

Unitate de calcul în virgulă mobilă: înmulțirea

Tomoiagă Andreea, Vereș Adela

Grupa 30235

Profesor îndrumător: Mocan Cristi

Data: 20.03.2017

**Cuprins**

[**1. Rezumat**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_umj4vsw72h1s) **4**

[**2. Introducere**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_tnsvskcks4p) **4**

[**3. Fundamentare teoretică**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_fqzby7ox208w) **6**

[3.1. VHDL](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_pvh836b0pebu) 6

[3.2. Tehnologii utilizate](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_7heevxvm1x36) 6

[3.2.1 Nexys4 DDR](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_go4etog1lhkf) 6

[3.2.2 PmodKYPD](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_kxnhlhpdinxk) 6

[3.3. Reprezentarea numerelor în virgulă mobilă, formatul IEEE 754](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_xhx0hgklkltd) 7

[3.4. Sumatorul elementar](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_3827gikrg6fp) 9

[3.4.1. Sumatorul cu propagarea succesivă a transportului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_20pd6xkoco1o) 11

[3.4.2. Sumatorul cu anticiparea transportului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_6mor4ka6ekf6) 12

[3.4.3. Sumatorul cu salvarea transportului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_dzsk9fmn5k2w) 15

[3.5. Metode de înmulțire a numerelor întregi](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_6ppf36ibntbs) 16

[3.5.1. Înmulțirea prin deplasare și adunare](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_28aiqz7x9eo9) 16

[3.5.2. Înmulțirea matricială](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_9puuxyq7665r) 18

[3.5.3. Arborele Wallace](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_ngsxu9br1nfw) 23

[3.6. Înmulțirea în virgulă mobilă](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_3s9y1jscc4ug) 26

[3.7. Considerații de precizie](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_5r8qn419eyyt) 29

[3.8. Soluția finală și obiectivul proiectului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_e6ba7nenhazz) 29

[**4. Proiectare şi implementare**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_bs5kzlarihg3) **30**

[4.1. Schemă înmulțitor](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_1azz4x7ymtgm) 30

[4.2. Sumator](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_qkbu5nheo5le) 32

[4.2.1 Sumator cu anticiparea transportului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_d5xg58nc0kay) 33

[4.2.2 Sumatorul sistemului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_nivtt4lkbccs) 34

[4.3. Detecție depășiri](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_vtu6fgr9v8d3) 35

[4.4. Determinarea semnului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_vtu6fgr9v8d3) 35

[4.5. Înmulțitor](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_ynh4jogv8l84) 35

[4.6. Circuit de normalizare și rotunjire](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_q6o5qqv5vcx4) 36

[4.6.1 Circuit de normalizare](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_k79qn02i8e2w) 36

[4.6.2 Circuit de rotunjire](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_6h7la49e9j2d) 36

[4.8. Schemă detaliată înmulțitor](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_vtu6fgr9v8d3) 38

[**5. Rezultate experimentale**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_jke7y1jfm3hd) **38**

[5.1. Sumator](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_ah3q3fylfazx) 38

[5.1.1 Sumator cu anticiparea transportului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_xue3omycdo4b) 38

[5.1.2 Sumatorul sistemului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_jgg9s0ezezko) 39

[5.2. Detecție depășiri](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_mcwzvxc6a4do) 40

[5.3. Înmulțitor](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_uiglw13ws1nb) 40

[5.4. Normalizarea și rotunjirea](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_mcwzvxc6a4do) 41

[5.5. Proiect final](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_mcwzvxc6a4do) 41

[**6. Manual utilizator**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_d56pym717j0y) **41**

[6.1. Simularea aplicației](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_flihr2fz6ucj) 41

[6.2. Configurarea proiectului pentru placuța](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_w6afja8zwhfq) 41

[6.3. Utilizarea înmulțitorului](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_lpq0uz3w8f6f) 41

[**7. Concluzii**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_fp29d97esrf7) **41**

[**Bibliografie**](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29) **41**

# **1. Rezumat**

# **2. Introducere**

Tema propusă spre implementare este cea de înmulțire a numerelor reprezentate în virgulă mobilă prin standardul IEEE 754. Majoritatea unităţilor centrale de procesare lucrează cu două tipuri de numere: în virgulă fixă şi în virgulă mobilă[[1]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29) . Acest lucru este datorat faptului că nu toate informațiiile necesară UCP-ului pentru a-și realiza operațiile specifice sunt numere întregi. Reprezentarea internă în calculator a datelor este realizată printr-un format în virgulă mobilă și pentru asigurarea portabilității programelor între calculatoare diferite, această reprezentare internă respecta un standard(IEEE 754) utilizat de marea majoritate a unităților de calcul și a coprocesoarelor matematice.

Multe aplicații necesită numere care nu sunt întregi. Există mai multe posibilități pentru reprezentarea acestor numere. Una dintre ele este reprezentarea în *virgulă fixă*. În acest caz, se poate utiliza aritmetica pentru numere întregi, plasând apoi virgula binară în poziția predefinită, de exemplu, după bitul de semn. În cazul reprezentării numerelor in acest format, deși virgula nu este reprezentată fizic în calculator, poziția virgulei binare, care este stabilită la proiectare, nu mai poate fi schimbată[[2]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29). Astfel, este necesară transformarea tuturor numerelor în acest format, lucru ce necesită o serie de operații de de scalare sau deplasare, atașând numerelor *factori de scală.* Evidența acestora trebuie realizată prin program, ceea ce mărește timpul de calcul.

O solutie la problemele aduse de reprezentarea numerelor în virgulă fixă este utilizarea unei tehnici de scalare automată, cunoscută sub numele de reprezentare în *virgulă mobilă*. În acest caz, factorul de scală devine o parte a cuvântului din calculator, poziția virgulei variind pentru fiecare număr în mod automat. Un număr reprezentat în virgulă mobilă are două componente: *mantisa(*M) și *exponentul(*E). Această reprezentare poate fi memorată într-un cuvânt cu trei câmpuri: semnul, mantisa și exponentul. În general, câmpul exponentului conține o valoare pozitivă, valoare obținută prin adunarea unui *deplasament*: în felul acesta, câmpul exponentului va avea întotdeauna o valoare pozitivă.

Reprezentarea numerelor în virgulă mobilă poate fi realizată în mai multe moduri prin diferite modalități de tratare a cazurilor de excepție, numărul de biți alocați fiecărui câmp în reprezentare, modul de rotunjire, etc. Din cauza multiplelor moduri de tratare a acestei reprezentări, programele scrise nu sunt portabile întrucât două calculatoare cu două moduri diferite de reprezentare vom realiza scrierea acestuia în moduri diferite.

Pentru a asigura portabilitatea programelor, *Societatea Calculatoarelor a IEEE(Institute of Electrical and Electronics Engineers* a elaborat un standard pentru reprezentarea numerelor în virgulă mobilă și pentru operații aritmetice în virgulă mobilă. Acest standard prevede o metodă de calcul cu numere în virgulă mobilă, care va produce același rezultat dacă prelucrarea se face în hardware, software sau o combinație a celor două. Rezultatele vor fi identice, independent de punere în aplicare, având în vedere aceleași date de intrare. De asemenea, erorile și condițiile de eroare  
produse în operații matematice vor fi raportate într-o manieră consistentă, indiferent de  
modul de implementare.

O problemă întâlnită la calculele în acest mod de reprezentare este modul de tratare a depășirilor inferioare și superioare. Depășirea superioară apare în momentul în care un exponent depășește valoarea maximă, iar depășirea inferioară apare atunci când exponentul are o valoare mai mică decât cea minimă.

Problema propusă spre soluționare, operația de înmulțire a numerelor în virgula mobilă, presupune adunarea caracteristicilor si înmultirea mantiselor[[3]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29). După efectuarea operației propiu-zise de înmulțire are loc rotunjirea rezultatului, întrucât produsul are lungimea dublă față de cea a operanzilor și este necesară aducerea sa la o reprezentare în gama numerelor care pot fi reprezentate pe 32 biți. Standardul permite mai multe moduri de rotunjire: rotunjire spre 0, rotunjire spre , rotunjire spre și rotunjire la cel mai apropriat număr reprezentabil.

Adunarea exponenților și înmulțirea mantiselor coincid cu operatiile corespunzătoare realizate pe numere întregi. Astfel, adunarea exponenților se poate realiza prin intermediul sumatoarelor existente de adunare a numerelor întregi, dintre care și sumatorul cu propagarea succesivă a transportului, sumatorul cu anticiparea transportului, sumatorul cu selecția transportului, sumatorul cu salvarea transportului, sumatorul serial, sumatorul zecimal. Operația de înmulțire a mantiselor se poate realiza prin intermediul diferitelor metode existente, cum ar fi: înmulțirea prin deplasare și adunare, înmulțirea prin tehnica Booth, înmulțirea într-o bază superioară, înmulțirea matricială, înmulțirea prin arborele Wallace, înmulțirea prin deplasarea peste zero și peste unu.

Obiectivele principale ale acestui proiect sunt: realizarea unei reprezentări ale numerelor corespunzătoare cu formatul IEEE 754 pe 32 biți, implementarea algoritmului de înmulțire corespunzător pentru numere reprezentate în virgulă mobilă, realizarea operației de rotunjire pentru obținerea rezultatelor cu o precizie cât mai mare și tratarea cazurilor excepționale ce pot apărea în cadrul acestei operații (depășire inferioară, depășire superioară, rezultat inexact).

În cele ce urmează, capitolul “Fundamentare teoretică” va conține informații mai detaliate referitoare la formatul standard IEEE 754 pe 32 biți și modul de realizare a operației de înmulțire pentru numere reprezentate în acest format.

# **3. Fundamentare teoretică**

## **3.1. VHDL**

VHDL este un limbaj de desciere a hardware-ului (Hardware Description Language -HDL), destinat descrierii comportamentului și arhitecturii unui modul electronic logic, cu alte cuvinte al unei funcțiuni logice combinatorii sau secvențiale. Alături de Verilog, este cel mai utilizat limbaj de proiectare a sistemelor electronice digitale. E una din uneltele principale pentru proiectarea circuitelor integrate moderne, aplicat cu succes în campul microprocesoarelorm, în telecomunicatii, automobile și altele. Acest limbaj este utilizat în concepția asistată pe calculator (CAD) a circuitelor integrate sau pentru configurarea FPGA-urilor.

## **3.2. Tehnologii utilizate**

## **3.2.1 Nexys4 DDR**

Placa Nexys4 DDR este o platformă completă, gata de utilizare pentru dezvoltarea de circuite digitale bazate pe cele mai recente FPGA-uri Artix-7 de la Xilinx®. Cu o mare capacitate FPGA, memorii externe generoase, precum și o colecție de USB, Ethernet și alte porturi, Nexys4 DDR poate găzdui modele variind de la circuite combinaționale introductive la procesoare puternice încorporate. Mai multe tipuri periferice încorporate, inclusiv un accelerometru, senzor de temperatură, MEMS microfon digital, un amplificator, și mai multe dispozitive I / O permit utilizarea plăcii Nexys4 DDR pentru o gamă largă de modele, fără a avea nevoie de alte componente[[4]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29). Sistemul implementat, realizat utilizând mediul de proiectare Vivado, va fi încărcat în circuitele plăcii Nexys4 DDR și va fi verificată funcționarea acestuia digital în condiții reale.

## **3.2.2 PmodKYPD**

PmodKYPD este o tastatură de 16 butoane aranjate într-un format hexadecimal (0-F). Conducând digital o coloană la un nivel scăzut logic și fiecărei linii, utilizatorii pot determina care buton este apăsat în prezent. Placa gazdă comunică cu tastatura prin intermediul protocolului GPIO[[5]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29). Această tastatură va fi utilizată pentru introducerea operanzilor în virgulă mobilă ai operației de înmulțire și va face utilizarea proiectului mult mai ușoară, întrucât datele de intrare nu vor fi adăugate în formatul binar, ci în formatul zecimal.

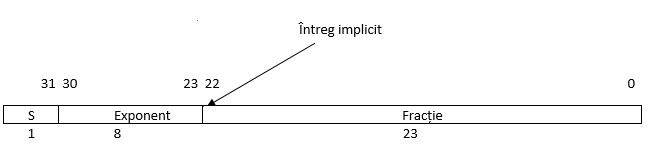
## **3.3. Reprezentarea numerelor în virgulă mobilă, formatul IEEE 754**

Multe operații necesită numere care nu sunt întregi. Aceste numere pot fi reprezentate atât în virgulă fixă, cât și în virgulă mobilă. Este preferată ultima variantă menționată, întrucât în celălalt mod de reprezentare poziția virgulei binare nu poate fi modificată și reprezentarea numerelor în acest format necesită unele operații de scalare sau deplasare, care fac timpul de execuție să crească.

În virgulă mobilă, un număr poate fi reprezentat prin două componente: mantisa, care indică valoarea exactă a numărului într-un anumit domeniu, și exponentul, care indică ordinul de mărime al numărului. Ținând cont de aceste componente, un număr în virgulă mobilă este reprezentat în felul următor:

Reprezentarea numerelor în formatul IEEE 754 asigură portabilitatea programelor de la un calculator la altul, întrucât acestea conțin aceeași reprezentare internă a numerelor în virgulă mobilă. Această reprezentare poate fi memorată într-un cuvânt binar cu trei câmpuri: semn, mantisă și exponent[[2].](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29) În cele ce urmează, va fi detaliat modul de reprezentare, intervalul numerelor definite și modul de tratare a unor condiții speciale și de eroare pentru formatul IEEE 754 pe 32 de biți.

În cazul reprezentării pe 32 de biți, standardul impune ca dimensiunea celor trei câmpuri componente ale reprezentării interne să fie următoarea: un bit pentru semn, 8 biți pentru exponent și 23 biți pentru mantisă. Acest format este numit format cu *precizie simplă* și poate fi vizibil în figura 3.1.



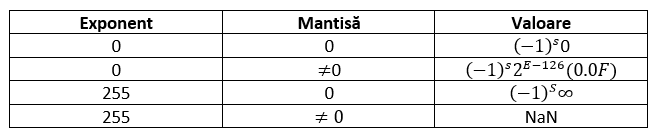
**Figura 3.1.**Formatul cu precizie simplă

Aceasta este o reprezentare în mărime și semn, întrucât semnul are un câmp separat pentru reprezentare. Acest câmp este 0 pentru numerele pozitive și 1 pentru cele negative. Numerele reprezentate în acest format sunt normalizate, adică au bitul cel mai semnificativ al mantisei egal cu 1.

În acest format apare noțiunea de *bit ascuns.* Acesta se referă fixarea unei valori pentru bitul cel mai semnificativ al mantisei prin intermediul formatului de reprezentare ,valoare ce nu poate fi modificată, motiv pentru care acest bit nu mai este reprezentat. În formatul precizie simplă, mantisa este alcătuită dintr-un bit ascuns cu valoarea 1, reprezentând partea întreagă a numărului), virgula binară implicită, urmată de biții de fracție. Utilizarea acestui bit implicit crește precizia operațiilor și reprezentărilor numerelor în virgulă mobilă, întrucât permite mantisei să aibă un bit în plus. Intervalul numerelor care pot fi reprezentate în precizie simplă este cuprins între 1,18\*și

3,4\*.

Câmpul exponentului în acest format, nu conține valoarea exponentului, ci o valoare numită *caracteristică.* Aceasta se obține prin adunarea unui deplasament de 127 la exponent, astfel încât valoarea caracteristicii este întotdeauna pozitivă. Deplasamentul este ales la valoarea 127 pentru ca cel mai număr posibil în reprezentarea cu semn să aibă caracteristica 0. Avantajul utilizării caracteristicii constă în simplificarea operațiilor asupra exponentului, pentru că valorile acesteia sunt numere pozitive și nu mai este necesară utilizarea operațiilor cu semn. Valoarea minimă și maximă ale acesteia sunt folosite pentru reprezentarea unor valori speciale. În tabelul 3.1 sunt prezentate valorile speciale ale formatului pe 32 de biți.



**Tabelul 3.1.**Valori speciale ale numerelor în IEEE 754

Valoarea 0 va fi reprezentată prin atribuirea câmpului de exponent și cel al mantisei a valorii 0. În acest caz, bitul ascuns este impicit 0, și nu 1.

Formatul cu precizie simplă prezintă probleme în cazul reprezentării unor numere ce nu se află în gama numerelor care pot fi reprezentate pe un cuvânt de 32 biți, probleme cunoscute sub numele de depășire superioară și depășire inferioară. Depășirea superioară apare atunci când exponentul depășește valoarea maximă(127), în timp ce depășirea inferioară apare atunci când exponentul este mai mic decât valoarea minimă(-126).

În cazul unei depășiri inferioare, standardul permite utilizarea numere care nu sunt normalizate, numite *numere denormalizate.* Acestea sunt generate printr-o tehnică care constă în deplasarea mantisei la dreapta și incrementarea exponentului până când acesta ajunge la valoarea minimă permisă, tehnică numită *depășire inferioară graduală.*

În cazul unei depășiri superioare, există o reprezentare specială pentru infinit. Aceasta constă din atribuirea valorii maxime exponentului și valorii 0 mantisei. Bitul de semn va distinge în acest caz, reprezentarea între . Utilizatorul va putea decide dacă depășirea superioară va fi tratată ca și o condiție de eroare sau va continua calculele cu valoarea infinit.

O altă valoare specială a acestui format, este valoarea NaN. Aceasta va fi utilizată pentru a indica diferite condiții de excepție. În cazul operației de înmulțire, această valoare va fi intâlnită în cazul operației nedefinite . Reprezentarea acestei valori speciale va fi realizată prin atribuirea valorii maxime exponentului și a unei valori diferite de 0 mantisei. Standardul specifică că atunci când argumentul unei operații este NaN rezultatul va fi tot NaN.

## **3.4. Sumatorul elementar**

Sumatorul este blocul operațional de bază implicat în majoritatea operațiilor complexe ale majorității circuitelor, fiind utilizat în special în structura internă a UAL- Unității Aritmetico-logice. Acestea fiind spuse, optimizarea sumatoarelor se traduce în optimizarea întregului sistem, deoarece viteza de operare a acestora va influența direct proporțional viteza de calcul a circuitului final.

Sumatorul elementar este un bloc combinațional ce primește trei intrări și are ca rezultat două ieșiri. Intrările sunt compuse din cei doi biți de adunat , respectiv , și bitul de transport de la nivelul precedent - bitul din poziția mai puțin semnificativă - notat cu . Ieșirile sunt reprezentate de bitul suma, , respectiv bitul generat de transport către următorul bit din poziție mai semnificativă, .[[2]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29)

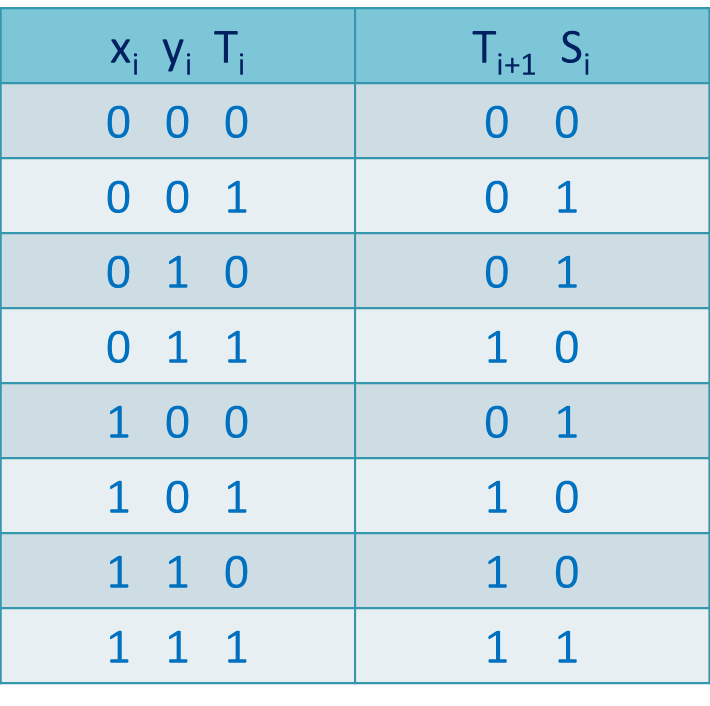
Expresiile booleene ale ieșirilor sunt după cum urmează:

.

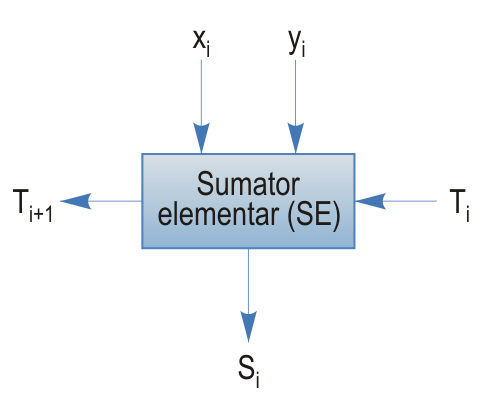
Operația “sau exclusiv” () are următoarea formulă de calcul:

.

Tabelul 3.2. Reprezintă tabelul de adevăr a operației de adunare cu ajutorul sumatorului elementar. Schema bloc a sumatorului elementar este dată de figura 3.2.



**Tabel 3.2.** Tabelul de adevăr al sumatorului elementar



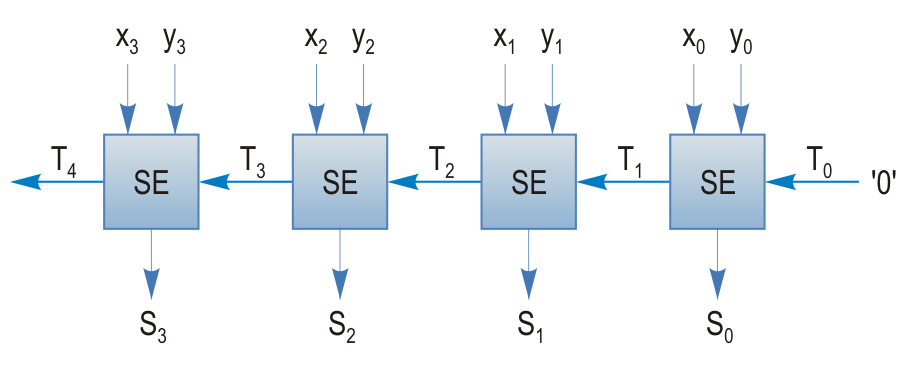
**Figura 3.2** Simbolul sumatorului elementar

Un sumator particular este *semisumatorul elementar* care primește ca și intrări cei doi biți de adunat (se omite bitul de transport) și generează la ieșire un bit de sumă și unul de transport.

În mod similar, *semisumatorul elementar* are ca și intrări doi biți - scăzătorul și descăzutul, iar la ieșire este generat un bit diferență și unul de împrumut. Scăzătorul elementar are în plus ca și intrare bitul de împrumut de la bitul mai puțin semnificativ.

### **3.4.1. Sumatorul cu propagarea succesivă a transportului**

Acest sumator este format dintr-o serie de sumatoare elementare conectate în cascadă și se realizează adunarea bit cu bit a două numere. Se utilizează câte un sumator pentru fiecare pereche de biți de adunat de la fiecare poziție. Spre exemplu adunarea a două numere reprezentate pe patru biți, și va utiliza patru sumatoare elementare; primul sumator va aduna biții , respectiv și va genera bitul cel mai puțin semnificativ al sumei, și un transport spre bitul următor, mai semnificativ (în cazul acesta, bitul de la poziția 1). Asftel ieșirea de transport a primului sumator este conectată la intrarea de transport a celui de-al doilea, realizându-se conectarea în serie a sumatoarelor. Primul sumator, deoarece are intrarea de transport setată pe 0, va putea fi implementat ca un semisumator. Ieșirea de transport a ultimului sumator (), în cazul în care va fi setată, va reprezenta cel mai semnificativ bit (de pe poziția 4, ). Schema bloc a acestui tip de sumator este redată în figura 3.3.



**Figura 3.3.** Sumator de 4 biți cu propagarea succesivă a transportului

Acest tip de sumator este avantajos ca și simplitate și datorită faptului ca nu necesită logică combinațională suplimentară, având un cost redus. Dezavantajul major, însă, este dat de viteza foarte redusă a acestuia. Această latență se datorează faptului că sumatorul de la fiecare etaj va trebui să aștepte după transportul de intrare propagat de etajul anterior, acest lucru fiind foarte costisitor pentru timpul de execuție.

Acest sumator se poate utiliza și cu funcționalitatea unui scăzător, în cazul în care se vor aduna unul din termeni (cel pozitiv) cu cel de-al doilea (negativ) reprezentat în complement față de doi. Transportul din poziția cea mai semnificativă a rezultatului se neglijează.

### **3.4.2. Sumatorul cu anticiparea transportului**

În sumatorul cu transport anticipat (Carry Look-Ahead), fiecare adunare pe bit elimină dependenţa de semnalul transportului generat anterior şi impune în schimb utilizarea valorilor celor doi operanzi de intrare. El funcționează prin generarea a două noi semnale (P și G) pentru fiecare rang binar în funcție de starea intrărilor. [[6]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29)

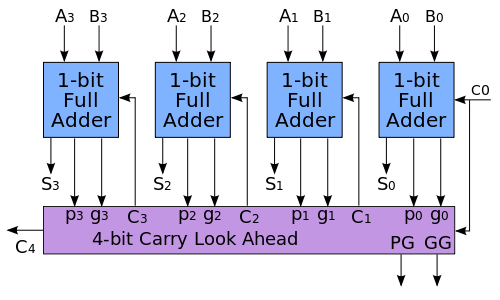
Fiecare etaj va avea un bloc de transport, conținând o logică combinațională ce va calcula transportul de intrare pentru acel etaj în funcție de transportul inițial și toți biții din etajele precedente (deci independent de alte semnale de transport).

Expresiile booleene pentru fiecare etaj de transport sunt date după cum urmează:

Semnalele P și G menționate anterior vin în ajutorul simplificării expresiei, referindu-se la *propagarea,* respectiv *generarea* transportului la ieșire. Acestea sunt definite astfel:

Funcția de *generare*  este denumită astfel deoarece etajul *i* va produce un transport () dacă ambii operanzi vor fi 1 (), independent de . Alternativ, *propagarea*  se referă la prezența unui transport () dacă având ca și intrare , unul din cei doi operanzi este setat (). Noua expresie devine:

Schema bloc a unui sumator de 4 biți cu anticiparea transportului care utilizează un generator de transport anticipat:



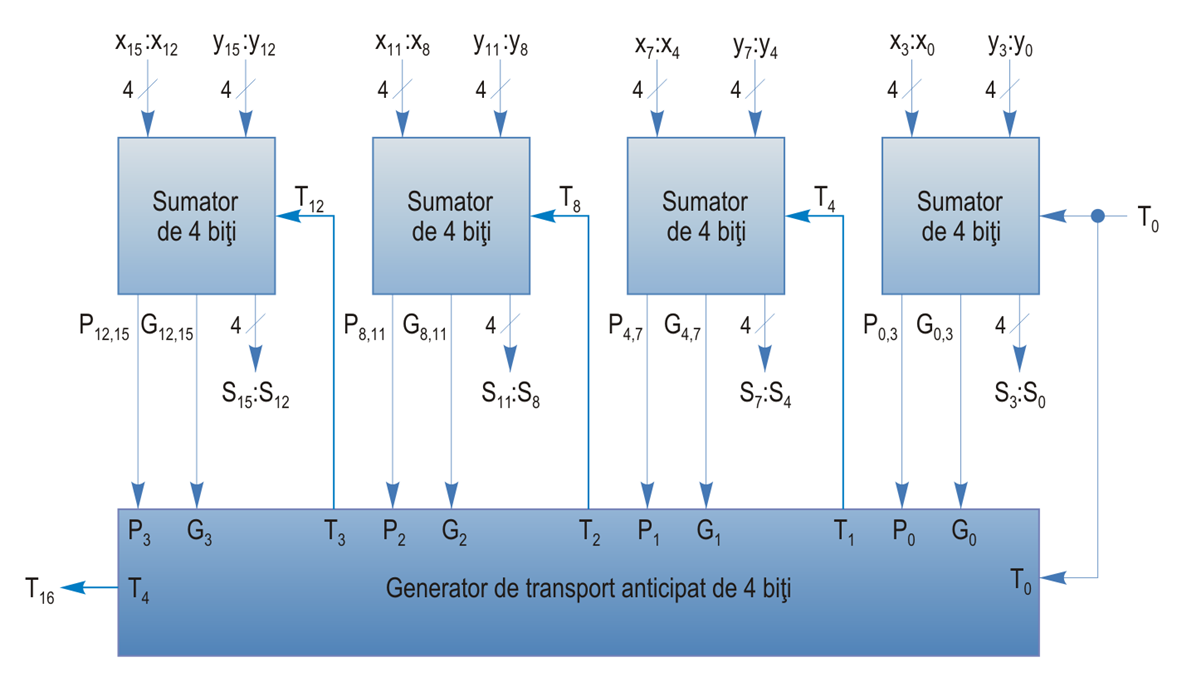
**Figura 3.4.** Sumator de 4 biți cu anticiparea transportului

Următoarele ecuații definesc ieșirile semnalelor de transport în cazul sumatorului pe 4 biți:

În cazul în care dorim să reducem complexitatea circuitului putem pune limitări la numărul porților și al intrărilor existente. Acest lucru se va traduce, însă, în necesitatea adăugării la circuit a unei logici suplimentare. Pentru a realiza acese limitări în cazul adunării a 4 biți (și nu numai), se definesc două noi funcții: pentru generarea transportului pe grup, , respectiv pentru propagarea transportului pe grup, , corespunzând bițiilor de la i până la k. Următorul exemplu ilustrează aceste două funcții în cazul sumatorului pe 4 biți:

Vom obține astfel:

Figura următoare arată transformarea sumatorului utilizând noile funcții, în cazul extinderii acestuia la un sumator de dimensiune mai mare. În locul utilizării sumatoarelor pe 1 bit se vor utiliza sumatoare pe grupuri de 4 biți.



**Figura 3.12.** Schema bloc a unui sumator de 16 biți format din sumatoare de 4 biți conectate prin semnale de transport generate anticipat

Intrarea de transport () a celui de-al treilea sumator se calculează după o ecuație similară cu următoarea:

, unde:

Ecuațiile pentru intrările și sunt asemănătoare calculelor pentru semnalele și utilizate în schema anterioară fără funcțiile de propagare, respectiv generare pe blocuri.

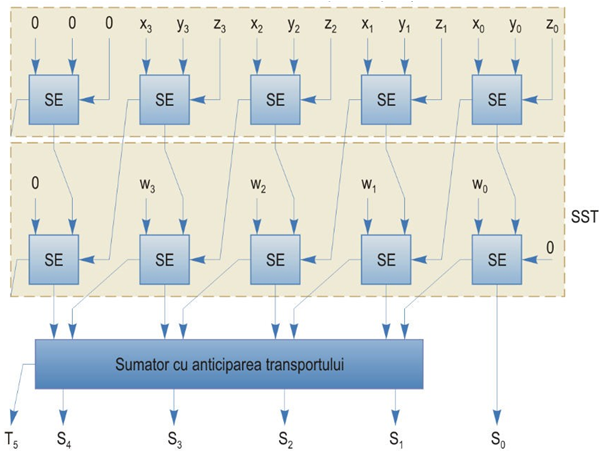
Acest tip de sumator este superior celui cu propagare succesivă datorită vitezei de execuție, însă necesită circuite combinaționale adiționale datorită complexității sale

### **3.4.3. Sumatorul cu salvarea transportului**

Adunarea cu salvarea transportului este o tehnică utilizată pentru reducerea timpului de propagare al semnalelor de transport, fiind o tehnică utilizată pentru creșterea vitezei operației de adunare atunci când trebuie adunate mai mult de două numere.

Un sumator bazat pe adunarea cu salvarea transportului este reprezentat printr-o colecție de n sumatoare elementare independente, n fiind numărul de biți al sumatorului. Semnalele de transport nu sunt propagate între sumatoarele elementare, viteza de adunare fiind substanțial mai ridicată prin generarea biților de sumă și de transport în paralel. Rezultatul final este obținut prin adunarea sumei și a transportului prin utilizarea unui sumator, diferit de cel cu salvarea transportului, numit *sumator cu propagarea transportului*.

Pe baza celor două faze existente în acest mod de adunare, cea de adunare independentă a biților și cea de adunare a sumei și transportului obținut, se poate realiza un sumator cu operanzi multipli. Acesta poate fi utilizat pentru realizarea circuitelor de înmulțire pentru acumularea produselor parțiale. Sumatorul standard cu salvarea transportului adună independent, în paralel, trei numere. Suma și transportul acestora pot fi folosite în altă operație de adunare, metodă prin care acest sumator poate fi folosit pentru adunarea a patru numere. Procesul acestei adunări constă în adunarea primelor trei numere, urmată de operția de adunare a sumei și produsului acestora cu cel de-al patrulea număr. Această adunare are ca rezultat o sumă și un transport ce vor fi adunate prin intermediul sumatorului cu propagarea transportului. În figura 3.4.1 este vizibil procedeul descris de adunare a patru numere utilizând două sumatoare cu salvarea transportului.



**Figura 3.4.1** Sumator cu salvarea transportului pentru adunarea a patru numere

Acest sumator este utilizat pentru adunarea a mai mult de două numere, acest număr nefiind limitat de vre constrângere. Schema bloc din figura 3.4.1 prezintă modul în care este realizată adunarea pentru patru numere. Numărul termenilor ce urmează a fi adunați poate fi mai mare decât cel prezentat în schemă, procedeu ce utilizează aceeași metodă a sumatorului pentru patru numere. Diferența în folosirea sumatorului pentru adunarea a mai mult de patru termeni constă în numărul de etape necesare pentru obținerea sumei și a transportului ce urmează a fi adunate în sumatorul cu propagarea transportului. Acestea se obțin după adunarea termenilor în sumatorul cu sumele și transporturile lor intermediare.

## **3.5. Metode de înmulțire a numerelor întregi**

### **3.5.1. Înmulțirea prin deplasare și adunare**

Varianta de înmulțire propusă este similară procedeului folosit în cazul înmulțirii manuale. Pentru aceasta fiecare bit din *înmulțitor* se va înmulți cu *deînmulțitul* creând produse parțiale ce vor fi deplasate cu câte o poziție spre stânga, la final însumându-se pentru a obține produsul final. Deoarece operațiile în baza 2 presupun lucrul cu biți de 0 și respectiv de 1, înmulțirea fiecărui bit din *înmulțitor* cu biții *deînmulțitului* va corespunde unei copieri a deînmulțitului în cazul în care bitul curent din înmulțitor este 1, respectiv cu generarea unui produs parțial egal cu 0, în cazul în care bitul curent din înmulțitor este 0.

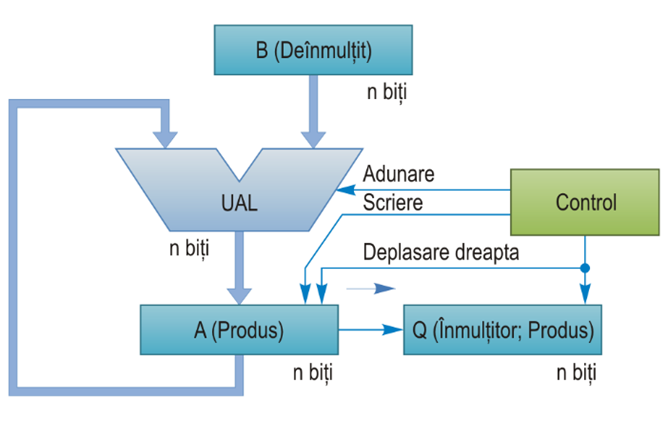
Figura 3.4. reprezintă o ilustrare a celor menționate până acum, folosind exemplul înmulțirii numărului 9 (1001) cu numărul 10 (1010), obținând 90 (1011010) [2] :

**Figura 3.5.** Înmulțirea numerelor întregi fără semn binare

Pentru a realiza procedeul explicat mai sus vom avea nevoie de regiștri pentru înregistrarea valorii deînmulțitului, a înmulțitorului și a valorilor intermediare a produsului. Va fi prezentă, de asemenea, o unitate aritmetico-logică (UAL) care va realiza adunarea produselor parțiale. Procedeul înmulțirii va fi comandat din blocul de control ce va trimite semnale corespunzătoare celorlalte componente.

Pentru o folosire mai eficientă a resurselor se va folosi un registru de *n* biți pentru stocarea deînmulțitului, acesta fiind static (față de prima variantă a înmulțitorului) și se va deplasa produsul spre dreapta, obținând o adunare corespunzătoare a rezultatelor parțiale. Registrul produsului va fi de asemenea de lungime *n*, deoarece ultimii biți (cei mai nesemnificativi) vor fi shiftați la dreapta, aceștia nefiind implicați în următoarele adunări, lucrându-se doar cu partea din stânga a produsului. Acești biți shiftați vor fi intrări pentru registrul înmulțitorului, care se golește prin shiftări la dreapta, pe măsură ce se analizează fiecare bit cu care se înmulțește (cel mai nesemnificativ). În acest mod, la final rezultatul se obține din concatenarea registrului produs cu registrul înmulțitorului (Rezultat = Produs;Înmulțitor).

Următoarea figură ilustrează forma finală a circuitului, unde am notat cu A registrul produs, cu Q registrul înmulțitor, iar cu B registrul deînmulțit.



**Figura 3.6.** Versiunea optimizată a circuitului de înmulțire prin deplasare și adunare [[7]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29)

### **3.5.2. Înmulțirea matricială**

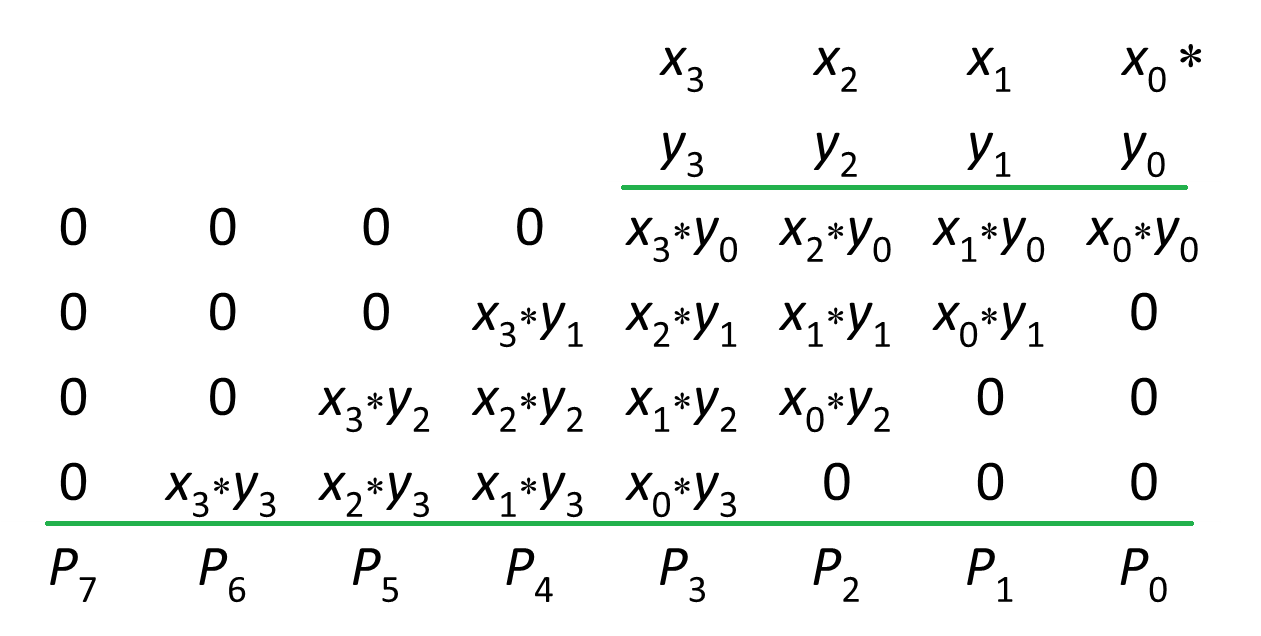
Circuitele de înmulțire matriceală se deosebesc de circuitele secvențiale prin utilizarea unei logici combinaționale suplimentare ce facilitează calculul produsului într-un pas. Elementele combinaționale folosite sunt elementare, ele realizând însumarea și deplasarea a doi biți sau a unui grup restrâns de biți.

Înmulțirea se va realiza pentru două numere întregi fără semn, *X* și *Y*, fiind reprezentați sub formele binare corespunzătoare: , respectiv . În acest caz, produsul *P* se poate scrie sub forma:

Ecuațita de mai sus se poate rescrie astfel:

Pentru a realiza înmulțirea prin formula dată mai sus avem nevoie de un bloc elementar pentru a calcula produsul . În acest scop vom folosi porți ȘI, deoarece înmulțirea aritmetică a doi biți coincide cu înmulțirea lor logică. Vom avea, în acest fel, un număr de porti ȘI, ce vor calcula în mod concurent toate produsele elementare dintre biții înmulțitorului și deînmulțitului. Operația de adunare se va realiza utilizând un număr de sumatoare elementare, dispuse matriceal, asemănător unui sumator bidimensional cu propagare succesivă a transportului. Dintre acestea, un număr de sumatoare pot fi înlocuite cu semisumatoare elementare. Deplasarea biților reprezentată de factorii și ai înmulțirii, se realizează prin dispunerea fizică în mod deplasat a nivelurilor de sumatoare, pe direcțiile și .

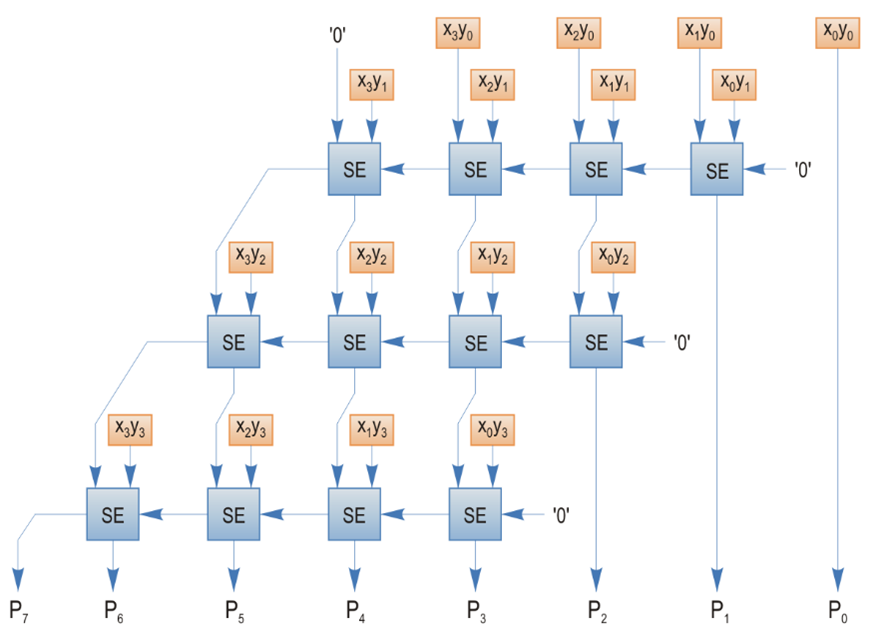
Exemplul unei astfel de înmulțiti descrise până acum este redat în figura 3.7.



**Figura 3.7.** Înmulțirea matriceală pe 4 biți

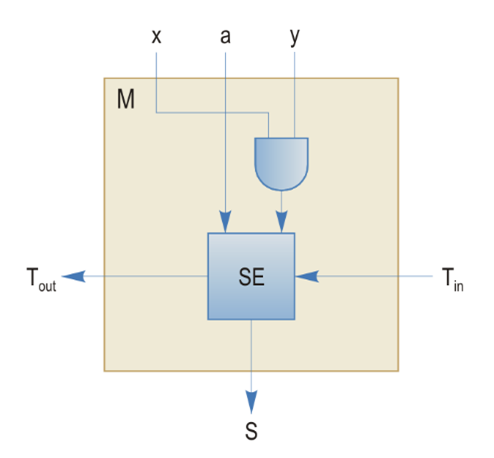
Biții produsului final se pot calcula după următoarele formule:

Figura următoare reprezintă o modalitate de calcul a produselor parțiale. Blocurile ce conțin termeni de forma sunt porți ȘI. Fiecare nivel logic conține sumatoare elementare legate în serie ca și în cazul sumatorului cu propagare succesivă a transportului. Transportul de pe un nivel este propagat la următorul nivel și în cele din urmă, dacă valoarea finală a transportului este 1, acest lucru se va traduce prin setarea celui mai semnificativ bit al produsului final. Rândul de sumatoare așezate la început calculează o sumă a primelor două produsuri parțiale, urmând ca mai apoi fiecare etaj de sumatoare să adauge un produs parțial la suma anterior calculată.



**Figura 3.8.** Matrice de sumatoare elementare pentru înmulțirea a două numere fără semn de câte patru biți

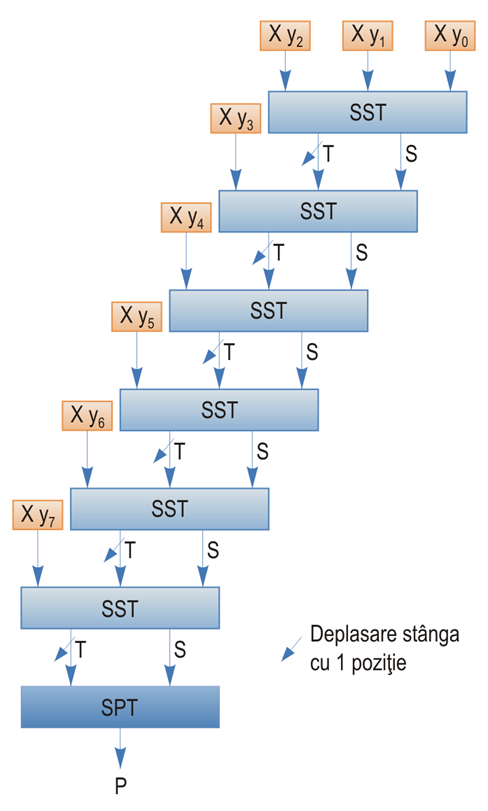
Se poate crea o celulă combinațională care să îmbine funcționalitatea porții ȘI logic și a sumatorului elementar pentru doi biți (figura 3.9.). Acest bloc de calcul este reprezentat de următoarea expresie aritmetică:



**Figura 3.9.** Celulă pentru înmulțirea matricială a numerelor fără semn

În celula M ilustrată mai sus, intrarea este asociată bitului corespunzător produsului parțial de la etajul anterior. Un circuit de înmulțire a biți va utiliza astfel de celule. Celulele aflate la extremități vor avea intrările setate la 0. Timpul execuției înmulțirii depinde de performanța sumatoarelor, iar întrucât acestea sunt sumatoare cu propagare succesivă a transportului, performanța întregului sistem este dată de propagarea transportului prin circuit în cazul cel mai defavorabil. Fără a lua în calcul diferența dintre celulele interne și cele periferice, timpul de execuție este , unde reprezintă timpul de propagare al transportului în cadrul unei singure celule.

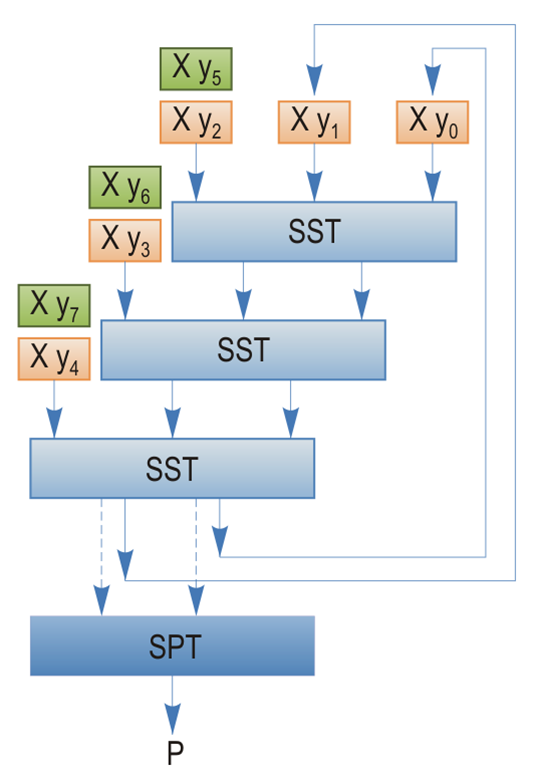
Pentru îmbunătățirea performanței înmulțitorului matricial, se poate lua în calcul folosirea unor sumatoare cu salvarea transportului (SST). Acestea au un timp de execuție mai redus datorită amânării adunării transportului până la ultimul etaj. Faptul acesta duce la o propagare mai rapidă a produsului parțial, eliminând de asemenea timpul necesar propagării transportului în rândul sumatoarelor de la același nivel.



**Figura 3.10.** Circuit de înmulțire matricială utilizând sumatoare cu salvarea transportului

Figura 3.10. Ilustrează schema bloc a circuitului de înmulțire a două numere pe 8 biți ce utilizează sumatoare cu salvarea transportului. Blocurile notate cu reprezintă produse parțiale pe 8 biți calculate cu ajutorul porților ȘI. Fiecare ieșire S este conectată la următorul sumator cu salvarea transportului, în timp ce bitul de transport T este deplasat cu o poziție, pentru a corespunde propagării normale a transportului, și apoi este transferat următorului transport. Structura utilizează un număr de șase sumatoare cu salvarea transportului, ultimul sumator fiind unul cu propagarea transportului (SPT), utilizat pentru a aduna transportul la produsul final.

Acest tip de sumator poate fi util pentru valori moderate ale lui . În cazul depășirii unor astfel de valori, însă, se poate ajunge la utilizarea unui număr excesiv de sumatoare cu salvarea transportului. Cu toate acestea, tehnica poate fi utilizată în continuare în cazul în care sumatorul se împarte în *k* segmente de câte *m* biți. În acest caz sunt calculate doar *m* produse parțiale, procesul fiind repetat de *k*  ori, iar rezultatele adunându-se la final prin utilizarea unui sumator cu propagarea transportului.



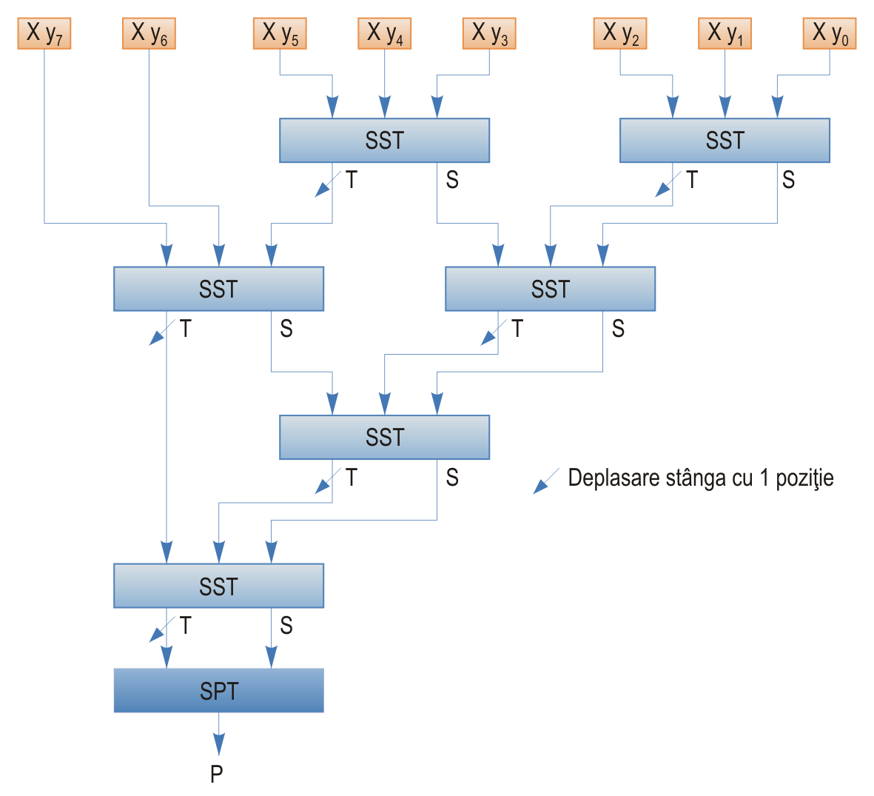
**Figura 3.11.** Circuit de înmulțire matricială cu două treceri

În figura precedentă se utilizează algoritmul de înmulțire cu două treceri. Primii 5 biți, mai puțin semnificativi, sunt înmulțiți la prima trecere (intrările utilizate la prima trecere sunt indicate în dreptunghiurile cu fundal deschis la culoare). Rezultatul obținut în urma primei treceri este folosit în etapa a doua, când este combinat cu următoarele trei produse parțiale. Finalitatea este dată de adunarea realizată cu ajutorul unui circuit de însumare cu propagarea transportului. În mod obișnuit acest tip de înmulțitor va necesita utilizarea unui semnal de ceas. Acest lucru se poate evita, însă, în cazul în care matricea din figura abordată are o dimensiune destul de mare pentru a nu permite niciunui semnal să se propage de la intrare la ieșire înainte ca sumatorul să-și fi stabilizat ieșirile.

### **3.5.3. Arborele Wallace**

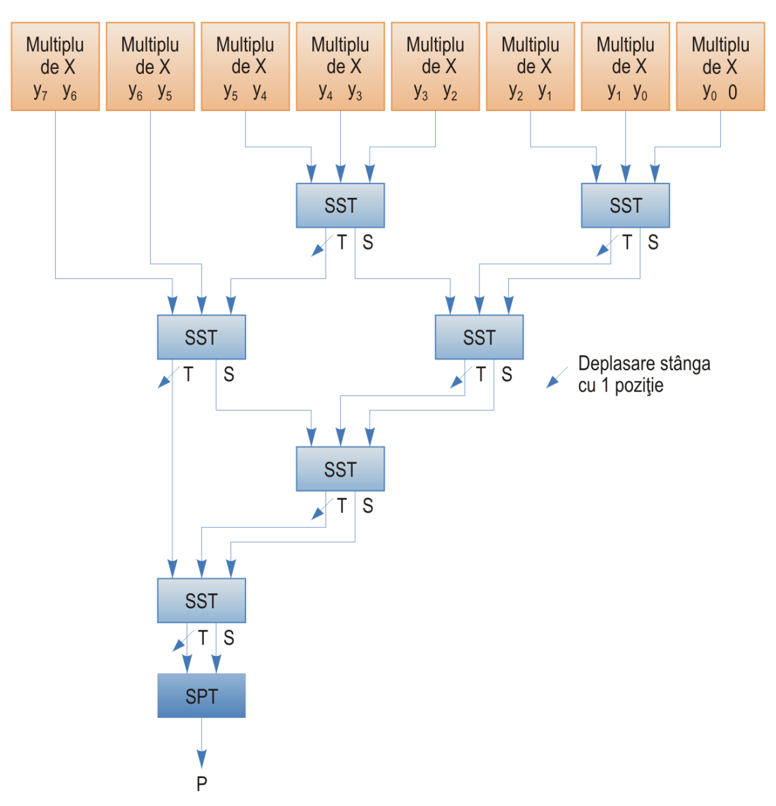
Tehnica prezentată anterior pentru înmuțirea a două numere pe n biți presupune adunarea a n produse parțiale și necesită un timp de execuție O(n). Timpul acesta poate fi redus la O(log n) prin utilizarea unui arbore.

Metoda lui C.S.Wallace pentru adunarea produselor parțiale utilizând structura de arbore presupune utilizarea nivelelor multiple de sumatoare cu salvarea transportului grupate în structura de arbore. Presupunând că produsele parțiale sunt generate simutan, în primul nivel al arborelui numerele sunt grupate câte trei și se utilizează câte un sumator cu salvarea transportului pentru adunarea numerelor din fiecare grup[[2]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29). Problema adunării a n produse parțiale se reduce la la problema adunării a 2\*n/3 produse parțiale, în fiecare nivel al arborelui fiind prezente sumatoare cu salvarea transportului ce permit adunarea a trei produse parțiale și produs o sumă și un transport, ce vor reprezenta intrări pentru urmărorul nivel al arborelui. Acest proces continuă până rămân de adunat doar două numere, operație realizată prin intermediul unui sumator cu propagarea transportului, în general acesta fiind un sumator cu anticiparea transportului. Fiecare nivel reduce numărul produselor parțiale cu 1.5, motiv pentru care înmulțirea poate fi realilzată utilizând acest arbore în aproximativ O(.În figura 3.12 este prezentată schema bloc pentru înmulțirea a două numere de câte 8 biți utilizând arborele Wallace. Cele 8 intrări prezente în schemă(X, X,..., X) reprezintă înmuțirea primului termen cu câte un bit din al doilea(produse parțiale).



**Figura 3.12** Schema bloc a unui circuit de înmulțire cu arborele Wallace pentru numere de 8 biți

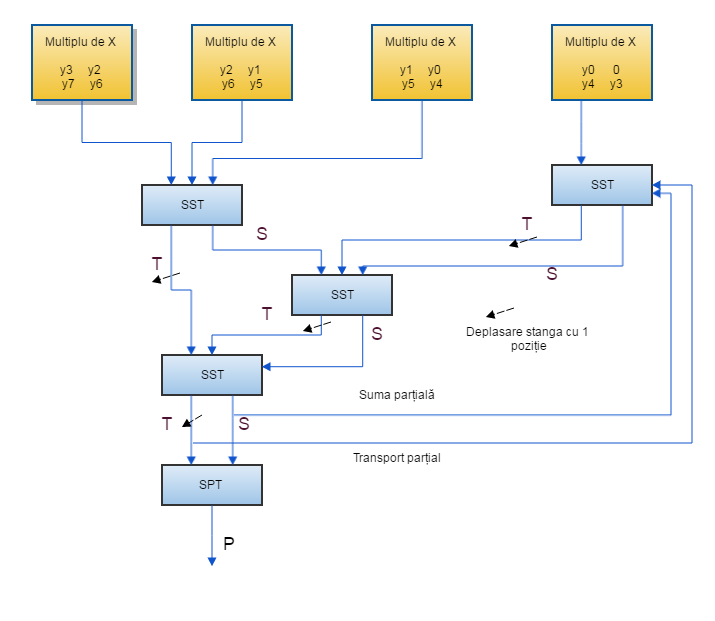
De multe ori, acest algoritm este utilizat împreună cu o altă metodă pentru a obține circuite rapide de înmulțire. Una dintre ele este metoda lui Booth, metodă utilizată pentru generarea produsele parțiale. Produsele obținute prin acestă tehnică vor fi adunate printr-un arbore Wallace. Figura 3.13 prezintă un exemplu de utilizare a metodei Booth împreuna cu arborele Wallace.



**Figura 3.13** Înmulțirea prin metoda Booth și un arbore Wallace

Figura 3.13 prezintă înmulțirea folosind metoda Booth cu testarea a doi biți pentru fiecare etapă. Produsele parțiale sunt însumate apoi printr-un set de sumatoare cu salvarea transportului structurate sub forma unui arbore Wallace.

O optimizare a acestui înmulțitor ar fi realizată prin reducerea numărului de circuite necesare. Înmulțirea numerelor mari necesită utilizarea unui număr mare de sumatoare cu salvarea transportului, valoare ce poate fi redusă prin treceri multiple în arborele Wallace. Acest procedeu presupune adunarea unui număr de rezultate parțiale între ele și adăugarea sumei și a transportului, obținut din această adunare, ca și intrări în noua etapă de adunare a produselor parțiale, etapă în care alte produse parțiale vor fi însumate. Acest procedeu se repetă până când toate produsele parțiale au fost adunate, moment în care suma și transportul obținute sunt însumate printr-un sumator cu propagarea transportului. O exemplificare a acestei optimizări este vizibilă în figura 3.14.



**Figura 3.14** Circuit de înmulțire cu două treceri ce combină metoda Booth cu un arbore Wallace

Figura 3.14 prezintă un circuit de înmulțire cu două treceri. În prima trecere se adună cele patru produse parțiale rezultate din cei patru biți mai puțin semnificativi, a doua trecere însumând celelalte patru produse parțiale împreună cu suma și transportul parțial rezultate.

## **3.6. Înmulțirea în virgulă mobilă**

Considerând două numere reprezentate în virgulă mobilă:

Operația de înmulțire a acestor numere se reprezintă astfel:

Pentru a realiza înmulțirea a două numere reprezentate în virgulă mobilă, este necesară, deci, utilizarea unor operații de adunare a numerelor întregi, înmulțire similară cu cea a numerelor întregi, normalizare, rotunjire precum și stabilire a semnului rezultatului final. În acest scop vom prezenta mai întâi algoritmul propriu-zis, la un nivel mai abstract, iar mai apoi vom detalia operațiile aflate la un nivel cu o granularitate mai mare și apropiat de implementare.

Algoritmul testează la primul pas fiecare dintre cei doi operanzi implicați pentru a verifica dacă vreunul din ei este egal cu 0. În caz afirmativ, se ajunge la finalul algoritmului, rezultatul luând valoarea 0.

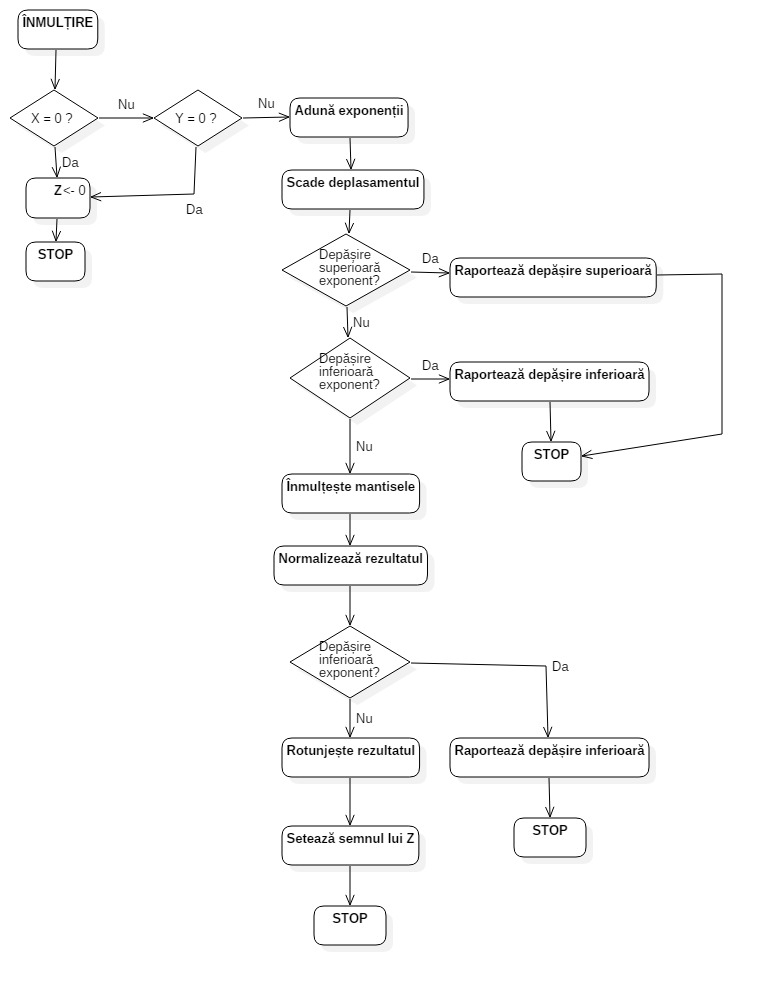
În cazul în care operanzii sunt nenuli, se trece la următorul pas ce presupune adunarea exponenților (defapt a caracteristicilor). Dacă exponenții sunt deplasați atunci în urma adunării lor va rezultat un deplasament dublat. În acest scop se scade deplasamentul, la următorul pas al procesului.

În acest punct se va testa încadrarea exponenților în limita admisă, deoarece prin adunarea acestora poate rezulta fie o depășire superioară sau inferioară. Aceasta este raportată, iar algoritmul se termină în acest caz.

Următoarea etapă, în cazul în care exponentul rezultat este încadrat între limitele admise, este aceea de înmulțire a mantiselor. După aceasta se va normaliza rezultatul obținut, testându-se încă o dată dacă exponentul rezultat se află domeniul permis. Deoarece rezultatul produsului va avea o lungime dublă față de cea a operanzilor, se va realiza rotunjirea rezultatului, biții suplimentari fiind eliminați.

În final se va stabili semnul produsului, acesta fiind pozitiv (0) în cazul în care biții de semn ai operanzilor inițiali au fost identici, iar în caz contrar va fi negativ (1).

Pașii algoritmului sunt ilustrați în figura 3.2. [[2]](file:///C:\Users\Mati\Desktop\Proiect_doc.docx#_okwn25hjzs29)



**Figura 3.2.** Înmulțirea numerelor în virgulă mobilă

## **3.7. Considerații de precizie**

Numerele în virgulă mobilă sunt aproximări ale unor numere reale ce nu pot fi reprezentate exact. Pentru o reprezentare cât mai realistă a acestora este necesară efectuarea unei operații de rotunjire.

Realizarea cu acuratețe a operațiilor aritmetice în virgulă mobilă necesită utilizarea unor cifre suplimentare. Dintre aceste cifre, *cifrele de garda g* sunt plasate la dreapta cifrelor de precizie ale mantisei pentru a evita pierderea acestor cifre. Aceste cifre pot fi deplasate la stânga în pozițiile biților de precizie prin operația de normalizare. După această operație se verifică valoarea cifrelor aflate la dreapta biților de precizie: dacă valorile acestora sunt diferite de 0 este necesarea rotunjirea numărului. *Cifra de rotunjire r* este plasată la dreapta cifrelor de gardă, astfel încât după deplasarea prin normalizare a cifrelor de gardă, rezultatul este rotunjit conform valorii acestei cifre.

Standardul IEEE specifică patru moduri de rotunjire:

* Rotunjirea spre plus infinit
* Rotunjirea spre minus infinit
* Rotunjirea spre zero
* Rotunjirea spre cel mai apropriat număr par

Doi biți suplimentari sunt suficienți pentru rotunjirea în primele trei moduri de

rotunjire. Rotunjirea spre cel mai apropriat număr par necesită un bit suplimentar, *bit-ul sticky)*, ce va fi setat când există biți diferiți de 0 la dreapta bitului de rotunjire. Acest tip de rotunjire este prevăzut pentru situațiile în care numărul real se află la jumătatea intervalului dintre două reprezentări în virgulă mobilă.

## **3.8. Soluția finală și obiectivul proiectului**

Proiectul de față propune implementarea unui înmulțitor care să funcționeze în cazul înmulțirii numerelor în virgulă mobilă, în formatul IEEE 754. Se urmărește utilizarea cât mai eficientă a resurselor, precum și obținerea unui timp de execuție cât mai redus, pentru a crește performanța sistemului. În cazul de față este dorită și facilitarea unei ușoare înțelegeri a modului de funcționare în cazul componentelor utilizate. Se ține cont și de dimensiunea operanzilor, care nu este foarte mare, lăsând loc implementării unor algoritmi ce au succes și în cazul în care nu se lucrează cu operanzi de dimensiuni mari.

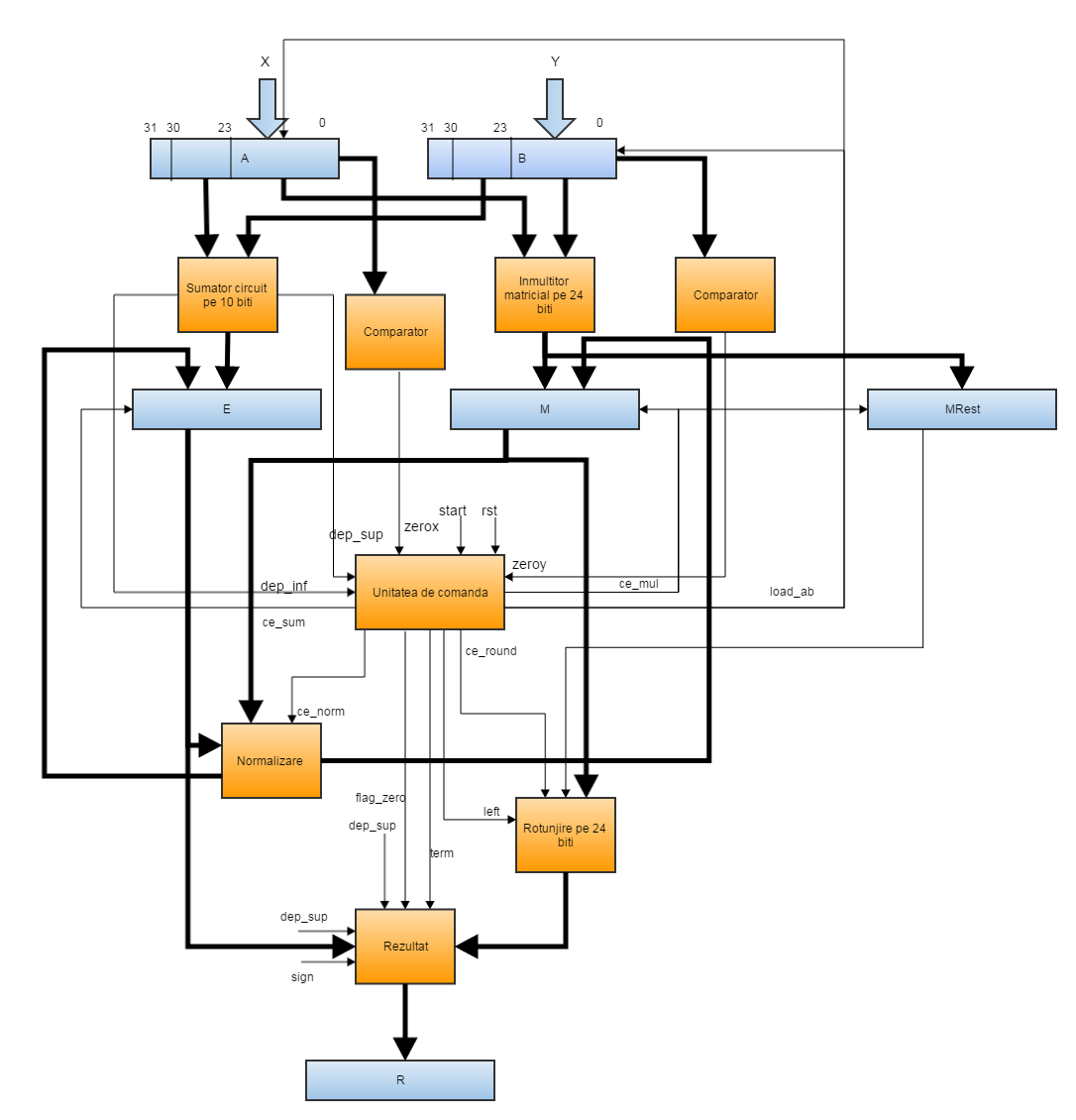
Soluția aleasă ia în calcul faptul că adunarea, respectiv înmulțirea a doi operanzi întregi se poate realiza în diverse modalități, fiecare urmărind un scop bine definit. Fie că este vorba de obținerea unei performanțe mai bune în ce privește timpul de execuție sau economisirea unor resurse în calculele combinaționale, alegerea strategiei va implica întotdeauna un compromis.

Pentru problema de față am ales lucrul cu un *sumator cu anticiparea transportului.* Acesta este eficient datorită creșterii vitezei de răspuns, prin reducerea timpului necesar formării transportului (spre deosebire de sumatoarele cu propagare succesivă a transportului). Alte opțiuni ar include alegeri precum sumatorul cu selecția transportului sau cel cu salvarea transportului, primul fiind evitat pentru utilizarea de resurse în mod ineficient, iar cel de-al doilea fiind o alegere mai potrivită în cazul adunării a mai mult de două numere. În cazul nostru, pentru motivele enumerate, vom realiza adunarea a două numere în format cu precizie simplă folosind tipul de sumator menționat.

Dintre algoritmii utilizați pentru înmulțirea numerelor întregi vom utiliza aici algoritmul de *înmulțire matriceală.* Acesta este preferat datorită rapidității execuției, respectiv, a calculului produsului într-un singur pas. În cadrul structurii sale se va folosi sumatorul elementar, respectiv sumatorul cu propagare succesivă a transportului, ca parte integrantă. Datorită faptului că nu se operează cu numere de dimensiuni mari, utilizarea unui înmulțitor matriceal cu sumatoare cu propagarea transportului (SPT), este comparabil ca și performanță cu înmulțitor matriceal ce folosește sumatoare cu salvarea transportului (SST), cel dintâi fiind ales pentru implementare în lucrarea de față.

# **4. Proiectare şi implementare**

## **4.1. Schemă înmulțitor**



**Figura 4.1** Schema bloc înmulțitor pe 32 biți în virgulă mobilă

În figura 4.1 este ilustrată schema înmulțitorului în virgulă mobilă a numerelor pe 32 de biți, numere a căror reprezentare respectă standardul IEEE 754. Înmulțitorul se folosește de o unitate de comandă pentru a sincroniza operațiile efectuate de celelalte componente ale sistemului. În funcție de intrările primite și de starea curentă, unitatea setează anumite semnale de ieșire. Aceste semnale determină momentul în care registrele sincrone vor fi scrise și momentul în care componentele de rotunjire, normalizare și reprezentarea rezultatului vor funcționa.

Componentele A si B sunt registre sincrone în care se încarcă numerele pe 32 de biți. Componentele E, M, MRest și R sunt tot registre sincrone. În registul E este reținută suma exponenților celor două numere, în M și MRest este reținut produsul mantiselor(în M se încarcă partea mai semnificativă a produsului și în MRest biții mai puțin semnificativi), în R se încarcă rezultatul operației de înmulțire.

Sunt utilizate două comparatoare pentru determinarea operanzilor egali cu 0. În acest caz, semnalele de ieșire ale unității de comandă flag\_zero și term vor fi setate, situație în care algortmul ia sfârșit și rezultatul produs de sistem este 0.

O componentă de adunare este utilizată pentru adunarea exponenților și scăderea deplasamentului suplimentar adăugat în urma acestei operații. Biții cei mai semnificativi ai sumatorului vor fi utilizați pentru determinarea cazurilor in care loc o depășire superioară sau inferioară a exponentului deplasat.

O componentă de înmulțire reprezentată printr-un înmulțitor matricial va fi utilizată pentru realizarea înmulțirii mantiselor. Componenta conține un înmulțitor matricial și două

O componentă de normalizare va fi utilizată pentru menținerea operanzilor și produsului acestora în formatul standard IEEE 754. Această componentă va prelua mantisa și exponentul numărului și va realiza operații de scădere/adunare asupra exponentului și de deplasare dreapta/stânga asupra mantisei până când numărul este reprezentat de mantisă și exponent în forma unui număr flotant în care partea întreagă este egală cu 1.

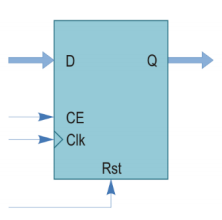
O componentă de rotunjire va fi utilizată pentru aproximarea rezultatului în funcție de toți cei 48 de biți produși în urma înmulțirii mantiselor. Acesta are 48 de biți, dar mantisa unui număr poate reține doar 24; pentru a nu pierde cei mai nesemnificativi biți ai produsului, biții reținuți în registrul M sunt rotunjiți în funcție de acești biți.

O componentă pentru reprezentarea rezultatului este utilizată pentru a returna rezultatul operației de înmulțire. Astfel, dacă este depistat un operand egal cu 0, rezultatul este tot 0; dacă se constată o depășire superioară asupra exponentului, rezultatul va fi

reprezentat prin valoarea specială a formatului standard pentru valoarea ; dacă se constată o depășire inferioară a exponentului, rezultatul va fi reprezentat prin valoarea specială a formatului standard pentru valoarea . Dacă nici una din situațiile menționate nu este întâlnită, rezultatul va fi reprezentat prin mantisa și exponentul produs, iar semnul numărului va fi determinat din semnele operanzilor inițiali.

* **Registru de n biți cu resetare sincronă**

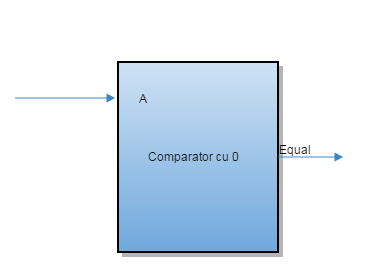
Intrarea de ceas a registrului este Clk, intrarea de date este vectorul D de n biți, iar ieșirea de date este vectorul Q de n biți. Dacă semnalul Rst este 1 logic, registrul este resetat în mod sincron, ieșirile sale fiind resetate la 0 logic. Dacă semnalul CE (Clock Enable) este 1 logic, la frontul crescător al semnalului de ceas se realizează încărcarea paralelă a registrului cu datele aplicate la intrare. În figura 4.2 este prezentat simbolul registrului sincron.



**Figura 4.2** Simbolul registrului cu resetare sincronă

* **Comparatorul pe n biți cu 0**

Intrarea comparatorului este A, iar ieșirea de date este Equal. Dacă semnalul A este egal cu 0, semnalul Equal este setat pe 1, iar în caz contrar acesta este 0. Comparatorul este realizat în mod generic, motiv pentru care poate compara numere de n biți cu reprezentarea pe n biți a valorii 0. În figura 4.3 este prezentat simbolul comparatorului pe n biți cu 0.



**Figura 4.3** Simbolul comparatorului cu 0

## **4.2. Sumator**

Sumatoarele necesită operanzi pe 10 biți întrucât suma a două numere pe 8 biți poate rezulta într-un număr pe 8 biți și scăderea deplasamentului specifică necesitatea utilizării numerelor cu semn.

### **4.2.1 Sumator cu anticiparea transportului**

Implementarea sumatorului cu anticipare este folosit în proiect pentru adunarea exponenților, cuprinși în biții [23-30] din formatul standard IEEE 754 a numerelor pe 32 biți.

Sumatorul conține următoarele intrări: două semnale de intrare pe 10 biți reprezentând exponenții numerelor, un semnal de intrare pentru transport, un semnal de iesire pe 10 biți reprezentând exponentul produs prin adunare și un semnal de ieșire pentru transportul produs în urma adunării.

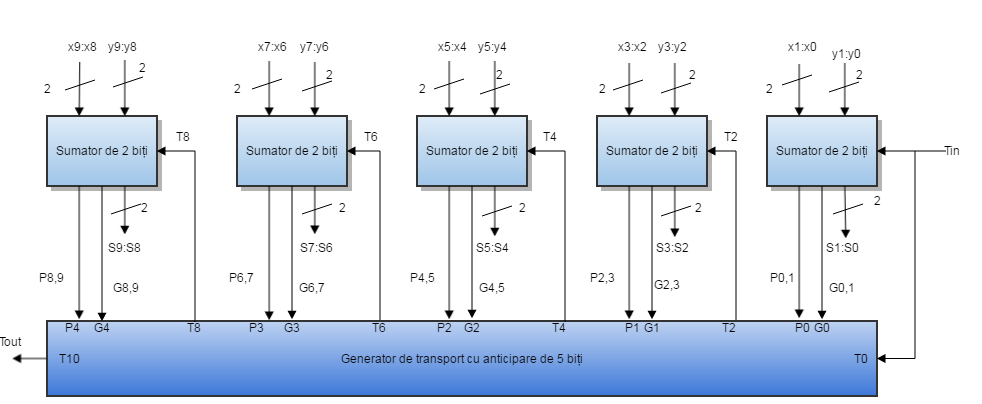
Sumatorul de 2 biți este un sumator cu anticiparea transportului în care intrările sunt reprezentate prin două semnale de 2 biți și un transport de intrare, iar ieșirile sunt reprezentate prin două semnale de un bit care reprezintă funcțiile pentru generarea, respectiv propagarea transportului pe grup de 2 biți pentru biții de intrare și un semnal pe 2 biți care reprezinta suma biților fără adunarea informațiilor legate de transport.

Ecuațiile funcțiilor pentru generarea și propagarea transportului pot fi deduse din următoarele ecuații:

Utilizând ecuațiile descrise anterior pot fi calculate funcțiile pentru generarea și propagarea transportului pe grup de 2 biți folosind următoarele ecuații:

Generatorul de transport utilizat de acest sumator preia o funcție pentru generarea și una pentru propagarea transportului pe grup de 2 biți si produce transportul pentru sumatorul cu anticiparea transportului de 2 biți. Ecuația acestuia este:

Schema sumatorului cu anticiparea transportului pe 10 biți este prezentată in figura 4.4 împreună cu generatorul de transport menționat anterior.



**Figura 4.4.** Schema bloc a unui sumator de 10 biți cu anticiparea transportului pe grupe de 2 biți.

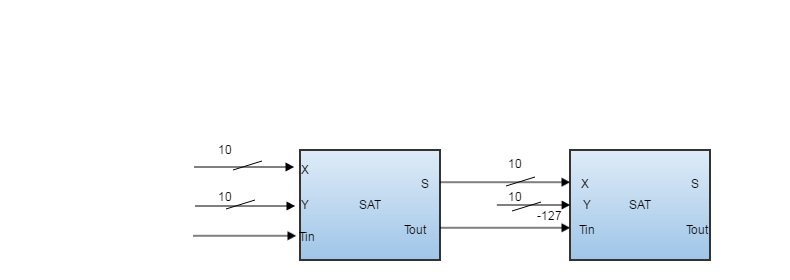
### **4.2.2 Sumatorul sistemului**

Sumatorul sistemului se folosește de sumatorul cu anticiparea transportului descris anterior pentru a realiza adunarea exponenților a două cu numere, urmată de scăderea deplasamentului suplimentar produs de această adunare.

Componenta sumatorului pentru exponenți este alcătuită din două sumatoare cu anticiparea transportului pe 10 biți. Primul sumator va fi folosit pentru adunarea exponenților celor doua numere flotante implicate în înmulțire, în timp ce al doilea va fi utilizat pentru scăderea deplasamentului suplimentar adăugat în urma primei sume.

Intrările componentei sunt 2 semnale de 10 biți, pe primele 8 poziții regăsindu-se exponenții numerelor șin un transport de intrare, iar ieșirile sistemului sunt reprezentate printr-un semnal de 10 biți ce indică rezultatul adunării celor doi exponenți și un transport de ieșire produs de această operație.

Schema componentei de adunare a exponenților este prezentată în figura 4.5.



**Figura 4.5.** Schema bloc a unui sumator de 10 biți al sistemului

## **4.3. Detecție depășiri**

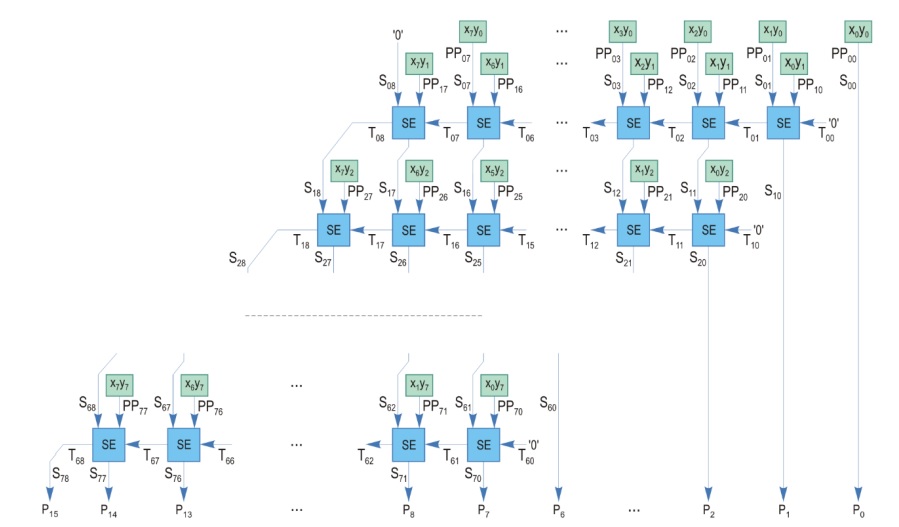
## **4.4. Determinarea semnului**

## **4.5. Înmulțitor**

Înmulțirea mantiselor se va realiza cu ajutorul înmulțitorului matriceal pentru înmulțirea numerelor întregi fără semn, ce utilizează sumatoare elementare pe un bit și blocuri logice ȘI. Înmulțirea va fi realizată pe 24 de biți( cei 23 de biți ai mantisei si bitul ascuns).

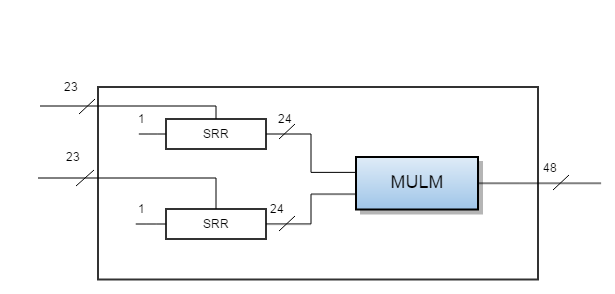
Înmulțitorul se folosește de produse parțiale pentru efecutarea operației. Aceste produse sunt simple porți ȘI între doi biți ai celor două numere și toate aceste produse se pot calcula simultan. Acești termeni sunt însumați cu ajutorul unei matrice de 24 \* 23 sumatoare elementare, dintre care un număr de 24 sumatoare elementare pot fi înlocuite prin semisumatoare elementare. Circuitul rezultat este similar cu un sumator bidimensional cu transport succesiv.

În figura următoare este prezentată schema bloc a înmulțitorului, fiind reprezentat un înmulțitor pe 8 biți întrucât unul pe 24 biți este greu de urmărit. Semnalele de transport din fiecare rând de sumatoare elementare sunt conectate ca și la un sumator cu propagarea succesivă a transportului. Primul rând de sumatoare elementare adună primele două produse parțiale. Următoarele rânduri de sumatoare elementare adună câte un produs parțial la suma rezultată.



**Figura 4.6.** Schema circuitului de înmulțire matriceală pentru numere de câte 8 biți cu specificarea semnalelor utilizate.

În figura 4.7 este descrisă componenta utilizată pentru înmulțirea mantiselor. Această componentă preia mantisele celor doi operanzi(valori pe 23 biți) pentru a realiza înmuțirea pe acestea. Înainte de a se realiza înmulțirea propriu-zisă, trebuie ținut cont de faptul că formatul IEEE 754 conține un bit ascuns, bit ce reprezintă partea întreagă a numărului. Pentru a adăuga acest bit în valoarea mantisei, vor fi folosiți doi regiștrii de șiftare la dreapta, regiștrii în care inițial se vor încărca valorile mantiselor, după care printr-o operație de șiftare va fi adăugat bitul ‘1’ pe pozițiiile semnificative ale acestora. După efectuarea acestei operatii, operanzii înmulțirii vor avea 24 de biți, operația de înmulțire fiind realizată prin înmulțirea matricială.



**Figura 4.7.** Schema înmulțitorului sistemului pe 24 biți

## **4.6. Circuit de normalizare și rotunjire**

### **4.6.1 Circuit de normalizare**

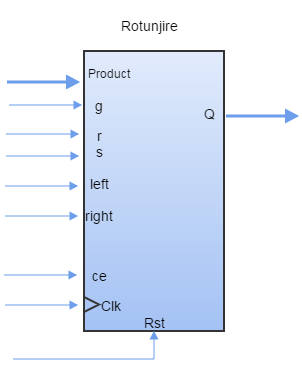
### **4.6.2 Circuit de rotunjire**

Rotunjirea utilizată este cea spre cel mai apropriat număr par. Acest tip de rotunjire este prevăzut pentru situațiile în care numărul real se află la jumătatea intervalului dintre două reprezentări în virgulă mobilă.

Metoda utilizează 3 biți suplimentari, reținuți în urma procesului de înmulțire al mantiselor. Primii 24 de biți obținuți din înmulțire sunt reținuți în câmpul corespunzător mantisei, iar următorii 3 biți ai acestuia sunt reținuți pentru realizarea procesului de rotunjire. Primul bit reținut este numit cifră de gardă și este folosit în realizarea procesului de rotunjire, fiind considerată cifra mai puțin semnificativă a rezultatului și cifra în urma căreia biții mantisei sunt rotunjiți. Al doilea bit reținut este numit cifră de rotunjire și este utilizat în locul cifrei de gardă în operația de rotunjire în cazul în care are loc o deplasare la stânga în operația de înmulțire(normalizarea). Al treilea bit reținut este numit bit sticky și este folosit în operația de rotunjire în cazul în care are loc o deplasare la dreapta în operația de înmulțire(denormalizare). Acest bit va fi și va rămâne setat când există biți diferiți de 0 la dreapta bitului de rotunjire.

Circuitul de rotunjire implementat conține următoarele intrări: o intrare de ceas(Clk), un semnal de intrare reprezentând primii 24 de biți obținuți în urma operației de înmulțire( Product), un semnal de intrare ce indică valoarea cifrei de gardă(g), un semnal de intrare ce indică valoarea cifrei de rotunjire(r), un semnal de intrare ce indică valoarea bitului sticky(s), un semnal de intrare ce specifică dacă a existat o deplasare la stânga în operația de înmulțire a numerelor în virgulă mobilă(left), un semnal de intrare ce specifică dacă a existat o deplasarea la dreapta în operația de înmulțire a numerelor în virgulă mobilă(right), un semnal de intrare care specifică dacă operația de rotunjire este activată(ce), un semnal de intrare care specifică dacă are loc resetarea componentei de rotunjire(Rst) și un semnal de ieșire pe 24 de biți care reprezintă semnalul obținut prin rotunjirea semnalului de intrare pe 24 de biți. Simbolul componentei este vizibilă în figura 4.8.

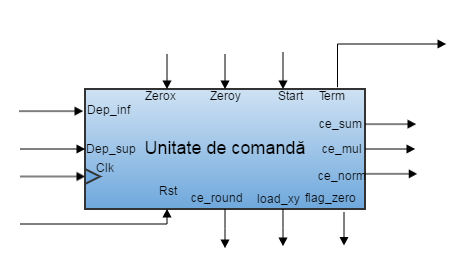
Dacă semnalul Rst este 1 logic, componenta de rotunjire este resetată în mod sincron pe frontul crescător al semnalului de ceas, ieșirile de date fiind resetate la 0 logic. În caz contrar, dacă semnalul ce este 1 logic, componenta rotunjește semnalul de intrare Product în felul următor: valoarea 1 pe semnalul left va realiza rotunjirea în funcție de semnalul r(dacă acesta este 1 ieșirea de date va fi obținută prin adunarea la semnalul Product a valorii 1, iar în cazul în care acesta este 0 ieșirea de date va fi reprezentată de semnalul Product), valoarea 1 pe semnalul right va realiza rotunjirea în funcție de semnalul s și valoarea 0 pe semnalele left și right va realiza rotunjirea în funcție de semnalul g.



**Figura 4.8** Simbolul componentei de rotunjire

## **4.7.Unitatea de comandă**

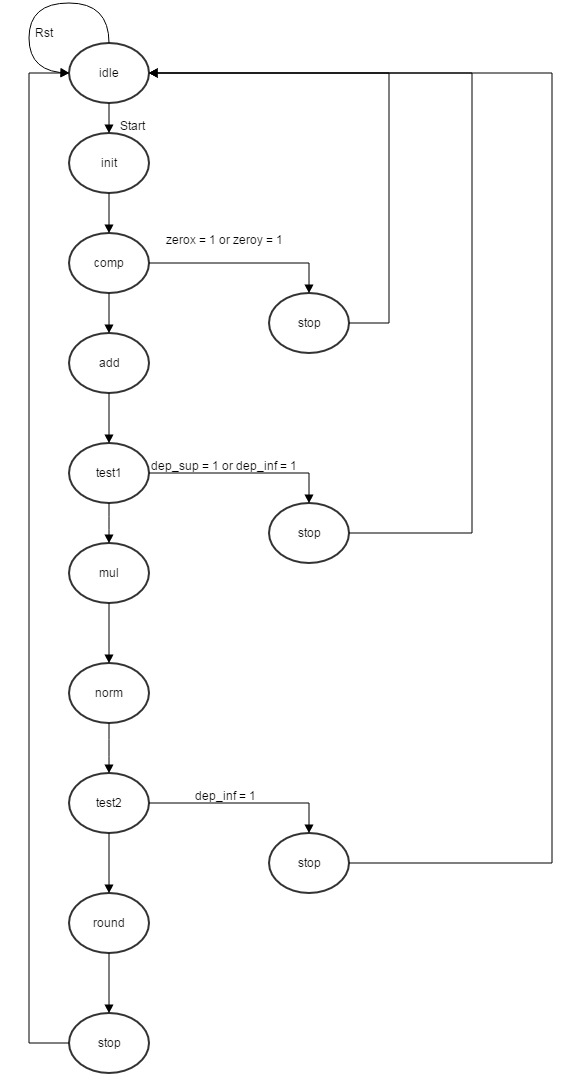
Înmulțitorul se folosește de o unitate de comandă pentru a sincroniza operațiile efectuate de celelalte componente ale sistemului. În funcție de intrările primite și de starea curentă, unitatea setează anumite semnale de ieșire astfel încât componentele sistemului să fie activate și folosite în ordinea necesară pentru implementarea operației de înmulțire, ordine precizată în capitolul 3, prin diagrama de stare a înmulțitorului descrisă în subcapitolul 6. În figura 4.9 este reprezentat simbolul acestei componente.



**Figura 4.9.** Simbolul unității de comandă

Intrările acestei componente sunt: intrarea de ceas(Clk), intrarea pentru resetarea componentei(Rst), semnalul ce semnalează depășirea superioară a exponentului (Dep\_sup), semnalul ce semnalează depășirea superioară a exponentului(Dep\_sup), intrarea ce indică egalitatea primului exponent al înmulțitorului cu 0(Zerox), intrarea ce indică egalitatea celui de-al doilea semnal al înmulțitorului cu 0 și semnalul Start care indică începerea operației de înmulțire. Ieșirile componentei sun următoarele: ieșirea ce indică semnalul de activare pentru registrul sincron al exponentului(ce\_sum), ieșirea ce indică semnalul de activare pentru registrul sincron al manitisei(ce\_mul), ieșirea ce indică semnalul de activare pentru componenta de rotunjirei(ce\_round), ieșirea ce indică semnalul de activare pentru componenta de normalizare(ce\_norm),ieșirea ce indică încărcarea registrelor celor doi operanzi(load\_xy), iieșirea ce indică rezultatul 0(flag\_zero) și ieșirea ce indică terminarea operației de înmulțire(Term).

Unitatea de comandă este implementată pe baza unui automat de stări, trecând sincron dintr-o stare în alta în funcție de starea curentă și semnalele de intrare. Fiecare stare, setează anumite semnale de ieșire la starea 1 logic, determinând realizarea anumitor operații asupra înmulțitorului. În figura 4.10. este prezentată diagrama de stare a acestui automat.



**Figura 4.10.** Diagrama de stare a automatului

Stările automatului permit funcționarea corectă a sistemului. Fiecare stare setează anumite ieșiri, valorile ieșirilor fiind prezentate în tabelul 4.11.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Stare** | **ce\_sum** | **ce\_mul** | **ce\_norm** | **ce\_round** | **load\_xy** | **flag\_zero** | **Term** |
| **idle** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **init** | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **comp** | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **add** | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **test1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **mul** | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **norm** | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **test2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **round** | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **stop** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figura 4.11.** Tabelul de tranziții al stărilor

## **4.8**. **Integrarea componentelor**

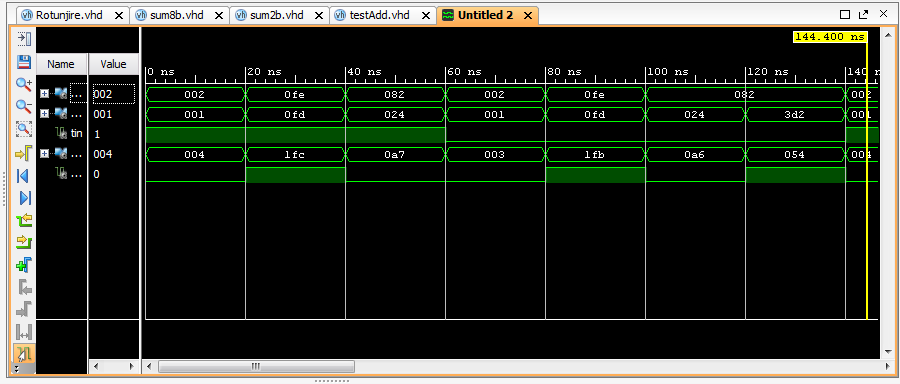
## **4.9. Schemă detaliată înmulțitor**

# **5. Rezultate experimentale**

## **5.1. Sumator**

### **5.1.1 Sumator cu anticiparea transportului**

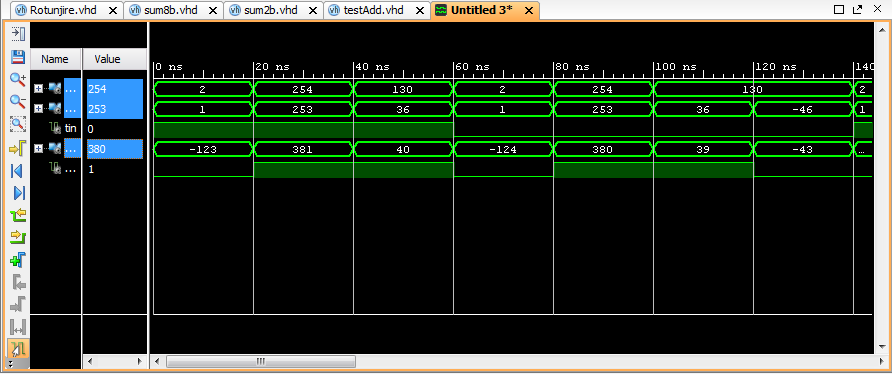
Simularea sumatorului pe 10 biți cu anticiparea transportului în mediul de lucru Vivado, are rezultatul afișat în figura 5.1, prezentată mai jos. Componenta de simulare testează funcționarea corectă a sumatorului pentru 5 valori specifice intervalului de valori: o adunaree apropiată de pragul minim de valori posibile, una apropiată de cel maxim, una cu operanzi aleși aproximativ la mijlocul intervalului de valori posibile cu transportul setat și încă două teste în care acest transport nu este setat. Astfel se testează sumatorul unor numere mici, respectiv, *001* și *002* cu transport ‘1’, (numerele sunt scrise în bază hexazecimală), rezultatul fiind corect: *004*. În mod similar se realizează adunarea numerelor *0fe* și *0fd* cu transport ‘1’, valori spre capătul superior al intervalului de valori, rezultatul fiind corect: *1fc*. Următoarele două teste sunt realizate pe același set de numere dar cu transport ‘0’, mod în care este verificată corectitudinea sumatorului în propagarea transportului, întrucât rezultatele obținute sunt corecte: *003* și *1fb*. Ultima testare implică numerele alese aleator de la jumătatea intervalului (una din cele două): *082* și *024* transport ‘1’, rezultatul lor fiind cel corect: *0*a6.



**Figura 5.1** Simulare funcționări sumatorului cu anticiparea transportului în mediul de lucru Vivado

### **5.1.2 Sumatorul sistemului**

Simularea sumatorului pe 10 biți utilizat de sistemul de înmulțire în mediul de lucru Vivado, are rezultatul afișat în figura 5.2, prezentată mai jos. Componenta de simulare testează funcționarea corectă a sumatorului pentru 5 valori specifice intervalului de valori: o adunare apropiată de pragul minim de valori posibile, una apropiată de cel maxim, una cu operanzi aleși aproximativ la mijlocul intervalului de valori posibile cu transportul setat și încă două teste în care acest transport nu este setat. Astfel se testează sumatorul unor numere mici, respectiv, *1* și *2* cu transport ‘1’, (numerele sunt scrise în bază zecimală), rezultatul fiind corect: *-123*. În mod similar se realizează adunarea numerelor *253* și *254* cu transport ‘1’, valori spre capătul superior al intervalului de valori, rezultatul fiind corect: *381*. Următoarele două teste sunt realizate pe același set de numere dar cu transport ‘0’, mod în care este verificată corectitudinea sumatorului în propagarea transportului, întrucât rezultatele obținute sunt corecte: *3* și *380*. Ultima testare implică numerele alese aleator de la jumătatea intervalului (una din cele două): *130* și *36* transport ‘1’, rezultatul lor fiind cel corect: *40*.

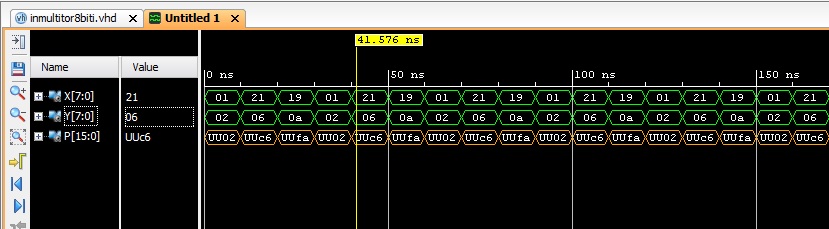


**Figura 5.2** Simulare funcționări sumatorului sistemului în mediul de lucru Vivado

## **5.2. Detecție depășiri**

## **5.3. Înmulțitor**

Simularea înmulțitorul matricial în mediul de lucru Vivado, are rezultatul afișat în figura 5.3., prezentată mai jos. Componenta de simulare testează funcționarea corectă a înmulțitorului pentru 3 valori specifice intervalului de valori: o înmulțire apropiată de pragul minim de valori posibile, una apropiată de cel maxim, și una cu operanzi aleși aproximativ la mijlocul intervalului de valori posibile. Astfel se testează înmulțirea unor numere mici, respectiv, *01*  și *02* , (numerele sunt scrise în bază hexazecimală), rezultatul fiind *UU02,* rezultat potrivit, deoarece primii biți (cei mai semnificativi) sunt necunoscuți încă, rezultatul propriu-zis al înmulțirii fiind trecut în locul biților mai nesemnificativi. În mod similar se realizează înmulțirea numerelor *21* și *06*, valori spre capătul superior al intervalului de valori, rezultatul fiind corect: *UUC6* (notație în hexazecimal). Ultima testare implică numerele alese aleator de la jumătatea intervalului (una din cele două): *19* și *0a*, rezultatul lor fiind unul satisfăcător, din nou, întrucât *UUfa* este într-adevăr răspunsul corect ( ; ; ).



**Figura 5.3.** Simulare funcționări înmulțitorului matricial în mediul de lucru Vivado

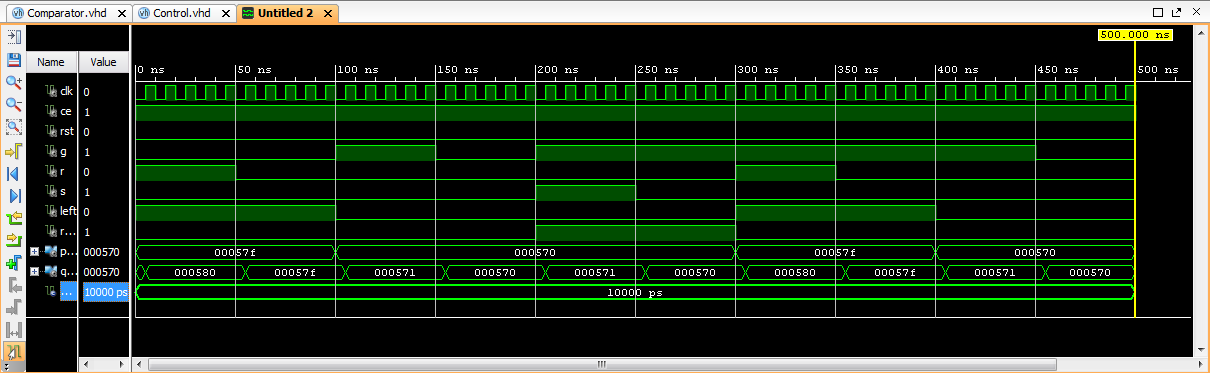
## **5.4. Normalizarea și rotunjirea**

### **5.4.1. Componenta de normalizare**

### **5.4.2. Componenta de rotunjire**

Simularea componentei de rotunjire a fost realizată în felul următor: pentru o anumită valoare reprezentând intrarea de 24 de biți a componentei au fost asignate diferite valori pentru intrările g(cifra de gardă), r(cifra de rotunjire) și s(bitul sicky). În funcție de intrările care specifică prezenta unei normalizări sau denormalizări realizate în algoritmul de înmulțire, rotunjirea semnalului va fi realizată în funcție de una dintre aceste intrări. Rotunjirea realizată este cea către cel mai apropriat număr par, astfel încât rotunjirea numărului se va face în funcție de bitul mai puțin semnificativ: dacă acesta este 1 se va adăuga 1 la intrarea de date, dacă nu semnalul de ieșire va fi reprezentat de valoarea intrării. Rezultatul simulării este vizibil în figura 5.4.

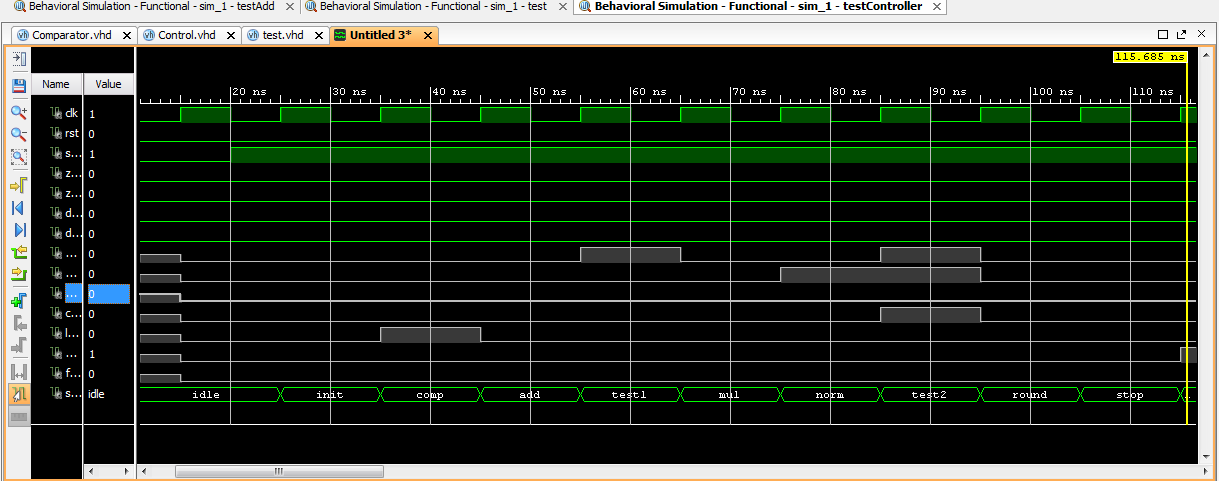
În primul test semnalul de intrare are valoarea *00057f,* valoarea intrării right este *1 (*rotunjirea se face în funcție de bitul sticky) și bitul sticky este *1.* Rezultatul este corect, întrucât aceasta este egal cu *000580* (00057f+1). Următorul test păstrează valorile semnalului de intrare și ale semnalui right, dar pun bitul sticky pe *0,* rezultatul fiind cel așteptat *00057f*. Al treilea test primește spre rotunjire valoarea *000570*, semnalele left și right au valoarea logică 0 (rotunjirea se face în funcție de bitul de gardă) și bitul de gardă are valoarea logică 1. Rezultatul rotunjirii este corect, întrucât acesta este egal cu *000571* (000570+1). Următorul test păstrează valorile semnalului de intrare propus spre rotunjire si ales semnalelor left și right, fiind modificată valoarea bitului de gardă la valoarea *0.* Se observă ca valoarea rotunjirii diferă cu modificarea acestui bit, rezultatul fiind egal cu *000570.* Al cincelea test păstrează valoarea dată spre rotunjire la valoarea *000570* și atribuie bitului left valoarea 1( rotunjirea se face în funcție de bitul de rotunjire). Rezultatul rotunjirii este corect, întrucât acesta este egal cu 000571 (000570+1). Următorul test păstrează valorile semnalului de intrare propus spre rotunjire si ales semnalelor left și right, fiind modificată valoarea bitului de rotunjire la valoarea 0. Se observă ca valoarea rotunjirii diferă cu modificarea acestui bit, rezultatul fiind egal cu 000570



**Figura 5.4.** Simulare funcționări componentei de rotunjire în mediul de lucru Vivado

## **5.5. Unitatea de comandă**

Simularea automatului de stare corespunzător unității de comandă este prezentată în figura 5.5. Este testată funcționarea corectă a aceste componente prin atribuirea valorii 1 logic a semnalului de intrare start. Acest semnal va porni funcționarea automatului prin scoaterea acestuia din starea idle. Odată setat acest semnal, automatul trece prin toate stările sale în următoarea ordine: init, shift, add, test1, mul, norm, tes2, round, stop. În fiecare stare sunt setate anumite semnale de ieșire conform tabelului 4.11.

Această simulare verifică funcționarea componentei în cazul în care execuția nu este întreruptă(cazurile de depășiri sau de existență a unui operand egal cu 0. 

**Figura 5.5.** Simulare funcționări unității de comandă în mediul de lucru Vivado

## **5.6. Proiect final**

# **6. Manual utilizator**

## **6.1. Simularea aplicației**

## **6.2. Configurarea proiectului pentru placuța**

## **6.3. Utilizarea înmulțitorului**

# **7. Concluzii**

# **Bibliografie**

[1] “Unitatea centrală de prelucrare (Central Processing Unit – CPU)”, [https://computerplusro.wordpress.com](https://computerplusro.wordpress.com/tag/numere-in-virgula-fixa-si-in-virgula-mobila/)

[2] Baruch Zoltan Francisc, “Structura sistemelor de calcul”, Editura U. T. PRES, Cluj-Napoca, 2002.

[3] Bazele aritmetice ale calculatoarelor numerice, 2004, [http://clodel.com/Capitolul\_1.pdf](http://clodel.com/facultate/Anul_II_2005_2006/CD-Informatica,%20ID,%20an%20II,%20sem%20I,%202005-2006/Suporturi_didactice/Arhitectura%20calculatoarelor%20I/Capitolul_1.pdf)

[4] “Nexys4 DDR™ FPGA Board Reference Manual ”, 2016, [https://reference.digilentinc.com/nexys4ddr\_rm.pdf](https://reference.digilentinc.com/_media/nexys4-ddr:nexys4ddr_rm.pdf)

[5] “PmodKYPD™ Reference Manual”, 2016,[https://reference.digilentinc.com/pmodKYPD\_rm.pdf](https://reference.digilentinc.com/_media/pmod:pmod:pmodKYPD_rm.pdf)

[6] <http://elf.cs.pub.ro/cn/wiki/lab/cn1/lab07>

[7]<http://users.utcluj.ro/~baruch/media/ssc/curs/SSC-Inmultire-1.pptx>

[8][http://andrei.clubcisco.ro/Adunarea si inmultirea in virgula mobila](http://andrei.clubcisco.ro/cursuri/1ii/laboratoare/Laborator%203%20Aritmetica%20in%20virgula%20mobila..PDF)