,
            rst=>rst,
```

```
            q=>outq);

q<=outq;
a<= outa(n-1 downto 0);



end Behavioral;
```

unitateaDeControl_nouaMultiplii

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity unitateaDeControl_nouaMultiplii is
generic(n: natural);
Port (
  verif,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);
  selNumar: out std_logic_vector(1 downto 0)
  );
end unitateaDeControl_nouaMultiplii;

architecture Behavioral of unitateaDeControl_nouaMultiplii is
type tip_stare is(idle,initializare,adunare2YNiv21,
comparare11,comparare12,shiftareA,scadMul5,compNiv1,
adunare2YNiv1,scadere2YNiv1,compFin,atribuireCat,stop,
scadereYNiv22,adunareYNiv23,scadere2YNiv24,comparare21,
comparare22,comparare23,comparare24,scadereYNiv31,
adunareYNiv32,comparare31,comparare32,cifraZero,cifraUnu,
cifraDoi,cifraTrei,cifraPatru,cifraCinci,cifraSase,cifraSapte,
cifraOpt,cifraNoua);
signal stare: tip_stare;
signal cn: natural:=5;
signal t: std_logic:='0';
signal n1: std_logic_vector(3 downto 0):="0000";
begin
process(clk,rst,start,an)
begin
    if rising_edge(clk) then
        if rst = '1' then
            stare<=idle;
```

```
end if;

case stare is
  when idle =>
                        t <= '0';
                        loada <= '0';
                        loadb <= '0';
                        loadq <= '0';
                        shla <= '0';
                        shlq <= '0';
                        subb <=  '0';
                        selmuxin <= '0';
                        n1<=(others =>'0');
                        nn<="0000";
                        cn <= 4;
                        selNumar <= "11";

                        if start = '1'  then
                           stare <= initializare;
                        else
                           stare <= idle;
                        end if;
  when initializare =>
                        loada <= '1';
                        loadb <= '1';
                        loadq <= '1';
```

```
                    selmuxin <= '1';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    n1<="0000";
                    selNumar <= "11";
                    stare <= shiftareA;
when shiftareA =>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '1';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    n1<="0000";
                    selNumar <= "11";
                    if verif = '0' then
                    stare <= scadMul5;
                    else
                    stare <= atribuireCat;
                    end if;

when scadMul5 =>
                    loada <= '1';
                    loadb <= '0';
```

```
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '1';
                    selmuxin <= '0';
                    selNumar <= "10";
                    stare <= compNiv1;
when compNiv1 =>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";

                      if(an ='1') then
                         stare <= adunare2YNiv1;
                      else
                         stare <= scadere2YNiv1;
                      end if;

when adunare2YNiv1 =>
                    loada <= '1';
                    loadb <= '0';
```

```
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selNumar <= "01";
                    stare <= comparare11;
when scadere2YNiv1 =>

                    loada <= '1';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '1';
                    selNumar <= "01";
                    stare <= comparare12;



when comparare11 =>
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
```

```
                    selNumar <= "11";

                    if(an ='1') then
                       stare <= adunare2YNiv21;
                    else
                       stare <= scadereYNiv22;
                    end if;

when comparare12 =>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <="11";

                       if(an ='1') then
                          stare <= adunareYNiv23;
                       else
                          stare <= scadere2YNiv24;
                       end if;


when adunare2YNiv21 =>
```

```
                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selNumar <= "01";
                stare <= comparare21;
when scadereYNiv22 =>

                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '1';
                selNumar <= "00";
                stare <= comparare22;

when adunareYNiv23 =>

                loada <= '1';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
```

```
                    subb <=  '0';
                    selNumar <= "00";
                    stare <= comparare23;

when scadere2YNiv24 =>

                    loada <= '1';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '1';
                    selNumar <= "10";
                    stare <= comparare24;


when comparare21 =>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";
```

```
                    if(an ='1') then
                        stare <= cifraZero;
                    else
                        stare <= scadereYNiv31;
                    end if;
when comparare22=>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";

                    if(an ='1') then
                        stare <= cifraTrei;
                    else
                        stare <= cifraPatru;
                    end if;

when comparare23 =>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
```

```
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";

                      if(an ='1') then
                        stare <= cifraCinci;
                      else
                        stare <= cifraSase;
                      end if;
when comparare24 =>
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";

                      if(an ='1') then
                        stare <= adunareYNiv32;
                      else
                        stare <= cifraNoua;
                      end if;
```

```
when scadereYNiv31 =>
                                loada <= '1';
                                loadb <= '0';
                                loadq <= '0';
                                shla <= '0';
                                shlq <= '0';
                                subb <=  '1';
                                selNumar <= "00";
                                stare <= comparare31;
when adunareYNiv32 =>
                                loada <= '1';
                                loadb <= '0';
                                loadq <= '0';
                                shla <= '0';
                                shlq <= '0';
                                subb <=  '0';
                                selNumar <= "00";
                                stare <= comparare32;

when comparare31 =>
                            loada <= '0';
                            loadb <= '0';
                            loadq <= '0';
                            shla <= '0';
                            shlq <= '0';
                            subb <=  '0';
```

```
                selmuxin <= '0';
                selNumar <= "11";

                    if(an ='1') then
                        stare <= cifraUnu;
                    else
                        stare <= cifraDoi;
                    end if;
when comparare32 =>
                loada <= '0';
                loadb <= '0';
                loadq <= '0';
                shla <= '0';
                shlq <= '0';
                subb <=  '0';
                selmuxin <= '0';
                selNumar <= "11";

                    if(an ='1') then
                        stare <= cifraSapte;
                    else
                        stare <= cifraOpt;
                    end if;
```

```
when cifraZero =>
                        loadb <= '0';
                        loada <= '0';
                        loadq <= '0';
                        shla <= '0';
                        shlq <= '0';
                        subb <=  '0';
                        selmuxin <= '0';
                        selNumar <= "11";
                        n1 <= "0000";
                        stare <= atribuireCat;
when cifraUnu =>
                        loadb <= '0';
                        loada <= '0';
                        loadq <= '0';
                        shla <= '0';
                        shlq <= '0';
                        subb <=  '0';
                        selmuxin <= '0';
                        selNumar <= "11";
                        n1 <= "0001";
                        stare <= atribuireCat;


when cifraDoi =>
                                loadb <= '0';
```

```
                    loada <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";
                    n1 <= "0010";
                    stare <= atribuireCat;
when cifraTrei =>
                    loadb <= '0';
                    loada <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";
                    n1 <= "0011";
                    stare <= atribuireCat;
when cifraPatru =>
                    loadb <= '0';
                    loada <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
```

```
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";
                    n1 <= "0100";
                    stare <= atribuireCat;
when cifraCinci =>
loadb <= '0';
loada <= '0';
loadq <= '0';
shla <= '0';
shlq <= '0';
subb <=  '0';
selmuxin <= '0';
selNumar <= "11";
n1 <= "0101";
stare <= atribuireCat;

when cifraSase =>
loada <= '0';
loadb <= '0';
loadq <= '0';
shla <= '0';
shlq <= '0';
subb <=  '0';
selmuxin <= '0';
selNumar <= "11";
```

```
n1 <= "0110";

when cifraSapte =>
loada <= '0';
loadb <= '0';
loadq <= '0';
shla <= '0';
shlq <= '0';
subb <=  '0';
selmuxin <= '0';
selNumar <= "11";
n1 <= "0111";
stare <= atribuireCat;

when cifraOpt =>
loada <= '0';
        loadb <= '0';
        loadq <= '0';
        shla <= '0';
        shlq <= '0';
        subb <=  '0';
        selmuxin <= '0';
        selNumar <= "11";
        n1 <= "1000";
        stare <= atribuireCat;
when cifraNoua =>
```

```
loada <= '0';
        loadb <= '0';
        loadq <= '0';
        shla <= '0';
        shlq <= '0';
        subb <=  '0';
        selmuxin <= '0';
        selNumar <= "11";
        n1 <= "1001";
        stare <= atribuireCat;

when atribuireCat =>
                        nn <= n1;
                        loada <= '0';
                        loadb <= '0';
                        loadq <= '0';
                        shla <= '0';
                        shlq <= '1';
                        subb <=  '0';
                        selmuxin <= '0';
                        selNumar <= "11";

                        stare <= compFin;
when compFin =>
                        loada <= '0';
                        loadb <= '0';
```

```
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";

                    cn <= cn - 1;

                    if cn = 0 then
                        stare <= stop;
                    else
                        stare <= shiftareA;
                    end if;
  when stop =>
                    t <= '1';
                    loada <= '0';
                    loadb <= '0';
                    loadq <= '0';
                    shla <= '0';
                    shlq <= '0';
                    subb <=  '0';
                    selmuxin <= '0';
                    selNumar <= "11";
  when others => stare <= idle;
end case;
```

```
    end if;
end process;
term <= t;


end Behavioral;
```

impartireNouaMultiplii

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.ALL;


entity impartireNouaMultiplii 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 impartireNouaMultiplii;

architecture Behavioral of impartireNouaMultiplii is

signal loada,loadb,loadq,shla,shlq,subb,ovf,selmuxin:  std_logic :='0';
signal selNum: std_logic_vector(1 downto 0):="00";
signal an,outsuman: std_logic:='1';
signal outb,insum,outsum,outq: std_logic_vector(n-1 downto 0);
signal outa,ina: std_logic_vector(n downto 0);
signal nn: std_logic_vector(3 downto 0):=(others =>'0');
signal verificare: std_logic :='0';

begin

unitateDeControl: entity work.unitateaDeControl_nouaMultiplii generic map ( n =>
n) port map (
        verif=>verificare,start=>start,clk=>clk,rst=>rst,
        an=>an,selmuxin=>selmuxin,loada=>loada,loadb=>loadb,
        loadq=>loadq,shla=>shla,shlq=>shlq,subb=>subb,
        term=>term,nn=>nn,selNumar=>selNum);
```

```
registrulb: entity work.nbit_register_9multiplii generic map(n => n) port map(
    d=>y,
    ce=>loadb,
    clk=>clk,
    rst=>rst,
    sel=>selNum,
    q=>outb);


insum <= x"99999" - outB when subb = '1' else outB;

suma: entity WORK.sumator_16biti
            port map(
            x=>outa(n-1 downto 0),
            y=>insum,
            cin=>subb,
            s=>outsum,
            cout=>outsuman);

an <= outsum(n-1) or outsum(n-2) or outsum(n-3) or outsum(n-4);
ina <= '0' & x"00000" when selmuxin = '1' else an & outsum;

registrula: entity WORK.slrn_registru_de_deplasare_stanga
      generic map(n => n+1) port map(
            d => ina,
            sri=> outq(n-1 downto n-4),
```

```
            load=>loada,
            ce=>shla,
            clk=>clk,
            rst=>rst,
            q=>outa);

verificare <= '1' when outa = x"00000" --and outq(n-1 downto n-4) = x"0"
              else '0';


registrulq: entity WORK.slrn_registru_de_deplasare_stanga
      generic map(n => n) port map(
            d => x,
            sri=> nn,
            load=>loadq,
            ce=>shlq,
            clk=>clk,
            rst=>rst,
            q=>outq);

q<=outq;
a<= outa(n-1 downto 0);
end Behavioral;
```

generator_multiplii

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity generator_multiplii is
 Port (
 Y : in STD_LOGIC_VECTOR (19 downto 0);
 outY: out STD_LOGIC_VECTOR (19 downto 0);
 out2Y: out STD_LOGIC_VECTOR (19 downto 0);
 out5Y: out STD_LOGIC_VECTOR (19 downto 0)
  );
end generator_multiplii;

architecture Behavioral of generator_multiplii is
signal outs1,outs2,outs3 : STD_LOGIC_VECTOR (19 downto 0):=x"00000";
signal cin,cout :  STD_LOGIC:='0';
begin
outY<=Y;
suma1: entity WORK.sumator_16biti
            port map(x=>y,y=>y,cin=>cin,s=>outs1,cout=>cout);
out2Y<=outs1;

suma2: entity WORK.sumator_16biti
```

```
        port map(x=>outs1,y=>outs1,cin=>cin,s=>outs2,cout=>cout);
suma3: entity WORK.sumator_16biti
        port map(x=>outs2,y=>y,cin=>cin,s=>outs3,cout=>cout);
out5Y<=outs3;


end Behavioral;
```

nbit_register_9multiplii

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;


entity nbit_register_9multiplii is
  GENERIC ( n : natural ) ;
Port (
  d: in std_logic_vector(n-1 downto 0);
  ce: in std_logic;
  clk: in std_logic;
  rst: in std_logic;
  sel: in std_logic_vector(1 downto 0);
  q: out std_logic_vector(n-1 downto 0)
```

```
 );
end nbit_register_9multiplii;

architecture Behavioral of nbit_register_9multiplii is
signal outy,out2y,out5y: STD_LOGIC_VECTOR (n-1 downto 0):=(others => '0');
signal y,doiy,cinciy : std_logic_vector(n-1 downto 0):=(others => '0');
begin

multiplii: entity WORK.generator_multiplii
            port map(y=>d,outy=>outy,out2y=>out2y,out5y=>out5y);

registru: process(d,ce,rst,clk,sel)
begin
   if rising_edge(clk) then
      if rst = '1' then
         y <= (others => '0');
         doiy <= (others => '0');
         cinciy <= (others => '0');
       elsif ce = '1' then
         y <= outY;
         doiy <= out2y;
         cinciy <= out5y;
      end if;
   end if;
end process registru;
```

```
selectie: process(sel)
begin
case sel is
    when "00" => q <= y;
    when "01" => q <= doiy;
    when "10" => q <= cinciy;
    when others => q <= x"00000";
end case;
end process selectie;


end Behavioral;
```

decodificator7seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity decodificator7seg is
  Port (
  intrare: in std_logic_vector(3 downto 0);
  Sseg: out std_logic_vector(7 downto 0)
   );
```

```
end decodificator7seg;

architecture Behavioral of decodificator7seg is

begin

process(intrare)
begin
        case intrare is
      when "0000" => Sseg <= "11000000";  -- 0
      when "0001" => Sseg <= "11111001";  -- 1
      when "0010" => Sseg <= "10100100";  -- 2
      when "0011" => Sseg <= "10110000";  -- 3
      when "0100" => Sseg <= "10011001";  -- 4
      when "0101" => Sseg <= "10010010";  -- 5
      when "0110" => Sseg <= "10000010";  -- 6
      when "0111" => Sseg <= "11111000";  -- 7
      when "1000" => Sseg <= "10000000";  -- 8
      when "1001" => Sseg <= "10010000";  -- 9
      when "1010" => Sseg <= "10001000";  -- A
      when "1011" => Sseg <= "10000011";  -- b
      when "1100" => Sseg <= "11000110";  -- C
      when "1101" => Sseg <= "10100001";  -- d
      when "1110" => Sseg <= "10000110";  -- E
      when "1111" => Sseg <= "10001110";  -- F
      when others => Sseg <= "11111111";
```

```
    end case;

end process;


end Behavioral;
```

displ7seg

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;



entity displ7seg is
  Port ( Clk  : in  STD_LOGIC;
         Rst  : in  STD_LOGIC;
         Data : in  STD_LOGIC_VECTOR (31 downto 0);
         An   : out STD_LOGIC_VECTOR (7 downto 0);
         Seg  : out STD_LOGIC_VECTOR (7 downto 0));
end displ7seg;
```

```
architecture Behavioral of displ7seg is
constant CLK_RATE  : INTEGER := 100_000_000;  -- frecventa semnalului Clk
constant CNT_100HZ : INTEGER := 2**20;        -- divizor pentru rata de
                                              -- reimprospatare de ~100 Hz
constant CNT_500MS : INTEGER := CLK_RATE / 2; -- divizor pentru 500 ms
signal Count       : INTEGER range 0 to CNT_100HZ - 1 := 0;
signal CountBlink  : INTEGER range 0 to CNT_500MS - 1 := 0;
signal BlinkOn     : STD_LOGIC := '0';
signal CountVect   : STD_LOGIC_VECTOR (19 downto 0) := (others => '0');
signal LedSel      : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
signal Digit1      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit2      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit3      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit4      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit5      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit6      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit7      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
signal Digit8      : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');

begin
  -- Proces pentru divizarea frecventei ceasului
  div_clk: process (Clk)
  begin
    if RISING_EDGE (Clk) then
      if (Rst = '1') then
```

```
      Count <= 0;
    elsif (Count = CNT_100HZ - 1) then
      Count <= 0;
    else
      Count <= Count + 1;
    end if;
  end if;
end process div_clk;

CountVect <= CONV_STD_LOGIC_VECTOR (Count, 20);
LedSel <= CountVect (19 downto 17);

  -- Date pentru segmentele fiecarei cifre
 comp1: entity WORK.decodificator7seg port map(Data (3 downto 0),Digit8);
 comp2: entity WORK.decodificator7seg port map(Data (7 downto 4),Digit7);
 comp3: entity WORK.decodificator7seg port map(Data (11 downto 8),Digit6);
 comp4: entity WORK.decodificator7seg port map(Data (15 downto 12),Digit5);
 comp5: entity WORK.decodificator7seg port map(Data (19 downto 16),Digit4);
 comp6: entity WORK.decodificator7seg port map(Data (23 downto 20),Digit3);
 comp7: entity WORK.decodificator7seg port map(Data (27 downto 24),Digit2);
 comp8: entity WORK.decodificator7seg port map(Data (31 downto 28),Digit1);



  -- Semnal pentru selectarea cifrei active (anozi)
An <= "11111110" when LedSel = "000" else
```

```
        "11111101" when LedSel = "001" else
        "11111011" when LedSel = "010" else
        "11110111" when LedSel = "011" else
        "11101111" when LedSel = "100" else
        "11011111" when LedSel = "101" else
        "10111111" when LedSel = "110" else
        "01111111" when LedSel = "111" else
        "11111111";


  Seg <= Digit8 when LedSel = "000" else
         Digit7 when LedSel = "001" else
         Digit6 when LedSel = "010" else
         Digit5 when LedSel = "011" else
         Digit4 when LedSel = "100" else
         Digit3 when LedSel = "101" else
         Digit2 when LedSel = "110" else
         Digit1 when LedSel = "111" else
         x"FF";
end Behavioral;
```

filtru_buton

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity filtru_buton is
    Port ( Clk  : in  STD_LOGIC;
           Rst  : in  STD_LOGIC;
           Din  : in  STD_LOGIC;
           Qout : out STD_LOGIC);
end filtru_buton;

architecture Behavioral of filtru_buton is
    signal Cnt : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
    signal Q1  : STD_LOGIC := '0';
    signal Q2  : STD_LOGIC := '0';
    signal Q3  : STD_LOGIC := '0';
begin

    process (Clk)
    begin
        if rising_edge (Clk) then
            if Rst = '1' then
                Cnt <= (others => '0');
            else
                Cnt <= Cnt + 1;
            end if;
```

```
    end if;
end process;

process (Clk)
begin
    if rising_edge (Clk) then
        if (Rst = '1') then
            Q1 <= '0';
        elsif Cnt = x"FFFF" then
            Q1 <= Din;
        end if;
    end if;
end process;

process (Clk)
begin
    if rising_edge (Clk) then
        if (Rst = '1') then
            Q2 <= '0';
            Q3 <= '0';
        else
            Q2 <= Q1;
            Q3 <= Q2;
        end if;
    end if;
end process;
```

```
    Qout <= Q2 and (not Q3);

end Behavioral;
```

implementare_pe_placuta

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity implementare_pe_placuta is
 Port (
   Clk  : in  STD_LOGIC;
   Rst  : in  STD_LOGIC;
   Start : in std_logic;
   SelAfis: in std_logic;
   SaveY:in std_logic;
   x: inout std_logic_vector(15 downto 0);
```

```
   An   : out STD_LOGIC_VECTOR (7 downto 0);
   Seg  : out STD_LOGIC_VECTOR (7 downto 0);
   Tout1: out std_logic;
   Tout2: out std_logic;
   Tout3: out std_logic;
   Op1: out std_logic;
   Op2: out std_logic;
   Op3: out std_logic;
   Op4: out std_logic
   );
end implementare_pe_placuta;

architecture Behavioral of implementare_pe_placuta is
signal startDebounce,selAfisDebounce,saveYDebounce: std_logic:='0';
signal y: std_logic_vector(15 downto 0):=(others => '0');
signal selectie: std_logic_vector(1 downto 0):="00";
signal in_data: std_logic_vector(31 downto 0):= (others => '0');
signal x_in,y_in: std_logic_vector(19 downto 0):=(others => '0');
signal a_icr,q_icr: std_logic_vector(19 downto 0):=(others => '0');
signal a_ifr,q_ifr: std_logic_vector(19 downto 0):=(others => '0');
signal a_i9m,q_i9m: std_logic_vector(19 downto 0):=(others => '0');
begin

pt_btn_Start: entity WORK.filtru_buton port map(Clk=>clk,Rst => rst, Din=> start,
Qout=>startDebounce);
pt_btn_selAfis: entity WORK.filtru_buton port map(Clk=>clk,Rst => rst, Din=>
selAfis, Qout=>selAfisDebounce);
```

```
pt_btn_saveY: entity WORK.filtru_buton port map(Clk=>clk,Rst => rst, Din=> saveY, Qout=>saveYDebounce);


salvare_y:process(saveYDebounce,x)
begin
   if rising_edge(saveYDebounce) then
      y <= x;
   end if;
end process salvare_y;


increment_selectie: process(selAfisDebounce)
begin
   if rising_edge(selAfisDebounce) then
      selectie <= selectie + 1 ;
   end if;
end process increment_selectie;


selectie_afisare: process(selectie)
begin
   case selectie is
   when "00" => in_data <= x & y;
   when "01" => in_data <= a_icr(15 downto 0) & q_icr(15 downto 0);
   when "10" => in_data <= a_ifr(15 downto 0) & q_ifr(15 downto 0) ;
   when "11" => in_data <= a_i9m(15 downto 0) & q_i9m(15 downto 0) ;
   when others => in_data <= x"00000000";
   end case;
```

```
end process selectie_afisare;

Op1<= '1' when selectie = "00" else '0';
Op2<= '1' when selectie = "01" else '0';
Op3<= '1' when selectie = "10" else '0';
Op4<= '1' when selectie = "11" else '0';

x_in <= "0000" & x;
y_in <= "0000" & y;

dut: entity work.impartireaCuRefacere generic map(n=>20) port map (
        clk => Clk,rst => Rst,start => startDebounce,x=>x_in,y=>y_in,a=>a_icr,q=>q_icr,term=>Tout1);
dut2: entity work.impartireaFaraRefacere generic map(n=>20) port map (
        clk => Clk,rst => Rst,start =>startDebounce,x=>x_in,y=>y_in,a=>a_ifr,q=>q_ifr,term=>Tout2);
dut3: entity work.impartireNouaMultiplii generic map(n=>20) port map (
        clk => Clk,rst => Rst,start => startDebounce,x=>x_in,y=>y_in,a=>a_i9m,q=>q_i9m,term=>Tout3);

afisare: entity WORK.displ7seg port map(Clk=> clk ,Rst => rst ,Data => in_data,An => An,Seg => Seg);



end Behavioral;
```