SITATEA "POLITEHNICA" TIMIŞOARA
ltatea de Electronică şi Telecomunicaţii

Ing. Lucian Aurel JURCA

PROCESOR ARITMETIC LOGARITMIC CU VIRGULĂ FLOTANTĂ

TEZĂ DE DOCTORAT

Conducător ştiinţific:

Prof. dr. ing. Mircea CIUGUDEAN

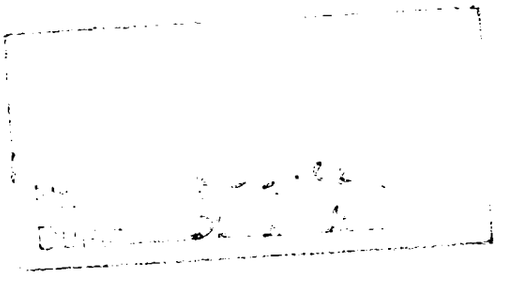
2002

UNIVERSITATEA "POLITEHNICA" TIMIŞOARA
Facultatea de Electronică şi Telecomunicaţii

Ing. Lucian Aurel JURCA

PROCESOR ARITMETIC LOGARITMIC CU VIRGULĂ FLOTANTĂ

TEZĂ DE DOCTORAT



Conducător ştiinţific:

Prof. dr. ing. Mircea CIUGUDEAN

2002

CUPRINS

Listă de abrevieri

ESC – Prefix ce indică o instrucţiune specifică procesorului aritmetic (*Escape*);
FP – Sistem de reprezentare a numerelor în virgulă flotantă (*Floating-Point*);
FPU – Unitate de calcul în virgulă flotantă (*Floating-Point Unit*);
LNS – Sistem logaritmic de reprezentare a numerelor (*Logarithm Number System*);
NDP – Procesor de date numerice, sau coprocesor (*Numeric Data Processor*);
NPX – Extensie numerică a procesorului, sau coprocesor (*Numeric Processor Extension*);
CU – Unitatea de control a coprocesorului (*Controll Unit*);
NEU – Unitatea de execuţie numerică a coprocesorului (*Numeric Execution Unit*);
CLA – Sumator cu transport anticipat (*Carry Look-Ahead Adder*);
CSA – Sumator cu conservarea transportului (*Carry Save Adder*);
BS – Circuit de deplasare programabil (*barrel shifter*);
ALU – Unitate aritmetică şi logică;
NZS – Numărător de zerouri semnifivative.

Introducere

Procesoarele din familia Intel 80×86, până la apariţia procesorului 80486DX, sunt unităţi de prelucrare a numerelor întregi, după diferite formate. Oricare însă dintre CPU - uri care operează cu numere întregi poate efectua prin soft operaţii cu numere reale, emulatoarele de coprocesor fiind un bun exemplu în acest sens. Totuşi utilizarea unui procesor dedicat operaţiilor matematice, adică a unui coprocesor (ca de exemplu 8087, 80287 sau 80387, apărute între 1980 şi 1986), a permis reducerea semnificativă a timpului de calcul.

Majoritatea programelor care utilizează coprocesorul şi care deci beneficiază de precizia şi viteza acestuia sunt aplicaţii profesionale şi ştiinţifice foarte răspândite cum ar fi: programe de prelucrare a datelor ştiinţifice, aplicaţii grafice, aplicaţii financiare sau aplicaţii funcţionând în timp real precum controlul maşinilor sau al proceselor, navigaţie, robotică, achiziţii de date etc.

Începând din 1989, odată cu apariţia procesorului Intel 80486DX, procesorul principal şi coprocesorul au putut fi integrate pe acelaşi cip. Cu toate acestea, coprocesorul este totuşi considerat ca şi unitate de calcul distinctă. Alţi termeni utilizaţi pentru acesta sunt conform [31]: procesor aritmetic, extensie numerică a procesorului (NPX), procesor de date numerice (NDP) sau procesor de calcul numeric, unitate în virgulă flotantă (FPU).

Acest ultim termen, stabilit prin extensie, nu este întotdeauna corect. Tipul unui coprocesor este dat de modul în care sunt reprezentate datele în interiorul său. Atunci când sunt cerute simultan o scară largă de reprezentare a numerelor reale cât şi precizie, se adoptă în general standardul sau sistemul de reprezentare în virgulă flotantă (FP). Încă de la început unităţile de calcul în virgulă flotantă au oferit suficiente avantaje pentru a fi larg răspândite şi de aceea performanţele lor au fost în mod continuu îmbunătăţite (ca de exemplu coprocesoarele integrate produse de firma Intel). Creşterea densităţii de integrare însă a permis dezvoltarea ca şi alternativă şi a unităţilor de calcul logaritmice [2], [4], [12], [18], [21], [22], *[37]*, *[40]*, *[42]*, *[44]*, [45], [50], [51], [57], [58], [59], [89], [90], [104]. Proprietăţile aritmeticii logaritmice permit dezvoltarea unor algoritmi de calcul competitivi, care în cazuri particulare oferă o viteză de operare mai mare decât sistemele de calcul în virgulă flotantă. În acelaşi timp, precizia pe care o oferă sistemul LNS (Logarithm Number System) în reprezentarea şi manipularea numerelor, este cu ceva mai mare decât a sistemului FP [57], pentru aceeaşi dimensiune a formatului. Având în vedere cele două aspecte, în lucrarea de faţă termenii FPU sau procesor LNS nu vor fi folosiţi pentru a desemna la modul general un coprocesor sau procesor aritmetic. În schimb va fi prezentată structura unui coprocesor hibrid care va avea calităţi superioare oricăruia dintre cele două tipuri menţionate.

Acest coprocesor va implementa cele patru operaţii aritmetice de bază şi va fi conceput astfel încât să fie foarte eficient în efectuarea unor sume/diferenţe multiple de produse/fracţii. Aşadar el va putea fi folosit în principal ca şi procesor de calcul,

specializat, putând implementa cu uşurinţă prin microprogramare diverşi algoritmi DSP (procesor de semnal), sau algoritmi pentru transformări de coordonate în cazul unor aplicaţii grafice. Având în vedere acest aspect, unitatea de execuţie numerică ce va fi prezentată în această lucrare va putea fi integrată într-un cip de sine stătător, în urma asocierii unui hardware care să controleze un set specializat de instrucţiuni.

În acelaşi timp, vor fi indicate soluţiile care să permită extinderea cu uşurinţă a capabilităţilor sale de operare rapidă şi în cazul calculului unor funcţii mai complexe cum ar fi rădăcina pătrată, ridicarea la putere, exponenţiala şi logaritmul în orice bază. În acest context, coprocesorul hibrid va putea fi integrat şi interfaţat, în cadrul aceluiaşi cip, cu procesorul principal, putând suporta un set lărgit de instrucţiuni prin care să se efectueze operaţii aritmetice simple sau calculul unor funcţii matematice complexe (cum ar fi cele trigonometrice sau transcedentale) dintre cele mai diverse.

Setul de instrucţiuni al coprocesorului va utiliza o codare similară cu a procesorului. Instrucţiunile care controlează coprocesorul sunt o extensie simplă a limbajului obişnuit al procesorului şi se numesc instrucţiuni "escape" (ESC) [31]. Cum procesorul şi coprocesorul sunt două unităţi de calcul distincte (chiar dacă sunt integrate pe acelaşi cip), coprocesorul este capabil să lucreze, într-un mod asincron, în paralel cu procesorul. Când procesorul întâlneşte o instrucţiune ESC (deci destinată coprocesorului), el calculează adresa memoriei dacă un operand este specificat şi trece imediat să decodeze şi să execute instrucţiunea următoare. Dacă aceasta nu este tot de tip ESC şi nici nu utilizează rezultatul instrucţiunii ESC în curs de execuţie, cele două procesoare lucrează în paralel. O codare îngrijită în asamblor (ţinând cont că o operaţie mai complexă poate dura sute de clock-uri) permite folosirea paralelismului pentru a obţine performanţe îmbunătăţite.

Operanzii sunt situaţi fie în memorie, fie în regiştrii coprocesorului, iar adresarea memoriei se face în acelaşi mod ca în cazul procesorului. Setul de instrucţiuni cuprinde instrucţiuni suplimentare de încărcare şi salvare pentru a transmite operanzi reali, întregi şi BCD între regiştrii coprocesorului şi memoria sistemului.

Indiferent de tipul sau destinaţia unui procesor aritmetic, în cadrul lui pot fi identificate două unităţi distincte: unitatea de control (CU) şi unitatea de execuţie numerică (NEU).

Unitatea de control efectuează interfaţarea cu magistrala şi decodarea instrucţiunilor coprocesorului, accede şi reţine operanzii şi opcodurile (cod maşină, cod executabil sau cod obiect) şi le transmite apoi către NEU. CU este responsabilă de sincronizarea coprocesorului cu procesorul. Sincronizarea este necesară când cele două procesoare trebuie să schimbe informaţii, să acceadă la rezultatele operaţiilor sau să decodeze instrucţiuni.

Unitatea de control este legată la liniile de stare ale procesorului şi este capabilă să aducă instrucţiuni în paralel cu procesorul, atunci când acesta le citeşte din memorie. Coprocesorul trebuie să citească şi să decodeze aceleaşi instrucţiuni

ca şi procesorul principal; trebuie deci să emuleze precitirile şi decodarea instrucţiunilor proprii procesorului asociat. De exemplu, coprocesorul trebuie să menţină o coadă de aşteptare de precitire de aceeaşi mărime cu a procesorului, să o manipuleze în aşa fel încât instrucţiunile să fie plasate sau retrase (cu ocazia iniţializărilor) în acelaşi timp cu procesorul.

Instrucţiunile destinate coprocesorului sunt inserate direct în codul destinat procesorului. Astfel procesorul şi coprocesorul decodează împreună clasa de instrucţiuni ESC. Procesorul decodează suficient instrucţiunile ESC pentru a diferenţia pe cele care fac referire la memorie de cele care nu fac. Dacă se face referire la memorie, procesorul calculează adresa operandului şi începe un ciclu de citire pe magistrală, dar ignoră data primită. Dacă nu se face referire la memorie, procesorul începe imediat să decodeze şi să execute instrucţiunea următoare. Procesorul ignoră operaţiile incluse în instrucţiunile ESC.

În caz de acces la memorie, NPX salvează adresa efectivă a operandului. Dacă un operand registru este specificat, valoarea acestui operand este salvată. Dacă extragerea unui operand durează mai mult de un ciclu de magistrală (un transfer de date) din cauza mărimii sau aliniamentului său, CU a coprocesorului ia controlul magistralei sistemului şi transferă restul operandului. La scrierea unui operand în memorie, după calculul adresei de către procesor, NPX salvează adresa şi începe ciclii de scriere în memorie a operandului.

Există instrucţiuni ale coprocesorului executate numai de CU (fără participarea NEU) care indică sau modifică starea coprocesorului, ca de exemplu: FINIT (iniţializarea NPX), FCLEX (ştergerea fanioanelor de excepţie), FSTSWAX (stocarea cuvântului de stare în registrul AX al CPU), FSTCW (stocarea cuvântului de control într-un operand în memorie).

Unitatea de execuţie numerică (NEU) este veritabilul coprocesor.

Aceasta efectuează toate operaţiile prin care se accede la regiştrii numerici ai NPX sau prin care se prelucrează datele conţinute de aceştia. Efectuează operaţii aritmetice, logice, transcedentale, furnizând şi un anumit număr de constante matematice înscrise într-un ROM intern al cipului. NEU conţine circuite specializate precum sumatoare rapide, circuite combinate de adunare/scădere, multiplicatoare rapide, circuite pentru împărţire şi de extragere a rădăcinii pătrate, circuite de deplasare programabile, circuite combinaţionale dedicate (cum ar fi circuitele de contorizare a numărului de zerouri de după virgulă, sau de anticipare a numărului acestora etc.) utilizate după nevoi pentru efectuarea unor funcţii matematice specifice. Pe măsură ce instrucţiunile ESC sunt decodate, ele trec în secvenţiatorul de instrucţiuni care programează subunităţile cipului pentru realizarea funcţiei cerute.

Când execuţia instrucţiunii ESC începe, NEU activează semnul său BUSY (ocupat) care poate fi supravegheat de procesorul principal, sau această stare este raportată în cuvântul de control al coprocesorului. Semnalul BUSY, combinat cu

instrucţiunea WAIT a procesorului principal permite resincronizarea procesorului şi a coprocesorului său atunci când este necesar.

În lucrarea de faţă va fi oferită o soluţie proprie privind proiectarea unităţii de execuţie numerică (NEU) care să permită concepţia unui procesor aritmetic performant. Acesta, fie va primi o destinaţie generală şi va fi ataşat unui procesor principal (caz în care vor fi indicate posibilităţile de extensie a capabilităţilor NEU), fie va reprezenta o unitate de calcul specializată, de sine stătătoare, dedicată unui anumit context matematic, respectiv dedicată anumitor tipuri de aplicaţii.

Ca punct de pornire, în capitolul 1 al tezei vor fi analizate principalele metode de operare numerică şi în primul rând operarea cu numere întregi, întrucât oricare altă metodă se reduce în cele din urmă la operare (adunare) de numere întregi. Vor fi astfel prezentate şi analizate principalele tipuri de sumatoare rapide cunoscute în literatura de specialitate, realizate într-o astfel de manieră încât informaţia numerică ce se prelucrează să străbată un număr cât mai mic de niveluri logice. În următoarele două paragrafe vor fi prezentaţi algoritmii prin care se implementează diversele operaţii aritmetice, atunci când informaţia binară este reprezentată în virgulă flotantă (FP) în format simplă sau dublă precizie, respectiv în virgulă fixă, în format logaritmic (LNS) simplă precizie. Vor fi evidenţiate avantajele şi dezavantajele fiecărui mod de reprezentare şi de operare a informaţiei numerice, ca şi premize teoretice pentru concepţia unui nou tip de coprocesor.

Pe baza concluziilor desprinse în primul capitol, în capitolul 2, cel mai important al tezei, va fi descrisă structura unui coprocesor hibrid, care va îngloba o subunitate logaritmică de calcul, ce va avea intrări şi ieşiri în formatul virgulă flotantă, simplă precizie. Capitolul 2 este aproape în întregime original, motiv pentru care el este redactat în mare parte cu caractere *italice* pentru a evidenţia contribuţiile autorului. De asemenea, figurile şi relaţiile originale, precum şi citarea lucrărilor proprii vor fi notate cu caractere *italice*. Subunitatea logaritmică va include convertoare de format la intrarea şi ieşirea sa, care să permită operarea în interiorul său în format LNS, în scopul realizării rapide a operaţiilor de înmulţire şi împărţire. De asemenea va fi prezentat un studiu al erorilor rezultate după efectuarea conversiilor de format şi va fi prezentată o metodă de reducere a acestora, în condiţiile în care şi aria ocupată pe cip va fi mai mică.

În capitolul 3 va fi concepută o subunitate în virgulă flotantă, pentru implementarea operaţiilor de adunare şi scădere. Structura proiectată va fi optimizată din punct de vedere al suprafeţei ocupate pe cip şi al realizării sincronizării cu subunitatea logaritmică. Astfel, ansamblul format din cele două subunităţi va funcţiona în parametri de mare performanţă, mai ales în cazul implementării prin microprogramare a unor algoritmi DSP, sau transformări de coordonate pentru aplicaţii grafice.

În capitolul 4 va fi demonstrată viteza mare de calcul a tuturor structurilor proiectate, pe baza simulărilor efectuate cu ajutorul programului "MSim" în cazurile

cele mai defavorabile din punct de vedere al propagării transportului prin toate aceste circuite.

În continuare, în ultimul capitol al tezei, vor fi prezentate concluziile desprinse în urma rezultatelor obţinute, vor fi evidenţiate toate contribuţiile teoretice şi aplicative aduse de autor pe cuprinsul lucrării şi vor fi enunţate perspectivele de dezvoltare ulterioară a temei abordate.

În anexele de la lucrare va fi prezentată pe părţi componente schema completă a unităţii de execuţie numerică a coprocesorului hibrid, programele Matlab pentru generarea valorilor ce trebuie memorate în ROM, în condiţiile minimizării erorii de conversie de format, programul C pentru depistarea cazurilor cele mai defavorabile pentru sumatorul cu transport anticipat pe 8 biţi, precum şi un exemplu de rulare al unui algoritm DSP simplu, prin care se efectuează o buclă Livermoore cu 20 de termeni (suma a 20 de produse de câte doi factori).

Cap.1 Metode de operare numerică

1.1 Circuite rapide de operare cu numere întregi

Indiferent de tipurile de date cu care lucrează coprocesorul, în final orice operaţie, indiferent cât de complexă ar fi ea, se reduce la o operaţie cu numere întregi. Aceasta înseamnă că pentru a îmbunătăţi viteza de calcul într-un procesor aritmetic trebuie acţionat inclusiv în această direcţie, respectiv trebuie alese acele structuri de operare cu numere întregi care oferă cei mai mici timpi de producere a rezultatului, iar pentru structuri care furnizează timpi comparabili, se alege acel circuit care ocupă cea mai mică suprafaţă pe cipul de siliciu. În funcţie de lungimea cuvintelor de date cu care se operează, se poate depista prin analiză şi simulare care este tipul optim de structură ce trebuie adoptat, pentru o tehnologie dată.

Cea mai frecvent utilizată operaţie elementară este operaţia de adunare. Toate celelalte operaţii elementare se reduc sau cel puţin fac apel şi la operaţia de adunare. Astfel, operaţia de scădere se efectuează prin adunarea la descăzut a complementului de doi al scăzătorului (A-B=A+/B+1, în care /B este B negat). Operaţia de înmulţire se realizează prin însumarea produselor parţiale iar operaţia de împărţire include scăderi repetate întreţesute cu comparări.

De asemenea, calculul funcţiilor trigonometrice şi transcedentale se face prin intermediul unor serii rapide sau algoritmi, [22], [23], [48], [88], al căror calcul implică tot utilizarea celor patru operaţii aritmetice elementare. Aceasta înseamnă că practic sumatoarele binare de numere întregi sunt printre cele mai solicitate componente hard într-un coprocesor, motiv pentru care ele vor fi prezentate şi analizate în acest paragraf.

Sumatorul simplu

Cel mai simplu sumator binar a două numere de n biţi, dar şi cel mai dezavantajos din punct de vedere al timpului de propagare, este cel realizat prin cascadarea a n sumatoare complete pe un bit. Un astfel de sumator complet (sumator complet 3:2) generează pe un bit suma s_i a celor 2 biţi de rang i din cei doi operanzi ţinând cont de transportul c_i furnizat de sumatorul anterior, iar pe un alt bit furnizează transportul c_{i+1} către sumatorul următor. Pot fi scrise următoarele relaţii:

$$s_i = a_i \oplus b_i \oplus c_i \qquad c_i = a_i \cdot b_i + b_i c_i + c_i \cdot a_i \tag{1.1}$$

Aceste relaţii conduc la implementarea prezentată în figura 1.1. Întrucât $a \oplus b = a \cdot \overline{b} + \overline{a} \cdot b$ se obţine pentru s_i următoarea expresie:

$$s_i = a_i \cdot \overline{b_i} \cdot \overline{c_i} + \overline{a_i} \cdot b_i \cdot \overline{c_i} + a_i \cdot b_i \cdot c_i + \overline{a_i} \cdot \overline{b_i} \cdot c_i \tag{1.2}$$

Deoarece transportul se propagă de la o celulă la alta traversând câte două niveluri logice (conform relaţiei 1.2), rezultă că e nevoie de 2n niveluri logice pentru obţinerea rezultatului. Pentru creşterea vitezei operaţiei de adunare au fost adoptate

diverse tehnici care să permită reducerea numărului de niveluri logice parcurse de informaţia binară ce se prelucrează.

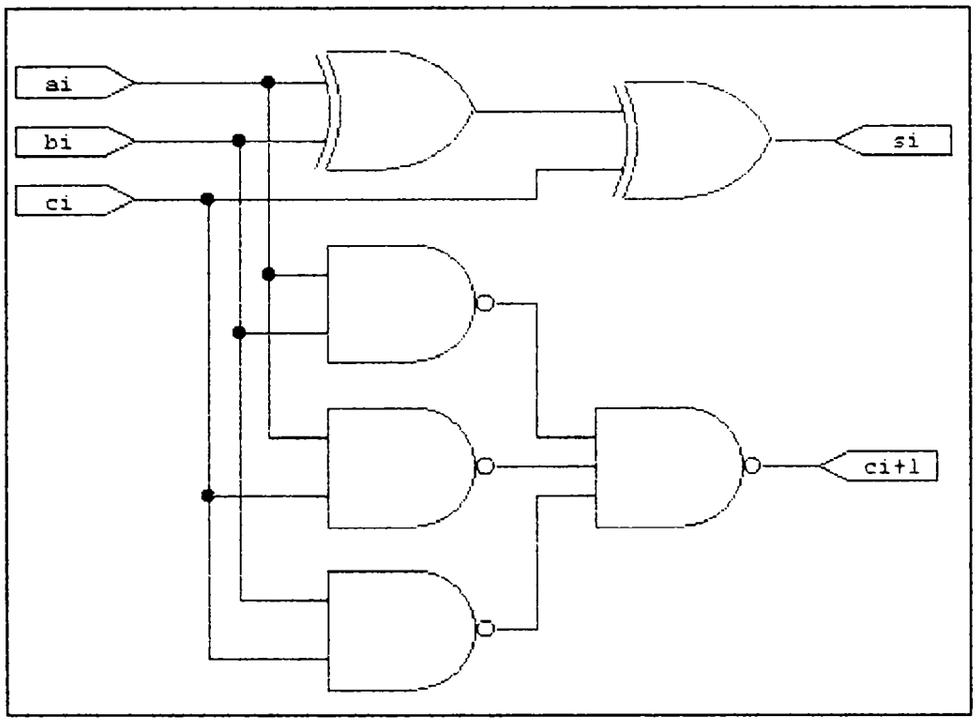


Fig. 1.1 Schema unui sumator complet pe un bit (sumator complet 3:2)

Reducerea numărului de niveluri se poate face utilizând următoarele relaţii:

$c_{i+1} = g_i + p_i c_i$ în care: $g_i = a_i b_i$ şi $p_i = a_i + b_i$ (1.3)

Făcând înlocuirile din aproape în aproape obţinem:

$$c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \ldots\ldots + p_i p_{i-1} \ldots p_1 g_0 + p_i p_{i-1} \ldots\ldots p_1 p_0 c_0 \quad (1.4)$$

În concluzie e nevoie de un nivel logic pentru a forma termenii p şi g, două niveluri logice pentru a forma transportul cu relaţia 1.4 şi două niveluri logice pentru sumă. În total rezultă cinci niveluri logice. Dezavantajele structurii care ar implementa relaţiile 1.3 şi 1.4 constau în faptul că sunt necesare porţi cu foarte multe intrări (de exemplu, semnalul p_{n-1} trebuie să apară la intrarea a n porţi ŞI), rezultând astfel o structură neregulată şi foarte multe fire de legătură. Această idee totuşi a fost folosită pentru a se construi un sumator care va avea $\log_2 n$ niveluri logice, deci mult mai puţine decât un sumator cu propagarea transportului şi în plus o structură mai simplă şi mai regulată.

Sumatorul cu transport anticipat

Sumatorul cu transport anticipat este cunoscut în literatura de specialitate, [73], [80], sub denumirea de "carry look-ahead adder" (CLA). Ideea folosită constă în construirea semnalelor auxiliare g şi p în paşi. Există deja $c_1 = g_0 + c_0 p_0$, care înseamnă că un transport este generat din poziţia 0 fie datorită faptului că poziţia în sine (prin a_0 şi b_0 egali cu 1) generează transport, fie datorită faptului că un transport anterior se poate propaga peste această poziţie (dacă a_0 sau b_0 este egal cu 1).

Similar $c_2 = G_{01} + P_{01}c_0$ unde $G_{01} = g_1 + p_1 g_0$ şi $P_{01} = p_1 p_0$. G_{01} arată că blocul format din primii doi biţi generează un transport, iar P_{01} arată că prin prin acest bloc se poate propaga un transport anterior.

În general, considerând i<j<k putem scrie următoarele relaţii:

$$C_{k+1} = G_{ik} + P_{ik}c_i \quad (1.5)$$

$$G_{ik} = G_{j+1,k} + P_{j+1,k}G_{ij} \quad (1.6)$$

$$P_{ik} = P_{ij}\,P_{J+1,k} \quad (1.7)$$

în care $P_{ii} = p_i$ şi $G_{ii} = g_i$. Relaţia (1.5) se interpretează astfel: un transport este generat la ieşirea blocului format din biţii de la i la k inclusiv, dacă partea superioară a acestui bloc (incluzând biţii de rang cuprins între j+1 şi k) generează un transport, sau dacă există un transport generat de partea inferioară (incluzând biţii de rang cuprins între i şi j) care se poate propaga prin partea superioară.

Pe baza acestor consideraţii se prezintă o variantă practică de sumator CLA [80]. Sumatorul constă din două părţi. Prima parte (figura 1.2) calculează diverse valori P şi G din p_i şi g_i utilizând ecuaţiile (1.6) şi (1.7).

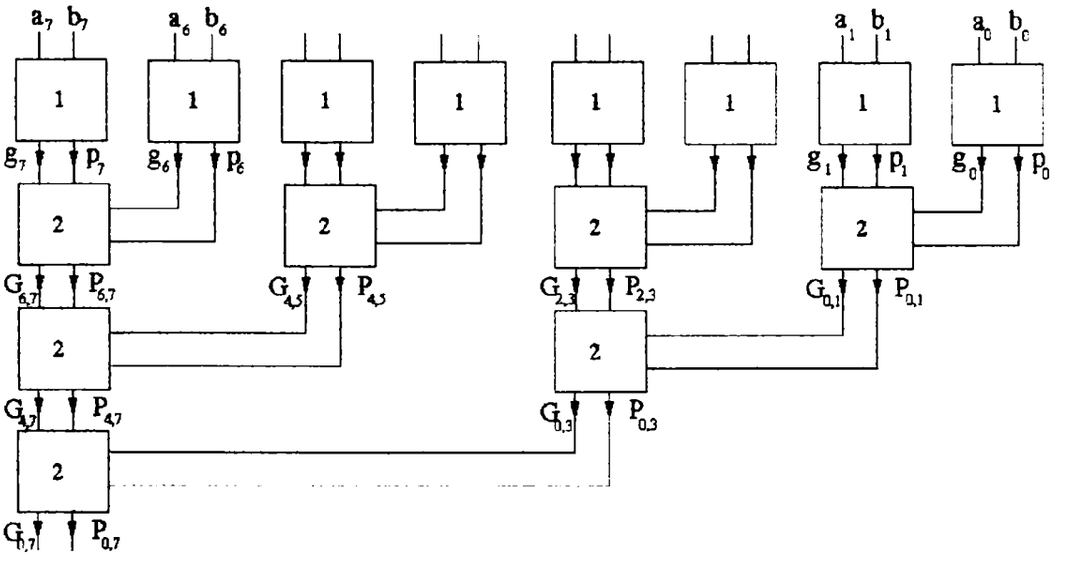


Fig.1.2 Schema bloc a circuitului care generează coeficienţii G şi P

Celulele de tip 1 vor calcula valorile g_i şi p_i din a_i şi b_i , iar celulele de tip 2 interconectate într-o structură de tip arbore, grupate două cîte două, vor calcula perechile corespunzătoare (P, G).

A doua parte foloseşte valorile P şi G pentru a calcula toate transporturile cu ajutorul ecuaţiei (1.5). Schema care se obţine este prezentată în figura 1.3.

Cele două tipuri de celule pot fi alăturate într-o structură unică aşa cum se vede în figura 1.4. Astfel, într-un sumator cu structură arborescentă de tip CLA, numerele binare ce trebuie adunate intră în partea de sus a arborelui, curg în jos pentru a se combina cu c_0 şi apoi informaţia curge în sus pentru a se calcula biţii sumei. Numărul de celule utilizate este 2n (inclusiv cea necesară pentru generarea lui c_8), deci de două ori mai multe decât la un sumator clasic, însă din punct de

vedere al spaţiului ocupat pe cip suprafaţa ocupată este egală cu $n \times \log_2 n$ celule dintr-un sumator clasic.

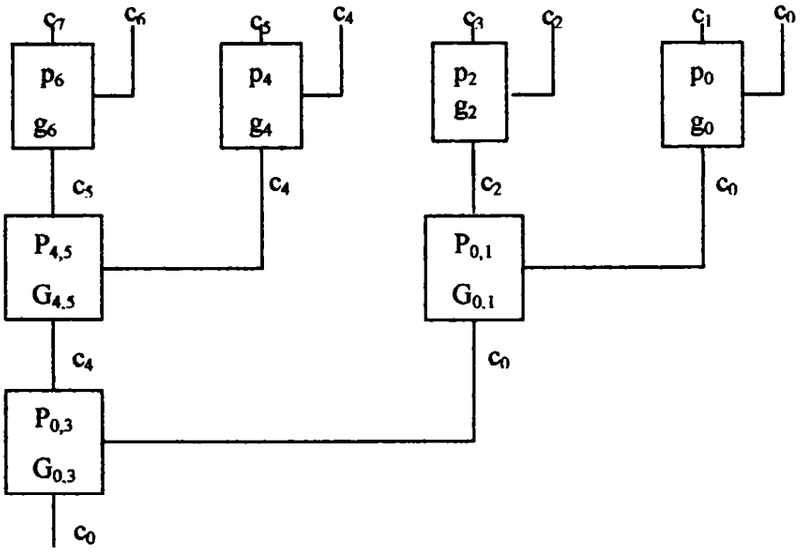


Fig. 1.3 Schema bloc a circuitului care generează transporturile c_i

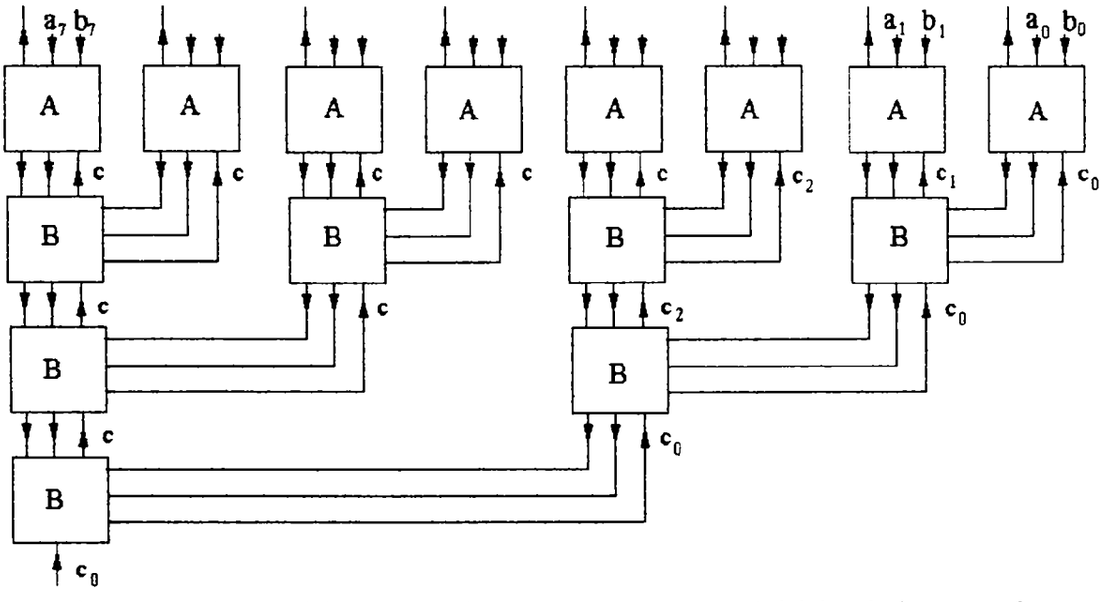


Fig.1.4 Schema bloc a unui sumator CLA pe 8 biţi, obţinută prin suprapunerea structurilor prezentate în figurile 1.2 şi 1.3

Structura blocurilor de tip A şi B este prezentată în figurile 1.5.a şi 1.5.b.

Concluzia care se desprinde totuşi aici este că printr-o mică investiţie în mărimea structurii se obţine o creştere substanţială a vitezei, comparativ cu un sumator cu propagarea din aproape în aproape a transportului. Acest lucru este cu atât mai pregnant cu cât numărul n de biţi ai numerelor ce trebuie adunate este mai mare.

Există mai multe tipuri de sumatoare CLA care se adoptă în funcţie de tehnologia utilizată. De exemplu dacă fiecare nod al arborelui ar avea trei intrări în loc de două, atunci înălţimea arborelui ar descreşte de la $\log_2 n$ la $\log_3 n$, dar şi

celulele ar fi mai complexe. Pentru tehnologii în care propagarea semnalelor este rapidă, o soluţie hibridă între un sumator cu propagarea transportului şi un arbore CLA, ar fi un bun compromis. În acest caz blocurile de tip A nu ar grupa perechi de biţi ci perechi de blocuri de biţi şi vor fi constituite din sumatoare simple cu număr redus de biţi, blocurile fiind organizate într-un arbore CLA, rezultând pe ansamblu o structură cu anticiparea transportului pe două niveluri ierarhice [13], [78].

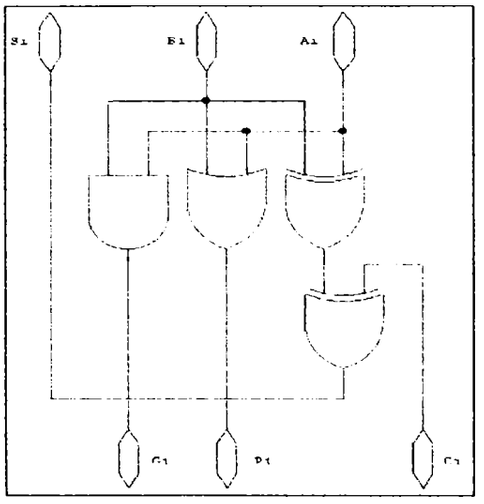


Fig. 1.5.a
Schema blocului de tip A

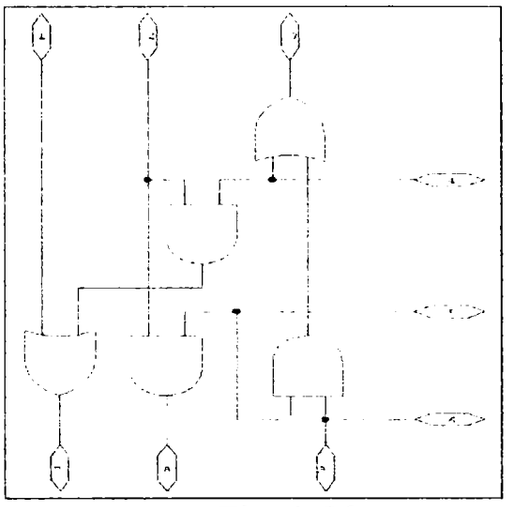


Fig. 1.5.b
Schema blocului de tip B

Sumatorul cu omiterea transportului

Un astfel de sumator se situează din punct de vedere al costului şi al performanţelor între un sumator cu propagarea transportului şi un sumator CLA. Un exemplu de sumator cu omiterea transportului este prezentat în figura 1.6. El este format din blocuri de sumatoare complete pe un bit cascadate, notate B_i în figura 1.6. Numărul sumatoarelor complete din fiecare bloc se stabileşte astfel încât să se obţină o viteză maximă de producere a rezultatului, pentru o lungime dată a operanzilor. Grupând de exemplu câte patru biţii cuvintelor ce trebuie adunate şi scriind pentru prima grupare ecuaţiile coeficienţilor G şi P se obţine:

$$P_{03} = p_0p_1p_2p_3; \qquad G_{03} = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 \qquad (1.8)$$

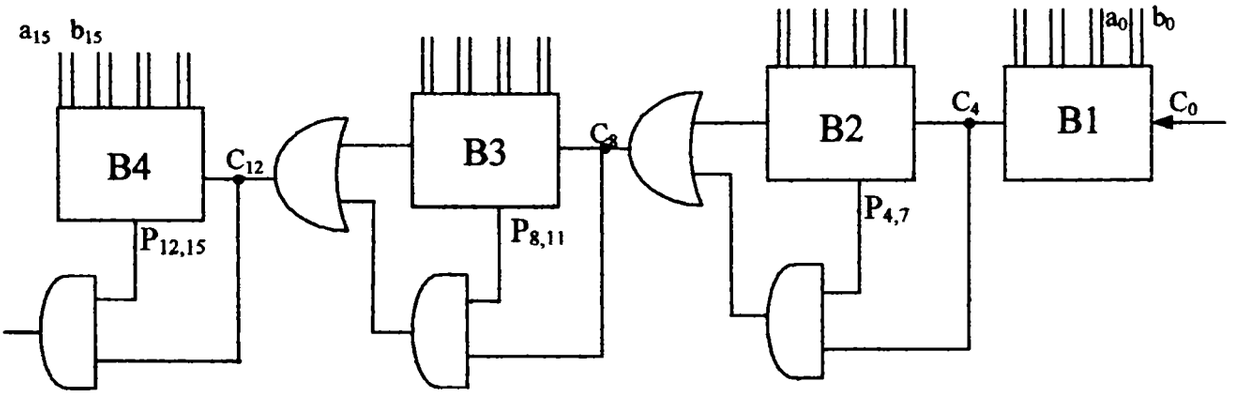


Fig. 1.6 Schema bloc a unui sumator cu omiterea transportului

Deoarece se observă că e mai simplu de calculat coeficienţii P decât G, acest tip de sumator calculează doar coeficienţii P. Propagarea transportului începe simultan în fiecare bloc. Dacă fiecare bloc va genera un transport atunci transportul generat de bloc va apare chiar dacă transportul de intrare al blocului nu este încă corect. Dacă la startul fiecărei operaţii de adunare, transportul la intrarea fiecărui bloc este zero, atunci nici un transport fals nu va fi generat, iar transportul generat poate fi apreciat ca fiind coeficientul G al acestuia.

După ce cel mai puţin semnificativ bloc îşi generează transportul, acesta va apare atât la intrarea blocului următor cât şi la intrarea porţii ŞI împreună cu semnalul P al acelui bloc. Dacă atât semnalul P al acestui bloc cât şi transportul de la blocul anterior sunt ambele 1 atunci imediat va fi generat transportul de intrare pentru următorul bloc. Astfel, a fost omis transportul pe care l-ar fi produs blocul actual în absenţa unui transport la intrarea sa.

Dimensiunea şi numărul optim de blocuri este în funcţie de lungimea cuvintelor de date care se adună, aşa cum se arată în [47]. De asemenea fiecare bloc al sumatorului poate fi organizat sub forma unui sumator cu omiterea transportului, rezultând pe ansamblu un sumator cu omiterea transportului pe 2 niveluri ierarhice, ce prezintă o viteză de calcul substanţial mărită, [28], [46], comparabilă cu a unui sumator cu anticiparea transportului.

Sumatorul cu selecţia transportului

Sumatorul cu selecţia transportului funcţionează după următorul principiu: două sumatoare simple lucrează în paralel în fiecare bloc al sumatorului (excepţie făcând cel mai puţin semnificativ bloc), un sumator considerând transportul de intrare ca fiind "0", alta considerându-l 1. Când transportul de intrare este în final cunoscut, se selectează suma corectă.

În figura 1.7 este construit un sumator pe 19 biţi format din patru blocuri. Dacă un bloc necesită k unităţi de timp pentru a calcula k biţi şi o unitate de timp pentru a furniza "carry in" al blocului următor din două semnale "carry out" disponibile şi un "carry in" al blocului anterior, rezultă că pentru a optimiza operaţia de adunare e necesar ca fiecare bloc să fie mai larg cu 1 bit decât anteriorul.

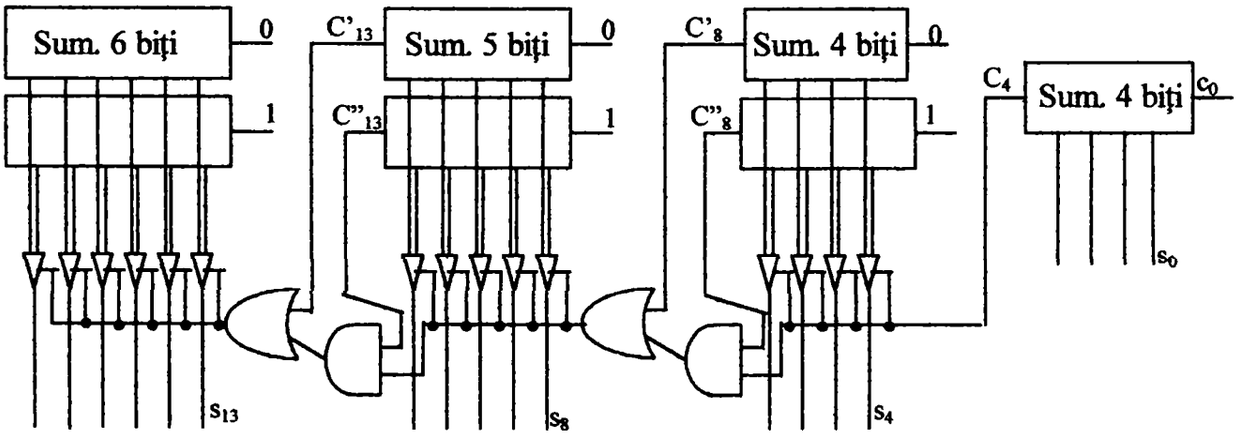


Fig. 1.7 Schema bloc a unui sumator cu selecţia transportului

Şi acest tip de sumator poate fi organizat pe două niveluri ierarhice în scopul creşterii vitezei de calcul. Aceasta înseamnă că în locul fiecărui bloc de sumatoare simple cascadate poate fi introdus un sumator cu selecţia transportului care va conţine subblocuri cu sumatoare simple cascadate.

În continuare va fi făcută o analiză a timpului de producere a rezultatului şi a ariei ocupate pe cip, pentru tipurile de sumatoare prezentate anterior. Se consideră unitatea de timp de propagare, T, ca fiind timpul necesar pentru producerea transportului în cazul unui sumator complet pe 1 bit. Aşa cum se observă în figura 1.1, c_{i+1} se obţine după parcurgerea a două niveluri logice, deci după un timp T; pentru obţinerea bitului sumă s_i este necesar un timp 2T. De asemenea se consideră unitatea de suprafaţă, S, ca fiind aria ocupată de un sumator complet pe 1 bit.

Astfel, pentru sumatorul simplu care adună cuvinte de n biţi, timpul de calcul al rezultatului este $t=(n-1)\times T+2T=(n+1)\times T\approx n\times T$, dacă n este suficient de mare. Suprafaţa ocupată va fi $n\times S$.

Pentru sumatorul cu transport anticipat, ţinând cont că transportul se propagă de sus în jos prin structura din figura 1.2 şi apoi de jos în sus prin structura din figura 1.3 şi că parcurgerea într-un sens sau altul a unui bloc de tip B (fig. 1.5.b) se face într-un timp T, iar a blocurilor de tip A (fig.1.5.a) în 0,5T (în jos), respectiv T (în sus), rezultă că timpul total de calcul este:

$$t_{CLA}=0,5\times T+(\log_2 n-1)\times T+(\log_2 n)\times T+T=(2\log_2 n)\times T+0,5\times T\approx(2\log_2 n)\times T \quad (1.9)$$

Am constatat astfel că formula prezentată în [80] ce furnizează t_{CLA} este greşită, ea furnizând o valoare ce reprezintă doar jumătate din valoarea corectă. Aria ocupată este $n\times\log_2 n\times S$ deoarece arborele CLA conţine $\log_2 n$ etaje. Deoarece însă numărul de blocuri de tip B se înjumătăţeşte spre vârful de jos al arborelui, o proiectare de layout optimă poate conduce la obţinerea unei arii mai mici, fără a afecta caracterul modular al sumatorului.

În cazul sumatorului cu omiterea transportului, timpul total de calcul este dat de timpul de propagare al blocului cu cea mai mare dimensiune [47]. Acest tip de sumator este însă din ce în ce mai puţin utilizat deoarece el este eficient doar dacă se realizează în tehnologii în care semnalele pot fi şterse la începutul fiecărei operaţii [73] şi care să permită integrarea de porţi logice cu multe intrări [46].

Pentru sumatorul cu selecţia transportului, numărul optim de blocuri componente, în cazul unei structuri regulate în care toate blocurile au aceeaşi dimensiune, este $n^{1/2}$ [73]. Dimensiunea unui astfel de bloc va rezulta $n^{1/2}\times S$, iar timpul de propagare al transportului prin bloc va fi deci $n^{1/2}\times T$. *Considerând timpul necesar selecţiei rezultatului corect al unui bloc, de valoare cel puţin T, rezultă că timpul total de calcul pentru acest sumator este cel puţin $t_{SEL}=2n^{1/2}\times T$. Acest rezultat corectează valoarea greşită furnizată de [80]. Suprafaţa ocupată de acest sumator este dată de relaţia (1.10):*

$$S_{SEL}=(\frac{\sqrt{n}-1}{\sqrt{n}}+1)n\times S+S_{mux}\approx 2n\times S, \qquad (1.10)$$

în care S_{mux} este aria ocupată de circuitele multiplexoare care selectează rezultatul corect la nivelul fiecărui bloc.

Utilizarea unuia sau altuia dintre tipurile de sumatoare prezentate (ce reprezintă tipuri de bază, fiind printre cele mai des utilizate), depinde de locul şi scopul în care sunt utilizate. Dacă ele apar în proiectare pe o cale critică de propagare (caz în care viteza lor influenţează direct frecvenţa de clock a procesorului) atunci se folosesc cele mai rapide, chiar dacă aria pe care o vor ocupa pe cip este mare. Dacă nu este aşa, atunci se alege tipul de sumator care ocupă aria cea mai mică, dar care este suficient de rapid pentru a nu deturna calea critică de propagare în structura proiectată.

Principalele caracteristici ale sumatoarelor prezentate, referitoare la timpul necesar pentru producerea rezultatului şi la aria ocupată pe cip, sunt prezentate în tabelul 1.1.

Tab. 1.1

Unitate de timp de propag., T şi de arie ocup., S	timp	supraf.
Sumator simplu cu n sum. complete cascadate	$n\times T$	$n\times S$
Sumator cu anticiparea transportului	$2\log_2 n\times T$	$n\times\log_2 n\times S$
Sumator cu omiterea transportului	$2n^{1/2}\times T$	$n\times S$
Sumator cu selecţia transportului	$2n^{1/2}\times T$	$2n\times S$

Cu aceste tipuri de sumatoare se pot realiza diverse combinaţii care permit obţinerea unor sumatoare mixte, mai rapide decât cele simple, organizate pe două niveluri ierarhice. Nivelul inferior se referă la structura blocurilor componente iar nivelul superior se referă la structura de ansamblu a sumatorului, respectiv la modul în care sunt interconectate blocurile componente.

Aşa cum se va vedea pe parcursul lucrării, anumite combinaţii vor conduce fie la obţinerea unei viteze superioare oricăruia din tipurile de bază menţionate, fie la obţinerea unui anumit optim, pe o cale necritică de propagare. Structura optimă pentru un anumit sumator (ce operează cu date de o lungime dată) se obţine pentru acea configuraţie care să poată fi integrată pe cip într-o arie minimă şi care să nu furnizeze rezultatul într-un timp mai mare decât cel impus. Motivaţia obţinerii acestei structuri optime constă în aceea că nu are rost să se adopte un sumator foarte rapid, adică un circuit care ocupă o arie relativ mare pe cip, dacă el nu poate contribui la creşterea frecvenţei de clock a sistemului de calcul.

În cazul în care unitatea de calcul este organizată pe niveluri pipeline, pe nivelul sau nivelurile critice de propagare (ce prezintă cel mai mare timp de producere a rezultatului şi care impun practic frecvenţa de clock a întregului sistem) vor fi concepute sau adoptate cele mai rapide structuri numerice, economia de arie pe cip ne mai fiind în acest caz prioritară.

1.2 Metode de calcul utilizate în aritmetica binară în virgulă flotantă

1.2.1 Formate de date în virgulă flotantă

Într-un calculator numărul cifrelor disponibile pentru a reprezenta un număr real este limitat. Dacă avem de-a face cu un coprocesor, numărul de biţi cu care sunt reprezentate numerele reale este însă suficient de mare pentru a avea o bună aproximaţie a acestora.

Atunci când sunt cerute simultan atât precizie cât şi o scară largă de reprezentare a numerelor reale se apelează de obicei la modul de reprezentare în virgulă flotantă (FP). Formatele în virgulă flotantă conţin trei câmpuri: de semn, de exponent şi de mantisă, aşa cum se vede în figura 1.8. Numerele cu care se lucrează sunt, implicit, normalizate. Un număr binar este normalizat atunci când are un singur 1 la stânga virgulei. Excepţia este numărul 0 care este tratat ca un caz special. Numărul 1 din stânga virgulei apărând întotdeauna, nu mai e necesar a fi memorat ceea ce permite creşterea cu o unitate a preciziei mantisei. El este însă implicit presupus, fiind restituit de coprocesor la începerea calculelor.

Semn	Exponent	Mantisă

Fig. 1.8 Formatul de reprezentare în virgulă flotantă

Exponentul este memorat sub formă cumulată. Valoarea de cumul (deplasamentul) a fiecărui format este o constantă şi a fost astfel aleasă încât însumată cu exponentul să rezulte întotdeauna un exponent pozitiv. Aceasta simplifică foarte mult comparările. În final exponentul real se obţine scăzând din exponentul numărului valoarea de cumul.

Tipurile de formate în virgulă flotantă sunt :

-real scurt (real în simplă precizie).

Acesta utilizează 32 biţi din care unul pentru semn, 8 pentru exponent, 23 pentru mantisă. Mantisa e normalizată, iar bitul de parte întreagă este implicit 1, rezultând o precizie de 24 cifre binare. Valoarea de cumul este +127. Acest format permite memorarea unor numere cuprinse în intervalul $\pm 10^{\pm 38}$.

-real lung (real în dublă precizie).

Utilizează 64 biţi din care unul este pentru semn, 11 pentru exponent, 52 pentru mantisă. Mantisa e normalizată, iar bitul de parte întreaga este 1 implicit, rezultând o precizie de 53 cifre binare. Valoarea de cumul este de +1023.

-real temporar (real de precizie extinsă).

Utilizează 80 biţi din care: unul pentru semn, 15 pentru exponent, 64 pentru mantisă. Bitul de parte întreagă al mantisei normalizate este direct memorat, spre deosebire de celelalte două formate reale. Valoarea de cumul: + 16383. Este formatul utilizat intern de FPU pentru toate calculele. Eroarea de rotunjire şi impreciziile de calcul se situează dincolo de limitele realului scurt şi lung.

1.2.2 Realizarea celor patru operaţii aritmetice elementare în virgulă flotantă

Relaţiile 1.11÷1.14 rezumă operaţiile de bază ce se efectuează în aritmetica binară în virgulă flotantă. Pentru adunare şi scădere trebuie asigurat ca cei doi operanzi să aibă acelaşi exponent. Aceasta implică deplasarea virgulei la unul dintre operanzi pentru a se realiza alinierea mantiselor. Înmulţirea şi împărţirea sunt mult mai directe. Având astfel două numere în virgulă flotantă, X şi Y:

$$X = M_x \times B^{Ex} \qquad Y = M_y \times B^{Ey}$$

atunci cele patru operaţii aritmetice elementare se efectuează în modul următor:

$$X+Y = (M_x \times B^{Ex-Ey} + M_y) \times B^{Ey} \tag{1.11}$$

$$X-Y = (Mx \times B^{Ex-Ey} - M_y) \times B^{Ey}, \qquad E_x \le E_y \tag{1.12}$$

$$X \times Y = (M_x \times M_y) \times B^{Ex+Ey} \tag{1.13}$$

$$X / Y = (M_x / M_y) \times B^{Ex-Ey} \tag{1.14}$$

în care B reprezintă baza de numeraţie în care se lucrează. În marea majoritate a aplicaţiilor B=2.

Problemele care pot apărea în cazul acestor operaţii sunt:

-supradepăşire exponent - un exponent pozitiv ar rezulta mai mare decât cea mai mare valoare posibilă a exponentului. În anumite sisteme acest caz poate duce la desemnarea lui ca $+\infty$ sau $-\infty$.

-subdepăşire de exponent - un exponent negativ ar rezulta în modul mai mare decât cea mai mare valoare posibilă a exponentului. Aceasta înseamnă că numărul este prea mic pentru a putea fi reprezentat şi poate fi interpretat ca fiind zero.

-subdepăşire mantisă - în procesul de aliniere a mantiselor, biţii se deplasează prea mult spre capătul din dreapta al mantisei, caz în care anumite metode de rotunjire sunt cerute.

-supradepăşire mantisă - adunarea a două numere de acelaşi semn poate duce la obţinerea unui transport peste cel mai semnificativ bit al mantisei. Acest lucru poate fi evitat prin realiniere.

Algoritmul de adunare şi scădere în virgulă flotantă în manieră sincronă.

În aritmetica în virgulă flotantă adunarea şi scăderea sunt mai complexe, din punct de vedere al etapelor ce trebuiesc parcurse, decât înmulţirea şi împărţirea. Aceasta se datorează necesităţii de realizare a alinierii mantiselor pentru ca cei doi operanzi să prezinte acelaşi exponent. Algoritmul de adunare sau scădere cuprinde patru faze principale:

-Verificarea de zero
-Alinierea mantiselor
-Adunarea sau scăderea mantiselor
-Normalizarea rezultatului

Schema logică tipică ce realizează aceste funcţii este dată în figura 1.9.

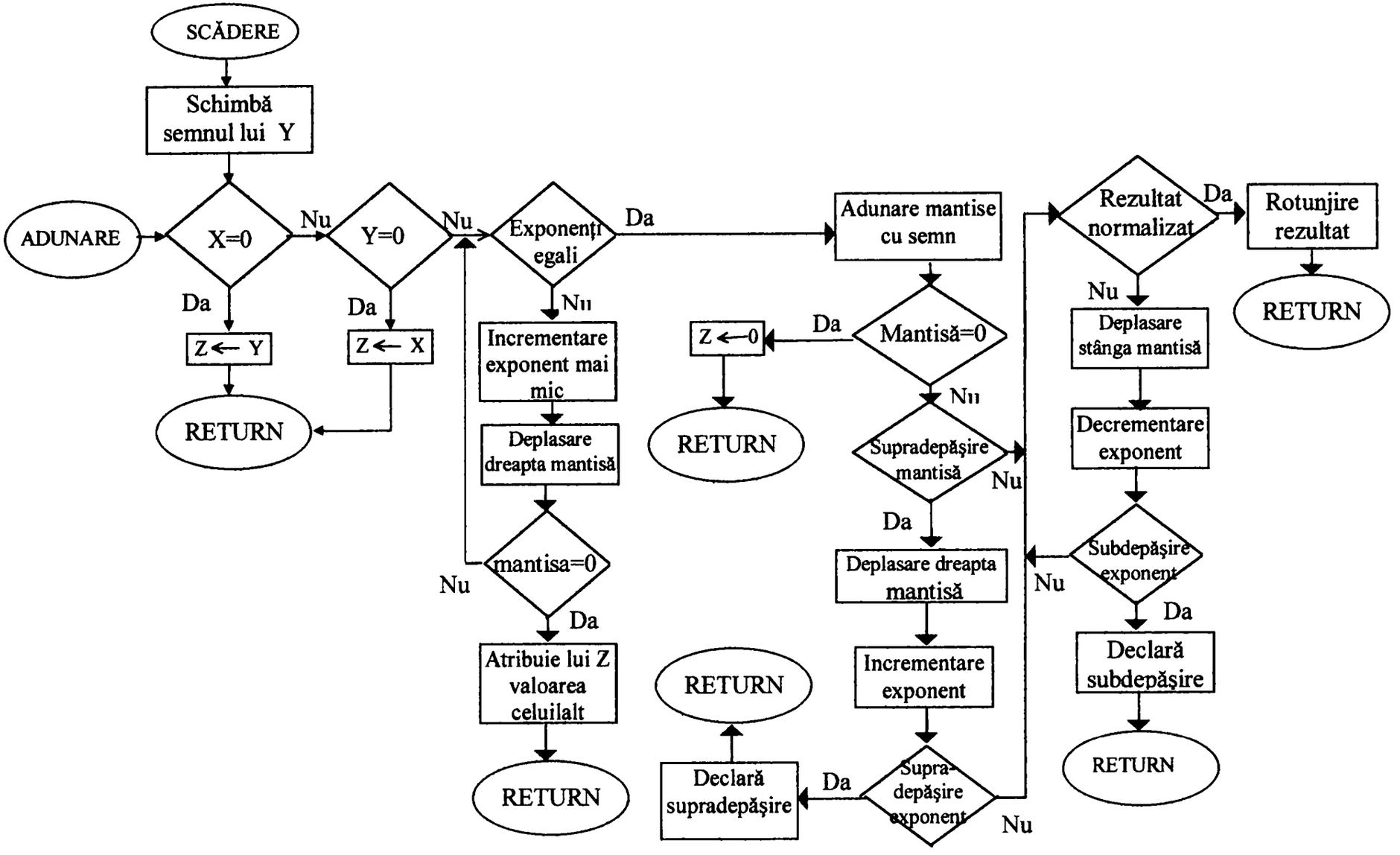


Fig. 1.9 Algoritmul de adunare/scădere în virgulă flotantă

Pentru realizarea operaţiilor de adunare sau scădere, cei doi operanzi trebuie transferaţi în regiştrii care vor fi utilizaţi de ALU. Dacă formatul în virgulă flotantă include implicit cel mai semnificativ bit, acesta trebuie să fie făcut explicit pentru realizarea operaţiei. În mod obişnuit exponenţii şi mantisele vor fi memoraţi în regiştri separaţi, pentru a fi reuniţi atunci când s-a produs rezultatul. Deoarece adunarea şi scăderea sunt identice, cu excepţia faptului că trebuie schimbat bitul de semn al scăzătorului în cazul scăderii, algoritmul este în rest acelaşi. Mai departe, dacă un operand este găsit zero, atunci celălalt operand este desemnat ca şi rezultat. Următoarea fază este de a manipula astfel numerele încât exponenţii să fie egali.

Alinierea mantiselor se realizează fie prin deplasarea celui mai mic număr spre dreapta, fie prin deplasarea celui mai mare număr spre stânga. Întrucât oricare din cele două variante ar putea duce la o pierdere de biţi, cel mai mic număr se va deplasa. Orice bit care ar putea fi pierdut este astfel de relativ mică importanţă. Alinierea este realizată prin deplasări repetate spre dreapta cu un bit şi incrementarea exponentului până cei doi exponenţi vor fi egali. (Dacă baza în care se lucrează este 16, deplasării cu un bit îi corespunde o deplasare cu câte 4 biţi).

Dacă acest proces duce la obţinerea unei mantise nule, celălalt număr este desemnat ca şi rezultat. Astfel, dacă două numere prezintă exponenţi care diferă semnificativ, atunci cel mai mic număr este pierdut.

În continuare cele două mantise sunt adunate împreună, luând în considerare şi semnul lor. Din moment ce semnul poate diferi, rezultatul poate fi zero. De asemenea există posibilitatea ca mantisa rezultatului să producă o supradepăşire cu un bit. Dacă e aşa, mantisa se va deplasa spre dreapta cu un bit, iar exponentul se va incrementa o dată. Dacă exponentul produce supradepăşire, operaţia se opreşte.

Următoarea fază este normalizarea rezultatului. Normalizarea constă în deplasarea spre stânga a mantisei rezultatului până când cel mai semnificativ bit (sau 4 biţi pentru exponent în baza 16) este nenul. Fiecare deplasare produce o decrementare a exponentului care ar putea duce la o subdepăşire de exponent. In final rezultatul va fi rotunjit şi reportat.

Algoritmul de înmulţire şi împărţire în virgulă flotantă, în manieră sincronă.

Înmulţirea în virgulă flotantă, ca de altfel şi împărţirea, sunt procese mult mai simple, din punct de vedere al numărului de etape ce trebuiesc parcurse, decât adunarea şi scăderea. Operaţia de înmulţire este ilustrată în schema logică din figura 1.10.

La început, dacă fiecare operand este zero, rezultatul este declarat nul. Următorul pas este adunarea exponenţilor. Dacă exponenţii sunt memoraţi în formă cumulată (cu deplasament), exponentul sumă va conţine dublul deplasamentului, astfel că o valoare de deplasament trebuie scăzută din suma exponenţilor. Rezultatul ar putea produce fie supradepăşire exponent, fie subdepăşire exponent şi acest lucru trebuie semnalizat la sfârşitul algoritmului. Dacă exponentul produsului este în domeniul bun, următorul pas constă în înmulţirea mantiselor, ţinând cont şi de semnul lor.

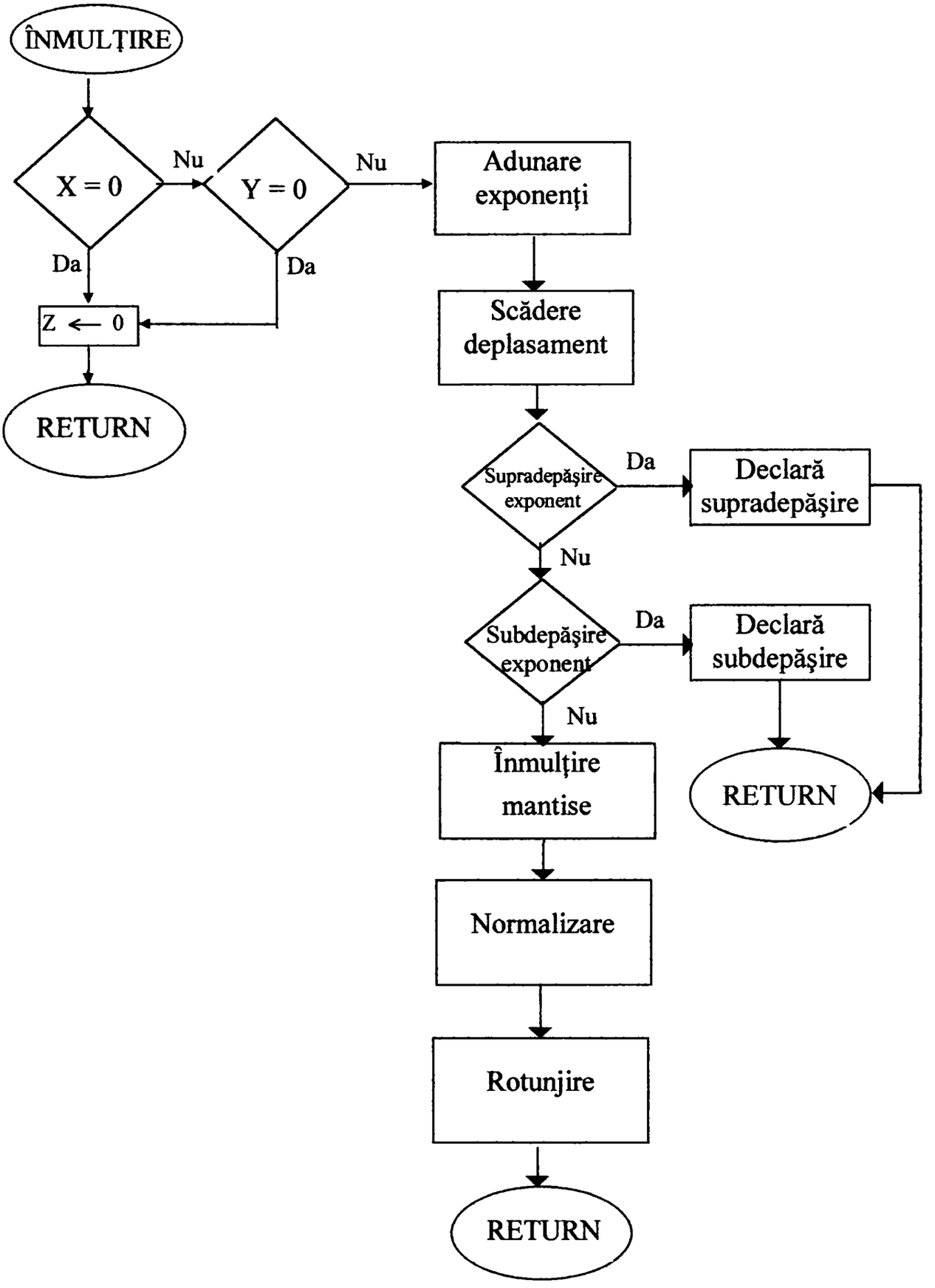


Fig. 1.10 Algoritmul de înmulțire în virgulă flotantă

Înmulţirea mantiselor este realizată în acelaşi mod ca şi în cazul numerelor întregi. În acest caz avem de-a face cu o reprezentare semn - mărime, astfel că semnul rezultatului este stabilit prin intermediul unei porţi SAU-exclusiv. Produsul obţinut va avea o lungime egală cu suma lungimilor deînmulţitului şi înmulţitorului. Biţii rezultaţi peste lungimea maximă vor fi pierduţi în timpul rotunjirii. Însumarea exponenţilor va ţine cont de semnul fiecărui exponent.

După ce produsul este calculat, rezultatul este apoi normalizat şi rotunjit aşa cum s-a făcut şi în cazul adunării, iar exponentul este ajustat, cu observaţia că normalizarea ar putea duce la o subdepăşire de exponent.

Schema logică a algoritmului de împărţire în virgulă flotantă este dată în figura 1.11. Şi aici primul pas constă în testarea operanzilor dacă au sau nu valoarea zero vreunul dintre ei. Dacă împărţitorul este zero este semnalizată eroare sau rezultatul este desemnat a fi infinit. Dacă deîmpărţitul este zero, atunci rezultatul este desemnat a fi zero.

În continuare exponentul împărţitorului este scăzut din exponentul deîmpărţitului . Aceasta va elimina valoarea de deplasament din exponent, astfel că deplasamentul trebuie adăugat din nou. De asemenea sunt făcute teste pentru subdepăşirea sau supradepăşirea exponentului.

Următorul pas este împărţirea mantiselor. Aceasta este apoi urmată de normalizare şi rotunjire, respectiv ajustarea exponentului. Şi în acest caz semnul rezultatului este stabilit prin intermediul unei porţi SAU-exclusiv.

Consideraţii privind mecanismul de funcţionare asincron/sincron al unei unităţi de calcul în virgulă flotantă.

Funcţionarea în manieră pur sincronă a unei unităţi de calcul în virgulă flotantă, conform cu algoritmii prezentaţi anterior, conduce la efectuarea celor patru operaţii aritmetice elementare într-un timp foarte mare, întrucât ar fi necesare zeci sau chiar sute de perioade de tact pentru obţinerea rezultatului. Realizarea pas cu pas a alinierii mantiselor sau a normalizării, în care deplasarea unui operand cu un bit este urmată de un test logic, conduce la o astfel de situaţie. Frecvenţa de clock a sistemului nu ar putea fi nici ea prea mare, fiind limitată de acea operaţie simplă în care informaţia străbate cele mai multe niveluri logice (ca de exemplu operaţia de adunare propriu-zisă a doi operanzi întregi).

Pe de altă parte, funcţionarea în manieră pur asincronă a unei unităţi de calcul în virgulă flotantă ar conduce la realizarea oricărei operaţii aritmetice elementare într-o singură perioadă de tact. Timpul total de calcul ar rezulta mai scurt decât în cazul precedent, dar totuşi mult mai lung decât în cazul unor operaţii uzuale de transfer de date între regiştrii coprocesorului, ceea ce pe ansamblu conduce la o anumită ineficienţă precum şi la viteză de operare scăzută.

Din acest motiv, o unitate de calcul în virgulă flotantă se organizează pe mai multe niveluri pipeline, separate de latch-uri. În interiorul unui nivel din structură informaţia se procesează în manieră asincronă, iar propagarea rezultatelor intermediare de la un nivel pipeline la altul se face sincron, în ritmul frecvenţei de clock.

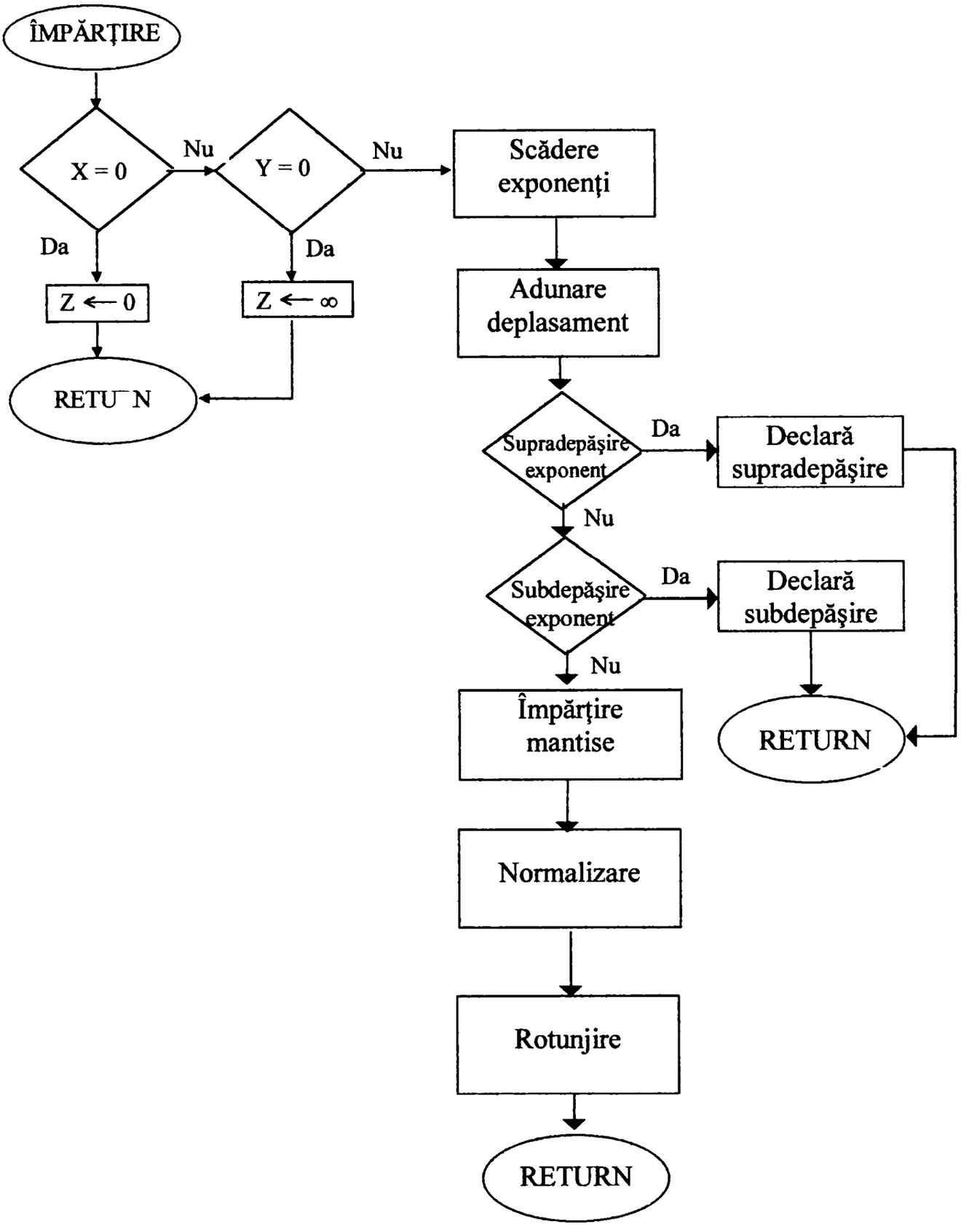


Fig. 1.11 Algoritmul de împărţire în virgulă flotantă

1.2.3 Metode de creştere a vitezei operaţiilor de adunare şi scădere în virgulă flotantă

Întrucât multiplicatoarele binare în virgulă flotantă sunt mari consumatoare de arie pe cip, conform [14], [25], [29], [35], [55], [62], [65], [67], [70], [72], [81], [84], [97], [103], [106], iar circuitele de împărţire sunt lente comparativ cu circuitele care efectuează celelalte operaţii aritmetice elementare (de circa 4÷6 ori mai lente decât multiplicatoarele binare) conform [24], [53], [54], [66], [69], [73], [80], ele vor fi implementate folosind principiile aritmeticii logaritmice, aşa cum se va vedea în capitolul următor. În schimb operarea pe baza circuitelor de adunare/scădere în virgulă flotantă reprezintă modalitatea de găsire a rezultatului operaţiilor amintite cea mai rapidă şi mai economică din punct de vedere al ariei ocupate pe cip. Din acest motiv, în acest paragraf vor fi amintite principalele soluţii descrise în literatura de specialitate, care concură la implementarea unor circuite de adunare/scădere în virgulă flotantă performante.

Aşa cum s-a remarcat în paragraful anterior, structura optimă a unei unităţi de calcul în virgulă flotantă presupune o organizare pe niveluri pipeline a acesteia. O proiectare judicioasă a unităţii de calcul trebuie să conducă la situaţia în care toate nivelurile pipeline să fie parcurse de informaţia binară în intervale de timp aproximativ egale şi acestea să fie cât mai scurte. În plus structura pipeline permite, în cazul operării consecutive a mai multor perechi de operanzi, obţinerea unui nou rezultat la fiecare perioadă de clock a sistemului.

În cazul circuitelor de adunare/scădere în virgulă flotantă, soluţia clasică, încă foarte frecvent adoptată, presupune o organizare pe trei niveluri pipeline a acestora (**soluţia I**), dar tendinţa actuală este de a se reduce numărul de niveluri la două (**soluţia II**) în condiţiile păstrării sau chiar creşterii frecvenţei de clock a sistemului. Preţul plătit constă în aproape dublarea numărului de porţi logice integrate, fapt care însă este suportat tehnologic datorită creşterii densităţii de integrare.

Etapele logice succesive care sunt parcurse la o adunare sau scădere în virgulă flotantă sunt următoarele:

a. Calcularea diferenţei exponenţilor;
b. Stabilirea exponentului mai mic;
c. Deplasarea corespunzătoare a mantisei asociate exponentului mai mic;
d. Operarea (adunarea/scăderea) mantiselor având asociate exponentul mai mare;
e. Determinarea numărului de zerouri după virgulă conţinute în diferenţa mantiselor, respectiv a rangului celui mai semnificativ 1 din partea întreagă a sumei mantiselor;
f. Normalizarea rezultatului;
g. Ajustarea exponentului rezultatului;
h. Rotunjirea rezultatului.

La elaborarea **soluţiei I** s-a avut în vedere în principal reducerea numărului de niveluri logice, prin introducerea unor structuri de calcul paralele şi tehnici de

selecţie a rezultatului corect în interiorul fiecărui nivel pipeline. Au putut fi astfel suprapuse aproape în totalitate, din punct de vedere temporal, anumite etape din lista enunţată care în mod natural se desfăşoară succesiv; imediat după stabilirea definitivă a rezultatului de la finele etapei anterioare s-a procedat la selectarea rezultatului corect.

Astfel, în **nivelul 1** al structurii pipeline, scăderea exponenţilor (etapa a.) se face concomitent cu deplasarea în paralel a celor două mantise (etapa c.), ţinând cont că circuitele "barrel shifter", care deplasează în manieră asincronă mantisele, sunt comandate pe măsură ce se produc biţii diferenţei exponenţilor, dinspre LSB spre MSB, conform [26], [66]. În momentul în care se cunoaşte deja semnul diferenţei exponenţilor (etapa b.), se selectează mantisa deplasată ce a corespuns exponentului mai mic.

De asemenea, în **nivelul 2** al structurii pipeline, operaţia de scădere a mantiselor (etapa d.), prin maniera în care a fost implementată, elimină timpul necesar reconversiei din cod complement de doi în cod semn-mărime, în cazul diferenţei negative, conform [7], [26], *[39]*, *[44]*, [56]. Acest lucru se realizează prin calcularea simultană a două rezultate, din care doar unul va reprezenta valoarea corectă şi va fi selectat în funcţie de semnul diferenţei celor două mantise. De asemenea operaţia de rotunjire (etapa h.) prin adaos, sau către cea mai apropiată valoare reprezentabilă în format, se poate efectua în acest nivel, prin calcularea simultană a încă unui rezultat.

Atunci însă când se efectuează scădere propriu-zisă şi operanzii sunt foarte apropiaţi în valoare absolută, pot rezulta multe zerouri înaintea celui mai semnificativ 1 din rezultat. Pentru a se efectua normalizarea (etapa f.), aceste zerouri trebuie în primă fază contorizate (etapa e.), urmând ca apoi să fie comandat un circuit "barrel shifter" similar cu cel de la alinierea mantiselor.

O economie importantă de timp se face dacă numărul acestor zerouri ("leading zeros") poate fi anticipat încă din etapa de operare a celor două mantise aliniate, astfel ca la finele acestei etape să înceapă efectiv deplasarea în vederea normalizării. O serie de articole şi lucrări ştiinţifice din literatura de specialitate au avut ca temă această problemă [7], [10], [85]. Soluţiile oferite au permis, prin adăugarea unor structuri hard suplimentare în nivelul 2 al structurii pipeline, să se estimeze numărul de zerouri cu o precizie de 1 bit, ceea ce face ca, în urma comandării circuitului "barrel shifter" de normalizare în **nivelul 3**, rezultatul să se găsească în domeniul [0.5, 2). Cu alte cuvinte, se obţine fie rezultatul normalizat, fie având un zero înaintea celui mai semnificativ 1. În urma testării bitului MSB al acestuia se mai comandă un circuit suplimentar de postnormalizare, care fie lasă rezultatul nedeplasat, fie execută o deplasare spre stânga cu un bit. În felul acesta, pe ansamblu, s-a făcut o economie de timp semnificativă. Tot în acest nivel se efectuează în paralel şi operaţia de ajustare a exponentului rezultatului (etapa g.).

Totuşi operaţia de postnormalizare nu este chiar foarte scurtă datorită necesităţii multiplicării capabilităţii de comandă a bitului MSB menţionat, ce trebuie să comande un număr foarte mare de grile MOS, din multiplexoarele 2:1

ale circuitului de postnormalizare. De aceea, au fost concepute şi circuite de corecţie a erorii de predicţie a numărului de zerouri [11], care elimină astfel timpul consumat la postnormalizare. Evident, acest lucru presupune şi o creştere suplimentară semnificativă a numărului de porţi logice utilizate.

La elaborarea **soluţiei II** s-a avut în vedere reducerea numărului de niveluri logice, prin introducerea unor structuri de calcul paralele şi tehnici de selecţie a rezultatului corect nu doar în interiorul fiecărui nivel pipeline ci la scara întregii unităţi de adunare/scădere în virgulă flotantă. Rezultă astfel două căi distincte de procesare a informaţiei, care nu se reunesc decât în urma selecţiei finale, de la ieşirea acestei unităţi de calcul. Acest mod de abordare pleacă de la următoarele premize:

- în cazul operaţiei de adunare, normalizarea ulterioară a rezultatului este foarte scurtă. Într-adevăr, având mantisa asociată exponentului mai mare, normalizată în intervalul [1,2), deoarece bitul de rang zero este 1 implicit, iar cealaltă mantisă, deplasată sau nedeplasată, e situată în intervalul [0,2), rezultă că suma nu se poate găsi decât în intervalul [1,4). Aceasta înseamnă că după testarea bitului de rang 1 din partea întreagă a rezultatului, se decide dacă suma rămâne nedeplasată sau se deplasează la dreapta cu un bit, în vederea normalizării;

- în cazul operaţiei de scădere, dacă operanzii sunt mult diferiţi ca şi valoare, este necesar un timp important pentru deplasarea mantisei mai mici, dar rezultatul operării mantiselor aliniate se va situa în intervalul [0.5, 2) şi normalizarea va fi foarte scurtă. În acest caz fie se lasă rezultatul nedeplasat, fie se deplasează la stânga cu un bit. Dacă în schimb operanzii sunt foarte apropiaţi, alinierea mantiselor este foarte rapidă, în sensul că fie se deplasează matisa mai mică cu un bit la dreapta, fie se lasă mantisele nedeplasate. În acest caz operaţia de normalizare este cea mai lungă, întrucât rezultă multe zerouri înaintea celui mai semnificativ 1 din diferenţa mantiselor. Pe ansamblu, timpul de efectuare a operaraţiei de scădere este aproximativ acelaşi în cele două cazuri.

Concluzia care se desprinde constă în aceea că etapa de aliniere a exponenţilor şi deplasare corespunzătoare a mantisei mai mici şi respectiv etapa de normalizare sunt complementare din punct de vedere al timpului necesar pentru parcurgerea lor, în cazul cel mai defavorabil al fiecăreia dintre ele. Astfel, în ultimele lucrări apărute în literatura de specialitate [7], [77] se definesc şi se implementează două căi distincte ("far path" sau cale îndepărtată şi "near path" sau cale apropiată) de operare pentru cele două cazuri prezentate, ceea ce presupune practic aproape dublarea ariei circuitelor de adunare/scădere în virgulă flotantă. Sunt necesare chiar şi două sumatoare/scăzătoare distincte deoarece în unitatea de calcul în virgulă flotantă nivelul la care intervin aceste circuite pe cele două căi este diferit. Ele pot fi chiar secţionate între două etaje consecutive, în scopul optimizării vitezei, ceea ce presupune şi utilizarea unor latch-uri suplimentare. În aceste condiţii, circuitul de adunare/scădere în virgulă flotantă poate fi organizat pe doar două niveluri pipeline, fără a reduce frecvenţa de clock a sistemului.

1.3 Metode de operare numerică în aritmetică logaritmică

1.3.1 Realizarea operaţiilor de înmulţire şi împărţire a două numere cu ajutorul logaritmilor

Operaţiile de înmulţire şi împărţire, efectuate cu ajutorul logaritmilor, sunt bazate pe următoarele relaţii:

$$A \times B = \text{antilog}\,(\log A + \log B) \quad (1.15)$$
$$A / B = \text{antilog}\,(\log A - \log B) \quad (1.16)$$

în care A şi B reprezintă modulul celor 2 numere care se înmulţesc. Semnul produsului este generat în exteriorul structurii hard care permite implementarea relaţiilor precedente.

Pentru a păstra o gamă dinamică largă de reprezentare a numerelor , cei doi factori A şi B vor fi reprezentaţi în format virgulă flotantă cu mantisa normalizată :

$$A = x \times 2^i \qquad B = y \times 2^j \quad (1.17)$$

unde $x \geq 1$, $y \geq 1$, $x,y < 2$. În mod uzual este utilizată baza 2 pentru reprezentarea numerelor, iar logaritmul va fi de asemenea considerat în bază 2.

Logaritmii produsului şi ai câtului sunt daţi de relaţiile (1.18 şi 1.19):

$$\log (A \times B) = i + j + \log xy \quad (1.18)$$
$$\log (A / B) = i - j + \log x/y \quad (1.19)$$

iar produsul, respectiv câtul sunt obţinute pe baza relaţiilor (1.20) şi (1.21):

$$A \times B = 2^{i+j}\,\text{antilog}\,(\log x + \log y) = 2^{i+j} \times z \quad (1.20)$$
$$A / B = 2^{i-j}\,\text{antilog}\,(\log x - \log y) = 2^{i-j} \times z' \quad (1.21)$$

Astfel generarea produsului (câtului) actual necesită o operaţie de adunare (scădere) realizată atât pentru logaritmii mantiselor cât şi pentru exponenţi, implementată cu sumatoare (scăzătoare) rapide, două operaţii de logaritmare ce se pot realiza simultan şi o operaţie de antilogaritmare.

Logaritmarea şi antilogaritmarea pot fi realizate fie prin utilizarea unor memorii ROM care memorează direct valoarea logaritmilor şi antilogaritmilor binari conform [9] (**metoda 1**), fie prin utilizarea unor structuri logice celulare (SLC) de generare a logaritmilor şi antilogaritmilor binari conform [16], [17] (**metoda 2**), fie utilizând algoritmi rapizi ce necesită memorii ROM de mult mai mică capacitate, care memorează valoarea logaritmului, respectiv antilogaritmului în anumite puncte între care se face interpolare liniară *[40]*, *[41]*, *[42]*, *[44]*, [50], [51], [107] (**metoda 3**).

Principalii indicatori de performanţă pentru un circuit de înmulţire logaritmic sunt următorii:

- eroarea cu care se calculează logaritmul şi antilogaritmul, respectiv produsul final;

- timpul sau viteza de generare a logaritmului şi antilogaritmului, respectiv a produsului final;
- aria ocupată pe cipul de siliciu;
- gradul de modularitate al circuitului, sau uşurinţa implementării în layout.

Aceşti indicatori vor fi analizaţi pe scurt în cazul celor trei metode amintite.

Metoda 1, descrisă în [9], este cea mai simplă şi mai directă. Eroarea care apare la generarea lui z sau z' din relaţiile (1.20) şi (1.21) este aceeaşi deoarece adunarea şi scăderea produc aceeaşi eroare absolută şi se determină, doar pentru cazul înmulţirii, pe baza următoarelor considerente :

- prima sursă de eroare este dată de însăşi reprezentarea numărului x pe n biţi, obţinându-se $x_{(n)}$:

$$e_1 = x_{(n)} - x \tag{1.22}$$

- a doua sursă de eroare apare la generarea pe m biţi a logaritmului binar $\log_{(m)} x$:

$$e_2 = \log_{(m)} x - \log x \tag{1.23}$$

- a treia sursă de eroare apare la generarea antilogaritmului binar pe n biţi (s-a considerat că produsul este reprezentat pe n biţi, ca şi factorii):

$$e_3 = \text{antilog}\,(\log x + e_2) + e_1 - x = \text{antilog}\,(\log_{(m)} x) + e_1 - x \tag{1.24}$$

Deoarece e_2 provine dintr-o reprezentare pe m biţi a unui număr subunitar ($0<\log x< 1$), rezultă că $e_2 \le \frac{1}{2}\frac{1}{2^m}$. Deoarece e_3 provine dintr-o reprezentare pe n biţi a unui număr cuprins între 1 şi 4, (produsul a două numere cuprinse între 1 şi 2), rezultă că $e_3 \le \frac{1}{2}\frac{4}{2^n} = \frac{1}{2}\frac{1}{2^{n-2}}$. În aceste condiţii, eroarea totală care apare la executarea operaţiei de înmulţire după această metodă este:

$$e \le \text{antilog}\,(\log x_{(n)} + \log y_{(n)} + \frac{1}{2}\frac{1}{2^m} + \frac{1}{2}\frac{1}{2^m}) + \frac{1}{2}\frac{1}{2^{n-2}} - x_{(n)}\, y_{(n)} =$$

$$= \text{antilog}\,(\log x_{(n)}\, y_{(n)} + \frac{1}{2^m}) + \frac{1}{2^{n-1}} - x_{(n)}\, y_{(n)} = x_{(n)}\, y_{(n)}[\text{antilog}(\frac{1}{2^m}) - 1] + \frac{1}{2^{n-1}} \tag{1.25}$$

Modelul matematic complet al înmulţirii logaritmice poate fi reprezentat astfel în figura 1.12:

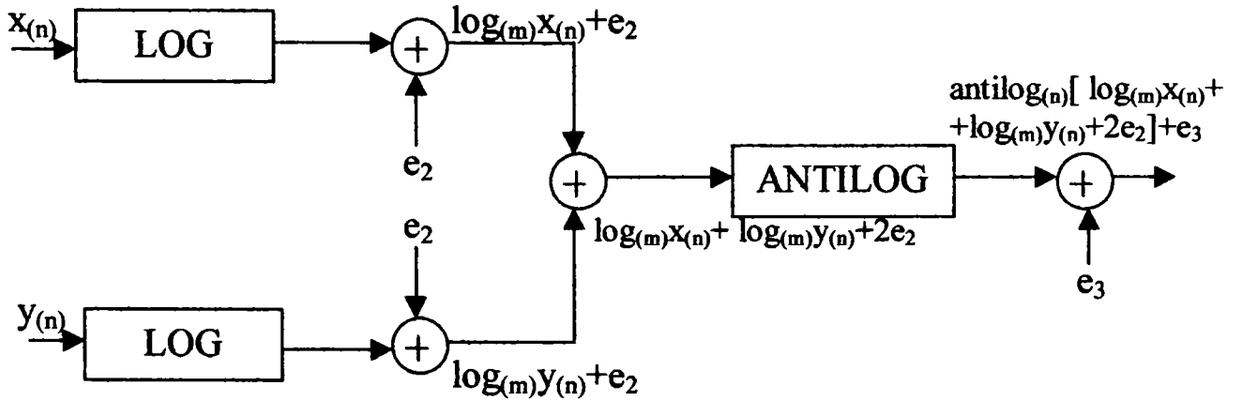


Fig. 1.12 Modelul matematic complet al înmulţirii logaritmice

Analizând relaţia (1.25), se constată că eroarea totală absolută depinde de valoarea operanzilor şi va avea valoarea maximă atunci când operanzii sunt foarte aproape de valoarea 2. Pentru acest caz se dezvoltă în serie Taylor expresia erorii totale, obţinându-se:

$$e \le \frac{1}{2^{n-1}} + \left(\frac{\ln 2}{2^{m-2}} + \frac{1}{2}\frac{(\ln 2)^2}{2^{2m-2}} + \ldots\right) \cong \frac{1}{2^{n-1}} + \frac{\ln 2}{2^{m-2}} \qquad (1.26)$$

Se observă din relaţia (1.26) că atunci când m → ∞, deci când logaritmii operanzilor sunt calculaţi şi reprezentaţi cu precizie infinitesimală, eroarea totală rămâne cea din cazul când se efectua înmulţirea propriu-zisă a doi operanzi reprezentaţi pe n biţi. Pentru a estima numărul de biţi cu care trebuie reprezentat logaritmul pentru ca eroarea totală să rămână acceptabilă, a fost reprezentată grafic în figura 1.13 eroarea totală pentru cazurile când n=22, n=23 şi n=24, în funcţie de m.

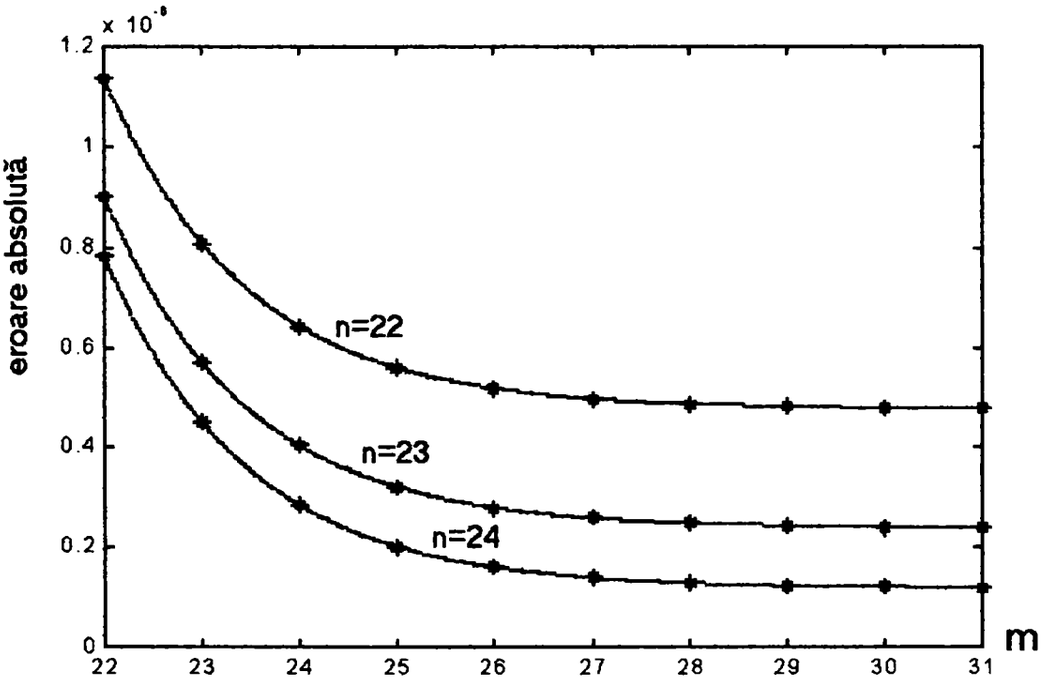


Fig. 1.13 Eroarea totală absolută în cazul înmulţirii logaritmice

Diagramele de genul celor din figura 1.13 sunt foarte utile la stabilirea formatului operanzilor şi la dimensionarea capacităţii memoriilor atunci când este impusă o anumită valoare maximă a erorii de înmulţire. De exemplu, în cazul în care se lucrează în simplă precizie (n=23), pentru ca eroarea totală să fie aproximativ aceeaşi cu cea de la înmulţirea propriu-zisă, numărul de biţi pe care trebuie reprezentat logaritmul este 27 sau chiar 28.

Inconvenientul major pe care îl prezintă **metoda 1** constă în cantitatea uriaşă de memorie care trebuie integrată. Volumul total de memorie necesară, în biţi, este $2 \times m \times 2^n + n \times 2^m$, atunci când sunt utilizate 2 memorii ROM pentru generarea simultană a logaritmilor lui x şi y. . Tabelul 1.2 prezentat mai jos, împreună cu figura 1.13 permit determinarea dimensiunii optime a memoriei ROM, exprimată în Mbit, din diversele combinaţii ale lungimii cuvântului produsului şi

logaritmului. Aşa cum se observă din acest tabel, este necesară o structură multicip pentru a se putea integra cantitatea de memorie necesară, ceea ce conduce şi la pierderea, într-o anumită măsură, a avantajului dat de viteza superioară de generare a logaritmului şi antilogaritmului prin citirea directă din ROM.

Tab. 1.2

	m→	22	23	24	25	26	27	28
n↓	22	277 Mb	377 Mb	570 Mb	948 Mb	1694 Mb	3179 Mb	6140 Mb
	23	466 Mb	579 Mb	789 Mb	1191 Mb	1980 Mb	3540 Mb	6644 Mb
	24	839 Mb	973 Mb	1208 Mb	1644 Mb	2483 Mb	4127 Mb	7382 Mb

Metoda 2, prezentată în [16] şi [17], este bazată pe descompunerea numărului ce se logaritmează în factori de forma $(1+k_i\ 2^{-i})$, respectiv pe descompunerea numărului ce se antilogaritmează în termeni de forma $k_i\ L_i$. Coeficienţii k_i pot fi 0 sau 1 şi se numesc coeficienţi de creştere, ei arătând dacă un factor respectiv termen este prezent sau nu în descompunere. Valorile L_i reprezintă logaritmii binari, ai numerelor de forma $1+2^{-i}$. Valoarea maximă a lui i determină precizia cu care se calculează logaritmul sau antilogaritmul.

Structura hard care implementează această metodă prezintă următoarele caracteristici:

- operează cu numere normalizate, fiind compatibilă cu sistemele de calcul în virgulă flotantă;
- schema sa are o structură celulară şi deci poate fi uşor reprodusă şi extinsă;
- durata maximă de generare a logaritmului sau antilogaritmului este aproximativ $500\times t_p$ (t_p- timpul de propagare printr-o poartă logică simplă) în simplă precizie;
- eroarea apărută la logaritmare sau antilogaritmare se poate face oricât de mică prin simpla creştere a numărului de celule.

Avantajul adus prin folosirea acestei metode constă în aceea că aria circuitelor creşte doar cu pătratul numărului de biţi ai operanzilor şi nu exponenţial, ca în cazul metodei precedente. Un dezavantaj important este însă dat de timpul prea mare de propagare a transportului prin structură, care nu poate fi micşorat semnificativ folosind sumatoare rapide, datorită algoritmului iterativ folosit, pe bază de comparări repetate.

Metoda 3 de calcul al logaritmului şi antilogaritmului binar are la bază un algoritm ce va fi prezentat pe larg în capitolul 2 al tezei şi care prin contribuţiile aduse de autor, permite o viteză de operare în formatul simplă precizie de circa 10 ori mai mare decât în cazul metodei precedente. Aria ocupată pe cip este redusă, iar eroarea furnizată este comparabilă cu a sistemelor în virgulă flotantă.

1.3.2 Realizarea unor operaţii matematice complexe cu ajutorul logaritmilor binari.

Reprezentarea numerelor binare în format logaritmic (LNS - Logarithm Number System) este o reprezentare în virgulă fixă având n biţi, din care i biţi penru partea întreagă şi f biţi pentru partea fracţionară.

Proprietăţile logaritmilor binari permit calculul foarte simplu pentru o serie de funcţii matematice care în virgulă flotantă necesită un timp mult mai îndelungat întrucât se apelează fie la algoritmi specifici complecşi, fie la dezvoltări în serie care pentru precizia dorită ajung să conţină foarte mulţi termeni. În continuare se va arăta pe scurt cum anumite funcţii matematice pot fi calculate foarte simplu folosind aritmetica logaritmică.

-calculul rădăcinii pătrate. Dacă x este un număr pozitiv atunci:

$$\sqrt{x} = \text{antilog}(\log\sqrt{x}) = \text{antilog}(\tfrac{1}{2}\log x) \qquad (1.27)$$

Se ştie că împărţirea la 2, în binar, echivalează cu o deplasare la dreapta cu un bit. În felul acesta se pierde cel mai puţin semnificativ bit iar în locul celui mai semnificativ bit se introduce un zero. Astfel, calculul rădăcinii pătrate se obţine printr-o logaritmare, o deplasare la dreapta cu un bit şi o antilogaritmare. Similar, calculul rădăcinii de ordin 4 implică o deplasare a logaritmului numărului x cu doi biţi la dreapta. Calculul rădăcinii de ordin 2^y (y fiind un întreg pozitiv) implică o deplasare a logaritmului numărului x cu y biţi la dreapta. În felul acesta se pierd cei mai puţin semnificativi y biţi iar în locul celor mai semnificativi y biţi se introduc zerouri. Întrucât astfel ar avea loc o anumită pierdere de precizie, reprezentarea LNS se face în interiorul procesorului logaritmic pe un număr ceva mai mare de biţi decât la interfaţa cu procesorul principal.

-calculul pătratului.

$$x^2 = \text{antilog}(\log x^2) = \text{antilog}(2\log x) \qquad (1.28)$$

Cu alte cuvinte calculul pătratului se face printr-o logaritmare, o deplasare la stânga cu un bit şi o antilogaritmare. Similar pentru calculul unei puteri, putere de 2 (2^y), se va face deplasarea spre stânga cu y biţi.

- calculul oricărei puteri cu exponent real (x, R pozitive).

$$x^{\pm R} = \text{antilog}\,(\log x^{\pm R}) = \text{antilog}\,\{\pm\text{antilog}[\log(\log x) + \log r]\} \qquad (1.29)$$

Aşadar calculul acestei funcţii complexe se face în doar 5 paşi: o dublă logaritmare pentru x şi log x simultan cu log R, o adunare, urmată apoi de două antilogaritmări consecutive.

- calculul unui logaritm în orice bază.

$$\log_B x = \frac{\log x}{\log B} = \text{antilog}\,[\log(\log x) - \log(\log B)] \qquad (1.30)$$

Acest calcul se face aşadar prin două logaritmări consecutive simultane, pentru x şi log x, respectiv pentru B şi log B, o scădere şi o antilogaritmare, adică în doar 4 paşi.

Se observă că toate aceste funcţii matematice fac în cele din urmă apel la operaţii de logaritmare şi antilogaritmare pentru care s-au conceput, aşa cum se va vedea mai târziu, algoritmi competitivi şi la adunare de numere întregi, pentru care au fost prezentate deja sumatoare rapide performante. Operaţia de scădere se reduce tot la adunare folosind reprezentarea în complement de doi a scăzătorului.

1.3.3 Realizarea operaţiilor de adunare şi scădere în aritmetica logaritmică

Dificultatea principală în realizarea unei unităţi aritmetice logaritmice complete constă în implementarea operaţiilor de adunare şi scădere în special. Implementările anterioare anului 1990, [52], [89], au fost limitate la un număr redus de biţi de precizie (sub dimensiunea formatului simplă precizie) datorită faptului că aria circuitelor utilizate în aritmetica logaritmică creşte exponenţial cu lungimea cuvintelor de date vehiculate, atunci când nu se folosesc algoritmi de interpolare. Ulterior, progresul tehnologic (creşterea densităţii de integrare) dar şi conceperea unor algoritmi pentru calculul logaritmului şi antilogaritmului cât şi pentru calculul funcţiilor logaritmice specifice necesare realizării operaţiilor de adunare şi scădere au permis implementarea unor unităţi logaritmice complete în simplă precizie [57], [58], [59], [104]. Într-o tehnologie submicronică, aceste unităţi de calcul chiar pot fi integrate într-un singur cip.

Calculul funcţiilor menţionate anterior este făcut utilizând interpolarea liniară între anumite valori memorate ale acestor funcţii, facilitând o reducere drastică (de peste 100 de ori) a necesarului de memorie ROM, în condiţiile în care eroarea de calcul este încă acceptabilă, comparativ cu sistemele de calcul în virgulă flotantă.

În continuare va fi prezentat pe scurt algoritmul prin care se implementează operaţiile de adunare şi scădere în aritmetica logaritmică.

Considerăm un număr x care este reprezentat de perechea (S_x,e_x) în care S_x dă informaţia cu privire la semnul lui x, iar e_x este logaritmul său. Astfel $x=(-1)^{Sx} \times 2^{ex}$ unde e_x este un număr reprezentat în virgulă fixă având N biţi, cu I biţi pentru partea întreagă şi F biţi pentru partea fracţionară. N=F+I. Valoarea lui e_x este dată de expresia (1.31):

$$e_x = \sum_{i=-F}^{I-1} e_{xi} \times 2^i \qquad (1.31)$$

Pentru a obţine reprezentarea lui 0, se utilizează un bit distinct Z_x ca şi indicator de zero. Dacă $Z_x=1$ atunci valoarea numărului este zero. Astfel, numărul x este reprezentat în formatul LNS de tripleta (S_x,Z_x,e_x) şi are valoarea :

$$x = (1-Z_x) \times (-1)^{Sx} \times 2^{ex} \qquad (1.32)$$

Formatul LNS simplă precizie ce include bit de semn şi bit indicator de zero presupune în consecinţă N=30, cu F=22 şi I=8.

Dacă operaţiile de înmulţire sau împărţire se rezumă la adunarea sau scăderea exponenţilor, operaţiile de adunare şi scădere sunt mai complexe. Considerăm A şi B două numere pozitive şi $A \geq B$. Nu se pierde din generalitate ci doar se simplifică notaţiile.

În cazul adunării (C = A + B), logaritmul e_c al sumei C are valoarea:

$$e_c = e_a + f_a(e_a - e_b) = e_a + f_a(r), \text{ în care: } f_a(r) = \log(1+2^{-r}) \qquad (1.33)$$

În cazul scăderii (C = A – B), logaritmul e_c al diferenţei C are valoarea:

$$e_c = e_a + f_s(e_a - e_b) = e_a + f_s(r), \text{ în care: } f_s(r) = \log(1 - 2^{-r}) \qquad (1.34)$$

Funcţia "log" reprezintă logaritmul în baza 2.

Metoda clasică pentru implementarea funcţiilor $f_a(r)$ şi $f_s(r)$ constă în utilizarea unor memorii ROM având valorile posibile ale lui r ca şi intrări şi furnizând la ieşire cuvintele f(r). Într-o implementare directă, pentru formatul simplă precizie descris anterior, ar rezulta însă un număr de cuvinte $2^N=2^{30}$ care reprezintă o cantitate uriaşă de memorie, imposibil de realizat practic. Reducerea semnificativă a necesarului de memorie se face pe baza formei particulare de variaţie a funcţiilor $f_a(r)$ şi $f_s(r)$ reprezentate în figura 1.14.

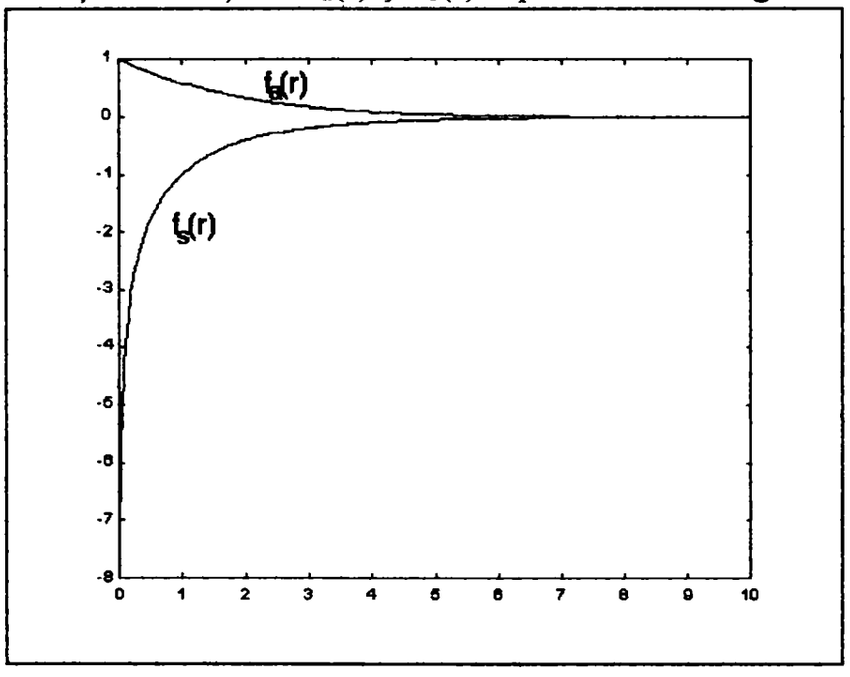


Fig. 1.14 Graficul funcţiilor $f_a(r)$ şi $f_s(r)$

Atunci când argumentul r (care e un număr pozitiv) depăşeşte o anumită limită, valorile funcţiilor $f_a(r)$ şi $f_s(r)$ sunt atât de aproape de zero încât valoarea lor nu mai poate fi reprezentabilă (coboară sub rezoluţia sistemului). Astfel, dacă r = F+2 atunci rezultă:

$$|f_a(r)| < 2^{-F-1},$$
$$|f_s(r)| < 2^{-F-1},$$

în care 2^{-F-1} este precizia cu care poate fi reprezentat un număr în format LNS.

Astfel memoria ROM trebuie implementată doar pentru valori ale lui r care satisfac dubla inegalitate : $0 < r \leq F+2$ ceea ce în cazul concret cu F=22 înseamnă $0 < r \leq 24$. De asemenea se demonstrează că pentru o precizie dată cu F biţi pentru partea fracţionară, în cazul unui r care satisface dubla inegalitate $i \leq r < i+1$, cu $i+1 \leq 24$, ultimii cei mai puţin semnificativi i biţi nu pot afecta rezultatul furnizat de tabela funcţiei f implementată în memoria ROM. Acest lucru permite partiţionarea domeniului util al lui r în subintervale egale cu unitatea şi implementarea pentru fiecare funcţie (f_a şi f_s) a F tabele separate (pentru fiecare subinterval) conţinând între 1 şi 2^F cuvinte, ceea ce în total înseamnă aproximativ 4×2^F cuvinte în loc de 2^N. Aceasta extinde posibilitatea implementării prin memorare directă până la maxim 16÷18 biţi de precizie fracţionară. Utilizând însă aproximarea seriei Taylor pentru funcţiile f_a şi f_s , se poate extinde numărul biţilor de precizie fracţionară la valoarea F=22.

Considerând funcţia $f : R \rightarrow R$, diferenţiabilă, aproximarea Taylor de ordin n a acestei funcţii, f^*, în vecinătatea lui r este :

$$f^*(r+\Delta r) = f(r) + \sum_{i=1}^{n} \frac{(\Delta r)^i}{i!} \times \frac{d^i f(r)}{dr^i} \qquad (1.35)$$

Aproximarea Taylor liniară care va fi folosită utilizează numai primii doi termeni din expresia (1.35). Argumentul r al funcţiei se consideră acum discretizat, fiind reprezentat pe N biţi şi se efectuează o partiţie a cuvântului r în patru câmpuri distincte după cum urmează :

$$r = r_i + r_h + r_l + r_e \qquad (1.36)$$

r_i conţine partea întreagă a lui r deci conţine I biţi: $r_i = r_{I-1}.....r_0$;
r_h conţine "p_l" biţi: $r_h = r_{-1}.....r_{-pl}$;
r_l conţine "(p_e-p_l)" biţi: $r_l = r_{-pl-1}.....r_{-pe}$;
r_e conţine "F- (p_e)" biţi: $r_e = r_{-pe-1}.....r_{-F}$.

În continuare, se defineşte $r_t = r_i + r_h$.

Valorile lui p_l şi p_e nu vor fi constante . Ele vor fi alese în funcţie de subintervalul în care se găseşte r. Aproximarea Taylor va utiliza ca şi argument r pe r_t iar ca şi cantitate Δr pe r_l. Cantitatea r_e va fi ignorată, în conformitate cu observaţia făcută la analiza necesarului de memorie. Astfel se poate scrie :

$$f^*(r) = f(r_t) + r_l \times \frac{df(r_t)}{dr} \qquad (1.37)$$

Memoria ar conţine deci două tabele, una pentru valorile lui f, alta pentru cele ale lui df/dr care ar conţine 2^{pl}, respectiv 2^{pe-pl} cuvinte. Totuşi relaţia (1.37) presupune utilizarea unui multiplicator. Evitarea acestuia se face apelând încă o dată la aritmetica logaritmică. Notând $2^x = \exp(x)$ se poate rescrie relaţia (1.37):

$$f^*(r) = f(r_t) + \mathrm{sgn}\left(\frac{df(r_t)}{dr}\right) \times \exp\left(\log\left(\left| \frac{df(r_t)}{dr} \right| \right) + \log(r_l) \right) \qquad (1.38)$$

Proprietăţile particulare ale derivatelor funcţiilor f_a şi f_s permit simplificarea în continuare a relaţiei (1.38). Astfel:

$$\frac{df_a}{dr} = \frac{2^r}{1+2^r}; \qquad \log\left|\frac{df_a}{dr}\right| = r - \log\left(1+2^r\right) = r - f_a(r) \qquad (1.39)$$

Similar se obţine: $$\log\left|\frac{df_s}{dr}\right| = = r - f_s(r) \qquad (1.40)$$

Astfel, se obţin expresiile (1.41) şi (1.42) care indică modalitatea de implementare a operaţiilor de adunare şi scădere în format LNS:

- în cazul adunării:

$$e_c = e_a + f_a(r_t) + \exp(\log(r_l) + r_t - f_a(r_t)) \qquad (1.41)$$

- în cazul scăderii:

$$e_c = e_a + f_s(r_t) - \exp(\log(r_l) + r_t - f_s(r_t)) \qquad (1.42)$$

Algoritmul prin care se realizează aceste operaţii este prezentat în tabelul 1.3 în care s-au notat şi explicitat rezultatele operaţiilor intermediare:

Tab. 1.3

adunare	scădere
$frt = f_a\_tbl(r_t)$	$frt = f_s\_tbl(r_t)$
$lrl = log\_tbl(r_l)$	$lrl = log\_tbl(r_l)$
$f_d = r_t - frt$	$f_d = r_t - frt$
$l_{cor} = lrl + f_d$	$l_{cor} = lrl + f_d$
$cor = exp\_tbl(l_{cor})$	$cor = exp\_tbl(l_{cor})$
$ff = frt + cor$	$ff = frt - cor$
$e_c = e_a + ff$	$e_c = e_a + ff$

S-a ţinut cont de faptul că pentru $\forall r > 0$, $df_a/dr > 0$ şi $df_s/dr < 0$.

Arhitectura structurii hard ce realizează operaţia de adunare şi scădere în aritmetică logaritmică este dată în figura 1.15.

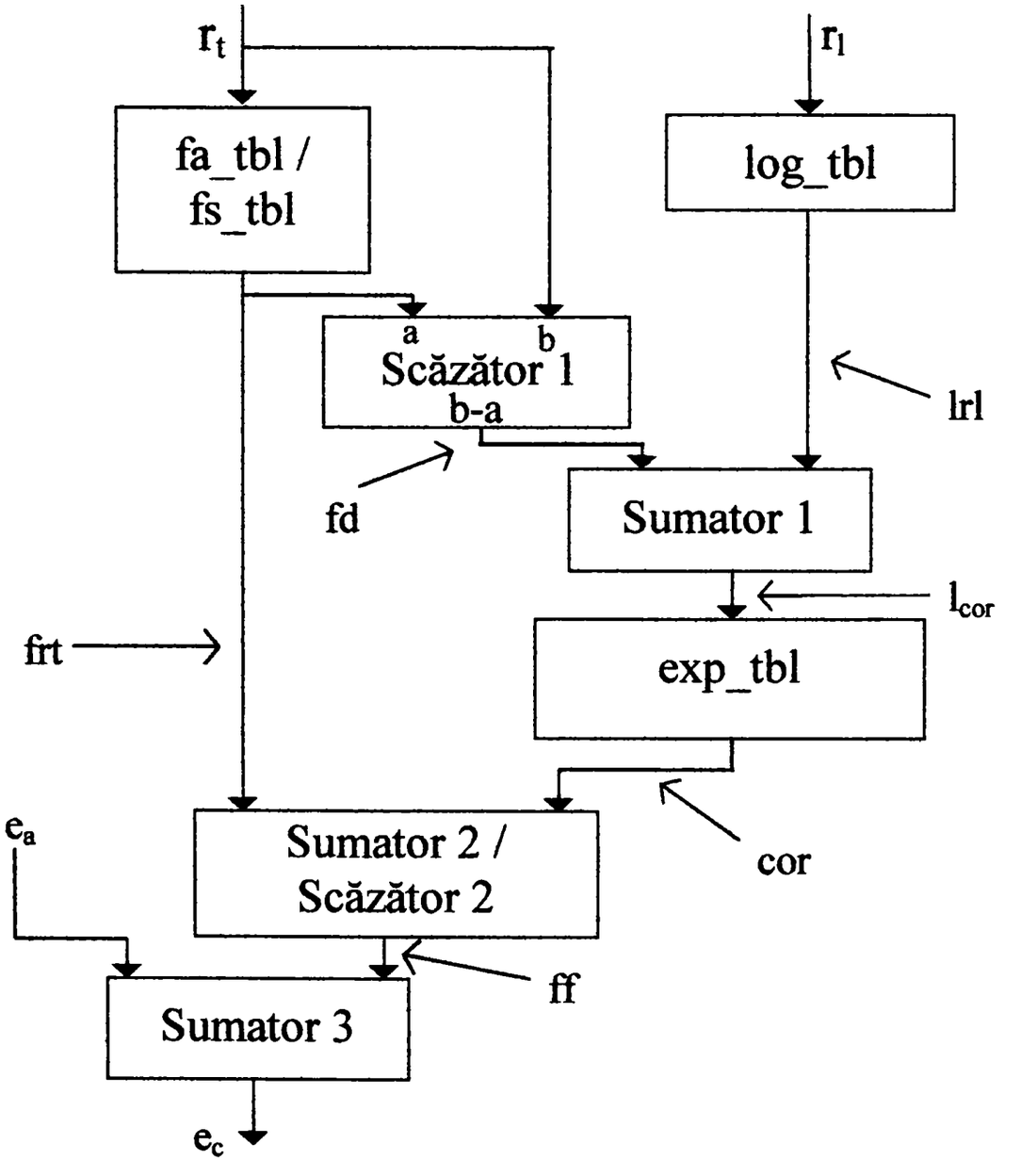


Fig.1.15 Schema bloc a circuitului de adunare/scădere LNS

Capacitatea memoriilor care furnizează pe $\log(r_l)$ şi $\exp(l_{cor})$ este considerabil mai mică decât a acelora care furnizează valorile funcţiilor f_a şi f_s. Maniera de programare a memoriilor fa_tbl şi fs_tbl nu este totuşi perfect asemănătoare datorită faptului că funcţia f_s prezintă o singularitate pentru r=0. Din acest motiv intervalul $0 < r \leq 1$ a fost subîmpărţit şi el în mai multe intervale. Valorile lui p_l şi p_e depind de valoarea lui r relativ la aceste intervale.

Două tipuri de erori pot afecta precizia operaţiilor de adunare şi scădere :
a) eroarea datorată utilizării aproximării Taylor de ordinul 1 şi care este aproximativ egală cu diferenţa dintre aproximarea Taylor de ordinul 2 şi aproximarea f^* utilizată pentru funcţiile f;
b) erorile care apar în fiecare etaj atunci când datele parcurg structura prezentată în figura 1.15 şi care generează o eroare cumulată la ieşirea întregului bloc. Această eroare apare datorită faptului că fiecare tabelă de memorie utilizată are o mărime finită şi că fiecare cuvânt vehiculat pe calea de date este reprezentat pe un număr finit de biţi.

Unele dintre aceste erori contribuie direct la eroarea finală, iar o altă parte, inclusiv din prima categorie contribuie şi indirect întrucât ele vor trece prin tabela care furnizează pe 2^x (exp_tbl).

În lucrarea [58] este prezentată metoda de calcul pentru optimizarea diverselor lungimi de cuvinte şi pentru deducerea valorilor p_e şi p_l, în condiţiile în care eroarea totală este 2^{-F-1}, adică cea corespunzătoare formatului simplă precizie. Metoda a constat în alocarea fiecărui termen din expresia dedusă pentru eroarea totală, a câte unei părţi din disponibilul de "2^{-F-1}", alocare efectuată pe baza următoarelor priorităţi:
-e preferabilă micşorarea lungimii unui cuvânt care reprezintă o adresă de intrare în memorie decât a lungimii unui cuvânt ce reprezintă o ieşire din memorie. Aceasta deoarece un bit adiţional la intrarea de adresă dublează capacitatea memoriei pe când un bit adiţional în cuvântul de ieşire îi măreşte capacitatea doar cu un număr de biţi egal cu numărul de cuvinte conţinute;
-în reducerea dimensiunii memoriilor, prioritate vor avea memoriile care furnizează pe $f_a(r)$ şi $f_s(r)$ deoarece sunt componentele hard cele mai expansive. Aceasta implică selectarea în primul rând a termenilor din expresia erorii totale care îl conţin pe p_l ce determină direct numărul de intrări în tabelele ROM menţionate; următoarele componente hard semnificative sunt tabelele ROM log_tbl şi exp_tbl.

Capacitatea memoriilor care furnizează pe f_a şi f_s a rezultat $(F+10)\times 2^{F/2}-8$ cuvinte de F+7 biţi, iar capacitatea totală de memorie $(F+18)\times 2^{F/2}-8$ cuvinte de diferite lungimi, rezultând pentru F=22, o capacitate de 2,66 Mbit. Această cantitate de memorie poate fi integrată într-o tehnologie submicronică într-un singur cip, împreună cu celelalte componente ale procesorului aritmetic logaritmic.

Viteza de operare a circuitului de adunare/scădere logaritmic este însă de 2÷3 ori mai redusă decât a unui circuit omolog în virgulă flotantă. Aşa cum se

observă din figura 1.15, calea critică de propagare este cea pe care se succed blocurile: fa_tbl/fs_tbl, Scăzător 1, Sumator 1, exp_tbl, Sumator 2/Scăzător 2 şi Sumator 3. În tehnologie CMOS 0,5 microni, doar timpul de acces pentru fiecare din cele două memorii de pe calea critică de propagare este de peste 5ns [107]. Utilizând un mecanism "CSA-carry save adder" (vezi paragraful 2.2.2.1 şi figura 2.5) şi metoda implementată în circuitul din figura 2.24 pentru a comasa blocurile Scăzător 1şi Sumator 1, respectiv Sumator 2/Scăzător 2 şi Sumator 3, pot fi obţinuţi timpi de operare pentru blocurile comasate de ordinul a 4 ns. În aceste condiţii, timpul total de generare a rezultatului este de minim 18 ns (5+4+5+4), în condiţiile în care o adunare/scădere în virgulă flotantă se poate efectua în 6÷8 ns [7].

În plus, atât operanzii de intrare cât şi rezultatul se găsesc în format LNS, deci la timpul de calcul estimat se mai adaugă şi timpul necesar conversiei din şi înapoi în formatul virgulă flotantă, dacă aplicaţia în care este utilizat procesorul aritmetic necesită acest lucru. Chiar şi din punct de vedere al ariei ocupate pe cip, raportul este net favorabil sistemului de operare în virgulă flotantă.

Totuşi, datorită simplităţii prin care se realizează operaţiile de înmulţire şi împărţire şi deci a vitezei obţinute în cazul operării în format LNS, un procesor de calcul pur logaritmic merită a fi propus în anumite aplicaţii particulare cum ar fi:
-implementarea unor algoritmi complecşi, în care rezultatele intermediare pot rămâne în format LNS şi numai un operand trebuie adus la intrarea sau de la ieşirea sistemului;
-aplicaţii în care nu este necesar a se adopta un format standardizat şi deci nu este necesară nici o conversie iniţială de format întrucât mărimea analogică ce este achiziţionată poate fi furnizată de un sistem cu caracteristică de transfer logaritmică;
-în sistemele de calcul complexe (multiprocesor), în care conversia de format poate fi realizată în timpul transferului DMA în bufferele de intrare sau ieşire.

Cap.2 Concepţia subunităţii logaritmice a unui procesor aritmetic hibrid, ce permite efectuarea operaţiilor de înmulţire şi împărţire

În acest capitol, precum şi în cel următor, va fi prezentată şi aplicată o nouă concepţie privind proiectarea unui procesor aritmetic care va îmbina proprietăţile aritmeticii logaritmice şi avantajele operării în virgulă fixă cu calităţile reprezentării numerelor în virgulă flotantă şi de operare cu acestea folosind cei mai performanţi algoritmi de calcul.

2.1 Premize teoretice. Deziderate propuse

Aşa cum s-a precizat în Introducere, tipul unui coprocesor este dat de modul în care sunt reprezentate datele în interiorul său. Atunci când simultan sunt cerute atât o scară largă de reprezentare a numerelor reale cât şi precizie, în mod uzual se adoptă sistemul de reprezentare în virgulă flotantă, motiv pentru care procesoarele aritmetice care operează cu date de acest tip sunt cele mai răspândite.

Creşterea, în plan tehnologic, a densităţii de integrare a permis dezvoltarea, ca şi alternativă şi a procesoarelor logaritmice, ce operează cu date în format LNS. Au apărut astfel o serie de unităţi de calcul logaritmice din ce în ce mai performante, fiind menţionate şi în literatura de specialitate: [2], [4], [12], [18], [21], [22], *[37]*, *[40]*, *[42]*, *[44]*, [45], [50], [51], [57], [58], [59], [89], [90], [104]. Singurul lor dezavantaj este că deocamdată nu permit decât calcule în simplă precizie. Din acest motiv şi reprezentarea în virgulă flotantă nu va face apel în capitolul următor decât la formatul simplă precizie.

Un procesor aritmetic hibrid, în virgulă flotantă şi logaritmic (FP-LNS) este un procesor care vehiculează date în interiorul său, reprezentate în ambele tipuri de formate, pe căi specifice fiecăreia dintre ele. Prof. M. Ciugudean enunţă acest concept în 1976 în teza de doctorat, [16], în care erau oferite soluţiile pentru efectuarea logaritmării şi antilogaritmării şi în care era prezentată arhitectura unui procesor hibrid şi modul de implementare a diverselor operaţii matematice. Ulterior, (1986) a reluat şi a dezvoltat această problematică [15], [18].

Prima implementare pe cip a unui astfel de procesor (1985) e datorată lui F. J. Taylor [90]. Unitatea de execuţie numerică (NEU) a acestui coprocesor realiza operaţiile matematice de bază în format LNS. Legătura dintre NEU şi unitatea de control (CU) a coprocesorului, ce vehicula date în virgulă flotantă era făcută prin intermediul a două convertoare de format: FP-LNS (pentru logaritmare) şi LNS-FP (pentru antilogaritmare). Dezavantajul principal care se făcea resimţit era dat de dimensiunea mică a cuvintelor de date cu care se opera. Întrucât conversiile de format utilizau memorii ROM interne care furnizau valoarea logaritmului sau antilogaritmului prin citire directă, capacitatea acestora nu putea fi foarte mare, ceea ce limita lungimea cuvintelor de date la o valoare inferioară standardului existent în FP. În acelaşi timp, lungimea cuvintelor de date era restricţionată şi de capacitatea memoriilor ROM ce trebuiau integrate pentru implementarea

operaţiilor de adunare şi scădere în format logaritmic. Ulterior, Lewis prin algoritmul propus în 1990 [58], beneficiind şi de evoluţia tehnologică, oferă o soluţie pentru realizarea operaţiilor de adunare şi scădere LNS în simplă precizie, dar nu este făcută nici o specificaţie cu privire la realizarea logaritmării respectiv antilogaritmării.

Ultimele menţiuni în literatura de specialitate cu privire la concepţia unui procesor aritmetic hibrid sunt făcute de Lai în 1991 [50], [51] şi 1993 [107]. Lai a propus şi un algoritm avantajos pentru efectuarea conversiilor de format. *Acest algoritm va fi adoptat şi în lucrarea de faţă cu unele modificări avantajoase, efectuate de autorul tezei, menite să reducă complexitatea logicii de comandă a ALU din subunitatea logaritmică de calcul precum şi să elimine o parte din circuitele de prelucrare a exponenţilor din convertoarele de format, asigurând şi o viteză de calcul ceva mai mare. Algoritmul va fi implementat, tot ca şi contribuţie proprie, printr-o proiectare completă până la nivel de poartă logică.*

În continuare, pe baza analizei efectuate asupra algoritmului cât şi asupra întregii structuri proiectate după acest algoritm, vor fi aduse, atât la nivel de principiu cât şi hard, contribuţii suplimentare importante. Va fi descrisă şi implementată o nouă metodă de structurare a arhitecturii unităţii logaritmice. Implementarea se va realiza utilizând circuite de concepţie proprie, mai performante decât cele descrise în literatura de specialitate. De asemenea programarea memoriilor ROM utilizate la generarea logaritmului şi antilogaritmului va utiliza o metodă suplimentară de corecţie a erorilor cauzate de metoda de interpolare cunoscută în bibliografie.

Toate acestea, aşa cum se va dovedi prin analiză Matlab şi simulare, vor conduce atât la o creştere de precizie cât şi la o creştere substanţială a vitezei de calcul a logaritmului şi respectiv antilogaritmului, în condiţiile în care numărul de porţi logice utilizate, respectiv numărul de tranzistoare integrate pe cip este şi mai redus.

Concepţia procesorului aritmetic hibrid ce va fi prezentat în acest capitol a plecat de la următoarele premize:

-procesoarele logaritmice sunt foarte eficiente în operaţii ca înmulţirea, împărţirea, ridicarea la putere, extragerea de radical, calculul funcţiei logaritm în orice bază sau al funcţiei de ridicare la orice putere reală. În schimb complexitatea operaţiilor de adunare şi scădere realizate în format LNS, aşa cum s-a văzut în paragraful 1.3, conduce la o pierdere inevitabilă de viteză şi la consum excesiv de arie pe cipul de siliciu datorită capacităţii mari de memorie ROM ce trebuie integrată.

-procesoarele în virgulă flotantă implementează algoritmul de adunare/scădere în format FP, prezentat în paragraful 1.2 al tezei, prin intermediul unor structuri hard cu funcţionare asincronă, astfel că, în ciuda complexităţii sale comparativ cu aritmetica în virgulă fixă, rezultatul poate fi furnizat într-un timp relativ scurt. Înmulţirea, de asemenea, se realizează foarte rapid (cu ceva mai mult timp decât adunarea), tot în manieră asincronă, utilizând aşa zisele arii de înmulţire

[80]. Aceasta însă se face cu preţul unui consum mare de arie pe cip [30], [35], [55], [62], [65], [67], [72], [81], [97], [103]. În ce priveşte împărţirea, aceasta este atât consumatoare de suprafaţă cât şi de circa 4-6 ori mai lentă decât înmulţirea [50], [53], [69], [73], întrucât stabilirea biţilor câtului este întreţesută cu operaţii de comparare/scădere. Calculul pentru toate celelalte funcţii matematice, inclusiv rădăcina pătrată, logaritmul şi exponenţiala, se face mult mai lent decât în cazul adunării şi scăderii în virgulă flotantă.

În concluzie, procesorul hibrid descris în acest capitol va realiza adunarea şi scăderea în virgulă flotantă, în simplă precizie, (suficientă încă în multe aplicaţii) şi toate celelalte operaţii prin intermediul unei subunităţi logaritmice de calcul. Aceasta conduce la o arhitectură originală ce presupune că însăşi unitatea de execuţie numerică (NEU) a procesorului aritmetic va avea o structură hibridă, iar convertoarele de format FP-LNS şi LNS-FP se vor situa în interiorul său.

Pentru creşterea vitezei de calcul şi asigurarea posibilităţii implementării cu uşurinţă a unor algoritmi DSP, subunităţile coprocesorului hibrid au fost proiectate în structură pipeline.

Întrucât atât subunitatea în virgulă flotantă cât şi cea logaritmică a NEU efectuează operaţii care în cele din urmă se reduc la adunare de numere întregi, vor fi adoptate sumatoare binare performante, cunoscute în literatură, însă *pe căile critice de propagare vor fi utilizate sumatoare de concepţie proprie, mai rapide.*

În plus, s-a căutat suprapunerea pe cât posibil a unor operaţii care în mod natural se desfăşoară succesiv, respectiv combinarea implementării unor structuri binare paralele cu tehnici de selecţie a rezultatului corect. Astfel, în cazul adunării în virgulă flotantă, scăderea exponenţilor se va face în paralel cu deplasarea mantisei, ţinând cont că circuitul care deplasează mantisa ("barrel shifter") este comandat pe măsură ce se produc biţii diferenţei exponenţilor, dinspre LSB spre MSB, conform [66]. De asemenea, operaţia de scădere a mantiselor, prin maniera în care a fost implementată, elimină timpul necesar reconversiei din cod complement de doi în cod semn-mărime, în cazul diferenţei negative, conform metodei descrise în [26], dar *uzând de o topologie îmbunătăţită a circuitelor utilizate, concepută de autor. De această proprietate va beneficia şi circuitul de adunare/scădere din secţiunea logaritmică, prin intermediul căruia se vor realiza operaţiile de înmulţire/împărţire şi care a fost optimizat în urma modificării efectuate de autor a algoritmului propus de Lai în* [50].

2.2 Subunitatea de calcul logaritmică pentru realizarea rapidă a operaţiilor de înmulţire şi împărţire în simplă precizie

2.2.1 Algoritmul de realizare a conversiilor de format FP-LNS şi LNS-FP

Algoritmul de realizare a conversiilor de format FP-LNS şi LNS-FP a fost descris de către F. Lai în [50] şi [51] şi indică de fapt maniera prin care se poate efectua logaritmarea şi respectiv antilogaritmarea.

În formatul virgulă flotantă (FP) simplă precizie un număr binar A, este reprezentat astfel:

S_A-1bit	E_A-8 biţi	M_A-23 biţi

în care S_A este bitul de semn, E_A este exponentul, iar M_A mantisa normalizată. Deoarece întodeauna există un singur 1 în stânga virgulei, acesta nu se mai memorează dar este implicit restituit în interiorul coprocesorului în structurile hard care operează cu mantise. Valoarea numărului A este dată de următoarea expresie:

$$A=(-1)^{S_A}(1+0,M_A)\,2^{E_A-127} \qquad (2.1)$$

În formatul simplă precizie, conform standardului IEEE754, valoarea de deplasament este 127 şi se adaugă la exponentul real al numărului (formând exponentul cumulat), pentru a avea întotdeauna un exponent pozitiv.

În format logaritmic (LNS) în simplă precizie, un număr binar z este reprezentat astfel:

S_z-1bit	I-8 biţi	F-23 biţi

Aceasta este o reprezentare în virgulă fixă cu i=8 biţi pentru partea întreagă I, cu f=23 biţi pentru partea fracţionară F şi un bit de semn S_z. Valoarea numărului z este dată de următoarea expresie:

$$z=(-1)^{Sz}\,2^{Nz},\ N_Z=I+F=I,F \qquad (2.2)$$

în care N_Z este logaritmul binar al numărului pozitiv $|z|$. În mod uzual se lucrează cu logaritmul în baza 2 deoarece este cel mai uşor de obţinut, [16] şi în plus majoritatea procesoarelor aritmetice operează cu numere în baza 2.

Considerând mantisa din formatul în virgulă flotantă ca fiind normalizată, adică în intervalul [1,2), partea întreagă a logaritmului numărului este dată chiar de valoarea exponentului fără deplasament, iar partea fracţionară de logaritmul mantisei. Această concatenare este posibilă deoarece logaritmul mantisei este întotdeauna un număr subunitar pozitiv. Într-adevăr, rescriind relaţia 2.1 după modelul relaţiei 2.2 se obţine:

$$A=(-1)^{S_A}\,2^{E_A-127+\log(1+0.M_A)},\qquad N_Z=E_A-127+\log(1+0,M_A) \qquad (2.3)$$

şi identificând termenii lui N_Z rezultă: $I=E_A-127$ şi $F=\log(1+0,M_A)$.

Datorită lungimii cuvintelor de date, nu a putut fi utilizată metoda simplă a citirii din memorie a logaritmului sau antilogaritmului fiecărui număr reprezentabil în formatul în simplă precizie întrucât ar fi fost nevoie de integrarea unei cantităţi enorme de memorie. Din acest motiv s-a utilizat o tabelă implementată într-o memorie ROM care va furniza doar anumite valori ce vor reprezenta valori de corecţie în metoda utilizată şi între care se va realiza interpolare liniară. Calculul logaritmului şi antilogaritmului utilizează aproximarea lui Mitchell, folosită şi de Maenner în [63], însă mult îmbunătăţită.

a)Conversia FP-LNS

Se efectuează partiţia argumentului şi memorarea doar a anumitor valori pentru reducerea cantităţii de memorie aşa cum a sugerat Maenner [63] şi în plus se aplică o metodă de corecţie care se bazează pe memorarea în anumite puncte şi a valorilor derivatei funcţiei, după care se realizează interpolarea.

Considerând numărul A dat de relaţia 2.1 şi făcând notaţia $0,M_A = y$, se poate scrie :

$$N_Z=\log_2|A| = E_A -127+ \log_2(1+y) \qquad (2.4)$$

Semnul se procesează separat, exponentului i se scade valoarea de cumul (deplasamentul de 127) deci conversia care mai trebuie făcută este de fapt calculul funcţiei $\log_2(1+y)$ pe 23 de biţi, în care $0 \le y < 1$.

Se partiţionează mantisa y în două cantităţi : y_1 conţinând cei mai semnificativi 11 biţi şi y_2 conţinând cei mai puţin semnificativi 12 biţi. Curba erorii de conversie în cazul utilizării aproximării lui Mitchell, arătată în figura 2.1.a, se discretizează în $2^{11} = 2048$ puncte ale căror proiecţii pe abscisă vor fi echidistante. Valorile în aceste puncte ale funcţiei $\log_2(1+y)-y$ sunt memorate într-un ROM ca şi valori de corecţie E_y , furnizate prin aplicarea adresei y_1.

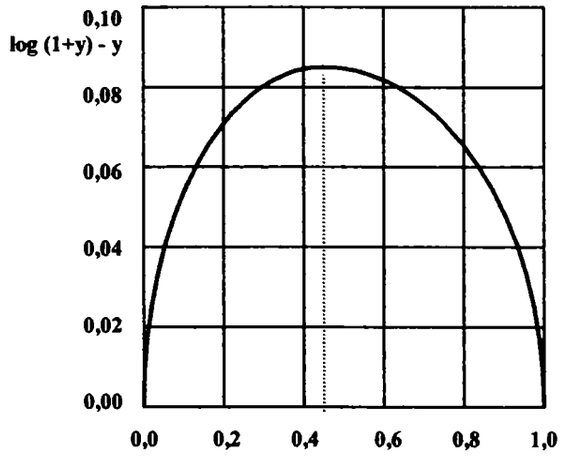


Fig.2.1.a

Curba erorii de conversie la logaritmare, în cazul utilizării aproximării lui Mitchell

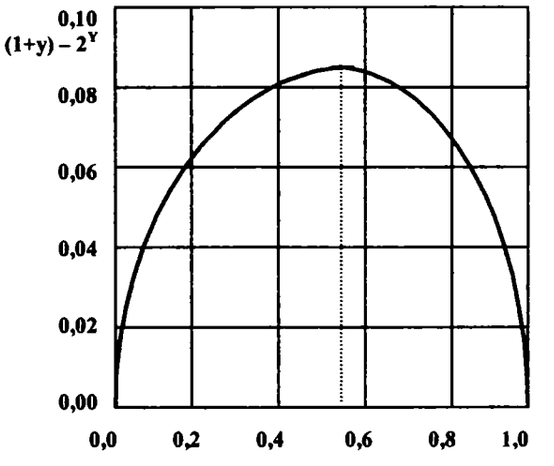


Fig.2.1.b

Curba erorii de conversie la antilogaritmare, în cazul utilizării aproximării lui Mitchell

Următoarea aproximare, îmbunătăţită, care se poate face acum, este :

$$\log_2(1+y) \approx y+E_y \qquad (2.5)$$

Totuşi în afară de cele 2048 puncte care furnizează valoarea exactă a funcţiei (afectată doar de precizia reprezentării pe 23 de biţi) celelalte valori rămân încă afectate de eroare. Reducerea în continuare a erorilor se face prin aplicarea interpolării liniare între puncte consecutive memorate. Pentru aceasta se memorează în ROM şi diferenţele ΔE_y dintre valorile învecinate ale funcţiei $\log_2(1+y)-y$. Obţinem astfel următoarea aproximare :

$$\log_2(1+y) \approx y+E_y \pm \Delta E_y \times y_2 \quad (2.6)$$

A doua tabelă ROM este mult mai mică întrucât $\Delta E_y << E_y$. Considerând pentru ΔE_y o reprezentare pe 12 biţi , conversia completă între cele două formate se face printr-o citire în tabelele de memorie, două adunări pe 24 de biţi şi o înmulţire de 12×12 biţi. Vor fi utilizate sumatoare rapide şi un multiplicator utilizând sumatoare de tip CSA aranjate în arbore Wallace.

b. Conversia LNS-FP

Procesul de antilogaritmare se tratează similar. Considerând x rezultatul antilogaritmării, se poate scrie:

$$x = 2^{(E-127)+(0.M)} = 2^{(E-127)}\, 2^{0.M} = 2^{(E-127)}2^y \quad (2.7)$$

în care E-127 reprezintă partea întreagă în format LNS, iar M reprezintă partea fracţionară. Se face următoarea aproximare:

$$x = 2^{(E-127)}\, 2^y \approx 2^{(E-127)}(1+y) \quad (2.8)$$

Se observă că şi în acest caz mantisa în format FP este, în primă instanţă, acelaşi lucru cu partea fracţionară M din format LNS. Cifra 1 este bitul implicit din formatul FP. Această primă aproximare dă o eroare de conversie de valoare (1+y) - 2^y a cărei curbă de variaţie este dată în figura 2.1.b; y se partiţionează în acelaşi mod şi se utilizează o memorie ROM pentru a memora eroarea de conversie E_y în 2048 puncte precum şi diferenţa ΔE_y . Rezultatul final al conversiei este :

$$2^y \approx (1+y)-E_y \pm \Delta E_y \times y_2 \quad (2.9)$$

Prin utilizarea algoritmilor pentru conversiile de format FP-LNS şi LNS-FP, operaţii ca înmulţirea sau împărţirea a două numere A şi B pot fi reduse la adunări sau scăderi în virgulă fixă conform relaţiilor de mai jos:

$$C = A \times B = \text{antilog}\,(\log A + \log B) \quad (2.10)$$

$$C = A / B = \text{antilog}\,(\log A - \log B) \quad (2.11)$$

În cazul numerelor cu semn, acesta se procesează separat utilizând o poartă SAU-exclusiv:

$$S_C = S_A \oplus S_B \quad (2.12)$$

Plecând de la relaţiile 2.6, 2.9, 2.10 şi 2.11, F. Lai defineşte arhitectura unei unităţi de calcul logaritmice [50], concepută pentru efectuarea operaţiilor de înmulţire (MUL) şi împărţire (DIV) în cazul a doi operanzi de intrare. Această arhitectură este prezentată în figura 2.2 şi ilustrează maniera în care este prelucrată informaţia de la intrările unităţii de calcul. În figura 2.2, A' şi B' corespund reprezentării în format FP cu exponent cumulat a celor două numere A şi B, y_A şi y_B sunt mantisele lor normalizate, y_{1A} şi y_{1B} sunt cantităţile cele mai semnificative din y_A şi y_B de lungime 11 biţi, y_{2A} şi y_{2B} sunt cantităţile cele mai puţin semnificative din y_A şi y_B de lungime 12 biţi, E_A şi E_B sunt exponenţii cu

deplasament, S_A şi S_B sunt biţii de semn, F_A şi F_B sunt părţile fracţionare ale operanzilor în format LNS, I_A şi I_B sunt părţile lor întregi, I_C şi F_C sunt partea întreagă respectiv fracţionară, în format LNS a rezultatului furnizat de ALU, iar E_C, y_C şi S_C sunt respectiv exponentul cu deplasament, mantisa şi bitul de semn al rezultatului înmulţirii/împărţirii celor doi operanzi de intrare, în format FP cu exponent cumulat.

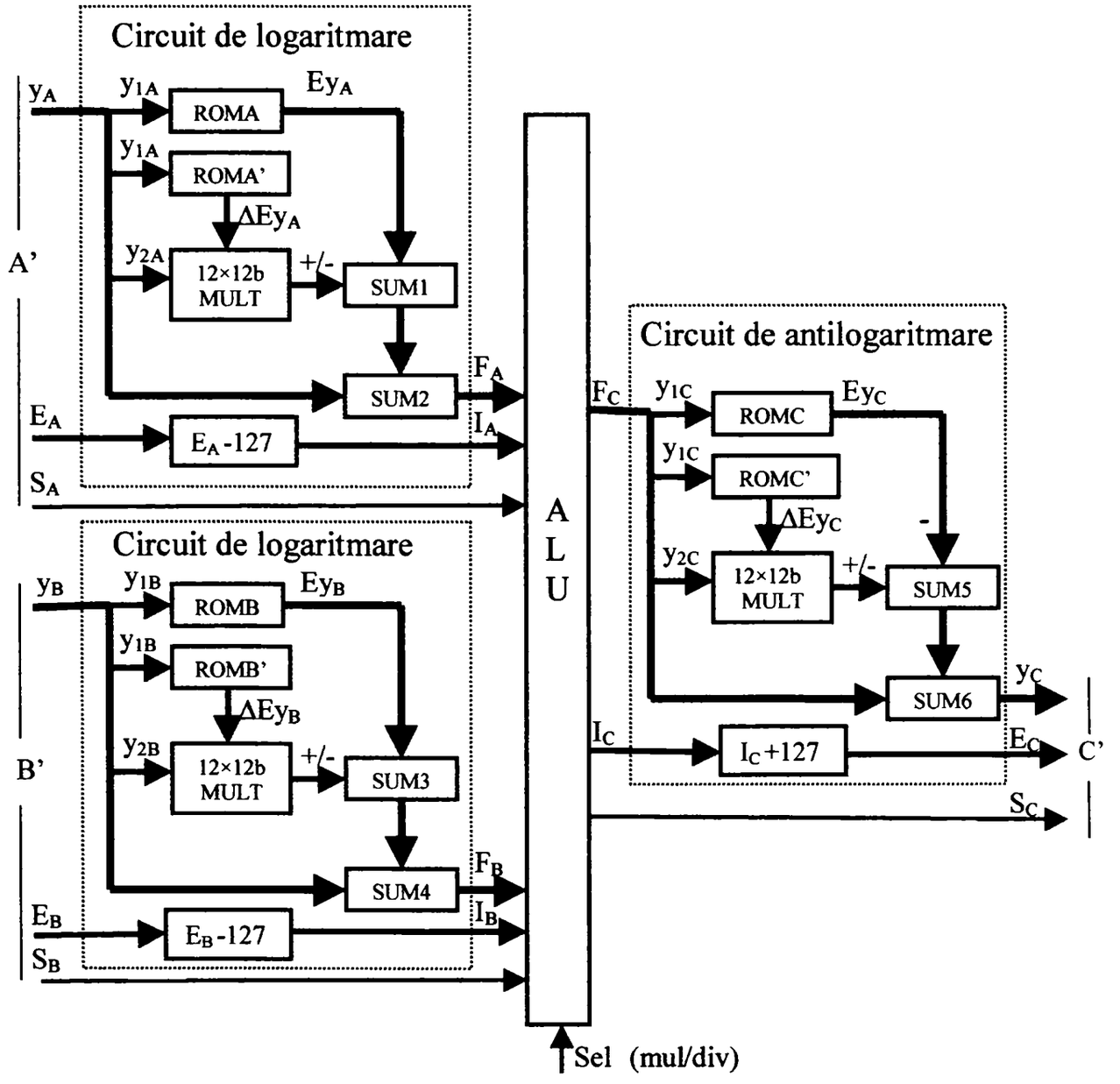


Fig. 2.2. Arhitectura subunităţii logaritmice ce implementează algoritmul lui F.Lai pentru realizarea conversiilor de format FP-LNS şi LNS-FP

Memoriile ROMA şi ROMA', accesate cu acelaşi cuvânt de adresă, sunt identice cu ROMB şi respectiv ROMB', accesate în aceeaşi manieră. Aşa cum se vede în figură, sunt necesare 2 memorii de tip ROMA+ROMA' şi o memorie de tip ROMC+ROMC'. Blocul ALU nu conţine altceva decât o poartă SAU-exclusiv

pentru implementarea relaţiei 2.12 şi un circuit sumator/scăzător pe 31 de biţi pentru implementarea relaţiei:

$$I_C,F_C = I_A ,F_A +/- I_B,F_B \qquad (2.13)$$

în care operaţia executată depinde de valoarea bitului Sel.

Arhitectura subunităţii de calcul logaritmice propusă de F. Lai în [50] pentru realizarea operaţiilor de înmulţire şi împărţire, aşa cum se observă în figura 2.2, presupune utilizarea a trei sumatoare pe 8 biţi, pentru a extrage din exponenţii celor 2 operanzi deplasamentul de 127, respectiv pentru a adăuga valoarea de deplasament la exponentul rezultatului. Deoarece în acest fel se vehiculează exponenţi care pot fi şi negativi, rezultă că ALU va opera cu date în virgulă fixă ce pot fi de orice polaritate. Aşadar apare în mod obligatoriu necesitatea asigurării conversiei în cod complement de doi pentru oricare din cele două date. În plus, atunci când se execută operaţia de împărţire care presupune scădere în ALU, trebuie făcut apel încă o dată la conversia în cod complement de doi a scăzătorului. Iar dacă rezultatul se dovedeşte a fi negativ, el este automat reprezentat în complement de doi şi deci trebuie obligatoriu reconvertit în cod semn-mărime. *Aşadar logica de comandă a ALU devine exagerat de complexă şi conduce la întârzierea producerii rezultatului.*

Soluţia pe care am propus-o şi care va fi prezentată în continuare elimină aceste dezavantaje şi nu utilizează decât un singur sumator pe 9 biţi pentru ajustarea exponentului rezultatului.

Legătura dintre numerele care intervin în relaţia 2.10 şi reprezentările lor în format virgulă flotantă cu exponent cumulat, marcate în figura 2.2, e dată de următoarele relaţii:

$$A'=A\times 2^{127},\ B'=B\times 2^{127},\ C'=C\times 2^{127}$$

Dacă se poate scrie $A \times B = C$, nu acelaşi lucru se întâmplă în cazul produsului $A'\times B'$. Întradevăr $A'\times B'= A\times 2^{127} \times B\times 2^{127} = C'\times 2^{127} \neq C'$. De aici rezultă:

$$C'= 2^{(\log A' + \log B')} \times 2^{-127} \qquad (2.14)$$

Aşadar pentru a obţine rezultatul C' plecând de la produsul $A'\times B'$ ce poate fi efectuat de către subunitatea logaritmică, este necesară scăderea din partea întreagă a rezultatului furnizat de ALU a deplasamentului de 127. În felul acesta, cele două sumatoare din figura 2.2 care extrag deplasamentul din exponenţii celor doi operanzi de la intrare nu mai sunt necesare.

În cazul operaţiei de împărţire, conform relaţiei 2.11, se poate de-asemenea scrie $A / B = C$, însă se poate constata că $A'/ B' \neq C'$. Într-adevăr $A'/ B'= (A\times 2^{127}) / (B\times 2^{127}) = C'\times 2^{-127}$. De aici rezultă:

$$C'= 2^{(\log A' - \log B')} \times 2^{+127} \qquad (2.15)$$

Cu alte cuvinte pentru a obţine rezultatul C' plecând de la raportul A'/ B' ce poate fi efectuat de către subunitatea logaritmică, este necesară adunarea la partea

întreagă a rezultatului furnizat de ALU a deplasamentului de 127. Nici în acest caz nu mai sunt necesare cele două sumatoare pe 8 biţi.

În aceste condiţii, noua arhitectură a subunităţii logaritmice se prezintă ca în figura 2.3.

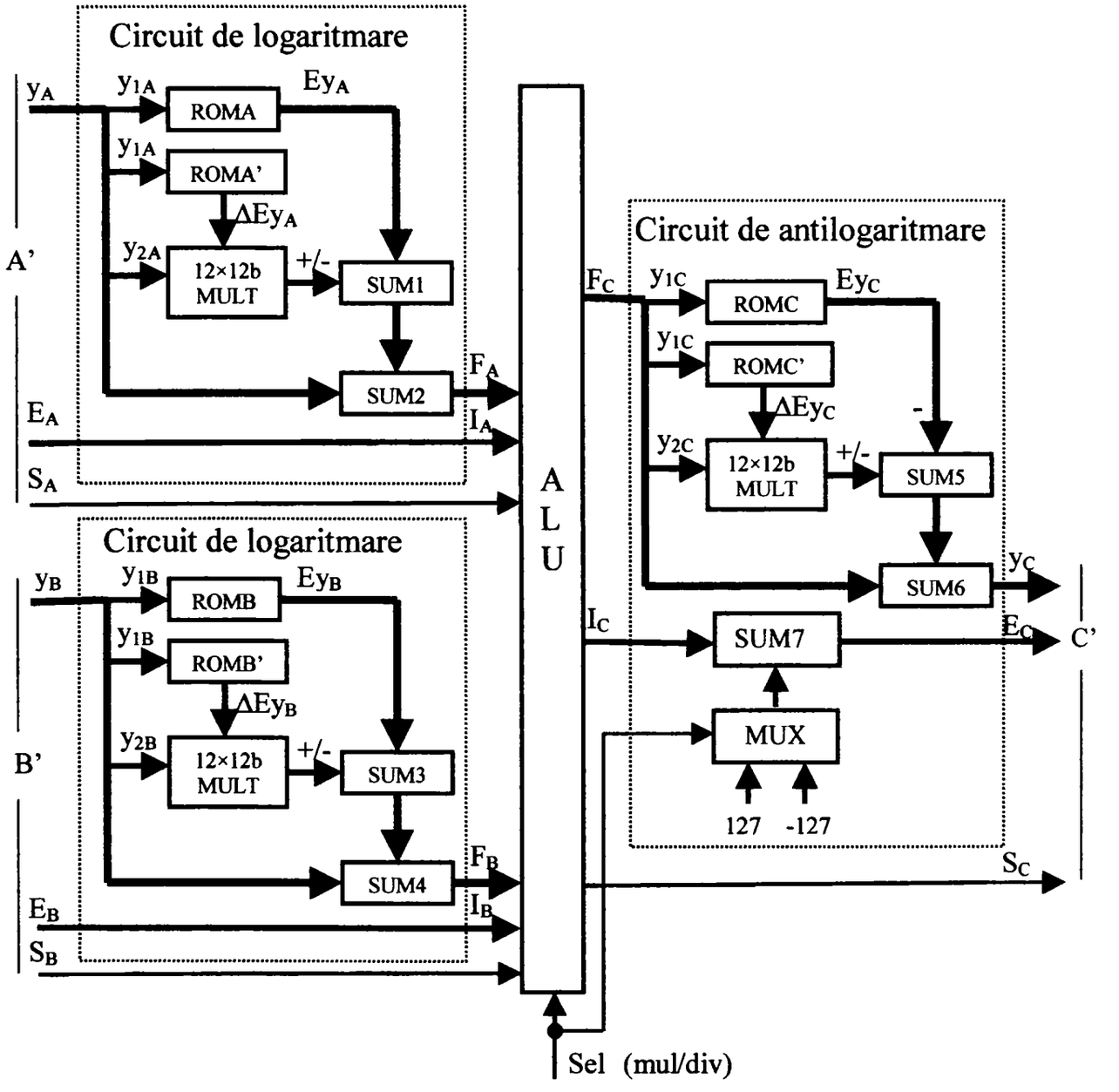


Fig.2.3. Arhitectura subunităţii logaritmice ce implementează algoritmul modificat

Aşadar, bitul Sel care specifică dacă se execută multiplicarea sau divizarea operanzilor, va selecta prin intermediul blocului MUX, scăderea respectiv adunarea deplasamentului de 127 la partea întreagă a rezultatului furnizat de ALU. Acest lucru se realizează cu sumatorul SUM7.

Sumatoarele din ALU se vor extinde de la 31 de biţi la 32 deoarece pot fi necesari 9 biţi pentru partea întreagă a rezultatului, ţinând cont că în cazul înmulţirii se adună şi deplasamentele incluse în exponenţii celor doi operanzi.

Acest lucru însă modifică nesemnificativ timpul de producere a rezultatului pentru sumatoarele cu transport anticipat folosite în ALU (de tip log(n), cu structură modulară arborescentă, conform prezentării din paragraful 1.1).

De-asemenea SUM7 va opera pe 9 biţi, dar aici întârzierea produsă la propagarea transportului datorită extinderii cu un bit, nu mai apare pe o cale critică. În plus, în cazul înmulţirii, bitul MSB (=1) de la acest sumator va fi indicator de supradepăşire (când rezultă exponent cumulat mai mare de 255), iar în cazul împărţirii acest bit în conjuncţie cu MSB al rezultatului furnizat de ALU va fi indicator de subdepăşire (când rezultă exponent cumulat negativ), respectiv de supradepăşire dacă valoarea sa logică este zero.

Eroarea maximă pe care o introduce conversia de format este 3×10^{-7}, dar *în paragraful 2.2.4 autorul va arăta cum această eroare poate fi redusă cu 40%.*

2.2.2 Proiectarea subunităţii logaritmice după metoda Lai folosind algoritmul de conversie de format modificat

În lucrarea [50] *în care a fost descris algoritmul nemodificat prezentat în paragraful anterior, au fost date doar câteva informaţii cu privire la arhitectura unităţii logaritmice şi la organizarea sa pe niveluri pipeline. Nu a fost făcută însă nici o specificaţie de proiectare cu privire la implementarea propriu-zisă a algoritmului sau la dimensionarea şi programarea memoriilor, pe baza unor calcule matematice. În consecinţă, în acest paragraf vor fi rezolvate aceste probleme printr-o proiectare completă până la nivel de poartă logică, însă pentru varianta modificată (proprie) a algoritmului, descrisă tot în paragraful precedent.* Pentru estimarea timpului total de calcul, vor fi cumulaţi timpii de propagare ai blocurilor din structura proiectată care se găsesc pe calea critică de propagare. Ulterior, pentru creşterea frecvenţei de clock şi a capacităţii de calcul va fi discutată posibilitatea împărţirii sale optime pe niveluri pipeline. Simulările ce vor fi efectuate pentru cazurile cele mai defavorabile din punct de vedere al propagării transportului, vor permite identificarea blocurilor ce reprezintă cale critică, în vederea repartizării cât mai uniforme a timpilor de propagare pe nivelurile pipeline.

2.2.2.1 Determinarea dimensiunii memoriilor. Proiectarea circuitului de logaritmare

Pentru a determina numărul maxim de biţi diferiţi de 0 pe care poate fi reprezentat E_y furnizat de memoria ROMA, în limitele formatului simplă precizie, se calculează maximul funcţiei $f(y)= \log_2(1+y)-y$ pe intervalul [0,1). Pentru aceasta se determină rădăcina ecuaţiei $f'(y) = 0$ pe acest interval:

$(\mathrm{Log}_2 e)/(1+y)- 1 = 0$

Se obţine y=0,4427. Pentru această valoare a lui y funcţia f(y) are valoarea maximă 0,086 care reprezentată ca număr fracţionar binar pe 23 de biţi prezintă 3 zerouri după virgulă. Aşadar, lungimea cuvintelor E_y ce trebuie memorate este 23 - 3 = 20 biţi.

Cuvintele ΔE_y reprezintă diferenţele dintre valorile învecinate ale funcţiei f(y) şi se memorează la fel ca şi E_y într-o tabelă de memorie de tip ROMA', care admite la intrare acelaşi cuvânt de adresă pe 11 biţi. Această a doua tabelă de memorie este mult mai mică decât cea corespunzătoare lui E_y întrucât $\Delta E_y << E_y$ şi deci numărul de zerouri după virgulă în reprezentarea fracţionară binară a lui ΔE_y este mult mai mare decât 3. Pentru a determina acest număr de zerouri (care nu mai trebuie memorate) calculăm maximul funcţiei f'(y) pe intervalul [0,1). Aşa cum se observă din figura 2.1.a, cel mai mare număr binar fracţionar ΔE_y pe 23 de biţi se obţine în punctul de pe grafic unde e cea mai mare pantă, respectiv la cea mai mică adresă diferită de 0, adică 00000000001 (001h). Această valoare maximă, în sistemul de numeraţie binar, este:

$$\log(1+0{,}00000000001) - 0{,}00000000001 = 0{,}00000000000101100100111$$

şi conţine 11 zerouri după virgulă. Deci lungimea cuvintelor ΔE_y ce trebuie memorate în cea de-a doua tabelă de memorie ROMA' este 23 - 11 = 12 biţi.

Pentru a determina numărul maxim de biţi diferiţi de 0 cu care poate fi reprezentat E_y furnizat de memoria ROMC, se calculează maximul funcţiei $f(y)=(1+y)-2^y$ pe intervalul [0,1). Pentru aceasta se determină rădăcina ecuaţiei f'(y)=0 pe acest interval:

$$1 - e^y \ln 2 = 0$$

Se obţine y=0,5288. Pentru această valoare a lui y funcţia f(y) are valoarea maximă 0,086, aceeaşi cu cea din cazul anterior. Aşadar, lungimea cuvintelor E_y ce trebuie memorate în ROMC este tot 20 biţi. În mod similar se determină lungimea maximă a cuvintelor ΔE_y memorate în ROMC', observând că cea mai mare pantă pe graficul din figura 2.1.b corespunde celei mai mari adrese. Se obţine o lungime tot de 12 biţi.

Maniera de programare a memoriilor de tip ROMA' şi ROMC' este următoarea: la adresa 0 se memorează valoarea E_y de la adresa 1 a memoriei ROMA, adică $E_y(1)$, la adresa 1 se memorează $|E_y(2)-E_y(1)|$, la adresa 2 se memorează $|E_y(3)-E_y(2)|$ ş.a.m.d. La adresa n se memorează $|E_y(n+1)-E_y(n)|$. La adresa 2047 se memorează $|0-E_y(2047)| = E_y(2047)$.

În plus, memoriile de tip ROMA' şi ROMC' trebuie să memoreze în fiecare locaţie pe un bit suplimentar, denumit bit de control şi semnul termenilor $\Delta E_y \times y_2$ din relaţiile 2.6 şi 2.9. Acest bit de control va avea valoarea 0 în cazul semnului "+" şi valoarea 1 în cazul semnului "-". În cazul memoriei ROMA', în acest bit se va memora 0 de la adresa 0 la adresa 906 şi 1 de la adresa 907 la adresa 2047. În

cazul memoriei ROMC' în acest bit se va memora 0 de la adresa 0 la adresa 1082 şi 1 de la adresa 1083 la adresa 2047.

În concluzie, cantitatea de memorie ROM ce trebuie integrată pentru realizarea conversiilor de format, în cazul a doi operanzi de intrare şi un rezultat la ieşire, este de: 2×[20×2k + (12+1)×2k] + [20×2k + (12+1)×2k] = 198 kilobiţi. Ţinând cont că la fiecare din cele 3 memorii de 66kb logica de decodificare ocupă cam 60% din suprafaţă rezultă că în tehnologie CMOS 0,5 μm, memoriile necesare se pot integra în 1,3 mm^2, iar timpul de acces este de 3,8 ns [50], [51].

Termenul $\Delta E_y \times y_2$ este obţinut cu ajutorul unui multiplicator de 12×12 biţi, astfel că numărul binar de la ieşirea acestuia va avea 24 de biţi. Deoarece y_2 este văzut ca număr binar fracţionar cu 12 biţi după virgulă (atunci când tgα într-un punct pe graficul din figura 2.1.a este normat la valoarea ΔE_y) iar numărul binar fracţionar ΔE_y are 23 de biţi după virgulă, din care primii 11 sunt zerouri, rezultă că de la ieşirea multiplicatorului vor fi reţinuţi doar cei mai semnificativi 12 biţi. Rezultatul propriu-zis este văzut ca un număr fracţionar pe 23 biţi cu 11 zerouri după virgulă, urmate de cei mai semnificativi 12 biţi de la ieşirea multiplicatorului. Acest calcul a fost necesar pentru alinierea corectă a celor trei termeni care se adună din relaţiile 2.6 şi 2.9. Schema bloc a circuitului care realizează adunarea acestor termeni, corespunzând circuitului de logaritmare, este prezentată în figura 2.4.

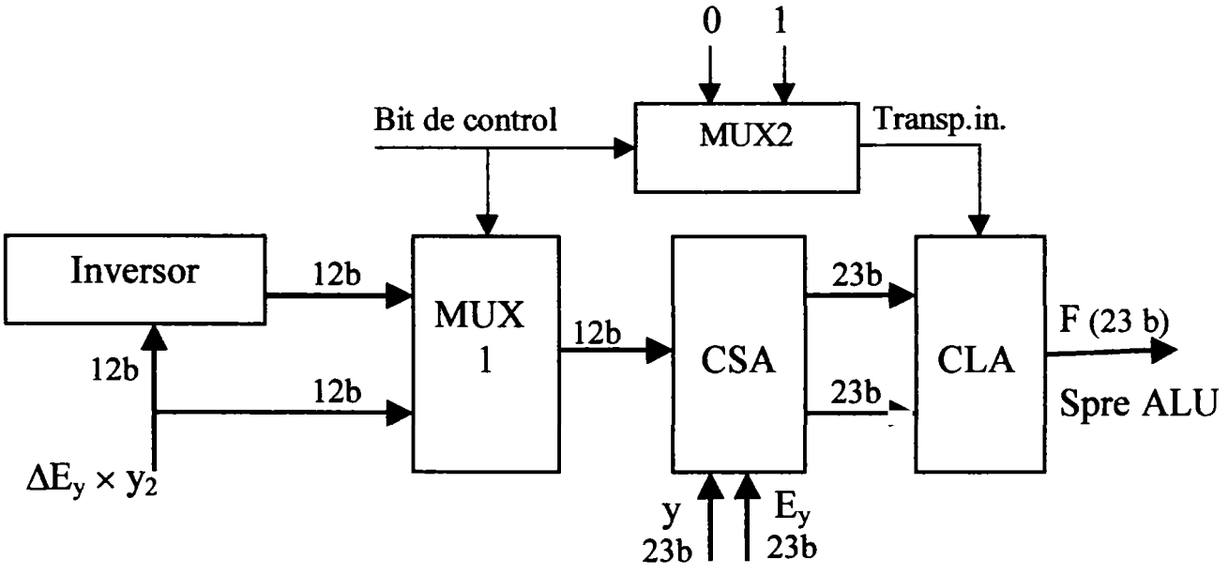


Fig. 2.4. Schema bloc a circuitului care însumează termenii y, E_y şi $\Delta E_y \times y_2$

Blocul inversor asigură complementarea biţilor termenului $\Delta E_y \times y_2$, în vederea scăderii sale. În cazul în care ΔE_y este furnizat plin aplicarea unei adrese mai mari de 906, bitul de control are valoarea 1 şi cuplează, prin intermediul blocului multiplexor MUX1, data de la ieşirea blocului inversor la

intrarea blocului CSA. În cazul în care bitul de control are valoarea 0, la intrarea blocului CSA se aplică direct termenul $\Delta E_y \times y_2$. Reprezentarea în complement de doi a termenului $\Delta E_y \times y_2$, presupune şi adăugarea unui 1 biţilor inversaţi. Având în vedere comutativitatea adunării, aceasta se poate face la sumatorul final CLA ("carry look-ahead" sau cu anticiparea transportului) luând în considerare un transport iniţial. Evident dacă termenul $\Delta E_y \times y_2$ trebuie adunat şi nu scăzut, atunci bitul de control, prin intermediul multiplexorului MUX2, selectează transport iniţial 0. Utilizarea blocului CSA ("carry save adder" sau sumator cu conservarea transportului) permite reducerea timpului de propagare pentru cele două adunări consecutive din relaţia 2.6 la aproape jumătate. Un segment din acest bloc este prezentat în figura 2.5:

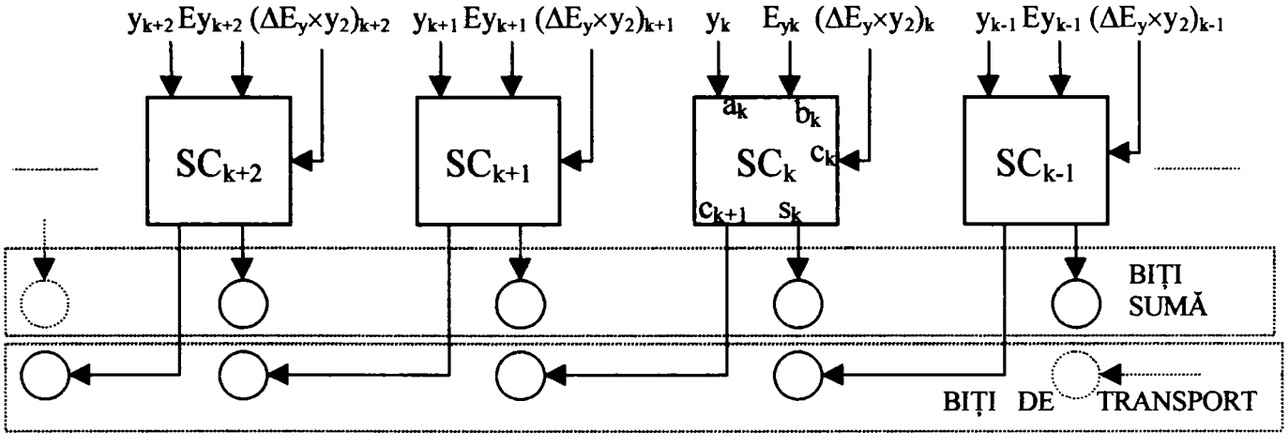


Fig. 2.5. Segment de 4 biţi din blocul CSA

Blocurile SC sunt sumatoare complete pe un bit, independente, structura lor fiind arătată în figura 1.1. Biţii de sumă şi de transport se obţin după parcurgerea a 4 şi respectiv 2 niveluri logice, adică după maxim 4 unităţi de timp de propagare. Astfel blocul CSA poate accepta la intrare 3 operanzi de 23 de biţi şi produce 2 cuvinte de câte 23 de biţi, decalate cu un bit, plasate în secţiunile "sumă" şi "transport", aşa cum se vede în figura 2.5.

Deoarece la nivelul unui sumator complet, din cele 23, adunarea celor 3 biţi este independentă de biţii de rang inferior sau superior, întreaga operaţie va dura 4 unităţi de timp de propagare. Termenii y, E_y şi $\Delta E_y \times y_2$ care se aplică la intrările CSA reprezintă numere fracţionare care au 23 de biţi, din care al doilea are primii 3 biţi după virgulă egali cu zero, iar al treilea are primii 11 biţi după virgulă 0 sau 1, în funcţie de valoarea bitului de control. Rezultatul final al adunării celor 3 operanzi se obţine cu ajutorul unui sumator final de tip CLA care adună cuvintele din secţiunile sumă şi transport ale blocului CSA. Sumatorul de tip CLA a fost descris în paragraful 1.3, dar ulterior va fi îmbunătăţit.

Pentru estimarea timpului de propagare prin circuitul care însumează termenii y, E_y şi $\Delta E_y \times y_2$, au fost folosite datele furnizate de [67] cu privire la timpii de propagare prin diversele porţi logice corespunzătoare unei tehnologii CMOS de 0,5 micrometri. Astfel, timpul de propagare printr-o poartă SAU-

exclusiv este 0,35 ns, prin porţi simple cu două intrări, de tip ŞI-NU şi SAU-NU, este de 0,2 ns, în timp ce pentru un inversor sau tranzistor de trecere acest timp este de 0,1 ns. *Rezultă astfel* $t_P=t_{inversor}+t_{MUX}+t_{CSA}+t_{CLA}= 0,1+0,1+2\times0,35+2,3=3,2$ *ns. Valoarea lui* t_{CLA} *a fost obţinută prin simulare pentru cazul cel mai defavorabil al unui sumator performant de tipul celui prezentat în figura 2.12, dar pe 23 de biţi.*

2.2.2.2 Proiectarea multiplicatorului de 12×12 biţi

Maniera asincronă în care a fost proiectat multiplicatorul de 12×12 biţi a urmărit furnizarea rezultatului înmulţirii celor doi operanzi într-un timp cât mai scurt. Din acest motiv au fost folosite blocuri CSA, similare cu cel descris în paragraful anterior, care grupează produsele parţiale din aria de înmulţire într-un arbore binar de tip Wallace. Viteza mare de calcul se obţine pe de o parte datorită propagării doar pe verticală a transportului prin blocurile CSA din aria de înmulţire, iar pe de altă parte datorită calculării simultane a mai multor produse parţiale intermediare, avantaj oferit de structura în arbore Wallace. Schema bloc a acestui arbore este prezentată în figura 2.6.

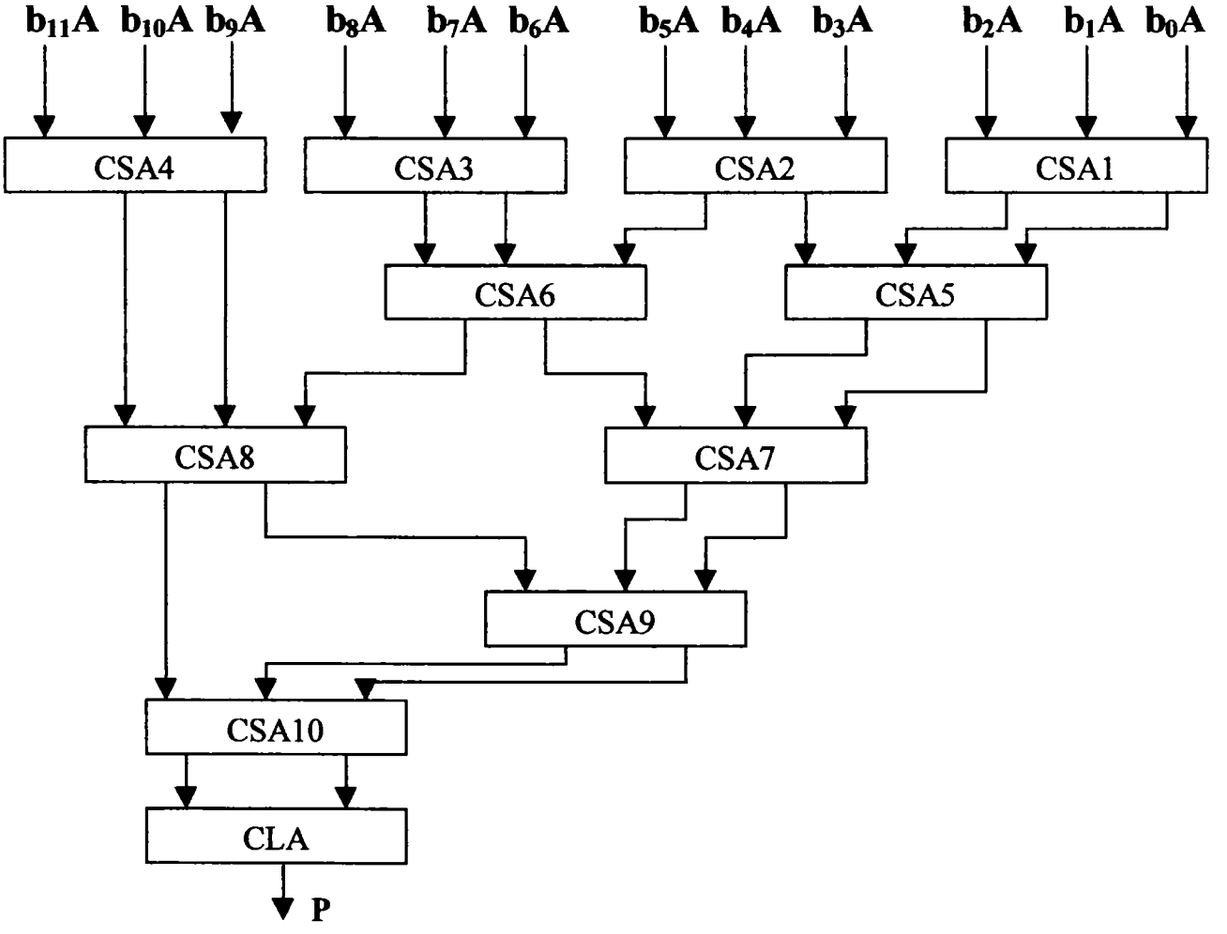


Fig. 2.6. Schema bloc a arborelui Wallace realizat cu blocuri CSA

Aşa cum se observă în această figură, fiecare bloc CSA acceptă la intrare trei produse parţiale iniţiale sau trei produse parţiale intermediare, furnizând la ieşire alte două produse parţiale intermediare. Are loc astfel o reducere treptată a

numărului de produse parţiale, de-a lungul a 5 niveluri de blocuri CSA, până când, în final, rezultă doar două cuvinte ce trebuie însumate. Toate blocurile CSA de pe acelaşi nivel lucrează simultan. Fiind vorba de structuri cu conservarea şi nu propagarea pe orizontală a transportului, un nivel CSA este parcurs în maxim 4 unităţi de timp de propagare. Aceasta înseamnă că timpul de propagare prin cele 5 niveluri este de 20 unităţi de timp de propagare.

Deoarece produsele parţiale iniţiale, au fiecare pondere dublă faţă de cel anterior (trebuie aliniate decalat cu câte un bit), iar diferitele produse parţiale intermediare care se grupează într-un bloc CSA au ponderi diferite (se aliniază decalat, în funcţie de ponderea fiecăruia), rezultă lungimi diferite pentru blocurile CSA utilizate. De exemplu, în blocul CSA5 ponderile şi lungimile celor trei produse parţiale intermediare care se grupează aici au condus la alinierea prezentată în figura 2.7. Lungimea acestui bloc este de 15 biţi, din care bitul LSB din secţiunea biţilor sumă de la ieşirea sa, ne mai combinându-se ulterior cu nimic, ajunge direct bitul P_2 al produsului final. Se observă de asemenea că în acest caz nu sunt necesare decât 11 sumatoare complete pe 1 bit şi un sumator incomplet, întrucât cei mai semnificativi trei biţi din al treilea produs parţial intermediar de intrare ajung direct în secţiunea de ieşire a biţilor sumă, de lungime 14 biţi. Secţiunea biţilor de transport conţine aici 12 biţi.

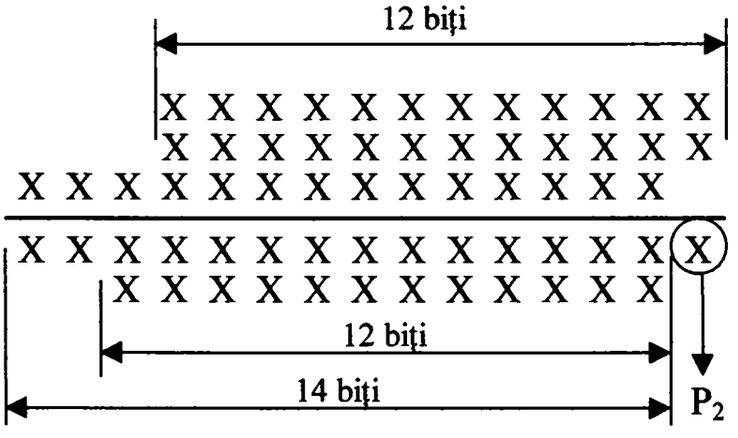


Fig. 2.7. Alinierea produselor parţiale intermediare în blocul CSA5

Spre vârful arborelui produsele parţiale intermediare devin din ce în ce mai lungi, dar aceasta nu conduce la creşterea semnificativă a numărului de porţi logice utilizate într-un bloc CSA deoarece un grad mai mic de suprapunere a cuvintelor de intrare presupune utilizarea mai multor sumatoare incomplete în loc de sumatoare complete pe un bit, sau poate însemna chiar transferul biţilor mai semnificativi direct la ieşire.

Aşa cum s-a putut constata şi în cazul blocului CSA5, cei mai puţin semnificativi 6 biţi ai produsului sunt furnizaţi de structura hard a arborelui Wallace, astfel că lungimea necesară a sumatorului final este de 18 biţi. S-a optat pentru un sumator rapid cu anticiparea transportului (de tip CLA). Acesta a fost proiectat conform regulilor descrise în paragrafele 1.1 şi 2.2.2.4. Pentru calculul produselor parţiale iniţiale de forma b_iA, cu $i = 0 \div 11$ au fost utilizate blocuri de

câte 12 porţi ŞI cu 2 intrări şi arbori de inversoare cu 2 etaje (cu 2 şi respectiv 4 inversoare pe nivel) pentru multiplicarea capabilităţii de comandă a biţilor celor doi factori, în vederea generării celor 12 produse parţiale. Timpul de propagare prin multiplicator rezultă: $t_P=2\times t_{inv}+t_{ŞI}+t_{arbore}+t_{CLA}= 0,2+0,3+10\times0,35+2,3= 6,3$ ns.

2.2.2.3 Circuitul de antilogaritmare

Proiectarea secţiunii circuitului de antilogaritmare care se ocupă de partea fracţionară a rezultatului furnizat de ALU are ca bază de plecare relaţia 2.9. Diferenţele, din punct de vedere formal, faţă de relaţia 2.6 care a condus la implementarea circuitului de logaritmare, constau în acelea că termenul E_y trebuie scăzut şi nu adunat şi că apare valoarea 1 în paranteza (1+y). Această valoare este memorată implicit în structurile hard care operează cu numere în virgulă flotantă normalizate, deci nu e necesar ca ea să se regăsească pe undeva în structura circuitului de antilogaritmare, acesta furnizând implicit o mantisă normalizată.

Se observă din relaţia 2.9 că dacă se înlocuieşte scăderea termenului E_y cu însumarea complementului de doi al acestuia, aceasta conduce la o implementare hard identică a circuitului de antilogaritmare cu cea a circuitului de logaritmare prezentat la nivel de schemă bloc în figura 2.4.

Diferenţele se regăsesc evident în conţinutul memoriilor ROMC şi ROMC'. Astfel ROMC va memora complementul de doi al cantităţilor E_y corespunzătoare reprezentării grafice din figura 2.1.b în loc de chiar E_y. În cazul memoriei ROMC', programarea sa se va face similar cu a memoriei ROMA' pe baza valorilor E_y precalculate, corespunzătoare graficului din figura 2.1.b. În locaţiile corespunzătoare bitului de control al semnului cantităţii ΔE_y se va memora 0 de la adresa 0 la adresa 1082 şi 1 de la adresa 1083 la adresa 2047.

În ceea ce priveşte secţiunea circuitului de antilogaritmare, care furnizează exponentul cumulat al rezultatului, aceasta se regăseşte în figura 2.3. Sumatorul SUM7 din figura 2.3 care produce pe E_C este prezentat în figura 2.8.

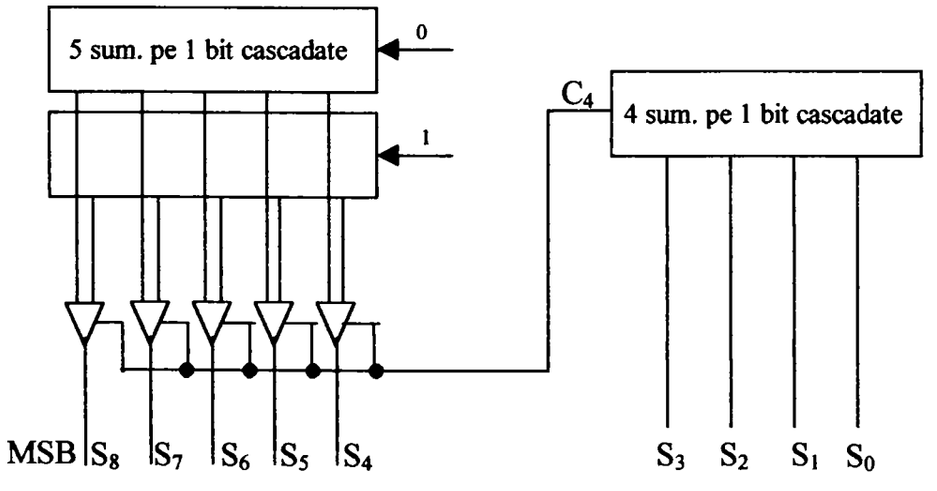


Fig.2.8. Schema bloc a sumatorului care produce exponentul rezultatului

SUM7 este un sumator cu selecţia transportului organizat pe două secţiuni de 4 şi respectiv 5 biţi. Transportul C_4 selectează rezultatul corect antecalculat de cele două sumatoare pe 5 biţi ce primesc transport iniţial 0, respectiv 1. Cele două secţiuni includ sumatoare complete pe un bit (sumatoare complete 3:2), cascadate, la care întârzierea produsă de propagarea din aproape în aproape a transportului se face pe o cale care nu este critică, întrucât SUM7 funcţionează în paralel cu cealaltă parte a circuitului de antilogaritmare, care furnizează pe y_C într-un interval de timp mai mare. Ieşirea circuitului de antilogaritmare reprezintă practic ieşirea subunităţii logaritmice.

2.2.2.4 Proiectarea circuitului de adunare/scădere a operanzilor în format LNS, cu deplasament

Cei doi operanzi în format LNS simplă precizie şi cu deplasament, furnizaţi de cele două circuite de logaritmare ce funcţionează simultan, au aşadar lungimea de 32 de biţi. Reprezentarea lor în virgulă fixă include 9 biţi pentru partea întreagă şi 23 de biţi pentru partea fracţionară. Din punct de vedere hard, circuitul care realizează adunarea sau scăderea lor este identic cu acela care realizează adunarea sau scăderea a doi operanzi întregi de 32 biţi.

Includerea deplasamentului în partea întreagă a operanzilor de la intrarea în ALU face ca aceştia să fie întodeauna pozitivi. În aceste condiţii structura ALU se simplifică, ea se reduce la un circuit de adunare/scădere care operează cu date pozitive.

Efectuarea operaţiei de împărţire în subunitatea logaritmică presupune scădere în ALU, adică conversia în complement de doi a scăzătorului. Dacă, în plus, rezultatul se dovedeşte a fi negativ, atunci el este automat reprezentat în complement de doi şi trebuie ulterior reconvertit în cod semn-mărime. Întârzierea produsă ar putea fi eliminată, dacă pentru cei doi operanzi A şi B de la intrarea în ALU se calculează simultan A-B şi B-A după care se selectează rezultatul pozitiv şi se stabileşte semnul corect. Şi în acest caz trebuie asigurată conversia în complement de doi pentru ambii operanzi cât şi posibilitatea efectuării adunării propriu-zise. Soluţia pe care am adoptat-o, [26], se bazează pe următoarele relaţii:

$$A - B = A + \overline{B} + 1 = A + /B + 1 \qquad (2.16)$$

$$B - A = \overline{A + \overline{B}} \quad (A\text{-}B = -(B\text{-}A) = \overline{A+\overline{B}+1}) \qquad (2.17)$$

în care /B este data obţinută prin inversarea biţilor lui B. Pentru efectuarea operaţiei de scădere, circuitul sumator/scăzător calculează simultan A+/B şi A+/B+1 utilizând două sumatoare pe 32 de biţi, unul fără transport iniţial iar celălalt cu transport iniţial. Dacă A+/B+1 este pozitiv, el este selectat la ieşirea ALU. Dacă A+/B+1 este negativ, atunci A+/B este selectat la ieşire, după ce în prealabil biţii săi au fost inversaţi. Schema bloc a acestui circuit este prezentată în figura 2.9. *Spre deosebire de schema propusă în* [26], *soluţia proprie are*

topologia blocurilor componente schimbată, permiţând selectarea corectă a rezultatului, nu numai în cazul scăderii, dar şi în cazul adunării operanzilor. În plus, schema propusă e mai rapidă deoarece blocul Inversor2 nu mai aşteaptă stabilirea bitului MSB, marcat în figură şi apoi rezultatul de la blocul Selector. În funcţie de valoarea bitului MSB, dar condiţionat de linia de control Sel (Sel=0 în cazul adunării, respectiv Sel=1 în cazul scăderii) este selectat rezultatul de la blocul Sumator1, inversat sau neinversat, sau de la blocul Sumator2.

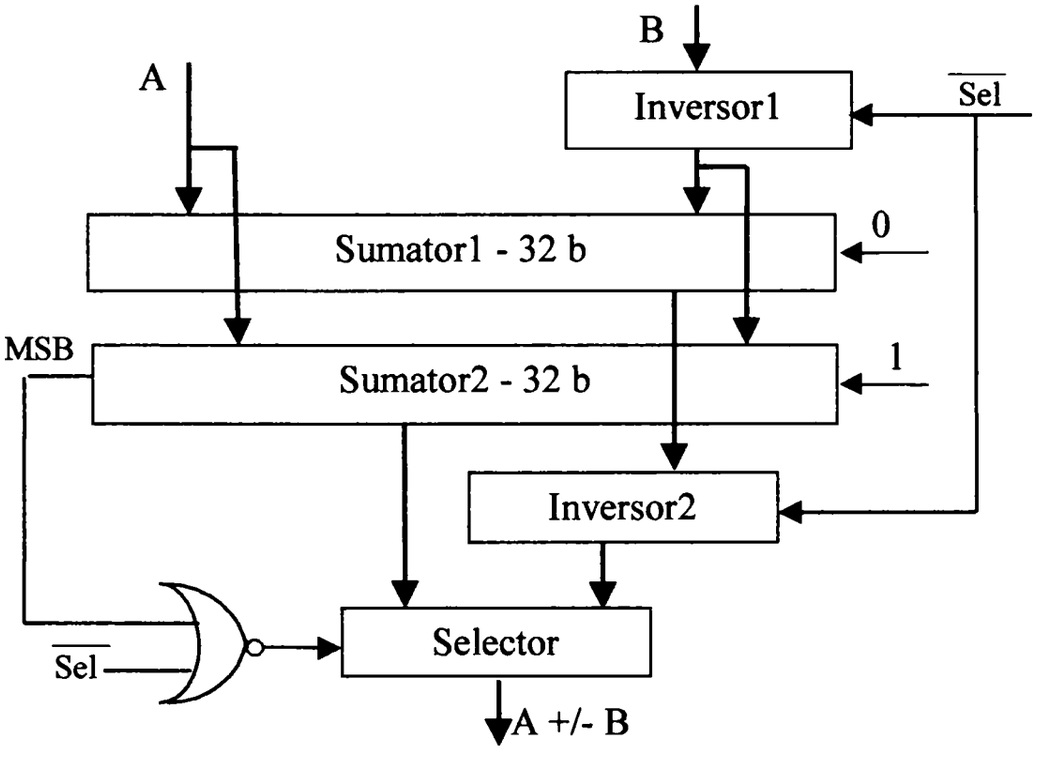


Fig. 2.9. Schema bloc a circuitului de adunare/scădere din componenţa ALU

Atunci când se efectuează adunare, doar blocul Sumator1, cu transport iniţial 0, operează, în timp ce blocurile Inversor1 şi Inversor2 vor fi transparente.

Cele două sumatoare sunt sumatoare rapide, alese într-o primă etapă, de tip CLA (carry look-ahead) sau cu anticiparea transportului şi au fost prezentate în paragraful 1.1. Dacă se analizează schema la nivel de poartă a blocurilor de tip A şi B prezentate în figurile 1.5 şi 1.6 care formează structura modulară a sumatorului, se constată că sunt utilizate porţi ŞI, porţi SAU şi porţi SAU-exclusiv. Dacă în tehnologie CMOS poarta SAU-exclusiv, aşa cum se va vedea într-un alt paragraf, are o structură optimizată, nu acelaşi lucru se întâmplă cu porţile ŞI şi SAU care practic se realizează prin adăugarea a câte unui inversor CMOS la porţile ŞI-NU, respectiv SAU-NU intrinseci. Aceasta ar însemna, pentru un sumator rapid pe 32 de biţi, integrarea unui număr de 219 inversoare, care conduce evident la creşterea suprafeţei ocupate, cât şi a timpului de operare.

Întrucât literatura de specialitate nu prezintă soluţii concrete pentru implementarea optimă în tehnologie CMOS a unui sumator rapid de tipul carry look-ahead (sau cu anticiparea transportului) cu structură modulară, de tipul celei

din figura 1.4, am oferit o soluţie personală pentru eliminarea dezavantajelor menţionate.

Astfel, am modificat structura blocurilor de tip A şi B care se regăsesc în schema bloc a sumatorului din figura 1.4. Noul bloc A' derivat din blocul de tip A va avea structura din figura 2.10.a. Acest bloc va furniza practic coeficienţii p şi g din relaţiile (1.3) complementaţi. Introducerea, într-o etapă intermediară de sinteză, a unor inversoare pe aceleaşi linii, pentru refacerea logicii, dar în interiorul blocurilor de tip B aflate imediat sub blocurile A' şi aplicarea apoi a transformărilor De Morgan, a condus la obţinerea blocurilor de tip B' a căror structură este prezentată în figura 2.10.b. Deoarece starea logică de la ieşirile blocurilor B' este identică cu cea de la ieşirea blocurilor de tip B, a doua linie de blocuri de tip B din schema iniţială, se transformă ca şi blocurile A, pentru a furniza ieşiri complementate, obţinându-se blocurile de tip B" (figura 2.10.c).

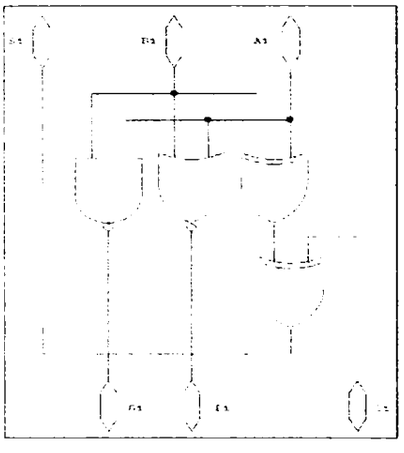


Fig.2.10.a
Schema blocului de tip A'

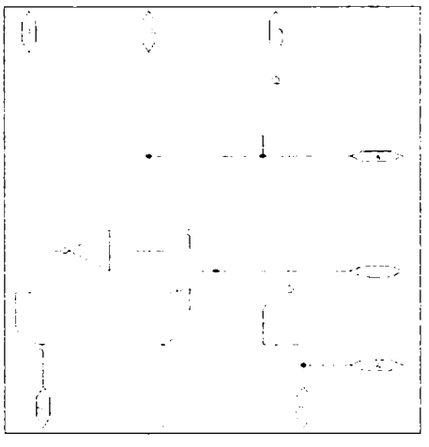


Fig.2.10.b
Schema blocului de tip B'

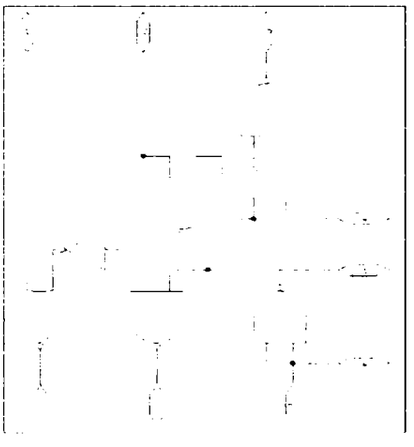


Fig.2.10.c
Schema blocului de tip B"

În acest fel se asigură compatibilitatea logică cu a treia linie de blocuri care va fi din nou de tip B'. A patra linie de blocuri va fi de tip B", păstrându-se astfel alternanţa. Schema bloc a sumatorului pe 32 de biţi menţine aşadar aceeaşi organizare, dar conformaţia blocurilor componente şi într-o anumită măsură şi dispunerea lor va fi diferită, aşa cum se observă în figura 2.11.

Schema internă la nivel de poartă logică a acestui sumator nu mai conţine decât porţi optimale pentru tehnologie CMOS, respectiv porţi cu două intrări de tip ŞI-NU, SAU-NU şi SAU-exclusiv, respectiv inversoare CMOS. Numărul de inversoare necesar a fi integrate s-a redus astfel de la 219 la 62, iar câştigul de viteză obţinut, aşa cum se va vedea în capitolul ce prezintă rezultatele simulărilor, a fost de peste 25%. Această procedură va fi extinsă la orice alt sumator de acelaşi tip din unitatea de calcul. Au fost făcute simulări cu programul Msim52, la care timpii de propagare ai diverselor porţi logice utilizate, extrase din bibleoteca de modele a simulatorului, au fost setaţi în conformitate cu datele oferite de [67], corespunzătoare unei tehnologii CMOS de 0,5μm, adoptată ca şi etalon.

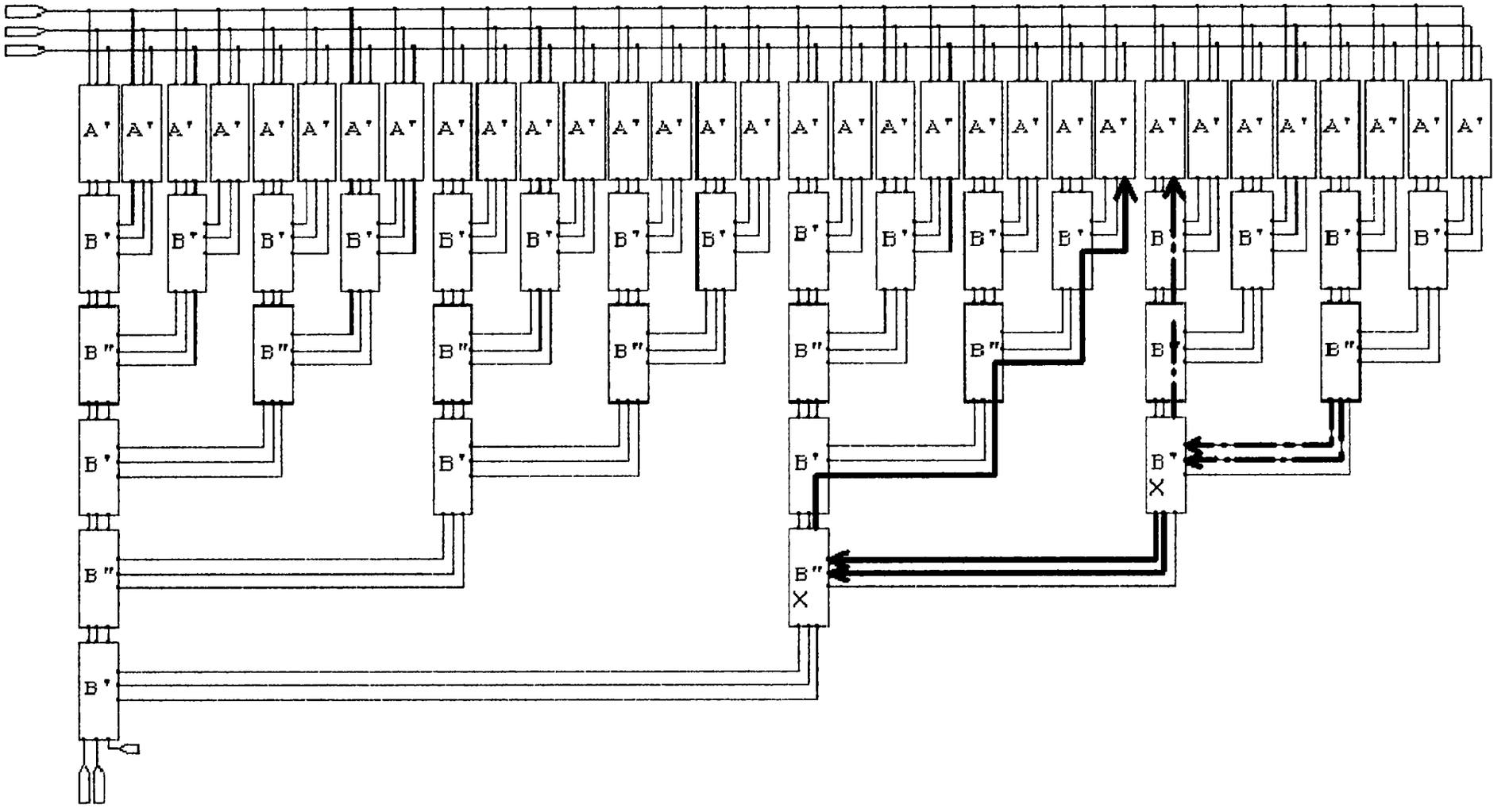


Fig. 2.11. Schema bloc a sumatorului CLA cu structură modulară, pe 32 de biţi, modificat

Pentru sumatorul de tip CLA pe 32 de biţi, în urma modificărilor efectuate, timpul de calcul a coborât de la 5,7 ns la 4,3 ns, pentru cazul cel mai defavorabil. În cazul unui sumator de tip CLA pe 16 biţi, timpul de calcul a fost de 3,5 ns, iar pentru unul pe 8 biţi, acest timp a fost de 2,4 ns.

Depistarea cazului celui mai defavorabil, respectiv a perechii de operanzi, pentru care se obţine cel mai mare timp de propagare, la un sumator CLA pe 8 biţi, am realizat-o cu ajutorul unui program scris în limbaj C. Programul implementează relaţiile recurente (1.5), (1.6) şi (1.7) şi asociază contoare care se incrementează la fiecare schimbare a stării logice după fiecare nivel logic parcurs. El este prezentat în Anexa2. În urma rulării programului au fost testate toate cele 65536 combinaţii posibile de operanzi şi au fost găsite 256 de perechi de valori pentru care se obţine timpul maxim de generare a rezultatului. Între aceste perechi de valori s-a găsit şi cea care corespunde cazului celui mai defavorabil de la sumatorul cu propagarea din aproape în aproape a transportului (la care 8 sumatoare complete pe un bit sunt cascadate), respectiv FFh şi 01h. În schimb cazul FFh şi 00h cu transport iniţial 1 nu reprezintă un cel mai defavorabil caz. Acest lucru a fost confirmat şi prin simulare. Concluzia importantă care s-a desprins a fost că, datorită structurii modulare a unui sumator CLA, cazul particular obţinut a putut fi extins ca şi cel mai defavorabil caz şi la sumatoare CLA mai mari. Pentru un sumator CLA pe 32 de biţi aceasta a însemnat FFFFFFFFh şi 00000001h.

Etapa următoare în proiectare, pe care am abordat-o, a fost analiza generării şi propagării informaţiei de la primul bloc de 8 biţi al sumatorului CLA pe 32 de biţi, în vederea stabilirii transportului iniţial al următorului bloc de 8 biţi. Observaţia importantă făcută a fost că transportul iniţial (marcat c8 în figura 2.11) pentru al doilea bloc de 8 biţi se obţine cu un interval de timp semnificativ înaintea unora din primii 8 biţi ai sumei. Într-adevăr, aşa cum se observă din figura 2.11, calea pentru generarea lui c8, de la ieşirea blocului B'x, marcată cu linie continuă îngroşată, nu presupune decât parcurgerea a două niveluri logice în blocul B"x. Celelalte blocuri de pe traseu vor fi transparente pe linia menţionată, aşa cum se observă din figurile 2.10 şi 2.11. În schimb, calea pentru generarea lui c7, necesar pentru calcularea bitului s7 al sumei, marcată cu linie întreruptă îngroşată, presupune parcurgerea a 4 niveluri logice prin blocurile B" şi B' de deasupra lui B'x. La acestea se adaugă încă două niveluri logice, în blocul A' pentru generarea lui s_7.

Această proprietate mi-a permis să concep un tip nou de sumator rapid pe 32 de biţi care în ansamblu este un sumator cu selecţia transportului, de tipul celui prezentat în figura 1.8, dar la care sumatoarele componente, pe câte 8 biţi, sunt de tipul CLA. Schema bloc a acestui sumator este prezentată în figura 2.12.

Blocurile de tip CONECTOR1,2 permit doar disocierea celor două magistrale de intrare de 32 de biţi în câte 4 magistrale de 8 biţi. Blocul CONECTOR3 asociază cele 4 magistrale de ieşire de 8 biţi într-una singură de 32 de biţi. Circuitul conţine un prim sumator CLA cu transport iniţial 0 şi trei perechi

de sumatoare CLA cu transport iniţial 0 (blocurile "SUM8_Cin=0"), respectiv 1 (blocurile "SUM8_Cin=1"). Toate sumatoarele lucrează simultan. În momentul în care transportul c8 de la ieşirea primului sumator este cunoscut, el va selecta prin intermediul blocului SELECTOR1 rezultatul de la acel sumator din prima pereche care a avut transportul iniţial egal cu c8. În acelaşi timp este selectat şi transportul de ieşire corect, furnizat de acelaşi sumator. Acest transport va selecta apoi, prin intermediul blocului SELECTOR2, rezultatul corect împreună cu transportul de ieşire asociat, din a doua pereche de sumatoare. *Deoarece transporturile c8 se obţin înaintea stabilirii tuturor biţilor de ieşire ai sumatoarelor, cele 4 grupe de câte 8 biţi ai sumei finale se obţin aproape instantaneu.*

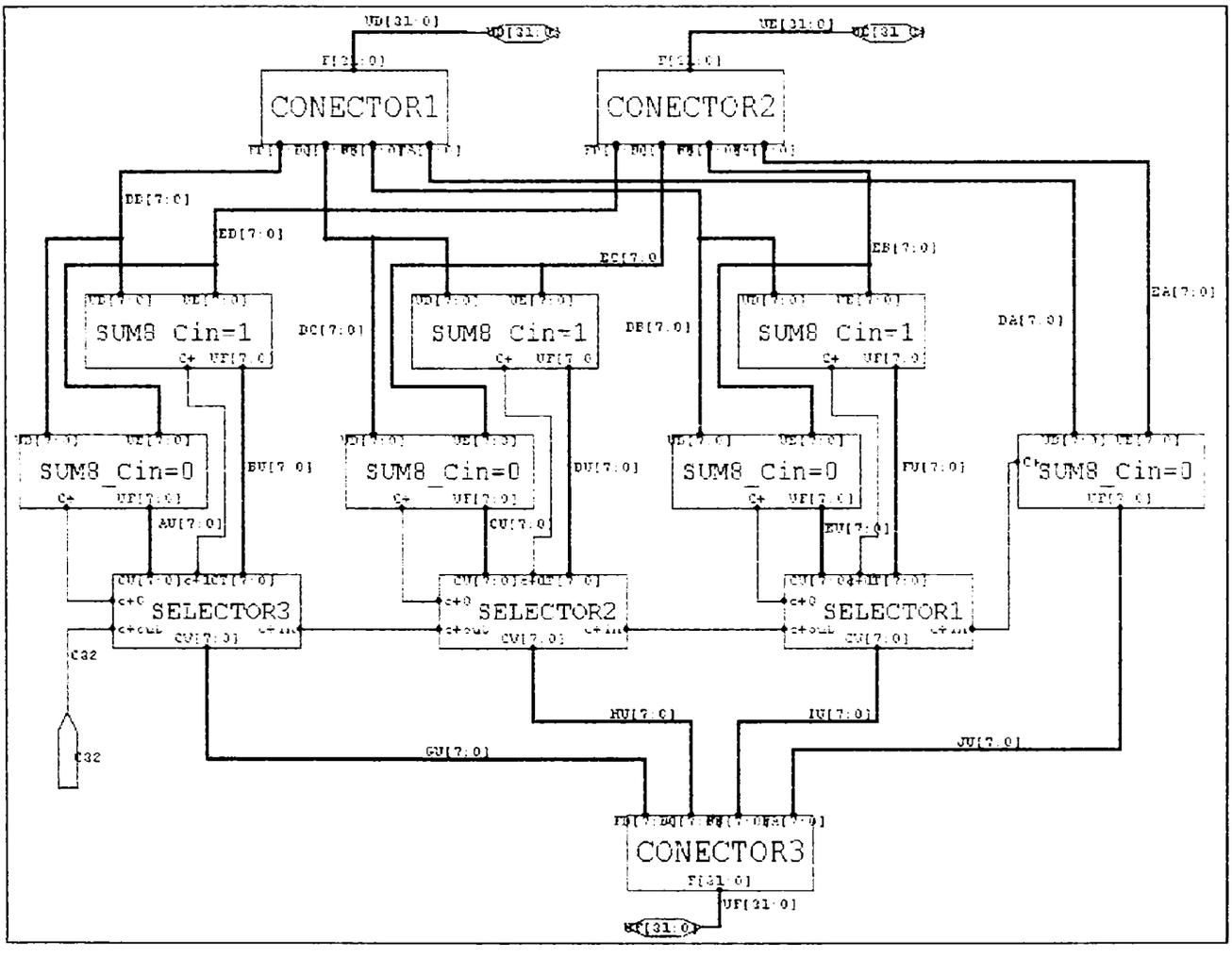


Fig. 2.12. Schema bloc a noului sumator rapid pe 32 de biţi

Blocurile selectoare sunt realizate cu câte 9 perechi de tranzistoare de trecere complementare. În realitate, fiecare bit c8 trebuie să comande în fiecare bloc selector 18 grile de tranzistoare MOS. Multiplicarea capabilităţii liniilor de selecţie cu ajutorul unor arbori cu inversoare CMOS ar conduce însă la o întârziere cumulată semnificativă. Acest neajuns este compensat în bună parte datorită proprietăţii evidenţiate anterior a sumatorului de tip CLA. În plus, am definit o linie de selecţie prioritară pentru selectarea transportului iniţial corect

pentru următoarea pereche de sumatoare, arborele de inversoare dezvoltându-se în aval de aceasta. Acest lucru poate fi observat în figura 2.13. Aici, transportul iniţial "C+in" selectează "C+out" dintre "C+0" şi "C+1" deja produşi şi ulterior se multiplică capabilitatea liniei de selecţie. Astfel se menţine practic viteza sumatorului pe 32 de biţi ca şi a unuia pe 8 biţi.

Prin mecanismul de selecţie adoptat, fiecărei ieşiri CMOS îi revine un număr de maxim 4 grile MOS pentru comandă. Deoarece arborele de inversoare are un număr impar de etaje, logica de selecţie a blocului multiplexor MuxTrib este inversă faţă de cea a blocurilor Mux2-1.

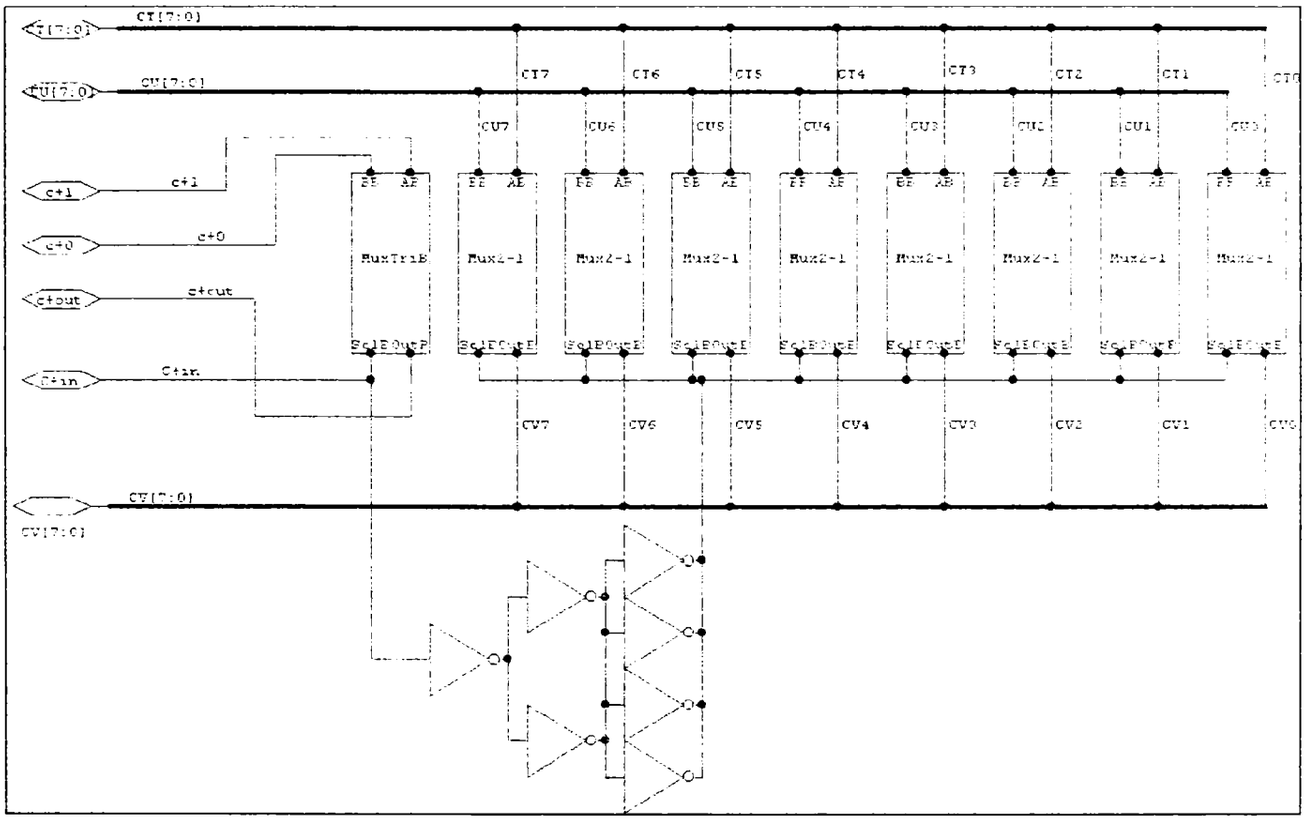


Fig. 2.13. Schema blocului Selector ce include calea de selecţie prioritară

În urma simulărilor, pentru acest sumator hibrid s-a confirmat un timp de propagare de 2,6 ns, pentru cazul cel mai defavorabil. Viteza obţinută este deci cu aproape 70% mai mare faţă de a sumatorului CLA modificat din figura 2.11 şi cu 120% mai mare decât a sumatorului CLA nemodificat de tipul celui din figura 1.4, extins la 32 de biţi. Aria ocupată este însă mai mare cu aproximativ 75% faţă de a sumatorului CLA modificat.

Făcând o comparaţie de viteză cu unele realizări recente în domeniu, sintetizată prin rezultatele din tabelul 2.1, se poate constata viteza superioară a sumatorului propriu. Toate valorile sunt raportate la tehnologia considerată etalon în această lucrare şi anume 0,5 micrometri CMOS.

Plecând de la acest tip de sumator a fost proiectat *circuitul sumator/scăzător pe 32 de biţi* a cărui schemă bloc a fost prezentată în *figura 2.9. Schema sa detaliată, organizată pe 4 niveluri ierarhice, este prezentată în Anexa 1.*

Pe lângă acest circuit, blocul ALU din figura 2.3 mai include poarta SAU-exclusiv pentru stabilirea semnului rezultatului înmulţirii/împărţirii celor doi operanzi de la intrarea subunităţii logaritmice de calcul.

Tab. 2.1

Sumator 32 biţi	propriu	[50]1991	[85]1996	[78]1999
Timp de propagare	2,6 ns	5ns	4,2ns	3ns

Simulările efectuate pentru circuitul sumator/scăzător, pentru cazul cel mai defavorabil, au confirmat viteza foarte mare de calcul. Timpul maxim de propagare a fost găsit de 3.3 ns, în tehnologie CMOS 0,5μm. Deoarece însă la blocul Inversor1 din figura 2.9 linia "/Sel" este deja setată (odată cu stabilirea tipului operaţiei pe care o execută subunitatea logaritmică) înainte ca circuitele de logaritmare din figura 2.3 să furnizeze cei doi operanzi pentru ALU, întârzierea produsă la multiplicarea capabilităţii de comandă a liniei /Sel, de 0,5 ns, este eliminată. În concluzie, timpul maxim de operare pentru ALU este de 2,8 ns. Şi în acest caz se poate constata viteza superioară circuitului propriu, faţă de unele realizări recente reprezentative marcate în tabelul 2.2.

Tab.2.2

Ad/Scăd. 32 biţi	propriu	[26]1992	[19]1999	[32]2000
Timp de propagare	2,8 ns	10ns	4,6ns	3,3ns

Viteza superioară a noului circuit de adunare/scădere este datorată, pe de o parte utilizării noului tip de sumator pe 32 de biţi, iar pe de alta, topologiei îmbunătăţite a circuitului de adunare/scădere.

2.2.2.5 Concluzii la paragraful 2.2.2

Calculul timpului total de producere a rezultatului se va face prin cumularea timpilor de propagare ai tuturor blocurilor aflate pe calea critică de propagare. Se obţine astfel:

$$t_{tot} = t_{ROMA} + t_{MUL.log} + t_{CSA+CLA.log} + t_{ALU} + t_{ROMC} + t_{MUL.alog} + t_{CSA+CLA.alog} =$$
$$= 3{,}8 \text{ ns} + 6{,}3 \text{ ns} + 3{,}2 \text{ ns} + 2{,}8 \text{ ns} + 3{,}8 \text{ ns} + 6{,}3 \text{ns} + 3{,}2 \text{ ns} = 29{,}4 \text{ ns} \quad (2.18)$$

Dacă întreaga structură din figura 2.3 ar funcţiona într-o manieră complet asincronă, deci dacă ar fi organizată într-un singur etaj, atunci frecvenţa de clock a sistemului nu va putea fi mai mare de 33 MHz. Din acest motiv s-a adoptat soluţia practicată în toate procesoarele aritmetice sau de semnal actuale, aceea de organizare a întregii structuri pe niveluri pipeline, aşa cum se vede în figura 2.14.

Structura prezentată în figura 2.14 reprezintă varianta optimă din punct de vedere al segmentării ei în scopul asigurării frecvenţei de clock mărite şi conţine 6 niveluri pipeline.

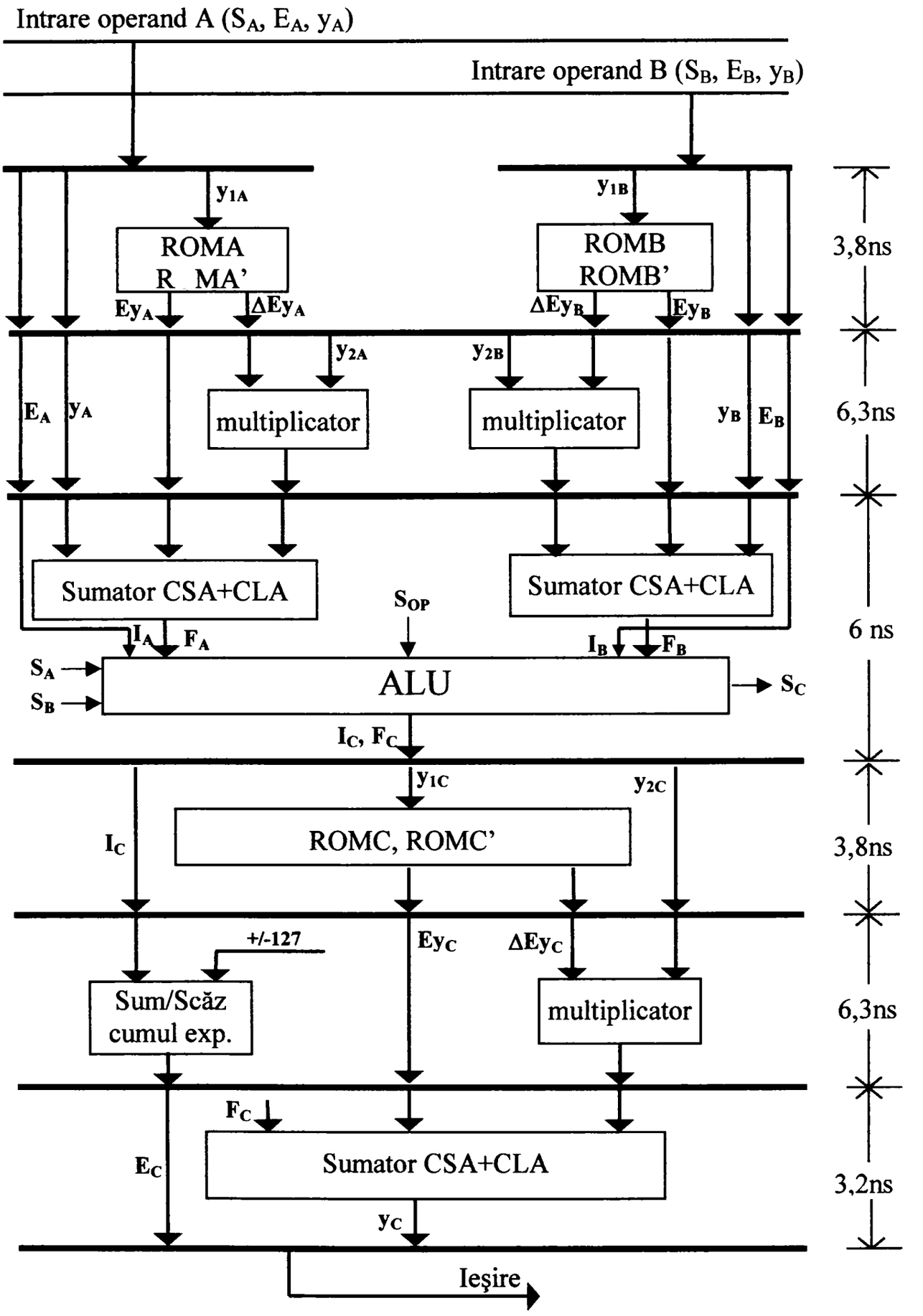


Fig. 2.14 Organizarea subunităţii logaritmice pe niveluri pipeline

Etajul critic al structurii este cel care conţine multiplicatorul (6,3 ns), deci sistemul va putea funcţiona cu o frecvenţă de clock maximă de aproximativ 150 MHz. Blocurile "Sumator CSA+CLA" şi ALU au fost comasate în acelaşi etaj, întrucât timpul lor cumulat de propagare este mai mic decât cel al multiplicatorului de 12×12 biţi. Rezultatul unei înmulţiri sau împărţiri va fi furnizat cu o latenţă de 6 cicli de 6,3 ns (t_L = 38 ns), dar rezultatul unor înmulţiri sau împărţiri succesive de operanzi se va produce la fiecare clock.

Rezultatele obţinute în ce priveşte viteza de calcul sunt similare cu cele prezentate în [50] *dar numărul de porţi logice utilizate este mai mic deoarece au fost eliminate două sumatoare de câte 8 biţi (pentru extragerea deplasamentului din exponenţii celor doi operanzi), precum şi două sumatoare pe 23 de biţi ce calculează termenii $y+E_y$ pentru cei doi operanzi şi s-a simplificat logica de comandă a ALU.*

Comparativ cu [67] la care o înmulţire în virgulă flotantă se face în 10 ns, în aceeaşi tehnologie, se poate spune că o operaţie singulară se efectuează de patru ori mai lent, dar în contextul utilizării procesorului în aplicaţii DSP sau grafice, transformări de coordonate etc., atunci când se efectuează un număr foarte mare de produse consecutive (când un produs se obţine la fiecare 6,3 ns), performanţele obţinute sunt superioare sistemelor de calcul în virgulă flotantă, dotate cu un multiplicator performant de tipul prezentat în [67]. *Avantajul provine din posibilitatea mult mai facilă de segmentare a subunităţii logaritmice pe niveluri pipeline şi deci de asigurare a unei frecvenţe de clock mai mari. În plus operaţia de împărţire, chiar şi efectuată singular, cu o latenţă de şase cicli, este mai rapidă decât una efectuată în virgulă flotantă [53], [54] cu cel puţin 50%.*

2.2.3 Implementarea algoritmului modificat de conversie de format FP-LNS şi LNS-FP prin metoda logaritmului mascat

2.2.3.1 Metoda logaritmului mascat

Structura prezentată în figura 2.14, deşi optimizată din punct de vedere al împărţirii pe niveluri pipeline, prezintă totuşi câteva *dezavantaje*:

- datorită funcţionării sincrone pe cele şase niveluri, se adaugă aproape 9 ns la timpul total de producere a rezultatului în manieră de funcţionare pur asincronă. Într-adevăr $t_L - t_{tot} = 38 - 29,4 = 9,2$ ns;

- avantajul de viteză adus prin propagarea pe verticală a transportului în aria de înmulţire organizată în arbore Wallace, este diminuat de timpul necesar pentru însumarea propriu-zisă a ultimelor două produse parţiale intermediare (când transportul se propagă pe orizontală), chiar dacă aceasta se face cu un sumator rapid;

- în nivelul al treilea al structurii, există două etape succesive de însumare propriu-zisă ce concură la obţinerea unui rezultat final unic, la care se pierde mult timp prin propagarea pe orizontală a transportului;

- avantajul de viteză adus prin concepţia sumatorului propriu, prezentat în paragraful 2.2.2.4, nu poate fi pe deplin valorificat datorită faptului că nivelul critic al structurii pipeline rămâne cel care conţine multiplicatorul;

- multiplicatorul de 12×12 biţi a fost proiectat fără a utiliza şi blocuri de compresori 4:2 pe lângă blocuri CSA de sumatoare complete 3:2, ca şi metodă eficientă de reducţie a numărului de produse parţiale (dar cu consum suplimentar de arie pe cip, deoarece un compresor 4:2 conţine două sumatoare complete 3:2); altfel, pentru structura din figura 2.14 nivelul al treilea devenea critic şi câştigul de viteză ar fi fost puţin semnificativ în raport cu aria suplimentară necesară.

Pentru eliminarea acestor dezavantaje am oferit o soluţie proprie care se bazează pe o metodă nouă de organizare a subunităţii logaritmice, pe care am denumit-o ***"metoda logaritmului mascat"****, [44]. La elaborarea acestei metode am avut în vedere ca prin aplicarea ei să se atingă şi următoarele scopuri:*

- noua structură să nu fie organizată pe mai mult de 6 niveluri pipeline,

- să nu crească semnificativ numărul de porţi logice utilizate pe fiecare nivel,

- să nu crească semnificativ capacitatea latch-urilor dintre nivelurile structurii pipeline,

- timpul de propagare pe nivelul critic al structurii să fie cât mai aproape de limita fizic realizabilă prin implementarea algoritmului prezentat şi anume aceea a timpului de acces la memoriile ROM integrate pe cip,

- timpii de propagare pe nivelurile pipeline să fie cât mai apropiaţi.

Prin această nouă abordare, termenii y_A, Ey_A, respectiv y_B, Ey_B care se însumează cu produsele $\Delta Ey_A \times y_{2A}$, respectiv $\Delta Ey_B \times y_{2B}$ (conform relaţiei 2.6), vor fi consideraţi ca şi cum ar fi produse parţiale iniţiale ale produselor menţionate. Aceşti termeni vor fi astfel incluşi în arborele Wallace de reducţie a numărului de produse parţiale, pe lângă cele 12 produse parţiale iniţiale ale fiecărui produs. În felul acesta au fost practic eliminate sumatoarele finale ale multiplicatoarelor, iar cei doi termeni rezultaţi în vârful inferior al arborelui, în secţiunile "sumă" şi "transport", vor conţine implicit şi valoarea produsului, distribuită între cele două secţiuni. Valorile propriu-zise ale celor două produse $\Delta Ey \times y_2$ ne mai regăsindu-se explicit în structură, devin astfel "mascate".

Problema care mai trebuie rezolvată este aceea a situaţiei în care în relaţia 2.6 termenul $\Delta Ey \times y_2$ este negativ şi ar fi necesară conversia în complement de doi a acestuia, adică a tuturor celor 12 produse parţiale. Acest lucru se întâmplă începând de la adresa 907 şi până la adresa 2047 a memoriilor ROMA şi ROMA', situaţie care corespunde sectorului coborâtor al graficului din figura 2.1.a. Generarea complementului de doi pentru fiecare produs parţial este însă foarte complexă întrucât produsele parţiale au ponderi diferite şi conduce la situaţia din figura 2.15.b. mult mai dificil de implementat decât în cazul când produsul este pozitiv, ca în figura 2.15.a. S-a considerat "p" o reprezentare generică pentru fiecare bit al produsului parţial de 12 biţi, indiferent de valoarea sa, iar "q" valoarea bitului "p", negată. Se constată că pentru a se putea implementa şi cazul

b este necesară extinderea la mult mai mulţi biţi a blocurilor ce reduc numărul de produse parţiale, iar selecţia de 1 sau 0 se face pentru un număr foarte mare de intrări în aceste blocuri.

...00000ppppppppppp
...0000ppppppppppp
...000ppppppppppp
...00ppppppppppp
...0ppppppppppp
...ppppppppppp
.....

Fig. 2.15.a
Porţiune din aria de înmulţire
în cazul $\Delta Ey \times y_2 > 0$

...11111qqqqqqqqqqq
...1111qqqqqqqqqqq1
...111qqqqqqqqqqq1
...11qqqqqqqqqqq1
...1qqqqqqqqqqq1
...1qqqqqqqqqq1
.....1

Fig.2.15.b
Porţiune din aria de înmulţire
în cazul $\Delta Ey \times y_2 < 0$

În figura 2.15.b bitul de valoare logică 1 de sub fiecare bit q de tip LSB este cel necesar pentru conversia în complement de doi a fiecărui produs parţial.

Am realizat evitarea acestui grav neajuns printr-un artificiu, care permite eliminarea completă a cazurilor în care produsul $\Delta Ey \times y_2$ trebuie scăzut. După cum se observă în figura 2.16, se poate scrie următoarea relaţie:

$$Ey_{(n)} - \Delta Ey_{(n)} \times y_2 = Ey_{(n+1)} + \Delta Ey_{(n)} \times (\bar{y}_2 + 1) = Ey_{(n+1)} + \Delta Ey_{(n)} \times \bar{y}_2 + \Delta Ey_{(n)} \qquad (2.19)$$

Implementarea acestei relaţii conduce la aranjarea produselor parţiale aşa cum se prezintă în figura 2.17. A fost folosită aceeaşi reprezentare generică, cu "q_y" pentru biţii "p_y" ai lui y_2 complementaţi, respectiv cu "p_d" pentru biţii lui ΔEy.

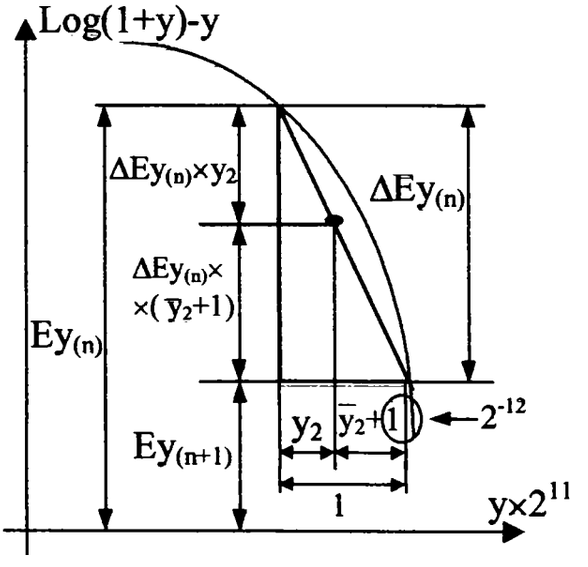


Fig. 2.16
Segment obţinut prin interpolare liniară între două valori memorate consecutive Ey, de pantă negativă

...0 0 0 0 0 $p_d p_d p_d p_d p_d p_d p_d p_d p_d p_d p_d p_d p_d$
...0 0 0 0 0 $q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y$
...0 0 0 0 $q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y$
...0 0 0 $q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y$
...0 0$q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y$
...0$q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y$
...$q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y q_y$
.....

Fig. 2.17

Porţiune din aria de înmulţire în cazul implementării relaţiei 2.19

Aşadar se poate obţine acelaşi rezultat al logaritmării dacă se implementează membrul drept al relaţiei 2.19. Începând de la locaţia de memorie corespunzătoare adresei 907 a lui ROMA, în loc să se memoreze valoarea $Ey_{(n)}$, se memorează valoarea $Ey_{(n+1)}$, adică cea care ar fi trebuit să se găsească la adresa următoare. În fiecare locaţie se va memora un bit suplimentar, denumit bit de control, de valoare 0 pentru adresele 0...906 şi de valoare 1 pentru adresele 907...2047. Dacă acest bit are valoarea 1 atunci generarea produselor parţiale se va face cu y_2 având biţii inversaţi şi se va mai adăuga un produs parţial de pondere egală cu a celui mai puţin semnificativ produs parţial, de valoare $\Delta Ey_{(n)}$. Dacă bitul de control are valoarea 0 atunci generarea produselor parţiale se va face cu y_2 neinversat iar biţii "p_d" din primul produs parţial din figura 2.17 vor fi toţi egali cu 0.

Arborele Wallace va avea astfel drept intrări 15 pseudo-produse parţiale iniţiale şi va furniza două cuvinte: "sumă" şi "transport", aşa cum s-a văzut în figura 2.5. Dacă în această fază s-ar efectua adunarea propriu-zisă a acestora s-ar obţine valoarea logaritmului mantisei fiecărui operand aplicat la intrarea celor două circuite de logaritmare cu funcţionare paralelă. În continuare, cele două numere fracţionare obţinute ar fi concatenate la exponenţii celor doi operanzi, pentru a fi obţinuţi logaritmii operanzilor. În final cei doi logaritmi s-ar aplica la ALU, pentru a fi adunaţi sau scăzuţi conform relaţiilor 2.10 şi 2.11. Totuşi şi în acest caz vor avea loc două adunări propriu-zise consecutive, ceea ce încetineşte producerea rezultatului.

Pentru evitarea acestei situaţii şi în acest caz se consideră cele două perechi de cuvinte "sumă" şi "transport" ca şi pseudo-produse parţiale astfel că ele vor fi introduse într-un nou bloc de reducţie. Acesta va furniza în final două cuvinte, "sumă" şi "transport" finale, ce vor putea fi adunate apoi cu un sumator rapid. Acest ultim bloc de reducţie a numărului de pseudo-produse parţiale va fi inclus în ALU întrucât asupra lui trebuie să acţioneze o logică de comandă care să permită implementarea atât a operaţiei de adunare cât şi a operaţiei de scădere.

În felul acesta au fost practic eliminate sumatoarele (din blocurile Sumator CSA+CLA, fig. 2.14) care furnizează valorile explicite ale logaritmilor mantiselor celor doi operanzi. Aceste valori devin astfel mascate, ele sunt conţinute implicit în cele 4 pseudo-produse parţiale şi concură la obţinerea rezultatului final. Din acest motiv, această metodă de operare a fost denumită "metoda logaritmului mascat".

Implementarea acestei metode conduce la generarea unui arbore Wallace care are la bază 30 de pseudo-produse parţiale şi al cărui vârf se găseşte în ALU. Pentru construcţia arborelui au fost utilizate drept blocuri de reducţie a numărului de pseudo-produse parţiale, blocuri de compresori 4:2 (v. paragraful 2.2.3.2) şi blocuri CSA de sumatoare complete 3:2, în aşa fel încât să se obţină cea mai eficientă structură, în care să nu existe pseudo-produse parţiale intermediare neoperate pe nici un etaj al arborelui. Structura obţinută este prezentată în figura 2.18, iar alinierea diverselor pseudo-produse parţiale iniţiale şi intermediare de la intrările şi ieşirile blocurilor de reducţie, în figurile 2.19 şi 2.20.

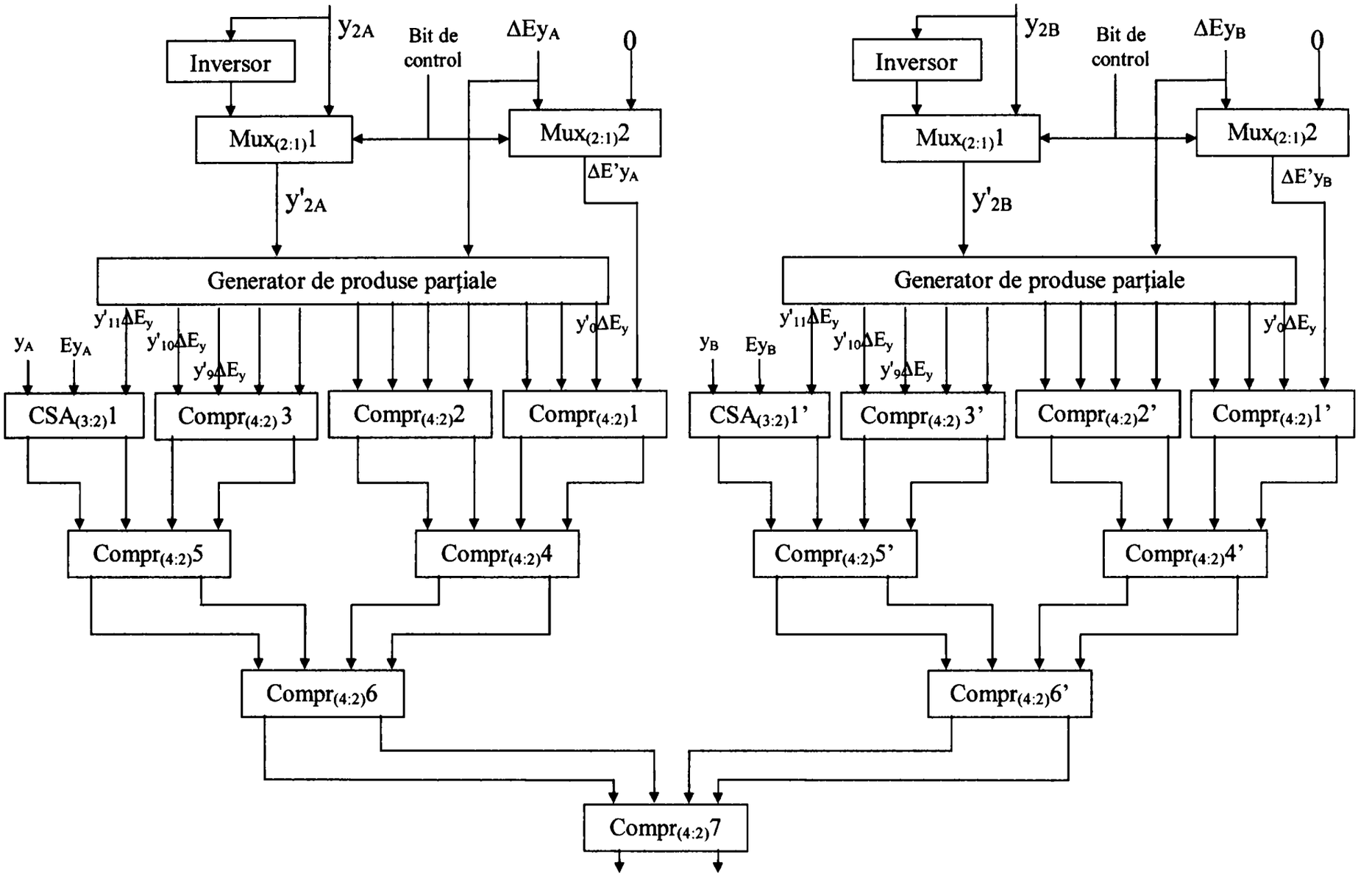


Fig.2.18 Structura arborelui Wallace de reducţie a numărului de pseudo-produse parţiale

z	e	c	i		0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3
u	n	i	t		1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
I		c	p	r	s	1																						x	x	x	x	x	x	x	x	x	x	x	x
I		c	p	r	s	1				I	-	I	n	t	r	a	r	e										x	x	x	x	x	x	x	x	x	x	x	x
I		c	p	r	s	1				e	-	I	e	ş	i	r	e										x	x	x	x	x	x	x	x	x	x	x	x	
I		c	p	r	s	1																				x	x	x	x	x	x	x	x	x	x	x	x		
I		c	p	r	s	2																			x	x	x	x	x	x	x	x	x	x	x	x			
I		c	p	r	s	2																		x	x	x	x	x	x	x	x	x	x	x	x				
I		c	p	r	s	2																	x	x	x	x	x	x	x	x	x	x	x						
I		c	p	r	s	2																x	x	x	x	x	x	x	x	x	x	x							
I		c	p	r	s	3															x	x	x	x	x	x	x	x	x	x	x								
I		c	p	r	s	3														x	x	x	x	x	x	x	x	x	x	x									
I		c	p	r	s	3													x	x	x	x	x	x	x	x	x	x	x										
I		c	p	r	s	3												x	x	x	x	x	x	x	x	x	x	x	x										
I	c	s	a														x	x	x	x	x	x	x	x	x	x	x	x											
I	c	s	a		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x											
I	c	s	a		0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x											
e		c	p	r	s	1																				x	x	x	x	x	x	x	x	x	x	x	x	x	x
e		c	p	r	s	1																				x	x	x	x	x	x	x	x	x	x	x	x	x	
e		c	p	r	s	2																x	x	x	x	x	x	x	x	x	x	x	x	x	x				
e		c	p	r	s	2																x	x	x	x	x	x	x	x	x	x	x	x	0					
e		c	p	r	s	3												x	x	x	x	x	x	x	x	x	x	x	x	x	x								
e		c	p	r	s	3												x	x	x	x	x	x	x	x	x	x	x	x	x	0								
e	c	s	a		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x											
e	c	s	a		0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0												

Fig.2.19 Alinierea pseudo-produselor parţiale în primul etaj al arborelui Wallace

z	e	c	i		0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3
u	n	i	t		1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
I		c	p	r	s	4																				x	x	x	x	x	x	x	x	x	x	x	x	x	
I		c	p	r	s	4			I	-	I	n	t	r	a	r	e										x	x	x	x	x	x	x	x	x	x	x	x	
I		c	p	r	s	4			e	-	I	e	ş	i	r	e						x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
I		c	p	r	s	4																x	x	x	x	x	x	x	x	x	x	x	x	0				
I		c	p	r	s	5												x	x	x	x	x	x	x	x	x	x	x	x	x	x	x						
I		c	p	r	s	5												x	x	x	x	x	x	x	x	x	x	x	x	x	0							
I	c	p	5		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x										
I	c	p	5		0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0											
e	c	p	4																			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
e	c	p	4																		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
e	c	p	5		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x							
e	c	p	5		0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0								
I	c	p	6																			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
I	c	p	6																		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
I	c	p	6		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x							
I	c	p	6		0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0								
e	c	p	6		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
e	c	p	6		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			

Fig. 2.20 Alinierea pseudo-produselor parţiale intermediare în al doilea şi al treilea etaj al arborelui Wallace

În figurile 2.19 şi 2.20 a fost folosită notaţia generică "x" pentru a desemna biţii pseudo-produselor parţiale, indiferent de valoarea lor. Blocul CSA grupează termenii y şi ΔEy împreună cu cel mai semnificativ produs parţial al produsului $y_2 \times \Delta Ey$, în scopul evitării creşterii exagerate a dimensiunii acestui bloc (şi deci şi a blocurilor conectate în aval de acesta) şi a magistralelor de legătură.

Un bloc de compresori 4:2 va conţine în realitate, la anumite ranguri, nu doar compresori 4:2, ci şi sumatoare complete 3:2, respectiv sumatoare incomplete 2:2 deoarece, aşa cum se observă în figurile 2.19 şi 2.20, nu sunt folosite întotdeauna toate cele 4 intrări de acelaşi rang. Considerarea acestui fapt va conduce la o economie semnificativă a numărului de porţi logice utilizate.

Pentru circuitul de antilogaritmare, procedura aplicată este identică, cuvintele "sumă" şi "transport" finale fiind obţinute după doar 3 etaje de compresori. Ele sunt apoi însumate cu un sumator rapid pentru obţinerea mantisei rezultatului final de la ieşirea subunităţii logaritmice. Şi aici vor fi luate măsuri pentru evitarea operaţiei de scădere a termenului $\Delta Ey \times y_2$, dar se ţine cont şi de observaţiile făcute în paragraful 2.2.2.3. Acestea sunt legate de faptul că în relaţia 2.9 şi termenul Ey trebuie scăzut, iar produsul $\Delta Ey \times y_2$ se scade pe porţiunea crescătoare a curbei din figura 2.1.b şi se adună pe porţiunea descrescătoare. Deoarece relaţia 2.19 nu mai poate fi folosită, suma celor doi termeni negativi din relaţia 2.9 se scrie astfel:

$$-Ey_{(n)}-\Delta Ey_{(n)}\times y_2 = -(Ey_{(n)}+\Delta Ey_{(n)}\times y_2) = -[Ey_{(n)}+\Delta Ey_{(n)}-\Delta Ey_{(n)}\times(\bar{y}_2+1)] =$$
$$= -(Ey_{(n)}+\Delta Ey_{(n)})+\Delta Ey_{(n)}\times\bar{y}_2+\Delta Ey_{(n)} = -Ey_{(n+1)}+\Delta Ey_{(n)}\times\bar{y}_2+\Delta Ey_{(n)} \qquad (2.20)$$

Aşadar în locaţiile memoriei ROMC de la adresa 0 la adresa 1082 se va memora complementul de doi al cantităţilor $Ey_{(n+1)}$, ce ar fi trebuit să se găsească la adresa următoare, precum şi valoarea 1 pentru bitul de control; de la adresa 1083 la adresa 2047 (când $\Delta Ey \times y_2$ este pozitiv) se va memora direct complementul de doi al valorilor $Ey_{(n)}$, precum şi valoarea 0 pentru bitul de control.

2.2.3.2 Prezentarea noului tip de compresor 4:2 folosit în proiectarea subunităţii logaritmice

Compresorul 4:2 a fost introdus în literatura de specialitate de către Weinberger în 1981. Prima implementare pe cip ce făcea apel la acest circuit apare însă abia în 1991, prin multiplicatorul 54×54 de biţi conceput de J. Mori [67], posibil de realizat în urma creşterii densităţii de integrare. De atunci au apărut şi alte variante îmbunătăţite de compresor 4:2, care vor fi comparate la finele paragrafului cu varianta proprie a acestui circuit, denumită compresor 4:2 L.

Denumirea generalizată de "compresor 4:2" nu reflectă însă întocmai caracteristicile circuitului deoarece suma celor 4 biţi de intrare nu poate fi reprezentată în toate cazurile posibile, cu ajutorul celor doi biţi de ieşire. Circuitul furnizează în plus un transport pe o linie suplimentară de ieşire şi are, în

consecinţă, nevoie de o linie pentru introducerea transportului de intrare. Există aşadar o cale de propagare pe verticală, "4:2" şi o cale de propagare pe orizontală "1:1". Totuşi propagarea pe orizontală a transportului este practic limitată la un singur bit, întrucât transportul de ieşire este generat astfel încât să nu depindă de transportul de intrare. Astfel, la scara întregului multiplicator, utilizarea acestui circuit permite deplasarea căii critice de propagare din centrul ariei de înmulţire (unde este gradul maxim de suprapunere a produselor parţiale) spre capătul din stânga (unde acest grad de suprapunere devine din ce în ce mai mic).

În plus, combinaţia dintre blocuri CSA de sumatoare complete 3:2 şi blocuri de compresori 4:2, permite construcţia eficientă a arborelui Wallace, prin eliminarea cazurilor în care una sau două produse parţiale iniţiale sau intermediare rămase negrupate traversează un etaj al arborelui fără a fi operate [97]. Numărul de etaje al arborelui Wallace devine astfel mai mic, respectiv se scurtează calea critică de propagare prin acesta.

Principial, un compresor 4:2 se obţine prin interconectarea a două sumatoare complete 3:2, a căror structură a fost prezentată în figura 1.1 şi care prezintă un timp de propagare echivalent a 4 porţi SAU-exclusiv cascadate.

O îmbunătăţire majoră a structurii compresorului 4:2 a fost adusă de D. Villeger în [106] şi [72] prin exploatarea intrărilor şi ieşirilor "rapide" ale sumatoarelor complete 3:2, în aşa fel încât timpul maxim de propagare devine echivalentul a doar 3 porţi SAU-exclusiv cascadate. Această structură optimizată este prezentată în figura 2.21.

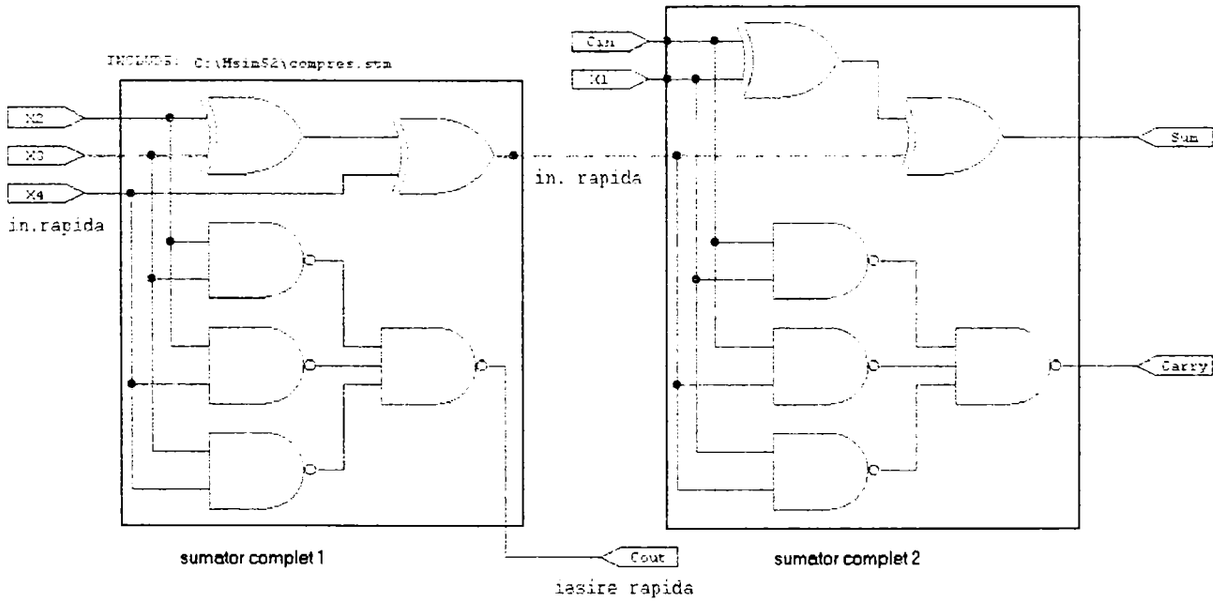


Fig.2.21 Structura unui compresor 4:2 optimizat [72]

Pentru acest compresor, cei 4 biţi de intrare sunt x1, x2, x3 şi x4, iar cei 2 biţi de ieşire sunt S_{um} şi C_{arry}. Transportul de intrare este C_{in} iar transportul de ieşire este C_{out}. Biţii C_{arry} şi C_{out} au pondere dublă faţă de bitul S_{um}. Bitul Sum este obţinut prin implementarea funcţiei SAU-exclusiv între cele 5 intrări ale circuitului. În presupunerea că timpul de propagare a transportului printr-o poartă

SAU-exclusiv este de două ori mai mare decât printr-o poartă ŞI-NU, se poate observa din figura 2.21 că, indiferent de calea de propagare a semnalelor de intrare, inclusiv a lui C_{in} generat de compresorul din amonte, timpul maxim de propagare nu depăşeşte echivalentul a şase porţi logice simple sau trei porţi SAU-exclusiv.

Se mai observă că ieşirea C_{out} nu depinde de intrarea C_{in}, deci propagarea transportului pe orizontală în blocul de compresori din figura 2.22 este limitată la un singur bit (o poziţie) spre stânga. Timpul de propagare pe verticală al întregului bloc este de fapt timpul de propagare pe verticală printr-un singur compresor. Aceasta înseamnă evident că înjumătăţirea numărului de produse parţiale se face într-un timp egal cu timpul de propagare pe calea critică prin compresorul 4:2. Din acest motiv, propagarea rapidă a semnalelor prin acest circuit este esenţială şi determină practic viteza de propagare prin întreg arborele Wallace.

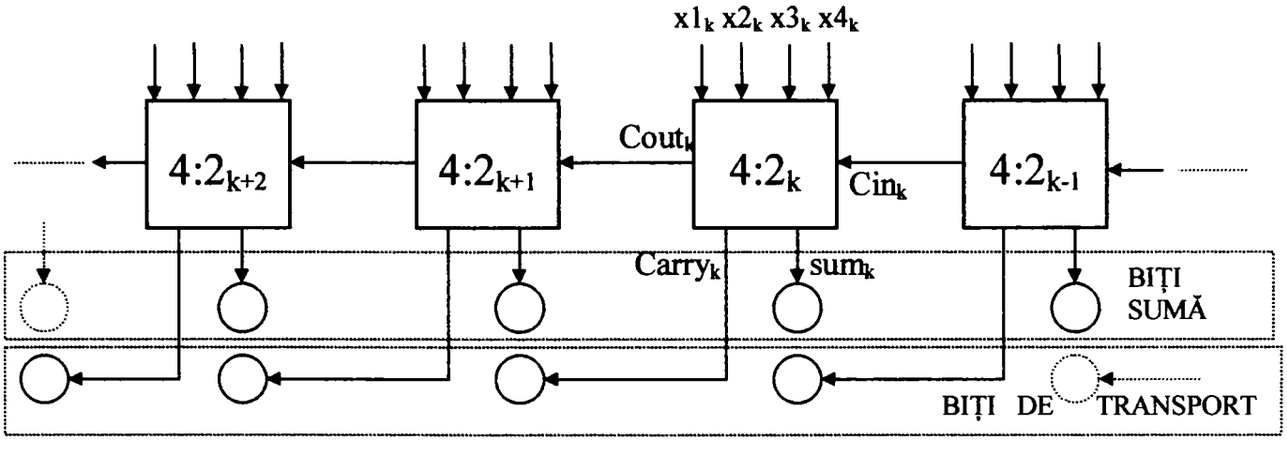


Fig. 2.22 Secţiune printr-un bloc de compresori 4:2

Compresorul 4:2 optimizat descris în lucrarea [72] şi prezentat în figura 2.21 *manifestă însă o serie de limitări care afectează viteza de propagare a transportului prin el:*

- porţile ŞI-NU cu trei intrări, în tehnologii submicronice avansate ce necesită tensiuni de alimentare mai mici, nu mai pot fi implementate sau se implementează prin porţi ŞI-NU cu două intrări, prin adăugarea încă a unui nivel logic. În acest fel calea critică de propagare prin circuit devine cea care include ieşirea rapidă C_{out} a compresorului actual şi intrarea C_{in} şi ieşirea C_{carry} a compresorului din aval. Această cale devine deci mai lentă decât calea de propagare pentru "x1" şi "x2" (fig. 2.21) prin trei porţi SAU-exclusiv succesive; chiar şi în tehnologii care permit integrarea porţii ŞI-NU cu trei intrări, timpul de propagare prin aceasta este mai mare decât printr-o poartă cu două intrări.

- utilizarea unor porţi SAU-exclusiv neconvenţionale [67], *mai rapide decât cele clasice, având $t_{P(SAU\text{-}excl)} < 2 \times t_{P(ŞI\text{-}NU)}$ nu va aduce avantajele scontate din cauză că propagarea prin trei porţi SAU-exclusiv cascadate nu mai reprezintă o cale critică în compresor.*

- există multe cazuri în care încărcarea capacitivă a unei ieşiri logice este considerată excesivă, atunci când ea apare pe o cale critică de propagare. Există

astfel în structura compresorului cazuri în care o ieşire comandă câte trei intrări, adică minim şase grile MOS. La capacitatea intrinsecă a acestora se adaugă şi capacitatea parazită a celor trei linii de semnal.

Eliminarea acestor dezavantaje s-a făcut prin conceperea unei variante proprii de compresor 4:2, [38], care aşa cum s-a dovedit prin simularea pe layout, este mai rapidă cu 10 % decât varianta din [72]. *Ţinând cont că în subunitatea logaritmică există 7 etaje de compresori 4:2 (4 etaje în circuitul de logaritmare şi ALU şi 3 în circuitul de antilogaritmare), aceasta duce în ansamblu la o economie de 0,7 ns (corespunzător implementării în tehnologie CMOS 0,5 microni).*

În conceperea variantei proprii am plecat de la ideea găsirii unei definiţii generale pentru compresorul 4:2, ale cărui intrări şi ieşiri sunt deja definite. Astfel acest circuit trebuie să asigure următoarele funcţii:

a) ieşirea S_{UM} să fie generată de relaţia 2.21:

$$S_{um} = x1 \oplus x2 \oplus x3 \oplus x4 \oplus C_{in} \qquad (2.21)$$

b) Cele două linii pe care apar transporturile de ieşire C_{arry} şi C_{out} să fie de rang imediat superior lui S_{um} (să aibă pondere dublă), astfel încât valoarea maximă a sumei celor 5 biţi de intrare, egală cu 5, să poată fi reprezentată. Astfel va fi îndeplinită relaţia 2.22:

$$x1+x2+x3+x4+C_{in} = C_{arry} + C_{out} + S_{um} \qquad (2.22)$$

Într-adevăr, atunci când toţi biţii de intrare sunt 1 se obţine $5=10_2+10_2+1$.

c) Una din liniile de transport de ieşire să poată fi generată de funcţii logice simple, care să includă un număr minim de niveluri logice şi să nu depindă de transportul de intrare C_{in}. Ea se denumeşte ieşire rapidă şi va asigura transportul de intrare pentru compresorul de rang imediat superior. În felul acesta propagarea pe orizontală a transportului se va limita la un singur bit şi nu va influenţa viteza de generare a rezultatului la ieşirea unui bloc de compresori 4:2 cascadaţi.

d) Toate combinaţiile posibile de biţi de intrare pentru care suma lor este 0 sau 1, trebuie să conducă la $C_{arry} = C_{out} = 0$.

e) Toate combinaţiile posibile de biţi de intrare pentru care suma lor este 2 sau 3, trebuie să conducă fie la $C_{arry} = 1$ şi $C_{out} = 0$, fie la $C_{arry} = 0$ şi $C_{out} = 1$. Cu alte cuvinte, toate aceste combinaţii posibile care produc fie $C_{arry} = 1$, fie $C_{out} = 1$ trebuie să se găsească în două mulţimi disjuncte.

f) Toate combinaţiile posibile de biţi de intrare pentru care suma lor este 4 sau 5, trebuie să conducă la $C_{arry} = C_{out} = 1$.

Având în vedere toate aceste funcţii, văzute ca şi cerinţe obligatorii, dar şi dezavantajele compresorului 4:2 de referinţă, a rezultat structura eficientă a compresorului 4:2 din figura 2.23, denumit compresor 4:2 L.

Datorită numărului mare de intrări, pentru generarea bitului S_{um} conform relaţiei 2.20, am adoptat o structură arborescentă de porţi SAU-exclusiv, care grupează intrările x1, x2, x3, x4 şi ulterior C_{in}. În continuare, pe baza grupării primilor patru biţi de intrare, a fost implementat C_{out} prin funcţia logică simplă:

$$C_{out} = x1\ x2 + x3\ x4 \qquad (2.23)$$

Se obţine astfel generarea ieşirii rapide "C_{out}" prin parcurgerea a doar două nivele logice, realizată exclusiv cu porţi intrinseci cu doar două intrări, aplicând relaţiile De Morgan. Nici una din liniile de intrare nu mai este astfel încărcată decât cu maxim două intrări logice. Se observă că relaţia 2.23 îndeplineşte cerinţele d) şi f). Pentru obţinerea ieşirii "C_{arry}" a fost implementată relaţia 2.24:

$$C_{arry} = (x1 \oplus x2)(x3 \oplus x4) + C_{in}(x1 \oplus x2) + C_{in}(x3 \oplus x4) + x1x2x3x4 \qquad (2.24)$$

Primii trei termeni ai sumei logice asigură îndeplinirea cerinţei e) relativ la C_{out}, iar termenul al patrulea asigură îndeplinirea cerinţei f) din lista enunţată. De asemenea se observă că cerinţa d) este îndeplinită de către toţi cei patru termeni.

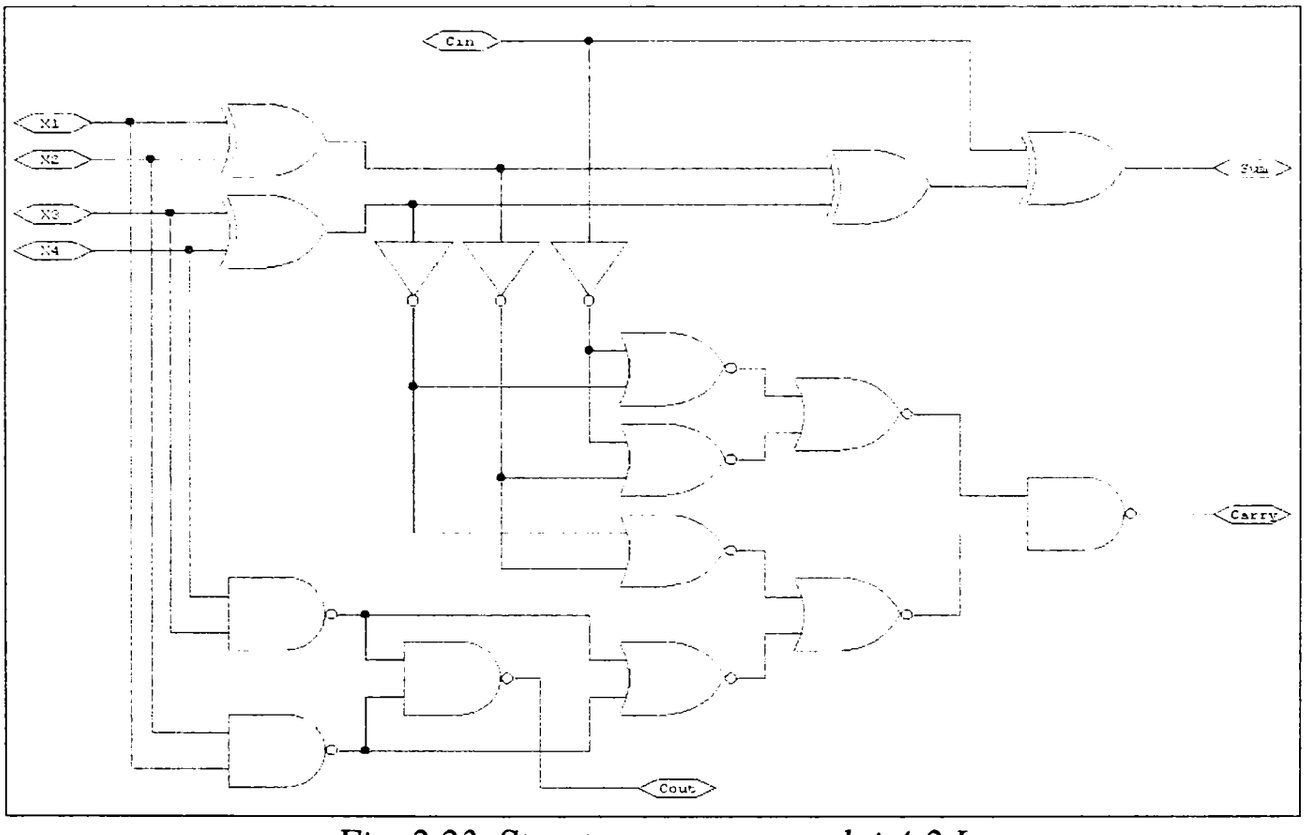


Fig. 2.23 Structura compresorului 4:2 L

Utilizarea celor trei inversoare din figură a permis eliminarea completă a dezavantajului încărcării suplimentare capacitive a anumitor ieşiri logice (o ieşire nu comandă mai mult de două intrări) şi în acelaşi timp a permis aplicarea relaţiilor De Morgan, pentru a utiliza exclusiv porţi intrinseci în tehnologie CMOS. O comparaţie de viteză între varianta proprie de compresor 4:2 şi câteva variante recente, apărute în literatura de specialitate, pentru o tehnologie CMOS de 0,5 microni, este ilustrată în tabelul 2.3:

Tab. 2.3

Tip compresor	[67]1991	[70]1995	[72]1996	[101]1999	Compres. 4:2 L
Timp de propagare	1,4 ns	1,4 ns	1,3 ns	1,2 ns	1,1 ns

Aşa cum se observă, compresorul 4:2 L este superior în viteză celorlalte variante.

2.2.3.3 Concepţia blocului ALU în cazul aplicării metodei logaritmului mascat

Datorită faptului că în blocul ALU trebuie implementată pe lângă operaţia de adunare şi operaţia de scădere, ce implică realizarea conversiei în complement de doi a scăzătorului, în mod obligatoriu blocul de compresori 4:2 L, Compr$_{(4:2)}$7 din figura 2.18 trebuie inclus în ALU. El va efectua reducţia celor 4 pseudo-produse parţiale de intrare după ce logica de comandă a circuitului sumator/scăzător a intervenit asupra scăzătorului.

Scăzătorul este reprezentat de două pseudo-produse parţiale, a căror însumare nu se mai execută, tocmai în scopul creşterii vitezei de propagare a transportului prin subunitatea logaritmică. Aceasta înseamnă că ambii termeni trebuie convertiţi în cod complement de doi. Evitarea reconversiei din cod complement de doi în cod semn-mărime a rezultatului ce va fi obţinut, în cazul când acesta este negativ, se face după aceeaşi metodă ca şi în paragraful 2.2.2.4, ce uzează de topologia de concepţie proprie a circuitului sumator/scăzător.

Se notează cu A_1, A_2, respectiv B_1, B_2 cele 4 pseudo-produse parţiale, caz în care $A=A_1+A_2$ reprezintă descăzutul, iar $B=B_1+B_2$ reprezintă scăzătorul, în cazul în care se efectuează o operaţie de scădere. Considerând convenţia de reprezentare a lui B negat (data obţinută prin inversarea tuturor biţilor lui B) ca fiind "/B", se pot scrie cei doi termeni care se calculează simultan în circuitul sumator/scăzător:

$$A - B = A - B_1 - B_2 = A + /B_1 + /B_2 + 2 \qquad (2.25)$$

$$/(B - A) = A + /B_1 + /B_2 + 1 \qquad (2.26)$$

Relaţia 2.26 se verifică în felul următor:

- în ecuaţia 2.27 de mai jos,

$$A - B = -(B - A) = /(B - A) + 1 \qquad (2.27)$$

se înlocuieşte termenul "/(B - A)" cu valoarea dată de relaţia 2.26.

- se obţine într-adevăr $A - B = (A + /B_1 + /B_2 + 1) + 1 = A + /B_1 + /B_2 + 2$, adică exact valoarea corectă din relaţia 2.25.

Principiul circuitului sumator/scăzător este următorul:

- dacă A-B rezultă pozitiv, el este selectat la ieşire, iar bitul de semn al rezultatului va fi 0;
- dacă A-B rezultă negativ înseamnă că B-A este pozitiv şi la ieşire se selectează termenul /(B-A) negat, adică B-A. Bitul de semn al rezultatului va fi 1.

Pentru a menţine similitudinea cu structura din figura 2.9, relaţiile 2.25 şi 2.26 se rescriu astfel:

$$A - B = (A_1 + A_2 + /B_1 + /B_2 + 1) + 1 \qquad (2.28)$$

$$/(B - A) = (A_1 + A_2 + /B_1 + /B_2 + 1) + 0 \qquad (2.29)$$

Cu alte cuvinte se menţine situaţia de transport iniţial 1, respectiv transport iniţial 0 la cele două sumatoare care lucrează în paralel, condiţionat de asigurarea unui transport de intrare egal cu 1 la blocul de compresori 4:2 L, în cazul efectuării

operaţiei de scădere. Evident acest transport iniţial va fi 0 în cazul efectuării operaţiei de adunare. Transportul de intrare se aplică pe intrarea C_{in} nefolosită a compresorului 4:2 L de rangul cel mai mic.

În continuare, dimensiunea blocului $Compr_{(4:2)}7$, inclus în ALU, se va suplimenta cu 9 biţi la stânga, pentru concatenarea exponenţilor pozitivi (cu deplasamentul de 127 inclus, conform algoritmului modificat) ce reprezintă partea întreagă a logaritmului cu deplasament a celor doi operanzi de intrare. Părţile fracţionare ale logaritmilor celor doi operanzi sunt "mascate", fiind distribuite între elementele A_1 şi A_2, respectiv B_1 şi B_2 ale celor două perechi de pseudo-produse parţiale. Concatenarea exponenţilor se va face la termenii A_1 şi B_1, obţinându-se pseudo-produsele parţiale finale $\underline{A_1}$ şi $\underline{B_1}$, în timp ce cele 9 poziţii de biţi de parte întreagă corespunzătoare la A_2 şi B_2 de pondere fracţionară, se vor completa cu zerouri. Se vor obţine astfel celelalte două pseudo-produsele parţiale finale $\underline{A_2}$ şi $\underline{B_2}$.

Schema bloc a noului sumator/scăzător, conceput în scopul implementării complete a metodei logaritmului mascat, este prezentată în figura 2.24. Linia S_{OP} are valoarea logică 0 în cazul adunării şi 1 în cazul scăderii.

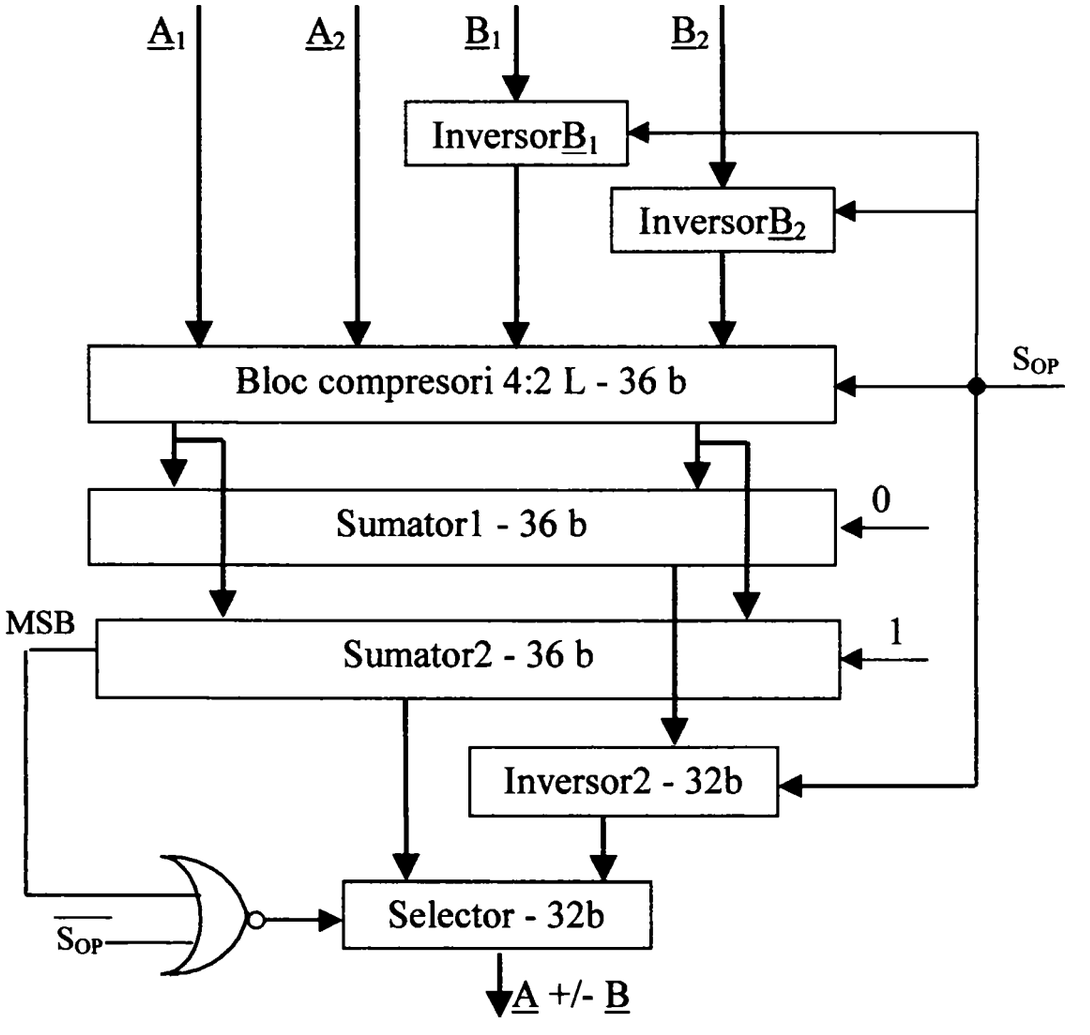


Fig. 2.24 Schema bloc a noului circuit sumator/scăzător

Blocurile InversorB_1, InversorB_2 şi Inversor2 vor fi transparente atunci când se efectuează o operaţie de adunare şi vor inversa biţii datelor de la intrări atunci când se efectuează o operaţie de scădere. Blocul Selector va selecta rezultatul de la Sumator1 neinversat, în cazul adunării, iar în cazul scăderii va selecta rezultatul de la Sumator2 sau de la Sumator1 inversat, în funcţie de valoarea bitului MSB a blocului Sumator2. Detaliile de funcţionare sunt similare cu cele prezentate în paragraful 2.2.2.4.

Cele două sumatoare funcţionează pe 36 de biţi, fiind implementate prin acelaşi tip de circuit performant de concepţie proprie, care a fost prezentat în figura 2.12. Diferenţa constă în aceea că în structura de sumator cu selecţia transportului mai apare o secţiune suplimentară CLA pe 4 biţi, în poziţia cea mai semnificativă. Cei mai puţin semnificativi 4 biţi de la ieşirile celor două sumatoare vor fi pierduţi, astfel că la ieşirea circuitului se regăseşte dimensiunea formatului simplă precizie plus un bit în poziţia MSB, care evită apariţia supradepăşirii datorată cumulării deplasamentelor în cazul înmulţirii.

Justificarea dimensiunii de 36 de biţi a sumatoarelor şi a blocului de compresori va fi făcută în paragraful 2.2.4.

Schemele detaliate ale întregului circuit, cât şi ale arborelui Wallace din amonte sunt prezentate în Anexa 3.

Timpul de propagare maxim prin arborele Wallace ce conţine trei etaje de compresori 4:2 L este 3 × 1,1 ns, la care se mai adaugă 0,7 ns pentru generarea produselor parţiale ceea ce conduce la un total de 4 ns, pentru tehnologie CMOS 0,5 microni.

Timpul de propagare maxim pentru circuitul sumator/scăzător în cazul în care sumatoarele sunt de 32 de biţi, este obţinut prin cumularea valorii din tabelul 2.2 din paragraful 2.2.2.4, adică 2,8 ns, cu timpul de propagare prin blocul de compresori 4:2 L, de 1,1 ns, rezultând 3,9 ns. În cazul de faţă se mai adaugă maxim 0,2 ns necesare pentru multiplicarea cu 2 a capabilităţii de comandă a ultimei linii de selecţie din sumatorul de 36 de biţi. Evaluarea cazului celui mai defavorabil, pentru a se efectua simularea, este dificilă datorită combinaţiei dintre cele două tipuri de circuite, care pentru o valoare precizată a intrărilor pot să nu se găsească simultan în cazul cel mai defavorabil. De aceea, prin simulare se pot obţine timpi mai mici de 4 ns.

Pentru sumatorul final pe 27 de biţi de la capătul arborelui Wallace din circuitul de antilogaritmare, de acelaşi tip cu sumatorul din figura 2.11, a fost găsit un timp de propagare în cazul cel mai defavorabil de 3,6 ns.

2.2.3.4 Concluzii la paragraful 2.2.3

Utilizarea metodei logaritmului mascat în implementarea subunităţii logaritmice de calcul a condus la obţinerea următorului timp total de producere a rezultatului (au fost luaţi în considerare timpii de propagare maximi ai tuturor blocurilor aflate pe calea critică de propagare):

$$t_{tot} = t_{ROMA} + t_{arbore.log} + t_{ALU} + t_{ROMC} + t_{arbore.alog} + t_{SumFinal} =$$
$$= 3{,}8 \text{ ns} + 4 \text{ ns} + 4\text{ns} + 3{,}8\text{ns} + 4\text{ns} + 3{,}6 \text{ ns} = 23{,}2 \text{ ns} \qquad (2.30)$$

Se observă că în acest caz, spre deosebire de rezultatele obţinute în paragraful 2.2.2 care erau comparabile în viteză cu cea mai performantă implementare descrisă în literatura de specialitate de către F. Lai [50], *timpii de propagare prin blocurile de pe calea critică de propagare sunt considerabil mai mici şi sunt mult mai echilibraţi. Acest lucru permite o organizare foarte eficientă într-o structură pipeline cu 6 niveluri, aşa cum se observă în figura 2.25. Aşa cum se va dovedi prin simulare, există rezerve de 0,2÷0,4 ns relativ la cele 4 ns din fiecare nivel, care compensează timpul de încărcare a latch-urilor pipeline.*

Frecvenţa de clock a sistemului ajunge astfel la aproape 250 MHz, ceea ce reprezintă o creştere de viteză cu 60% faţă de referinţa menţionată. Toate avantajele legate de viteza de calcul, pe care le aduce un astfel de procesor şi evidenţiate în paragraful 2.2.2, se multiplică cu un factor de 1,6.

În plus, se pot remarca şi alte caracteristici notabile:

- numărul de porţi logice utilizate în întreaga structură a scăzut semnificativ, datorită eliminării a şase sumatoare rapide de minim 18 biţi, cu preţul suplimentării cu 4 biţi a trei sumatoare rapide;

- timpul de propagare prin nivelul critic al structurii pipeline a coborât practic la limita fizică minimă, dată de timpul de acces la memorii;

- capacitatea latch-urilor dintre nivelurile pipeline a scăzut cu aproximativ 30 %, fapt datorat structurii arborescente a subunităţii logaritmice;

- o economie suplimentară de suprafaţă, de aproximativ 25%, s-a realizat la implementarea celor trei înmulţiri de tip $\Delta Ey \times y_2$, posibilitate demonstrată în urma unui calcul al erorilor în cazul efectuării produsului menţionat. Acest lucru va fi prezentat în paragraful 2.2.4.

2.2.4 Studiul erorilor produse la generarea logaritmului şi antilogaritmului binar

În urma analizei cu programul Matlab, a erorilor introduse prin utilizarea algoritmului prezentat în paragraful 2.2.1, s-a confirmat valoarea de 3×10^{-7} menţionată în [50], ca valoare maximă a erorii de conversie. Aceasta eroare este totuşi de circa 2,5 ori mai mare decât eroarea în formatul simplă precizie, care este de $1{,}19 \times 10^{-7}$. *Pentru minimizarea în continuare a erorii de conversie am propus o soluţie proprie, care va adăuga valori de corecţie pe anumite intervale de adresă ale memoriilor de tip ROMA şi ROMC, în locaţiile corespunzătoare de memorie. Ţinând cont că eroarea în simplă precizie, adică valoarea celui mai puţin semnificativ bit pe care o furnizează o ieşire oarecare a memoriei ROMA sau ROMC, este de $1{,}2 \times 10^{-7}$, înseamnă că la valorile calculate ale lui Ey pot fi adăugate corecţii de ordinul a un LSB, după care sunt memorate direct (ROMA), respectiv se transformă în complement de doi şi apoi se memorează (ROMC).*

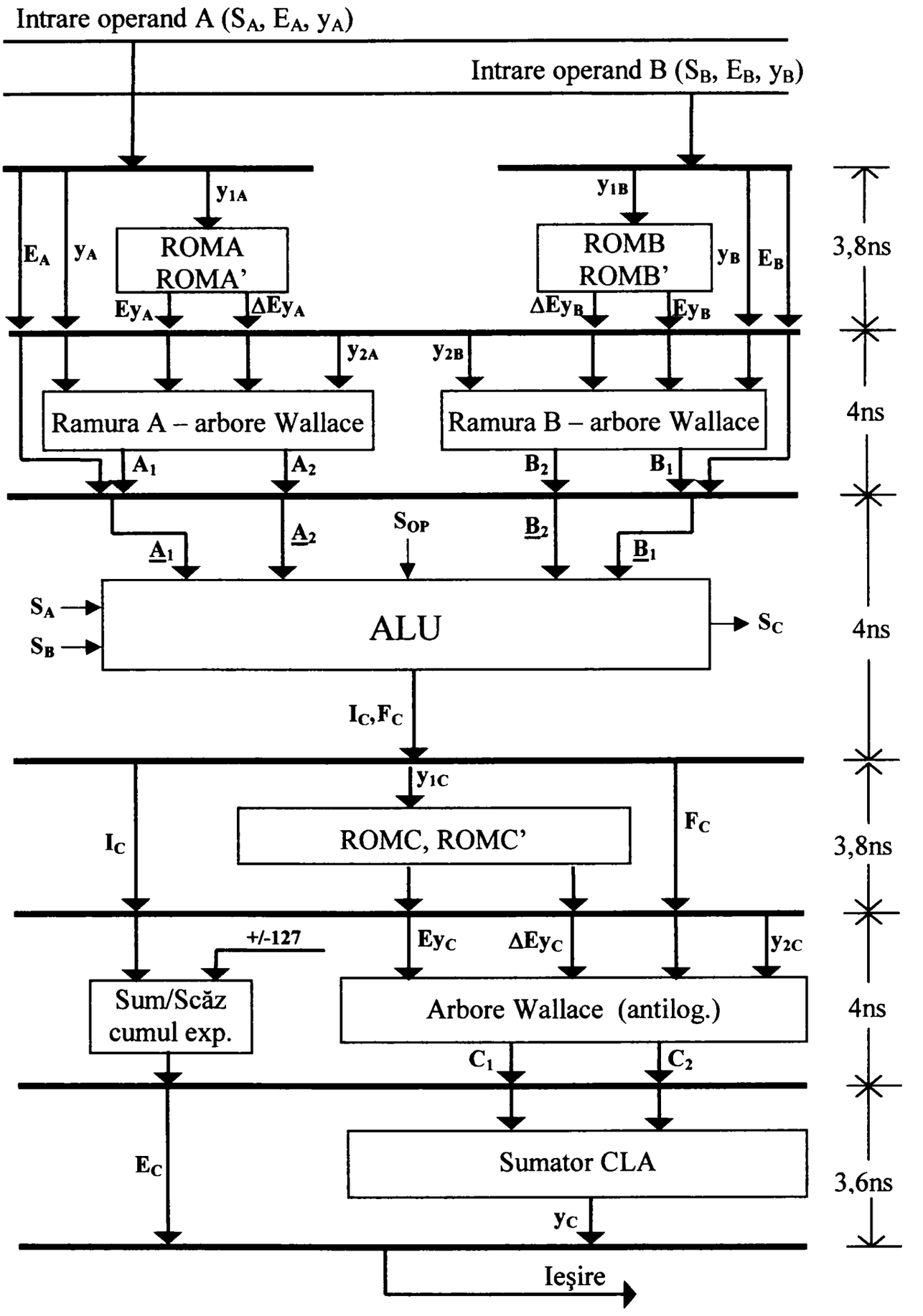


Fig. 2.25 Organizarea noii subunităţi logaritmice pe niveluri pipeline

Aşa cum se vede în figura 2.26, teoretic eroarea totală de conversie se poate reduce la jumătate, dacă la valoarea memorată Ey se adaugă o cantitate egală cu jumătate din eroarea maximă de conversie, adică $1,5\times10^{-7}$. Această cantitate nu poate fi reprezentată pe un număr întreg de biţi, care ar fi putut să fie 1 sau 2. Soluţia de compromis este deci adăugarea unui 1 (LSB) în acele locaţii pentru care eroarea de conversie este mai mare decât 1 LSB, adică mai mare decât $1,2\times10^{-7}$.

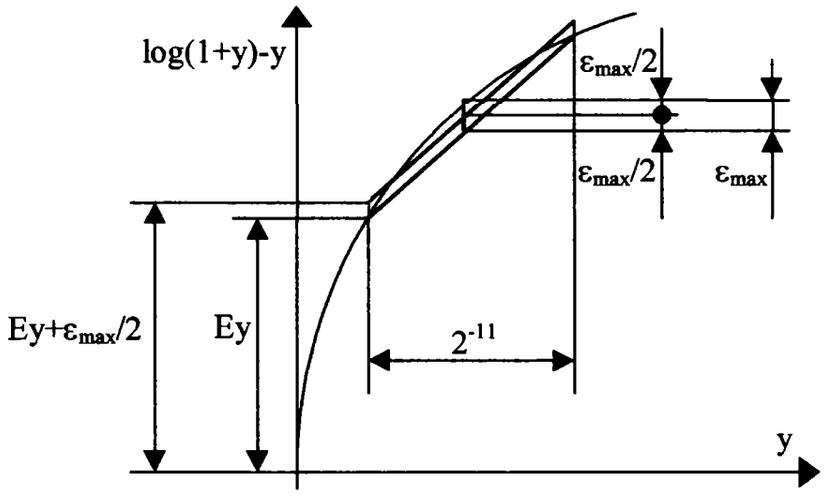


Fig. 2.26 Model teoretic pentru reducerea la jumătate a erorii maxime de conversie

În Anexa 4 sunt prezentate programele Matlab care identifică intervalele de adresă în care trebuie făcută corecţia cu 1 bit şi furnizează valorile optime Ey, necesare pentru calculul logaritmului şi antilogaritmului. Aceste programe vor furniza şi valorile ΔEy necesare pentru cele două conversii de format. În felul acesta, eroarea de conversie, atât la logaritmare cât şi la antilogaritmare a scăzut la $1,8\times10^{-7}$.

În ce priveşte calcularea produsului $\Delta Ey\times y_2$, se poate constata din figura 2.19 unde este prezentată aria de înmulţire, că dacă se face calculul erorii de trunchiere, în cazul cel mai defavorabil, când toţi biţii de rang mai mic sau egal cu "–28" (marcaţi cu caractere neîngroşate) sunt egali cu 1, se obţine valoarea:

$$2\times10^{-35}+3\times10^{-34}+4\times10^{-33}+5\times10^{-32}+6\times10^{-31}+7\times10^{-30}+8\times10^{-29}+9\times10^{-28}=$$
$$=5,96\times10^{-8}$$

Această valoare reprezintă jumătate din eroarea de reprezentare în simplă precizie.

Înlăturarea tuturor biţilor din zona în care ei sunt marcaţi cu caractere neîngroşate, din figura 2.19 şi figura 2.20, adică eliminarea structurilor hard din tot arborele Wallace şi ALU care corespund acestei zone, nu introduce eroare suplimentară dacă valorile Ey care se memorează în ROMA se rotunjesc la 23 de biţi prin adaos, iar valorile Ey al căror complement de doi se memorează în ROMC, se rotunjesc prin lipsă.

Cap.3 Concepţia unei subunităţi de calcul în virgulă flotantă pentru realizarea operaţiilor de adunare şi scădere

În acest capitol va fi prezentată o subunitate de calcul în virgulă flotantă, care implementează operaţiile de adunare şi scădere. Structura proiectată va fi optimizată din punct de vedere al suprafeţei ocupate pe cip şi al realizării sincronizării cu subunitatea logaritmică. Astfel, ansamblul format din cele două subunităţi va funcţiona în parametri de mare performanţă, mai ales în cazul implementării prin microprogramare a unor algoritmi DSP, sau transformări de coordonate pentru aplicaţii grafice.

Ţinând cont că procesorul de calcul descris în această lucrare a fost conceput în primul rând pentru acest gen de aplicaţii şi apoi ca şi coprocesor propriu-zis, nu atât de importantă se consideră a fi durata totală a unei operaţii singulare (ce depinde de numărul maxim de niveluri al structurii pipeline), cât timpul de propagare a transportului pe un nivel al structurii pipeline. Un algoritm DSP poate presupune zeci sau chiar sute de perioade de clock şi atunci contează mai puţin faptul că la numărul total de cicli se mai adaugă încă câţiva pentru introducerea şi procesarea progresivă a primilor operanzi respectiv pentru recuperarea ultimelor date procesate.

În acest context şi în conformitate cu aspectele analizate în paragraful 1.2.3 legate de organizarea unei subunităţi de adunare/scădere în virgulă flotantă, am adoptat pentru aceasta o structură pipeline pe trei niveluri, care să poată funcţiona la o frecvenţă de clock de 250 MHz, ca şi subunitatea logaritmică, atunci când tehnologia utilizată este CMOS 0,5 μm. Deşi literatura de specialitate oferă soluţii care permit reducerea numărului de niveluri pipeline la două în condiţiile menţinerii aceleiaşi frecvenţe de lucru a sistemului, preţul plătit prin creşterea exagerată a suprafeţei ocupate pe cip de această subunitate de calcul (aşa cum s-a menţionat în paragraful 1.2.3) nu justifică creşterea puţin semnificativă a vitezei de procesare în cazul unor aplicaţii DSP.

Datele iniţiale, respectiv parametrii între care trebuie să se încadreze subunitatea de adunare/scădere în virgulă flotantă se referă aşadar la organizarea sa pe trei niveluri pipeline, iar timpul maxim de propagare al transportului pe fiecare nivel nu trebuie să depăşească 4 ns. Astfel, în nivelul 1 se efectuează alinierea mantiselor, în nivelul 2 se operează mantisele aliniate, iar în nivelul 3 se realizează normalizarea rezultatului. Ca şi în cazul subunităţii logaritmice, rotunjirea rezultatului se va face implicit, prin lipsă.

Au fost proiectate şi simulate cu ajutorul programului MSim circuite rapide de aliniere a mantisei şi de normalizare precum şi un sumator/scăzător în virgulă fixă pe 24 de biţi. Aşa cum se va vedea ulterior, s-au obţinut timpi de propagare pe fiecare nivel pipeline sub 3 ns, comparabili sau chiar mai mici decât în cazul unor implementări prezentate în literatura de specialitate. În aceste condiţii, pentru a menţine sincronizarea cu unitatea logaritmică la un preţ plătit cât mai mic, a avut

loc o reorientare în concepţia subunităţii în virgulă flotantă, în sensul optimizării ei din punct de vedere al suprafeţei ocupate pe cip. Complexitatea circuitelor necesare pentru implementarea operaţiilor de adunare şi scădere a fost redusă până la acel punct ce corespunde situaţiei în care durata de procesare a informaţiei binare pe fiecare nivel pipeline încă nu a depăşit 4 ns.

3.1 Proiectarea circuitului de aliniere a mantiselor

Se ştie că deplasarea la dreapta cu un bit a mantisei implică incrementarea exponentului asociat cu o unitate; respectiv dacă diferenţa exponenţilor celor doi operanzi este 1, mantisa asociată celui mai mic exponent trebuie deplasată la dreapta cu un bit. Aceasta înseamnă că în general dacă bitul de ordin i (s_i) din diferenţa exponenţilor este 1, mantisa corespunzătoare celui mai mic exponent trebuie deplasată la dreapta cu 2^i biţi. Se denumeşte mantisă inferioară, mantisa asociată exponentului mai mic. Cantitatea totală de deplasare a mantisei inferioare este deci dată de rezultatul diferenţei celor doi exponenţi. Totuşi biţii de ordin 5, 6 şi 7 de valoare 1 ar implica deplasări de 32, 64 şi respectiv 128 de biţi ceea ce este imposibil întrucât mantisa are doar 23 de biţi în simplă precizie. Aceasta înseamnă că dacă cel puţin unul din aceşti biţi este 1, toţi biţii mantisei inferioare vor deveni egali cu zero.

Deoarece însă iniţial nu se cunoaşte care este semnul diferenţei exponenţilor, nu se poate stabili care este mantisa inferioară, pentru a se putea starta deplasarea acesteia. Circuitul care realizează deplasarea mantisei, "barrel shifter", introduce zerouri în stânga mantisei, pe măsură ce aceasta se deplasează spre dreapta. El este compus din mai multe etaje, sau selectoare, succesive. Selectorul de rang i fie lasă data nedeplasată, fie o deplasează cu 2^i poziţii, în funcţie de valoarea bitului s_i din diferenţa pozitivă a exponenţilor. O secţiune din blocul "barrel shifter" este prezentată în figura 3.1.

Pentru a evita reconversia din cod complement de doi în cod semn-mărime în cazul diferenţei negative a celor doi exponenţi ExpA şi ExpB care sunt ambii pozitivi (având inclus deplasamentul de 127), se calculează simultan ExpA-ExpB şi ExpB-ExpA şi se reţine diferenţa pozitivă.

De asemenea selectoarele din blocul "barrel shifter" se aranjează astfel încât să înceapă să opereze cu deplasări mici spre deplasări mari, ţinând cont că şi biţii diferenţei exponenţilor se stabilesc dinspre LSB spre MSB. Soluţia pe care am propus-o ca şi **varianta 1**, similară cu cea prezentată în [91], a fost utilizarea a două circuite "barrel shifter"care operează în paralel şi care sunt comandate de biţii diferenţelor ExpA-ExpB, respectiv ExpB-ExpA. Astfel cele două mantise, a căror valoare iniţială este conservată, încep să se deplaseze simultan cu stabilirea biţilor celor două diferenţe. În final doar una din cele două date deplasate va fi utilă. Ea va fi selectată împreună cu cealaltă mantisă nedeplasată, în momentul în care este cunoscut rezultatul pozitiv al diferenţei celor doi exponenţi. Schema bloc a circuitului de aliniere a mantiselor este prezentată în figura 3.2.

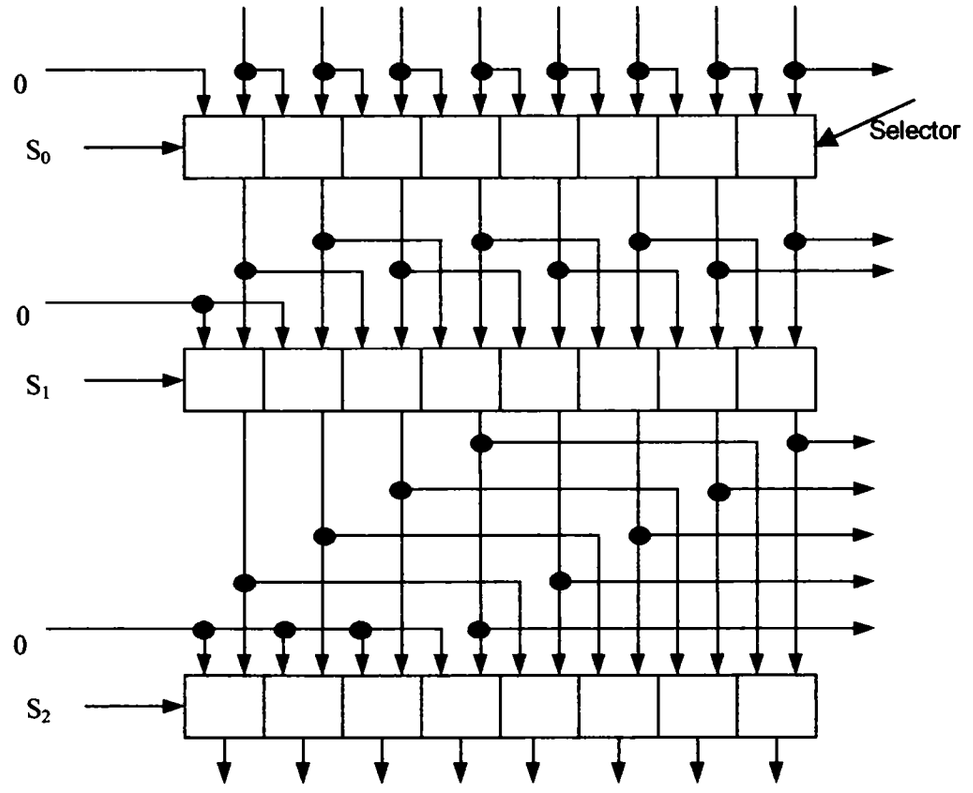


Fig. 3.1 Secţiune din blocul "barrel shifter"

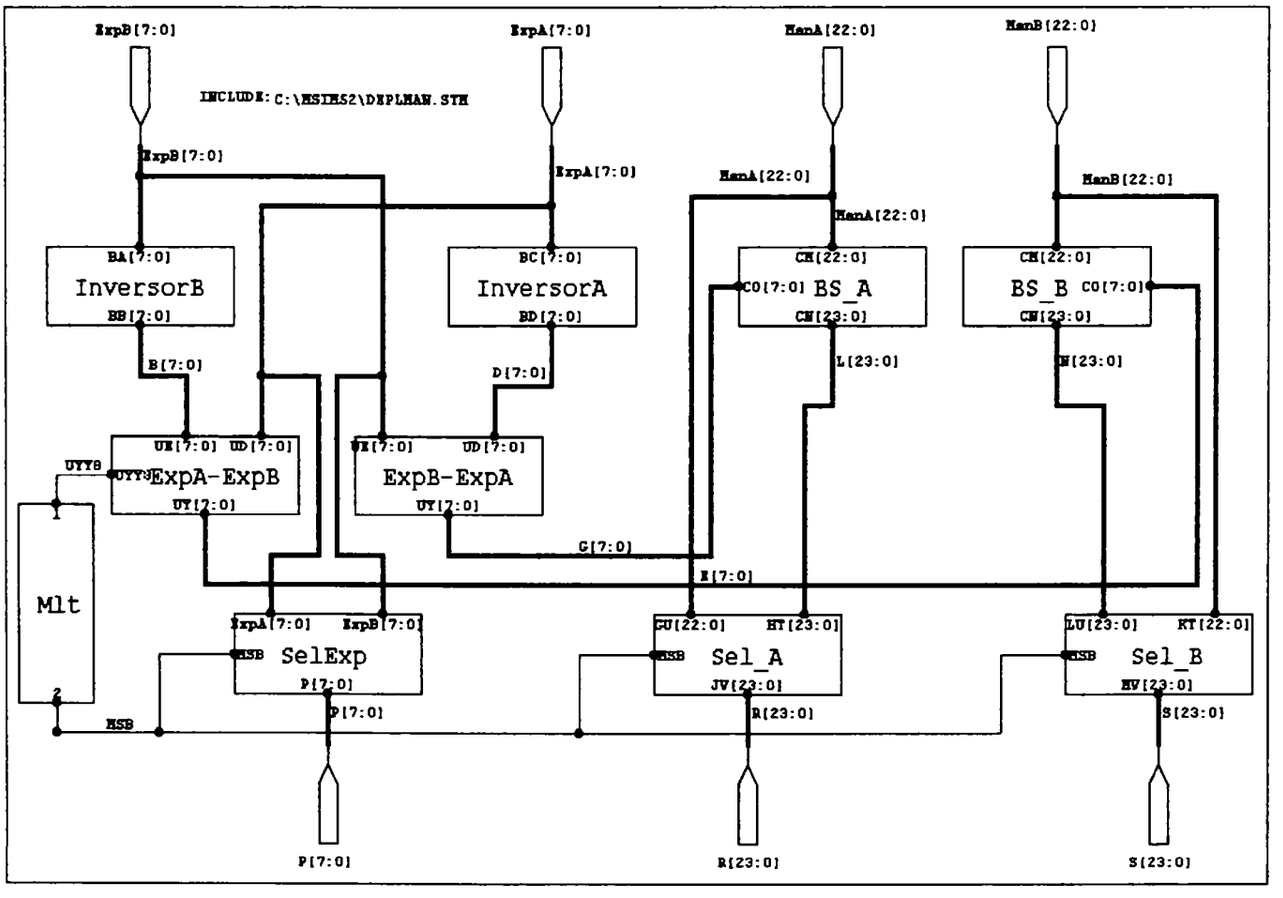


Fig.3.2 Schema bloc a circuitului de aliniere a mantiselor – varianta 1

Diferenţele ExpA-ExpB, respectiv ExpB-ExpA sunt efectuate folosind reprezentarea în complement de doi, cu ajutorul blocurilor InversorA, InversorB şi al sumatoarelor "ExpA-ExpB", respectiv "ExpB-ExpA" prin intermediul cărora se calculează ExpA+/ExpB+1, respectiv ExpB+/ExpA+1 (/ExpA(B) este data obţinută prin inversarea biţilor lui ExpA(B)). Sumatorul "ExpA-ExpB" a fost extins la 9 biţi pentru ca bitul MSB (UYY8 în figura 3.2) al sumei obţinute să poată sesiza semnul diferenţei celor 2 exponenţi pozitivi, extinşi şi ei la 9 biţi prin adăugarea la stânga a câte unui zero. *Spre deosebire de soluţia prezentată în* [91], *cele două sumatoare folosite sunt sumatoare de concepţie proprie, organizate pe două niveluri ierarhice, similare cu sumatorul din figura 2.12. În figura 3.3 este prezentată schema bloc a sumatorului rapid "ExpA-ExpB". Nivelul ierarhic inferior este reprezentat de trei sumatoare CLA ("carry look-ahead") pe 4 biţi (CLA1 cu transport iniţial 1, necesar pentru efectuarea diferenţei exponenţilor) şi respectiv 5 biţi (CLA2 cu transport iniţial 1 şi CLA3 cu transport iniţial 0). Nivelul ierarhic superior implementează un mecanism de selecţie a transportului prin care blocul SEL selectează rezultatul de la CLA2 sau CLA3 în funcţie de transportul c4 generat de CLA1.* Blocurile A', B' şi B" au fost prezentate în figura 2.10, iar blocul CON este conector de magistrală.

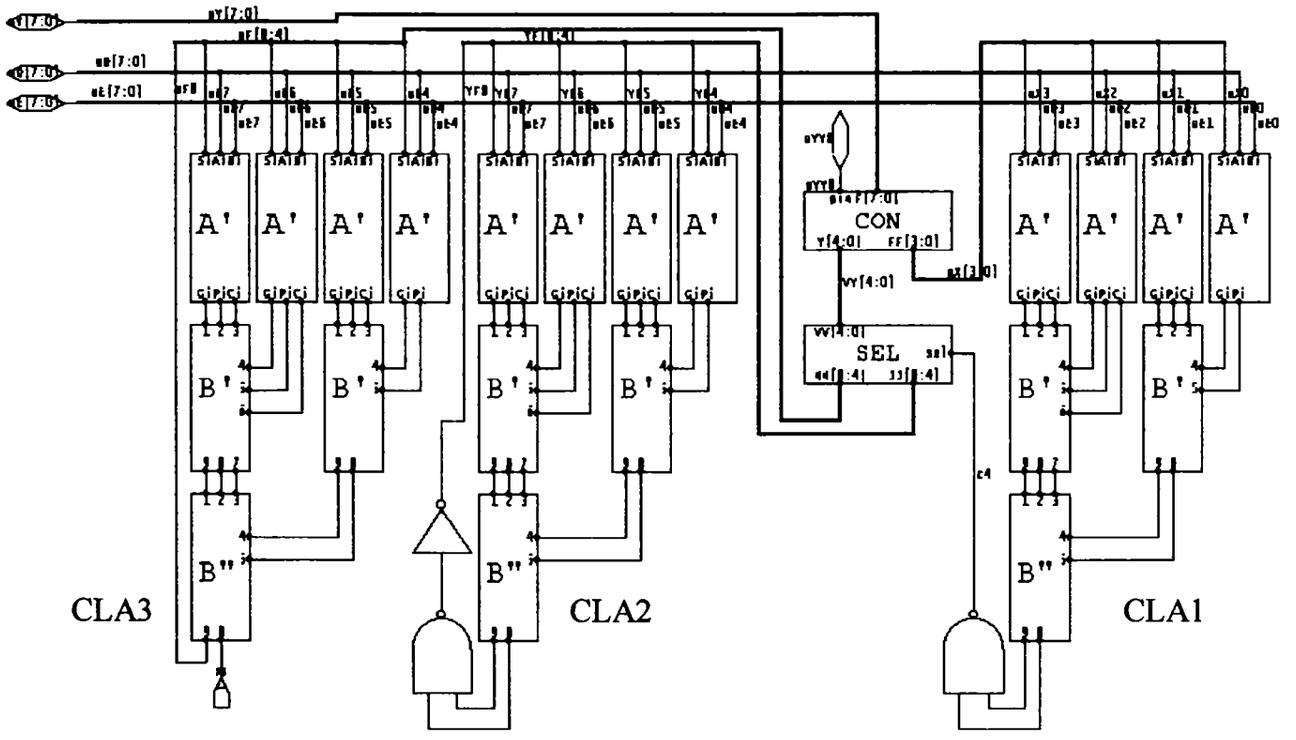


Fig. 3.3 Schema bloc a sumatorului rapid "ExpA-ExpB"

Cele două blocuri "barrel shifter" din figura 3.2, BS_A şi BS_B, conţin câte 6 selectoare de genul celor din figura 3.1, dar cu 24 de celule fiecare. Primul selector din fiecare circuit "barrel shifter" va avea la intrare obligatoriu MSB=1, adică partea întreagă a mantisei ce este astfel restituită părţii fracţionare pe 23 de biţi. Sunt necesare doar 6 selectoare şi nu 8 deoarece, aşa cum s-a precizat anterior,

deplasări de 32, 64, respectiv 128 biţi nu au sens, astfel că biţii s_5, s_6 şi s_7 ai diferenţelor exponenţilor au fost grupaţi printr-un circuit SAU pe o singură linie, notată s_{5+}. Selectoarele au fost proiectate utilizând porţi de transmisie, respectiv tranzistoare de trecere ("pass-transistors") complementare, pentru selectarea uneia sau alteia din cele două căi de acces ale fiecărei celule din selector. În acest fel fiecare linie de comandă, de la s_0 la s_{5+} ar trebui să suporte încărcarea capacitivă a 2×24=48 grile de tranzistoare MOS. Acest lucru ar fi dezastruos atât pentru porţile logice ce furnizează biţii s_i cât şi pentru viteza de operare. Problema se pune şi mai pregnant pentru linia MSB din figura 2.2 care trebuie să comande 112 grile de tranzistoare MOS.

Evitarea acestor situaţii s-a făcut utilizând structuri arborescente de inversoare CMOS, pentru multiplicarea capabilităţii de comandă a liniilor menţionate. Arborele de la capătul fiecărei linii s_i va conţine patru etaje de inversoare ce nu introduc însă o întârziere semnificativă.

În cazul liniei MSB, în blocul "Mlt" din figura 2.2, se va utiliza un arbore cu 5 niveluri, la care ultimul nivel va avea 32 ieşiri CMOS, care vor fi folosite pentru comanda celor 96 grile MOS din Sel_A şi Sel_B şi pentru cele 16 grile din Sel_Exp. Astfel fiecărei ieşiri CMOS i s-au considerat alocate maxim 4 grile de tranzistoare.

Bitul MSB de la "ExpA-ExpB" este cel care selectează la ieşirea etajului exponentul mai mare cu ajutorul blocului SelExp, respectiv perechea corectă mantisă nedeplasată - mantisă deplasată, cu ajutorul blocurilor Sel_A şi Sel_B. Logica de selecţie este următoarea: dacă MSB este 1, atunci va fi selectat ExpB şi rezultatul furnizat de blocul BS_A, iar dacă MSB este 0, atunci va fi selectat ExpA şi rezultatul furnizat de BS_B.

Toate blocurile componente şi subblocurile din schemele ierarhice inferioare au fost proiectate până la nivel de poartă logică sau poartă de transmisie şi sunt prezentate în Anexa 5.

Au fost efectuate simulări pentru cazul cel mai defavorabil şi a fost obţinut un timp maxim de propagare prin circuitul din figura 2.2 de 2,4 ns (utilizând etalonul - tehnologie CMOS 0,5 microni). Acest rezultat recomandă circuitul proiectat pentru procesoarele aritmetice în virgulă flotantă, însă nu e potrivit cu ceea ce se doreşte în cazul de faţă, adică timp de propagare aproape de 4ns şi arie ocupată de circuit cât mai mică.

Într-adevăr aria ocupată pe cip este mare deoarece două circuite "barrel shifter" înseamnă nu numai număr dublu de tranzistoare ci şi spaţiu de rutare între selectoare dublu. Într-adevăr, aşa cum se poate observa chiar şi din figura 2.1, spaţiul de rutare pentru deplasare cu 2^i este proporţional cu 2^i, adică devine din ce în ce mai mare pe măsură ce se accede spre selectoarele care realizează deplasări mari. La acest spaţiu se adaugă şi cel necesar pentru multiplicarea capabilităţii de comandă a liniei de selecţie s_i, acelaşi pentru fiecare selector. De asemenea cele două sumatoare rapide, precum şi cele trei blocuri selectoare ocupă o suprafaţă importantă.

Pentru a realiza o mai bună apropiere de scopul propus, la elaborarea ***variantei 2*** *de implementare a circuitului de aliniere a mantiselor, am plecat de la soluţia descrisă în* [26] *la nivel de schemă bloc şi pe care am implementat-o (cu anumite modificări) printr-o proiectare completă până la nivel de poartă logică. Rezultatele obţinute au fost prezentate în [39].* În această variantă se utilizează trei circuite "barrel shifter", două care operează deplasări mici pentru fiecare mantisă, până la maxim 7 poziţii (conţin câte 3 selectoare) şi un al treilea care primeşte deja mantisa corectă, parţial deplasată şi care conţine tot trei selectoare. În acest fel, spaţiul de rutare excesiv, necesar pentru realizarea deplasărilor mari s-a redus la jumătate. Schema bloc a acestui circuit este prezentată în figura 3.4.

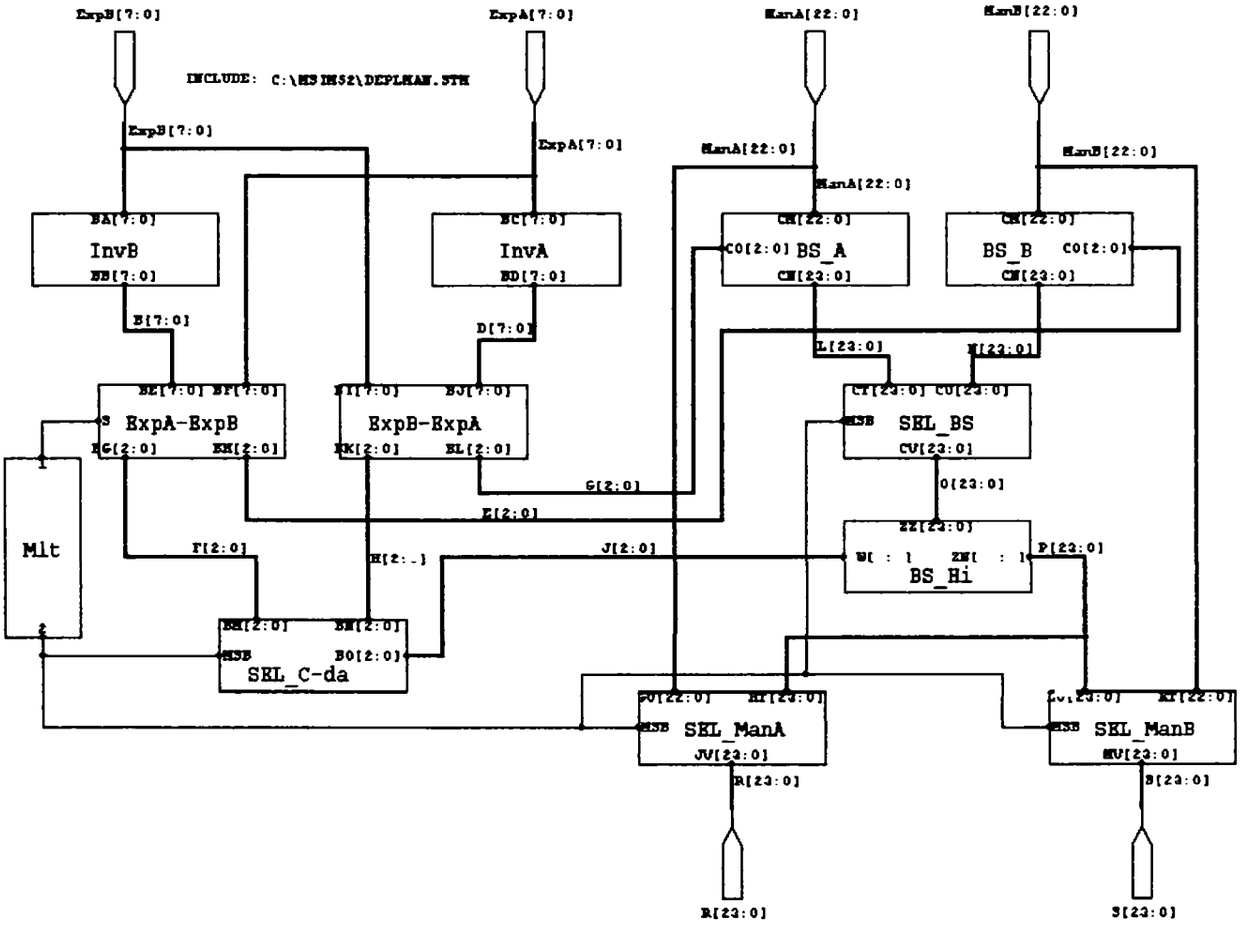


Fig. 3.4 Schema bloc a circuitului de aliniere a mantiselor – varianta 2

Pentru simplitate, din schema bloc lipseşte circuitul de selecţie a exponentului mai mare, dar acest lucru nu afectează viteza de operare deoarece circuitul menţionat nu se găseşte pe calea critică de propagare.

Cele 6 linii de comandă (de la s_0 la s_{5+}) ale circuitelor "barrel shifter", furnizate de blocurile "ExpA-ExpB" şi "ExpB-ExpA", sunt grupate în două perechi de câte 3 biţi. Perechea ce conţine cei trei biţi mai puţin semnificativi ($s_0 \div s_2$) de la ieşirea fiecărui sumator, este folosită pentru comanda circuitelor BS_A şi BS_B care realizează deplasările mici. Din cealaltă pereche de triplete de biţi se va selecta în blocul "Sel_C-dă" cu ajutorul aceluiaşi MSB de la sumatorul "ExpA-ExpB", tripleta care va comanda circuitul "barrel shifter" BS_Hi. În acelaşi

timp bitul MSB selectează prin intermediul blocului Sel_BS mantisa corectă, parţial deplasată ce se aplică la intrarea blocului BS_Hi. Acest bloc va realiza prin intermediul celor 3 selectoare din interiorul său deplasări de 8, 16, respectiv 32 biţi. Ţinând cont că dinspre stânga se introduc zerouri în mantisă atunci când ea se deplasează spre dreapta, o deplasare de 32 biţi echivalează practic cu anularea tuturor biţilor mantisei. Această situaţie se poate întâlni atunci când ordinul de mărime al celor doi operanzi este mult diferit.

Deosebirea faţă de soluţia descrisă în [26] *constă în modificarea tipului de sumator folosit pentru a calcula diferenţa exponenţilor. Un sumator simplu, format din 9 sumatoare complete pe un bit cascadate, cum s-a indicat în* [26] *nu ar fi fost suficient de rapid pentru ca procesarea informaţiei binare pe acest nivel pipeline să se facă sub 4 ns. Din acest motiv am adoptat un sumator cu selecţia transportului organizat pe două secţiuni de 4 şi respectiv 5 biţi. Schemele de detaliu ale acestei variante de circuit sunt prezentate de asemenea în Anexa 5.*

În urma simulărilor efectuate pentru cazul cel mai defavorabil, s-a obţinut un timp de propagare de 3,7 ns, în condiţiile în care economia de suprafaţă faţă de varianta precedentă a fost estimată la 25%.

Varianta cea mai practică de implementare a circuitului de aliniere a mantiselor pentru cazul particular al procesorului hibrid descris în această lucrare, este **varianta 3** care conduce la schema bloc prezentată în figura 3.5.

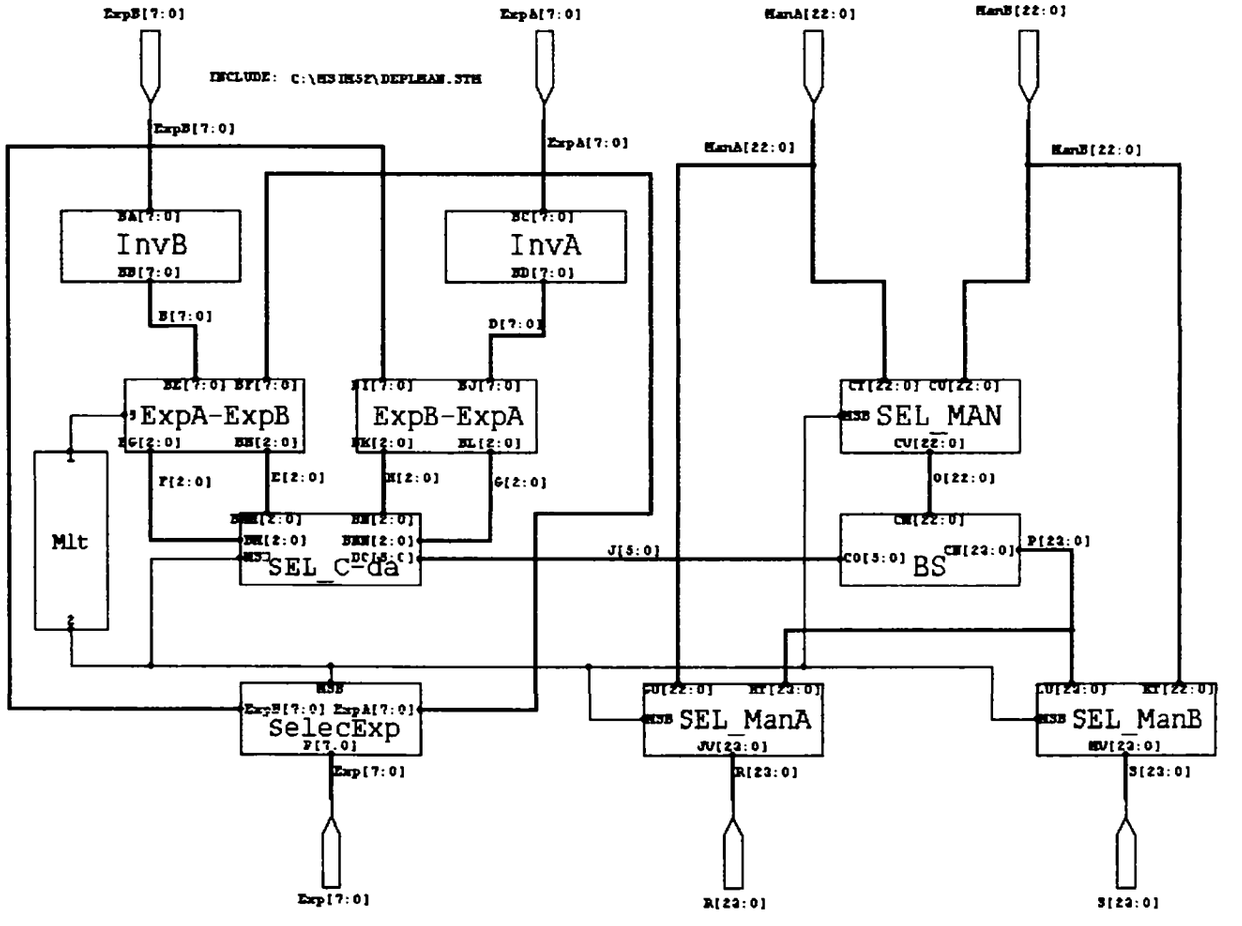


Fig. 3.5 Schema bloc a circuitului de aliniere a mantiselor – varianta 3

În această variantă este folosit un singur circuit "barrel shifter", (blocul BS din figura 3.5) care deplasează mantisa inferioară, stabilită după aflarea semnului diferenţei celor doi exponenţi. Nu poate fi folosit însă un sumator unic pentru calculul unei singure diferenţe a exponenţilor întrucât, dacă rezultatul se dovedeşte a fi negativ el va fi reprezentat în complement de doi şi trebuie reconvertit în cod mărime-semn, ceea ce întârzie prea mult producerea rezultatului.

Linia MSB de la ieşirea sumatorului "ExpA-ExpB", identic cu cel folosit în varianta 2, realizează următoarele funcţiuni:

- selectează mantisa asociată exponentului mai mic, adică mantisa inferioară, pentru a fi aplicată la intrarea blocului BS;
- selectează prin intermediul blocului Sel_C-dă liniile $s_0 \div s_{5+}$ furnizate de sumatorul al cărui rezultat este pozitiv, pentru a comanda deplasările mantisei inferioare în blocul BS;
- selectează mantisa asociată exponentului mai mare, adică mantisa superioară, direct la ieşirea circuitului. Prin intermediul blocurilor Sel_ManA şi Sel_ManB este păstrată poziţia relativă a celor două mantise după alinierea lor;
- selectează prin intermediul blocului SelecExp exponentul mai mare la ieşirea circuitului.

Ca şi în variantele anterioare, capabilitatea de comandă a liniei MSB a trebuit să fie multiplicată folosind un arbore de inversoare CMOS, ceea ce a implicat introducerea în schema simulată a blocului Mlt (figura 3.5).

Un alt aspect important de care s-a ţinut cont la construcţia circuitului "barrel shifter" a fost limitarea capacităţilor parazite care se acumulează datorită cascadării porţilor de transmisie din circuite selectoare consecutive şi care încarcă excesiv latch-ul de date de la intrarea acestui nivel pipeline. Din acest motiv blocul BS a fost secţionat de un rând dublu de inversoare CMOS, care nu modifică valoarea mantisei inferioare parţial deplasate ce se aplică mai departe următoarelor selectoare din circuitul "barrel shifter". În acest fel fronturile semnalelor propagate prin structură sunt refăcute şi nu apar întârzieri critice.

În urma simulărilor efectuate, s-a obţinut un timp maxim de propagare pe acest nivel pipeline de 4 ns. Schemele detaliate ale circuitului sunt de asemenea prezentate în Anexa 5. Economia de suprafaţă pe cip a fost estimată la aproape 50% faţă de varianta 1.

3.2 Proiectarea circuitului de adunare/scădere a mantiselor aliniate

Circuitul de adunare/scădere a mantiselor aliniate operează pe 25 de biţi întrucât la cei 23 de biţi memoraţi ai mantiselor s-a restituit bitul implicit din stânga virgulei, precum şi bitul de semn. *Circuitul este de acelaşi tip cu cel utilizat în unitatea logaritmică şi prezentat în figura 2.9, deci implementează în acelaşi mod operaţia de scădere, dar este mai complex întrucât include blocuri inversoare pentru ambii operanzi (oricare din ei poate fi negativ) şi un bloc logic care controlează selectoarele din blocurile inversoare, respectiv selectorul final, în*

funcţie de operaţia ce trebuie efectuată (adunare sau scădere) cât şi de semnul celor doi operanzi. Schema lui bloc este prezentată în figura 3.6.

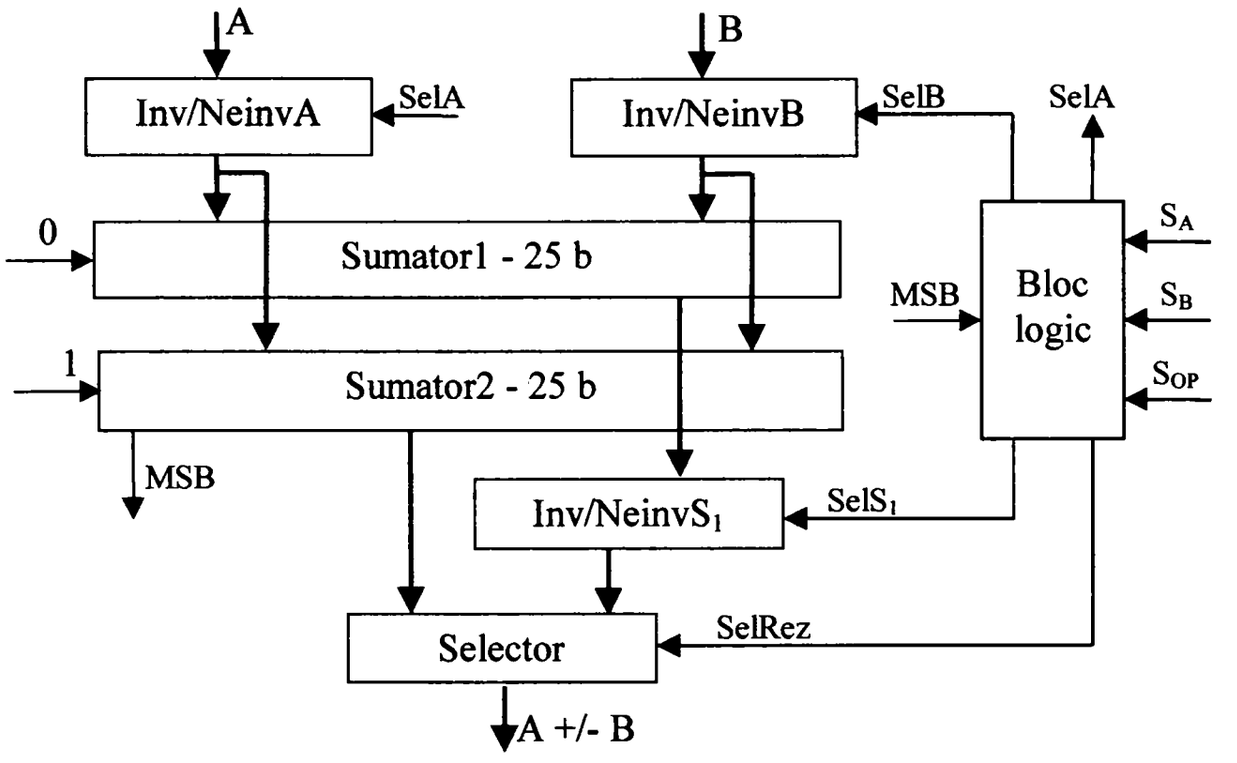


Fig. 3.6 Schema bloc a circuitului de adunare/scădere a mantiselor aliniate

Considerând A şi B ca fiind cele două mantise aliniate, S_A şi S_B semnele lor, iar S_{OP} bitul asociat operaţiei (0 pentru adunare şi 1 pentru scădere), atunci blocul logic implementează următoarea regulă:

-dacă:

(a1): $S_A = 0$, $S_B = 0$, $S_{OP} = 0$, sau
(a2): $S_A = 1$, $S_B = 1$, $S_{OP} = 0$, sau
(a3): $S_A = 0$, $S_B = 1$, $S_{OP} = 1$, sau
(a4): $S_A = 1$, $S_B = 0$, $S_{OP} = 1$,

atunci se efectuează adunare propriu-zisă, iar semnul rezultatului este semnul lui A ($S_{Rez} = S_A$). În acest caz: $S_A \oplus S_B \oplus S_{OP} = 0$.

-dacă:

(s1): $S_A = 0$, $S_B = 0$, $S_{OP} = 1$, sau
(s2): $S_A = 1$, $S_B = 1$, $S_{OP} = 1$, sau
(s3): $S_A = 0$, $S_B = 1$, $S_{OP} = 0$, sau
(s4): $S_A = 1$, $S_B = 0$, $S_{OP} = 0$,

atunci se efectuează scădere propriu-zisă pe 25 de biţi. Se constată că în acest caz: $S_A \oplus S_B \oplus S_{OP} = 1$.

În cazul adunării propriu-zise, blocurile Inv/NeinvA, Inv/NeinvB şi Inv/NeinvS_1 trebuie să fie transparente (SelA=SelB=SelS_1=0) iar blocul Selector va selecta rezultatul furnizat de Sumator1 (SelRez=0).

În cazul scăderii propriu-zise, blocul Inv/NeinvA va fi transparent în situaţiile (s1) şi (s3) pentru care linia SelA=0, iar blocul Inv/NeinvB va fi transparent în situaţiile (s2) şi (s4) pentru care linia SelB=0. Blocul Inv/NeinvS_1 va efectua inversarea biţilor de la Sumator1 (SelS_1=1), iar blocul Selector va selecta prin semnalul SelRez rezultatul de la Sumator2 dacă MSB=1, respectiv de la blocul Inv/NeinvS_1 dacă MSB=0. Semnul rezultatului este în acest caz dat de MSB. În concluzie Blocul logic trebuie să asigure, în concordanţă cu constatările anterioare, următoarele funcţii:

$$SelA = S_A S_B S_{OP} + S_A /S_B /S_{OP} = S_A \overline{(S_B \oplus S_{OP})} \qquad (3.1)$$

$$SelB = /S_A /S_B S_{OP} + /S_A S_B /S_{OP} = /S_A (S_B \oplus S_{OP}) \qquad (3.2)$$

în care $/S_A$, $/S_B$ şi $/S_{OP}$ sunt S_A negat, S_B negat, respectiv S_{OP} negat. De asemenea Blocul logic mai furnizează:

$$SelS_1 = S_A \oplus S_B \oplus S_{OP} \qquad (3.3)$$

$$SelRez = /MSB\ (S_A \oplus S_B \oplus S_{OP}) \qquad (3.4)$$

Schema blocului logic este prezentată în figura 3.7. Aşa cum se observă din această figură, blocul logic a fost astfel proiectat încât nici o ieşire să nu comande mai mult de două intrări logice, respectiv mai mult de 4 grile MOS. Blocul logic mai furnizează prin intermediul ieşirii Srez, semnul rezultatului final. Pe ansamblul circuitului de adunare/scădere, calea critică de propagare trece şi prin interiorul acestui bloc şi cuprinde linia MSB. Din acest motiv, căile care prelucrează bitul MSB includ un număr minim de niveluri logice.

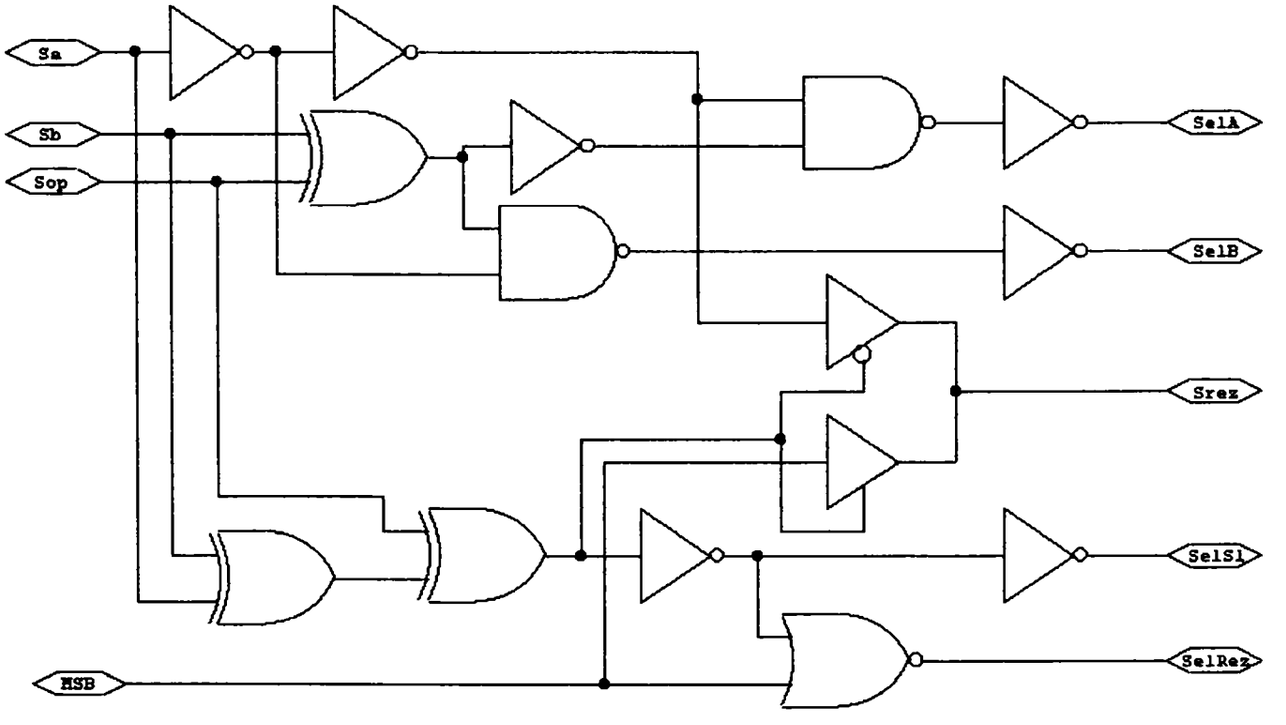


Fig. 3.7 Schema detaliată a Blocului logic

*În **varianta 1** de realizare a circuitului de adunare/acădere blocurile Sumator1 şi Sumator2 din figura 3.6, cu transport iniţial 0 respectiv 1, sunt sumatoare cu selecţia transportului, la care secţiunile componente de câte 8 biţi sunt realizate cu sumatoare de tip CLA. Aceste blocuri au structura similară cu cea a sumatorului prezentat în figura 2.12, beneficiind astfel de toate îmbunătăţirile aduse acestuia în privinţa vitezei de operare. Schema bloc a acestui tip de sumator, de concepţie proprie este prezentată în figura 3.8.*

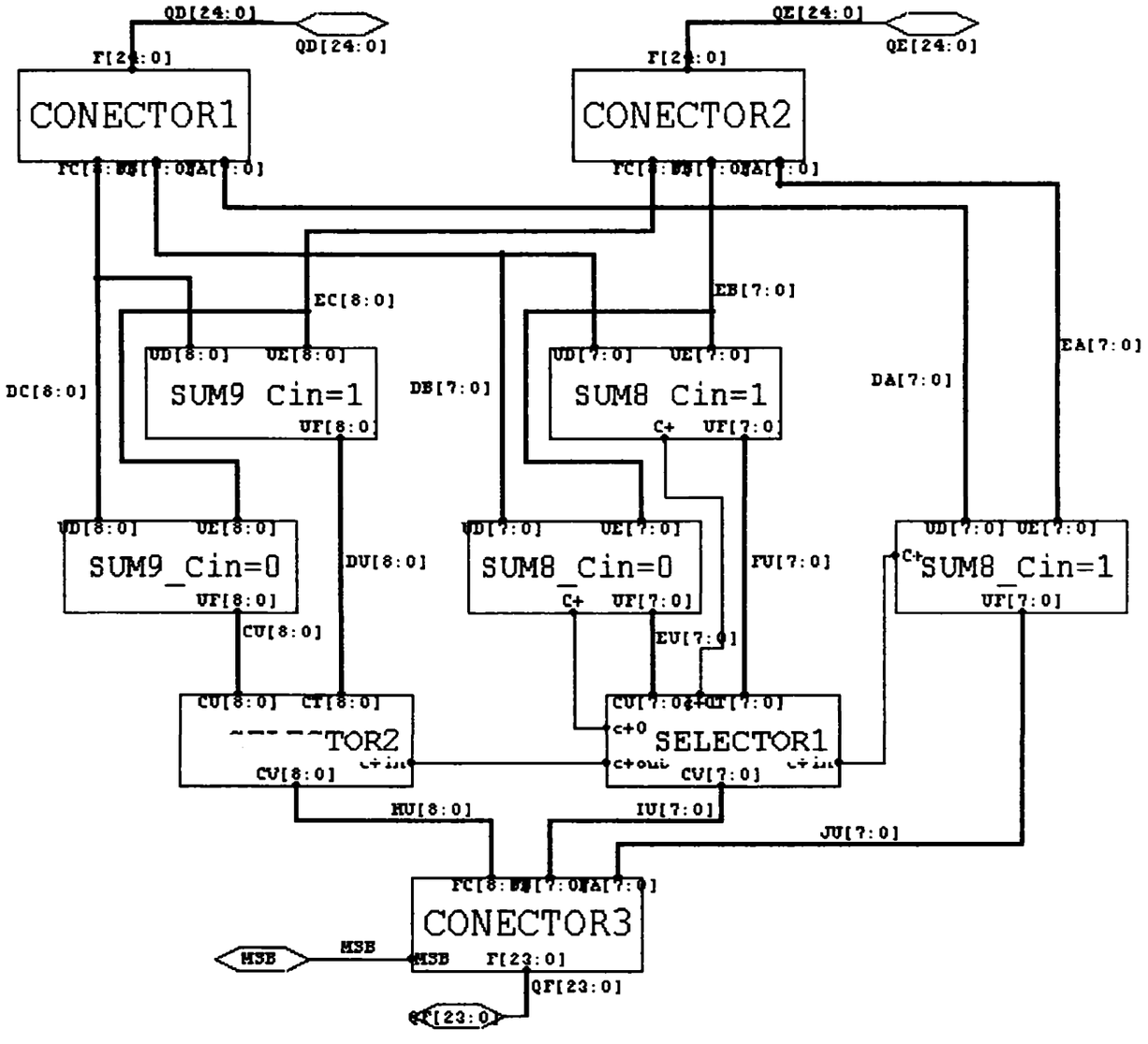


Fig. 3.8 Schema bloc a sumatorului pe 25 de biţi – varianta 1

Bitul al 25-lea (MSB) al sumei, în concordanţă cu observaţia făcută în paragraful 2.2.2.4 la figura 2.11, se generează mai repede decât al 24-lea bit şi deci nu conduce la creşterea timpului de operare al ultimei secţiuni a sumatorului, care va fi pe 9 biţi.

În figura 3.6 blocurile Inv/NeinvA, Inv/NeinvB, Inv/NeinvS_1 şi Selector includ arbori de inversoare CMOS pentru multiplicarea capabilităţii de comandă a liniilor de selecţie SelA, SelB, SelS_1 şi SelRez. Multiplicarea capabilităţii de comandă a liniilor SelA, SelB şi SelS_1 nu afectează viteza sumatorului deoarece S_A,

S_B şi S_{OP} determină starea lor logică încă din timpul operaţiei de aliniere a mantiselor. În schimb, în cazul liniei SelRez, care depinde şi de MSB, viteza sumatorului este întrucâtva afectată. Totuşi acest efect este compensat parţial de faptul că în timpul multiplicării capabilităţii de comandă a liniei SelRez se efectuează inversarea şi selectarea inversării biţilor de la Sumator1 în blocul Inv/NeinvS1.

Arhitectura propusă pentru circuitul de adunare/scădere prezentată în figura 3.6, diferită de cea prezentată în [26], *oferă următoarele avantaje faţă de aceasta:*

- datorită inversării ordinii celor două blocuri finale, Inv/NeinvS1 şi Selector, se elimină timpul necesar inversării şi selecţiei inversării pentru rezultatul final;

- prezenţa Blocului logic în interiorul circuitului de adunare/scădere permite adunarea şi scăderea operanzilor în toate cazurile posibile din punct de vedere al semnului lor şi nu doar adunare de operanzi de acelaşi semn sau de semne contrare, cum se procedează în anumite unităţi de calcul;

- controlul circuitului de adunare/scădere este foarte eficient şi este astfel conceput încât pentru generarea semnalelor necesare numărul nivelurilor logice parcurse este minim; în plus Blocul logic poate fi distribuit între primul şi al doilea nivel pipeline al subunităţii de calcul în virgulă flotantă, ceea ce permite scurtarea semnificativă a căii critice de propagare;

- utilizarea sumatoarelor cu selecţia transportului, la care secţiunile componente de câte 8 biţi sunt realizate cu sumatoare de tip CLA îmbunătăţite, a condus la o creştere substanţială a vitezei de calcul.

În urma simulărilor efectuate, considerând tehnologie CMOS 0,5 μm, timpul de propagare pentru cazul cel mai defavorabil a fost găsit de 4 ns. În realitate însă, timpul total de calcul pentru circuitul de adunare/scădere este mult mai scurt deoarece valoarea logică a liniilor SelA şi SelB din figura 3.6 poate fi cunoscută încă din timpul etapei de deplasare a mantiselor. De asemenea, propagarea transportului prin arborele de multiplicare a capabilităţii de comandă a acestor linii poate fi realizată în etapa precedentă. În concluzie, timpul total de operare este de doar 2,9 ns. Acest rezultat recomandă utilizarea circuitului la concepţia unor unităţi de calcul performante în care se operează exclusiv în virgulă flotantă.

*În **varianta 2** de realizare a circuitului de adunare/scădere se păstrează aceeaşi arhitectură avantajoasă şi se exploatează eficienţa Blocului logic, de această dată nu pentru a obţine o viteză maximă de calcul, ci pentru a putea utiliza, pentru adunarea/scăderea mantiselor aliniate, sumatoare care necesită o suprafaţă mai mică pe cip. Condiţia care a trebuit respectată a fost ca timpul de producere al rezultatului să se încadreze în 4 ns. Prin utilizarea unui sumator CLA îmbunătăţit, de tipul celui din figura 2.12, dar pe 25 de biţi, s-a confirmat un timp de propagare pentru cazul cel mai defavorabil de 4 ns. Numărul porţilor logice şi al porţilor de transmisie a scăzut de la circa 700 la aproximativ 350 ceea ce înseamnă că s-a obţinut 50% economie de suprafaţă faţă de varianta 1.* Schemele de detaliu ale celor două variante de circuit sumator/scăzător se găsesc în Anexa 6.

3.3 Circuitul de normalizare

Circuitul de normalizare ocupă al treilea nivel pipeline al subunităţii de adunare/scădere în virgulă flotantă şi are schema bloc prezentată în figura 3.9.

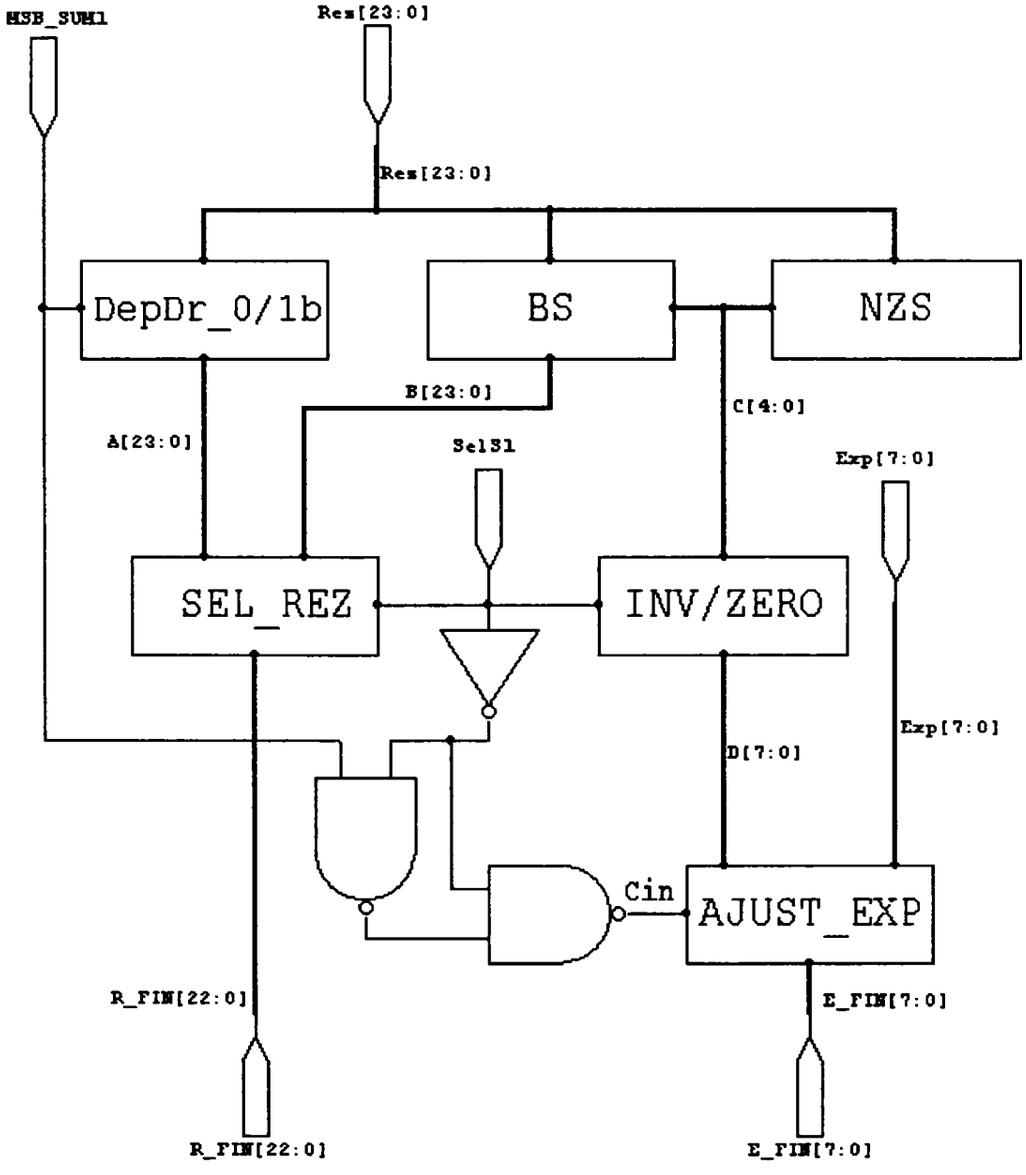


Fig. 3.9 Schema bloc a circuitului de normalizare

În urma efectuării unei adunări propriu-zise, rezultatul în valoare absolută se poate găsi în intervalul [0,4) ceea ce înseamnă că operaţia de normalizare e foarte simplă. Rezultatul fie se lasă nedeplasat, fie se deplasează la dreapta cu un bit, prin intermediul blocului "DepDr_0/1b", bineînţeles în funcţie de valoarea bitului MSB_SUM1 de la blocul Sumator1 din figura 3.6. Atunci însă când se efectuează o scădere propriu-zisă, dacă operanzii sunt foarte apropiaţi în valoare absolută, pot rezulta multe zerouri înaintea celui mai semnificativ 1 din rezultat. Aceste zerouri se denumesc zerouri semnificative. Pentru a se efectua normalizarea, zerourile semnificative sunt în primă fază contorizate în blocul NZS, urmând ca apoi să fie comandat circuitul "barrel shifter" BS, similar cu cel utilizat la deplasarea mantisei

inferioare, în vederea normalizării. Evident în acest caz are loc o pierdere inevitabilă de precizie, întrucât deplasările succesive la stânga ale rezultatului diferenţă implică introducerea de zerouri dinspre dreapta în poziţiile rămase libere.

În final, blocul SEL_REZ va selecta rezultatul de la ieşirea blocului "DepDr_0/1b" sau de la ieşirea blocului BS în funcţie de faptul că în nivelul pipeline 2 s-a efectuat o adunare propriu-zisă (SelS1 din figura 3.6 şi dat de relaţia 3.3 - egal cu zero) sau o scădere propriu-zisă (SelS1=1).

În acelaşi timp, exponentul transmis de la ieşirea primului nivel pipeline al subunităţii de adunare/scădere în virgulă flotantă, se ajustează în nivelul 3 în funcţie de rezultatul normalizării mantisei furnizate la ieşirea nivelului 2. Pentru a realiza acest lucru, în mod normal sunt necesare două circuite sumatoare pe 9 biţi. Unul asigură incrementarea exponentului cu o unitate, în cazul unei adunări propriu-zise în care rezultă doi biţi în stânga virgulei (MSB_SUM1=1). Celălalt asigură scăderea a n unităţi din exponent, în cazul unei scăderi propriu-zise în care rezultă n zerouri semnificative în mantisa furnizată de nivelul pipeline 2. În final este selectat rezultatul corespunzător operaţiei efectuate.

O contribuţie proprie a fost adusă şi în cazul acestui circuit, concretizată prin utilizarea unui singur circuit sumator pe 9 biţi, blocul AJUST_EXP, aşa cum se observă în figura 3.9. Pe baza semnalelor MSB_SUM1 şi SelS1 furnizat de Blocul logic din nivelul 2 (figura 3.7) se asigură un transport iniţial pentru sumatorul AJUST_EXP în următoarele două situaţii:

- atunci când se efectuează scădere propriu-zisă şi trebuie implementată relaţia:

$$Exp - Nzs = Exp + /Nzs + 1 \qquad (3.5)$$

în care Exp este exponentul furnizat de nivelul 1 al subunităţii în virgulă flotantă şi la care se ataşează un zero ca bit de semn, Nzs este numărul zerourilor semnificative exprimat în cod mărime-semn pe 9 biţi, iar /Nzs este data obţinută prin inversarea biţilor lui Nzs cu ajutorul blocului INV/ZERO (figura 3.9) atunci când SelS1=1; dacă rezultatul obţinut la ieşirea acestui bloc rezultă negativ (bitul Exp_8=1), în ciuda deplasamentului de 127 inclus, se semnalează subdepăşire;

- atunci când se efectuează adunare propriu-zisă şi trebuie incrementat cu o unitate exponentul, întrucât prin însumarea celor două mantise au rezultat doi biţi în stânga virgulei şi mantisa a fost deplasată spre dreapta cu un bit; în acest caz (SelS1=0) la ieşirea blocului INV/ZERO este furnizat un cuvânt nul şi se va calcula suma Exp+1; dacă în acest caz rezultă bitul Exp_8=1, fiind vorba de adunarea a două numere pozitive, se semnalează supradepăşire.

Astfel, pentru a obţine semnalul Cin, ce reprezintă transportul iniţial al sumatorului pentru ajustarea exponentului, a fost implementată relaţia:

$$Cin = SelS1 + (/SelS1)MSB\_SUM1 \qquad (3.6)$$

În continuare se va face o analiză a timpului de propagare pe calea critică de propagare prin structura din figura 3.9, care include blocurile NZS, INV/ZERO şi AJUST_EXP. Analiza se face în scopul determinării tipului de sumator ce poate

fi folosit pentru ca timpul de generare a exponentului ajustat să se încadreze în 4 ns, iar suprafaţa ocupată pe cip să fie minimă.

Blocul NZS a fost proiectat în detaliu după metoda descrisă în [85]. Schema sa bloc este prezentată în figura 3.10.

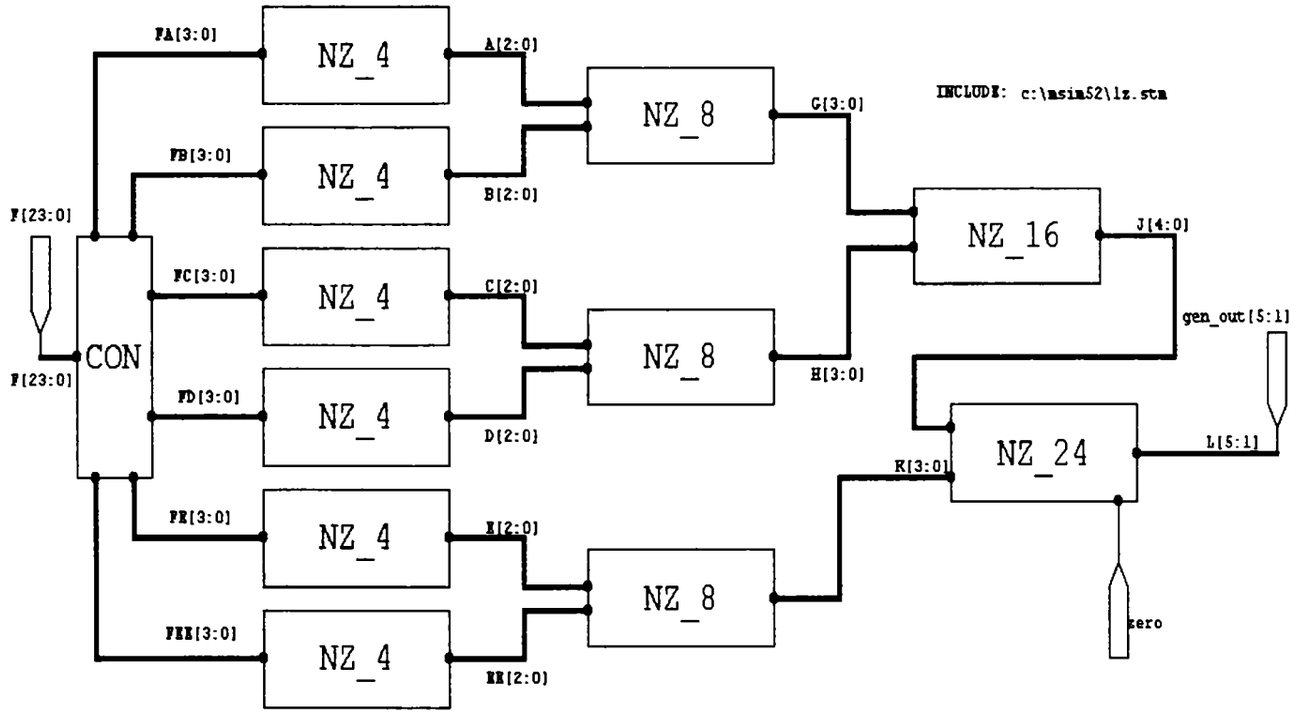


Fig. 3.10 Schema bloc a numărătorului de zerouri semnificative (NZS)

Aşa cum se observă în figura 3.10, circuitul NZS are o structură arborescentă, formată din patru tipuri de subblocuri, fiecare dintre acestea putând detecta un număr de zerouri semnificative (marcat în tipul subblocului) corespunzător grupelor de biţi de intrare asociate lor. Blocul CON serveşte doar la disocierea magistralei de intrare de 24 de biţi în 6 magistrale de câte 4 biţi, prezenţa sa fiind impusă de simulator. Rangul celor 24 de biţi creşte de sus în jos. În fiecare subbloc există o poartă SAU şi unul (în NZ_4), doi (în NZ_8), trei (în NZ_16), respectiv patru multiplexori cu 2 intrări (în NZ_24), prin care este implementat un mecanism prioritar de selecţie. Poarta SAU din subblocul curent comandă multiplexorii subblocului din aval. La ieşirea subblocului NZ_24 se obţine numărul reprezentat pe 5 biţi, al zerourilor semnificative ale cuvântului de 24 de biţi aplicat la intrarea circuitului NZS. Bitul "zero" furnizat de acelaşi subbloc indică situaţia în care toţi biţii de la intrare sunt nuli, caz în care operaţia de normalizare nu mai are sens, iar rezultatul adunării/scăderii celor 2 operanzi este declarat zero. Schemele detaliate ale tuturor blocurilor şi subblocurilor componente se găsesc în Anexa 7.

Timpul critic de propagare printr-un subbloc este dat de poarta SAU (de fapt există porţi SAU-NU alternate cu porţi ŞI-NU, pentru a realiza funcţii SAU cu porţi intrinseci) şi este de 0,2 ns în tehnologie CMOS de 0,5 μm. În concluzie, timpul de propagare prin întregul bloc este de 0,8 ns. La acest timp se mai adaugă

0,2 ns ale blocului INV/ZERO, astfel că pentru sumatorul de ajustare a exponentului rămân 3 ns din cele 4 ns alocate şi acestui nivel pipeline. *Utilizarea unui sumator rapid pe 9 biţi, identic cu cel prezentat în figura 3.3 ar conduce la obţinerea unui timp de propagare pe acest nivel sub 3 ns. Utilizarea însă a unui sumator CLA modificat, de tipul celui din figura 2.11 dar pe 9 biţi, asigură un timp de propagare de 2,5 ns în cazul cel mai defavorabil şi reprezintă varianta optimă ce va fi adoptată în cazul de faţă. Timpul maxim de propagare prin ultimul nivel pipeline va fi deci de 3,5 ns.*

În concluzie, subunitatea de adunare/scădere în virgulă flotantă organizată pe trei niveluri pipeline este capabilă să funcţioneze cu o perioadă de tact de 4 ns, ca şi subunitatea logaritmică, în condiţiile în care au fost utilizate circuite optimizate ce ocupă o suprafaţă minimă pe cip. Cele două subunităţi de calcul ale procesorului aritmetic permit astfel implementarea unei structuri de acumulator, pentru a se putea efectua rapid sume de produse de factori. În Anexa 8 este prezentat un exemplu de rulare al unui algoritm DSP simplu, prin care se efectuează o buclă Livermoore cu 20 de termeni (suma a 20 de produse de câte doi factori).

Cap.4 Rezultatele simulărilor blocurilor şi subblocurilor componente ale procesorului hibrid în virgulă flotantă şi logaritmic

În acest capitol vor fi prezentate rezultatele simulărilor majorității blocurilor şi subblocurilor componente ale celor două subunități de calcul ale procesorului aritmetic hibrid conceput în lucrare. Au fost simulate astfel toate blocurile şi subblocurile componente de pe căile critice de propagare, pentru a confirma contribuțiile aduse de autorul tezei şi pentru a dovedi performanța sistemului de calcul proiectat. De asemenea au fost simulate şi acele blocuri la care contribuţiile aduse s-au referit la optimizarea structurii lor din punct de vedere al suprafeței ocupate pe cip, atunci când aceste blocuri nu reprezentau o cale critică de propagare.

În cazul subunității logaritmice au fost făcute simulări pentru ambele variante prezentate. Au fost astfel evidenţiate avantajele de viteză aduse prin implementarea metodei logaritmului mascat comparativ cu prima variantă şi cu rezultatele obținute de F. S. Lai în [50], [51] şi [107] ce au reprezentat ultimele realizări în domeniul procesoarelor hibride, prezentate de literatura de specialitate.

În cazul subunității de calcul în virgulă flotantă au fost de asemenea simulate toate variantele elaborate, pentru a putea fi depistată structura optimă.

Analizele au fost efectuate pentru acele combinaţii de valori numerice ale operanzilor de intrare care au reprezentat cazurile cele mai defavorabile din punct de vedere al propagării transportului prin structurile proiectate. De asemenea au fost efectuate analize pentru a confirma corectitudinea realizării funcţiei specifice a fiecărui bloc. A fost utilizat simulatorul MicroSim, în ale cărui librării de componente au fost modificaţi timpii de propagare prin diversele porți logice utilizate în conformitate cu datele furnizate de [67], corespunzătoare unei tehnologii CMOS 0,5 μm, adoptată ca şi etalon pentru a permite o mai facilă comparaţie cu circuite similare studiate în referinţele bibliografice. Astfel, timpii de propagare prin diversele porți logice sau de transmisie utilizate au fost considerați după cum urmează: 0,35 ns pentru porți SAU-exclusiv neconvenționale, mai rapide decât cele clasice, 0,2 ns pentru porți ŞI-NU şi SAU-NU cu două intrări, 0,1 ns pentru inversoare CMOS şi pentru tranzistoare de trecere.

Deoarece nu am dispus de un soft profesional pentru simulare după extragere din layout, am încercat să modelez funcţionarea reală a structurilor numerice proiectate prin introducerea unor întârzieri suplimentare pe anumite căi de procesare a informației. Astfel au fost folosiți arbori de inversoare CMOS pentru multiplicarea capabilității de comandă a unor linii de selecție ce comandau un număr mare de grile MOS; de asemenea valoarea capacităților parazite ce se acumulează datorită cascadării porţilor de transmisie din circuite selectoare consecutive a fost limitată prin intercalarea unor circuite latch, realizate tot cu inversoare CMOS.

Datorită numărului mare de porţi logice utilizate la construcţia unui nivel pipeline (de ordinul miilor sau chiar mai mult), nu întotdeauna simulatorul a făcut

față situației, caz în care simularea s-a făcut doar pe blocuri componente, nu şi pe ansamblu. În acest caz, verificarea prin simulare a blocurilor componente a condus la un calcul acoperitor al timpului total de propagare printr-un nivel pipeline deoarece pentru o procesare dată, blocurile componente ale ansamblului pot să nu se găsească simultan în cel mai defavorabil caz.

4.1 Rezultatele simulărilor blocurilor şi subblocurilor componente ale subunității de calcul logaritmice

Subunitatea logaritmică, concepută în prima variantă după metoda descrisă de F. S. Lai în [50], dar folosind algoritmul de conversie de format modificat (paragraful 2.2.2), a fost organizată pe şase niveluri pipeline (figura 2.14). Nivelurile 2 şi 5 ce conțin multiplicatoare de 12×12 biți sunt nivelurile critice întrucât prezintă cel mai mare timp de propagare şi anume 6,3 ns, confirmat în diagrama din figura 4.1.

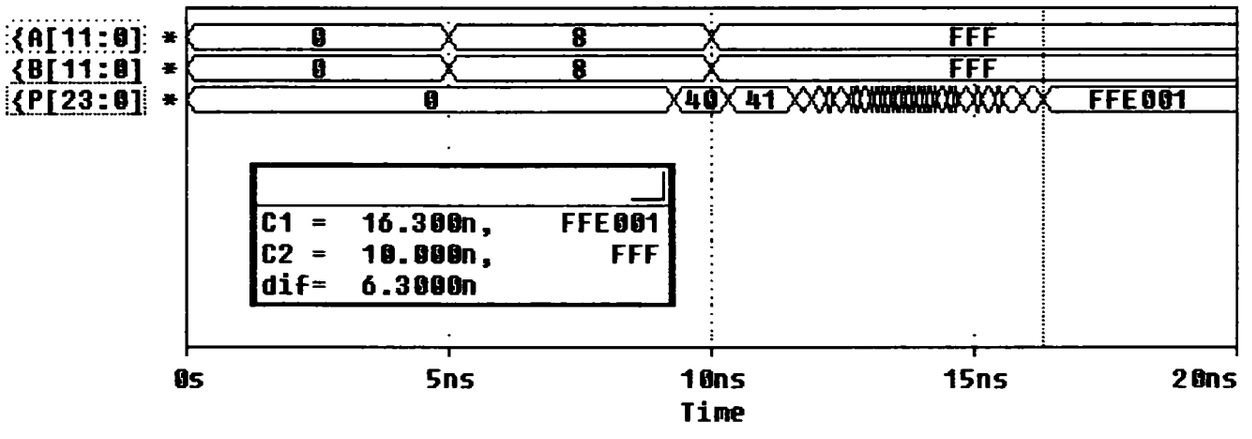


Fig. 4.1 Simularea multiplicatorului de 12×12 biți

Aşa cum se observă în figura 4.1, dacă factorii A şi B de 12 biți au valoarea maximă FFFh, atunci produsul P este reprezentat pe 24 de biți şi are valoarea corectă FFE001h, obținută după 6,3 ns de la aplicarea stimulilor. Toate produsele parțiale având valoarea FFF, fiecare sumator complet din arborele Wallace al ariei de înmulțire se va găsi în cel mai defavorabil caz al său, în care toate cele 3 intrări ale sale au valoarea logică 1. Funcționarea corectă a multiplicatorului se poate constata mai uşor în cazul când operanzii au valoarea 8. Rezultatul este 64 adică 40h şi e obținut într-un timp mai mic de 5 ns.

Următorul nivel pipeline consumator de timp al structurii din figura 2.14 este cel care conține blocul ALU. Pentru a fi obținuți timpi de propagare cât mai mici au fost modificate sau chiar concepute diferite sumatoare rapide pe 32, 16 şi 8 biți. Astfel, dacă sumatorul CLA clasic cu structură modulară pe 32 de biți produce rezultatul UF pentru combinația cea mai defavorabilă de operanzi UD şi UE (justificată în paragraful 2.2.2.4, pag.55, aliniatul 2) în 5,7 ns, aşa cum se observă

în diagrama din figura 4.2.a, sumatorul CLA cu structura blocurilor componente modificată de autorul tezei produce acelaşi rezultat în doar 4,3 ns, aşa cum se observă în diagrama din figura 4.2.b.

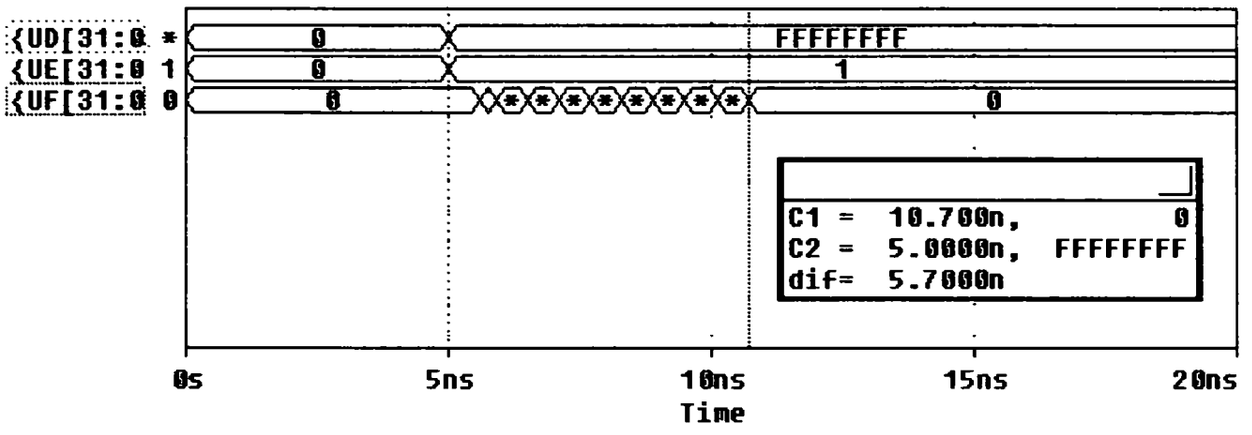


Fig. 4.2.a Simularea sumatorului CLA pe 32 de biţi clasic

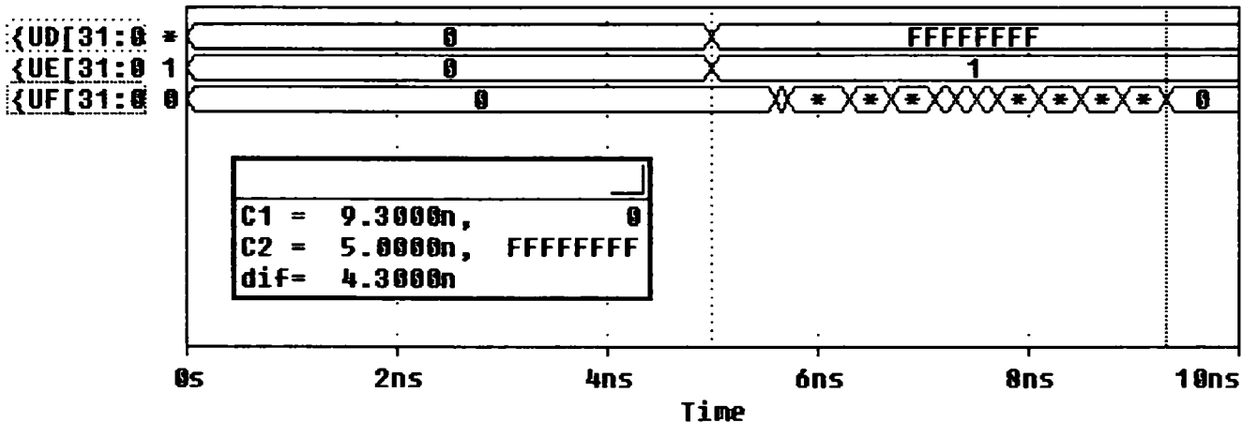


Fig. 4.2.b Simularea sumatorului CLA pe 32 de biţi modificat

Pentru micşorarea în continuare a timpului de producere a rezultatului, a fost conceput un nou tip de sumator pe 32 de biţi ce a avut ca punct de plecare observaţia importantă făcută în paragraful 2.2.2.4, pag. 55, aliniatul 3. Noul tip de sumator este în ansamblu un sumator cu selecţia transportului, dar la care blocurile componente sunt sumatoare de tip CLA pe un număr mai mic de biţi. Din acest motiv au fost analizate şi sumatoare CLA pe 16 şi 8 biţi cu structura blocurilor componente modificată. Aşa cum se observă în diagramele din figurile 4.3.a şi 4.3.b, au fost obţinuţi timpi maximi de propagare de 3,6 ns şi respectiv 2,5 ns. În realitate aceşti timpi sunt cu 0,1 ns mai mici deoarece faptul că transportul iniţial al acestor sumatoare este 0 în cazul cel mai defavorabil (vezi paragraful 2.2.2.4) conduce la o simplificare a blocurilor de tip B' şi B" situate pe rama inferioară a structurii modulare a sumatorului CLA.

Structura optimă a noului sumator cu selecţia transportului, din punct de vedere al vitezei de operare, a fost cea în care blocurile componente erau reprezentate de sumatoare CLA pe 8 biţi. Această structură reprezintă un optim

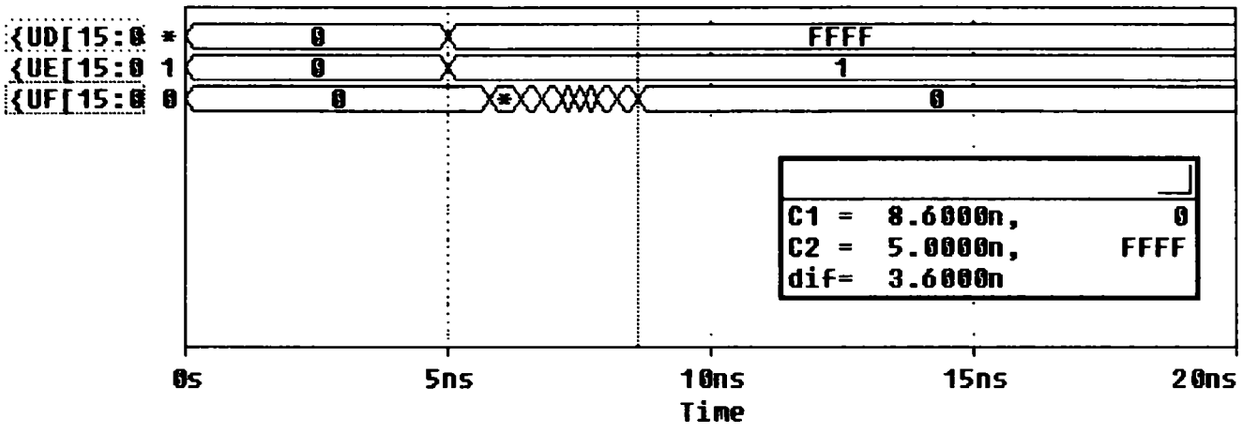


Fig. 4.3.a Simularea sumatorului CLA pe 16 biţi modificat

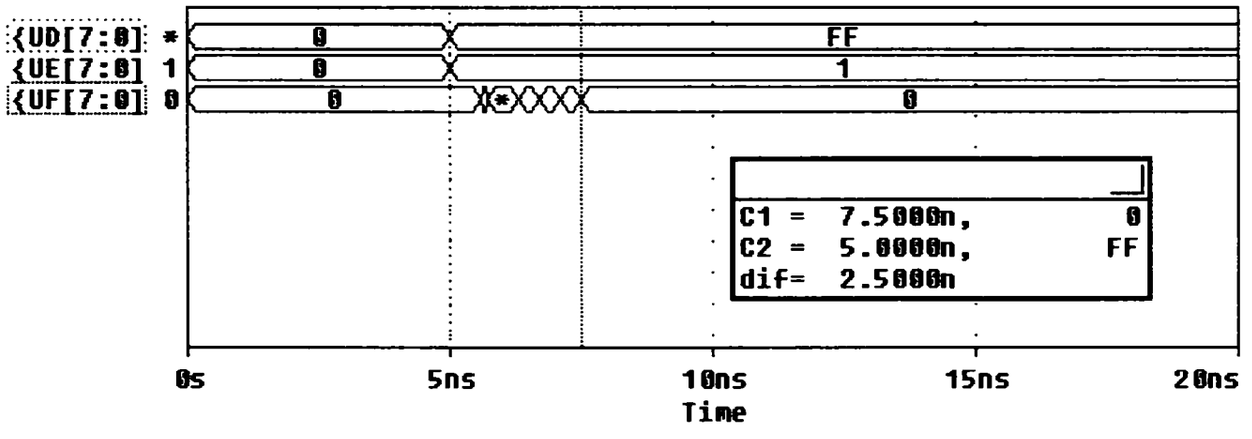


Fig. 4.3.b Simularea sumatorului CLA pe 8 biţi modificat

deoarece intervalul de timp cuprins între generarea transportului furnizat de sumatorul CLA de rangul cel mai mic şi stabilirea celor mai puţin semnificativi 8 biţi ai sumei acoperă aproape în întregime timpul necesar pentru multiplicarea capabilităţii de comandă a liniilor de selecţie a transportului de intrare corect pentru sumatoarele CLA de rang superior. Din acest motiv, timpul de producere al rezultatului complet este cu doar 0,2 ns mai mare decât al unui sumator CLA pe 8 biţi. Aşa cum se observă în diagrama din figura 4.4, acest timp este de 2,6 ns.

Combinaţia de operanzi ce conduce la cel mai defavorabil caz din punct de vedere al propagării transportului prin circuit, rezultată în urma împărţirii cuvintelor ce se însumează în patru secţiuni de câte 8 biţi, a fost găsită pe baza următoarelor constatări:

- sumatorul CLA de rangul cel mai mare (al cărui rezultat se stabileşte cel mai târziu) trebuie să se găsească în cel mai defavorabil caz al său ceea ce corespunde aplicării la intrarea sa a combinaţiei FFh şi 01h;

- sumatoarele CLA care funcţionează cu transport iniţial 0 pot reprezenta o cale critică de propagare, întrucât asigurarea transportului iniţial 1 nu se regăseşte în nici o combinaţie de intrare ce reprezintă cel mai defavorabil caz (vezi aliniatul 2, pag. 55) pentru sumatorul CLA. Pe de altă parte, aceste sumatoare nu trebuie să

genereze transport 1, pentru a se păstra aceeaşi situaţie defavorabilă la sumatorul de rang imediat superior. În consecinţă cel mai defavorabil caz corespunde combinaţiei de cuvinte de intrare 7Fh şi 01h. În compensaţie, liniile de selecţie a rezultatului corect de la fiecare pereche de sumatoare de acelaşi rang, rămân tot timpul pe 0 şi selecţia este foarte rapidă. Aşadar, structura particulară a noului sumator propus de autorul tezei nu permite aflarea simultană în cazul cel mai defavorabil a sumatoarelor componente;

- sumatoarele CLA care funcţionează cu transport iniţial 1 pot reprezenta o cale critică de propagare, deşi nu se încadrează în cel mai defavorabil caz al sumatorului de tip CLA, deoarece generarea transportului de ieşire al sumatorului şi deci selecţia rezultatului corect de la perechea de sumatoare din aval se face cu întârziere. Aşadar şi în acest caz are loc o compensare benefică a întârzierilor de pe liniile de selecţie şi de pe circuitele sumatoare.

În concluzie, simularea se va face pentru seturile de operanzi FF7F7F7Fh şi 01010101h, respectiv FFFFFFFFh şi 00000001h, precum şi pentru o combinaţie între acestea. Aşa cum se observă în diagrama din figura 4.4, ambele seturi conduc la obţinerea unui timp de propagare de 2,6 ns. Orice altă combinaţie între aceste două seturi conduce la obţinerea mai rapidă a rezultatului.

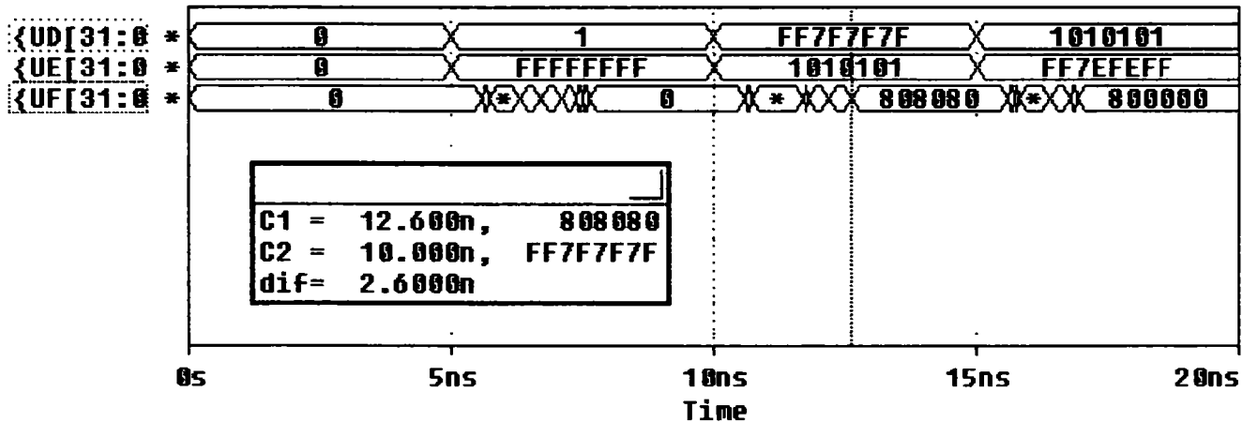


Fig. 4.4 Simularea noului sumator rapid pe 32 de biţi

Etapa următoare a constat în simularea blocului ALU din figura 2.14, adică a circuitului sumator/scăzător prezentat în figura 2.9. Deoarece acest circuit a fost construit cu două sumatoare de tipul celui a cărui simulare a fost prezentată anterior, combinaţia cea mai defavorabilă de operanzi coincide cu cea din situaţia anterioară, dacă se efectuează o operaţie de adunare (/SEL=1, figura 2.9). Dacă se efectuează însă o operaţie de scădere, cazul cel mai defavorabil se obţine pentru rezultat negativ (pentru a fi selectat rezultatul de la blocul Sumator1 cu transport iniţial 0, care funcţionează mai lent), atunci când descăzutul şi scăzătorul iau cele mai mari valori posibile, diferite între ele. Aşa cum se observă în diagrama din figura 4.5, dacă se efectuează scădere (/SEL=0), rezultatul UC are valoarea corectă "–1", adică 80000001h cu bitul de semn inclus, după un timp de 3,3 ns. Şi operaţia de adunare se efectuează tot în 3,3 ns, în cazul cel mai defavorabil, deoarece toate

blocurile selectoare din circuitul din figura 2.9 (Inversor1, Inversor2 şi Selector) sunt incluse în calea critică de propagare.

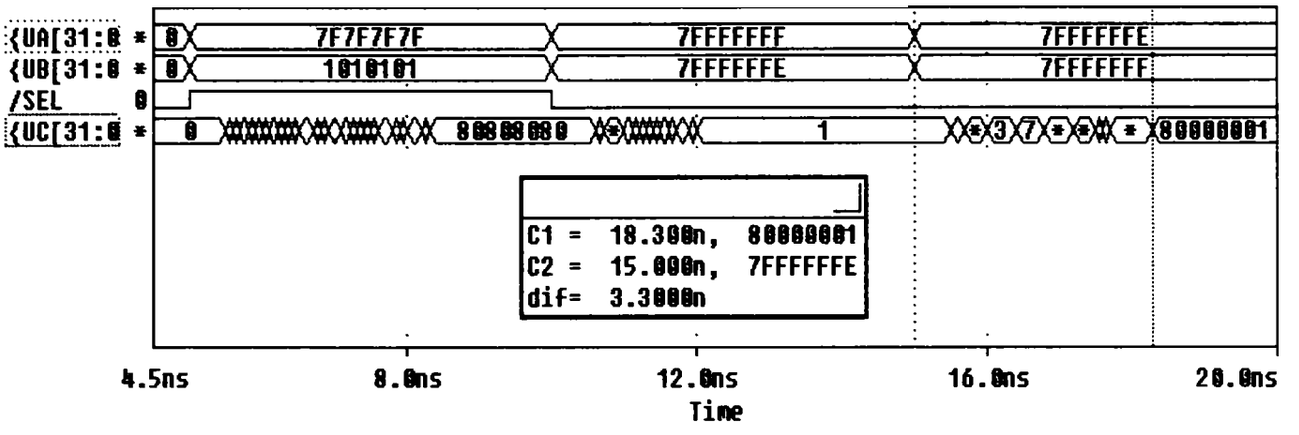


Fig. 4.5 Simularea circuitului de adunare/scădere din componenţa blocului ALU

Deoarece însă la blocul Inversor1 linia "/Sel" este deja setată (odată cu stabilirea tipului operaţiei pe care o execută subunitatea logaritmică) înainte ca circuitele de logaritmare din figura 2.3 să furnizeze cei doi operanzi pentru ALU, întârzierea produsă la multiplicarea capabilităţii de comandă a liniei /Sel, de 0,5 ns, este eliminată. În concluzie, timpul maxim de operare pentru ALU este de 2,8 ns.

Tot în nivelul 3 al structurii pipeline din figura 2.14 mai există blocurile "Sumator CSA+CLA". Aşa cum s-a precizat în paragraful 2.2.2.1, pag. 47, timpul maxim de propagare printr-un bloc CSA este dat de timpul de propagare prin două porţi SAU-exclusiv cascadate, adică 0,7ns. Timpul maxim de propagare prin sumatorul pe 23 de biţi din componenţa aceluiaşi bloc, în cazul cel mai defavorabil este de 2,5 ns, aşa cum se observă în diagrama din figura 4.6.

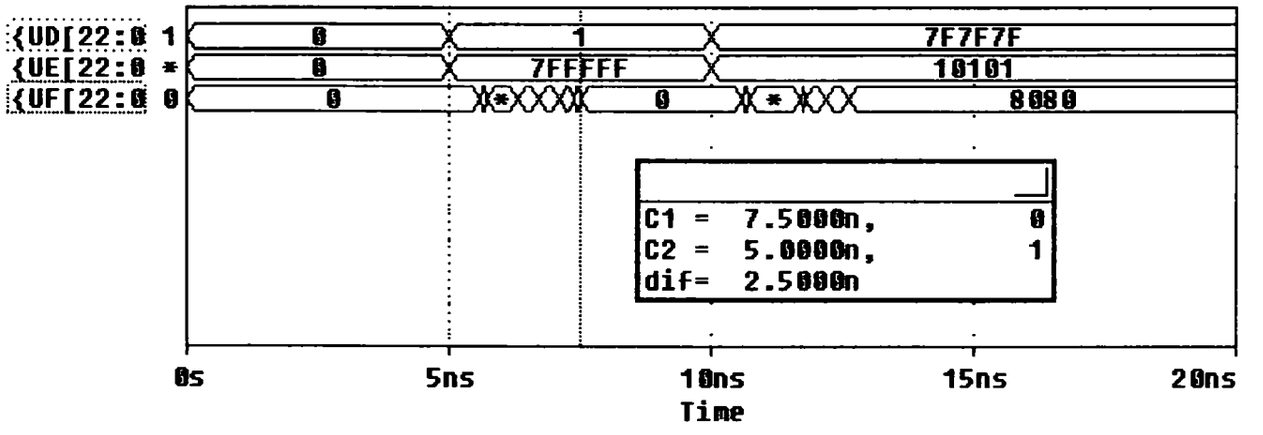


Fig. 4.6 Simularea sumatorului pe 23 de biţi din blocul "Sumator CSA+CLA"

În concluzie, timpul total de propagare prin nivelul pipeline 3 al structurii din figura 2.14 este 0,7+2,5+2,8 = 6 ns, aşa cum s-a precizat în paragraful 2.2.2.5. Astfel, toate blocurile sau subblocurile componente ale structurii pipeline din figura 2.14 au fost simulate, confirmând rezultatele prezentate în paragraful 2.2.2.

În continuare vor fi prezentate rezultatele simulărilor blocurilor şi subblocurilor componente ale subunităţii logaritmice proiectate prin implementarea metodei logaritmului mascat şi a cărei schemă bloc a fost organizată tot pe 6 niveluri pipeline (figura 2.25). În acest caz au fost folosite aceleaşi tipuri de sumatoare performante de concepţie proprie, însă extinse pe un număr mai mare de biţi, după cum s-a precizat în paragraful 2.2.3.3. În plus, au fost folosite blocuri de compresori 4:2 pentru care timpul de propagare este dat practic de timpul de propagare printr-un singur compresor, aşa cum s-a explicat în paragraful 2.2.3.2. Compresorul 4:2 este astfel considerat celula de bază pentru structurile numerice din nivelurile pipeline 2 şi 5, dar el este prezent şi în blocul ALU de pe nivelul 3. Pentru a demonstra funcţionarea corectă a compresorului 4:2 L propus de autorul tezei, au fost efectuate simulări pentru toate combinaţiile posibile ale intrărilor circuitului. Utilizând aceiaşi stimuli, au fost efectuate apoi simulări şi pentru compresorul 4:2 optimizat, prezentat în [72]. Rezultatele obţinute sunt prezentate în diagramele din figurile 4.7.a şi 4.7.b.

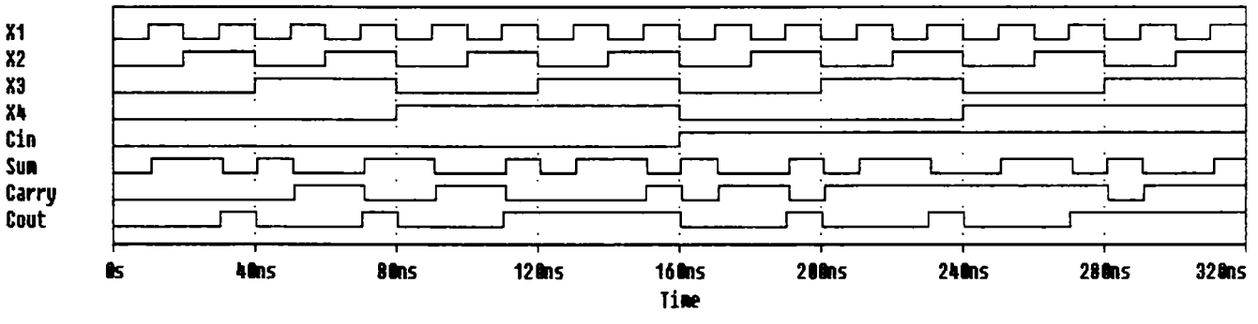


Fig. 4.7.a Simularea compresorului 4:2 L

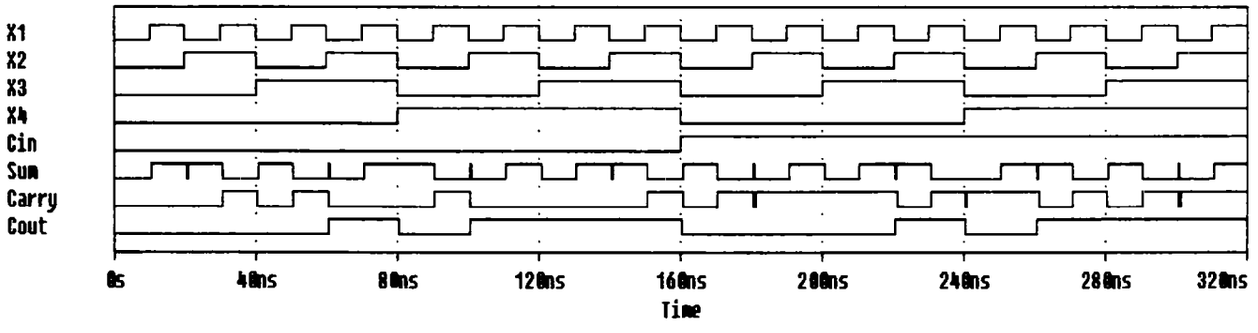


Fig.4.7.b Simularea compresorului 4:2 prezentat în [72]

După cum se observă în aceste diagrame, pe cele cinci linii de intrare X1, X2, X3, X4 şi C_{in} au fost aplicate toate combinaţiile posibile de valori logice. Analiza a fost efectuată în primă fază pe intervale de timp de câte 10 ns începând din momentul aplicării unei noi combinaţii la intrare, adică pe intervale de timp mult mai mari decât timpii de propagare prin circuit. Scopul acestei analize a fost de a verifica funcţiile logice asigurate de circuit, în conformitate cu definiţia dată pentru compresorul 4:2 în paragraful 2.2.3.2. Se constată astfel că ieşirea S_{um} este

aceeaşi în ambele diagrame şi în concordanţă cu funcţia a) enunţată la pag. 70. Ieşirile C_{arry} şi C_{out} prezintă aceeaşi stare logică în ambele diagrame, dacă suma $X1+X2+X3+X4+C_{in}$ este egală cu 0, 1, sau 5, fapt care este conform cu funcţiile d) şi f) ce trebuie asigurate de circuit (pag.70). Dacă suma menţionată este egală cu 2 sau 3, atunci suma $C_{arry}+C_{out}$ este întotdeauna egală cu 1, în ambele diagrame; cu alte cuvinte ieşirile C_{arry} şi C_{out} sunt complementare în ambele circuite. Acest fapt este în concordanţă cu funcţia e) a compresorului 4:2.

Pentru compresorul 4:2 L s-a făcut şi o analiză în termeni de viteză pentru cea mai defavorabilă combinaţie de intrare care va determina propagarea unui transport pe calea critică de propagare a circuitului. Aşa cum se observă în figura 2.23, calea critică de propagare include ieşirea C_{out} a compresorului din amonte (care poate genera un transport după 0,4 ns de la schimbarea stării logice a intrărilor, deci mai târziu decât o poartă SAU-exclusiv de pe intrarea compresorului curent), intrarea C_{in} a compresorului curent şi ieşirea C_{arry} a acestuia. Una din combinaţiile de intrare posibile ce permite generarea unui transport pe această cale este cea utilizată în simularea al cărei rezultat este prezentat în figura 4.8. Se observă că ieşirea C_{arry} devine stabilă după 1,1 ns de la aplicarea stimulilor. Această valoare reprezintă deci timpul maxim de propagare prin compresorul 4:2 L.

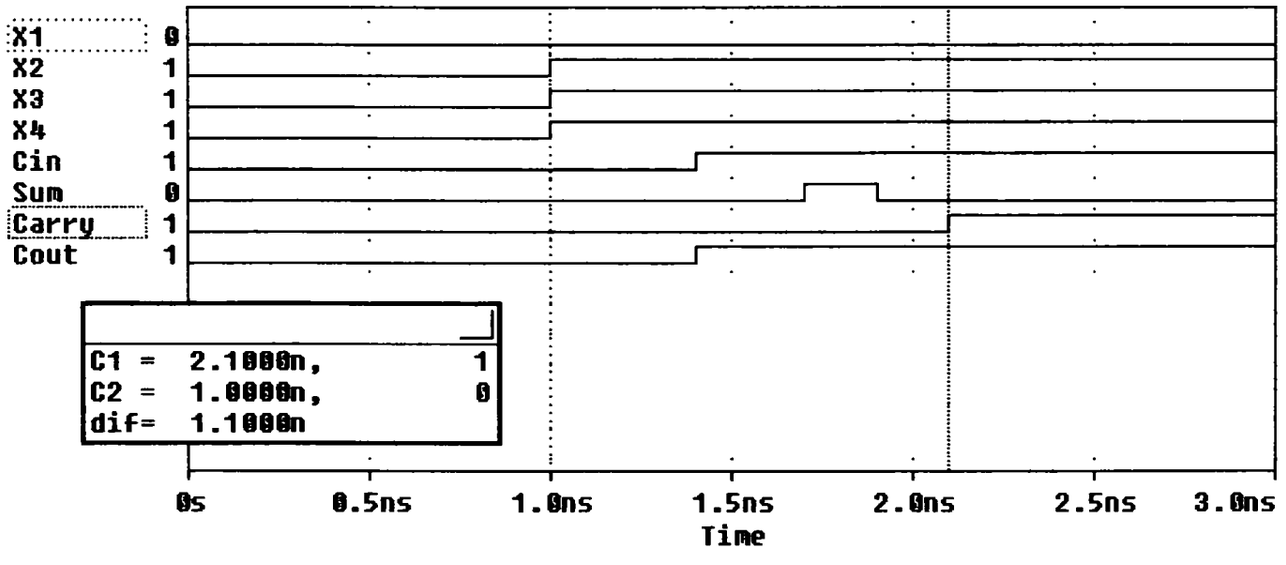


Fig. 4.8 Simularea compresorului 4:2 L în cazul cel mai defavorabil

Următoarea etapă a constat în simularea arborelui Wallace care ocupă întreg nivelul 2 al structurii pipeline în care a fost organizată subunitatea de calcul logaritmică, în cazul implementării metodei logaritmului mascat. Deoarece la construcţia acestui nivel au fost utilizate peste 3000 de porţi logice, porţi de transmisie şi inversoare CMOS, simulatorul de care am dispus nu a mai făcut faţă volumului imens de calcul, motiv pentru care am apelat la un artificiu. Acesta a constat în secţionarea uneia dintre magistralele interne de date, ale cărei terminale au fost extrase la două porturi exterioare, cel de ieşire notat TT[13:0], iar cel de

intrare notat T[13:0]. Astfel, simularea arborelui Wallace s-a efectuat în două etape: în prima etapă a fost găsită valoarea numerică de pe liniile magistralei de ieşire TT[13:0] şi timpul scurs din momentul aplicării stimulilor până când această valoare a devenit stabilă; în a doua etapă au fost aplicaţi aceeaşi stimuli plus încă unul la portul de intrare T[13:0] ce introducea valoarea numerică găsită anterior şi întârzierea aferentă.

Simularea a fost făcută în scopul verificării funcţionării corecte a circuitului. În ce priveşte timpul de propagare în cazul cel mai defavorabil, aşa cum s-a arătat în paragraful 2.2.3.2, pag.69 şi paragraful 2.2.3.3, pag. 74, acesta este de 4 ns. După efectuarea simulării, s-a constatat că întârzierea obţinută în urma propagării informaţiei binare prin structurile hard ce formează al doilea nivel pipeline s-a încadrat în 4 ns, aşa cum se observă în diagramele din figura 4.9.

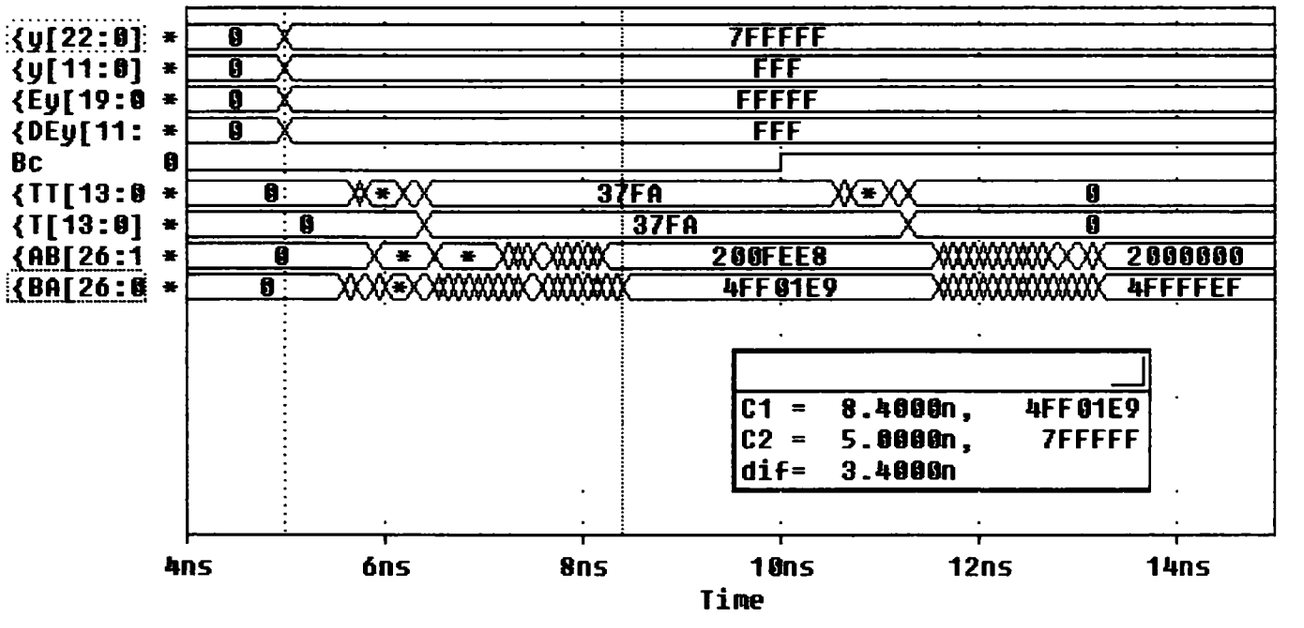


Fig. 4.9 Simularea structurii hard a nivelului pipeline 2 al subunităţii logaritmice concepută prin implementarea metodei logaritmului mascat

Aşa cum se observă în figură, datele de intrare y, y_2 (y[11:0]), Ey şi DEy au valorile maxime, pentru a încadra primul etaj al arborelui Wallace în cazul cel mai defavorabil, dar şi pentru o verificare matematică mai uşoară a rezultatului final, ce s-a dovedit a fi corect. Cuvintele furnizate la ieşire, AB şi BA pe 26 şi respectiv 27 de biţi sunt decalate cu un bit, datorită mecanismului prin care au fost produse, astfel că data AB va primi la dreapta un zero care va deveni LSB şi va avea rangul –27 ca şi LSB al lui BA. Cuvintele AB şi BA sunt cele care trebuie "să mascheze" partea fracţionară a logaritmului operandului aplicat la o intrare a subunităţii logaritmice, dacă valorile de intrare Ey şi DEy ar fi furnizate de memoriile ROMA şi ROMA'. Linia Bc reprezintă bitul de control memorat în locaţiile lui ROMA şi asigură funcţionarea algoritmului descris în paragraful 2.2.3.1.

Structura hard a nivelului pipeline 5 este identică cu cea simulată anterior, diferenţele regăsindu-se doar în valorile datelor aplicate la intrarea sa (v. pag.67).

În continuare va fi prezentată simularea unei secţiuni importante a blocului ALU de pe nivelul pipeline 3 al subunităţii logaritmice. Această secţiune cuprinde circa 1000 de porţi logice ce reprezintă aproximativ 60% din numărul total de porţi utilizate la construcţia blocului ALU. Simularea întregului bloc nu a fost posibilă datorită limitărilor impuse de simulator, dar acest lucru nu a împietat asupra evaluării şi verificării timpului total de propagare prin nivelul pipeline 3 întrucât blocurile care nu au fost incluse în secţiunea simulată prezintă un timp de propagare cunoscut şi verificat prin simulări anterioare. Este vorba de blocul de compresori (1,1 ns - timp maxim de propagare) din figura 2.24 şi cele două blocuri multiplexoare InversorB1 şi InversorB2 care operează în paralel (0.2 ns - timp maxim de propagare). În consecinţă secţiunea simulată cuprinde blocurile principale Sumator1 şi Sumator2 pe 36 de biţi, precum şi blocurile multiplexoare Inversor2 şi Selector împreună cu poarta SAU-NU (fig. 2.24). Pentru a putea fi efectuată simularea, datele aplicate la intrare au fost distribuite pe câte două magistrale întrucât simulatorul nu a suportat stimuli extinşi pe mai mult de 32 de biţi. Deoarece sumatoarele utilizate sunt de acelaşi tip cu cel simulat în figura 4.5 a fost folosită aceeaşi structură a stimulilor pentru cazul cel mai defavorabil. Datorită faptului că un cuvânt de date din cele două furnizate de blocul de compresori nu conţine bit în poziţia LSB, unul dintre stimulii aplicaţi circuitului este reprezentat doar pe 31 de biţi. Acest lucru constituie un avantaj deoarece combinaţia cea mai dezavantajoasă de operanzi la intrările sumatoarelor nu mai este posibilă. Au fost efectuate atât simulări corespunzătoare efectuării operaţiei de adunare (Sop=0), cât şi operaţiei de scădere. Aşa cum se observă în figura 4.10, timpul maxim de propagare a fost găsit de 2,4 ns. Cumulând la această valoare încă 0,2 ns şi 1,1ns se obţin 3,7 ns ceea ce asigură o rezervă de 0,3 ns în raport cu perioada de tact impusă sistemului.

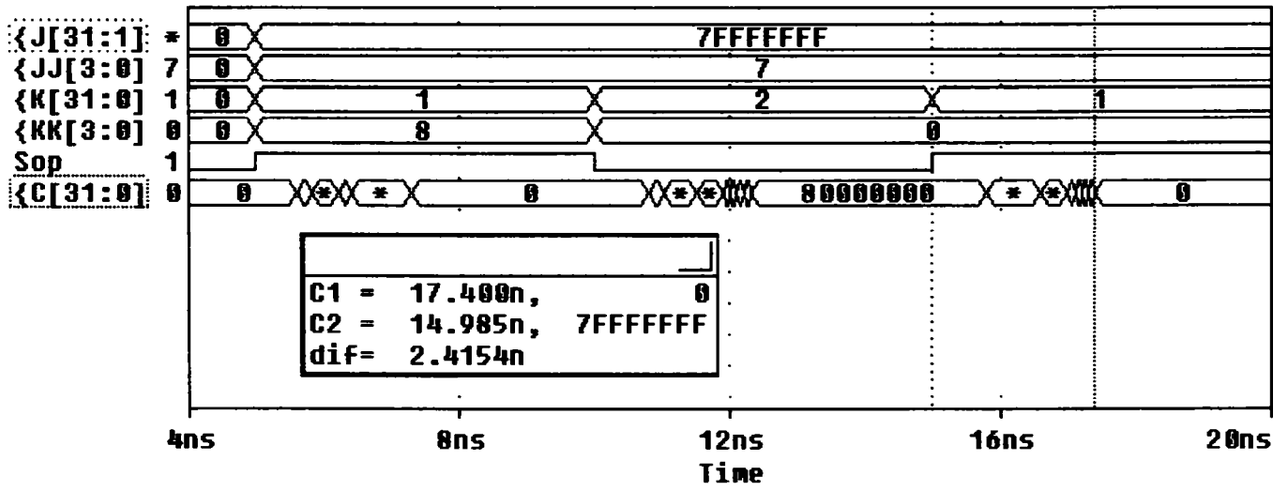


Fig. 4.10 Simularea secţiunii principale a blocului ALU

Ultimul nivel pipeline a cărui funcţionare mai trebuie confirmată prin simulare este nivelul 6, în care se vor însuma două cuvinte de 27 de biţi, din al

căror rezultat vor fi reţinuţi doar cei mai semnificativi 23 de biţi pentru a forma mantisa din formatul în virgulă flotantă simplă precizie. Şi în acest caz combinaţia cea mai dezavantajoasă de operanzi la intrările sumatorului nu mai este posibilă, deoarece operanzii sunt furnizaţi de un bloc de compresori, ca şi în cazul discutat anterior. Din acest motiv, dar şi pentru că se operează pe 27 de biţi în loc de 32, a fost posibilă utilizarea cu succes a unui sumator CLA, de tipul celui din figura 2.11, mai puţin expansiv decât sumatoarele din ALU. În urma simulării a fost obţinut un timp maxim de propagare de 3,6 ns, aşa cum se poate constata în diagrama din figura 4.11.

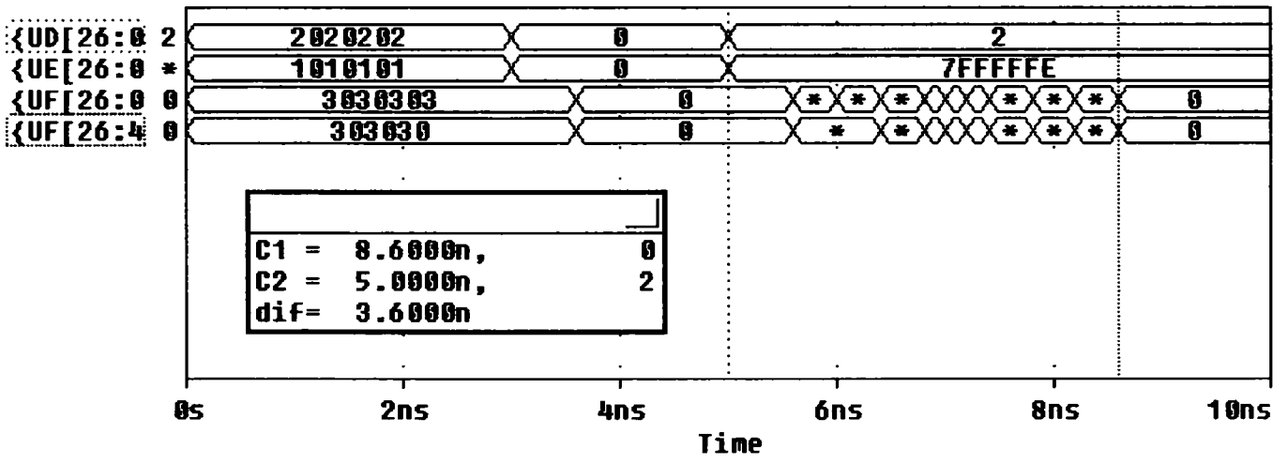


Fig. 4.11 Simularea sumatorului CLA pe 27 de biţi de pe nivelul pipeline 5

4.2 Rezultatele simulărilor blocurilor şi subblocurilor componente ale subunităţii de calcul în virgulă flotantă

La simularea celor trei variante de circuite de aliniere a mantiselor s-a avut în vedere atât determinarea timpului maxim de propagare prin circuit utilizând combinaţia cea mai defavorabilă de operanzi la intrare, cât şi verificarea funcţionării corecte a circuitului, pentru diverse combinaţii de operanzi.

Cuvintele de date ce se aplică acestui circuit sunt exponenţii şi mantisele celor doi operanzi, însă cazul cel mai defavorabil depinde doar de combinaţia de exponenţi, întrucât mantisele s-ar propaga deplasate sau nedeplasate în acelaşi timp prin selectoarele circuitelor "barrel shifter", dacă liniile de selecţie ale acestora ar avea valori logice iniţial cunoscute şi stabile. Deoarece în cazul **variantei 1** circuitele care efectuează cele două diferenţe ale exponenţilor sunt de acelaşi tip cu sumatoarele din blocul ALU al subunităţii logaritmice, combinaţiile cele mai defavorabile de exponenţi vor avea aceeaşi structură. S-a ţinut cont şi de faptul că ambele sumatoare funcţionează cu transport iniţial 1. Combinaţia vizată corespunde setului de valori F7h şi EFh, respectiv FFh şi FFh pentru care au fost afişate întârzierile la propagarea celor două mantise în diagramele din figura 4.12. Aşa cum se observă în diagrame, timpul maxim de propagare este de 2,4 ns în ambele situaţii.

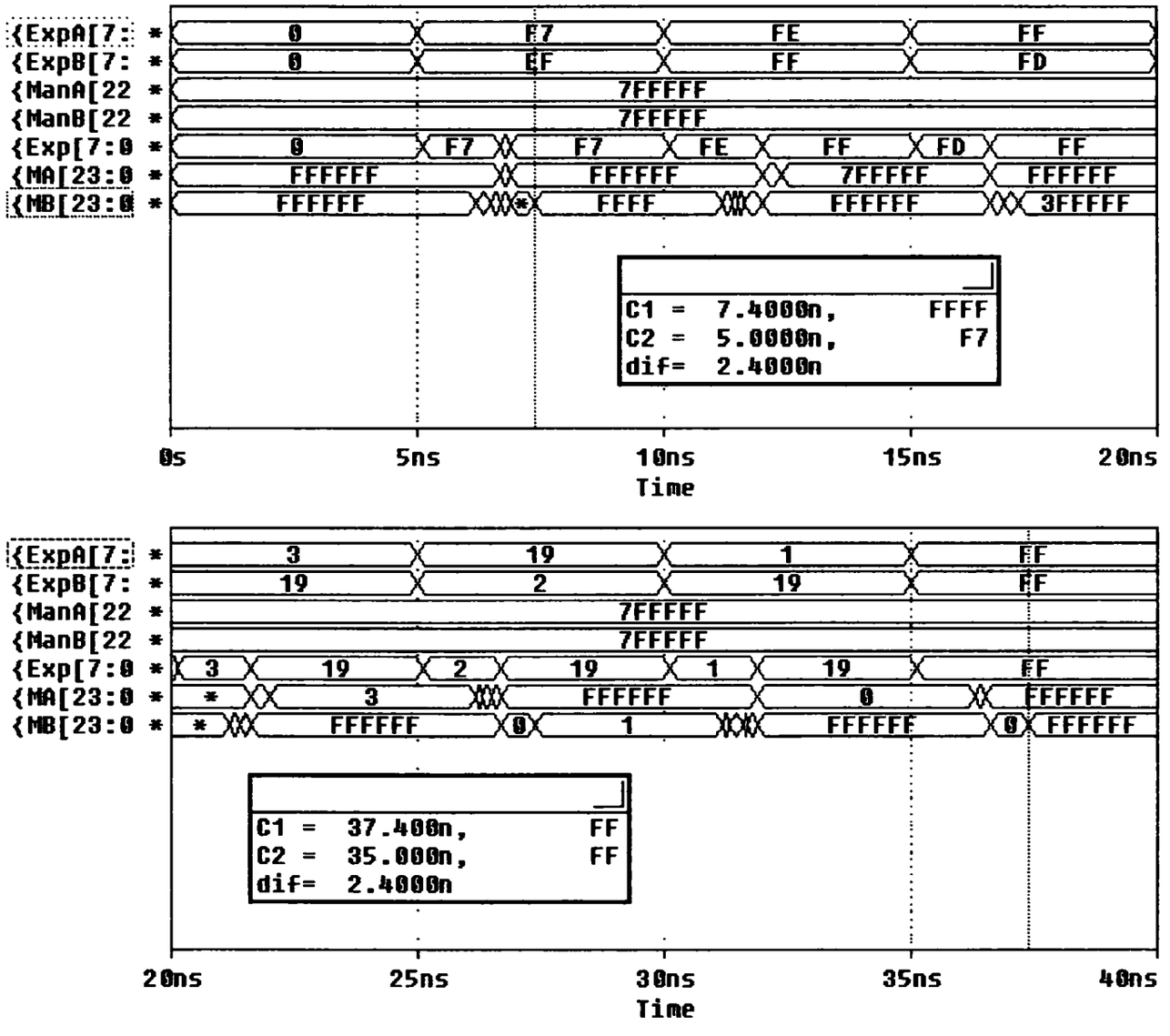


Fig. 4.12 Simularea circuitului de deplasare a mantiselor, varianta 1

În scopul verificării uşoare a deplasării corecte a mantiselor, acestea au fost stabilite ca având toţi biţii egali cu 1. De asemenea, pentru a pune în evidenţă deplasarea mantisei corespunzătoare exponentului mai mic situaţia relativă a celor doi exponenţi a fost alternativ schimbată, la fiecare 5 ns. Aşa cum se observă în figură, au fost obţinute deplasări la dreapta ale mantisei inferioare cu 8, 1 şi 2 biţi în diagrama de sus şi cu 22, 23 şi 24 de biţi în diagrama de jos. Deplasării cu 24 de biţi la dreapta a mantisei inferioare îi corespunde evident anularea biţilor acesteia.

Se mai observă că la ieşire este întotdeauna selectat exponentul mai mare împreună cu mantisa asociată, nedeplasată.

La simularea **variantei 2** a circuitului de aliniere a mantiselor a fost folosit acelaşi fişier de stimuli din motivele practice care au fost explicate anterior. Totuşi în acest caz combinaţia cea mai defavorabilă de exponenţi nu mai este F7h şi EFh deoarece s-a schimbat şi tipul sumatoarelor componente ale sumatorului cu selecţia transportului folosit pentru calcularea diferenţei exponenţilor. Fiind folosite

sumatoare complete pe 1 bit cascadate, combinaţia cea mai defavorabilă de exponenţi este FFh şi FEh ce corespunde calculării sumei FFh+01h+1, ce reprezintă cel mai defavorabil caz de la sumatorul simplu. Această combinaţie este de asemenea furnizată de fişierul de stimuli folosit. Ea conduce la obţinerea unei întârzieri maxime de 3,7 ns aşa cum se observă în diagramele din figura 4.13.

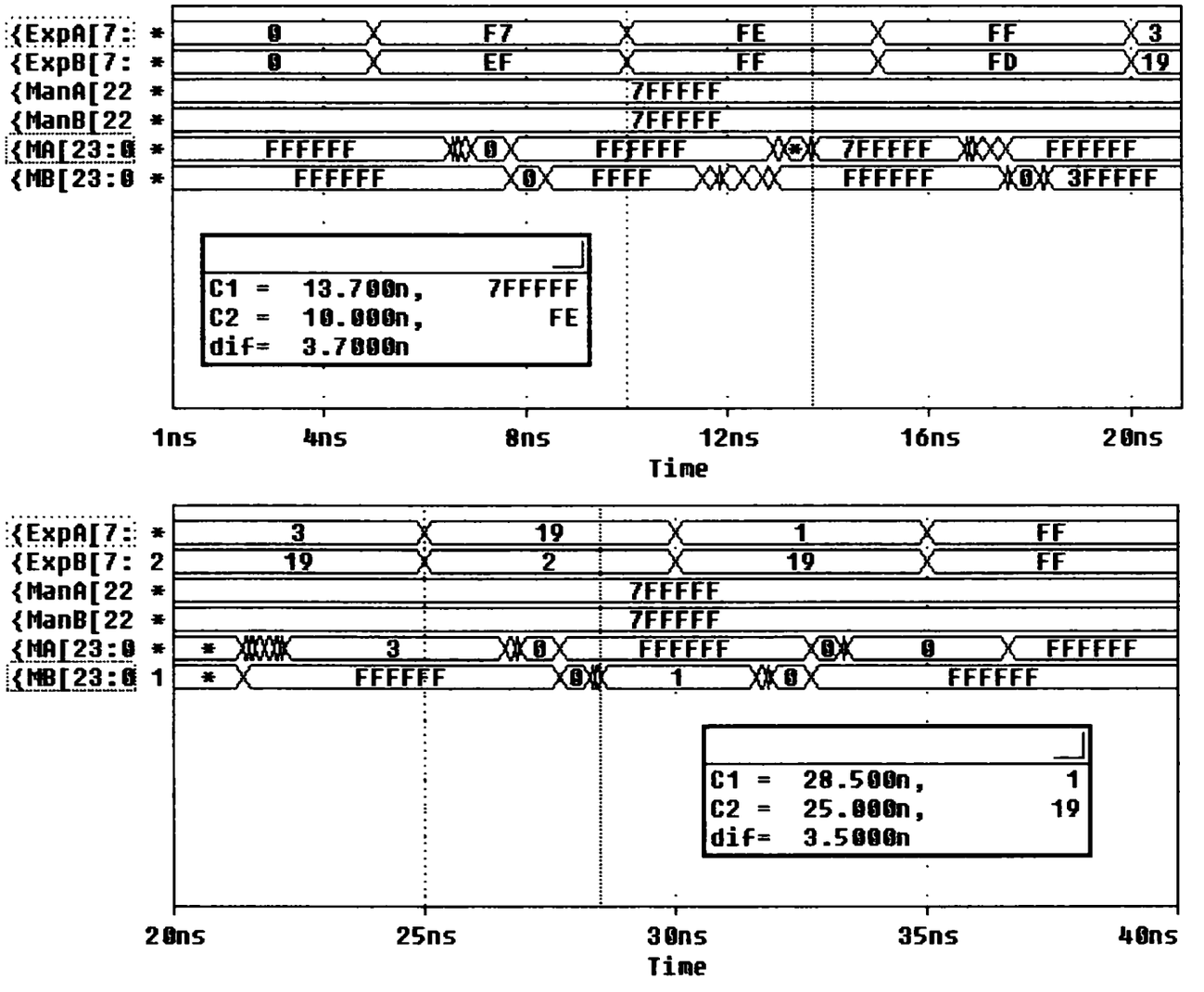


Fig. 4.13 Simularea circuitului de deplasare a mantiselor, varianta 2

De asemenea se constată că şi acest circuit funcţionează corect, iar întârzierea maximă produsă este sub 4 ns. El este deci mai potrivit a fi utilizat decât cel anterior deoarece ocupă o suprafaţă mai mică pe cipul de siliciu şi întruneşte condiţiile de viteză impuse.

Elaborarea **variantei 3** a circuitului de deplasare a mantiselor, a condus însă la obţinerea optimului, în sensul că a permis cea mai semnificativă economie de suprafaţă comparativ cu prima variantă (aproape 50%), iar întârzierea maximă obţinută a fost de 4 ns, aşa cum se observă în diagrama din figura 4.14.

Deoarece s-a utilizat acelaşi tip de circuit pentru efectuarea celor două diferenţe ale exponenţilor, ca şi în cazul variantei 2, combinaţia cea mai defavorabilă este identică cu cea din acest caz.

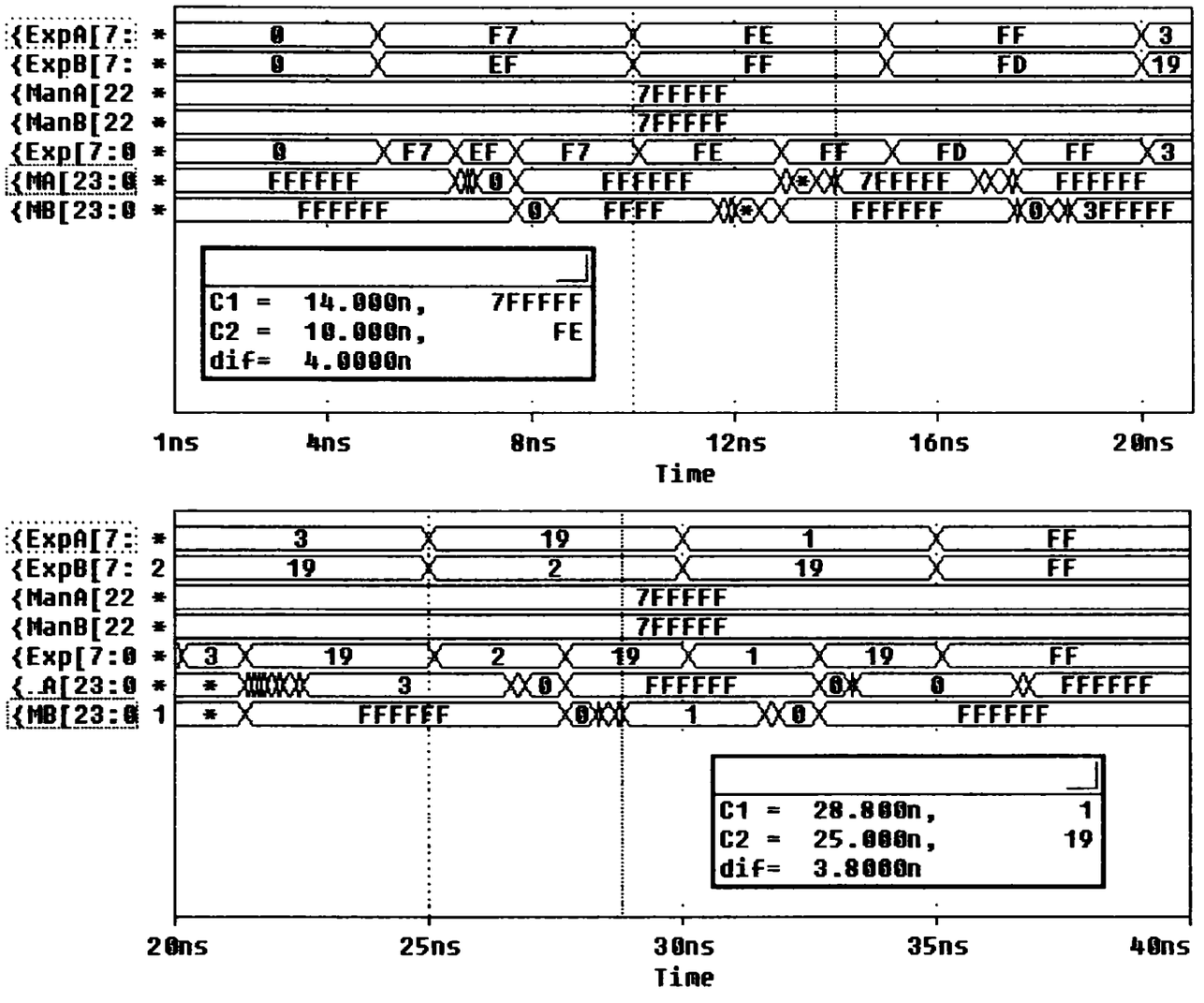


Fig. 4.14 Simularea circuitului de deplasare a mantiselor, varianta 3

Din diagramele prezentate în figura 4.14 se poate constata funcţionarea corectă a circuitului. A fost folosit acelaşi fişier de stimuli ca şi în simulările anterioare.

În continuare au fost efectuate simulări pentru stabilirea variantei optime de circuit sumator/scăzător cu care este realizat blocul ALU din subunitatea de calcul în virgulă flotantă, a cărui schemă bloc a fost prezentată în figura 3.6. Viteza acestui circuit depinde în principal de tipul celor două sumatoare pe 25 de biţi din componenţa sa. În **varianta 1** de circuit sumator/scăzător, se utilizează sumatoare performante a căror schemă bloc a fost descrisă în figura 3.8 şi sunt similare cu cele utilizate în blocul ALU al subunităţii logaritmice. Din acest motiv structura stimulilor urmează acelaşi tipar. Datorită complexităţii sporite a acestui bloc verificările vor fi efectuate pentru toate cele 8 combinaţii posibile din punct de vedere al semnului operanzilor şi al operaţiei care se execută. În cazul în care se efectuează adunări propriu-zise combinaţia cea mai defavorabilă este FF7F7Fh şi 010101h, iar în cazul în care se efectuează scăderi propriu-zise, această combinaţie

este FF7F7Fh şi FEFEFEh (/FEFEFEh=010101h, în care operatorul "/" este operatorul negaţie). Timpii care sunt obţinuţi prin simulare includ un interval de timp de 1,1 ns necesar pentru stabilirea valorii logice a unor linii de selecţie de la intrarea circuitului sumator/scăzător. Valoarea acestor linii nu depinde decât de semnele celor doi operanzi precum şi de semnul operaţiei care se efectuează (0 pentru adunare şi 1 pentru scădere). În consecinţă stabilirea valorii logice a acestor linii, cât şi multiplicarea capabilităţii lor de comandă poate fi realizată în nivelul pipeline anterior, dacă se utilizează tranzistoare dimensionate corespunzător în latch-urile corespondente dintre primul şi al doilea nivel pipeline.

În urma simulării circuitului în cazurile cele mai defavorabile, a fost obţinut un timp maxim de propagare de 4 ns, aşa cum se poate constata în diagramele din figura 4.15. Având în vedere observaţia anterioară, timpul real ce poate fi obţinut în cazul cel mai defavorabil este de 2,9 ns.

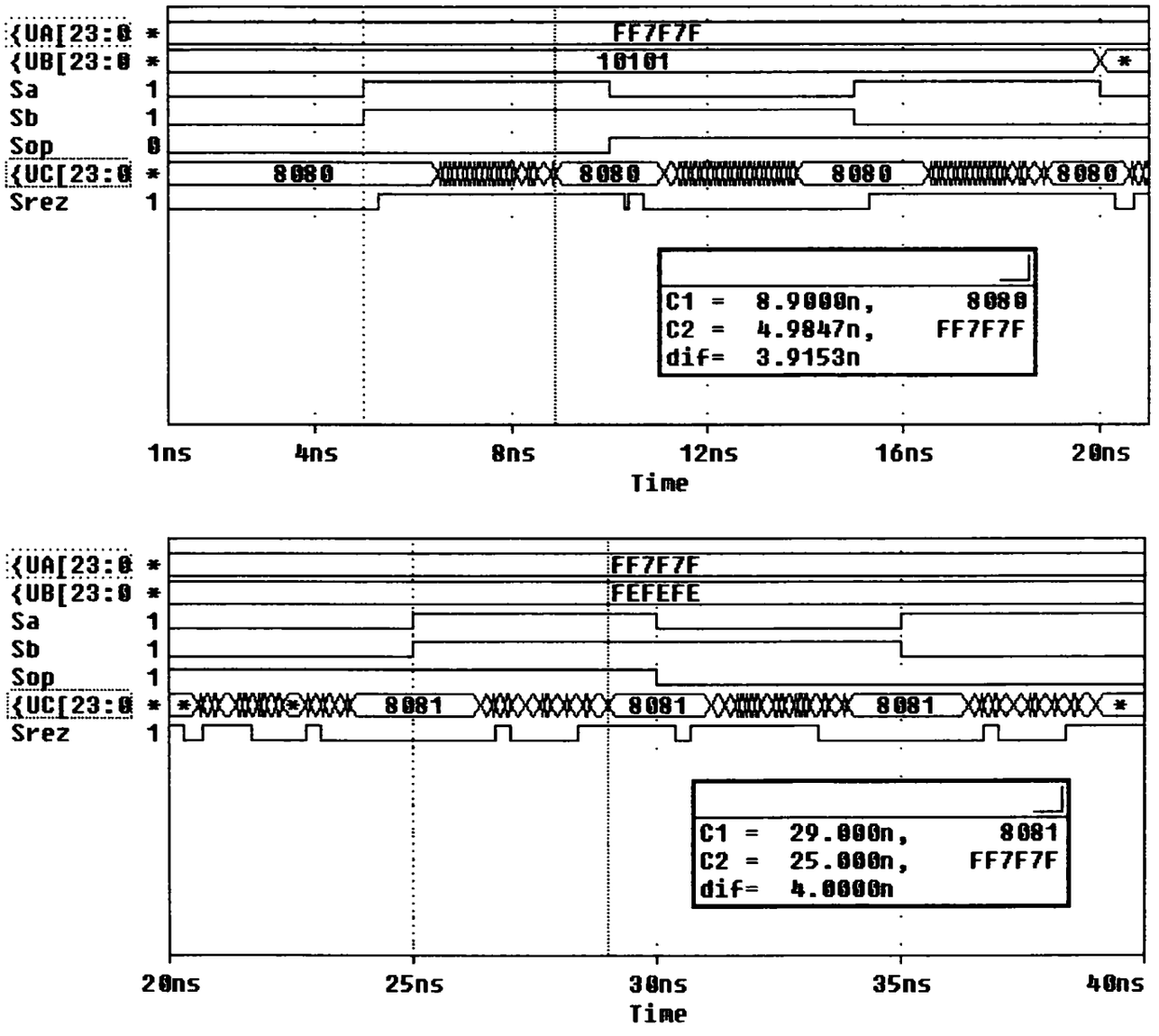


Fig. 4.15 Simularea variantei 1 a circuitului sumator/scăzător al subunităţii de calcul în virgulă flotantă

Stimulii aplicaţi circuitului au fost astfel construiţi încât în primele patru intervale de câte 5 ns se efectuează adunări propriu-zise, iar în ultimele patru intervale se efectuează scăderi propriu-zise, în conformitate cu regulile enunţate în paragraful 3.2, pag. 86. Valoarea absolută a operanzilor este menţinută aceeaşi, în schimb semnul lor se modifică periodic, ca şi semnul operaţiei, pentru a obţine toate cazurile posibile de operare. Analizând diagramele din figura 4.15, se poate constata corectitudinea rezultatului obţinut şi a semnului acestuia, Srez. Se mai poate constata că bitul Srez este întotdeauna stabilizat înaintea rezultatului corespondent şi că rămâne stabil până la următorul moment în care are loc schimbarea stimulilor aplicaţi la intrarea circuitului.

Cazurile cele mai defavorabile de la adunarea propriu-zisă, prezentate în diagrame, corespund însă unei situaţii de supradepăşire mantisă, fapt indicat de bitul MSB de la blocul Sumator1 al circuitului din figura 3.6.

În **varianta 2** de circuit sumator/scăzător, am utilizat sumatoare CLA modificate, de tipul celui descris în figura 2.11, dar pe 25 de biţi (inclusiv bitul de semn). Din acest motiv structura stimulilor se modifică corespunzător cu cel mai defavorabil caz de la acest tip de sumator. Verificările vor fi efectuate pentru toate cele 8 combinaţii posibile din punct de vedere al semnului operanzilor şi al operaţiei care se execută. În cazul în care se efectuează adunări propriu-zise combinaţia cea mai defavorabilă este FFFFFFh şi 000001h, iar în cazul în care se efectuează scăderi propriu-zise, această combinaţie este FFFFFFh şi FFFFFEh (/FFFFFEh=000001h, în care operatorul "/" este operatorul negaţie). Din acelaşi motiv menţionat anterior, timpul de propagare real prin blocul ALU din nivelul pipeline 2 este mai scurt cu 1,1 ns. Aşa cum se observă în diagramele din figura 4.16, timpul maxim obţinut prin simulare a fost de 5,1 ns, ceea ce corespunde unui timp real de propagare prin nivelul pipeline 2 al subunităţii în virgulă flotantă de 4 ns.

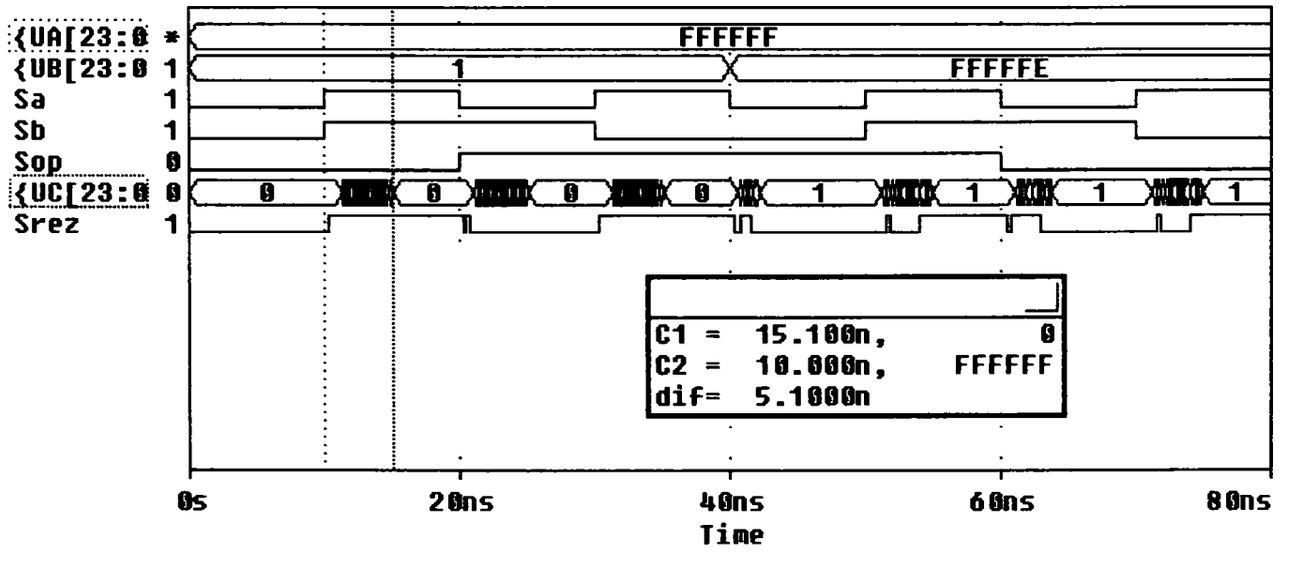


Fig. 4.16 Simularea variantei 2 a circuitului sumator/scăzător al subunităţii de calcul în virgulă flotantă

Analizând diagramele din figura 4.16, se poate constata corectitudinea rezultatului obţinut şi a semnului acestuia, Srez. Cazul cel mai defavorabil corespunde unei adunări propriu-zise, în care este selectat întotdeauna rezultatul furnizat de blocul Sumator1 din figura 3.6 care funcţionează cu transport iniţial 0. Bitul MSB al acestuia va indica însă supradepăşire mantisă, dar acest fapt nu influenţează generarea corectă a bitului de semn al rezultatului.

În urma rezultatelor obţinute la simulare, s-a constatat că varianta a doua de circuit de adunare/scădere reprezintă optimul întrucât este mai puţin expansivă în suprafaţă pe cip, iar timpul maxim de propagare pe care îl prezintă se încadrează în cele 4 ns impuse de subunitatea logaritmică.

În continuare vor fi prezentate rezultatele simulărilor celui de-al treilea nivel pipeline al subunităţii de calcul în virgulă flotantă. În figura 3.17 este prezentat rezultatul simulării blocului NZS din figura 3.9 care efectuează contorizarea zerourilor semnificative ale rezultatului furnizat de ALU în cazul efectuării unei operaţii de scădere.

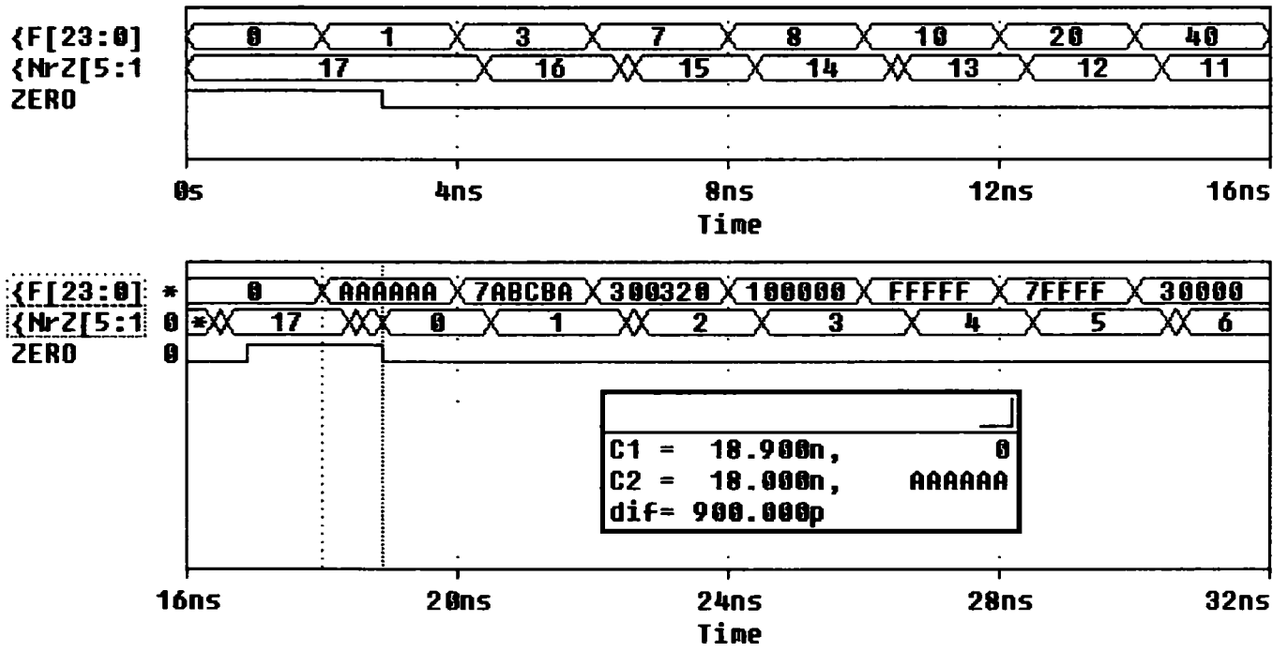


Fig. 4.17 Simularea numărătorului zerourilor semnificative ale diferenţei mantiselor

După cum se observă în aceste diagrame, au fost efectuate simulări pentru cuvinte de date ce conţin 23÷17 (17h÷11h) şi respectiv 0÷6 (00h÷06h) zerouri semnificative, rezultatul fiind redat corect în toate situaţiile. În cazul în care toţi biţii diferenţei mantiselor sunt egali cu zero, normalizarea nu mai are sens, acest fapt fiind indicat prin fanionul "zero" care va invalida rezultatul furnizat la ieşirea blocului NZS, care în acest caz ar genera valoarea maxim posibilă, adică 23 (17h).

Timpul maxim de propagare prin circuit a fost găsit de 0,9 ns. Diferența de 0,1ns față de valoarea evaluată teoretic în paragraful 3.3, pag.92, e dată de inversorul CMOS care a fost introdus pentru setarea pe 1 logic a fanionului "zero" atunci când se obține rezultat nul la ieşirea blocului ALU.

În continuare a fost simulat circuitul compus din blocurile BS ("barrel shifter") şi NRZ din figura 3.9 care deplasează rezultatul diferență furnizat de ALU, în scopul obținerii unei mantise normalizate. Deoarece ansamblul format din aceste două blocuri nu se găseşte pe calea critică de propagare, aşa cum s-a arătat în paragraful 3.3, simularea acestui circuit nu a urmărit decât verificarea funcționării sale corecte. În diagramele din figura 4.18 sunt prezentate rezultatele acestor simulări.

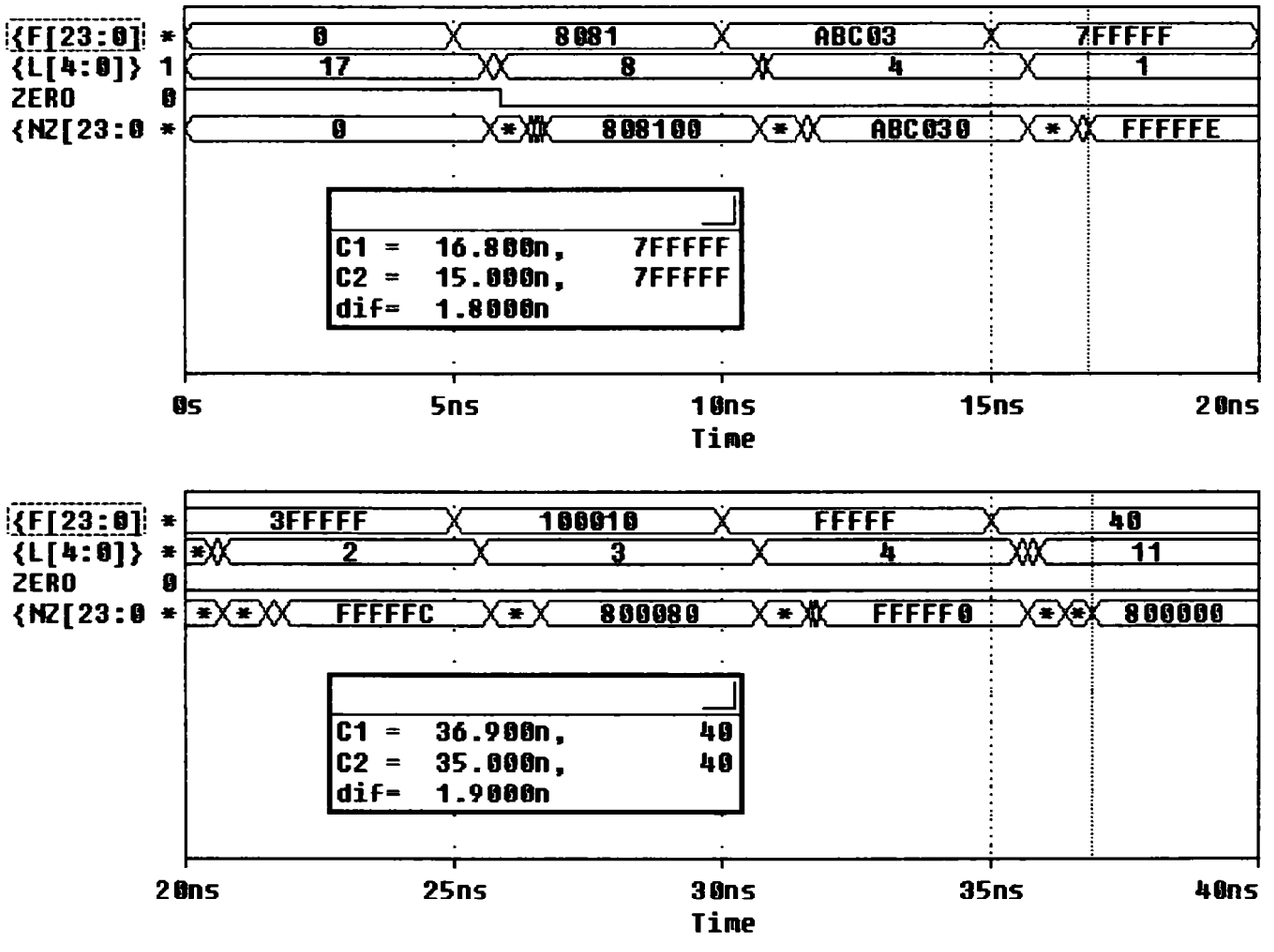


Fig. 4.18 Simularea circuitului de normalizare a rezultatului diferență furnizat de blocul ALU al subunității de calcul în virgulă flotantă

În această figură, F este rezultatul diferență furnizat de ALU, L este cuvântul pe 5 biți generat de blocul NZS, iar NZ reprezintă mantisa normalizată generată la ieşire. După cum se poate constata, toate datele numerice generate la ieşire au cel mai semnificativ bit egal cu 1 şi poartă amprenta cuvintelor din care provin. În fiecare cuvânt de ieşire există un număr de zerouri în pozițiile cel mai puțin semnificative egal cu valoarea numerică a cuvântului generat de blocul NZS.

Întârzierea maximă obținută pentru cazurile aleatoare simulate este de 1,9 ns şi se apropie de valoarea maximă estimată teoretic, de 2,1 ns, care oricum nu afectează întârzierea de pe calea critică de propagare din nivelul pipeline 3. Timpul maxim de propagare prin acest nivel se obţine prin cumularea timpilor maximi obținuți în diagramele din figurile 4.17 (pentru blocul NZS) şi 4.3.b (pentru blocul AJUST_EXP din figura 3.9), precum şi a timpului de propagare prin multiplexorul INV/ZERO (fig. 3.9) de 0.2 ns. Se obține astfel un timp total de 3,6 ns, în condițiile în care suprafața necesitată pentru implementarea pe cip este şi în acest caz minimă.

Concluzii, contribuţii şi perspective

În lucrarea de faţă a fost prezentată o soluţie proprie privind concepţia unităţii de execuţie numerică a unui procesor aritmetic performant, care implementează cele patru operaţii aritmetice elementare şi care operează în simplă precizie. Pe baza analizelor efectuate în capitolul 1 al tezei cu privire la utilizarea metodelor numerice cunoscute de procesare a informaţiei, s-a arătat la începutul capitolului 2 şi ulterior s-a demonstrat practic prin proiectare propriu-zisă (capitolele 2 şi 3) şi s-a verificat prin simulare (capitolul 4) că se poate obţine o viteză de operare foarte mare pentru un procesor aritmetic, atunci când se adoptă o structură hibridă pentru acesta. Noua structură a fost obţinută prin includerea unei subunităţi de calcul logaritmice pentru efectuarea operaţiilor de înmulţire şi împărţire care prezintă intrări şi ieşiri în format virgulă flotantă şi a unei subunităţi de calcul în virgulă flotantă pentru efectuarea operaţiilor de adunare şi scădere.

Performanţele în ce priveşte viteza de operare, obţinute prin utilizarea aritmeticii logaritmice, comparativ cu sistemele de calcul în virgulă flotantă sunt următoarele:

- deşi o operaţie singulară de înmulţire se efectuează de 2,5÷3 ori mai lent decât una în virgulă flotantă, în contextul utilizării procesorului hibrid în aplicaţii DSP, viteza de operare este aceeaşi datorită structurii pipeline în care este organizat;
- operaţia de împărţire, efectuată chiar şi singular, este de circa 3 ori mai rapidă decât una în virgulă flotantă;
- subunitatea logaritmică permite şi implementarea cu uşurinţă a unor operaţii de extragere a rădăcinii pătrate, de ridicare la orice putere reală, sau de logaritmare în orice bază, care se pot efectua mult mai rapid decât în virgulă flotantă (de 5÷10 ori).

Evaluarea cea mai potrivită a performanţelor sistemului de calcul proiectat se face însă prin compararea cu ultimele realizări prezentate în literatura de specialitate în domeniul procesoarelor aritmetice hibride, care aparţin lui F. S. Lai şi E. Wu. Faţă de aceste referinţe, similare în performanţe, procesorul aritmetic hibrid de concepţie proprie este de 1,6 ori mai rapid. Obţinerea acestor performanţe a fost posibilă datorită unui şir de contribuţii teoretice şi aplicative, pe care autorul tezei le-a adus prin elaborarea şi implementarea unor metode specifice de operare numerică.

Ca şi contribuţii teoretice au fost considerate şi metodele originale aplicate în cazurile particulare de proiectare a unor blocuri funcţionale concrete, dar care prezintă un grad de generalitate ce permite utilizarea lor şi în alte aplicaţii de proiectare hardware. Contribuţiile aduse de autorul tezei pe parcursul acesteia vor fi menţionate în continuare.

A. Contribuţii teoretice:

1. Introducerea formatului LNS simplă precizie cu deplasament, ce a permis modificarea parţială a algoritmului descris şi utilizat de F. S. Lai în [50] şi [107] (v.

paragraful 2.2.1, relaţiile 2.14-2.15 şi explicaţiile aferente). Partea întreagă a logaritmului unui operand are astfel inclusă valoarea 127, ceea ce asigură operanzi de intrare în blocul ALU al subunităţii logaritmice, întotdeauna pozitivi. Utilizarea formatului LNS cu deplasament a condus la economie de hard prin eliminarea unor sumatoare pentru exponenţi şi prin simplificarea blocului ALU şi deci la creşterea vitezei de operare a acestuia.

2. Introducerea metodei "logaritmului mascat" ca metodă de creştere a vitezei de operare în cazul utilizării unui algoritm oarecare nerecursiv ce presupune calculul unei sume de produse de factori şi de alţi termeni distincţi, prin considerarea tuturor termenilor ca şi pseudo-produse parţiale şi includerea lor într-un arbore Wallace (v. paragraful 2.2.3.1).

3. Îmbunătăţirea algoritmului de interpolare liniară între valori memorate succesive ale unor funcţii continue nemonotone, prin eliminarea operaţiilor de scădere a unor termeni (prin memorarea directă a complementului de doi a scăzătorului) sau a unor produse de factori corespunzătoare porţiunilor coborâtoare ale funcţiilor (în urma translatării cu o locaţie pe aceste porţiuni a conţinutului memoriilor ROM utilizate) (v. paragraful 2.2.3.1, relaţiile 2.19, 2.20, fig. 2.16 şi explicaţiile aferente). În aplicaţia concretă, acest lucru a condus la creşterea nemijlocită a vitezei de operare a circuitelor de logaritmare şi antilogaritmare cât şi la obţinerea unei structuri hard identice pentru cele două circuite, diferenţele regăsindu-se în conţinutul memoriilor ROM utilizate).

4. Demonstrarea posibilităţii de reducere a erorilor de conversie la logaritmare şi antilogaritmare, prin adăugarea unor valori de corecţie în anumite locaţii ale memoriilor ROM utilizate (v. paragraful 2.2.4, fig. 2.26 şi explicaţiile aferente).

5. Demonstrarea posibilităţii de reducere a dimensiunii structurilor numerice care implementează algoritmi de interpolare liniară pe baza calculării erorii de trunchiere cumulate în cazul eliminării celor mai puţin semnificative poziţii de biţi din arborele Wallace folosit (v. paragraful 2.2.4, explicaţiile referitoare la fig 2.19).

6. Redefinirea compresorului 4:2 pe baza ieşirilor sale redundante, ceea ce a permis lărgirea familiei de circuite din care face parte (v. paragraful 2.2.3.2).

7. Corectarea formulelor date de W. Stallings în [80] care permit determinarea timpului maxim de generare a rezultatului la sumatoarele cu anticiparea transportului şi cu selecţia transportului (v. paragraful 1.1, relaţiile 1.9, 1.10 şi explicaţiile aferente).

B. Contribuţii aplicative:

1. Arhitectură (inclusiv organizare pe niveluri pipeline în două variante) de concepţie proprie a subunităţii logaritmice (v. fig. 2.3, fig. 2.14 şi fig. 2.25).

2. Determinarea dimensiunii memoriilor ROM utilizate la calculul logaritmului şi antilogaritmului (v. paragraful 2.2.2.1).

3. Topologie de concepţie proprie pentru circuitul sumator/scăzător, obţinută prin schimbarea amplasamentului a două blocuri selectoare din structura clasică, ce

a condus la scurtarea căii critice de propagare prin circuit (v. fig. 2.9 şi explicaţiile aferente).

4. Modificarea structurii clasice a blocurilor componente ale sumatorului cu anticiparea transportului (CLA) cu structură modulară, ce a condus la creşterea vitezei de generare a rezultatului şi la un număr mai mic de porţi logice utilizate (v. fig. 2.10.a, fig. 2.10.b, fig. 2.10.c. fig. 2.11 şi explicaţiile aferente).

5. Conceperea unui program scris în limbaj C pentru depistarea celui mai defavorabil caz de propagare a transportului prin sumatorul de tip CLA (v. pag. 55, aliniatul 2 şi Anexa 3).

6. Conceperea unui nou tip de sumator cu selecţia transportului organizat pe două niveluri ierarhice, la care blocurile componente sunt sumatoare de tip CLA (v. paragraful 2.2.2.4, fig. 2.11, fig. 2.12 şi explicaţiile aferente)

7. Definirea căilor de selecţie prioritare la sumatorul menţionat anterior, ce a condus la o creştere suplimentară a vitezei acestuia (v. fig. 2.13 şi explicaţiile aferente).

8. Găsirea şi explicarea celui mai defavorabil caz la propagarea transportului prin sumatorul menţionat anterior (v. paragraful 4.1, pag. 97-98).

9. Construcţia nivelului pipeline 2 al subunităţii logaritmice prin implementarea metodei logaritmului mascat şi a metodei de interpolare liniară îmbunătăţite (v. paragraful 2.2.3.1, fig. 2.17, fig. 2.18, fig. 2.19, fig. 2.20 şi explicaţiile aferente).

10. Conceperea blocului ALU al subunităţii logaritmice în cazul implementării metodei logaritmului mascat (v. paragraful 2.2.2.3, relaţiile 2.25÷2.29, fig. 2.24 şi explicaţiile aferente).

11. Conceperea compresorului 4:2 L, mai rapid decât variantele descrise în literatura de specialitate (v. paragraful 2.2.3.2, relaţiile 2.23-2.24, fig. 2.23 şi explicaţiile aferente).

12. Conceperea blocului logic ce controlează funcţionarea ALU din subunitatea de calcul în virgulă flotantă (pentru toate combinaţiile posibile din punct de vedere al semnului operanzilor şi al operaţiei care se execută) şi distribuţia sa între nivelele pipeline 1 şi 2, cu efect în creşterea vitezei de operare, respectiv în optimizarea din punct de vedere al suprafeţei ocupate pe cip a structurii blocului ALU (paragraful 3.2, relaţiile 3.1÷3.4, fig. 3.7 şi explicaţiile aferente).

13. Optimizarea circuitelor de aliniere a mantiselor, de adunare/scădere şi de normalizare din subunitatea de calcul în virgulă flotantă în scopul reducerii suprafeţei ocupate pe cip, în condiţiile garantării sincronizării (funcţionării cu aceeaşi perioadă de tact) cu subunitatea de calcul logaritmică (v. cap.3 în întregime).

14. Elaborarea diagramelor ce furnizează dimensiunea operanzilor, în cazul unui format nestandardizat, atunci când se realizează înmulţirea logaritmică prin citirea directă în memoria ROM, iar eroarea maximă de calcul este impusă (v. paragraful 1.3.1, relaţia 1.26, fig. 1.13, tab. 1.2 şi explicaţiile aferente).

15. Conceperea unui program Matlab pentru generarea valorilor ce trebuie memorate în ROM, în condiţiile menţinerii erorii de conversie la logaritmare şi antilogaritmare sub $1{,}8\times10^{-7}$.

Toate blocurile funcţionale ale subunităţii logaritmice şi ale subunităţii de calcul în virgulă flotantă au fost proiectate cu ajutorul programului MicroSim până la nivel de poartă logică sau poartă de transmisie şi apoi simulate, confirmându-se pe părţi şi pe ansamblu funcţionarea corectă şi în parametrii de viteză menţionaţi.

Evaluarea timpului de procesare a informaţiei s-a efectuat în condiţiile utilizării tehnologiei CMOS 0,5μm. Utilizarea însă a unei tehnologii submicronice mai avansate va permite operarea la o frecvenţă de clock a sistemului de calcul proiectat ce se apropie de 1 GHz ceea ce îl face competitiv cu cele mai recente realizări din domeniu.

Pe cadrul reprezentat de sistemul de calcul conceput în această lucrare poate fi adăugat hardware pentru realizarea unor funcţii specifice, dedicate unui anumit context matematic oferit de aplicaţia concretă în care se doreşte utilizarea procesorului aritmetic. Acest fapt deschide şi perspectivele de dezvoltare ulterioară a temei abordate, susţinute şi de avansul tehnologic actual:

- conceperea unui mecanism de acumulare la nivelul ALU din subunitatea logaritmică pentru efectuarea eficientă a unor produse sau fracţii multiple;
- conceperea unui procesor aritmetic puternic pentru aplicaţii DSP ce solicită volum mare de calcul în timp real şi care presupune operarea paralelă în mai multe subunităţi logaritmice sau în mai multe circuite de logaritmare, pentru creşterea numărului de termeni ce se calculează simultan, respectiv a numărului de factori ai produselor calculate. În ambele situaţii utilizarea metodei logaritmului mascat va permite creşterea vitezei de operare şi economie de hard prin utilizarea unui singur acumulator;
-concepţia unui procesor aritmetic de uz general, caz în care implementarea operaţiilor de extragere a rădăcinii pătrate, ridicare la putere şi logaritmare presupune introducerea în ALU a unui circuit de deplasare precum şi a unui sumator într-unul din cele două circuite de logaritmare, pentru extragerea deplasamentului din exponentul operandului. De asemenea va trebui adăugat hard suplimentar în ambele subunităţi ale procesorului aritmetic pentru introducerea mecanismelor de rotunjire a rezultatului în conformitate cu standardul IEEE754.
- reorganizarea subunităţii de calcul în virgulă flotantă pe doar două niveluri pipeline, prin secţionarea sumatoarelor de tip CLA din ALU şi aplicarea metodelor menţionate în paragraful 1.2.3;
- în cazul unei posibilităţi de implementare practică pe cip, proiectarea circuitelor de distribuţie a clock-ului pentru latch-urile dintre nivelurile pipeline şi desriere VHDL pentru toată structura hard proiectată;
- concepţia unui procesor aritmetic hibrid care să opereze în dublă precizie.

Bibliografie

[1] Al-Twaijry H. A., Flynn M. J., *Technology Scaling Effects on Multipliers*, IEEE Transactions on Computers, Vol. 47, No. 11, Nov. 1998.

[2] Arnold M. G, Bailey T., Cowles J., *Arithmetic Co-Transformations in the Real and Complex Logarithmic Number System*, IEEE Transactions on Computers, Vol. 47, No. 7, July 1998.

[3] Arnold M. G., Bailey T., Cowles J., *Applying Features of IEEE754 to Sign/Logarithm Arithmetic*, IEEE Transactions on Computers Vol. 41, No.8, August 1992.

[4] Arnold M. G., Bailey T., Cowles J., *Redundant Logarithmic Arithmetic*, IEEE Transactions on Computers Vol.39 NO.8, August 1990.

[5] Arnold M., Bailey T., Cowles J., *Coments on "An Architecture for Addition and Subtraction of Long Word Length Numbers in the Logarithm Number System"*, IEEE Transactions on Computers, Vol. 41, No. 6, June 1992.

[6] Bârsan R. M., *Fizica şi tehnologia circuitelor MOS integrate pe scară mare*, Editura Academiei R.S.R., Bucureşti, 1989.

[7] Beaumont-Smith A., Burgess N., Lefrere S., Lim C.C., *Reduced Latency IEEE Floating-Point Standard Adder Architectures*, 14th IEEE Symposium on Computer Arithmetic, Adelaide, April 1999.

[8] Benschneider B. J., Bowhill W. J., *A 50MHz Uniformly Pipelined 64b Floating-Point Arithmetic Processor*, IEEE International Solid-State Circuits Conference, 1990.

[9] Brubaker T. A., Becker J. C., *Multiplication Using Logarithms Implemented with Read-Only Memory*, IEEE Transactions on Computers Vol. C-24 No.8 August 1975.

[10] Bruguera J. D., Lang T., *Leading-One Prediction Scheme for Latency Improvement in Single Datapath Floating-Point Adders*, Proc. Intel Conf. Computer Design ICCD'98, Austin, Texas, 1998.

[11] Bruguera J. D., Lang T., *Leading-One Prediction with Concurrent Position Correction*, IEEE Transactions on Computers, Vol. 48, No. 10, October 1999.

[12] Chen. C. & all, *Pipelined Computation of Very Large Word Length LNS Addition/Subtraction with Polynomial Hardware Cost*, IEEE Transactions on Computers, Vol. 49, No. 7, July 2000.

[13] Cheng F., Unger S. H., Theobald M., *Self-Timed Carry-Lookahead Adders*, IEEE Transactions on Computers, Vol. 49, No. 7, July 2000.

[14] Ciminiera L., Montuschi P., *Carry-Save Multiplication Schemes Without Final Addition*, IEEE Transaction on Computers, Vol.45, No.9, September 1996.

[15] Ciugudean M., *Algoritmi pentru operaţii cu logaritmi binari în unităţi aritmetice cu virgulă flotantă*, Simpozionul Naţional de Teoria Sistemelor, Universitatea Craiova, 1986.

[16] Ciugudean M., *Algoritmi şi structuri logice celulare de generare a logaritmilor şi antilogaritmilor binari pentru dispozitive aritmetice cu virgulă mobilă*, Teză doctorat, I.P. T.V., Timişoara, 1976.

[17] Ciugudean M., *Structures logiques cellulaires pour la génération des logarithmes et antilogarithmes binaires*, Buletinul I.P.T.V. Timişoara, Tom 23(37), Fascicola 1, 1978.

[18] Ciugudean M., *Unitate aritmetică în virgulă flotantă logaritmică*, Simpozionul Naţional de Teoria Sistemelor, Universitatea Craiova, 1986.

[19] Clouser J. & all, A 600-MHz Superscalar Floating-Point Processor, IEEE Journal of Solid-State Circuits, Vol. 34, No. 7, July 1999.

[20] Coleman J., Chester E., Softley C., Kadlec J., *Corrections to "Arithmetic on the European Logarithmic Microprocessor"*, IEEE Transactions on Computers, Vol. 49, No. 10, Oct. 2000.

[21] Coleman J., Chester E., Softley C., Kadlek J., *Arithmetic on the European Logarithmic Microprocessor*, IEEE Transactions on Computers, Vol. 49, No. 7, July 2000.

[22] Das D. & all, *Implementation of Four Common Functions on an LNS Co-Processor*, IEEE Transactions on Computers, Vol. 44, No. 1, January 1995.

[23] Ercegovac M. D. & all, *Reciprocation, Square-Root and Some Elementary Functions Using Small Multipliers*, IEEE Transactions on Computers, Vol. 49, No. 7, July 2000.

[24] Fenn S. T. J., Benaissa M., Taylor D., *Multiplication and Division Over Dual Basis*, IEEE Transaction on Computers, Vol.45, No.3, March 1996.

[25] Fiore P. D., *Parallel Multiplication Using Fast Sorting Networks*, IEEE Transactions on Computers, Vol. 48, No.6, June 1999.

[26] Fujii H. & all, *A Floating-Point Cell Library and a 100-MFLOPS Image Signal Processor*, IEEE Journal of Solide-State Circuits, Vol.27, No.7. July 1992.

[27] Goto J. & all, *250-MHz BiCMOS Super-High-Speed Video Signal Processor (S-VSP) ULSI*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 12, December 1991.

[28] Guyot A., Muller J., *A Way to Build Efficient Carry-Skip Adders*, IEEE Transactions on Computers, Vol. C-36, No. 10, October 1987.

[29] Hanyu T., Kameyama M., *A 200 MHz Pipelined Multiplier Using 1.5 V-Supply Multiple-Valued MOS Current-Mode Circuits with Dual-Rail Source-Coupled Logic*, IEEE Journal of Solid-State Circuits, Vol. 30, No. 11, November 1995.

[30] Hiasat A. A., *New Efficient Structure for a Modular Multiplier for RNS*, IEEE Transactions on Computers, Vol. 49, No. 2, Feb. 2000.

[31] Hummel R., *Processeur et Coprocesseur*, Editeur DUNOD, 1992.

[32] Ide N. & all, *2.44-GFLOPS 300-MHz Floating-Point Vector-Processing Unit for High-Performance 3-D Graphics Computing*, IEEE Journal of Solid-State Circuits, Vol. 35, No. 7, July 2000.

[33] Ide N., Kondo H.F., *A 320-MFLOPS CMOS Floating-Point Processing for Superscalar Processors*, IEEE Journal of Solid-State Circuits, Vol.28, No.3, March 1993.

[34] Inoue T. & all, *A 2000-MOPS Embedded RISC Processor with a Rambus DRAM Controller*, IEEE Journal of Solid-State Circuits, Vol. 34, No. 7, July 1999.

[35] Itoh N. & all, *A 600-MHz 54×54-bit Multiplier with Rectangular-Styled Wallace Tree*, IEEE Journal of Solid-State Circuits, Vol. 36, No. 2, February 2001.

[36] Jessani R. M., Putrino M., *Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units*, IEEE Transactions on Computers, Vol. 47, No. 9, Sept. 1998.

[37] Jurca L. A., *Concepţia unui procesor aritmetic cu virgulă flotantă logaritmic*, Referat Nr.2, Timişoara, 1997.

[38] Jurca L. A., Drăgoi B., *Using 4:2 L Compressor in Partial Product Reduction Tree for Fast Binary Multipliers*, Buletinul UPT, Tom 46(60), Fasc. 1, 2001.

[39] Jurca L. A., *Floating-Point Unit for Fast Performing Addition and Subtraction in Single Precision Format*, Buletinul UPT, Tom 45(59), Fasc. 2, 2000.

[40] Jurca L. A., *Logarithmic Unit for Fast Performing of Multiplication and Division Operations in Single Precision Format*, Buletinul UPT, Tom 45(59), Fasc. 2, 2000.

[41] Jurca L. A., *Method and Circuit for the Fast Calculation of the Binary Logarithm and Antilogarithm in Single Precision Format*, Buletinul UPT, Tom 45(59), Fasc.1, 2000.

[42] Jurca L. A., *Some Considerations Regarding the Design of a Hybrid Logarithmic, Floating-Point Mathematical Processor*, Buletinul UPT, Tom 45(59), Fasc. 1, 2000.

[43] Jurca L. A., *Stadiul actual al procesoarelor aritmetice cu virgulă flotantă*, Referat Nr.1, Timişoara, 1997.

[44] Jurca L. A., *Using Masked Logarithm Method for Fast Multiplication and Division Logarithmic Unit Design*, Buletinul UPT, Tom 46(60), Fasc. 1, 2001.

[45] Kadlec J., Softley C. & all, *32-bit Logarithmic ALU for Handel C 2.1 and Celoxica DK1 (53 MHz for XCV2000E-6 based RC1000 board)*, Celoxica User Conference, Stratford, UK, April 2001.

[46] Kantabutra V., *Accelerated Two-Level Carry-Skip Adders – A Type of Very Fast Adders*, IEEE Transactions on Computers, Vol. 42, No. 11, November 1993.

[47] Kantabutra V., *Designing Optimum One-Level Carry-Skip Adders*, IEEE Transactions on Computers, Vol. 42, No. 6, June 1993.

[48] Kantabutra V., *On Hardware for Compputing Exponential and Trigonometric Functions*, IEEE Transactions on Computers, Vol.45, No.3, March 1996.

[49] Karagianni K. & all, *Operation-Saving VLSI Architectures for 3D Geometrical Transformation*, IEEE Transactions on Computers, Vol. 50, No. 6, June 2001.

[50] Lai F., *A 10-ns Hybrid Number System Data Execution Unit for Digital Signal Processing Systems*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 4, Apr. 1991.

[51] Lai F., Wu C.F.E., *A Hybrid Number System Processor with Geometric and Complex Arithmetic Capabilities*, IEEE Transactions on Computers Vol. 40 No.8 Aug.1991.

[52] Lang J. H. & all, *Integrated-Circuit Logarithmic Arithmetic Units*, IEEE Trans. Comp Vol C-34 No.5, August 1985.

[53] Lang T., Montuschi P., *Boosting Very-High Radix Division with Prescaling and Selection by Rounding*, IEEE Transactions on Computers, Vol. 50, No. 1, January 2001.

[54] Lang. T., Montuschi P., *Verry High Radix Square Root with Prescaling and Rounding and a combined Division/Square Root Unit*, IEEE Transactions on Computers, Vol. 48, No. 8, August 1999.

[55] Larsson-Edefors P., *A 965-Mb/s 1.0-ηm Standard CMOS Twin-Pipe Serial/Parallel Multiplier*, IEEE Journal of Solid-State Circuits, Vol. 31, No. 2, February 1996.

[56] Lev L. A. & all, *A 64-b Processor with Multimedia Support*, IEEE Journal of Solid-State Circuits, Vol. 30, No. 11, November 1995.

[57] Lewis D. M., *114 MFLOPS LNS Arithmetic Unit for DSP Applications*, IEEE Journal of Solide-State Circuits, vol.30, No.12, December 1995.

[58] Lewis D. M., *An Architecture for Addition and Subtraction of Long Word Length Numbers in the Logarithm Number System*, IEEE Transactions on Computers Vol. 39, No.11, Nov.1990.

[59] Lewis D. M., *Interleaved Memory Function Interpolators with Application to Accurate LNS Arithmetic Unit*, IEEE Transactions on Computers Vol. 43, No.8 August 1994.

[60] Lineback J. R., *Single Chip Processors*, Electronics, March 1988.

[61] Lo H., Aoki Y., *Generation of a Precise Binary Logarithm with Difference Grouping Programmable Logic Array*, IEEE Transactions on Computers Vol. C-34 No.8, August 1985.

[62] Lu F., Samueli H., *A 200-MHz CMOS Pipelined Multiplier-Accumulator Using a Quasi-Domino Dinamic Full-Adder Cell Design*, IEEE journal of Solid-State Circuits, Vol.28, No.2, February 1993.

[63] Maenner R., *A Fast Integer Binary Logarithm of Large Arguments* IEEE Micro, December 1987.

[64] Minami T. & all, *A 300-MOPS Video Signal Processor with a Parallel Architecture*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 12, December 1991.

[65] Montoye R. K., Cook P. W., Hokenek E., Havvreluk R. P., *An 18ns 56-Bit Multiply-Adder Circuit*, IEEE International Solid-State Circuits Conference, 1990.

[66] Montoye R. K., Hokenek E., Runion S. L.: *Design of the IBM RISC Sistem/6000 Floating-Point Execution Unit*, IBM J. Res. Develop., Vol.34 No.1, January 1990.

[67] Mori J., Nagamatsu M. *A 10-ns 54×54-b Parallel Structured Full Array Multiplier with 0.5-μm CMOS Tehnology*, IEEE Journal of Solid-State Circuits, Vol.26, No.4, April 1991.

[68] Nakayama T., Kojima S., *An 80b 6.7MFLOPS Floating-Point Processor with Vector / Matrix Instructions*, IEEE International Solid-State Circuits Conference, 1989.

[69] Oberman S. F., Flynn M. J., *Division Algorithms and Implementations*, IEEE Transactions on Computers, Vol. 46, No. 8, August 1997.

[70] Ohkubo N. & all, *A 4.4 ns CMOS 54×54 b Multiplier Using Pass-Transistor Multiplexer*, IEEE Journal of Solid-State Circuits, Vol. 30, No. 3, March 1995.

[71] Okamoto F. & all, *A 200-MFLOPS 100-MHz 64-b BiCMOS Vector-Pipelined Processor (VPP) ULSI*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 12, December 1991.

[72] Oklobdzija V. G., Villeger D., Liu S. S., *A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipiliers Using an Algorithmic Approach*, IEEE Transaction on Computers, Vol.45, No.3, March 1996.

[73] Petterson D., Hennessy J. L., *Computer Architecture. A Quantitative Approach*, Morgan Press, 1996.

[74] Phatak D. S., Goff T., Koren I., *Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binari Reprezentations*, IEEE Transactions on Computers, Vol.50, No. 11, Nov. 2001.

[75] Phatak D. S., Goff T., Koren I., *Redundancy Management in Arithmetic Processing via Redundant Binary Reprezentations*, Asilomar '99.

[76] Schulte M.J., Swartzlander E.E., *Hardware Design for Exactly Rounded Elementary Functions*, IEEE Transactions on Computers Vol.43, No.8, Aug.1994.

[77] Seidel P.M., Even G., *An IEEE Floating-Point Adder Design Optimized For Speed*, 15th IEEE Symposium on Computer Arithmetic, Adelaide, April 2000.

[78] Senthinathan R. & all, *A 650-MHz, IA-32 Microprocessor with Enhanced Data Streaming for Graphics and Video*, IEEE Journal of Solid-State Circuits, Vol. 34, No. 11, November 1999.

[79] Srinivas H. R., Parhi K. K., *A fast VLSI Adder Architecture*, IEEE Journal of Solide-State Circuits, Vol.27, No.5, May 1992.

[80] Stallings W., *Computer Architecture and Organization*, Prentice Hall Inc, 1996.

[81] Stelling P. F., Oklobdzija V. G. & all, *Optimal Circuits for Parallel Multipliers*, IEEE Transactions on Computers, Vol. 47, No. 3, March 1998.

[82] Stouraitis T., Taylor F. J., *Floating-Point to Logarithmic Encoder Error Analysis*, IEEE Transaction on Computers, Vol.37, No.7, July 1988.

[83] Strugaru C., Popa M., *Microprocesoare pe 16 biţi*, Editura TM 1992.

[84] Sunar B., Koc C., *An Efficient Optimal Normal Basis Type II Multiplier*, IEEE Transactions on Computers, Vol. 50, No.1, January 2001.

[85] Suzuki H. & all, *Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition*, IEEE Journal of Solid-State Circuits, Vol. 31, No. 8, Aug. 1996.

[86] Suzuoki M., *A Microprocessor with a 128-Bit CPU, Ten Floating-Point MAC's, Four Floating-Point Dividers, and an MPEG-2 Decoder*, IEEE Journal of Solid-State Circuits, Vol. 34, No. 11, November 1999.

[87] Swartzlander E. E., Alexopoulos A. G., *The Sign/Logarithm Number System*, IEEE Transactions on Computers December 1975.

[88] Takagi N., *Powering by a Table Look-Up and a Multiplication with Operand Modification*, IEEE Transactions on Computers, Vol. 47, No. 11, Nov. 1998

[89] Taylor F. J. & all, *A 20 bit Logarithmic Number System Processor*, IEEE Transactions on ComputersVol.37, No.2, Feb.1988.

[90] Taylor F. J. *A Hibrid Floating - Point Logarithmic Number System Processor*, IEEE Trans. Circuits & Systems, Vol. CAS-32, January 1985.

[91] Timmermann D., Rix B., Hahn H., Hosticka J., *A CMOS Floating-Point Vector-Arithmetic Unit*, IEEE Journal of Solid-State Circuits, Vol.29, No.5, May 1994.

[92] Toacşe G., *Introducere în microprocesoare*, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1986.

[93] Toacşe G., Nicula D., *Electronică Digitală*, Editura Teora, Bucureşti, 1996.

[94] Toma C. I., *Dispozitiv de înmulţire 4×2 integrat*, Buletinul I.P.T.V. Timişoara, Tom 21(35), Fascicola 1, 1976.

[95] Um J., Kim T., *An Optimal Allocation of CSA in Arithmetic Circuits*, IEEE Transactions on Computers, Vol. 50, No. 3, March 2001.

[96] Vinnakota B. *Implementing Multiplication with Split Read-Onli Memory*, IEEE Transaction on Computers, Vol.44, No.11, November 1995.

[97] Wang Z., Jullien H. A., Miller W. C., *A New Design Tehnique for Column Compression Multipliers*, IEEE Transactions on Computers, Vol.44, No.8, August 1995.

[98] Weste N., Eshraghian K., *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley Publishing Company, 1985.

[99] Williams T., Patkar N., Shen G., *SPARC64 A64-b 64-Active-Instructions Out-of-Order-Execution MCM Processor*, IEEE Journal of Solid-State Circuits, Vol. 30, No. 11, November 1995.

[100] Woods J. V. & all, *Amulet 1: An Asynchronous ARM Microprocessor*, IEEE Transactions on Computers, Vol. 46, No. 4, April 97.

[101] Yan H. & all, *A Low-Power 16×16-b Parallel Multiplier Utilizing Pass-Transistor Logic*, IEEE Journal of Solid-State Circuits, Vol. 34, No. 10, October 1999.

[102] Yang C., Sano B., Lebek A. R., *Exploiting Parallelism in Geometry Processing with General Purpose Processors and FP SIMD Instructions*, IEEE Transactions on Computers, Vol. 49, No. 9, Sept. 2000.

[103] Yeh W., Jen C., *High-Speed Booth Encoded Parallel Multiplier Design*, IEEE Transactions on Computers, Vol. 49, No. 7, July 2000.

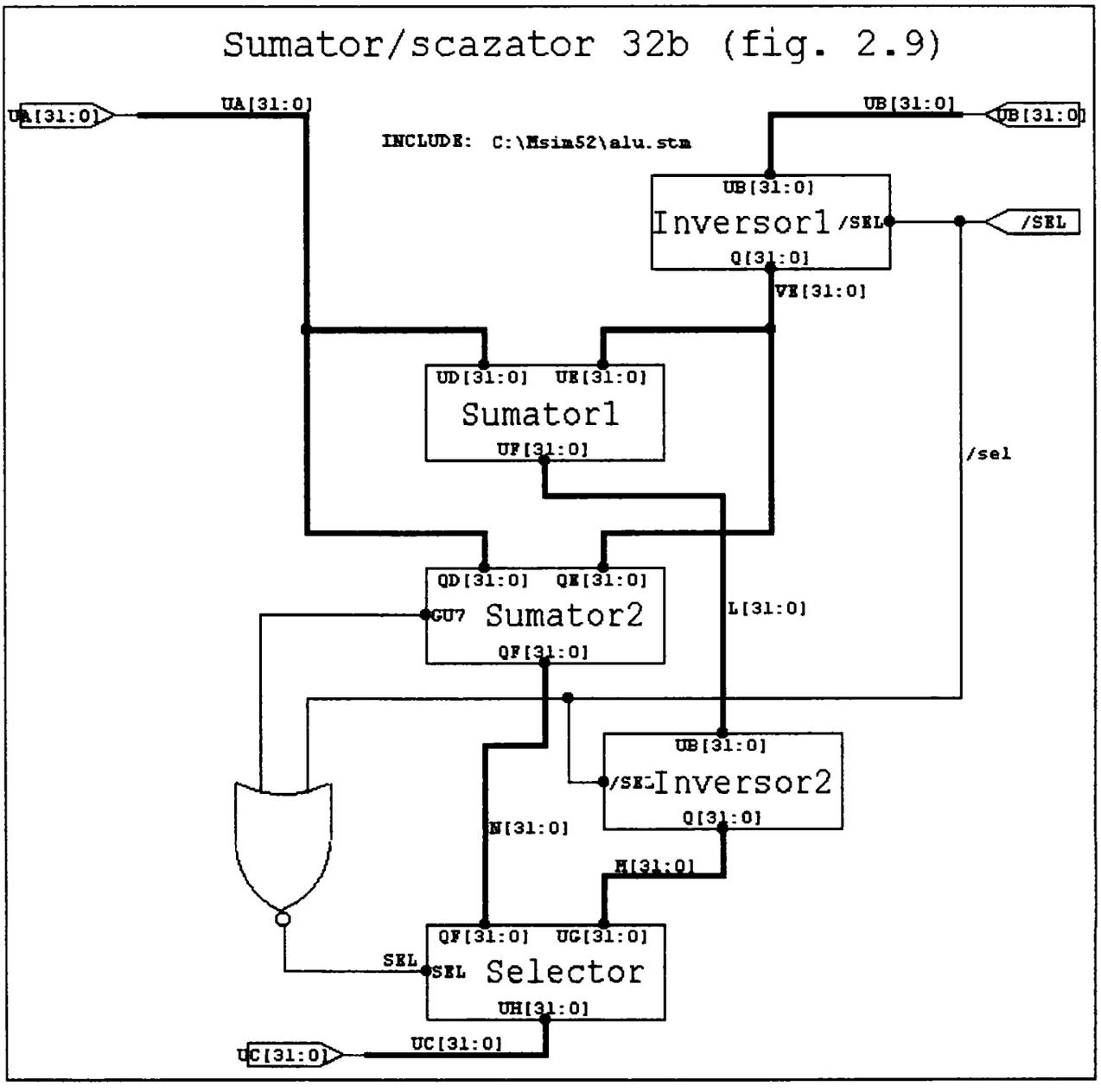
[104] Yu L. K., Lewis D. M., *A 30-b Integrated Logarithmic Number System Processor*, IEEE Journal of Solid-State Circuits, Vol.26, No.10, October 1991.

[105] Zhunang N., Wu H., *A new Design of the CMOS Full Adder*, IEEE Journal of Solide-State Circuits, Vol. 27, No.5, May 1992.

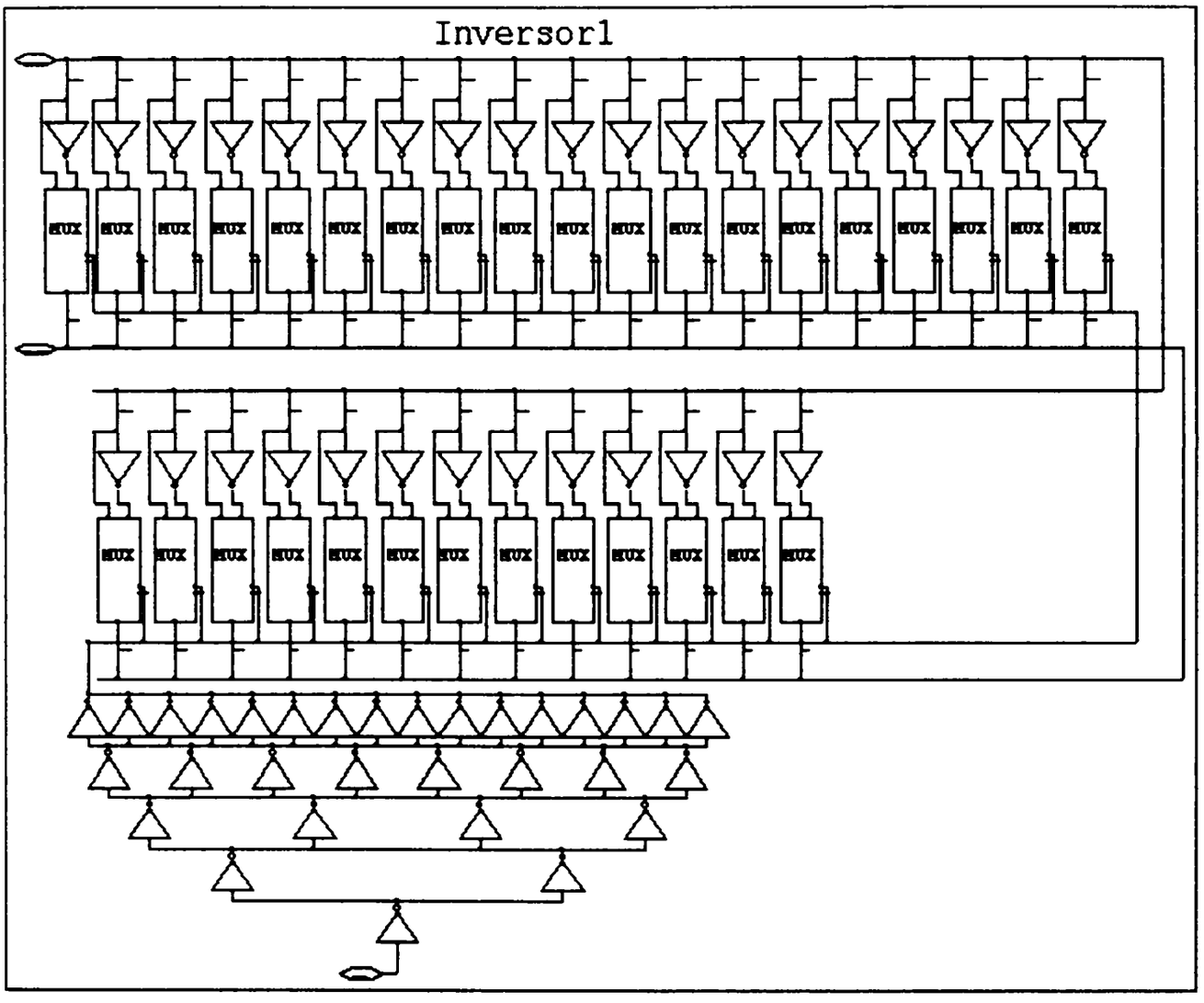
[106] Villeger D., *Fast Parallel Multipliers*, Final Report, Ecole Superieure d'Ingenieurs en Electrotechnique et Electronique, Noisy-le-Grand, May 1993.

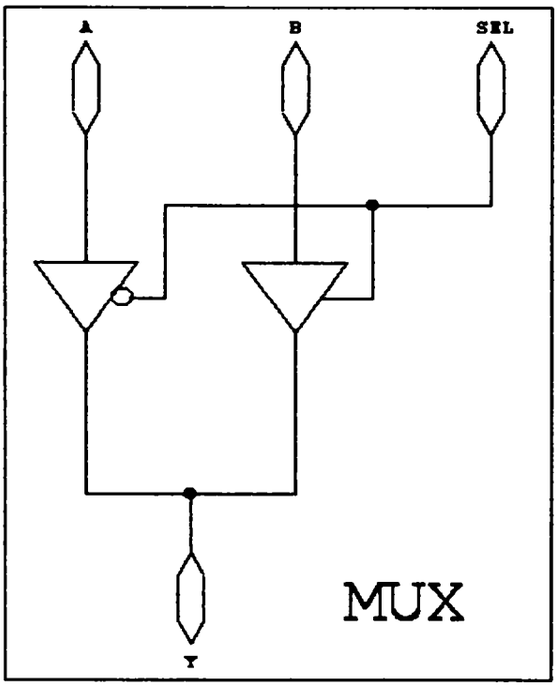
[107] Lai F., *The Efficient Implementation and Analysis of a Hybrid Number System Processor*, IEEE Transactions on Circuits and Systems, Vol. 46, No. 6 ICSPE5, June 1993.

[108] Butas J. & all, *Asynchronous Cross-Pipelined Multiplier*, IEEE Journal of Solid-State Circuits, Vol. 36, No. 8, August 2001.


Sumator/scazator 32b (fig. 2.9)
UA[31:0]
UA[31:0]
INCLUDE: C:\Msim52\alu.stm
UB[31:0]
UB[31:0]
UB[31:0]
Inversor1
/SEL
Q[3:0]
/SEL
VE[31:0]
UD[31:0]
UE[31:0]
Sumator1
UF[31:0]
/sel
QD[31:0]
QE[31:0]
CU?
Sumator2
QF[31:0]
L[31:0]
UB[31:0]
/SEL
Inversor2
Q[3:0]
N[31:0]
M[31:0]
QF[31:0]
UG[31:0]
SEL
SEL
Selector
UH[31:0]
UC[31:0]
UC[31:0]

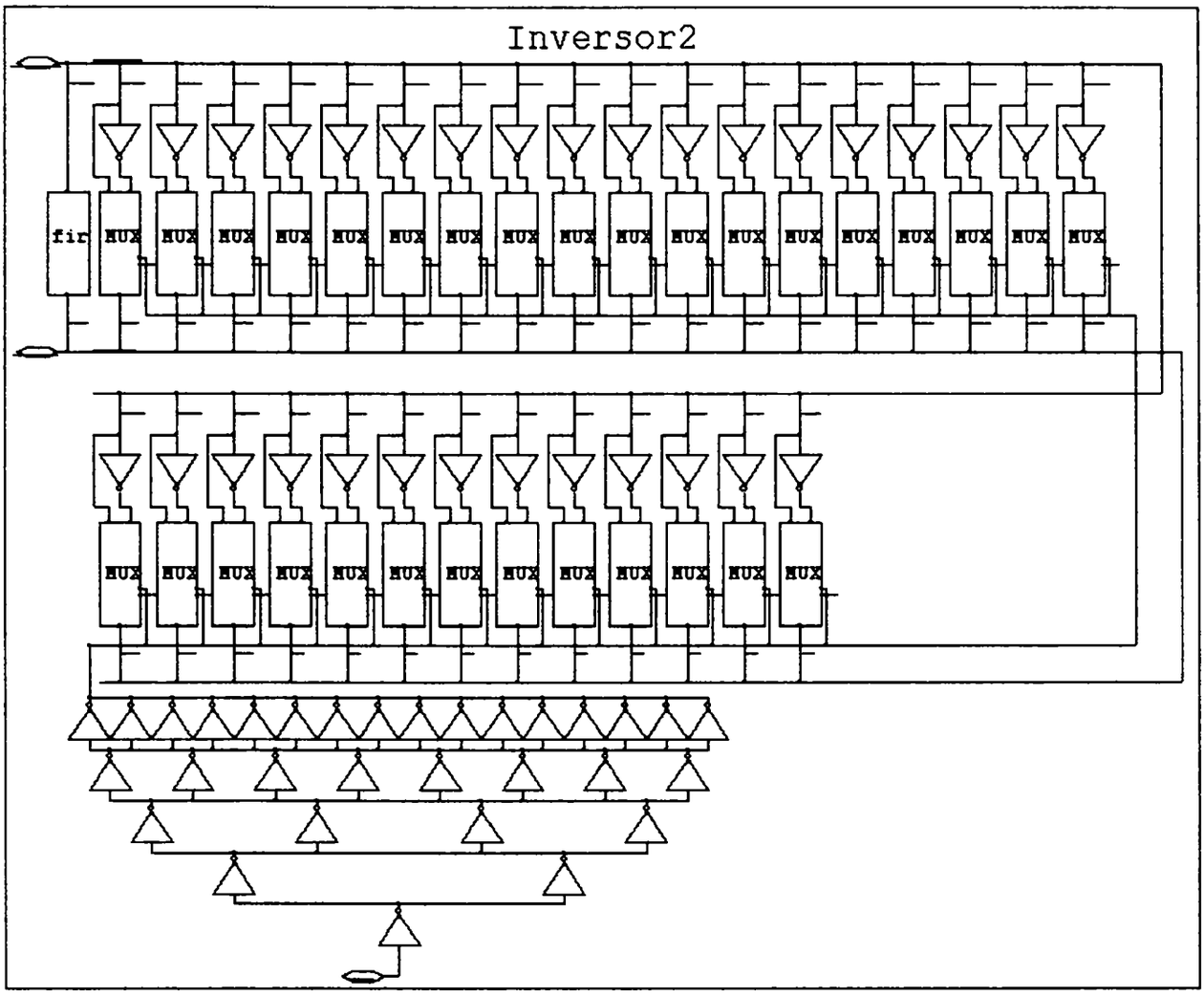
ANEXA 1

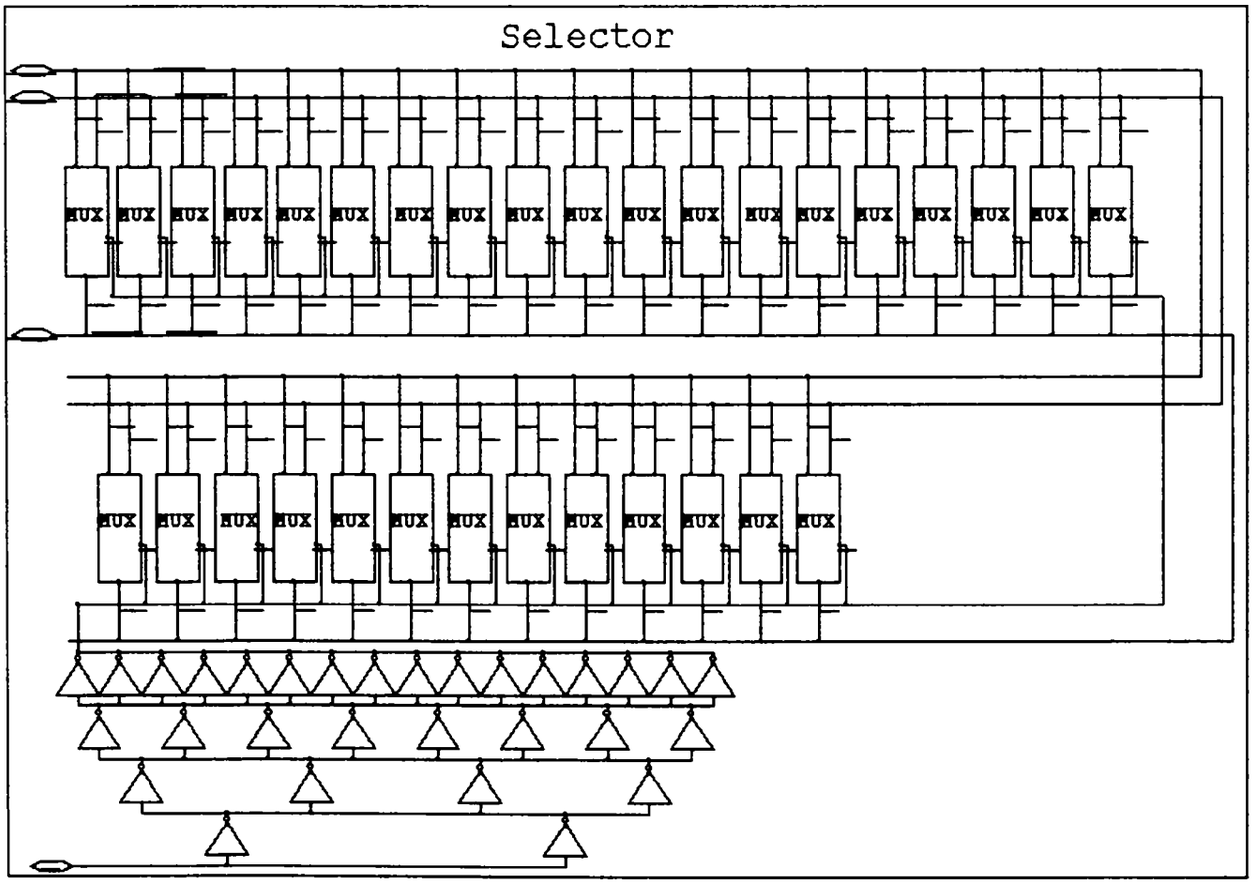


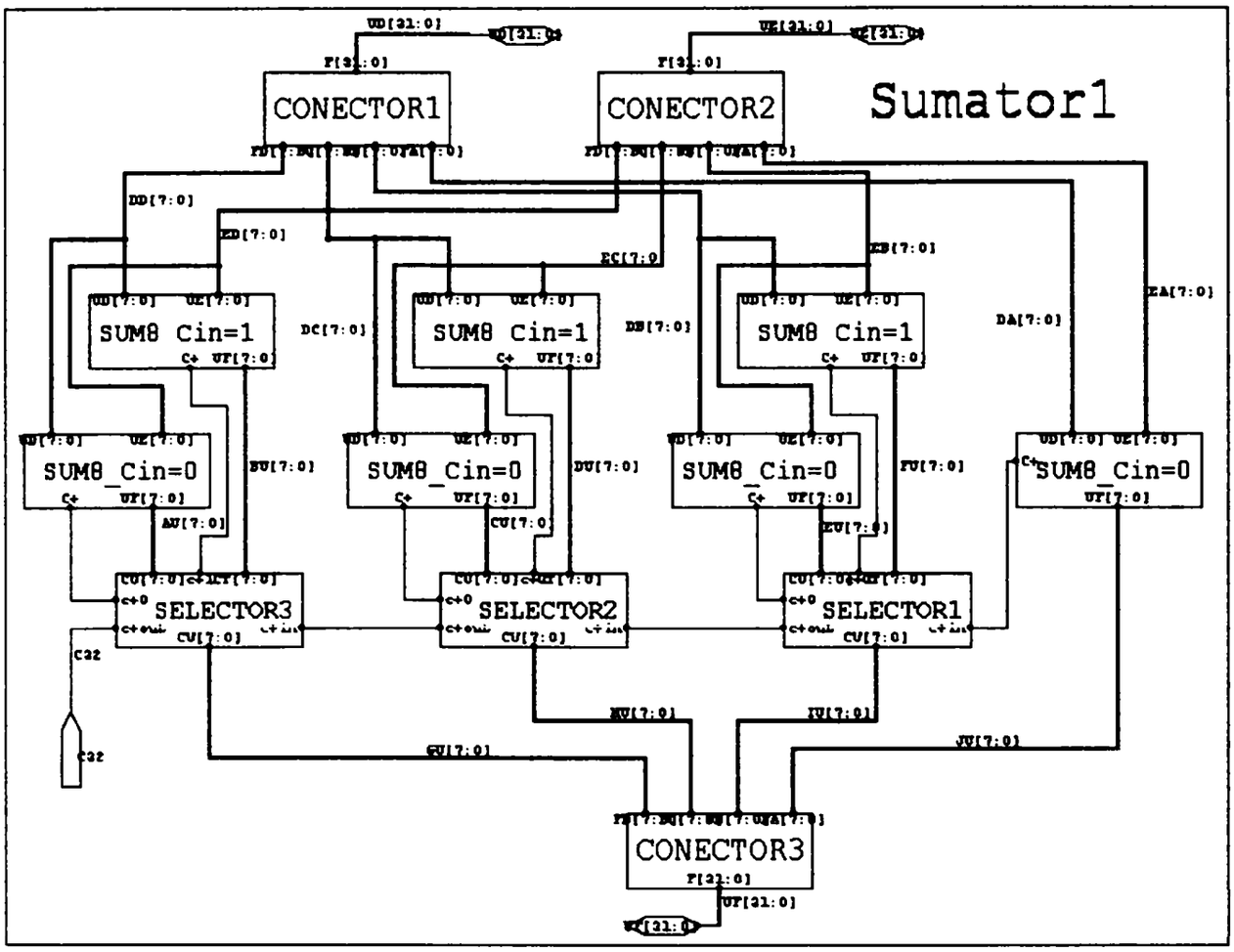


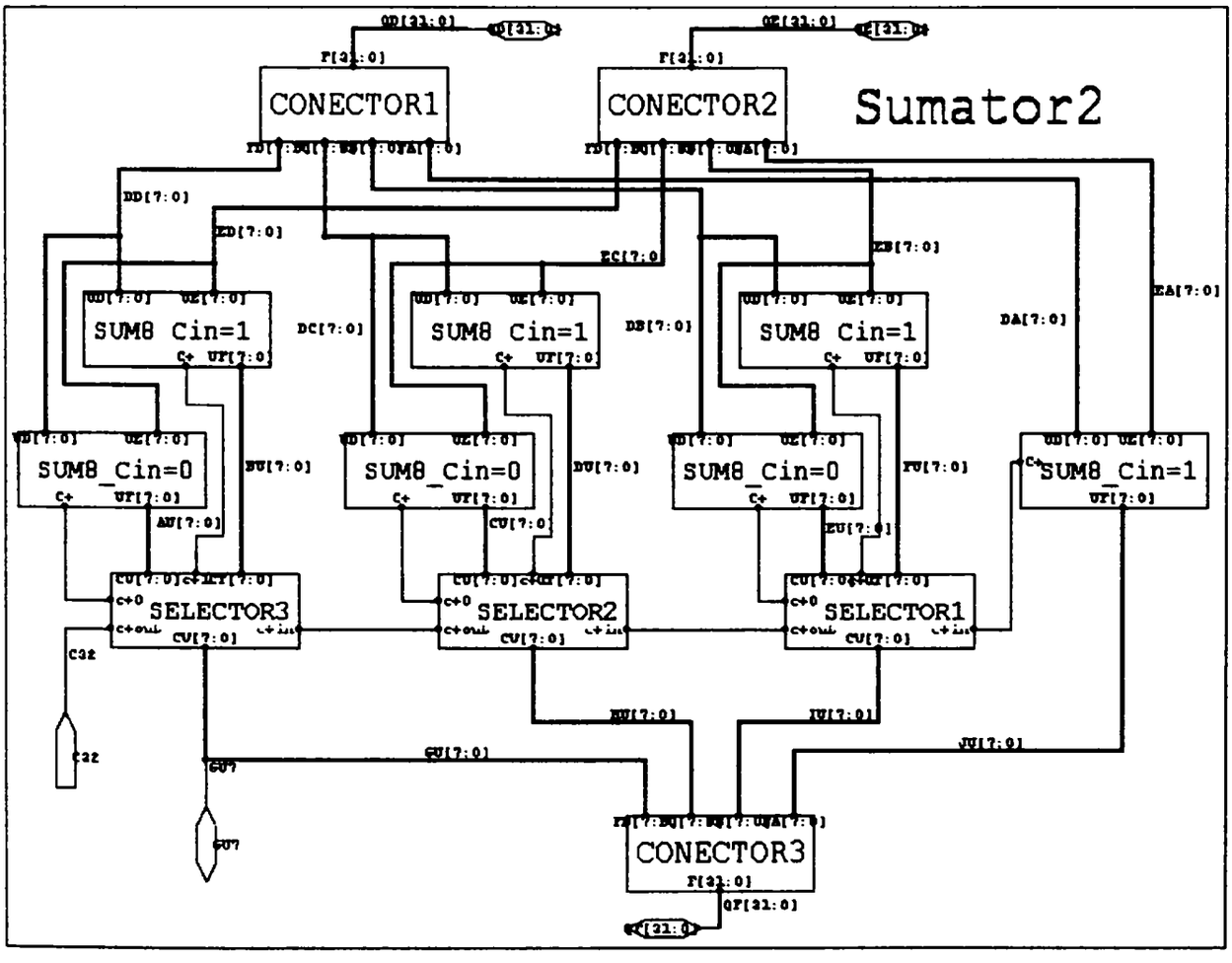
A1.2

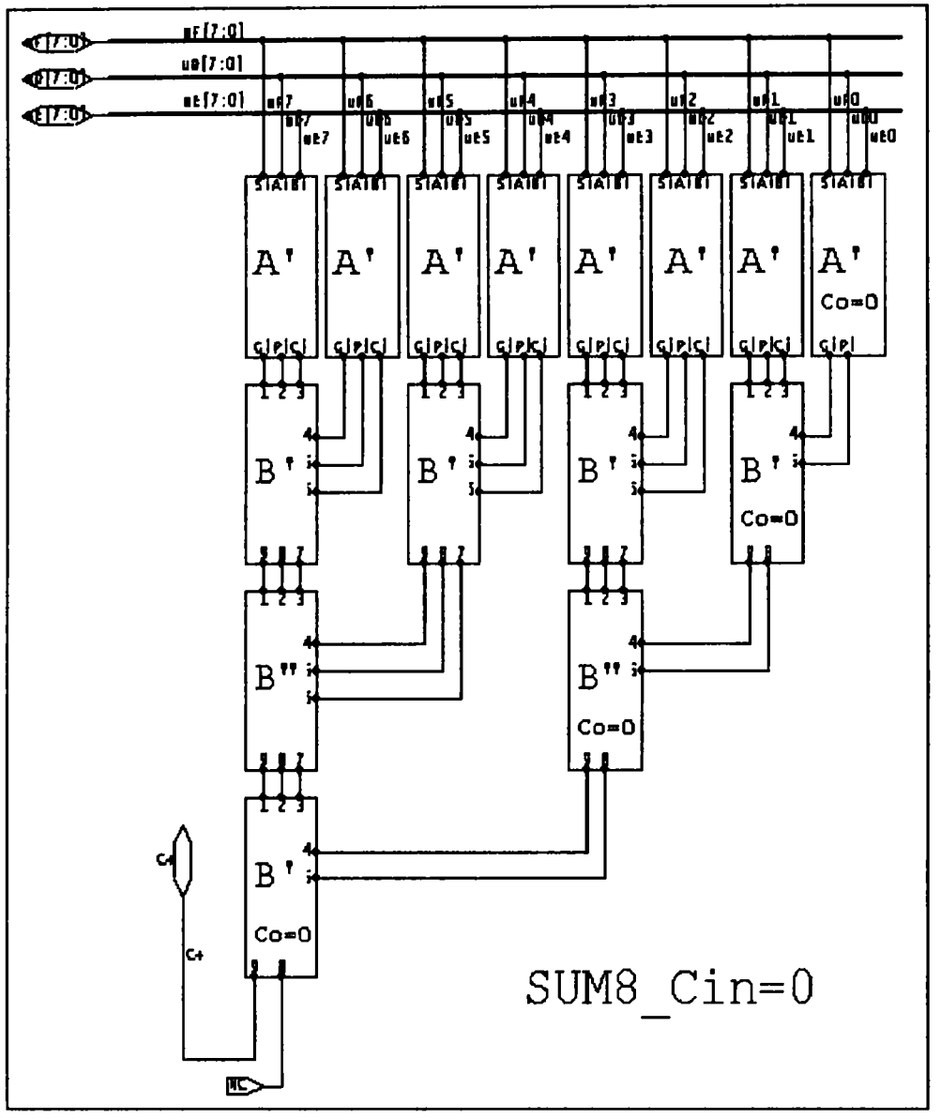
ANEXA 1

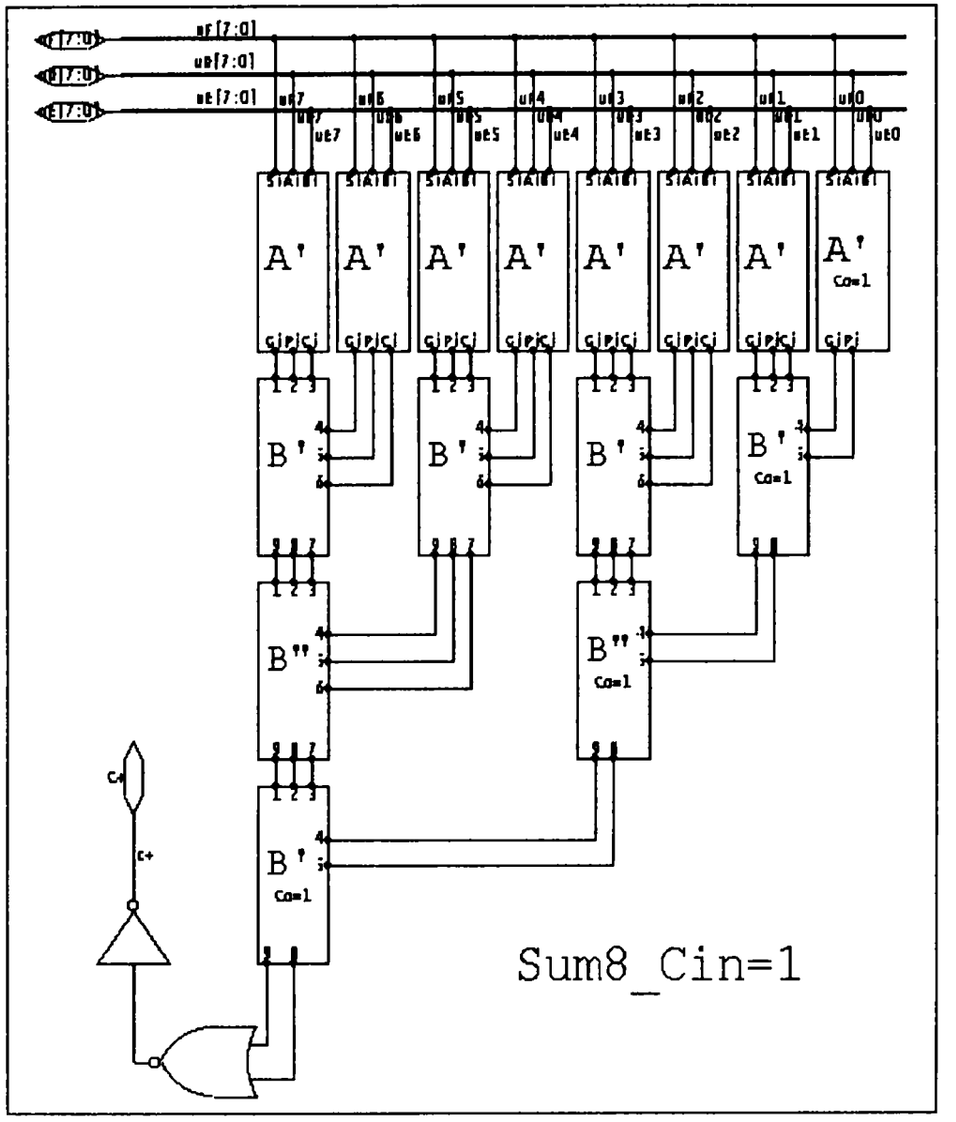




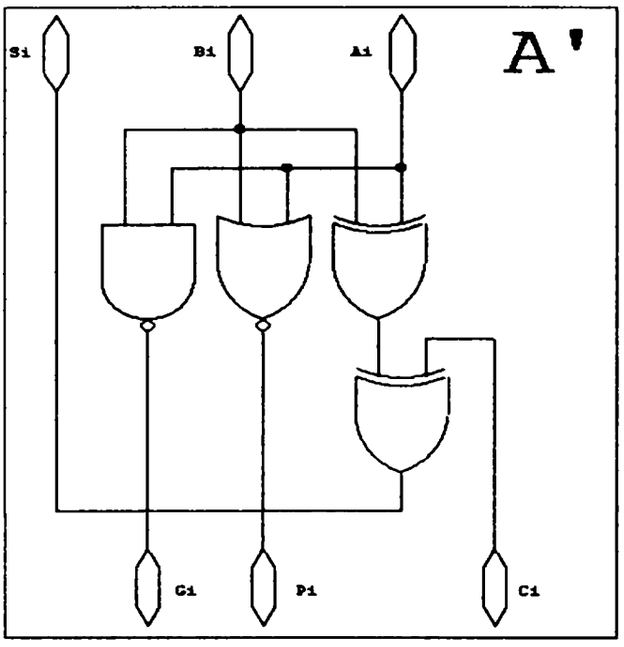

Sumator1
CONECTOR1
CONECTOR2
CONECTOR3
SUM8 Cin=1
SUM8 Cin=1
SUM8 Cin=1
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SELECTOR3
SELECTOR2
SELECTOR1
C32
C32

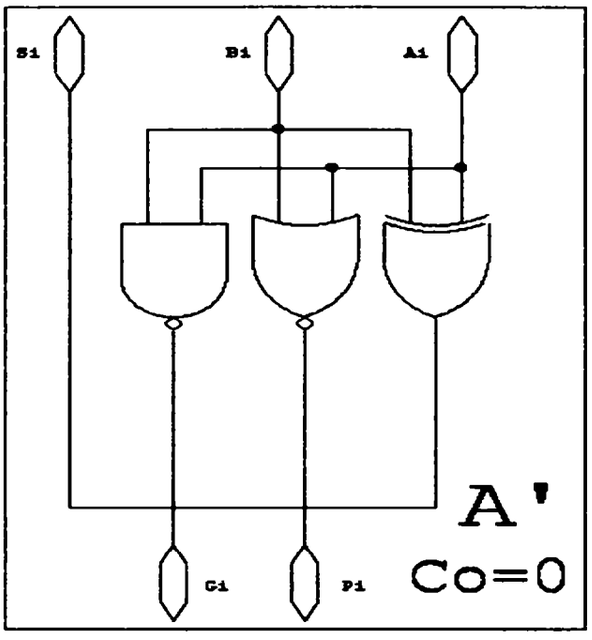

Sumator2
CONECTOR1
CONECTOR2
CONECTOR3
SUM8 Cin=1
SUM8 Cin=1
SUM8 Cin=1
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=1
SELECTOR3
SELECTOR2
SELECTOR1
C32
C32
GU7
GU7

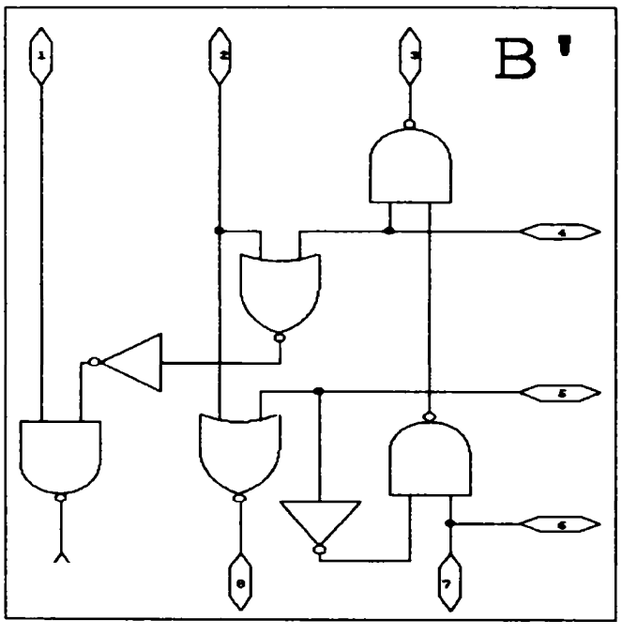


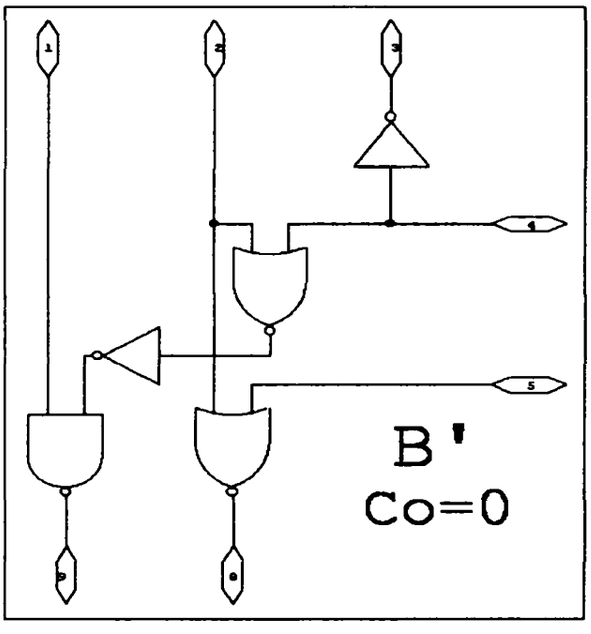


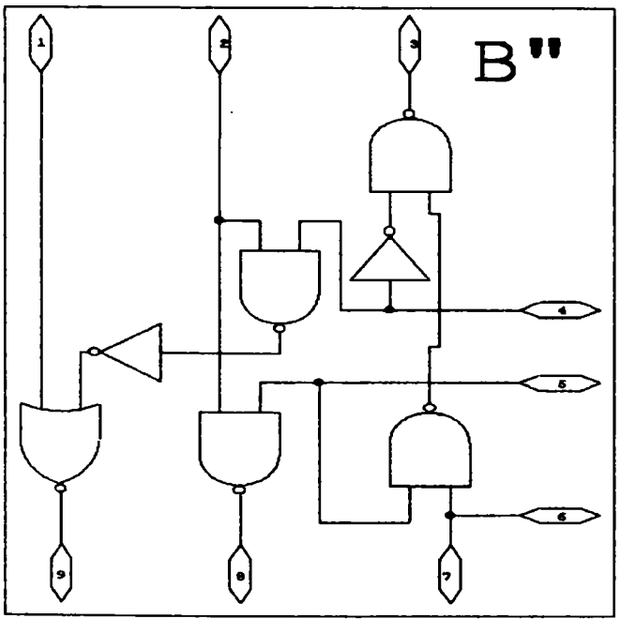
ANEXA 1

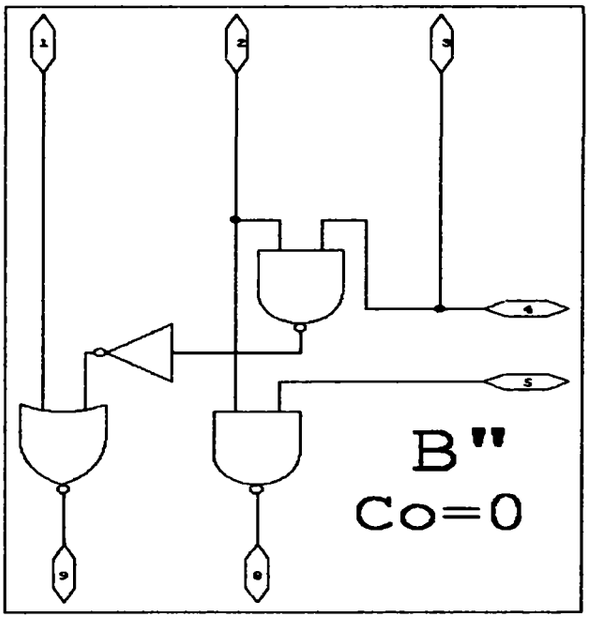


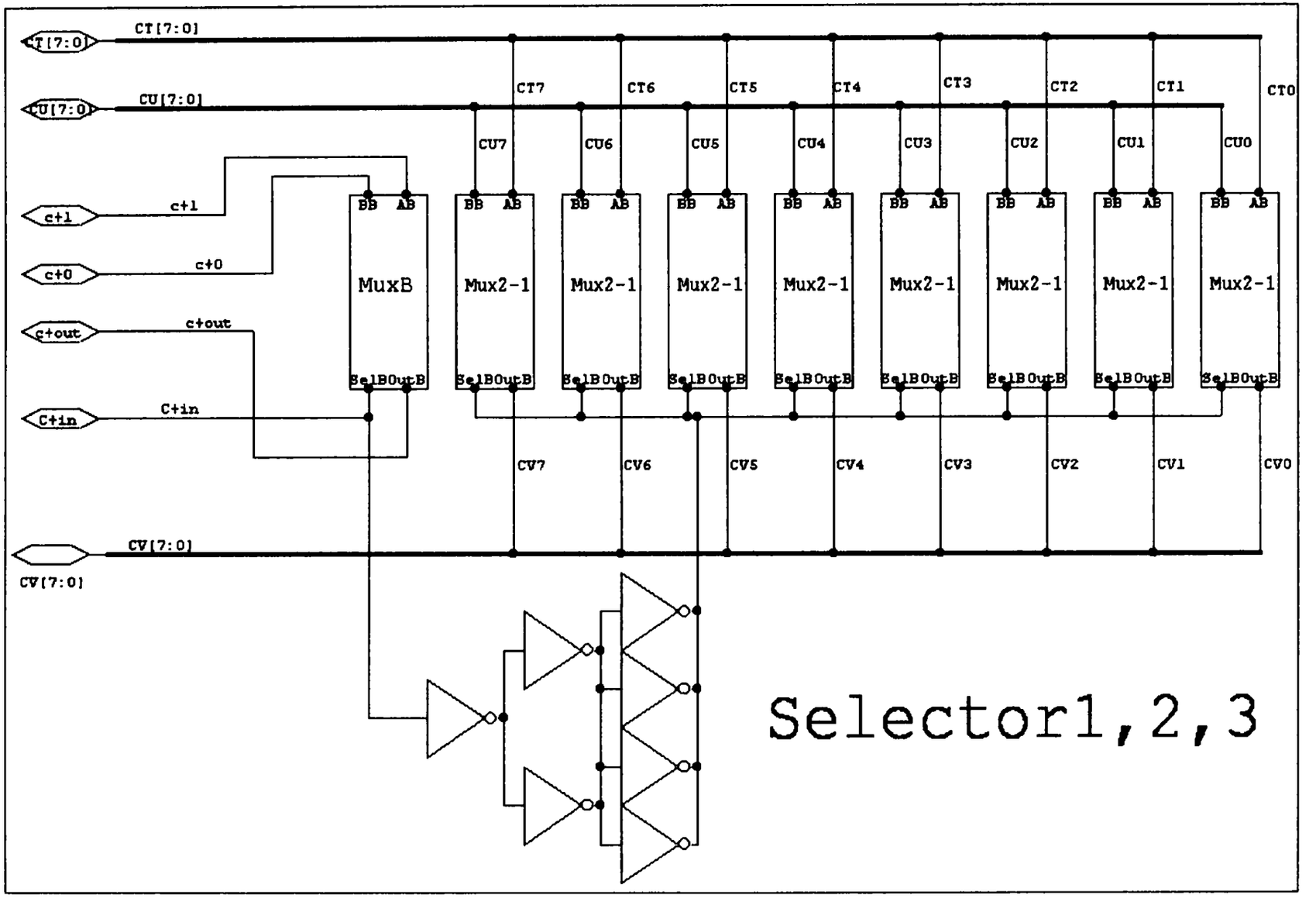


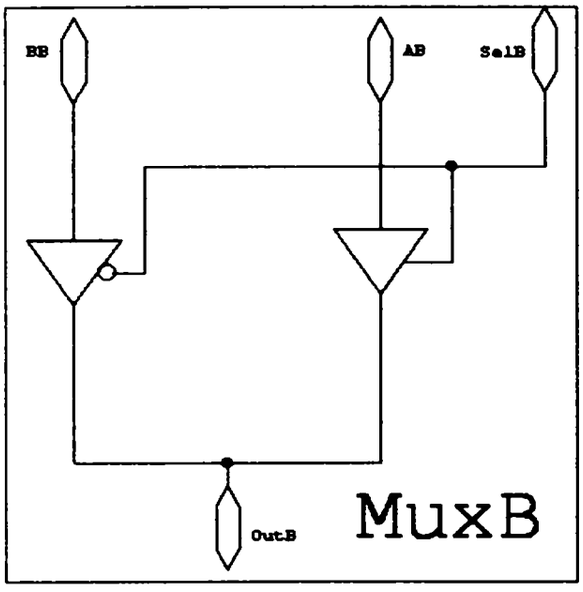


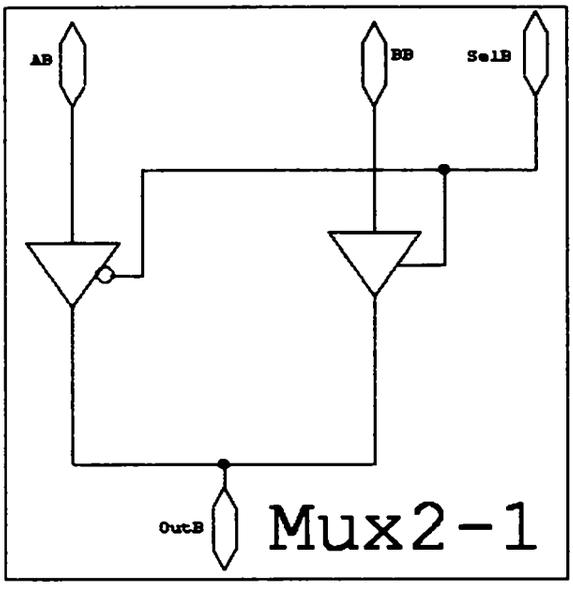


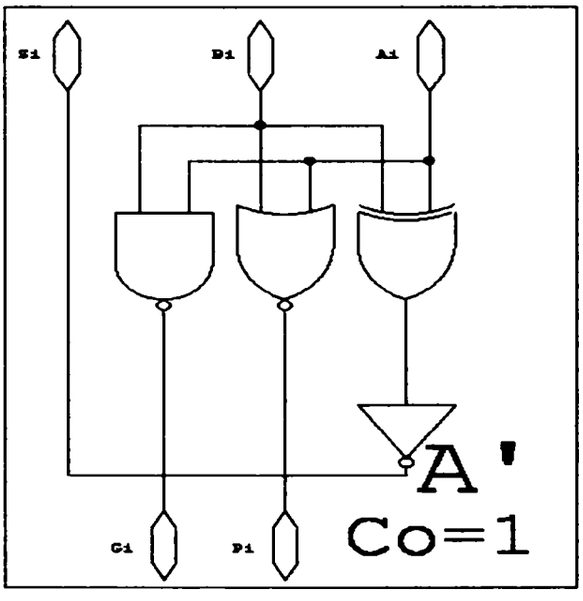


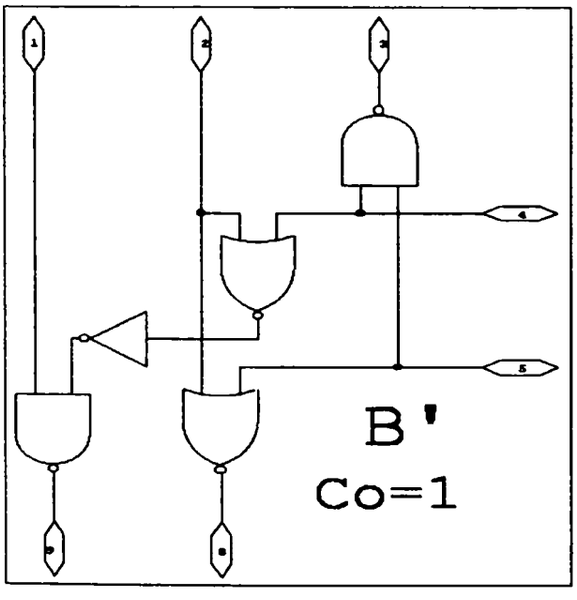


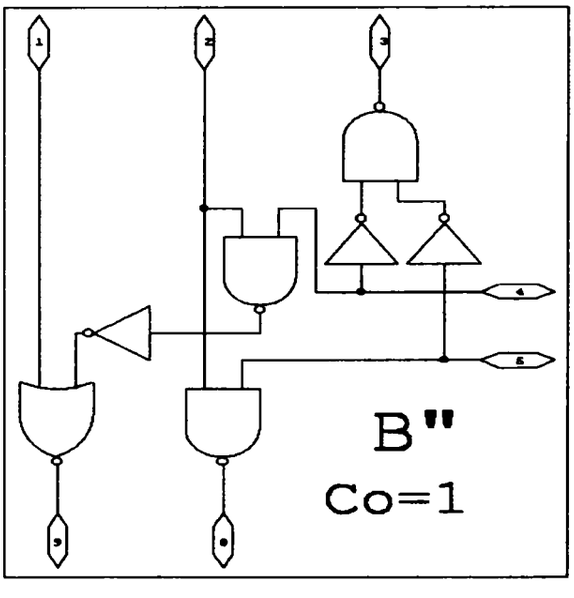

CT[7:0]
CU[7:0]
c+1
c+0
c+out
C+in
CV[7:0]
CT7
CT6
CT5
CT4
CT3
CT2
CT1
CT0
CU7
CU6
CU5
CU4
CU3
CU2
CU1
CU0
BB AB
MuxB
Mux2-1
SelBOutB
CV7
CV6
CV5
CV4
CV3
CV2
CV1
CV0
Selector1,2,3


BB
AB
SelB
OutB
MuxB


AB
BB
SelB
OutB
Mux2-1


Si
Di
Ai
A'
Gi
Pi
Co=1


1
2
3
4
5
B'
Co=1


1
2
3
4
5
B''
Co=1

ANEXA 2

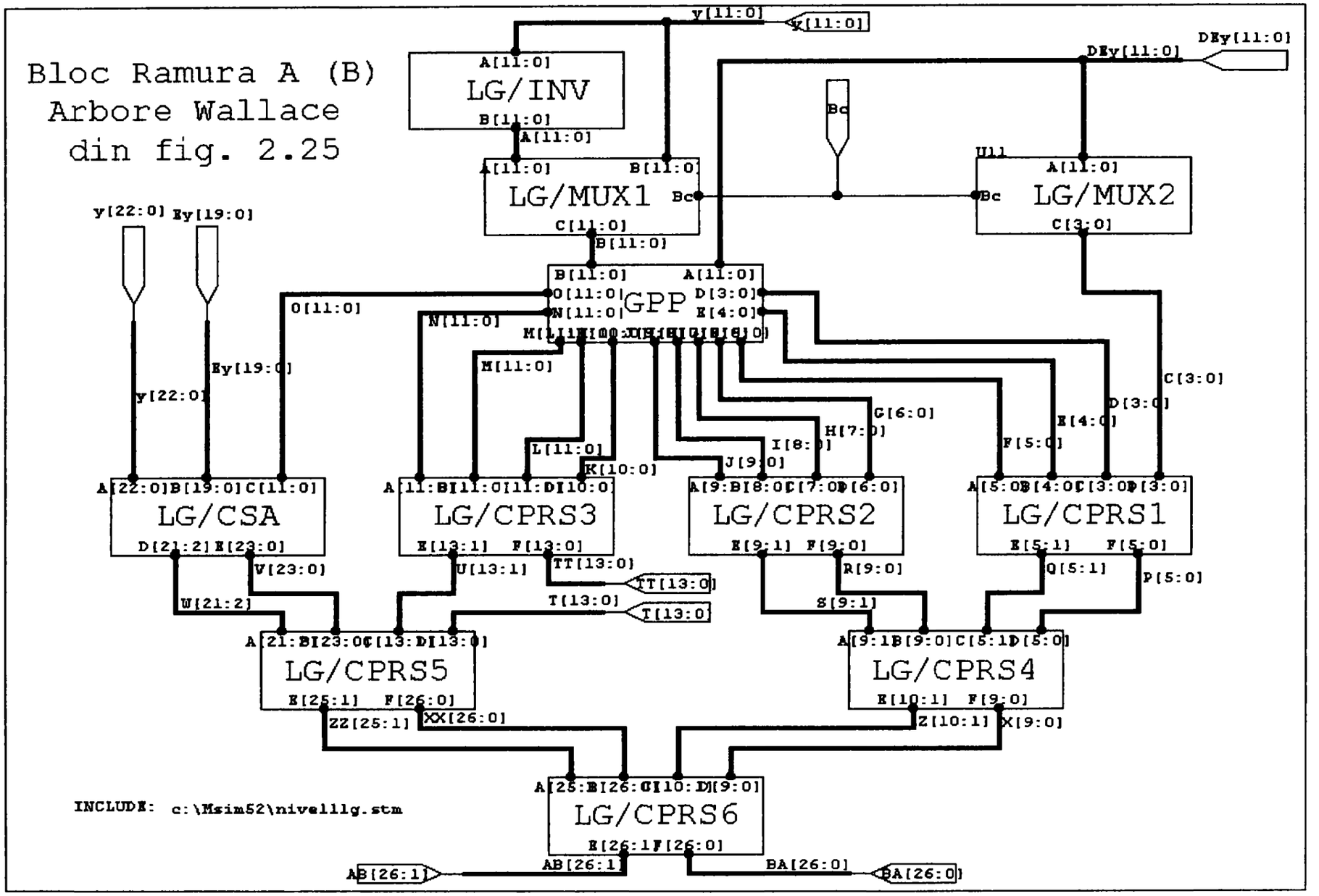
```
#include <iostream.h>
#include <stdio.h>                    // PROGRAM PENTRU GASIREA CELUI MAI DEFAVORABIL
#include <conio.h>                    // CAZ LA SUMATORUL CU ANTICIPAREA TRANSPORTULUI
#include <stdlib.h>                   // (CARRY LOOCK-AHEAD ADDER) PE 8 BITI
#define marime 8
typedef struct
    {
    char g,
         p,
         s,
         c,
         mem1,
         mem2,
         mem3;
    } stare;
long double rang (int nb,int exp);
void genereaza_sir(long double a,char *ch);
char or(char a,char b);
char xor(char a,char b);
char and(char a,char b);
void init_stare(stare *gp1,stare *gp2,stare *gp3,stare *gp4);
int compara(stare stare_1,stare stare_2);
stare element_a(char x,char y, char c,stare a);
stare element_b(stare *gp, stare b,char c);
int carry_lookahead (long int _a_,long int _b_,long int _c_);

long double rang (int nb,int exp)
{
long double p;
int i;
if(exp==0)                        // caz rang 0
  {
  p=1;
  }
p=nb;                            // caz rang 1
if(exp>1)                         // caz rang > 1
  {
  for(i=1;i<exp;i++)
    {
    p=nb*p;
    }
  }
return(p);                        // trimite n^exp
}

void genereaza_sir(long double a,char *ch)
{
int i=0;
char x[marime+1];
long double max;
max=rang(2,(marime-1-i));         // calcul rang bit
for(i=0;i<marime;i++)
  {
  if(0>a-max)                     // testeaza  bitul i
    ch[i]='0';
  else
    {
    ch[i]='1';
    a=a-max;
    }
```

```
  max=max/2;
 }
ch[marime]='\0';                    // plaseaza caracter de sfarsit de sir
}
char or(char a,char b)               // SAU
{
if((a==b)&(a=='0'))
 {
 return('0');
 }
else
 {
 return('1');
 }
}
char xor(char a,char b)              // SAU-exclusiv
{
if(a==b)
 {
 return('0');
 }
else
 {
 return('1');
 }
}
char and(char a,char b)              // SI
{
if((a==b)&(a=='1'))
 {
 return('1');
 }
else
 {
 return('0');
 }
}
void init_stare(stare *gp1,stare *gp2,stare *gp3,stare *gp4)
{
int i;
gp4->g='0';                         // initializeaza fiecare camp al structurii
gp4->p='0';
gp4->mem1='0';
gp4->mem2='0';
gp4->mem3='0';
gp4->s='0';
gp4->c='0';
for(i=0;i<8;i++)                    //initializeaza pe zero ceilalti g si p
{
 gp1[i]=*gp4;
 while(i<4)
  {
  gp2[i]=*gp4;
  while(i<2)
   {
   gp3[i]=*gp4;
   break;
   }
  break;
  }
```

```
 }
}

int compara (stare stare_1,stare stare_2)
{                              // compara fiecare camp al structurii
int test=0;
if(stare_1.g!=stare_2.g)test=1;         // in caz de diferenta pune test pe 1
if(stare_1.p!=stare_2.p)test=1;
if(stare_1.mem1!=stare_2.mem1)test=1;
if(stare_1.mem2!=stare_2.mem2)test=1;
if(stare_1.mem3!=stare_2.mem3)test=1;
if(stare_1.c!=stare_2.c)test=1;
if(stare_1.s!=stare_2.s)test=1;
return(test);                       // trimite test
}

stare element_a(char x,char y, char c,stare a)
{
stare A;
A.g=and(x,y);                       // x&y=g
A.p=or(x,y);    //p                 // x|y=p
A.s=a.mem3;     //s                  // + 1 timp de propagare=s
A.mem3=xor(a.mem2,x);                  // (y^c)^x
A.mem2=a.mem1;                        // + 1 timp de propagare
A.mem1=xor(y,c);                     // y^c
A.c='\0';                          // neutilizat
return (A);                         // trimite la
}

stare element_b(stare *gp,stare b ,char c)
{
stare B;
B.g=or(gp[0].g,b.mem1);               // g1|(p1&g2)=G
B.mem1=and(gp[0].p,gp[1].g);           // p1&g2
B.p=and(gp[0].p,gp[1].p);             // p1&p2=P
B.c=or(gp[1].g,b.mem2);               // g2|(p2&c)=C
B.mem2=and(gp[1].p,c);                // p2&c
B.mem3='0';                         // neutilizat
B.s='0';                           // neutilizat
return(B);                          //trimite b
}

int carry_lookahead (long int _a_,long  int _b_,long  int _c_)//gest. etapelor
{
int i,j=0,test=1;
char a[marime+1],b[marime+1],c[marime+1],C[marime+1],s[marime+1];
stare ch,gp1[marime],gp2[marime/2],gp3[2],gp4;

init_stare(gp1,gp2,gp3,&gp4);
genereaza_sir(_c_,C);               // creeaza un sir pentru C nou
genereaza_sir(_c_,c);                      // creeaza un sir pt. c vechi
genereaza_sir(_a_,a);               // creeaza un sir pt. a
genereaza_sir(_b_,b);               // creeaza un sir pt. b
while(test!=0)
  {
  test=0;                          // initializare test
  j++;
                                          //  nivel 4: 1 element
  ch=element_b(gp3,gp4,c[7]);          // ultima celula B opereaza
  test=test+compara(gp4,ch);           // test de stabilitate
```

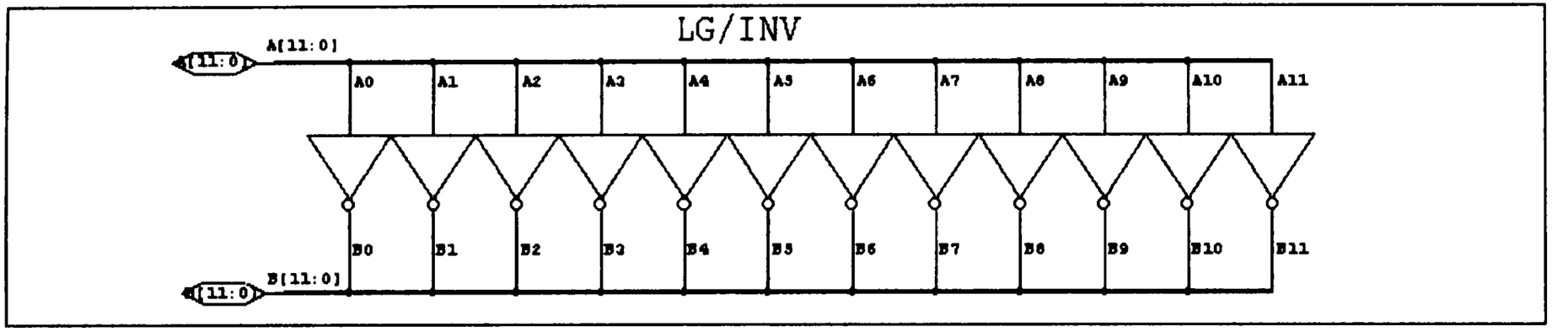
```
 gp4=ch;                        // actualizarea acestui element
 C[3]=ch.c;                     // memoreaza transportul
 for (i=0;i<2;i++)                // nivel 3: 2 elemente
  {
  ch=element_b(gp2+2*i,gp3[i],c[3+4*i]);// cele 2 celule B opereaza
  test=test+compara(gp3[i],ch);     // test de stabilitate
  gp3[i]=ch;                    // actualizarea acestor elemente
  C[1+4*i]=ch.c;                  // memoreaza transporturile
  }
 for (i=0;i<4;i++)                //al doilea nivel: 4 elemente
  {
  ch=element_b(gp1+2*i,gp2[i],c[1+2*i]); //cele 4 celule B opereaza
  test=test+compara(gp2[i],ch);     // test de stabilitate
  gp2[i]=ch;                    // actualizarea acestor elemente
  C[2*i]=ch.c;                   // memoreaza transporturile
  }
 for (i=0;i<marime;i++)             // nivel 1: 8 elemente
  {
  ch=element_a(a[i],b[i],c[i],gp1[i]);//cele 8 celule A opereaza
  test=test+compara(gp1[i],ch);     // test de stabilitate
  gp1[i]=ch;                    // actualizarea acestor elemente
  if (c[i]!=C[i])test=1;           // test de stabilitate a transp.
  c[i]=C[i];                          // actualizarea transp.
  }
 }
return(j);                      // trimite numarul de etape
}

void main()
{
double max;
unsigned long int a,b,c,i;
int t,t_max=0;
max=rang(2,marime);             // calcul numar maxim in functie de numar de biti
for(a=0;a<max;a++)                // bucle pt. toate combinatiile a,b,c
 {
 for(b=0;b<max;b++)
  {
  for(c=0;c<2;c++)
   {
   t=carry_lookahead(a,b,c);       // calculeaza timpul de calcul
   if(t_max<t)                  // daca s-a depasit valoarea maxima
         {
         i=0;                    // initializeaza i
         t_max=t;                  // memoreaza aceasta valoare
         printf("carry_lookahead:\n");  // tipareste
         printf("timp de propagare:%d  valori:\n",t_max);
         }
   if(t_max==t)                 // pt. acest timp de propagare
         {
         printf("%ld,%ld,%ld\t",a,b,c); // tipareste valorile a, b si c
         i++;                    // incrementeaza i
         }
   }
  }
 }
printf("\ntimpul maxim este %d timpi de propagare pentru %ld valori ale lui a,b,c",t_max,i);
getch();                        // tipareste rezultatul final si asteapta
}
```

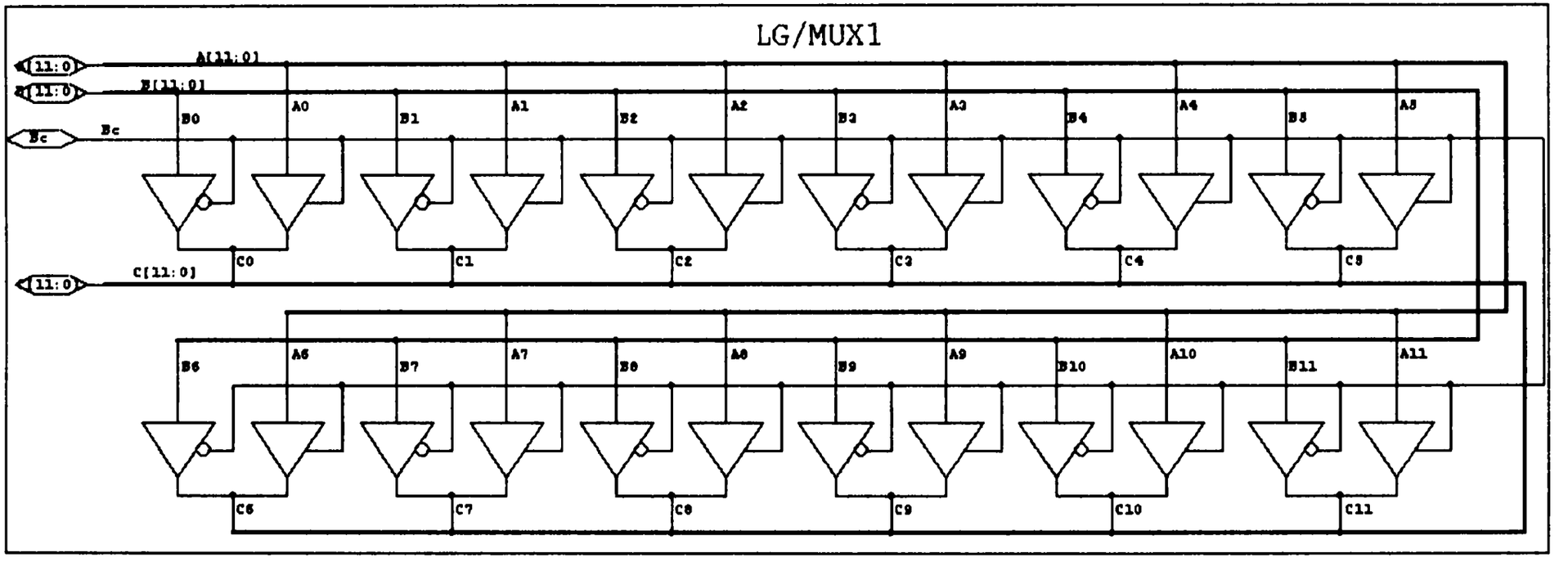
ANEXA 3



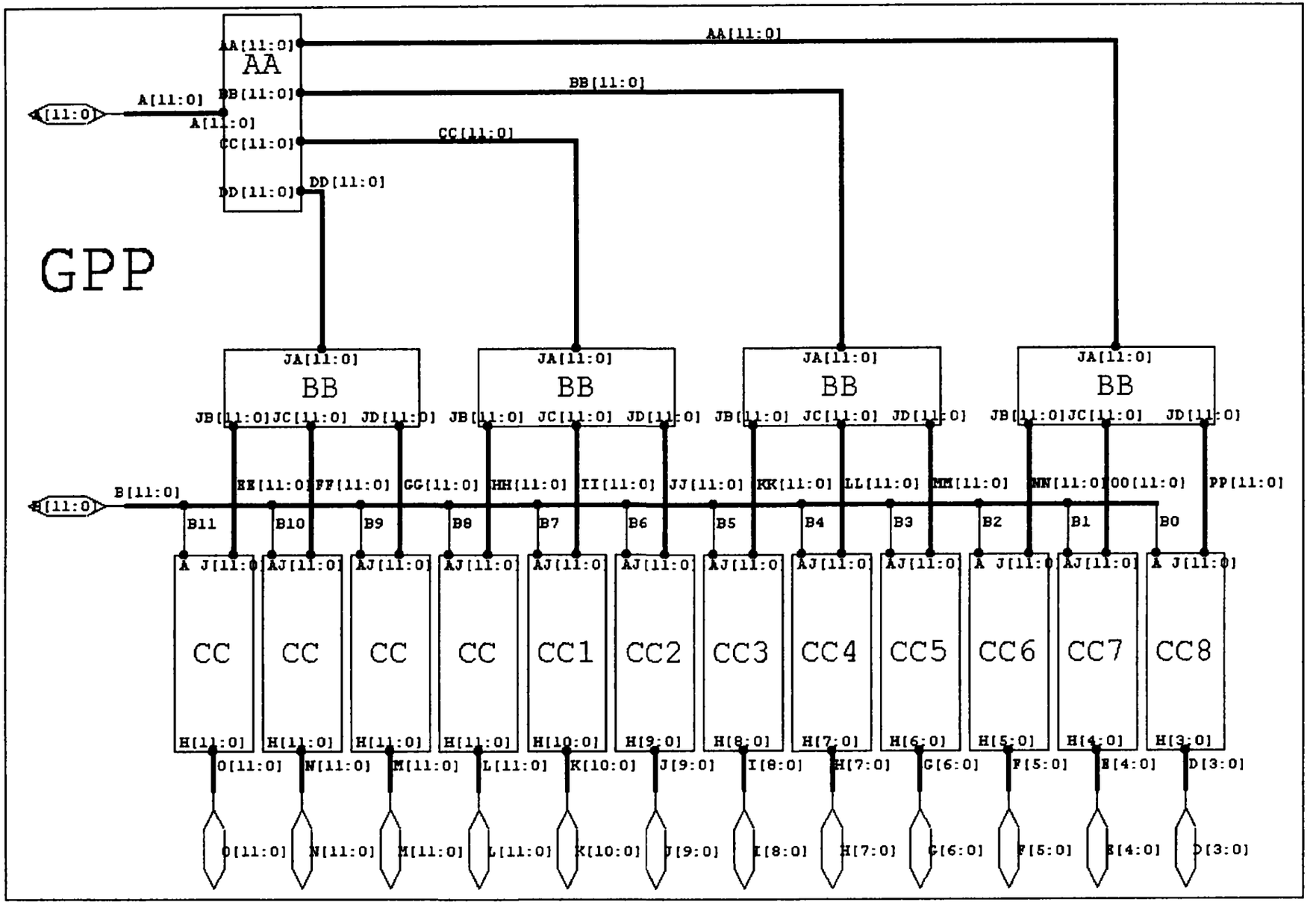
A3. 1

ANEXA 3


LG/INV
A[11:0]
A[11:0]
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B[11:0]
B[11:0]

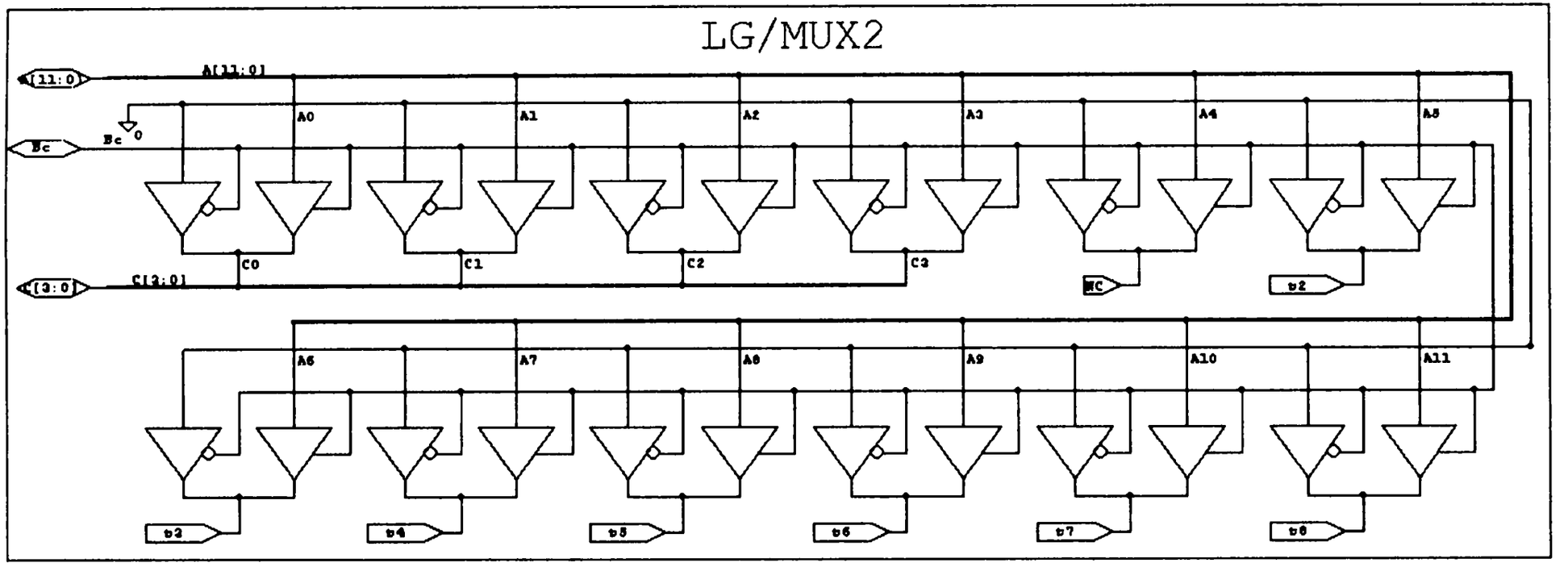

LG/MUX1
11:0
A[11:0]
11:0
B[11:0]
Sc
Sc
B0
A0
B1
A1
B2
A2
B3
A3
B4
A4
B5
A5
C0
C1
C2
C3
C4
C5
11:0
C[11:0]
B6
A6
B7
A7
B8
A8
B9
A9
B10
A10
B11
A11
C6
C7
C8
C9
C10
C11

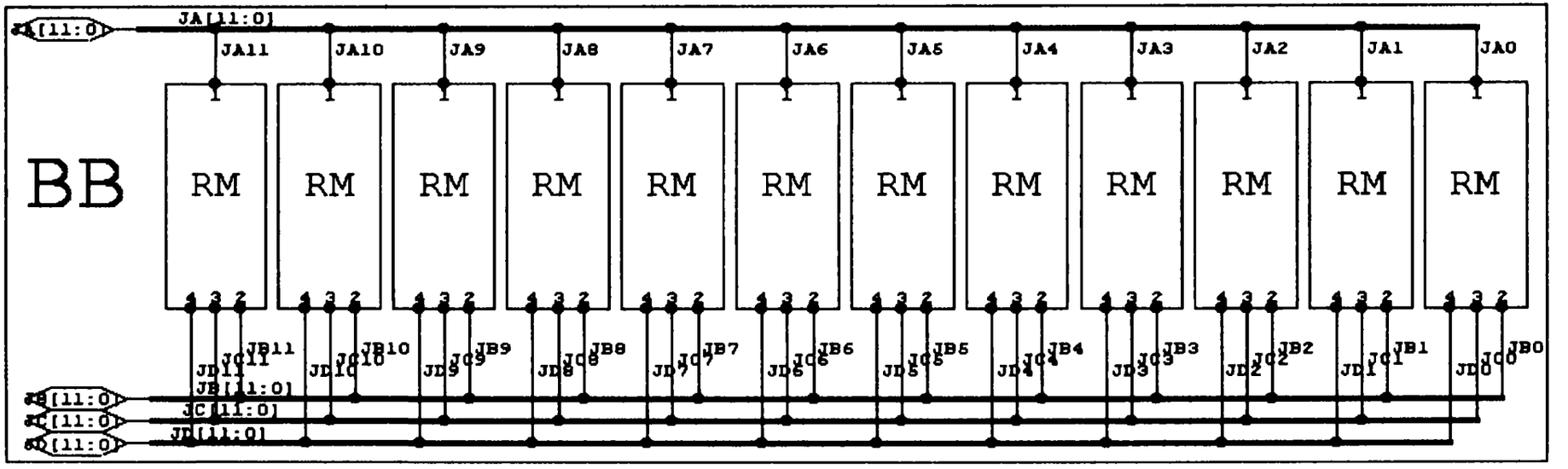
ANEXA 3



A3.3

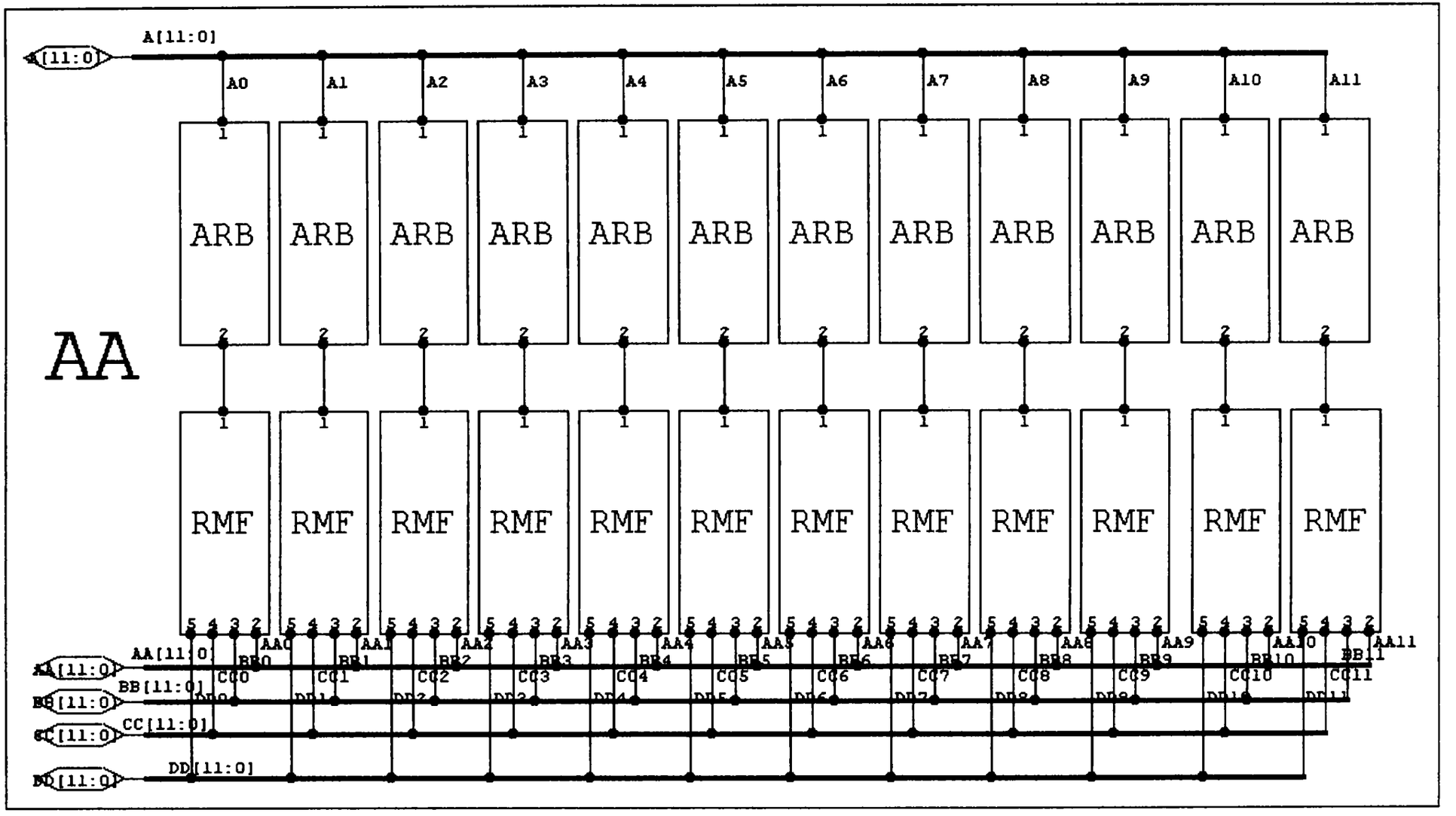
ANEXA 3

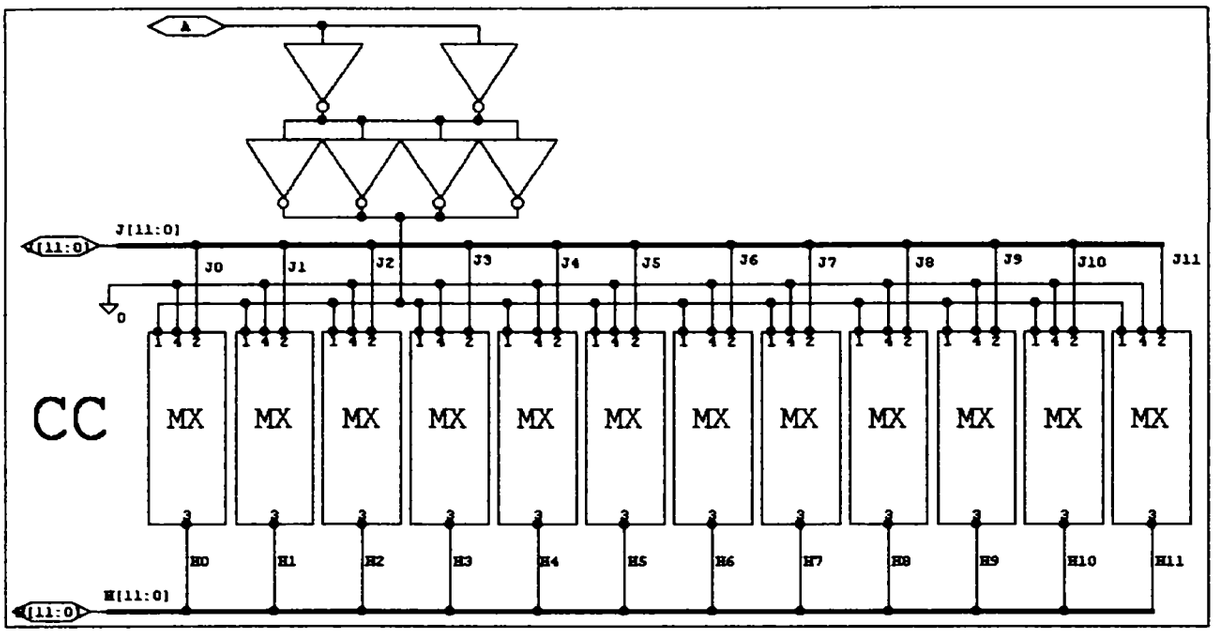


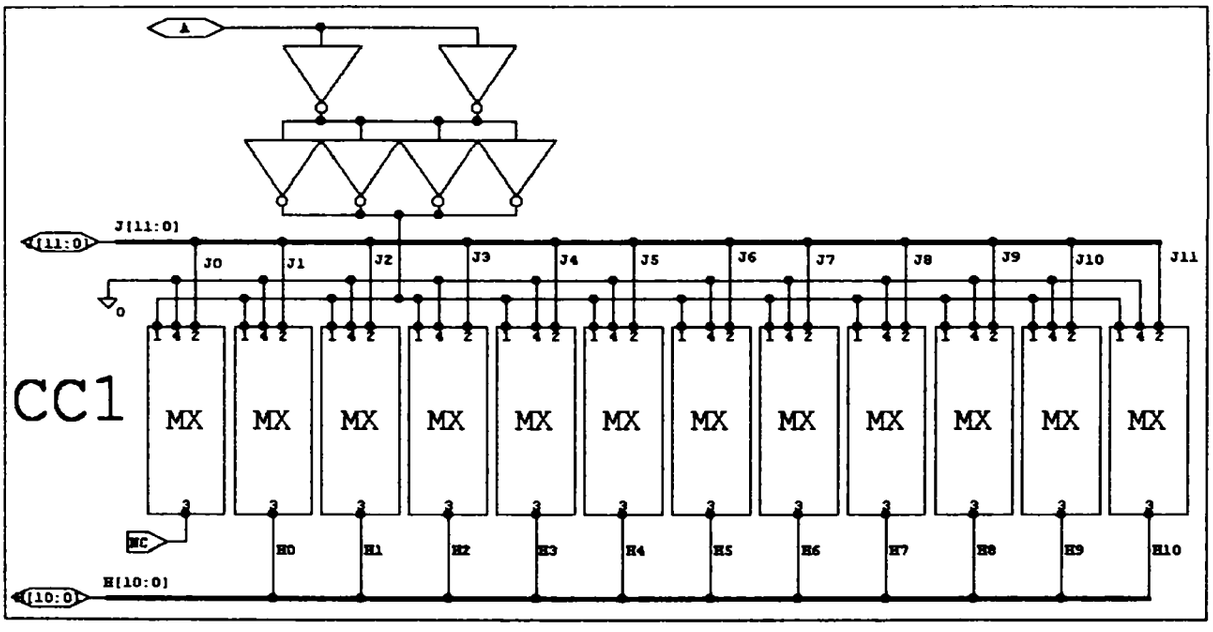


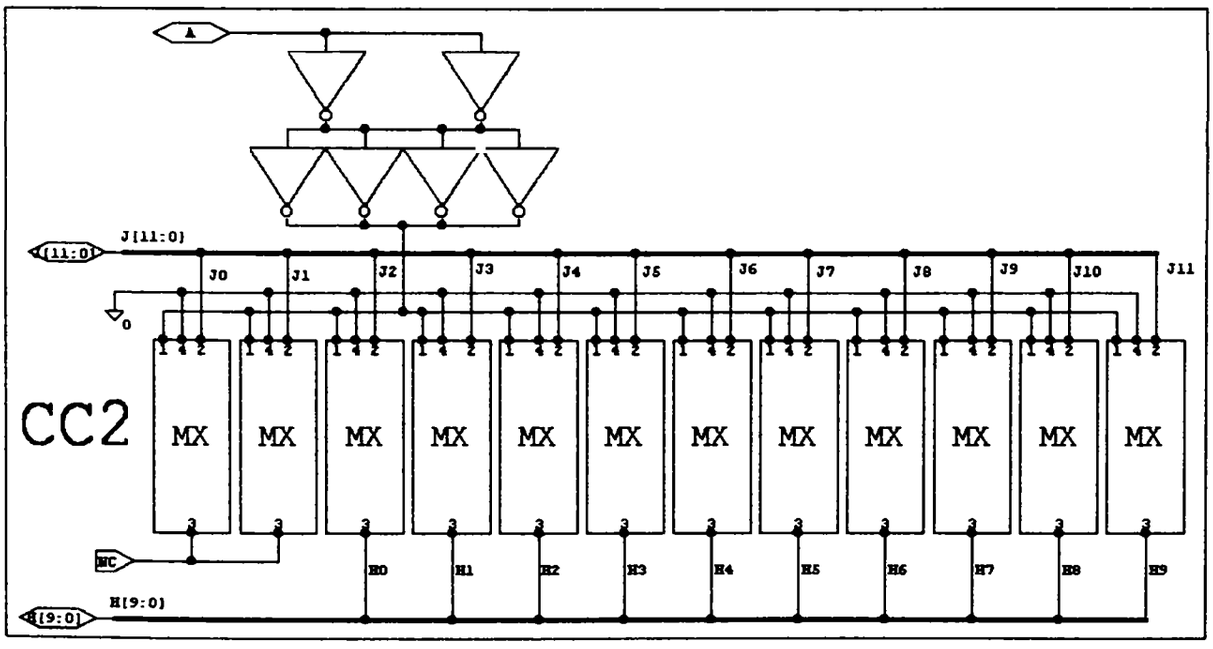
A3.4

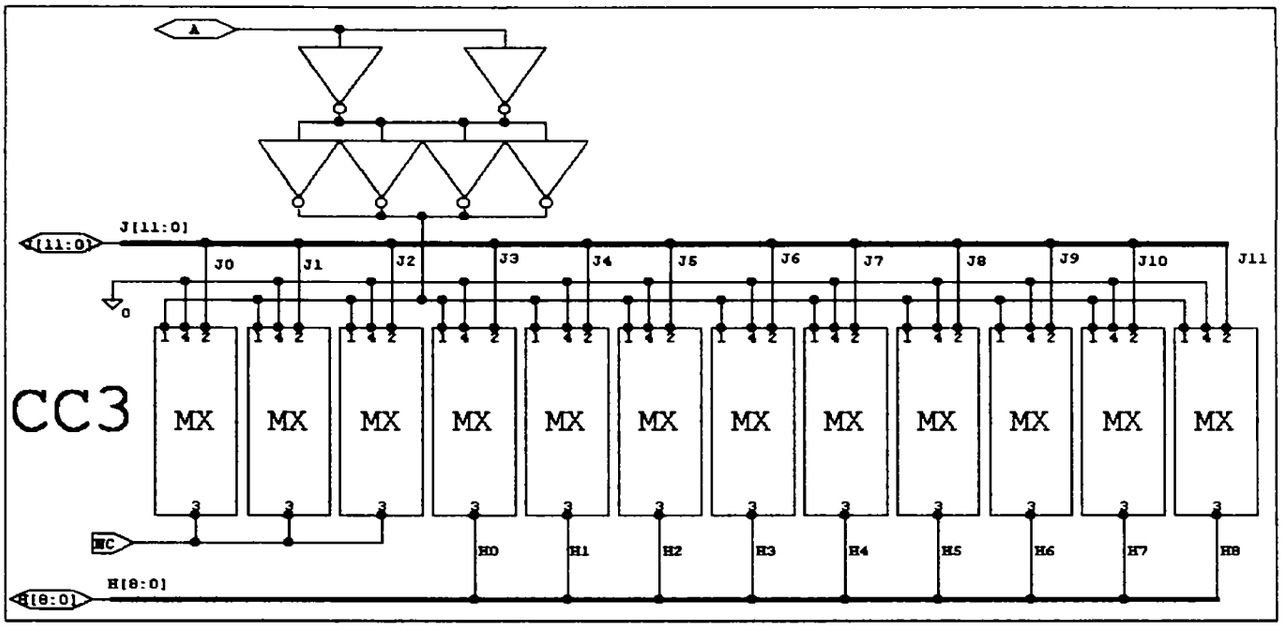
ANEXA 3

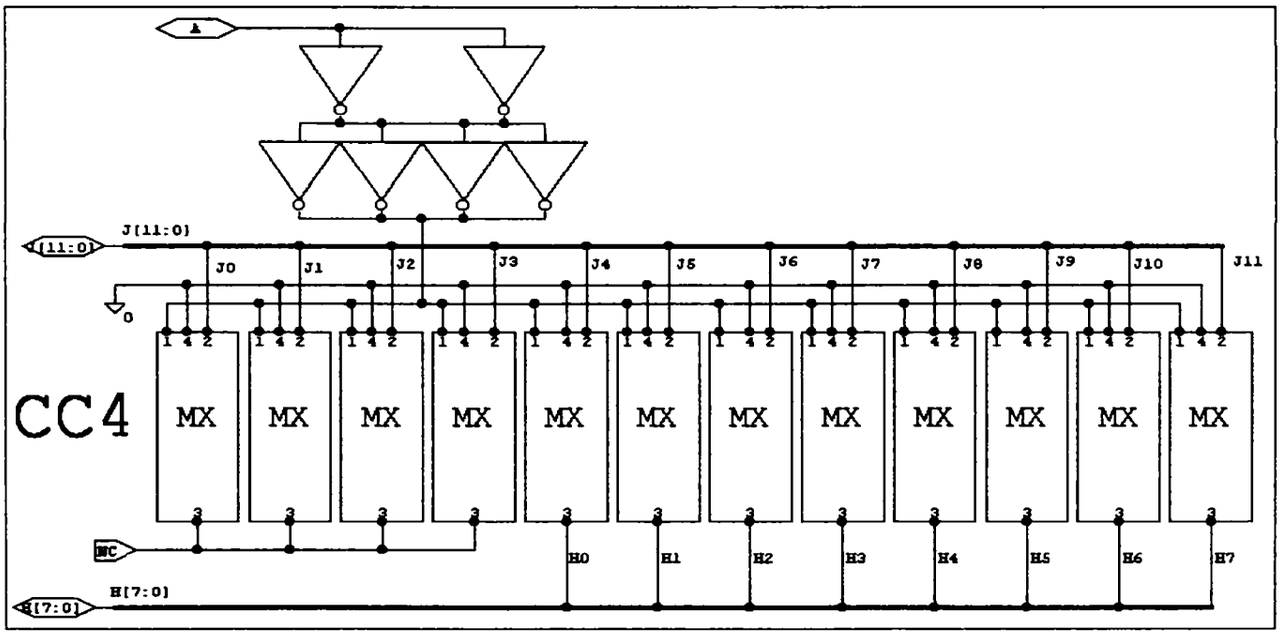

A[11:0]
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
ARB
AA
RMF
AA[11:0]
BB[11:0]
CC[11:0]
DD[11:0]

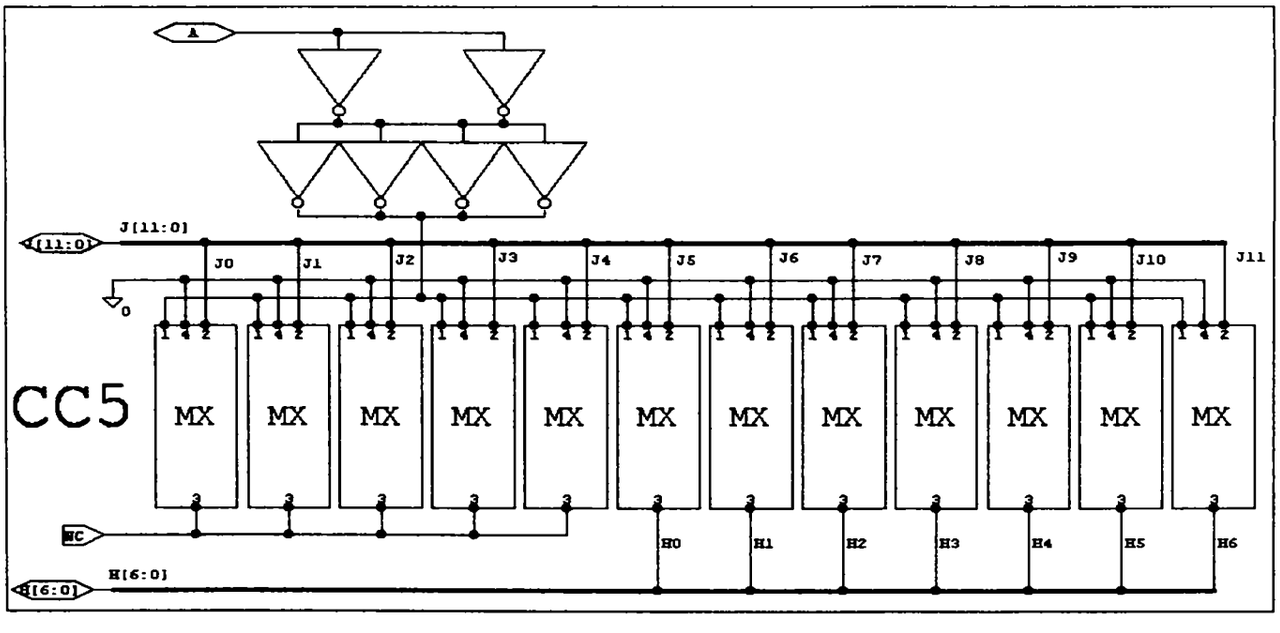

A
J[11:0]
{11:0}
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC
MX
H0
H1
H2
H3
H4
H5
H6
H7
H8
H9
H10
H11
H[11:0]
{11:0}

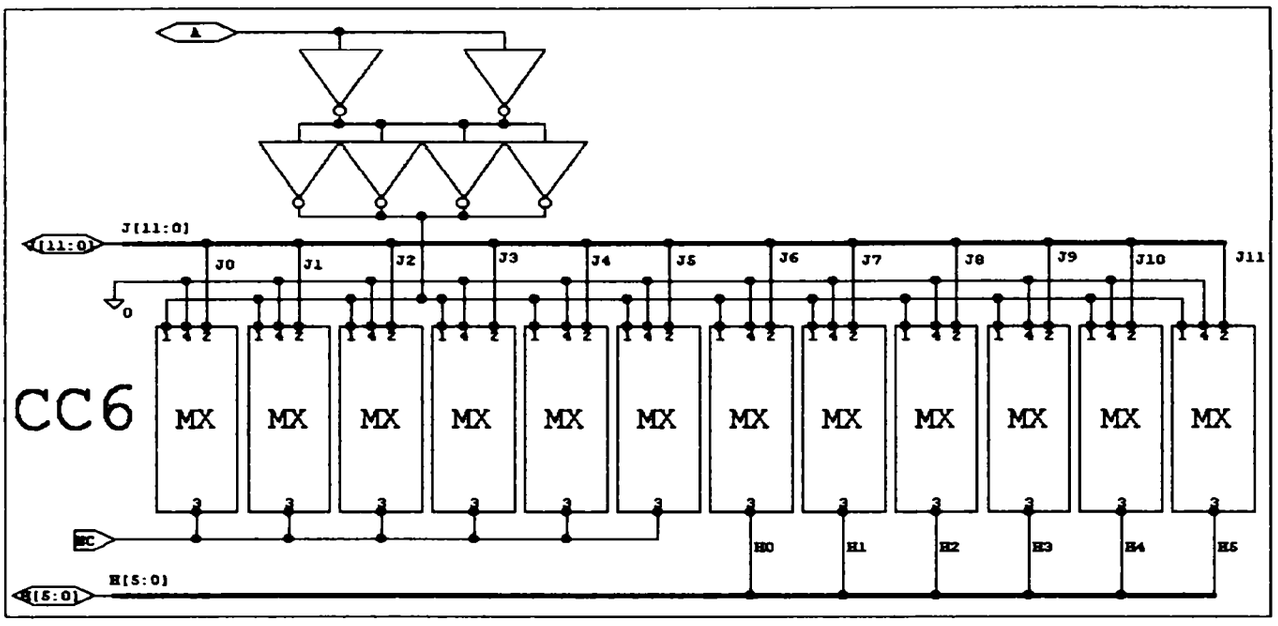

A
J[11:0]
{11:0}
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC1
MX
NC
H0
H1
H2
H3
H4
H5
H6
H7
H8
H9
H10
H[10:0]
{10:0}

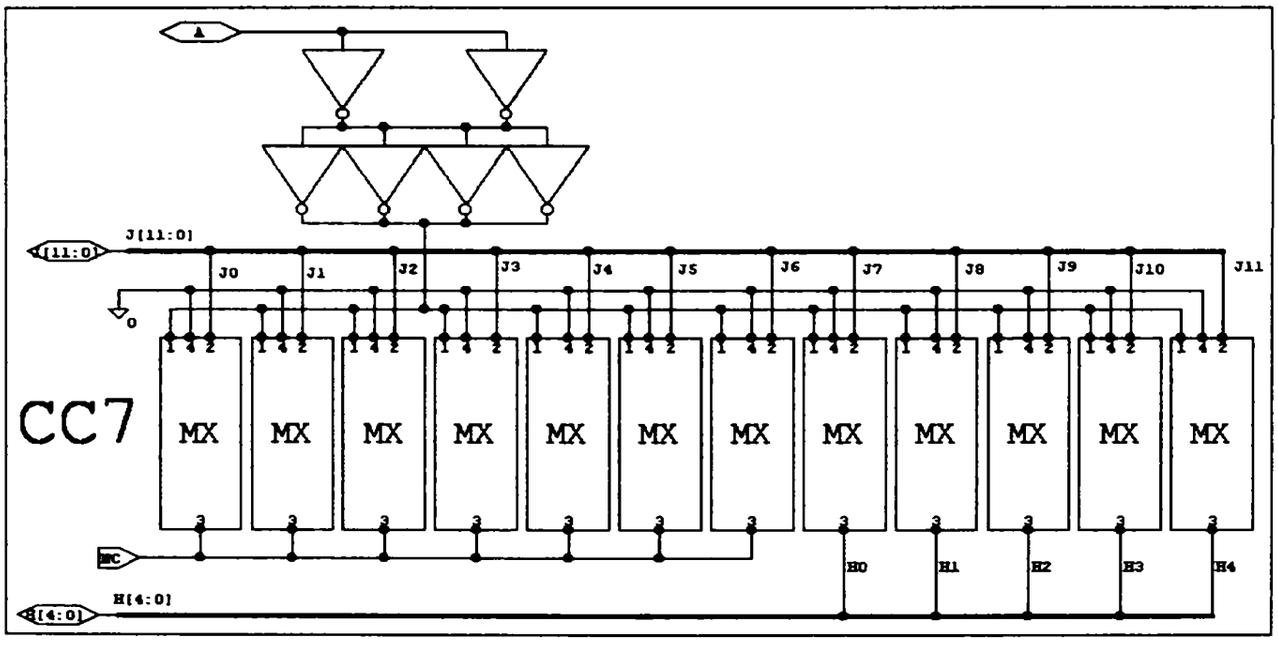

A
J[11:0]
{11:0}
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC2
MX
NC
H0
H1
H2
H3
H4
H5
H6
H7
H8
H9
H[9:0]
{9:0}

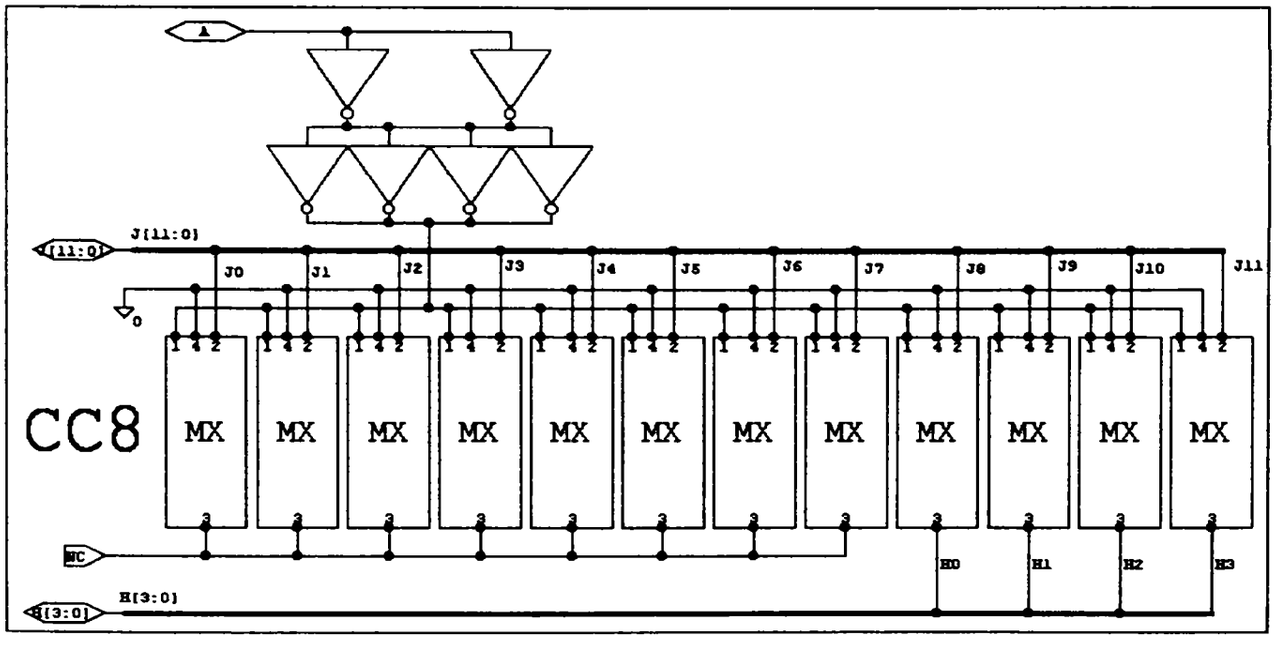

A
J[11:0]
J[11:0]
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC3
MX
NC
H0
H1
H2
H3
H4
H5
H6
H7
H8
H[8:0]
H[8:0]

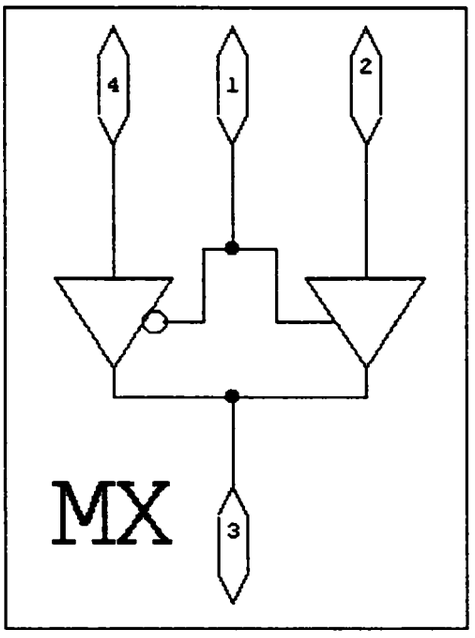

A
J[11:0]
[11:0]
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC4
MX
NC
H0
H1
H2
H3
H4
H5
H6
H7
H[7:0]
[7:0]

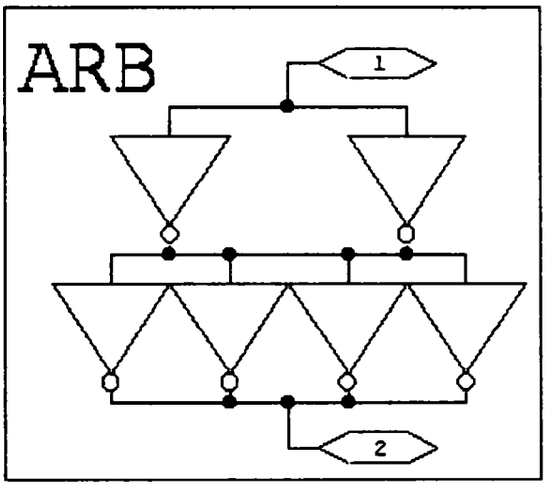

A
J[11:0]
[11:0]
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC5
MX
NC
H0
H1
H2
H3
H4
H5
H6
H[6:0]
[6:0]

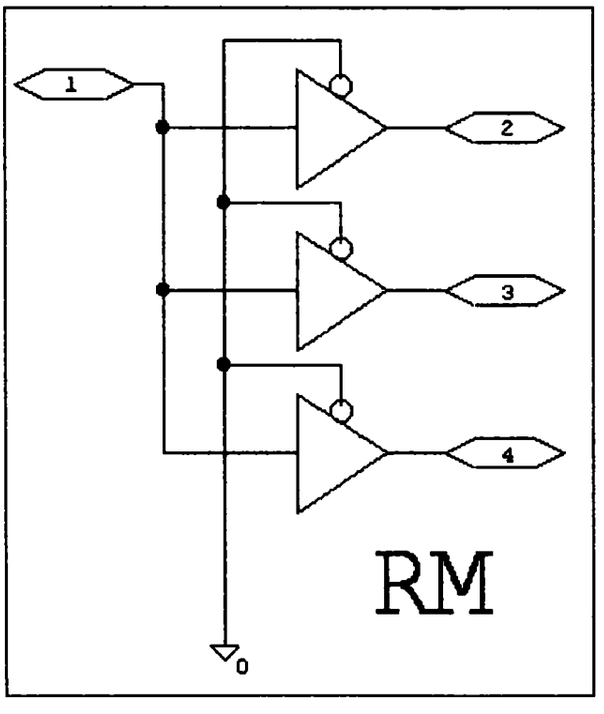

A
J[11:0]
J[11:0]
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC6
MX
SC
H0
H1
H2
H3
H4
H5
H[5:0]
S[5:0]

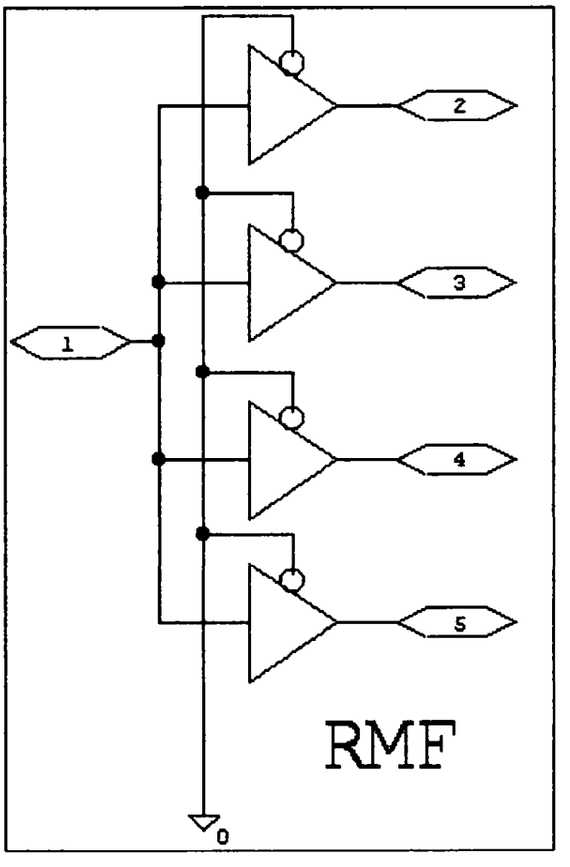

A
J[11:0]
J[11:0]
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC7
MX
SC
H0
H1
H2
H3
H4
H[4:0]
S[4:0]

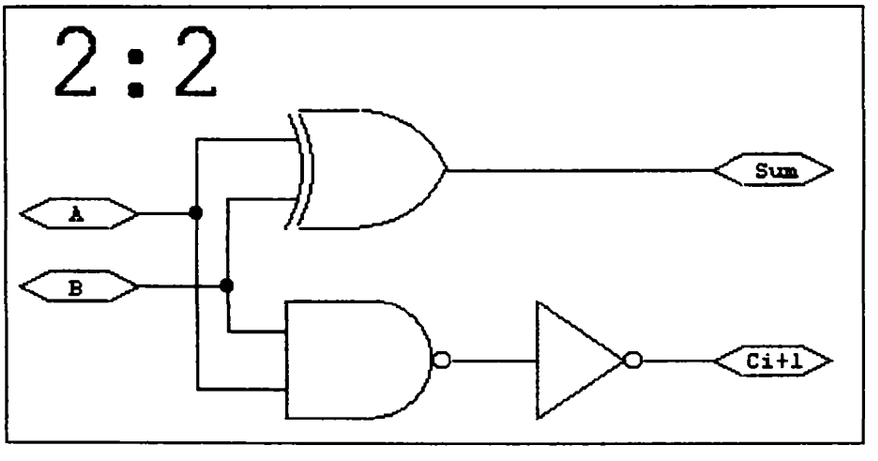

A
J[11:0]
J[11:0]
J0
J1
J2
J3
J4
J5
J6
J7
J8
J9
J10
J11
0
CC8
MX
SC
H0
H1
H2
H3
H[3:0]
S[3:0]



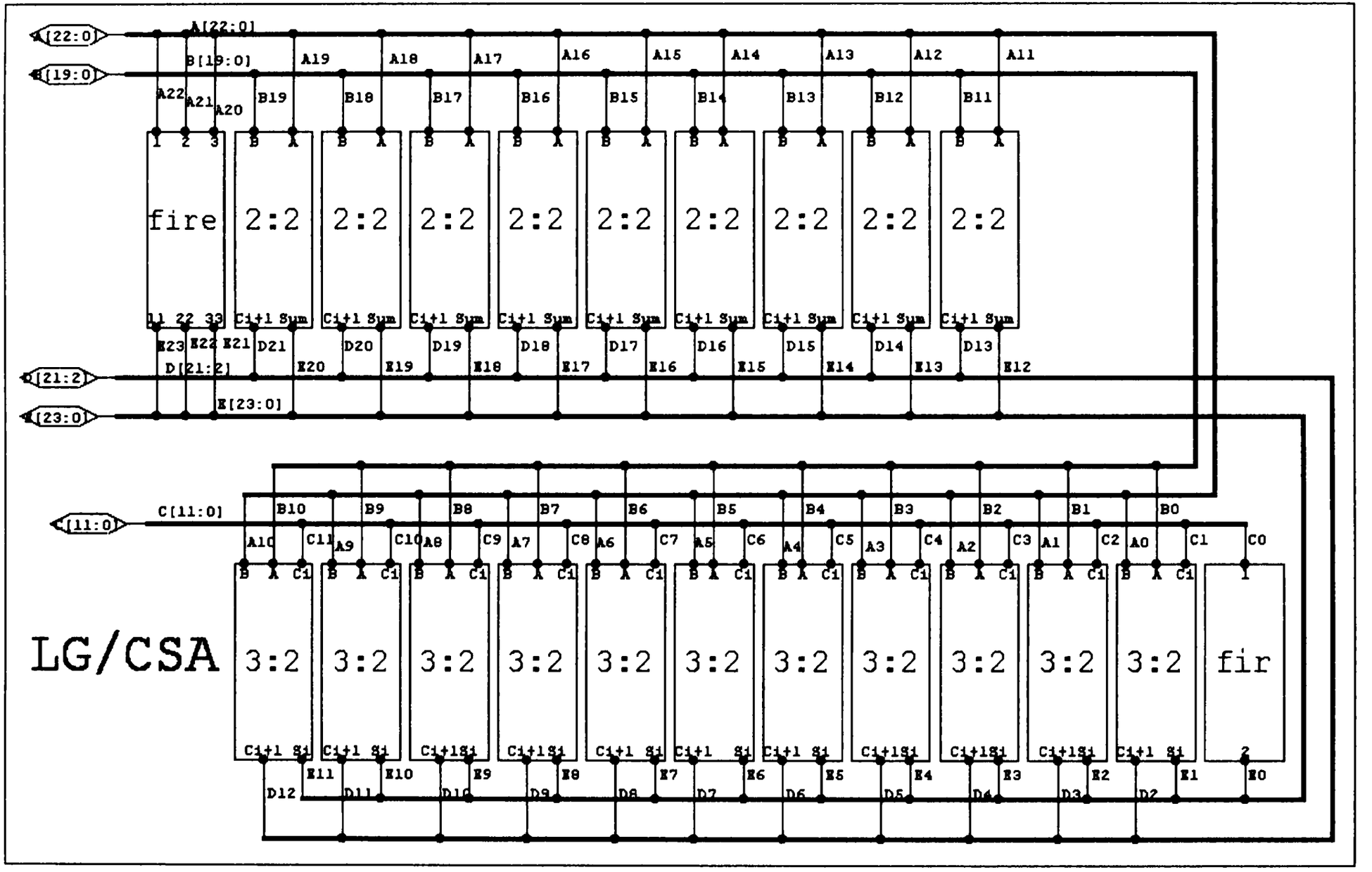


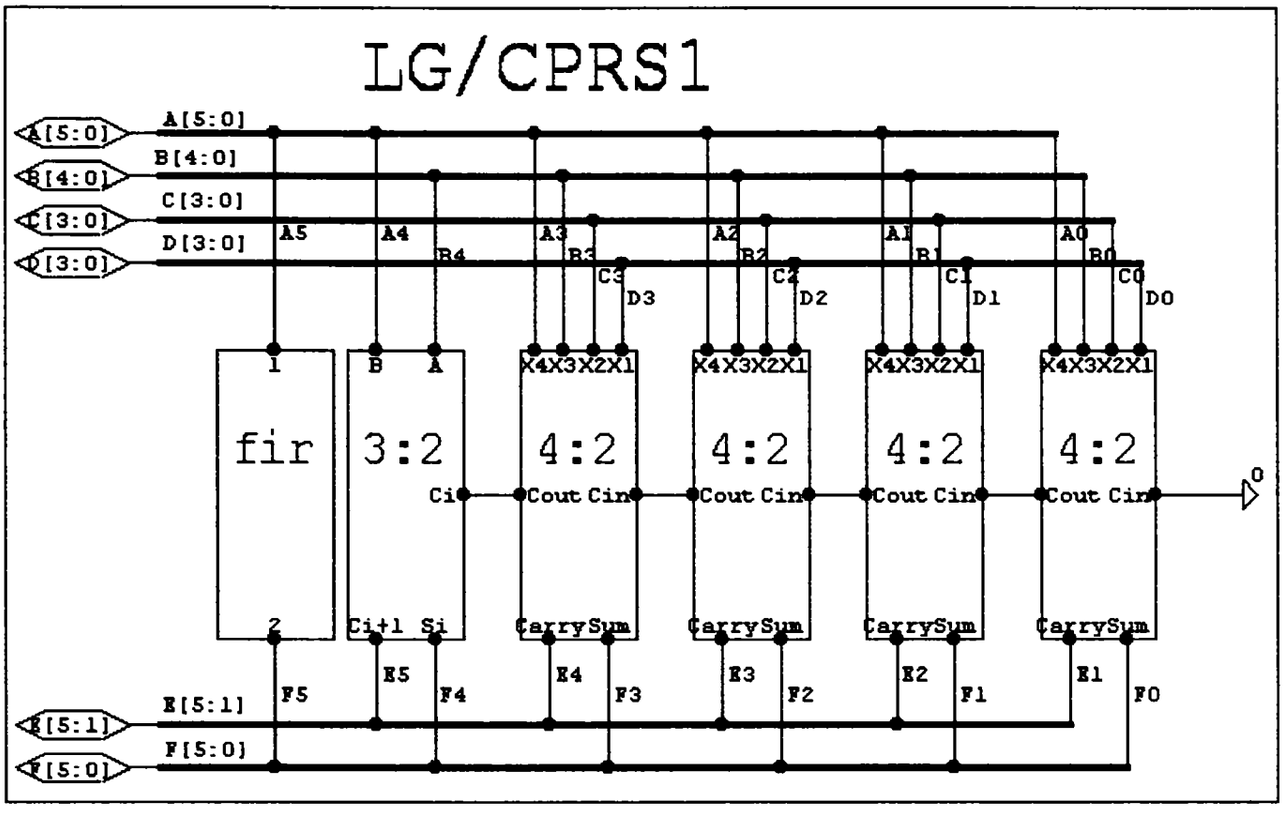


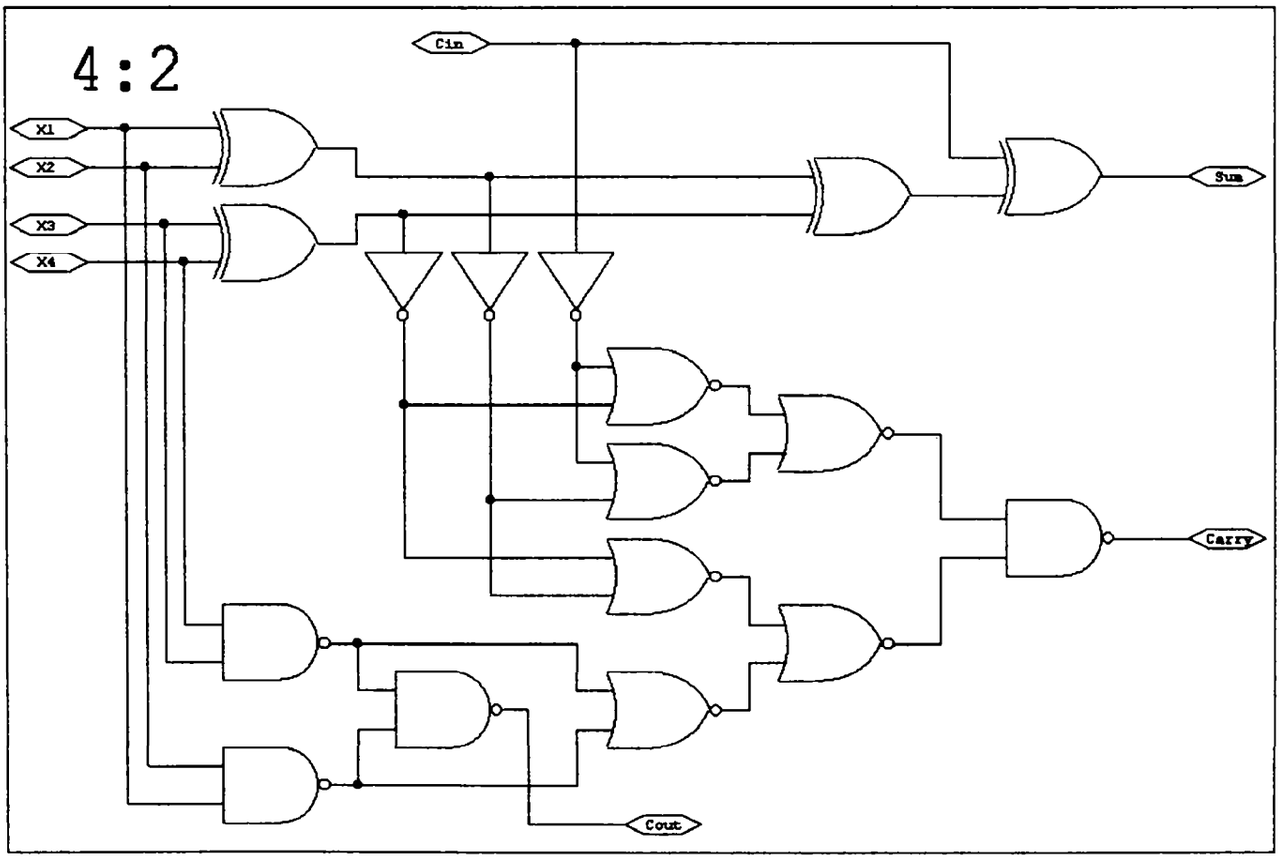


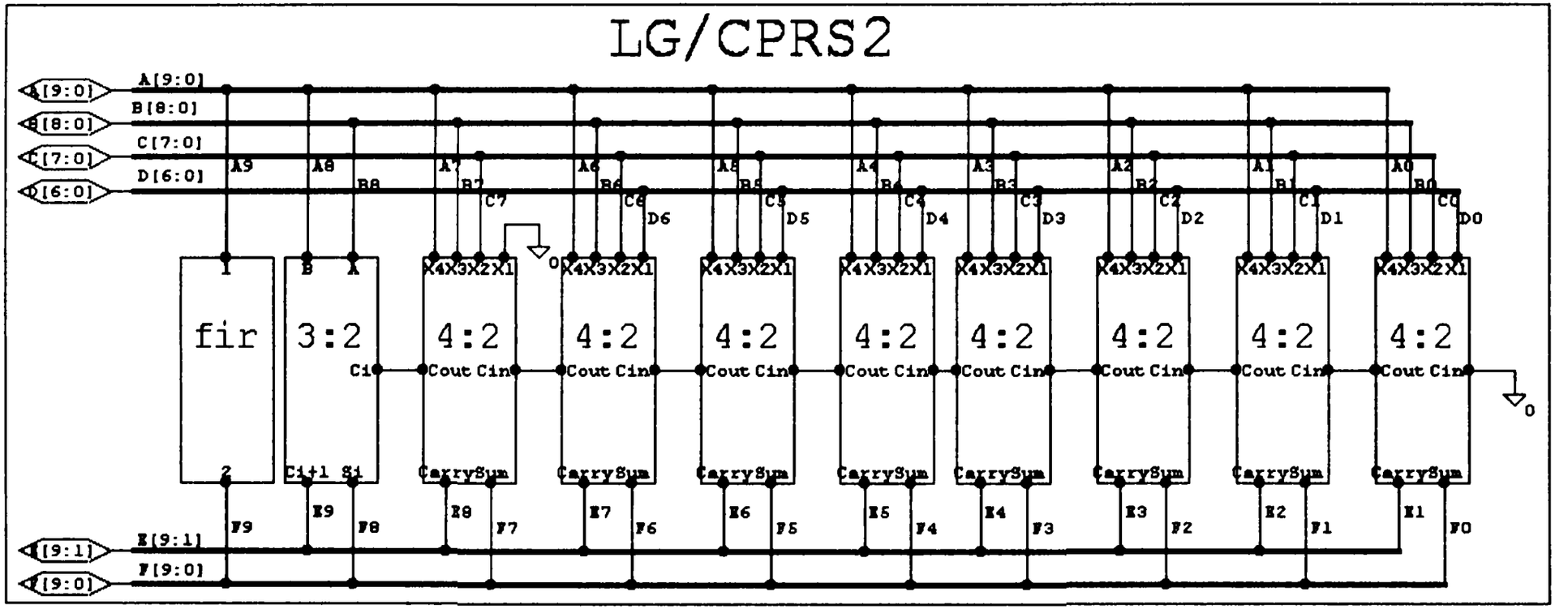


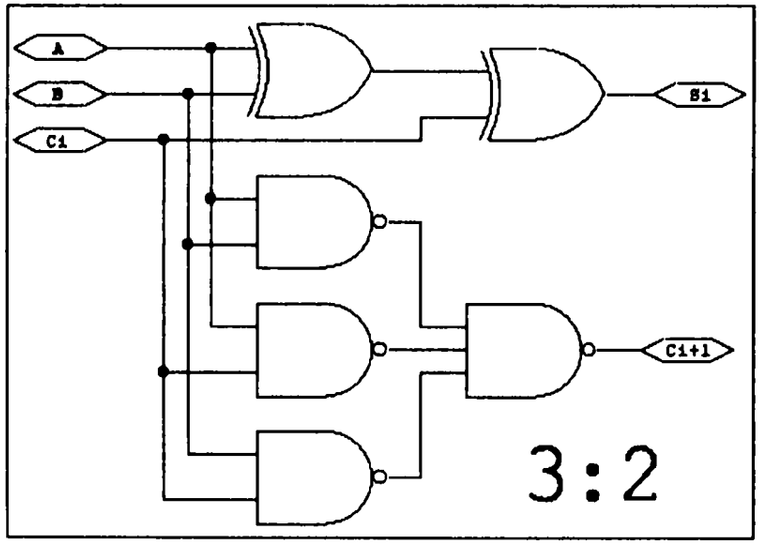
ANEXA 3




LG/CPRS1
A[5:0]
B[4:0]
C[3:0]
D[3:0]
E[5:1]
F[5:0]
fir
3:2
4:2
Ci
Cout Cin
CarrySum
O

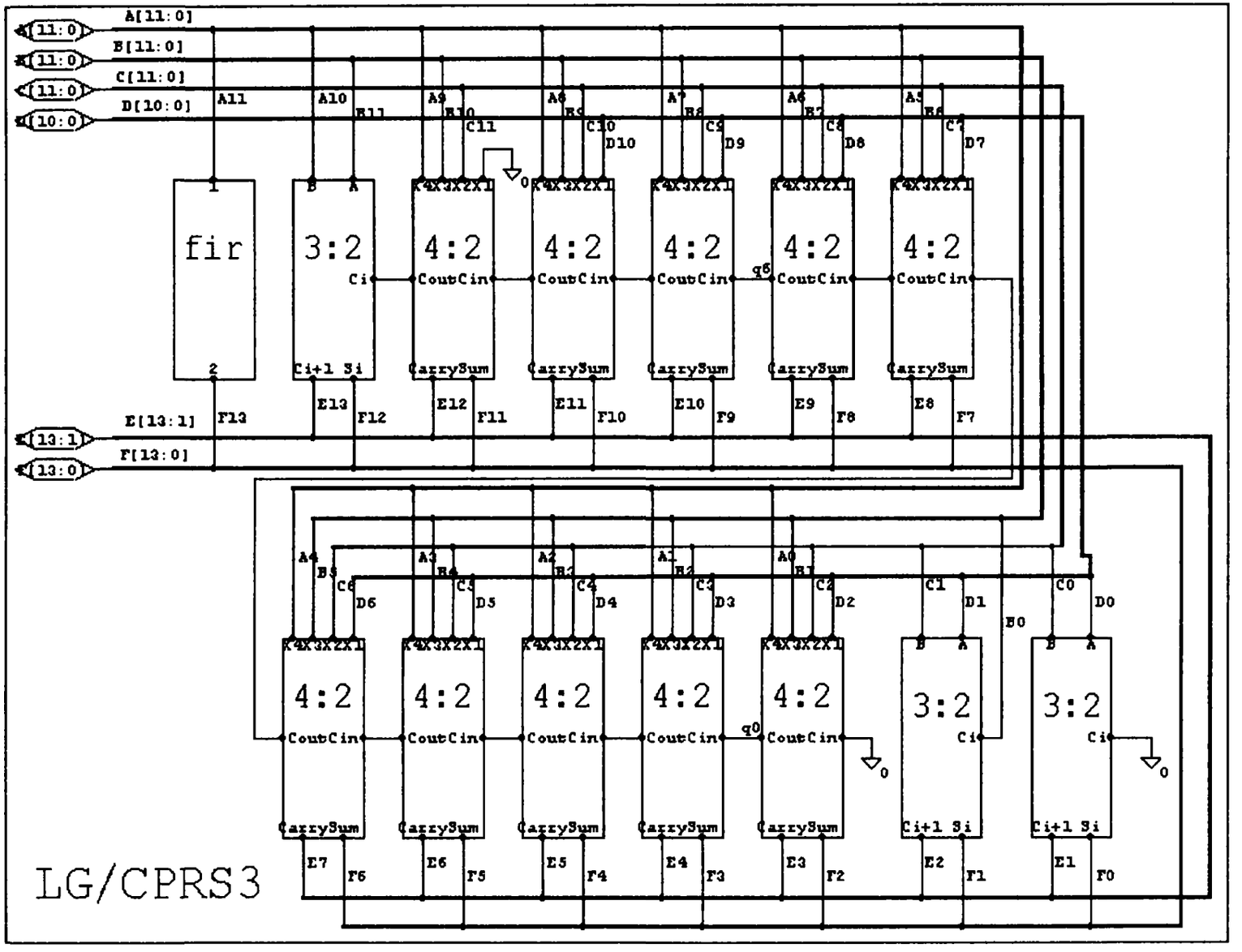

4:2
Cin
X1
X2
X3
X4
Sum
Carry
Cout


LG/CPRS2
A[9:0]
B[8:0]
C[7:0]
D[6:0]
fir
3:2
4:2
Cout Cin
CarrySum
X4X3X2X1
E[9:1]
F[9:0]

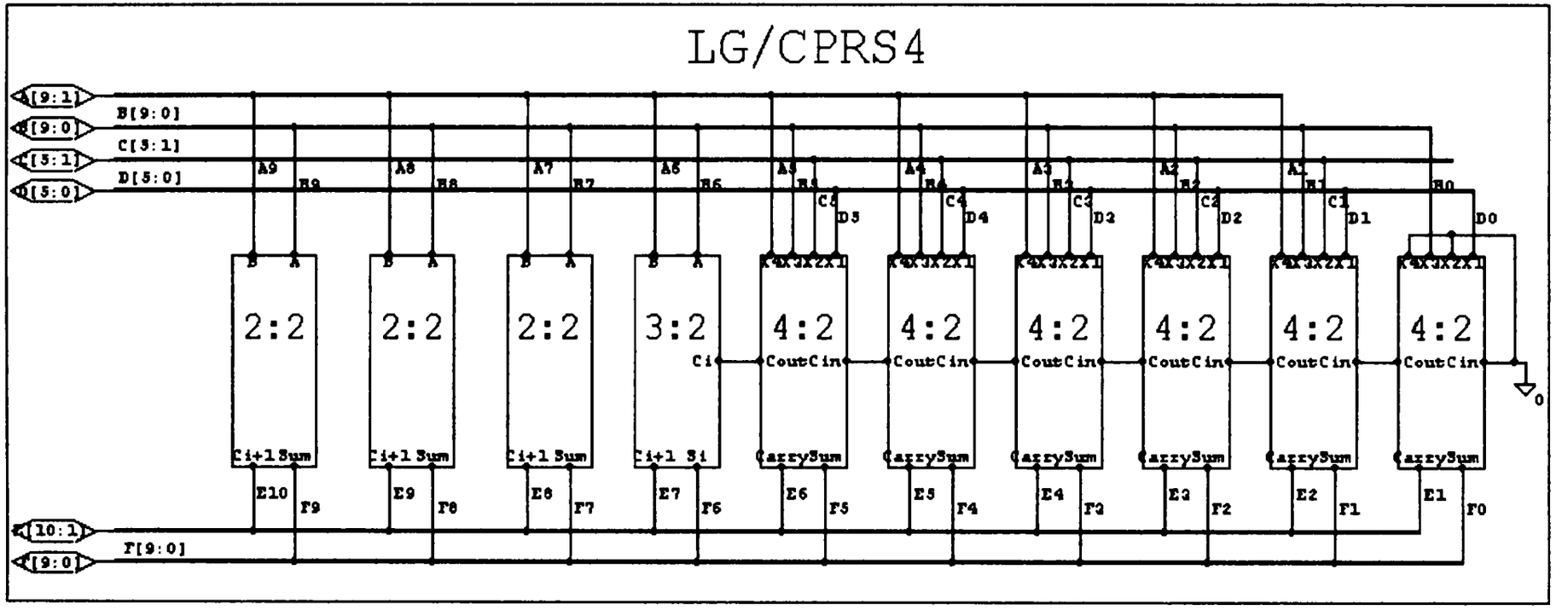

A
B
Ci
Si
Ci+1
3:2

A3.12

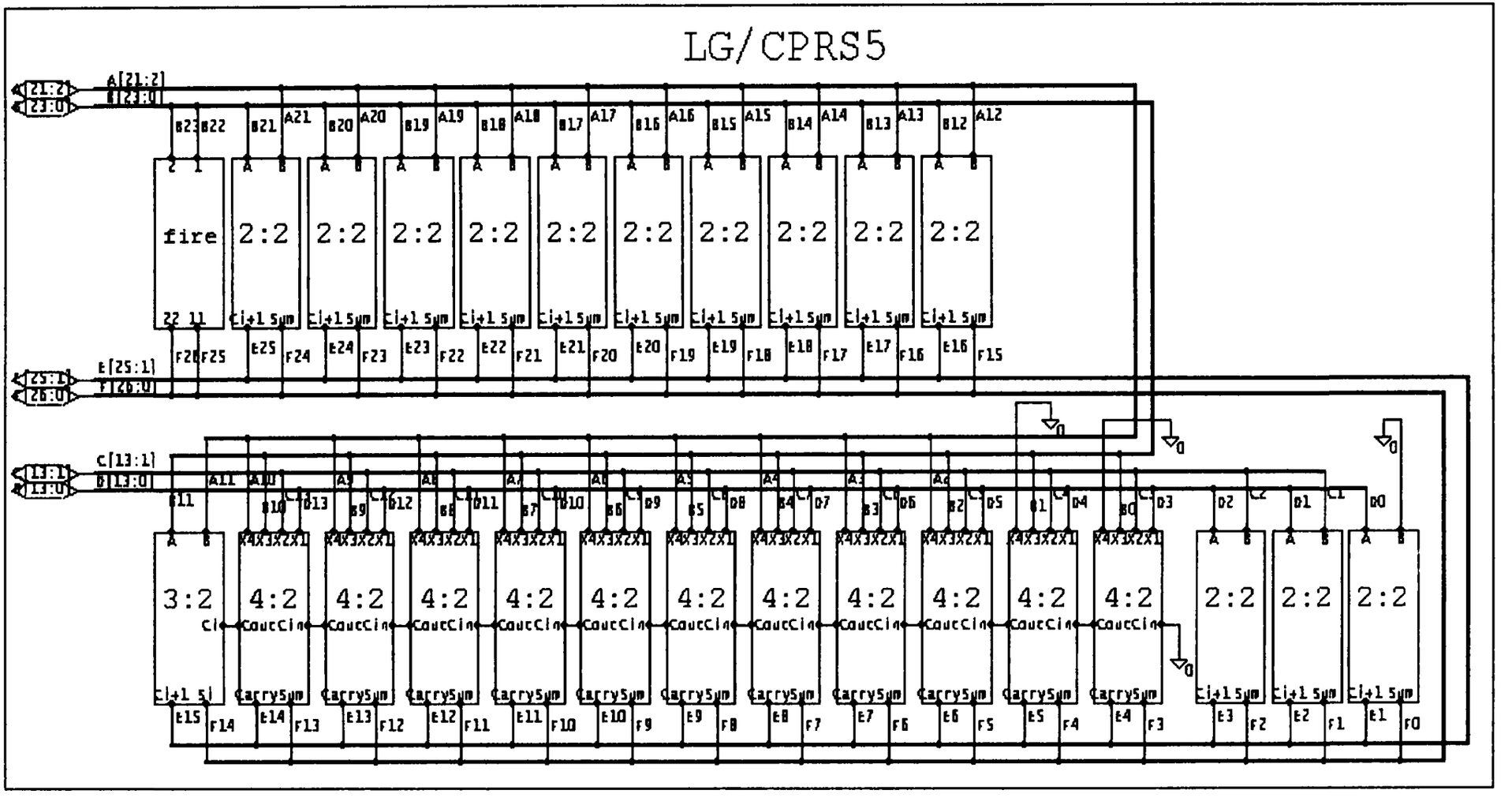
ANEXA 3

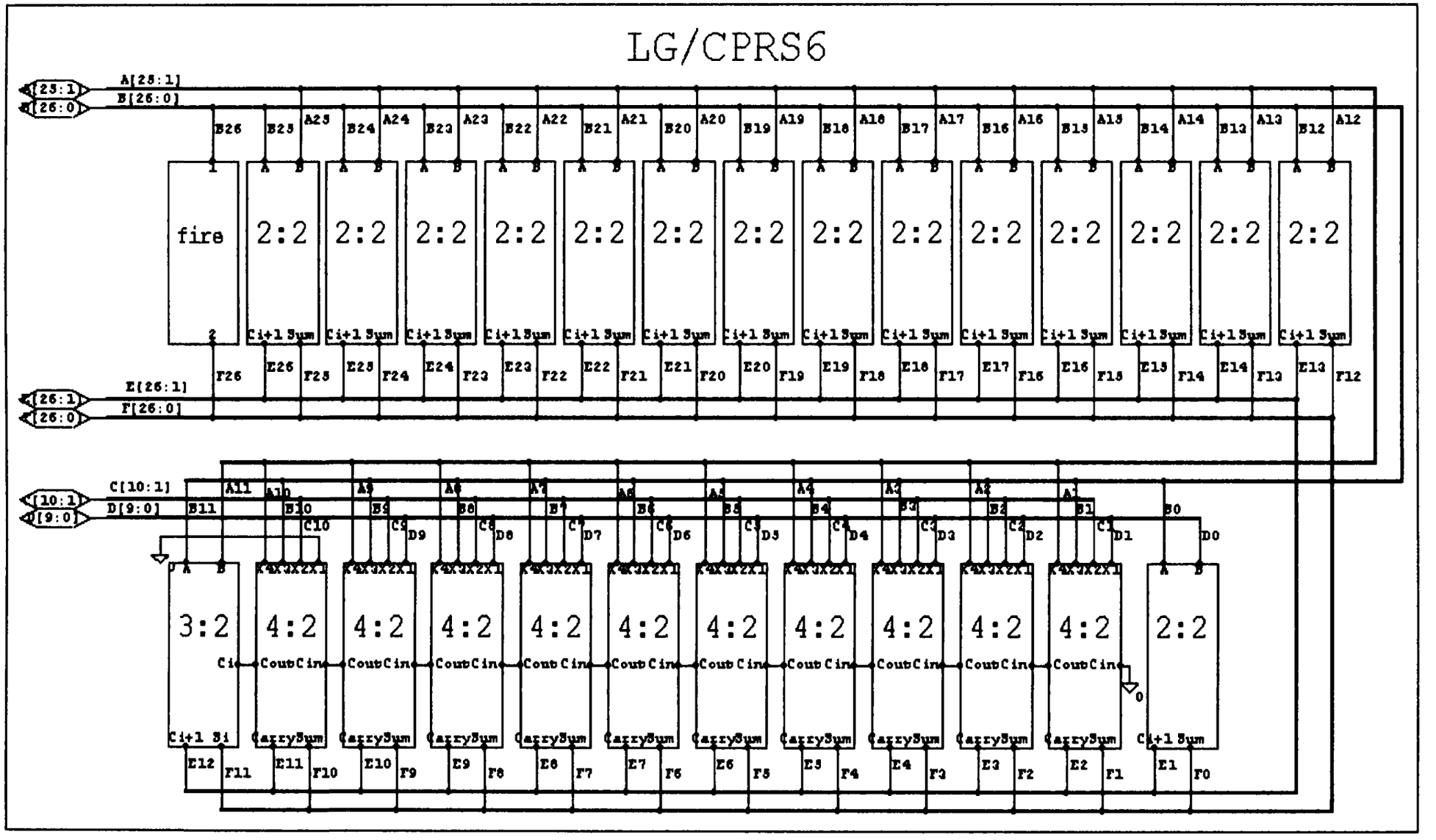


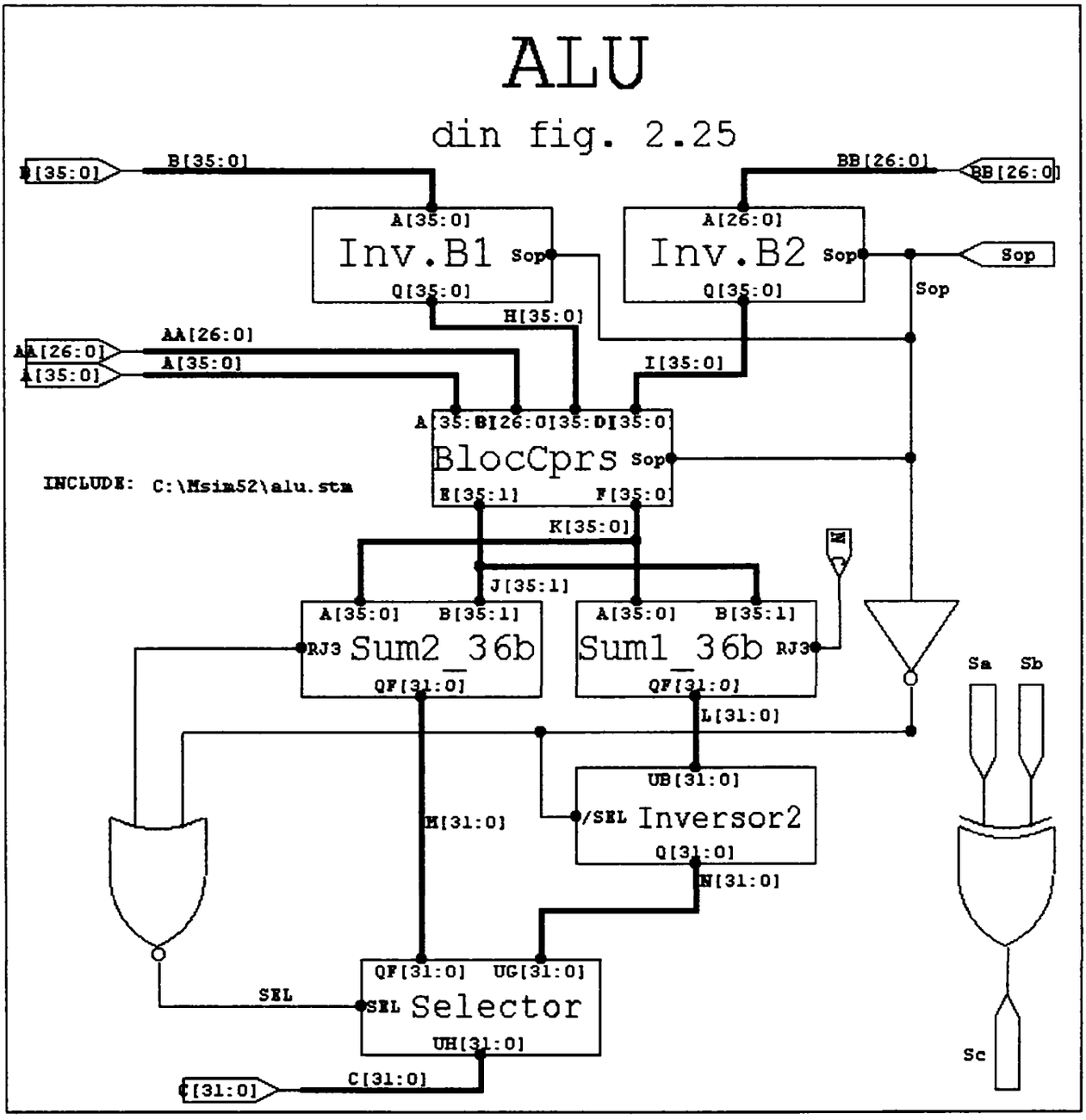
A3.13

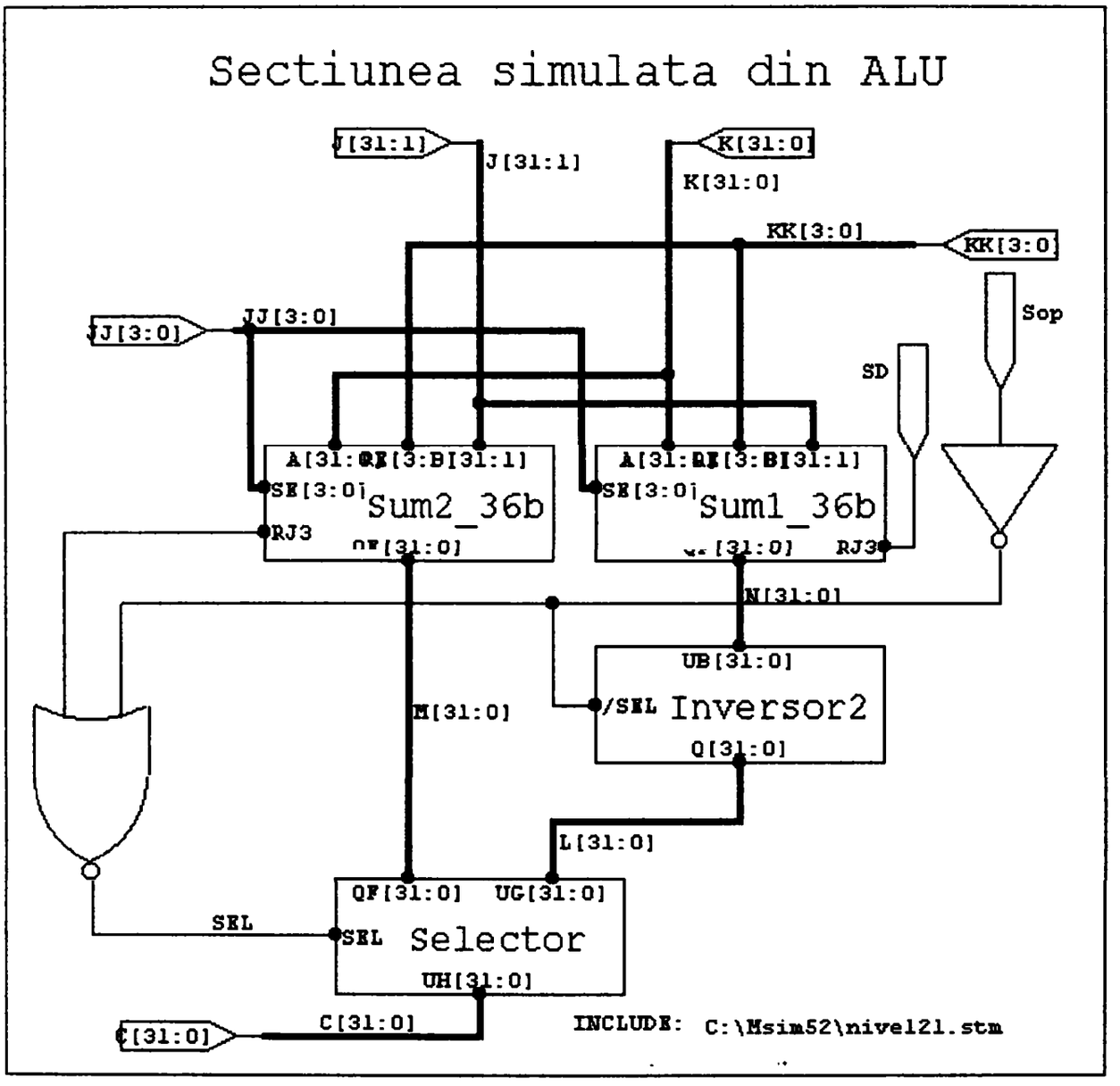

LG/CPRS4
A[9:1]
B[9:0]
C[5:1]
D[5:0]
A9
B9
A8
B8
A7
B7
A6
B6
A5
B5
C5
D5
A4
B4
C4
D4
A3
B3
C3
D3
A2
B2
C2
D2
A1
B1
C1
D1
B0
D0
2:2
2:2
2:2
3:2
4:2
4:2
4:2
4:2
4:2
4:2
Ci
CoutCin
Ci+1 Sum
CarryCin
E10
F9
E9
F8
E8
F7
E7
F6
E6
F5
E5
F4
E4
F3
E3
F2
E2
F1
E1
F0
0
A[10:1]
F[9:0]

ANEXA 3

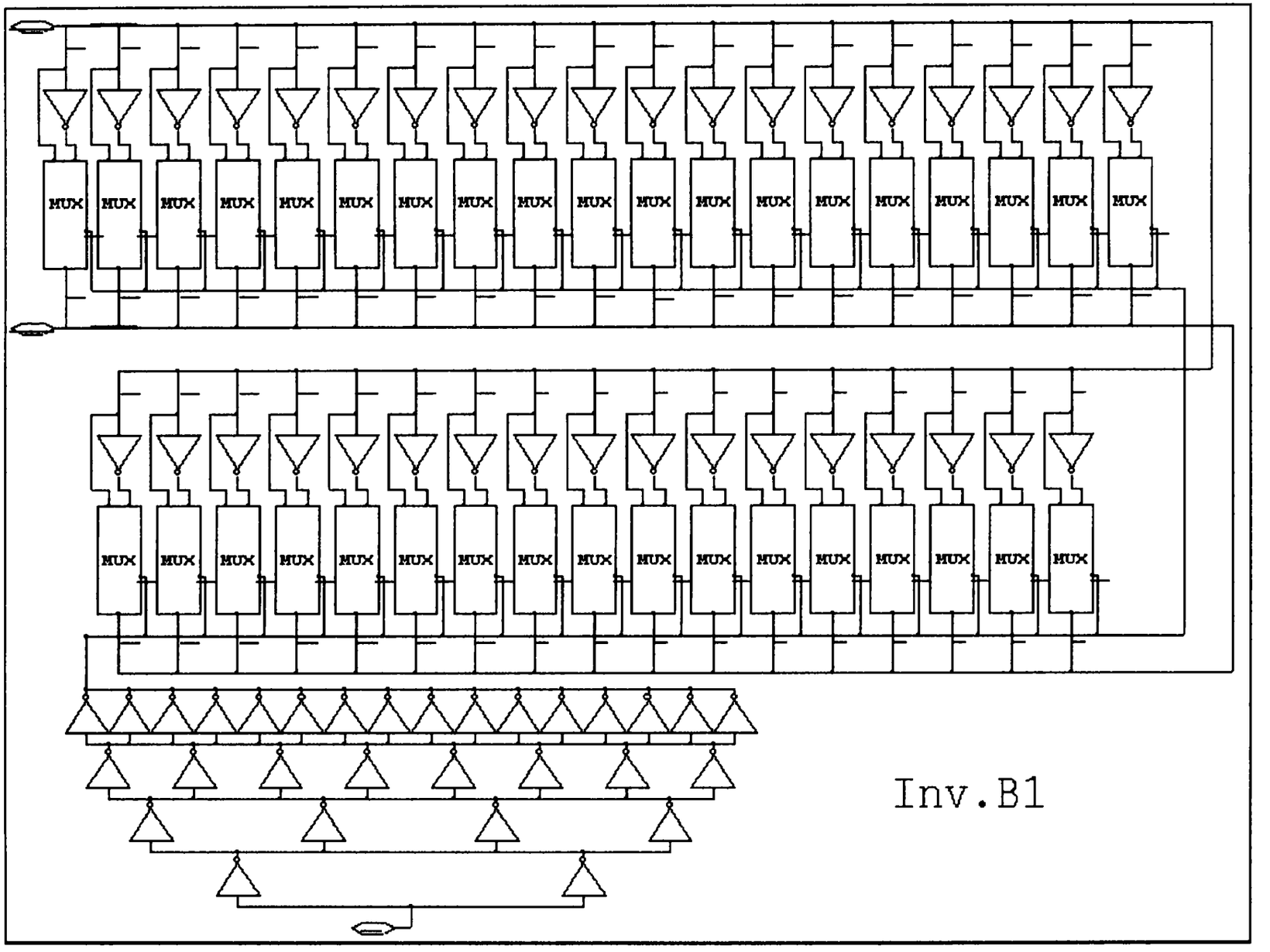

LG/CPRS5
A[21:2]
B[23:0]
fire
2:2
E[25:1]
F[26:0]
C[13:1]
D[13:0]
3:2
4:2
CoutCin
Carry5un
Ci+1 5un


LG/CPRS6
A[25:1]
B[26:0]
E[26:1]
F[26:0]
C[10:1]
D[9:0]
fire
2:2
3:2
4:2
Ci+1 Sum
Ci+1 Si
CoutCin
CarrySum


ALU
din fig. 2.25
B[35:0]
B[35:0]
BB[26:0]
BB[26:0]
A[35:0]
Inv.B1
Sop
Q[35:0]
A[26:0]
Inv.B2
Sop
Q[35:0]
Sop
Sop
H[35:0]
AA[26:0]
AA[26:0]
A[35:0]
A[35:0]
I[35:0]
A[35:0] B[26:0] D[35:0]
BlocCprs
Sop
E[35:1] F[35:0]
INCLUDE: C:\Msim52\alu.stm
K[35:0]
J[35:1]
A[35:0] B[35:1]
RJ3
Sum2_36b
QF[31:0]
A[35:0] B[35:1]
Sum1_36b
RJ3
QF[31:0]
Sa
Sb
L[31:0]
UB[31:0]
/SEL
Inversor2
Q[31:0]
M[31:0]
N[31:0]
QF[31:0] UG[31:0]
SEL
SEL
Selector
UH[31:0]
C[31:0]
C[31:0]
Sc

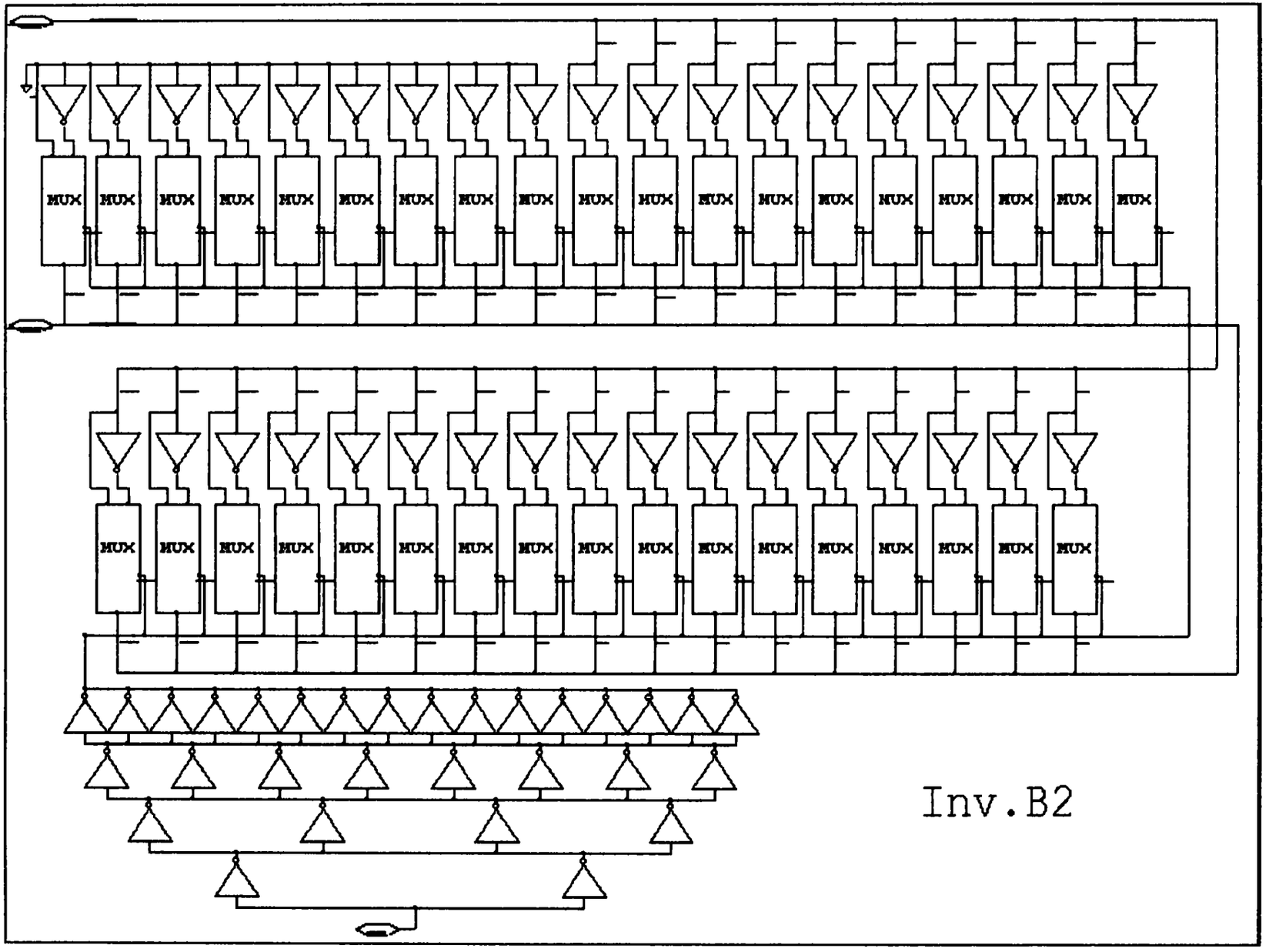

Sectiunea simulata din ALU
J[31:1]
J[31:1]
K[31:0]
K[31:0]
KK[3:0]
KK[3:0]
JJ[3:0]
JJ[3:0]
Sop
SD
A[31:0] B[31:1]
SE[3:0]
Sum2_36b
RJ3
OF[31:0]
A[31:0] B[31:1]
SE[3:0]
Sum1_36b
RJ3
N[31:0]
UB[31:0]
/SEL
Inversor2
Q[31:0]
M[31:0]
L[31:0]
QF[31:0]
UG[31:0]
SEL
SEL
Selector
UH[31:0]
C[31:0]
C[31:0]
INCLUDE: C:\Msim52\nivel21.stm

ANEXA 3

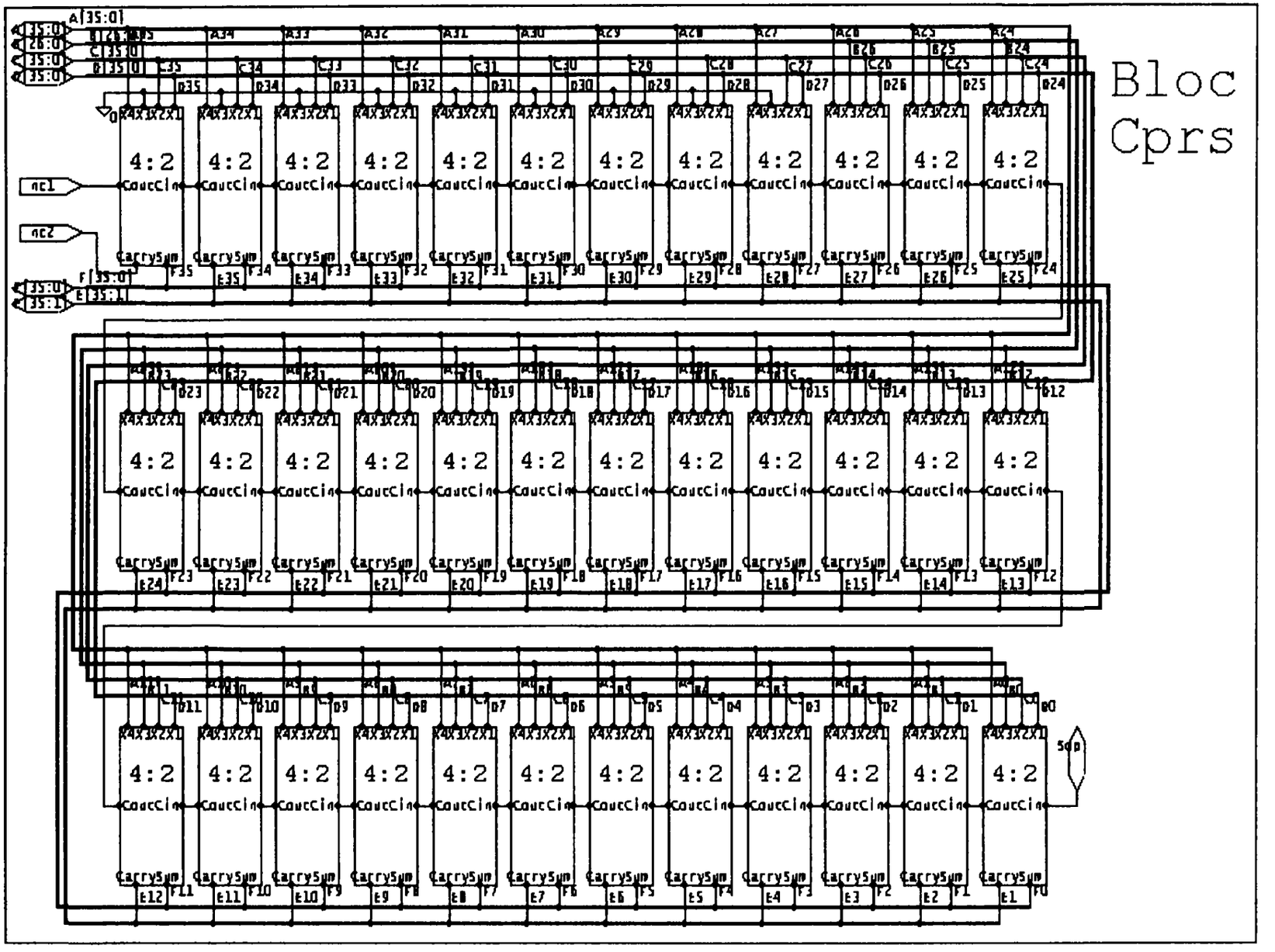


A3.19

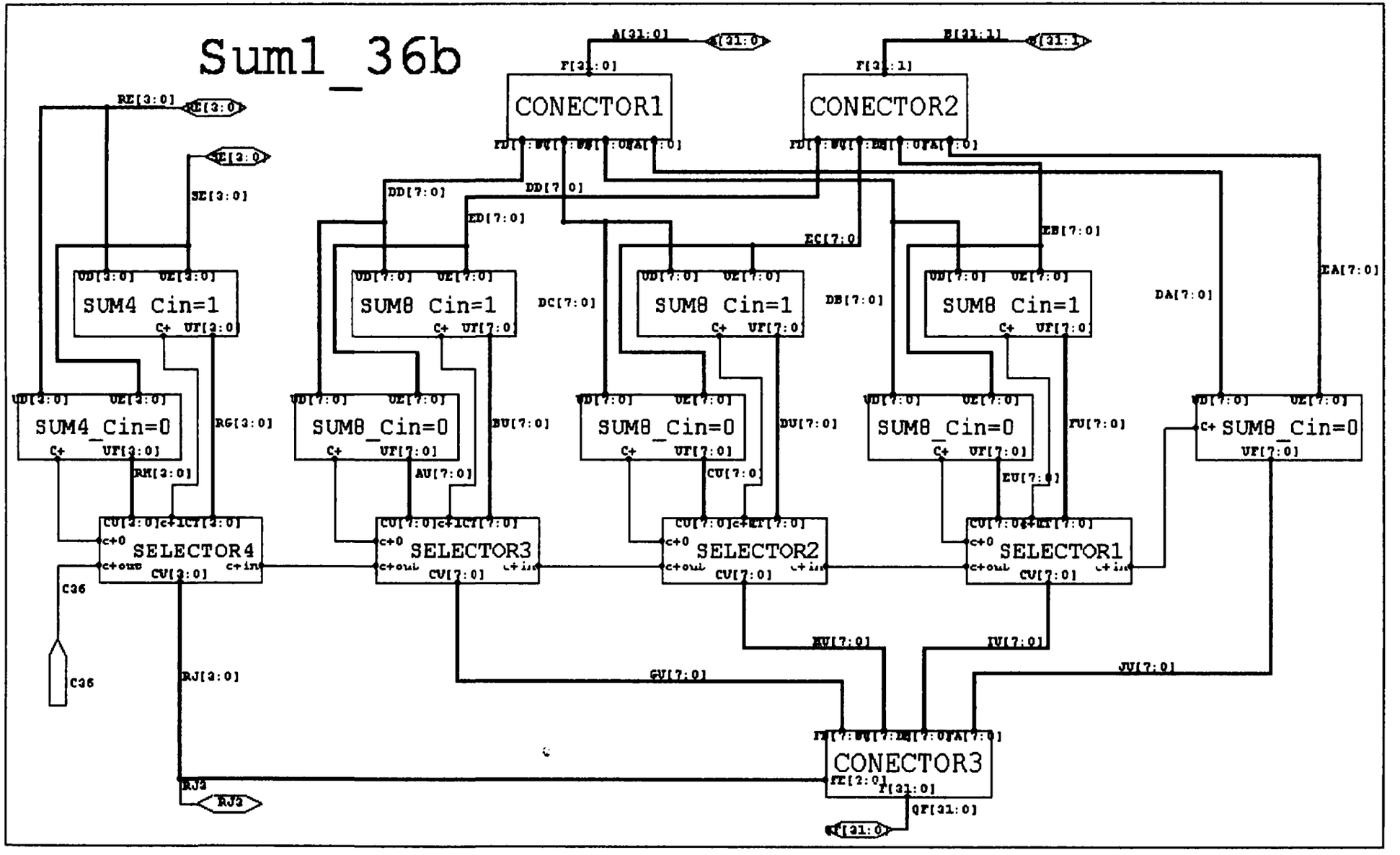
ANEXA 3



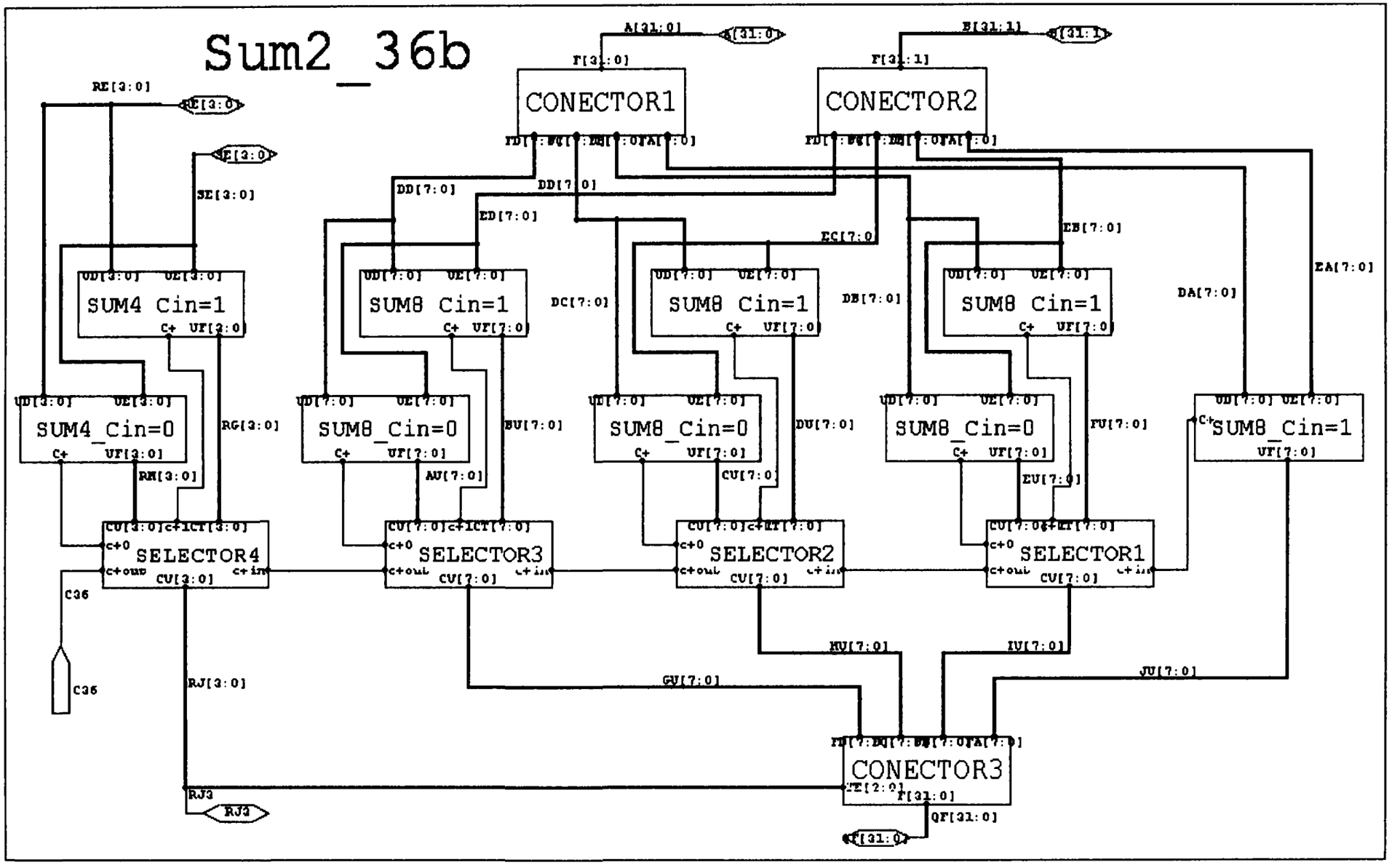
ANEXA 3

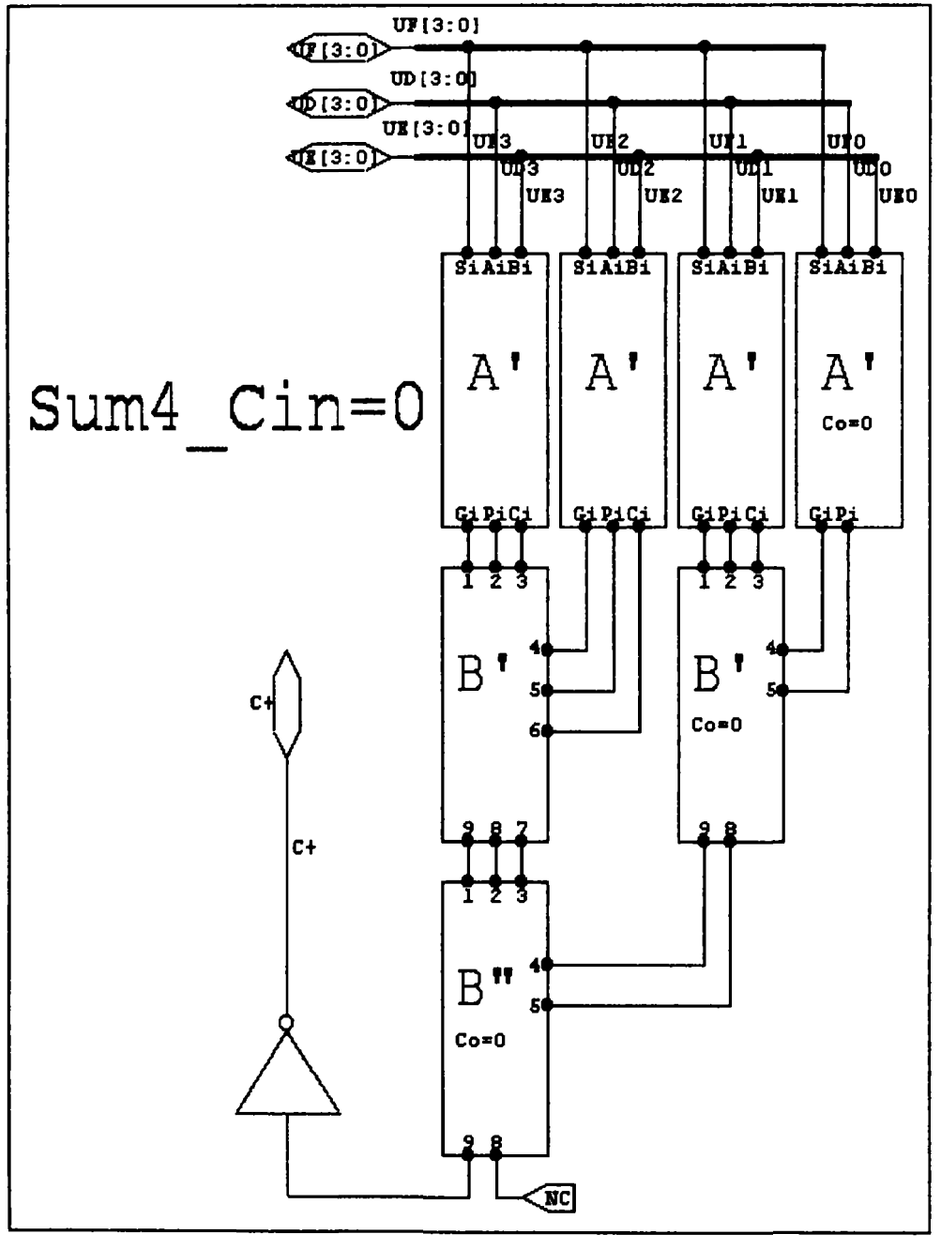


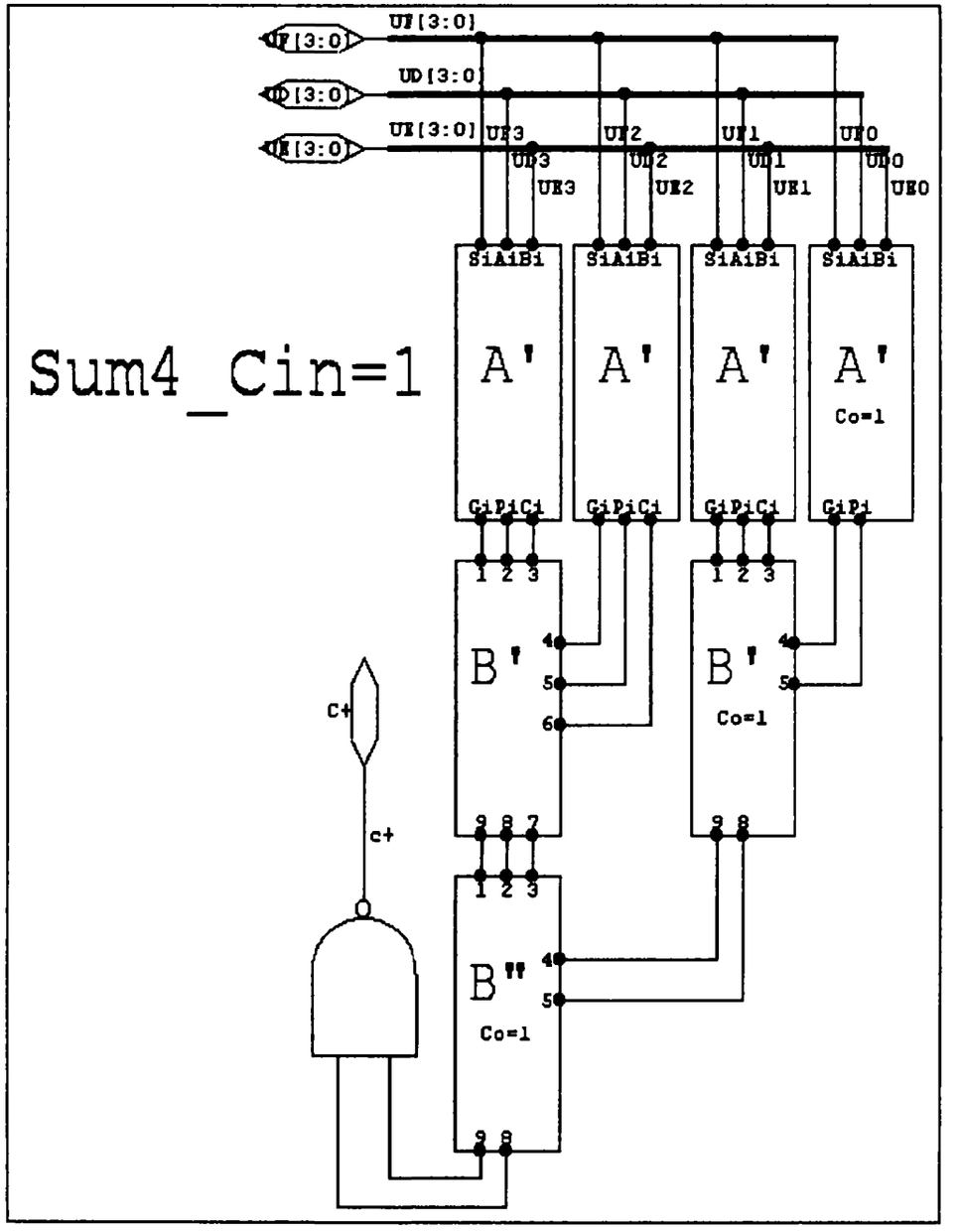
A3.21

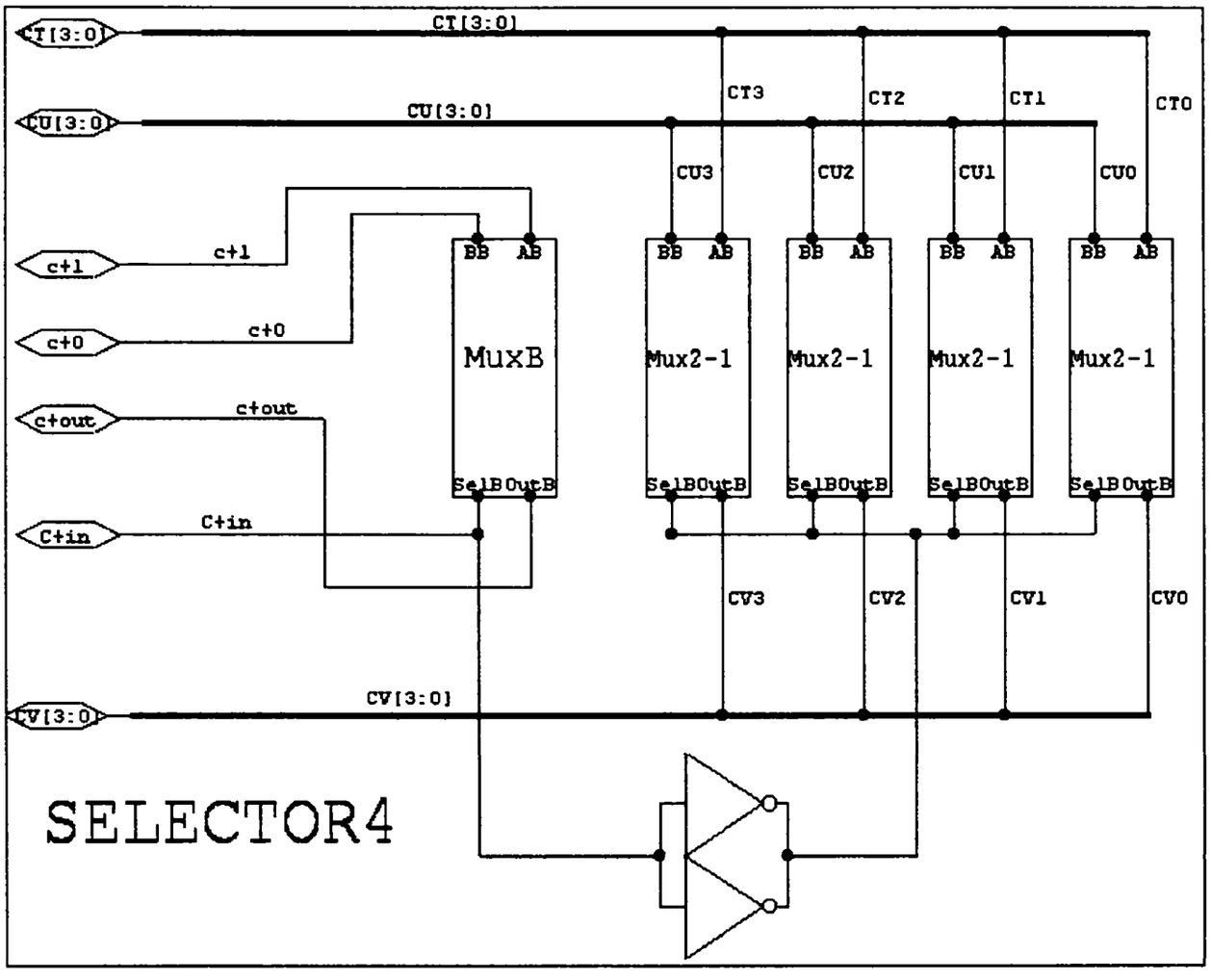

Sum1_36b
A[31:0]
A[21:0]
B[31:1]
B[31:1]
F[21:0]
F[31:1]
CONECTOR1
CONECTOR2
RE[3:0]
SE[3:0]
DD[7:0]
ED[7:0]
EC[7:0]
EB[7:0]
EA[7:0]
SUM4 Cin=1
SUM8 Cin=1
SUM8 Cin=1
SUM8 Cin=1
DC[7:0]
DB[7:0]
DA[7:0]
SUM4_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=0
RG[3:0]
BU[7:0]
DU[7:0]
FU[7:0]
RH[3:0]
AU[7:0]
CU[7:0]
EU[7:0]
SELECTOR4
SELECTOR3
SELECTOR2
SELECTOR1
C36
C36
RJ[3:0]
GV[7:0]
HV[7:0]
IV[7:0]
JV[7:0]
RJ3
RJ3
CONECTOR3
FF[2:0]
F[31:0]
QF[21:0]

ANEXA 3


Sum2_36b
A[31:0]
B[31:1]
CONECTOR1
CONECTOR2
CONECTOR3
SUM4 Cin=1
SUM8 Cin=1
SUM8 Cin=1
SUM8 Cin=1
SUM4_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=0
SUM8_Cin=1
SELECTOR4
SELECTOR3
SELECTOR2
SELECTOR1
RE[3:0]
SE[3:0]
DD[7:0]
ED[7:0]
EC[7:0]
EB[7:0]
EA[7:0]
DA[7:0]
DB[7:0]
DC[7:0]
RG[3:0]
RH[3:0]
BU[7:0]
AU[7:0]
DU[7:0]
CU[7:0]
FU[7:0]
EU[7:0]
C36
RJ[3:0]
RJ3
HU[7:0]
IU[7:0]
GU[7:0]
JU[7:0]
QF[31:0]


Sum4_Cin=0
UF[3:0]
UD[3:0]
UE[3:0]
SiAiBi
A'
B'
B''
GiPiCi
GiPi
Co=0
C+
NC


Sum4_Cin=1
UF[3:0]
UD[3:0]
UE[3:0]
SiAiBi
A'
B'
B''
GiPiCi
GiPi
Co=1
C+

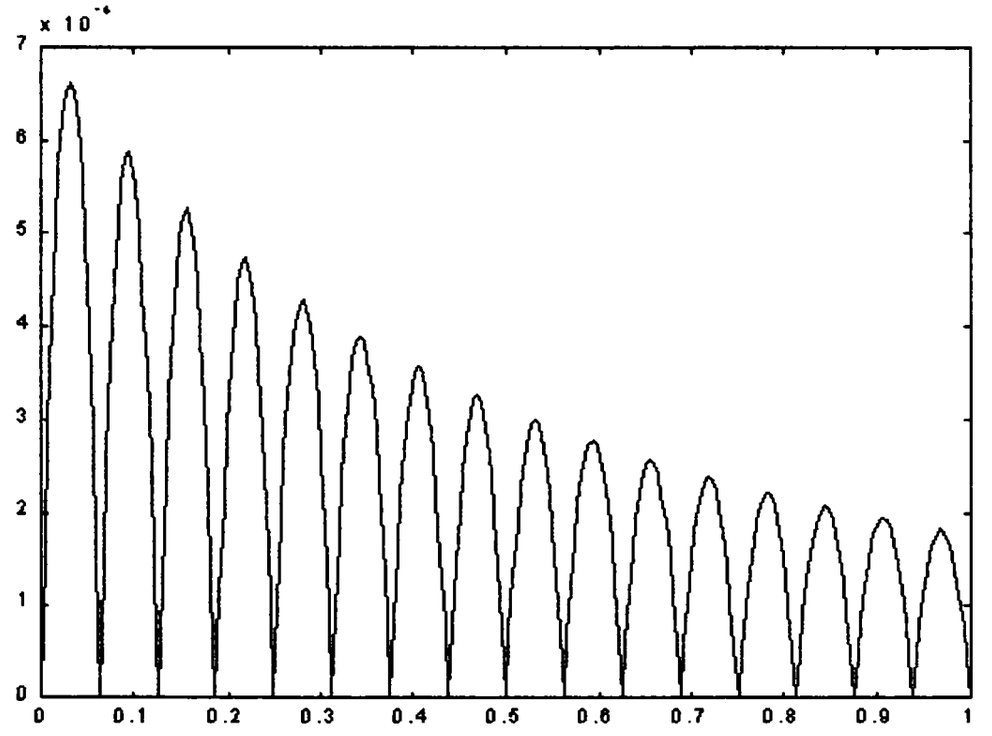


Următoarele blocuri sunt identice cu cele prezentate în Anexa 1:

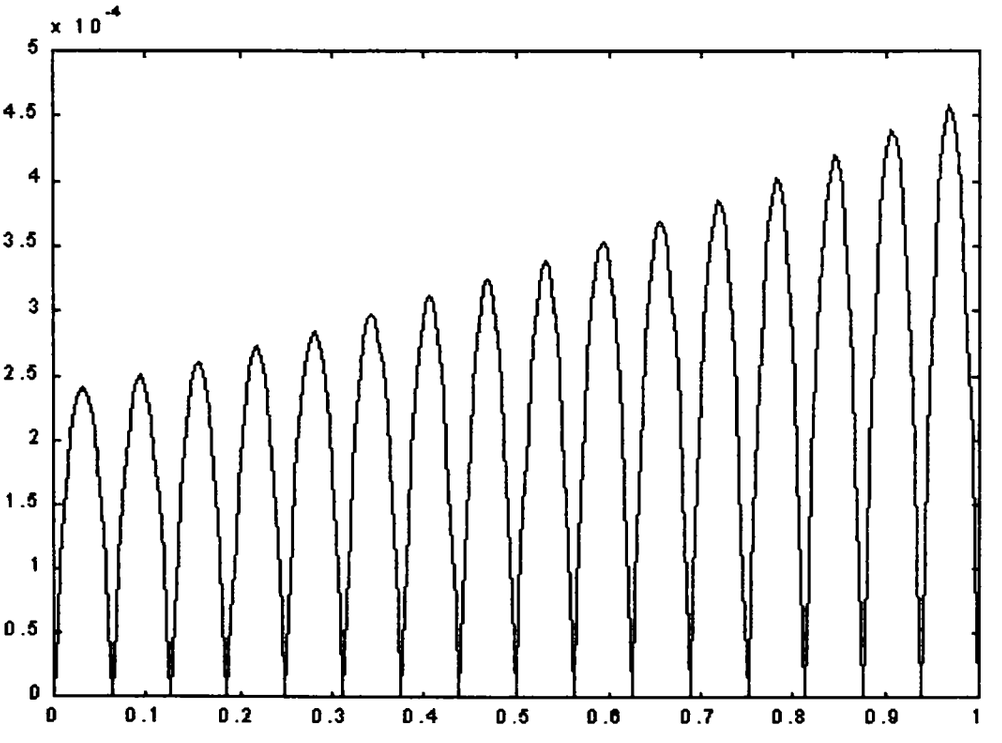
Inversor2;
Selector;
SELECTOR1,2,3;
MUX;
MuxB;
Mux2-1;
SUM8_Cin=0;
SUM8_Cin=1;
A', B', B";
A' Co=0, A' Co=1;
B' Co=0, B' Co=1;
B" Co=0, B" Co=1.

ANEXA 4

Eroarea de conversie la logaritmare, obţinută în cazul simplificat când interpolarea liniară se face între 16 valori memorate ale funcţiei $f(y)=\log_2(1+y)-y$, pentru 256 de puncte intermediare:



Eroarea de conversie la antilogaritmare, obţinută în cazul simplificat când interpolarea liniară se face între 16 valori memorate ale funcţiei $f(y)=(1+y)-2^y$, pentru 256 de puncte intermediare:



ANEXA 4

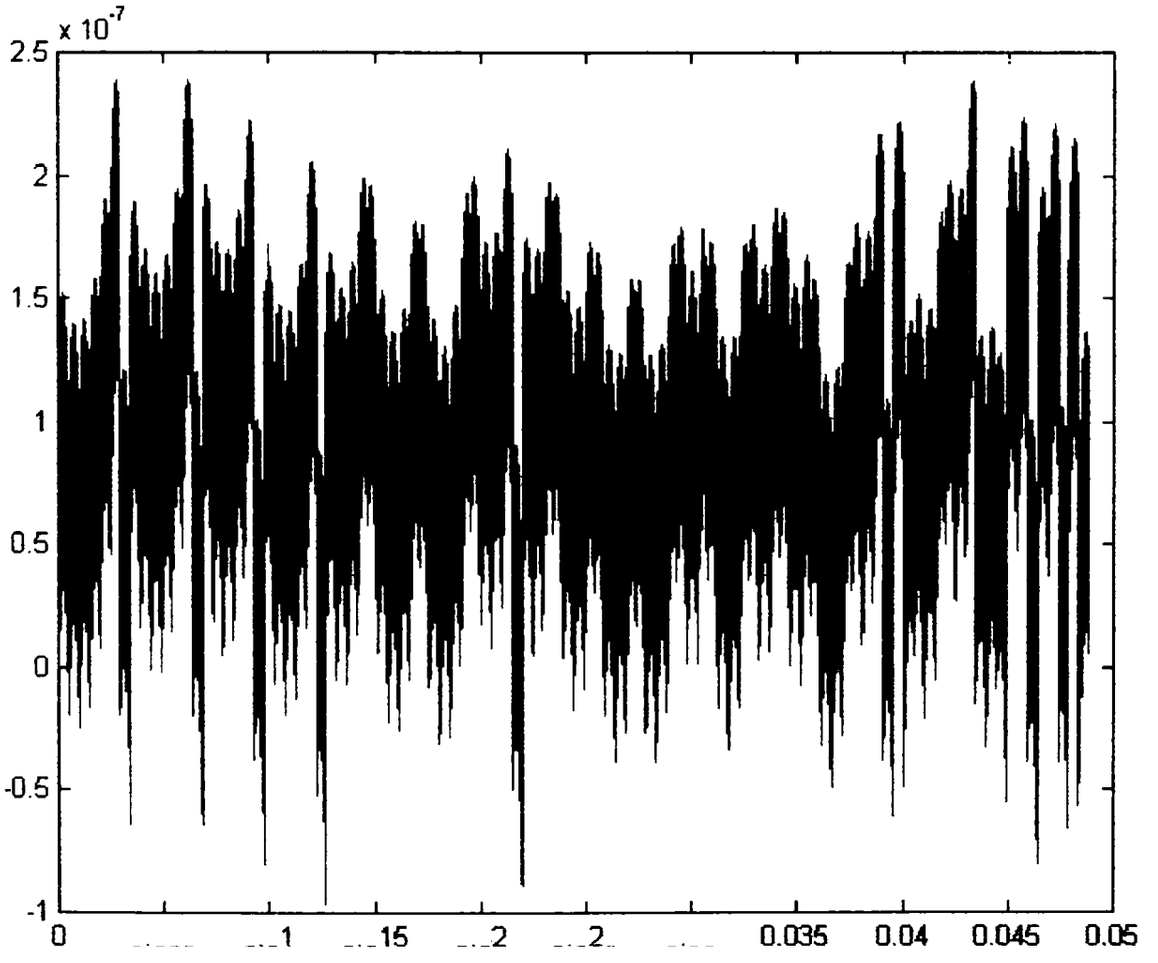
Secvenţele Matlab utilizate în cazurile anterioare:

logaritmare	antilogaritmare

y=0:16:256;
L=log2(1+y/256)-y/256;
yy=0:1:256;
T(:,1)=y';
T(:,2)=L';
z=table1(T,yy);
LL=log2(1+yy/256)-yy/256;
er=LL-z';
yyy=yy/256;
plot(yyy,er)
	y=0:16:256;
	AL=1+y/256-2 .^(y/256);
	yy=0:1:256;
	T(:,1)=y';
	T(:,2)=AL';
	z=table1(T,yy);
	ALL=1+yy/256-2 .^(yy/256);
	er=ALL-z';
	yyy=yy/256;
	plot(yyy,er)

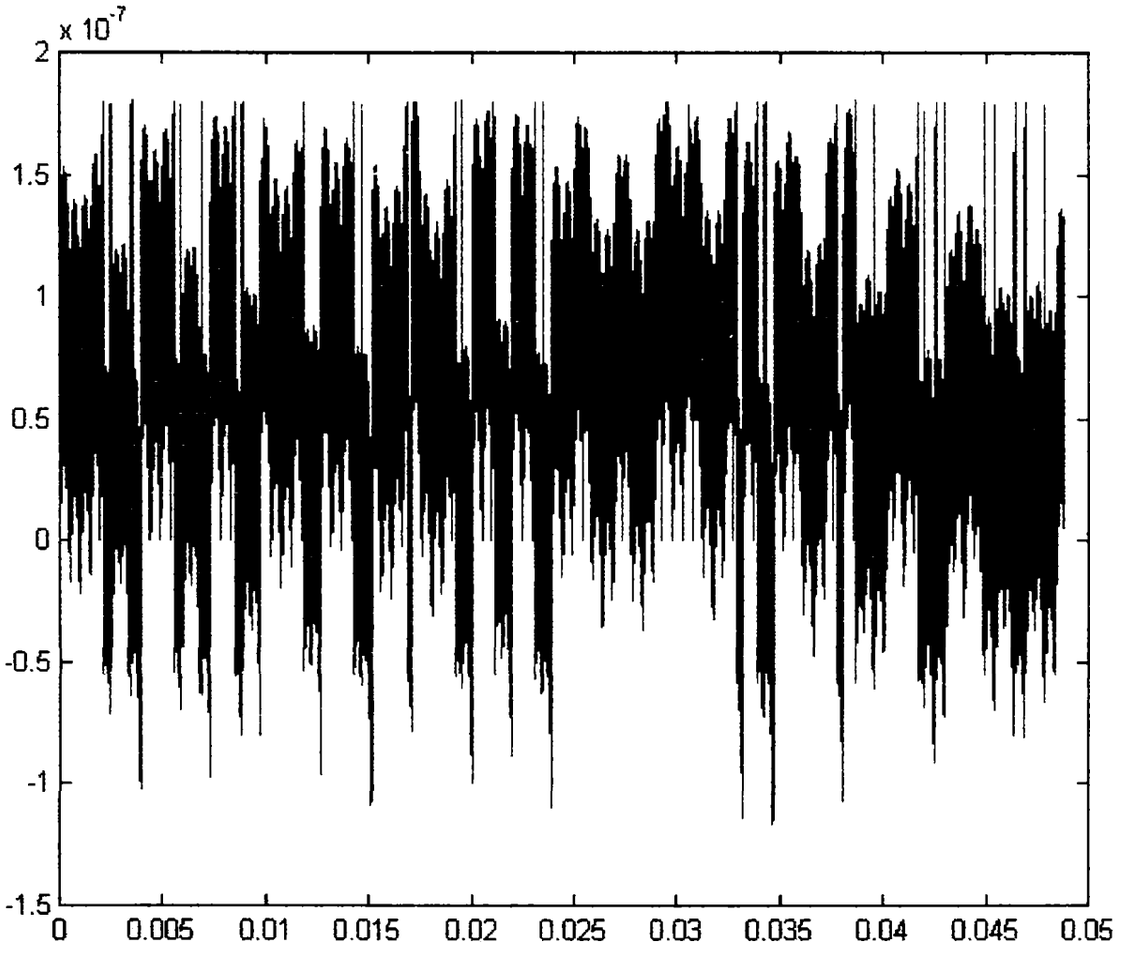
Simulările au fost făcute pentru cazul ideal, când reprezentarea binară se face pe un număr infinit de biţi, pentru a depista intervalul pe care se obţine eroare maximă prin algoritmul de interpolare.

În continuare se prezintă distribuţia erorilor de la logaritmare pentru primele 409600 valori calculate prin algoritmul de interpolare propus de F. Lai. Calculul s-a efectuat pe baza primelor 100 de valori (din totalul celor 2048) memorate în memoriile ROMA şi ROMA' cu rezoluţie de 23 de biţi.



ANEXA 4

Următoarea diagramă prezintă distribuţia erorilor de la logaritmare pentru primele 409600 valori calculate prin algoritmul de interpolare îmbunătăţit, propus de autorul tezei. Calculul s-a efectuat pe baza primelor 100 de valori (din totalul celor 2048) memorate în memoriile ROMA şi ROMA' cu rezoluţie de 23 de biţi şi care includ în anumite locaţii de memorie valori de corecţie pentru limitarea erorii maxime la $1{,}8\times10^{-7}$.



Programul matlab pentru generarea valorilor E_y (ce includ corecţii pe anumite intervale, pentru a limita eroarea de conversie la $1{,}8\times10^{-7}$) şi ΔE_y ce se memorează în primele 100 de locaţii ale memoriilor ROMA şi ROMA' este prezentat mai jos:

```
y=0:99;
L=log2(1+y/2048)-y/2048;
La=round(L*2^23)/2^23;
x=1:100;
LD=log2(1+x/2048)-x/2048;
D=LD-L;
Da=round(D*2^23)/2^23;
Lteor(1,:)=La;
```

```
er(1,1:100)=0;
z=0:409599;
Lz=log2(1+z/8388608)-z/8388608;
Lz=reshape(Lz,4096,100);
for i=1:100,
for j=1:4095,
Lteor(1+j,i)=La(i)+(j/4096)*Da(i);
Lteora(1+j,i)=fix(Lteor(1+j,i)*2^23)/2^23;
er(1+j,i)=Lz(1+j,i)-Lteora(1+j,i);
if er(1+j,i)>1.8*10^-7
La(i)=La(i)+1.2*10^-7;
Lteor(1+j,i)=La(i)+(j/4096)*Da(i);
Lteora(1+j,i)=fix(Lteor(1+j,i)*2^23)/2^23;
er(1+j,i)=Lz(1+j,i)-Lteora(1+j,i);
end
end
end
er=reshape(er,1,409600);
z=z/8388608;
plot(z,er)
La=round(La*2^23);
Lmem=dec2hex(La)
Da=Da*2^23;
Dmem=dec2hex(Da)
```

Adresa	E_y		Adresa	E_y		Adresa	E_y
00	00000		10	06FE5		20	0DCF3
01	00714		11	076CC		21	0E3AC
02	00E25		12	07DAF		22	0EA63
03	01533		13	0848F		23	0F115
04	01C3F		14	08B6C		24	0F7C6
05	02347		15	09247		25	0FE74
06	02A4C		16	0991F		26	1051F
07	0314F		17	09FF4		27	10BC8
08	0384E		18	0A6C7		28	1126D
09	03F4B		19	0AD96		29	1190F
0A	04645		1A	0B462		2A	11FAF
0B	04D3D		1B	0BB2C		2B	1264D
0C	05431		1C	0C1F3		2C	12CE7
0D	05B22		1D	0C8B8		2D	1337E
0E	06211		1E	0CF79		2E	13A13
0F	068FC		1F	0D637		2F	140A6

ANEXA 4

Adresa	E_y		Adresa	E_y		Adresa	E_y
30	14735		40	1AEB5		50	2137F
31	14DC1		41	1B516		51	219B5
32	1544B		42	1BB74		52	21FE8
33	15AD2		43	1C1D0		53	22618
34	16156		44	1C828		54	22C46
35	167D8		45	1CE7F		55	23272
36	16E57		46	1D4D2		56	2389A
37	174D3		47	1DB22		57	23EC0
38	17B4C		48	1E170		58	244E3
39	181C3		49	1E7BB		59	24B03
3A	18837		4A	1EE04		5A	25121
3B	18EA8		4B	1F44A		5B	2573C
3C	19516		4C	1FA8D		5C	25D55
3D	19B82		4D	200CE		5D	2636B
3E	1A1EB		4E	2070B		5E	2697E
3F	1A851		4F	20D47		5F	26F8F
						60	2759D
						61	27BA8
						62	281B1
						63	287B7

Adresa	ΔE_y		Adresa	ΔE_y		Adresa	ΔE_y
00	714		10	6E6		20	6B9
01	711		11	6E3		21	6B6
02	70E		12	6E0		22	6B3
03	70B		13	6DE		23	6B1
04	708		14	6DB		24	6AE
05	705		15	6D8		25	6AB
06	703		16	6D5		26	6A8
07	700		17	6D2		27	6A5
08	6FD		18	6CF		28	6A3
09	6FA		19	6CD		29	6A0
0A	6F7		1A	6CA		2A	69D
0B	6F4		1B	6C7		2B	69A
0C	6F1		1C	6C4		2C	698
0D	6EF		1D	6C1		2D	695
0E	6EC		1E	6BF		2E	692
0F	6E9		1F	6BC		2F	68F

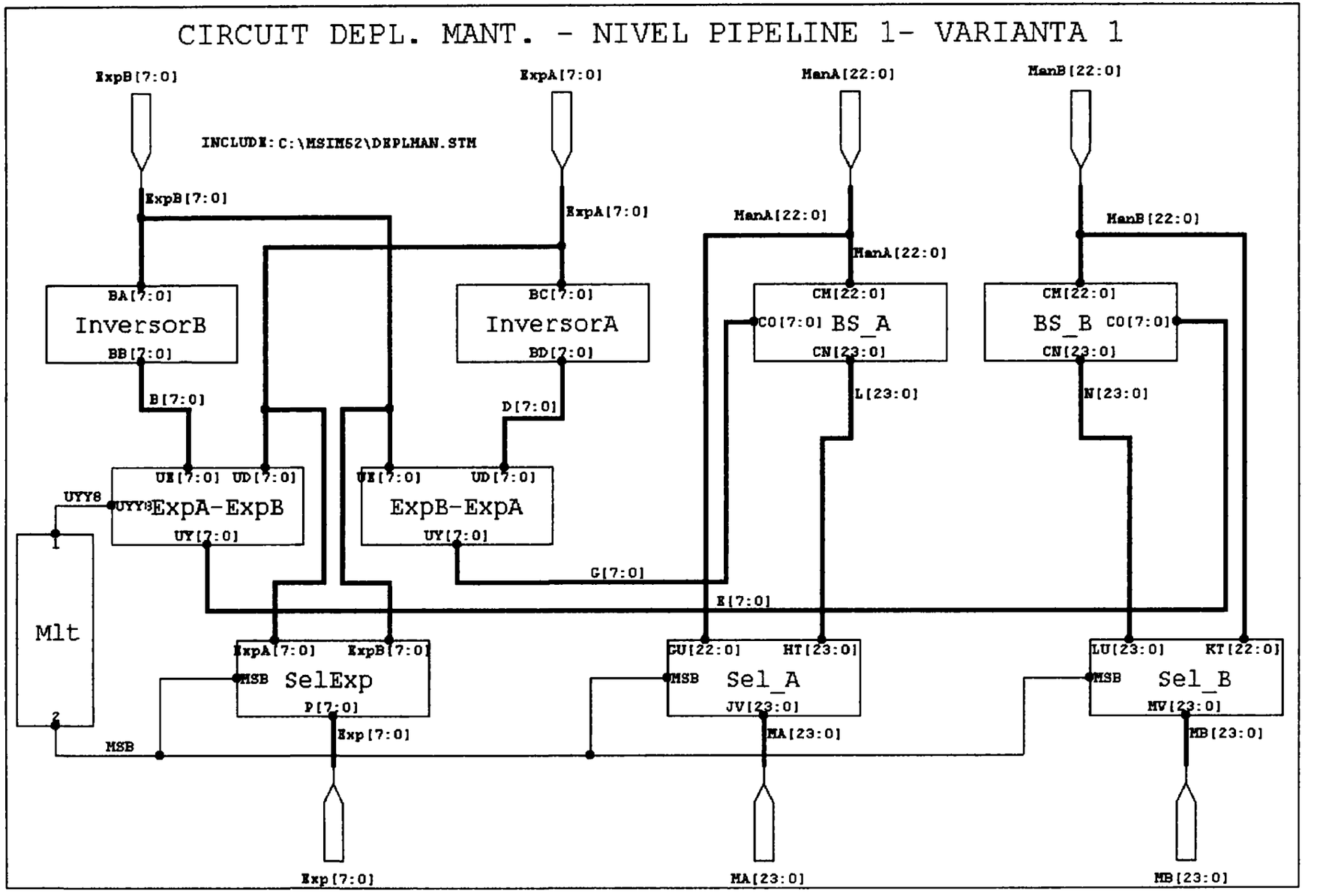
ANEXA 4

Adresa	ΔE_y		Adresa	ΔE_y		Adresa	ΔE_y
30	68D		40	661		50	636
31	68A		41	65E		51	633
32	687		42	65B		52	630
33	684		43	659		53	62E
34	682		44	656		54	62B
35	67F		45	653		55	628
36	67C		46	651		56	626
37	679		47	64E		57	623
38	677		48	64B		58	620
39	674		49	649		59	61E
3A	671		4A	646		5A	61B
3B	66E		4B	643		5B	619
3C	66C		4C	640		5C	616
3D	669		4D	63E		5D	613
3E	666		4E	63B		5E	611
3F	664		4F	638		5F	60E
						60	60B
						61	609
						62	606
						63	603

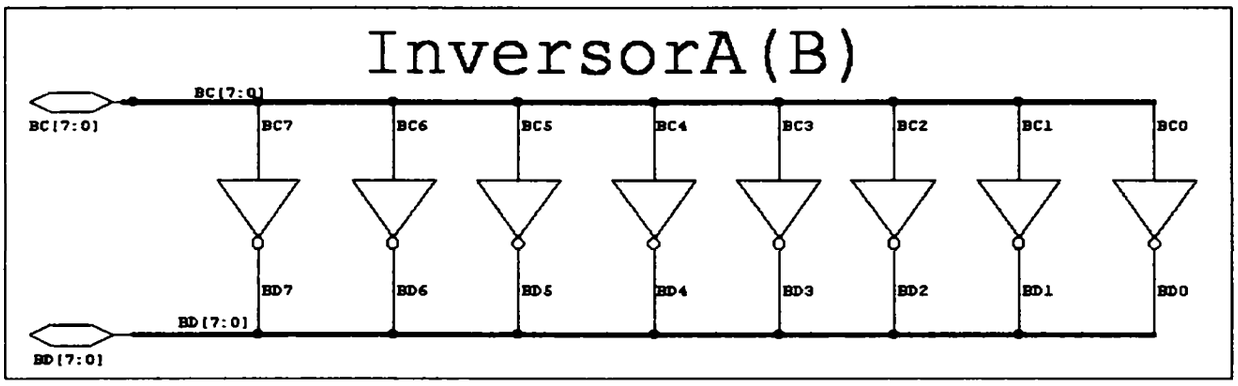
Secvenţa matlab pentru obţinerea diagramei de la pagina A4.2:

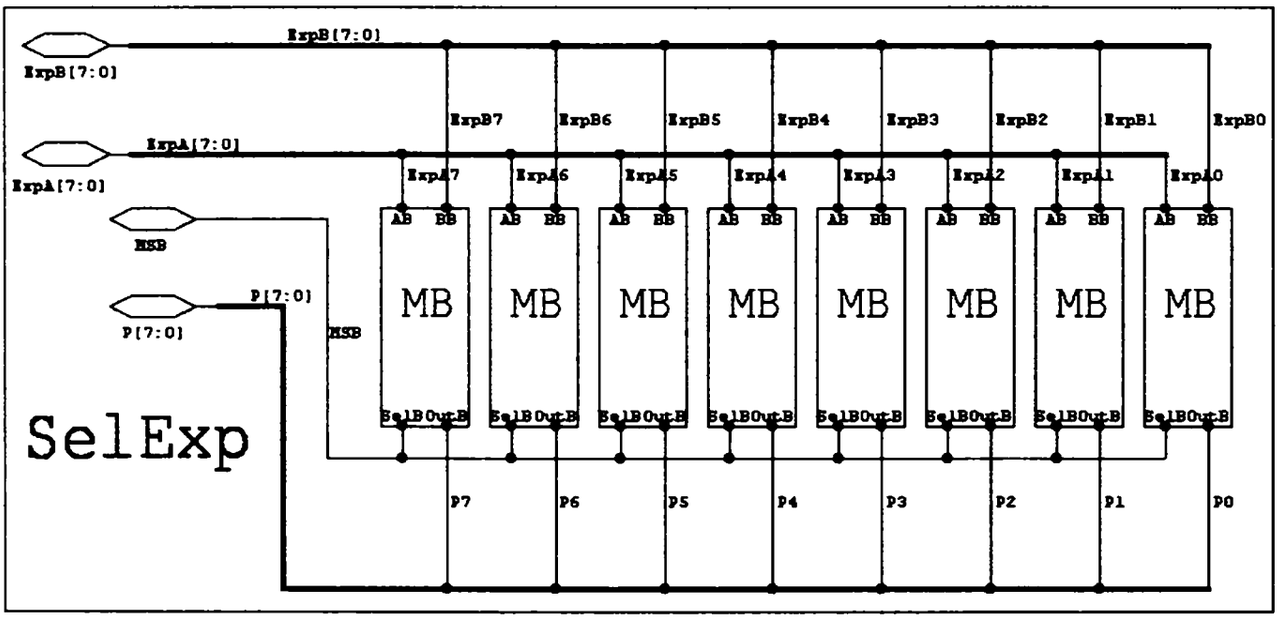
```
y=0:99;
L=log2(1+y/2048)-y/2048;
La=round(L*2^23)/2^23;
x=1:100;
LD=log2(1+x/2048)-x/2048;
D=LD-L;
Da=round(D*2^23)/2^23;
Lteor(1,:)=La;
for i=1:100,
for j=1:4095,
Lteor(1+j,i)=La(i)+(j/4096)*Da(i);
end
end
Lteora=fix(Lteor*2^23)/2^23;
Lalgoritm=reshape(Lteora,1,409600);
z=0:409599;
Lz=log2(1+z/8388608)-z/8388608;
er=Lz-Lalgoritm;
z=z/8388608;
plot(z,er)
```

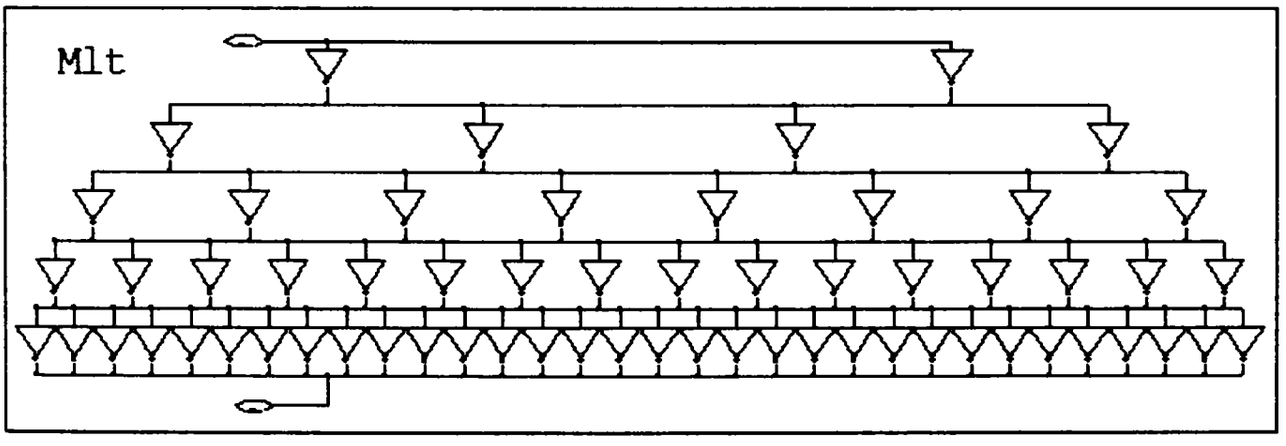
ANEXA 5

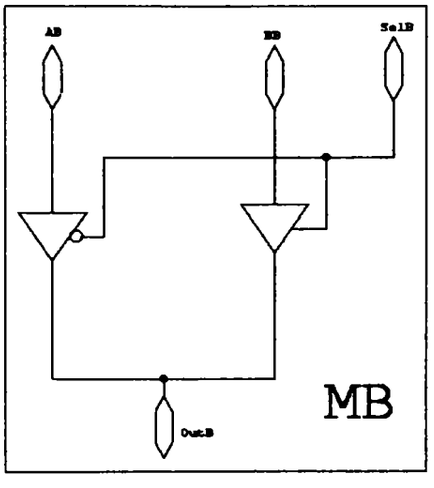


A5. 1

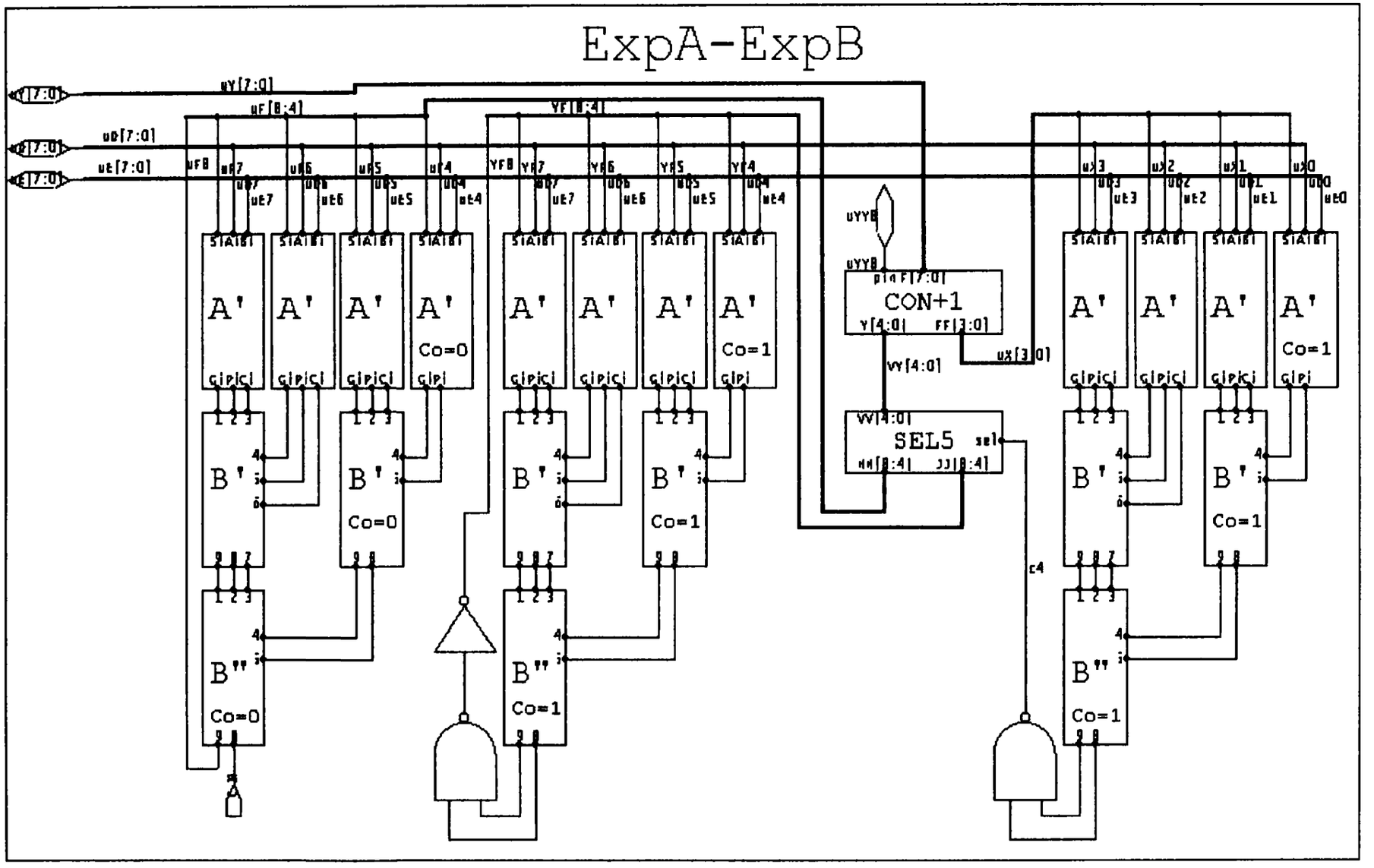

InversorA(B)
BC[7:0]
BC7
BC6
BC5
BC4
BC3
BC2
BC1
BC0
BD7
BD6
BD5
BD4
BD3
BD2
BD1
BD0
BD[7:0]

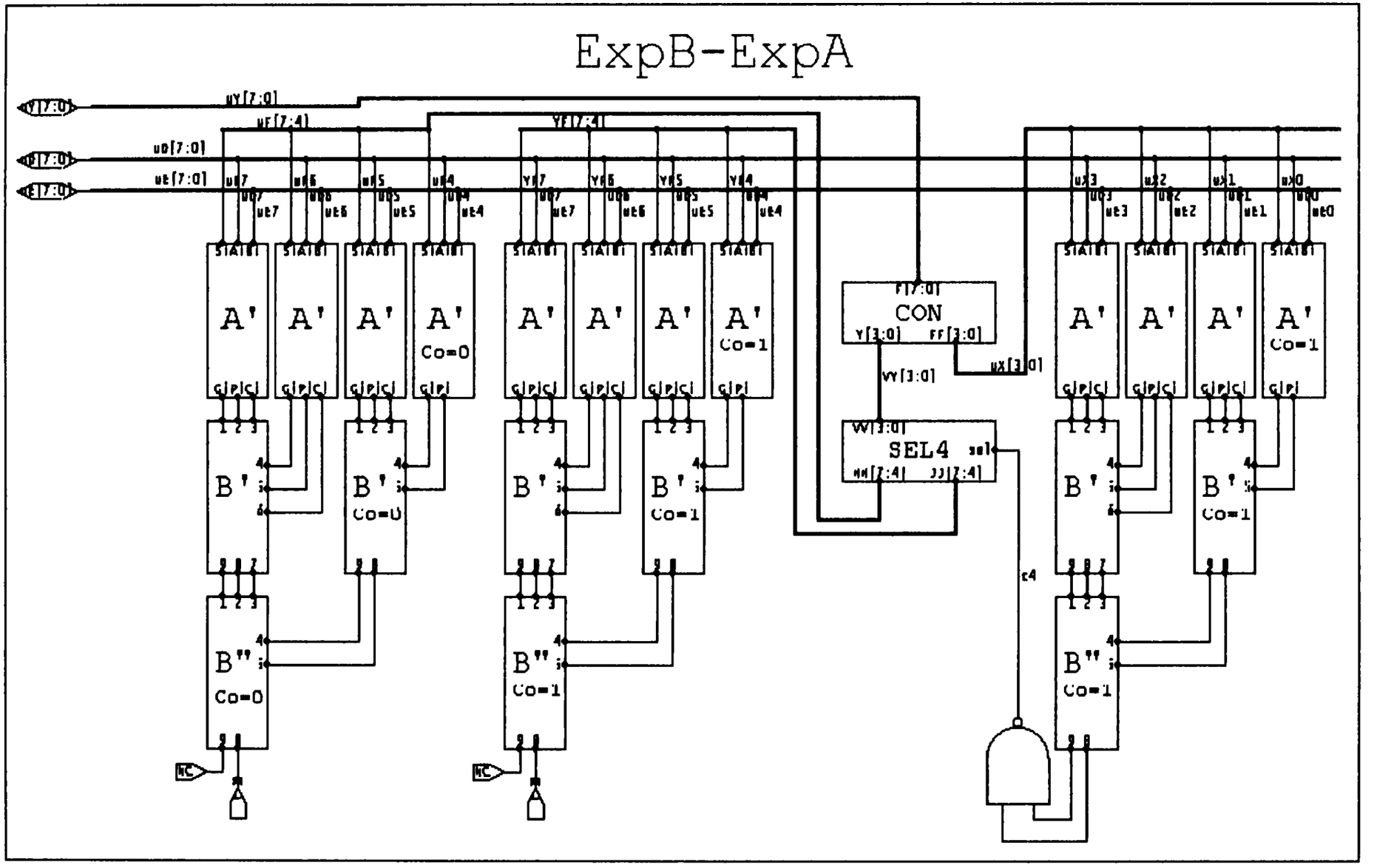

ExpB[7:0]
ExpA[7:0]
MSB
P[7:0]
MB
SelExp
P7
P6
P5
P4
P3
P2
P1
P0


Mlt

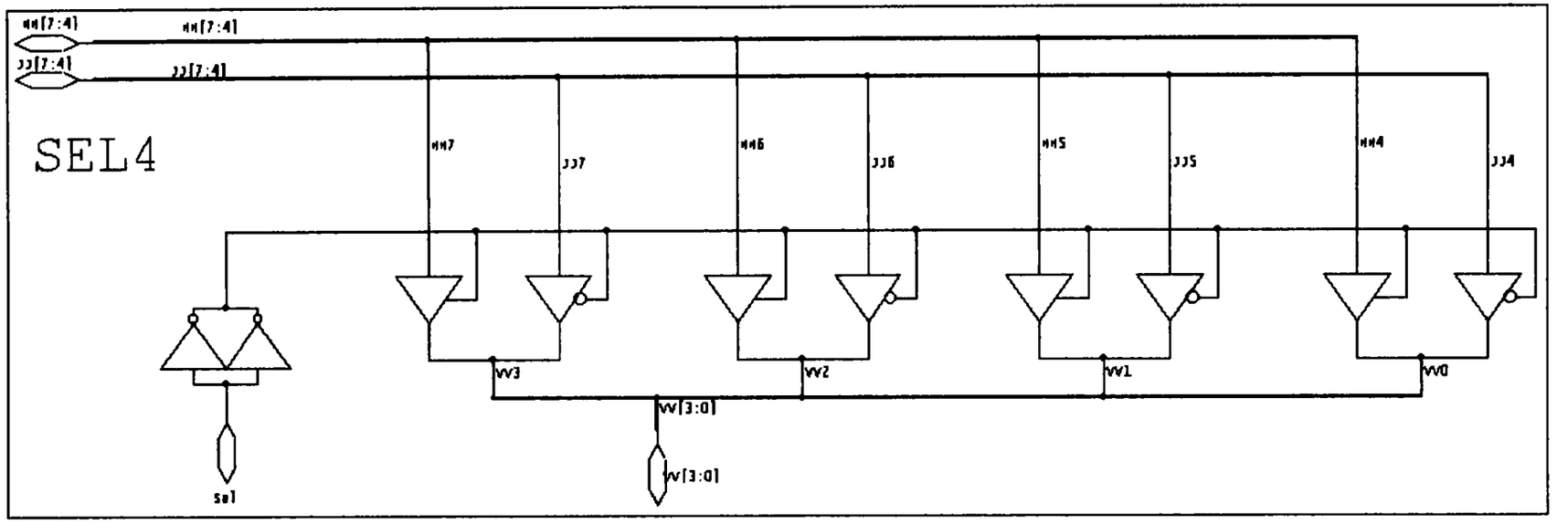

AB
BB
SelB
OutB
MB

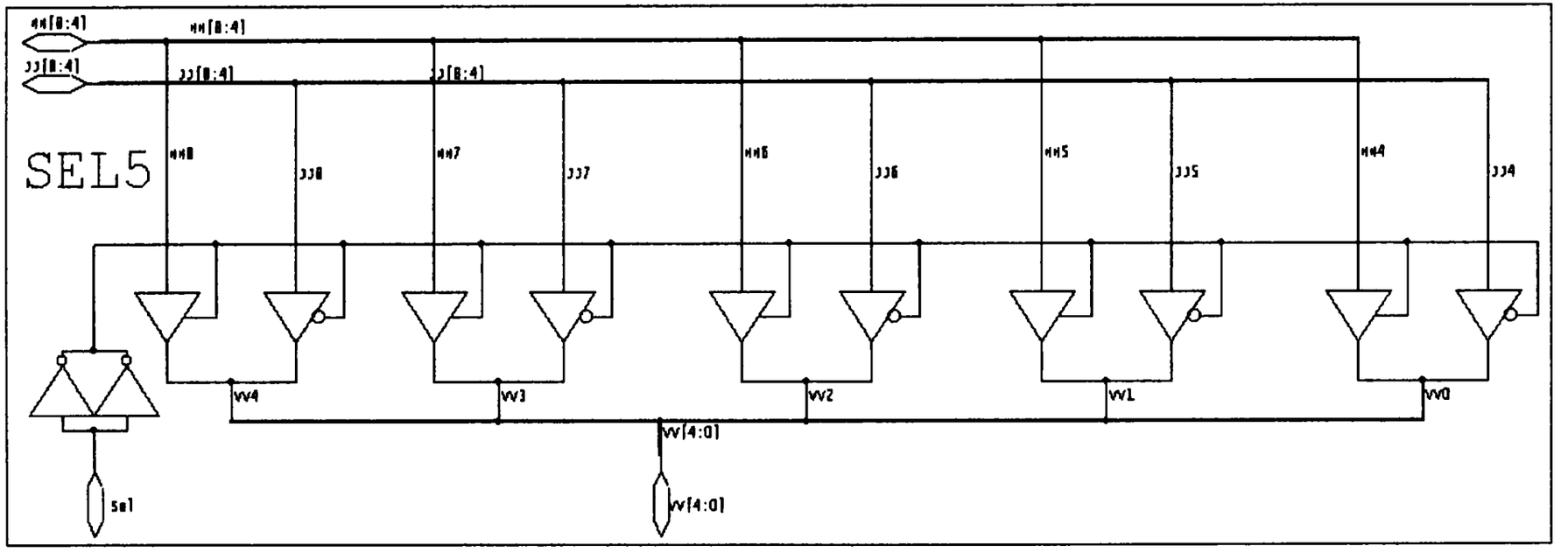
ANEXA 5



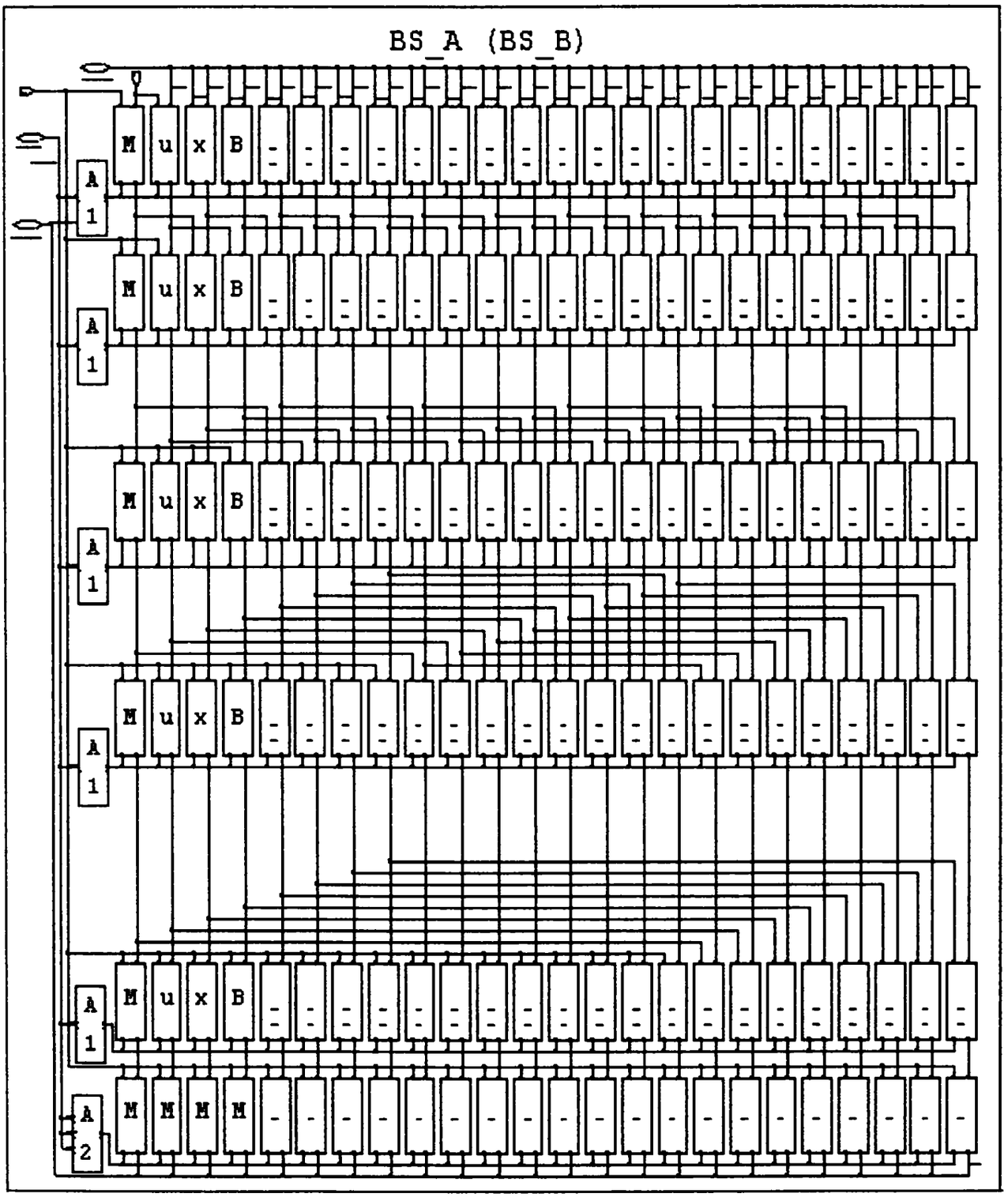


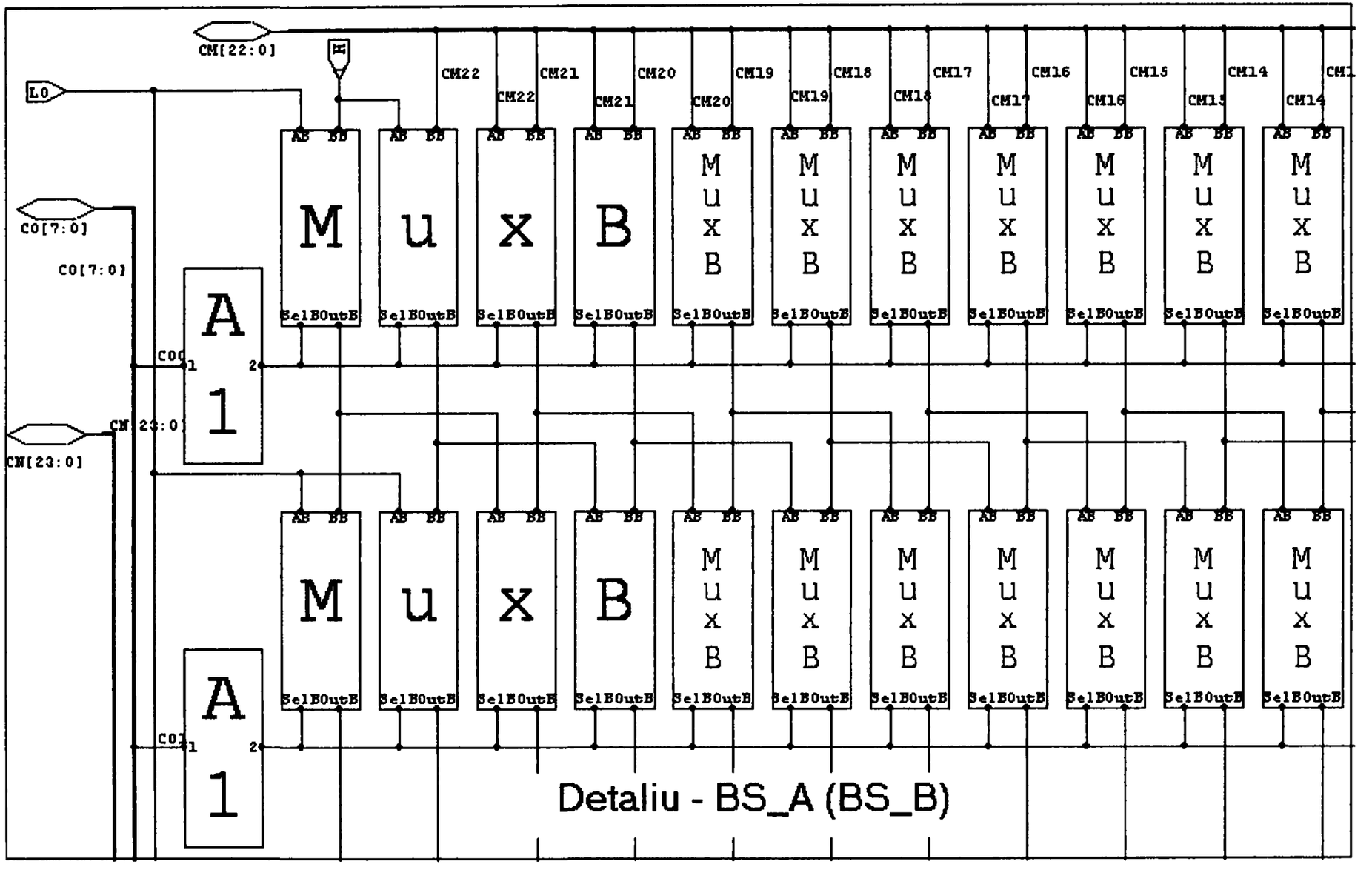
ANEXA 5

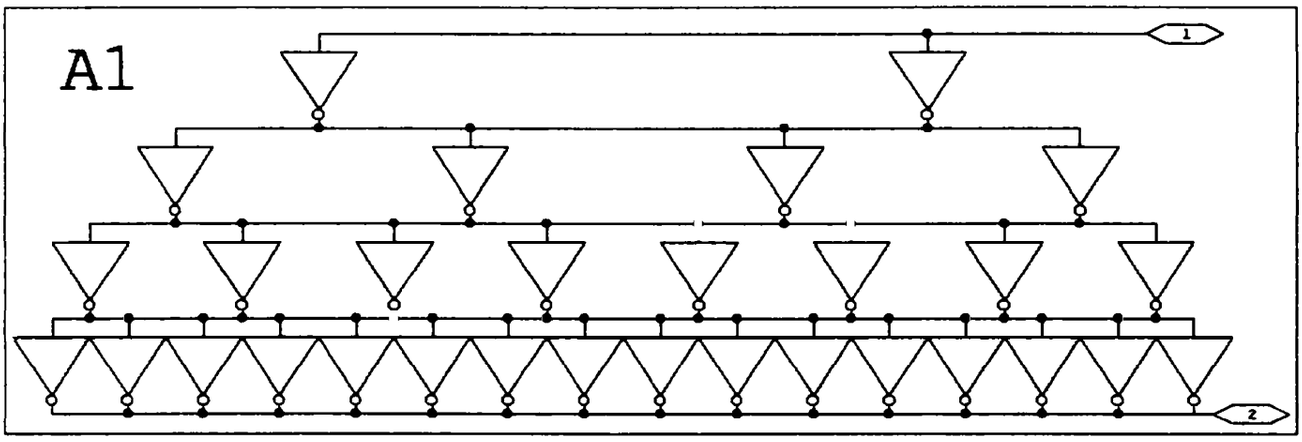


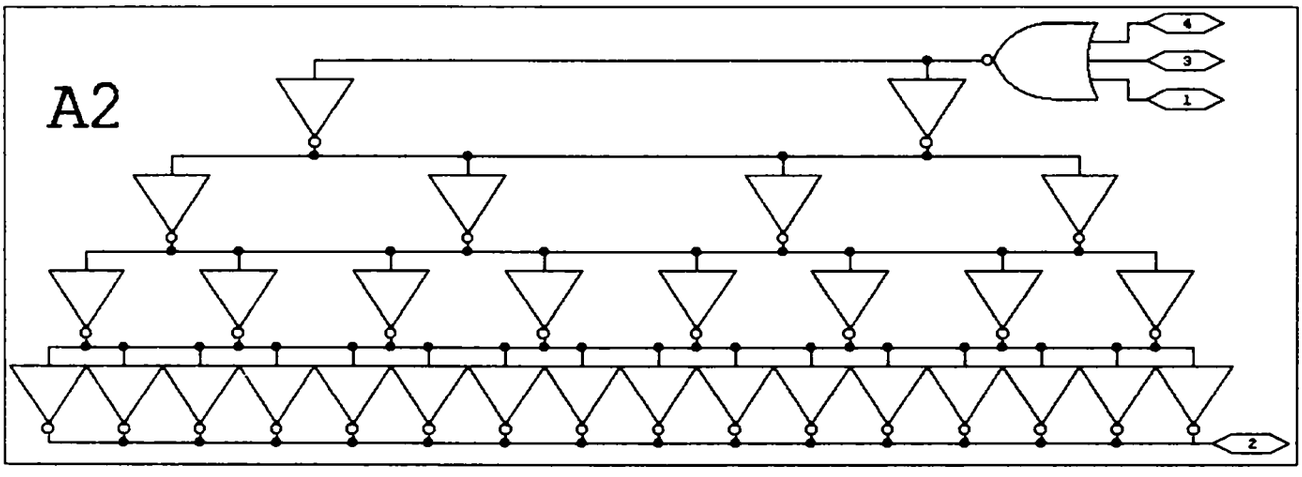


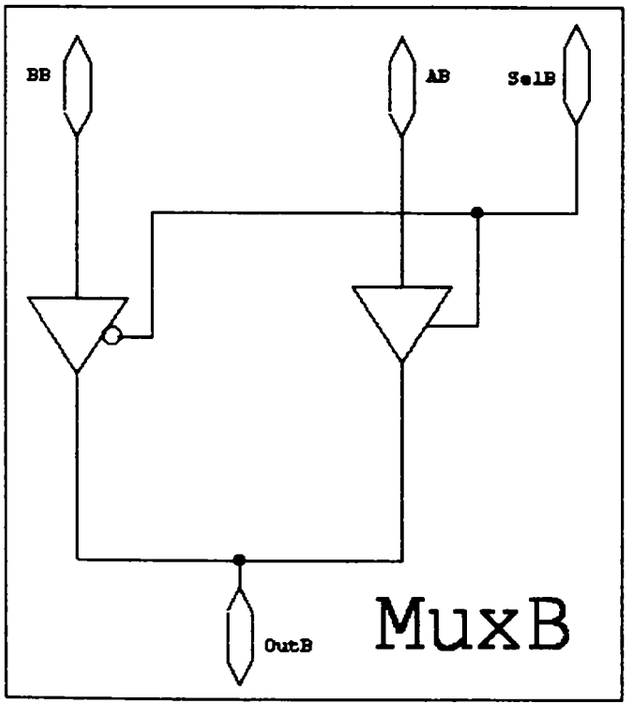
A5. 5

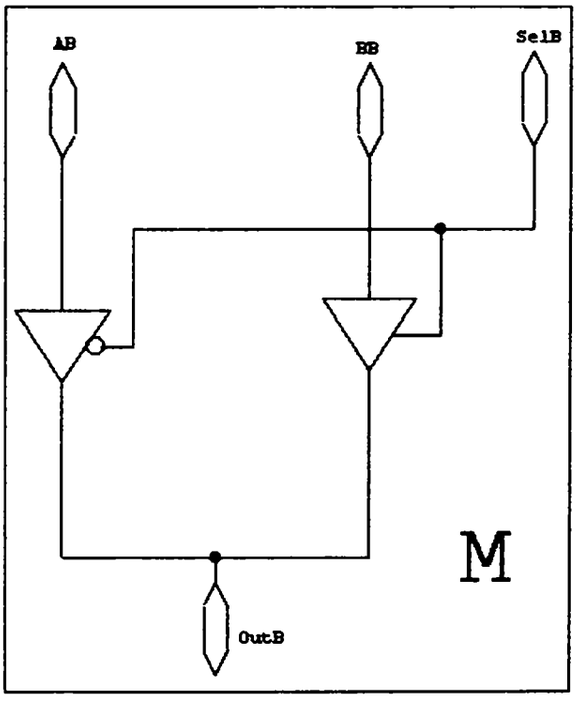



CM[22:0]
L0
C0[7:0]
C0[7:0]
CM[23:0]
CM[23:0]
A
1
A
1
Mux
B
SelB0utB
Detaliu - BS_A (BS_B)

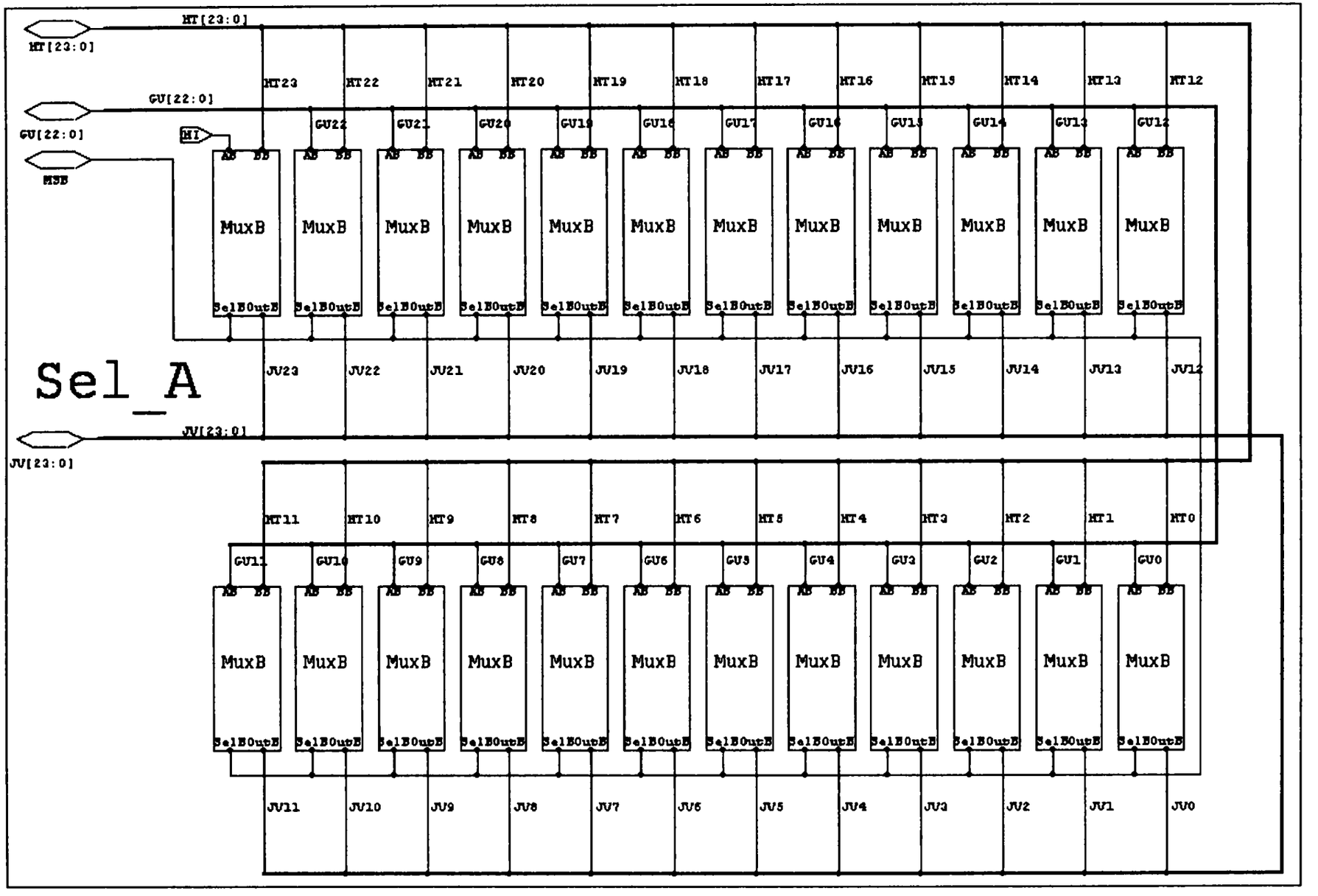




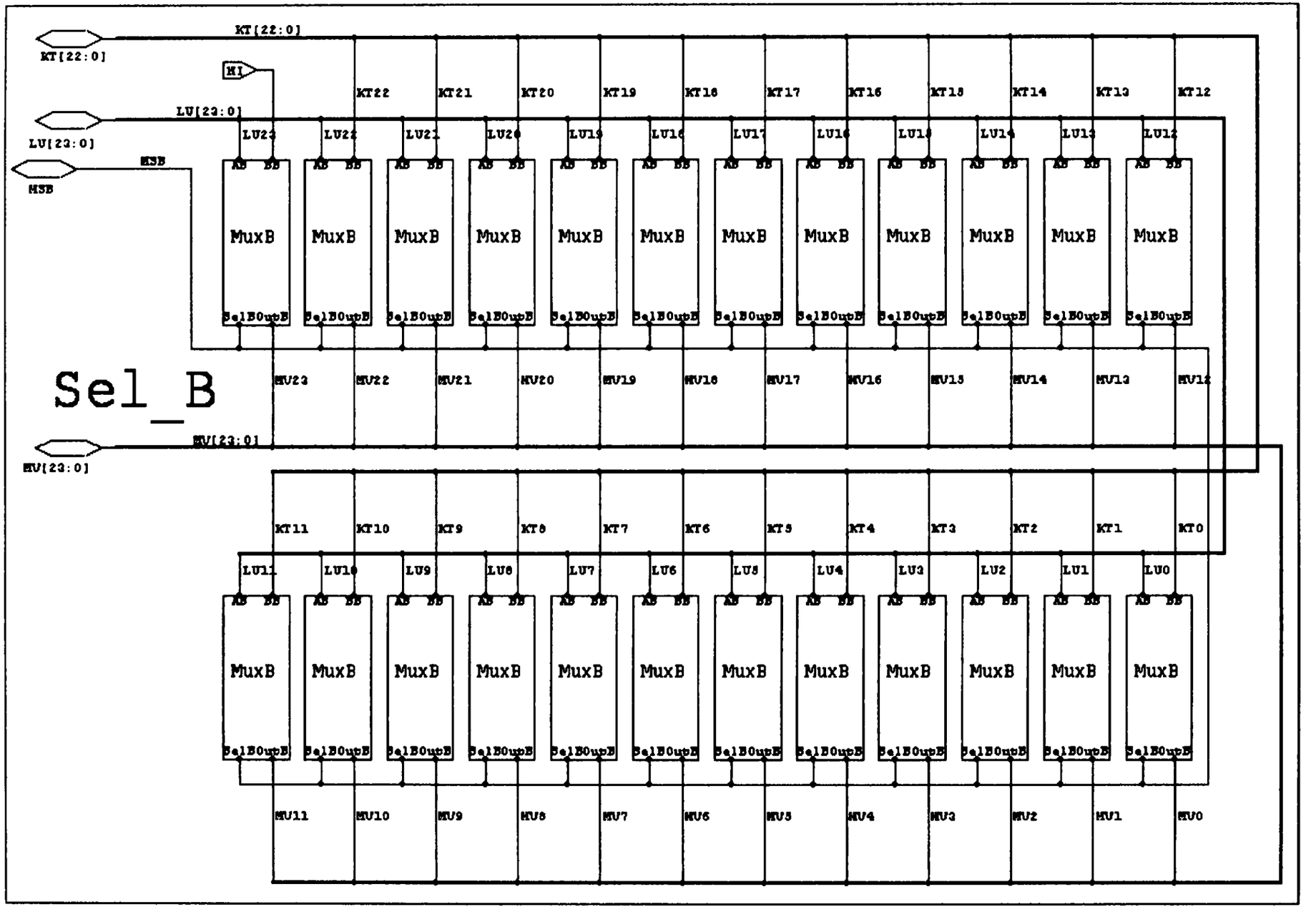




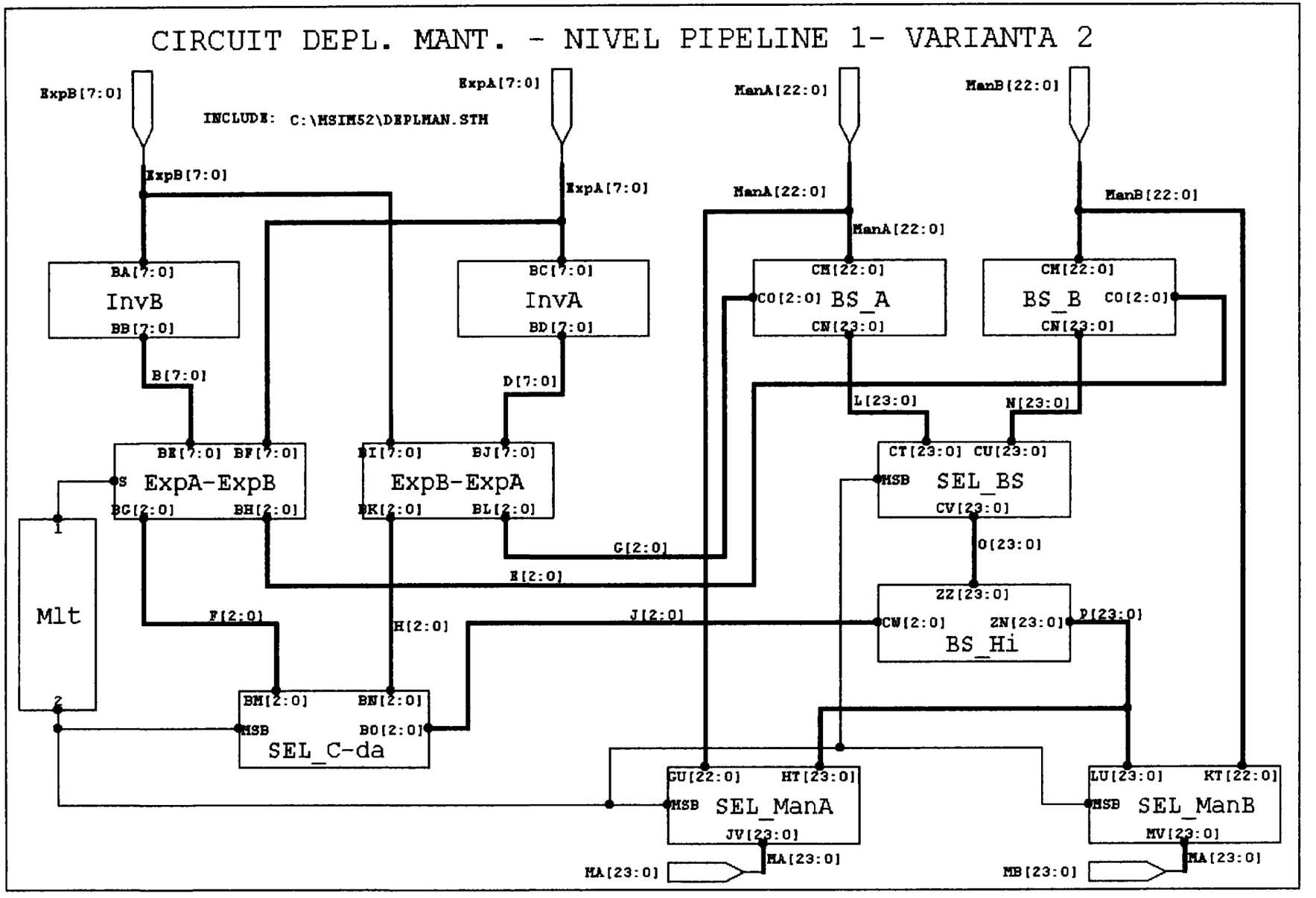
ANEXA 5

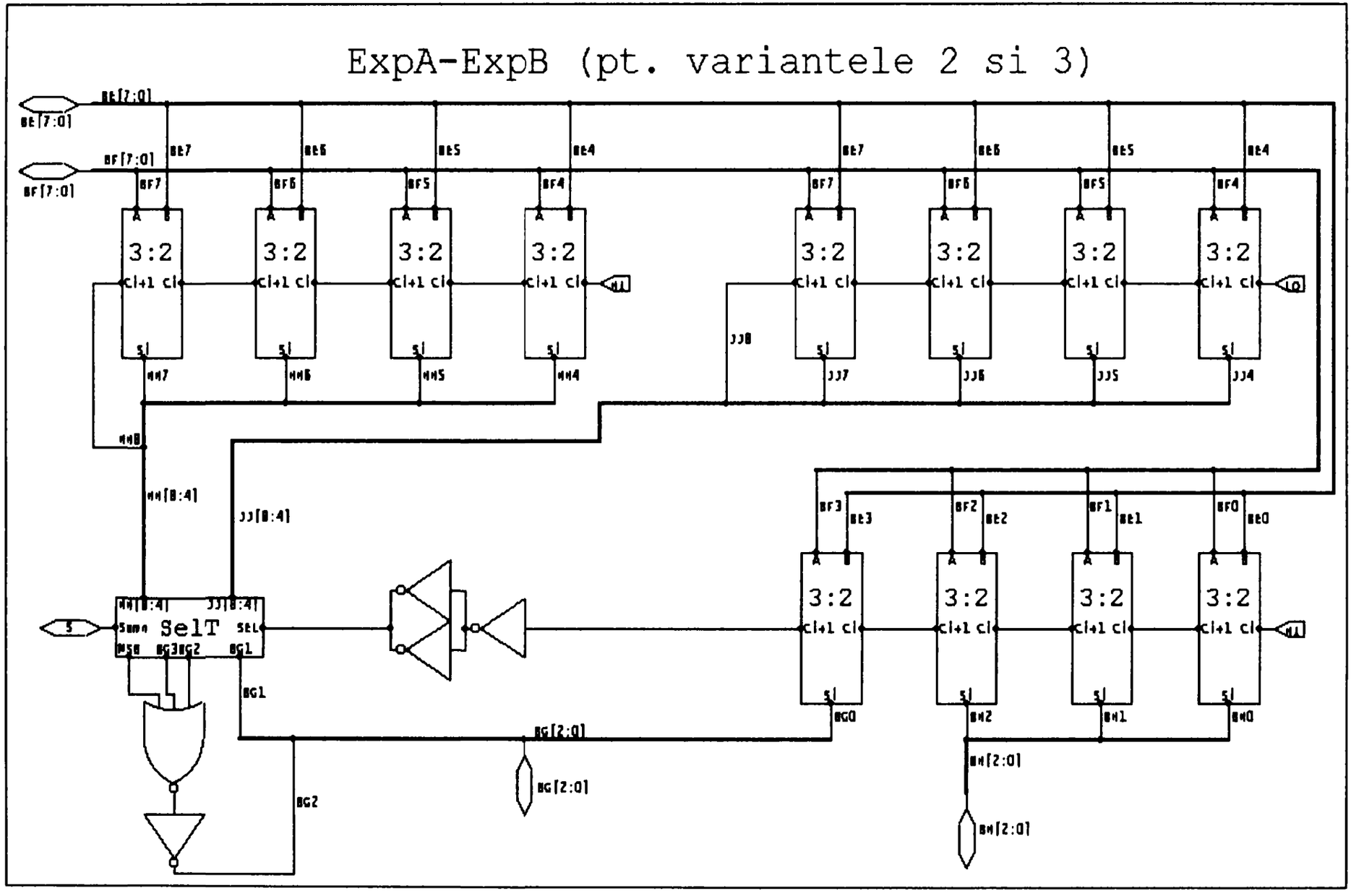

HT[23:0]
HT[23:0]
GU[22:0]
GU[22:0]
MI
MSB
Sel_A
JV[23:0]
JV[23:0]
MuxB
SelBOutB
HT23 HT22 HT21 HT20 HT19 HT18 HT17 HT16 HT15 HT14 HT13 HT12
GU22 GU21 GU20 GU19 GU18 GU17 GU16 GU15 GU14 GU13 GU12
JV23 JV22 JV21 JV20 JV19 JV18 JV17 JV16 JV15 JV14 JV13 JV12
HT11 HT10 HT9 HT8 HT7 HT6 HT5 HT4 HT3 HT2 HT1 HT0
GU11 GU10 GU9 GU8 GU7 GU6 GU5 GU4 GU3 GU2 GU1 GU0
JV11 JV10 JV9 JV8 JV7 JV6 JV5 JV4 JV3 JV2 JV1 JV0

ANEXA 5

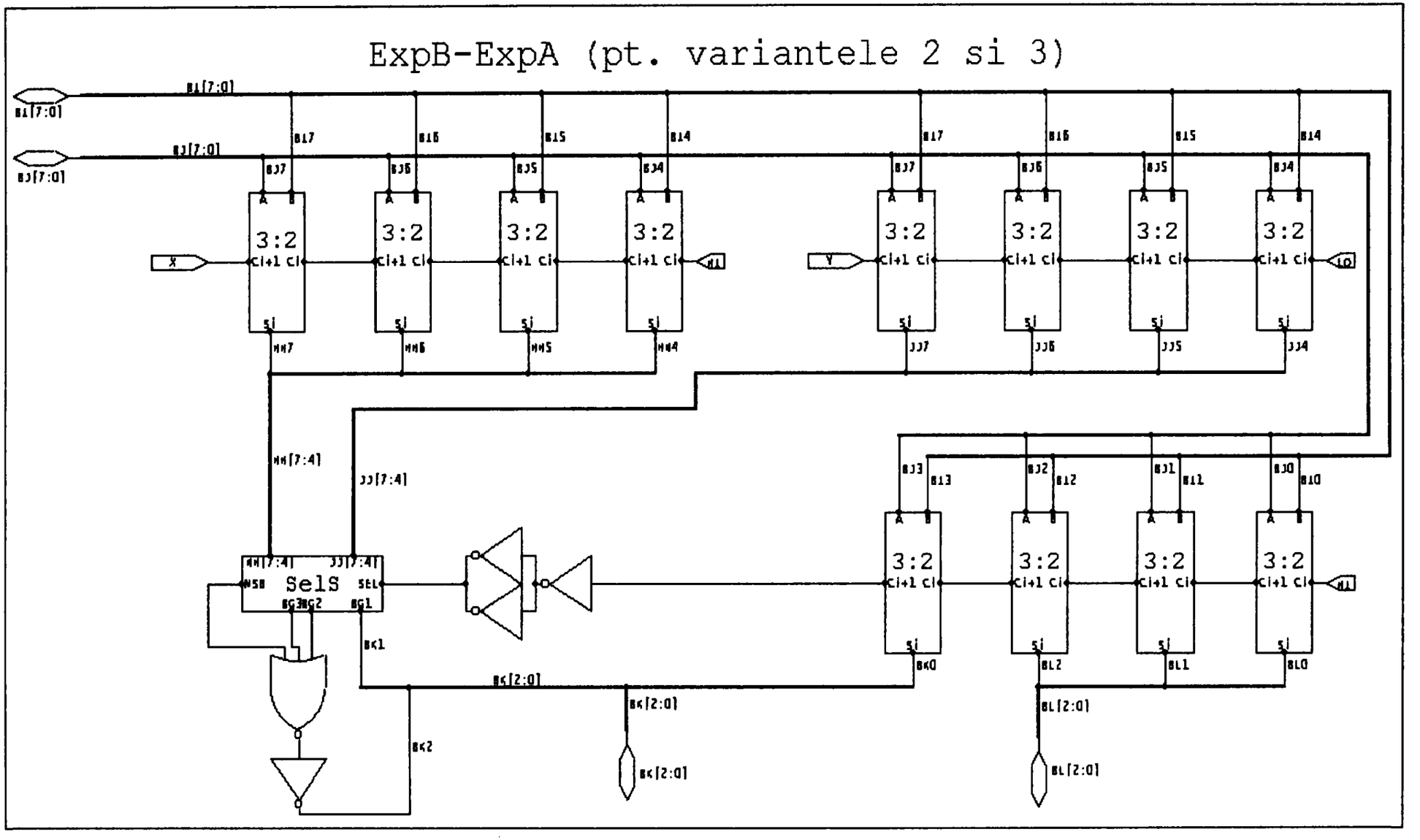

KT[22:0]
KT[22:0]
MI
LU[23:0]
LU[23:0]
MSB
MSB
Sel_B
MV[23:0]
MV[23:0]
MuxB
SelB0upB
KT22
KT21
KT20
KT19
KT18
KT17
KT16
KT15
KT14
KT13
KT12
KT11
KT10
KT9
KT8
KT7
KT6
KT5
KT4
KT3
KT2
KT1
KT0
LU11
LU10
LU9
LU8
LU7
LU6
LU5
LU4
LU3
LU2
LU1
LU0
MV23
MV22
MV21
MV20
MV19
MV18
MV17
MV16
MV15
MV14
MV13
MV12
MV11
MV10
MV9
MV8
MV7
MV6
MV5
MV4
MV3
MV2
MV1
MV0

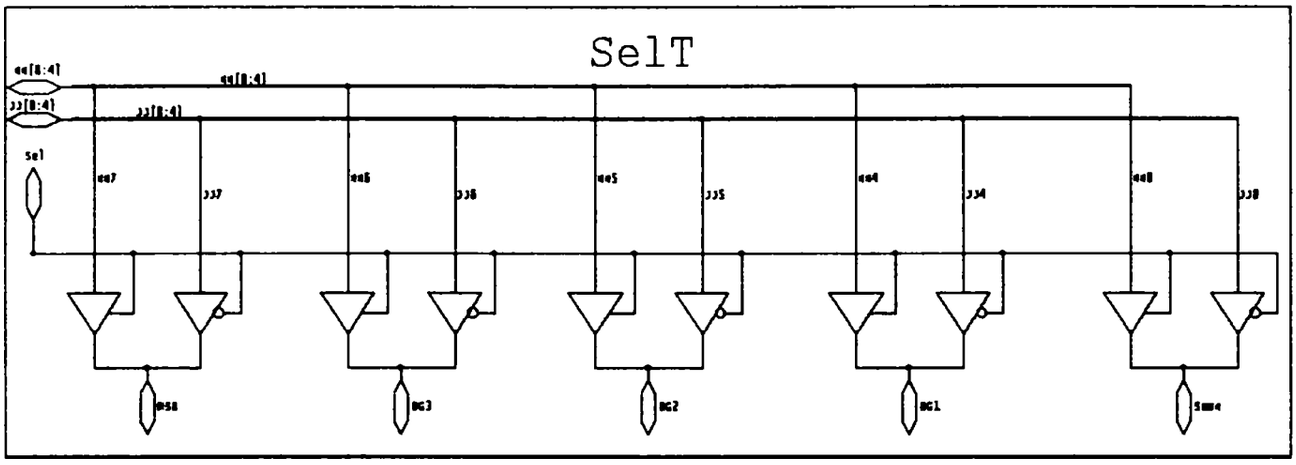
ANEXA 5

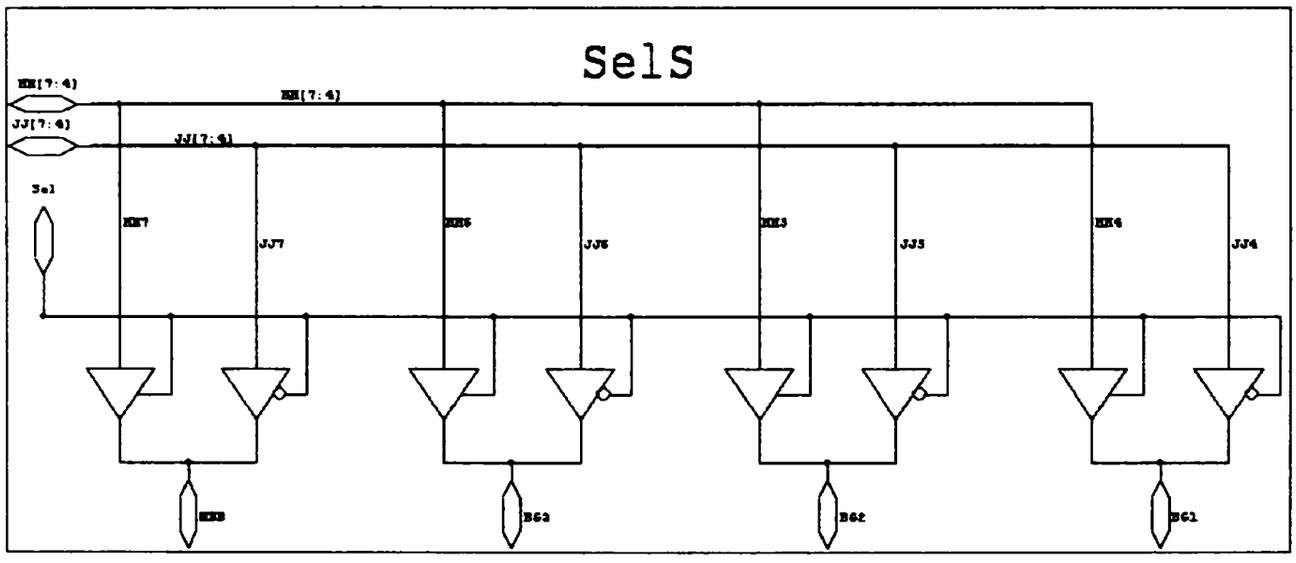

CIRCUIT DEPL. MANT. - NIVEL PIPELINE 1- VARIANTA 2
INCLUDE: C:\MSIM52\DEPLMAN.STM
ExpB[7:0]
ExpA[7:0]
ManA[22:0]
ManB[22:0]
BA[7:0]
InvB
BB[7:0]
BC[7:0]
InvA
BD[7:0]
CM[22:0]
CO[2:0]
BS_A
CN[23:0]
BS_B
B[7:0]
D[7:0]
L[23:0]
N[23:0]
BE[7:0] BF[7:0]
ExpA-ExpB
BG[2:0] BH[2:0]
BI[7:0] BJ[7:0]
ExpB-ExpA
BK[2:0] BL[2:0]
CT[23:0] CU[23:0]
MSB
SEL_BS
CV[23:0]
G[2:0]
E[2:0]
O[23:0]
ZZ[23:0]
Mlt
F[2:0]
H[2:0]
J[2:0]
CW[2:0]
ZN[23:0]
P[23:0]
BS_Hi
BM[2:0] BN[2:0]
MSB
BO[2:0]
SEL_C-da
GU[22:0] HT[23:0]
SEL_ManA
JV[23:0]
LU[23:0] KT[22:0]
SEL_ManB
MV[23:0]
MA[23:0]
MB[23:0]

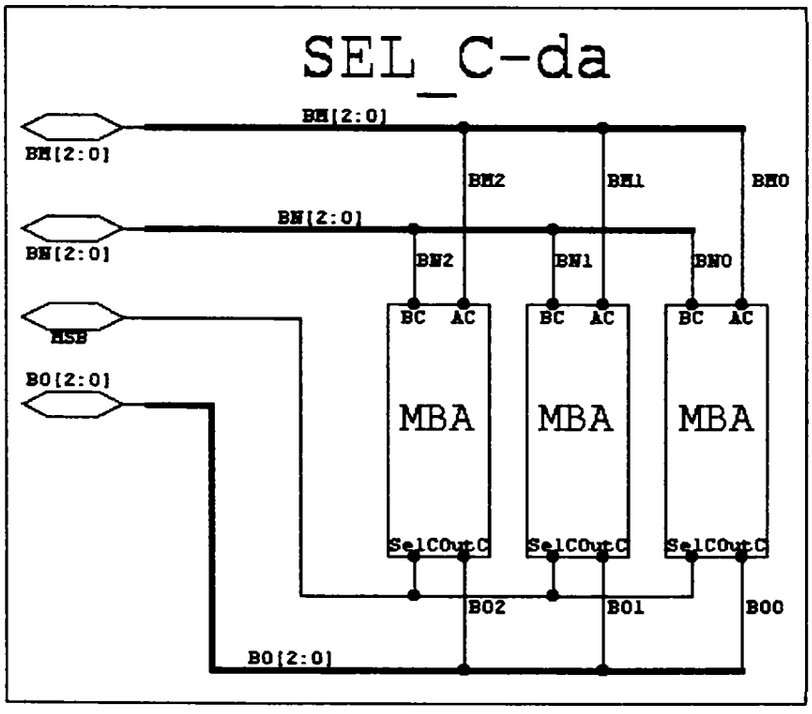

ExpA-ExpB (pt. variantele 2 si 3)
BE[7:0]
BF[7:0]
3:2
SelT
HH[8:4]
JJ[8:4]
BG[2:0]
BH[2:0]

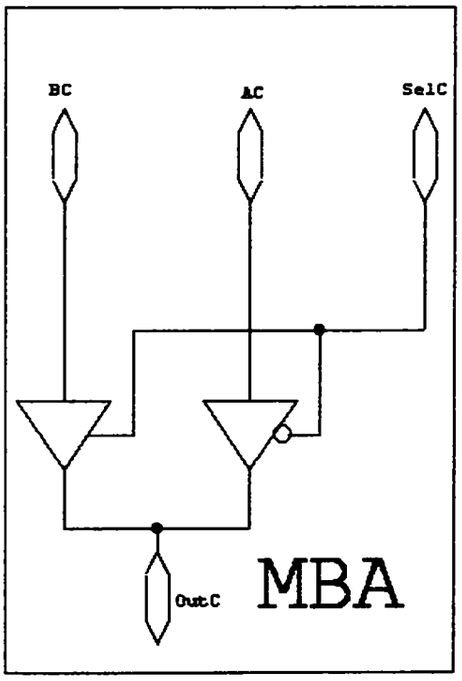
ANEXA 5



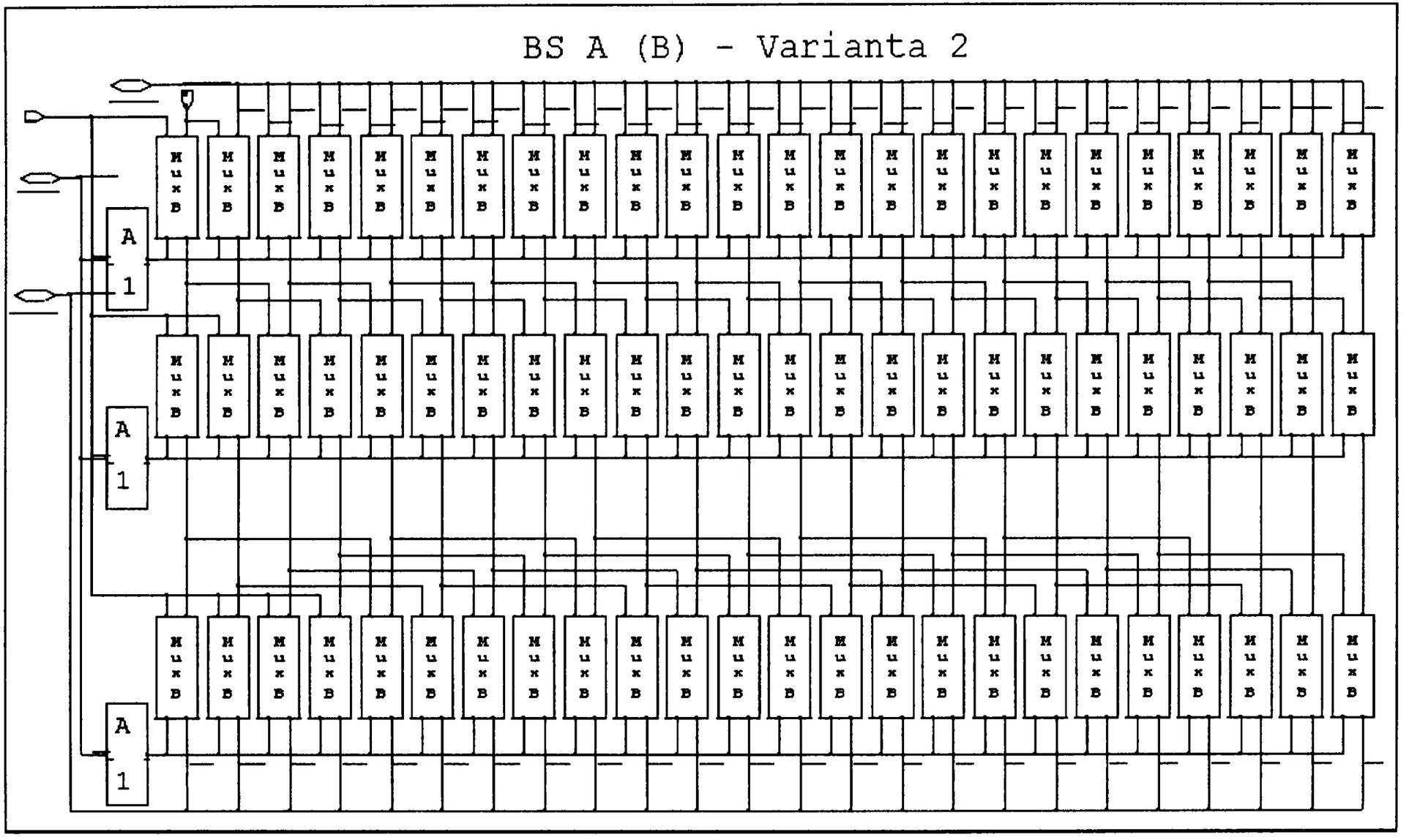

SelT
Sel


SelS
Sel

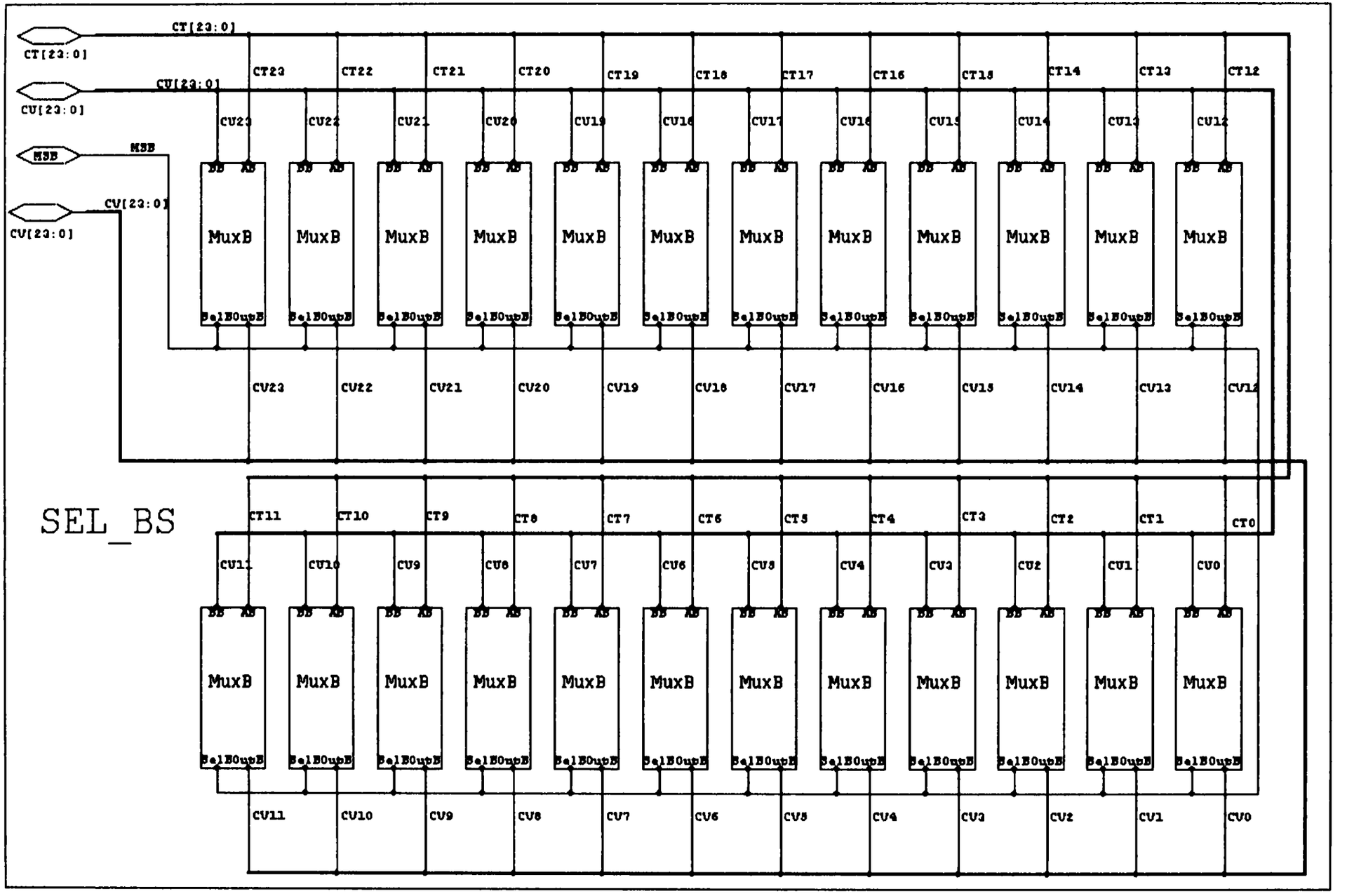

SEL_C-da
BN[2:0]
BN2
BN1
BN0
MSB
B0[2:0]
BC
AC
MBA
SelCOutC
B02
B01
B00


BC
AC
SelC
OutC
MBA

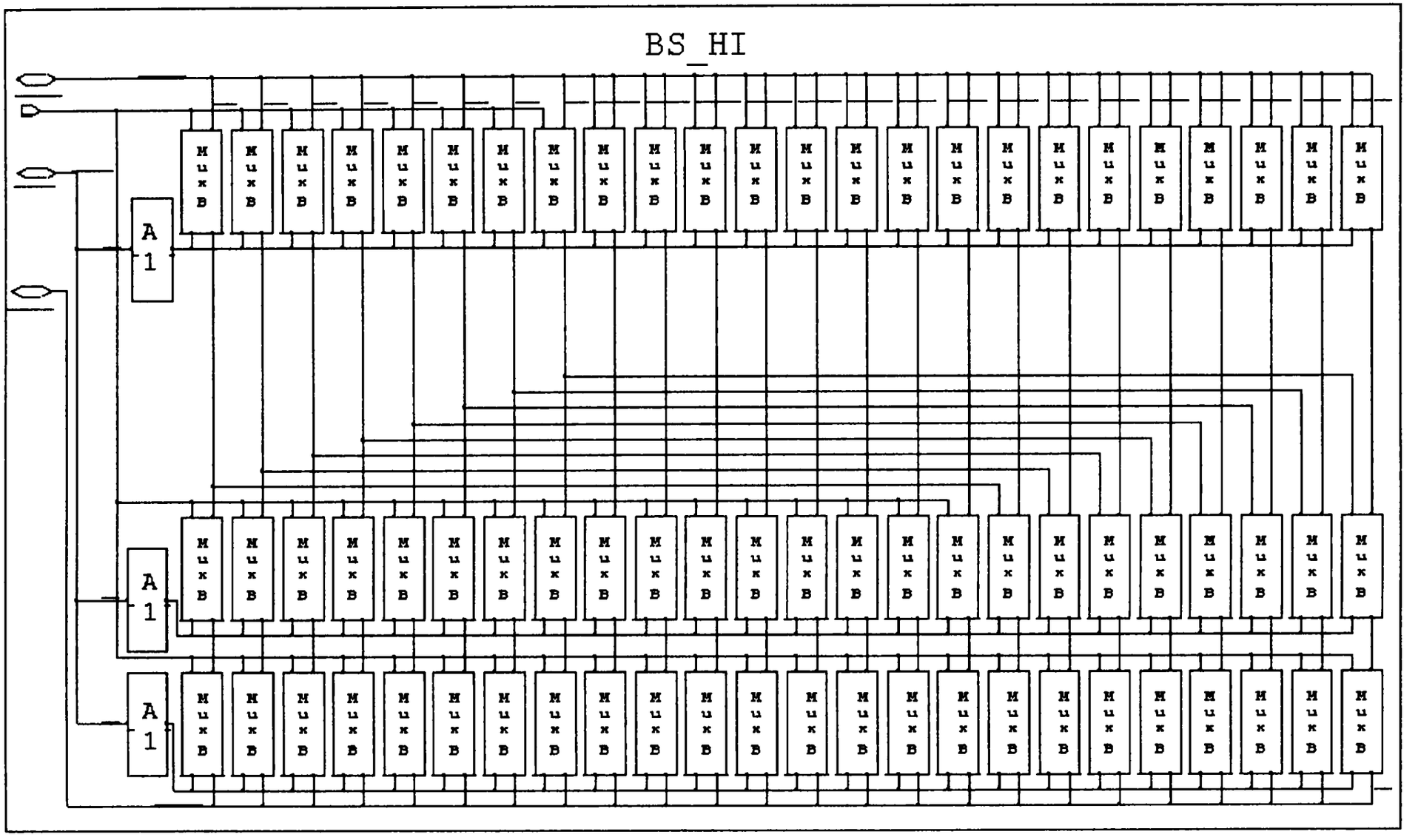
ANEXA 5

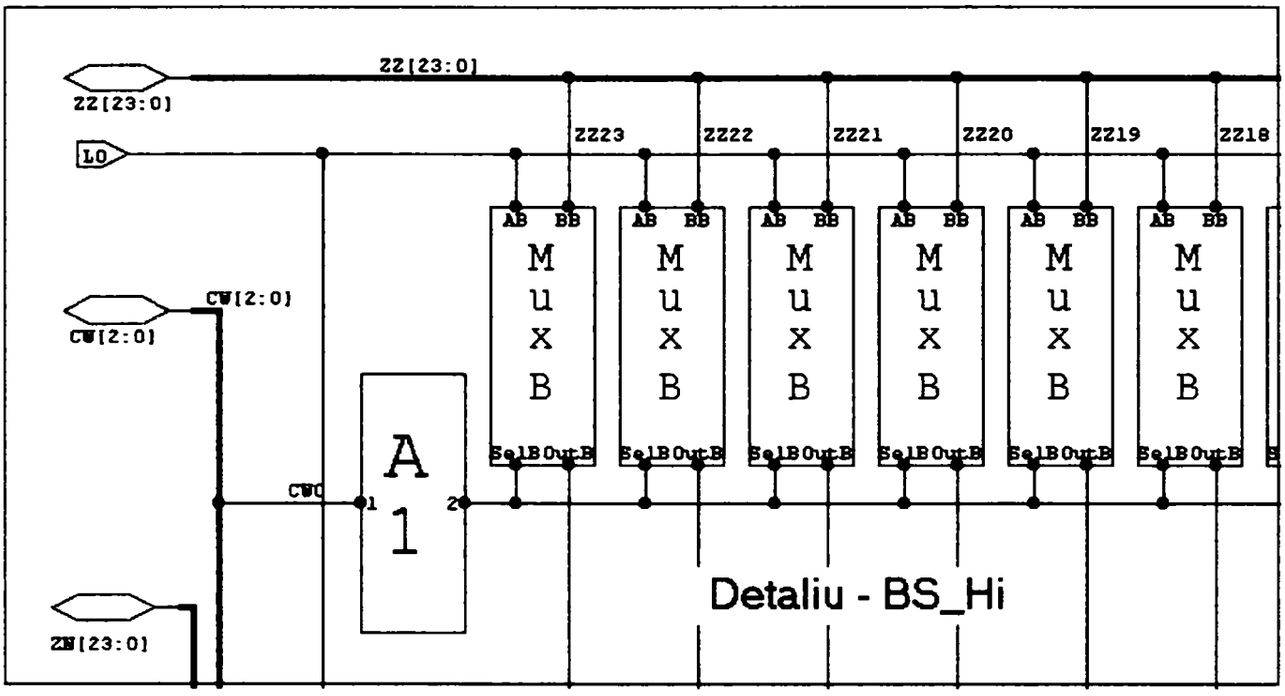


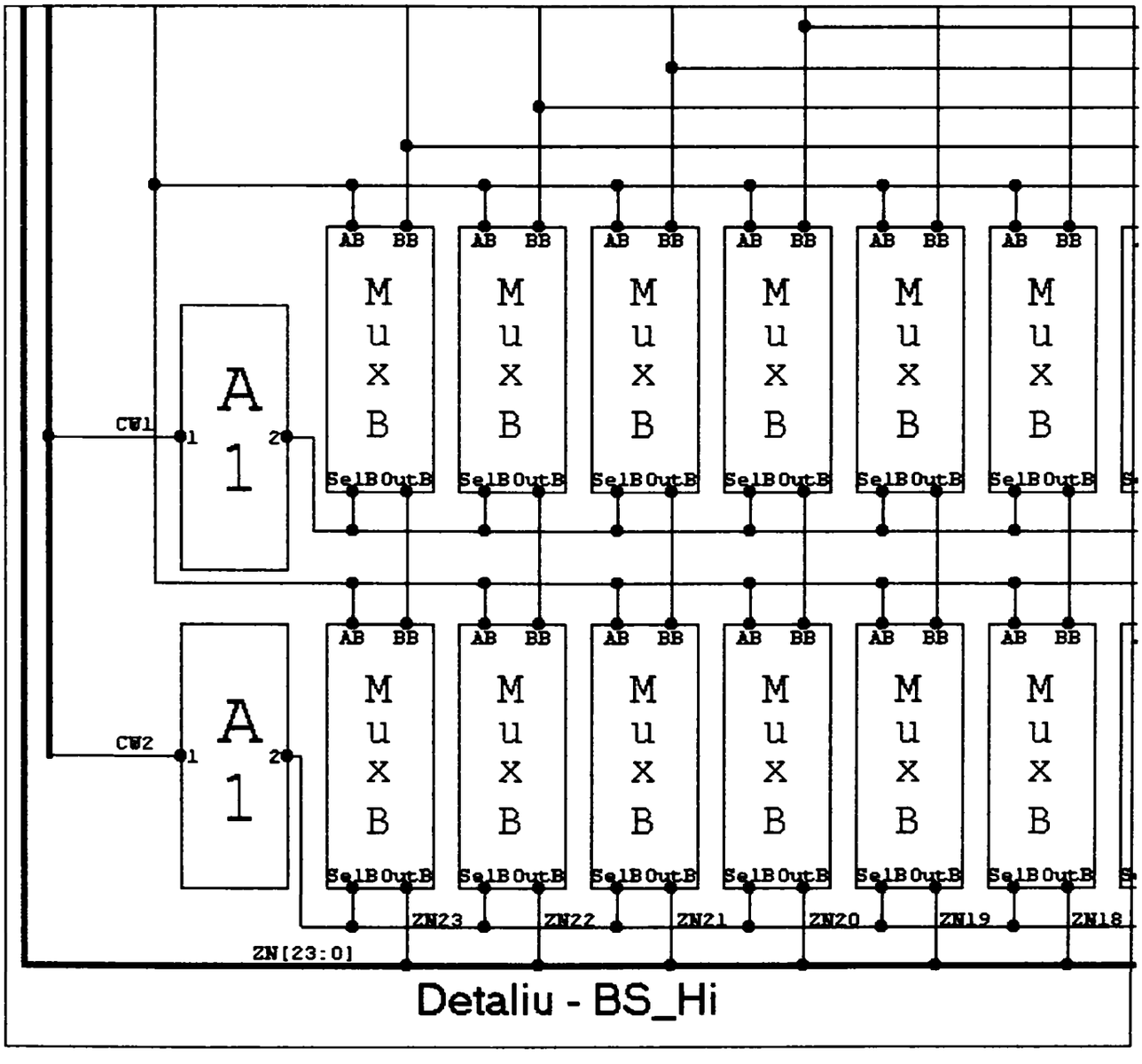
ANEXA 5


CT[23:0]
CV[23:0]
MSB
SEL_BS
MuxB
SelE0ubE
CT23 CT22 CT21 CT20 CT19 CT18 CT17 CT16 CT15 CT14 CT13 CT12
CV23 CV22 CV21 CV20 CV19 CV18 CV17 CV16 CV15 CV14 CV13 CV12
CT11 CT10 CT9 CT8 CT7 CT6 CT5 CT4 CT3 CT2 CT1 CT0
CV11 CV10 CV9 CV8 CV7 CV6 CV5 CV4 CV3 CV2 CV1 CV0

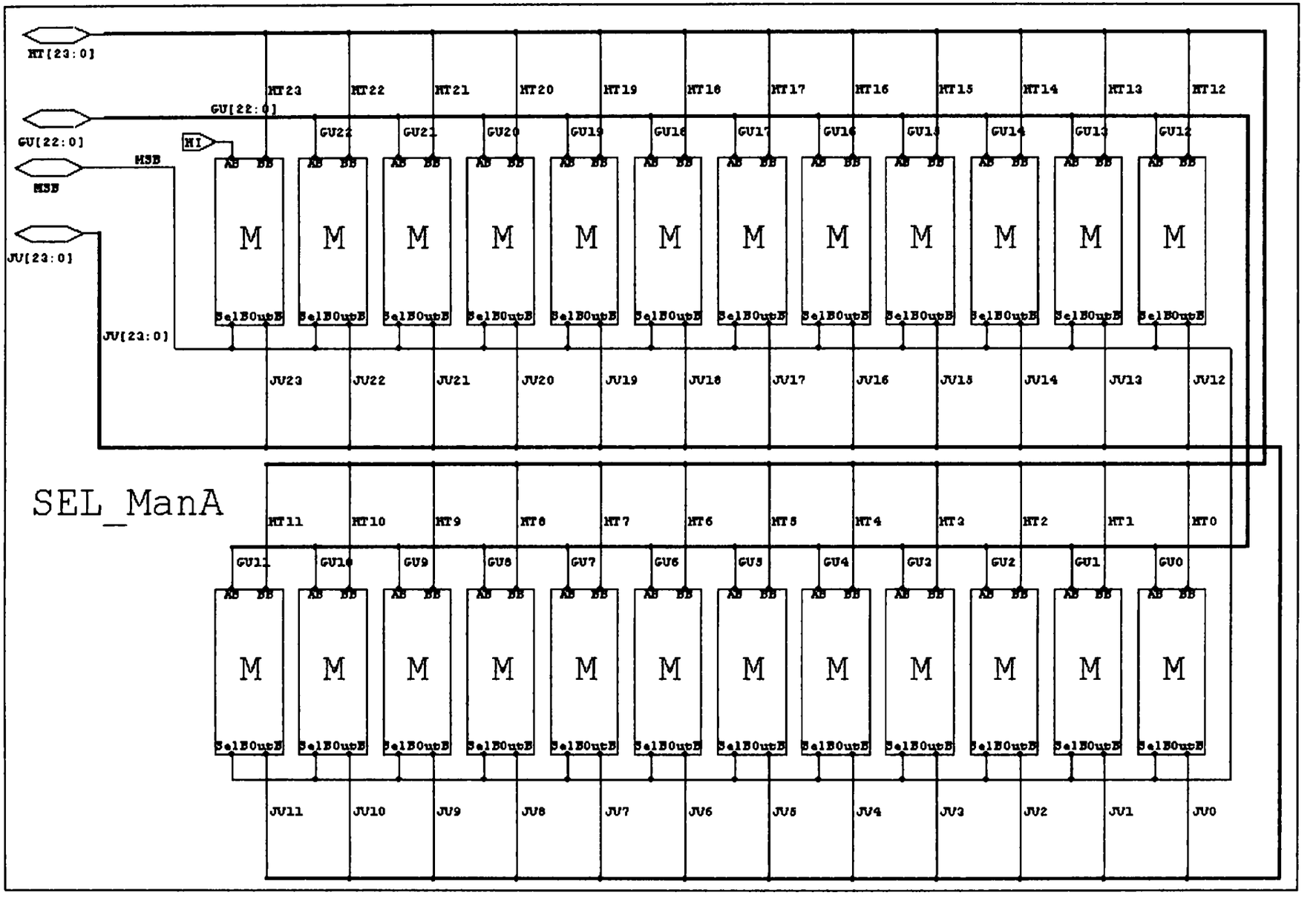
ANEXA 5



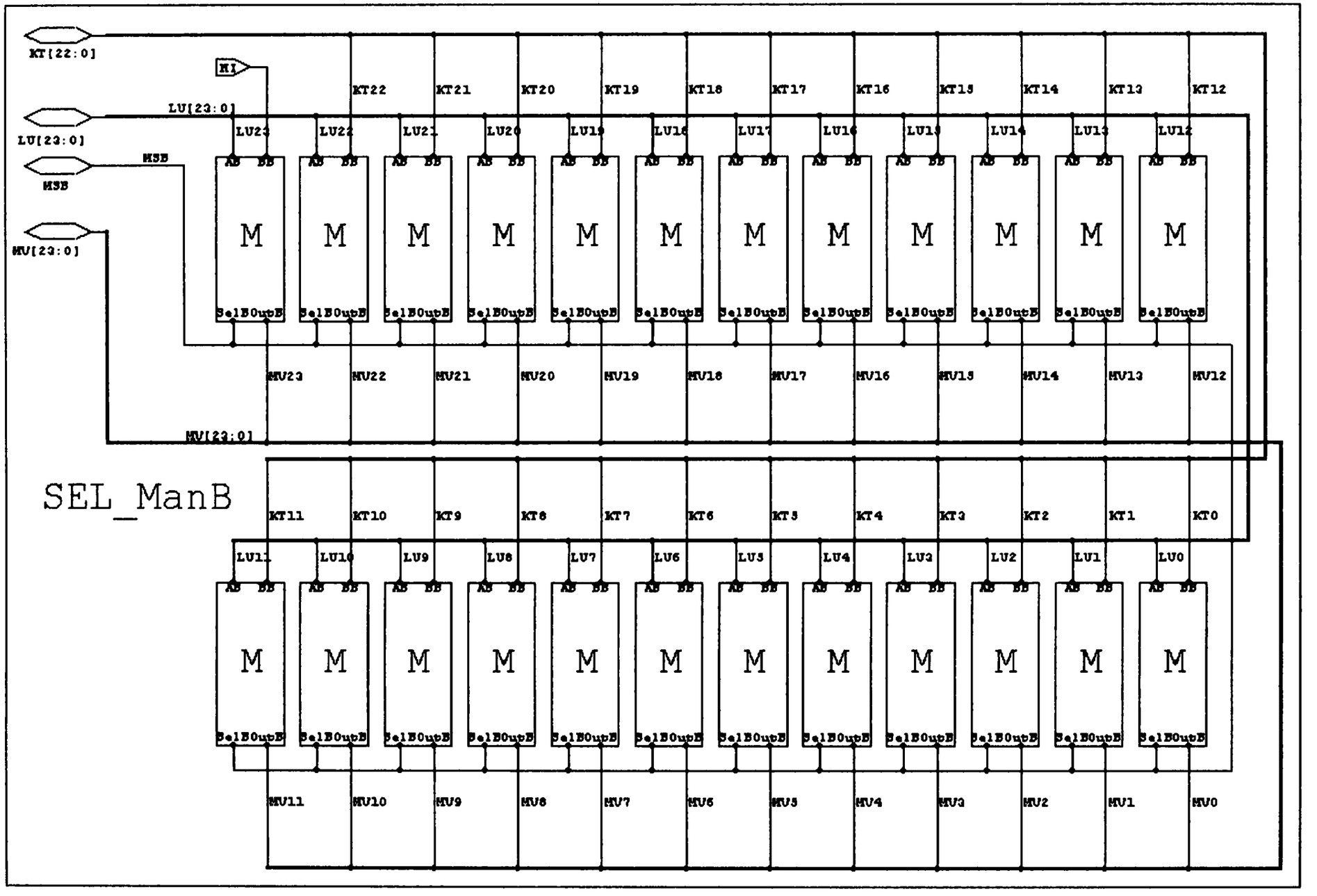

ZZ[23:0]
ZZ[23:0]
LO
ZZ23
ZZ22
ZZ21
ZZ20
ZZ19
ZZ18
CW[2:0]
CW[2:0]
A
1
CW0
ZN[23:0]
AB BB
M
u
x
B
SelBOutB
Detaliu - BS_Hi


CW1
CW2
A
1
AB BB
M
u
x
B
SelBOutB
ZN23
ZN22
ZN21
ZN20
ZN19
ZN18
ZN[23:0]
Detaliu - BS_Hi

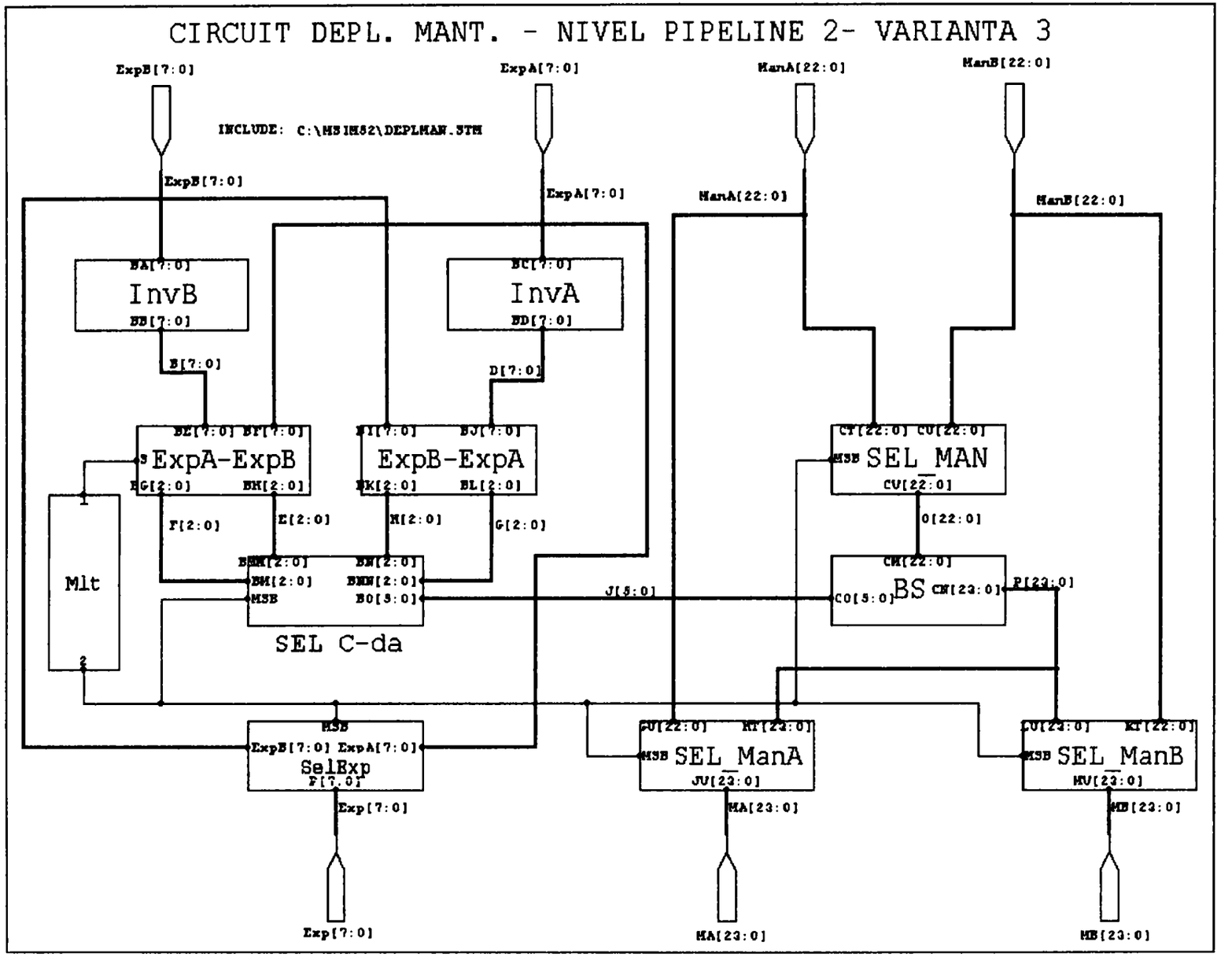
ANEXA 5

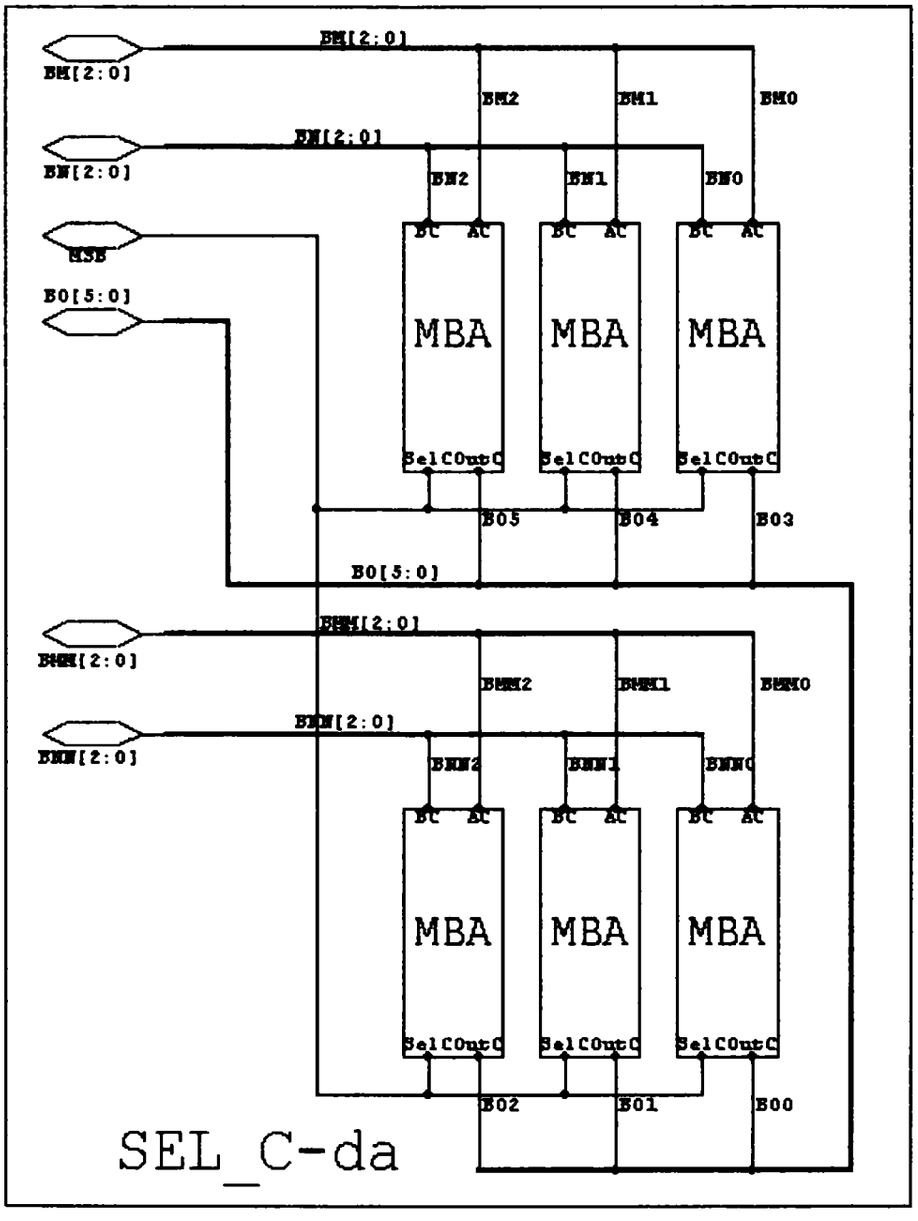

MT[23:0]
GU[22:0]
MI
MSB
JV[23:0]
SEL_ManA
M
SelBOutB
AB
BB
MT23
MT22
MT21
MT20
MT19
MT18
MT17
MT16
MT15
MT14
MT13
MT12
GU22
GU21
GU20
GU19
GU18
GU17
GU16
GU15
GU14
GU13
GU12
JV23
JV22
JV21
JV20
JV19
JV18
JV17
JV16
JV15
JV14
JV13
JV12
MT11
MT10
MT9
MT8
MT7
MT6
MT5
MT4
MT3
MT2
MT1
MT0
GU11
GU10
GU9
GU8
GU7
GU6
GU5
GU4
GU3
GU2
GU1
GU0
JV11
JV10
JV9
JV8
JV7
JV6
JV5
JV4
JV3
JV2
JV1
JV0

ANEXA 5

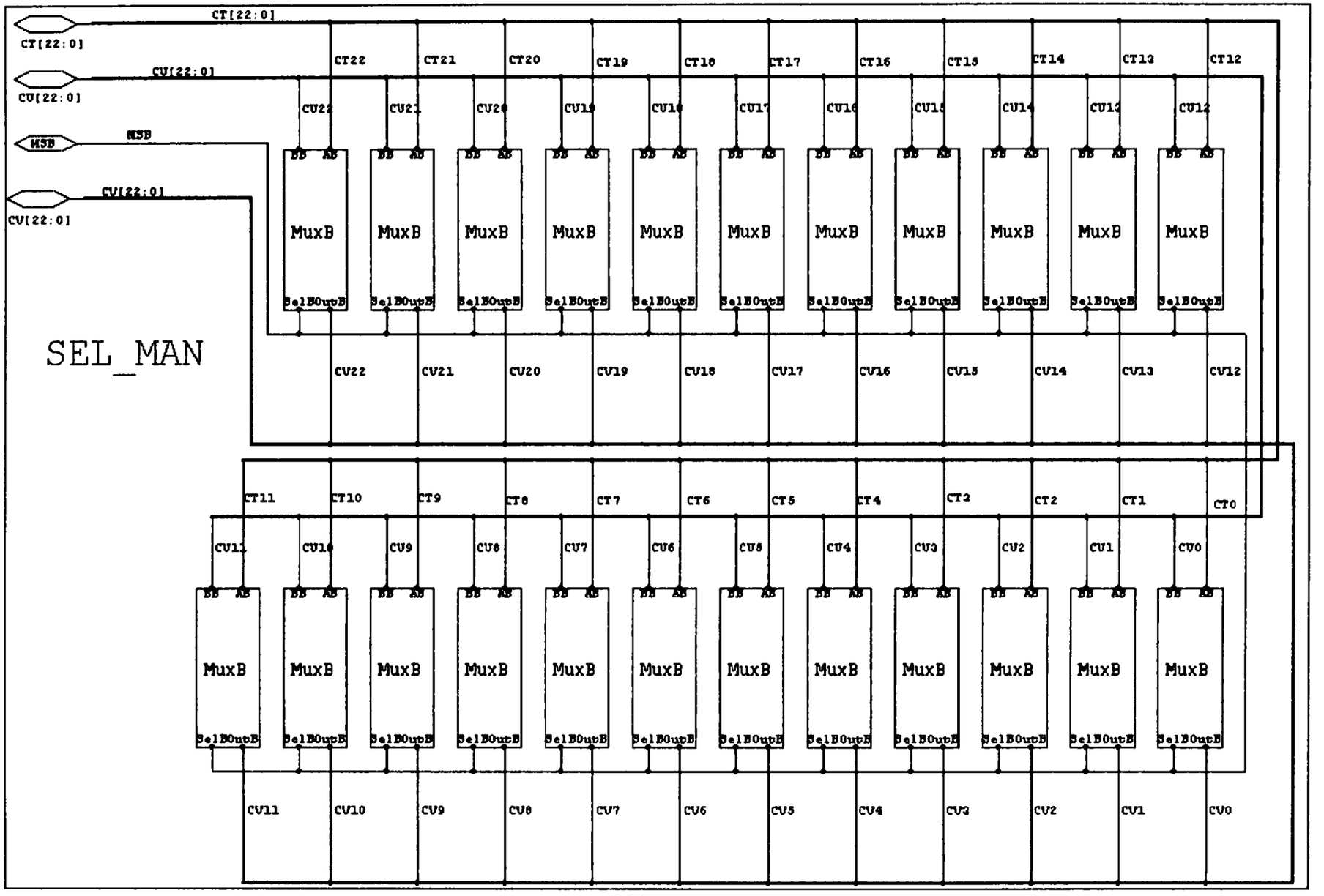

KT[22:0]
MI
LU[23:0]
LU[23:0]
MSB
MSB
MV[23:0]
MV[23:0]
SEL_ManB
M
AB
BB
SelB0utB
KT22
KT21
KT20
KT19
KT18
KT17
KT16
KT15
KT14
KT13
KT12
LU23
LU22
LU21
LU20
LU19
LU18
LU17
LU16
LU15
LU14
LU13
LU12
MV23
MV22
MV21
MV20
MV19
MV18
MV17
MV16
MV15
MV14
MV13
MV12
KT11
KT10
KT9
KT8
KT7
KT6
KT5
KT4
KT3
KT2
KT1
KT0
LU11
LU10
LU9
LU8
LU7
LU6
LU5
LU4
LU3
LU2
LU1
LU0
MV11
MV10
MV9
MV8
MV7
MV6
MV5
MV4
MV3
MV2
MV1
MV0

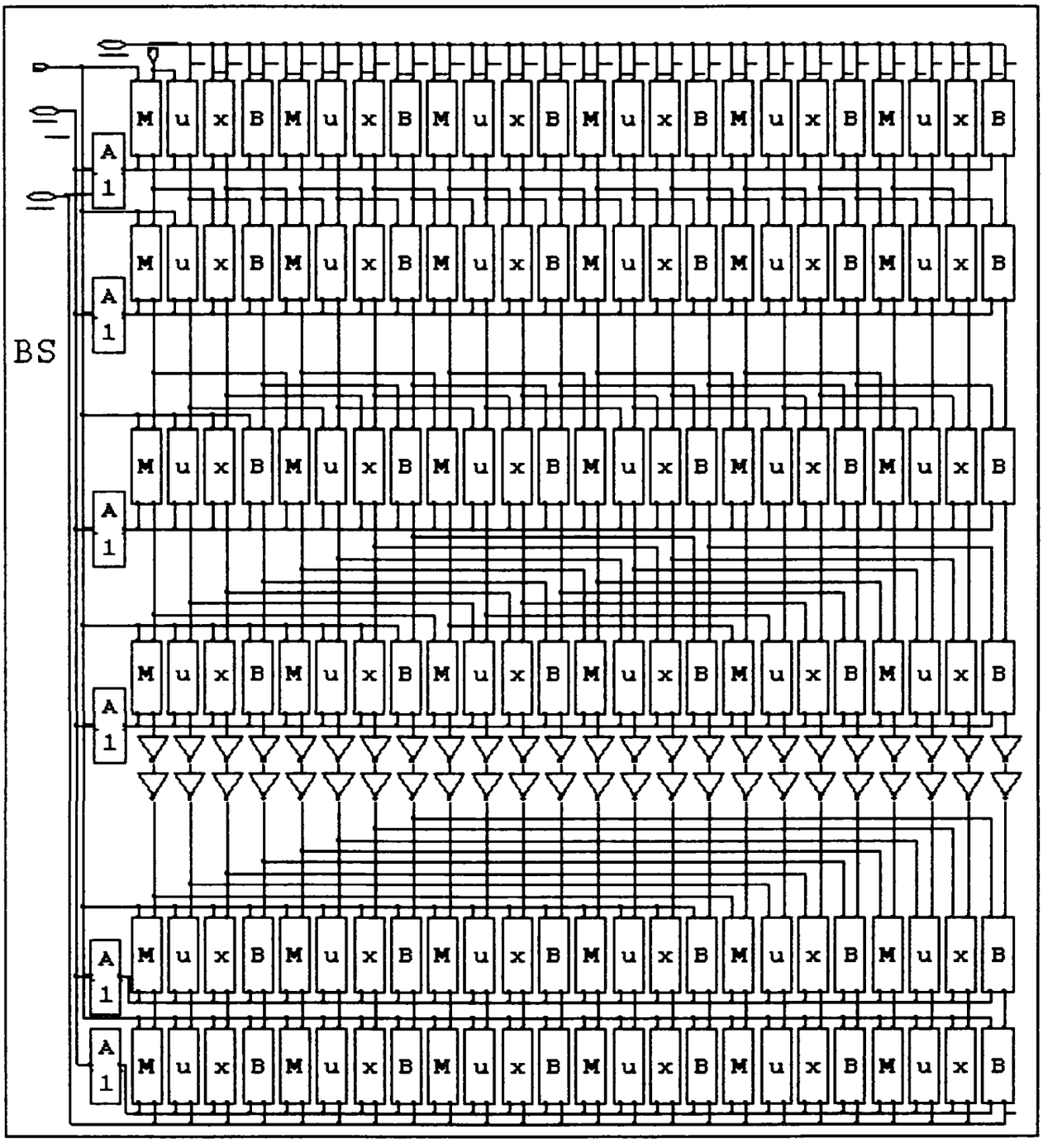
ANEXA 5

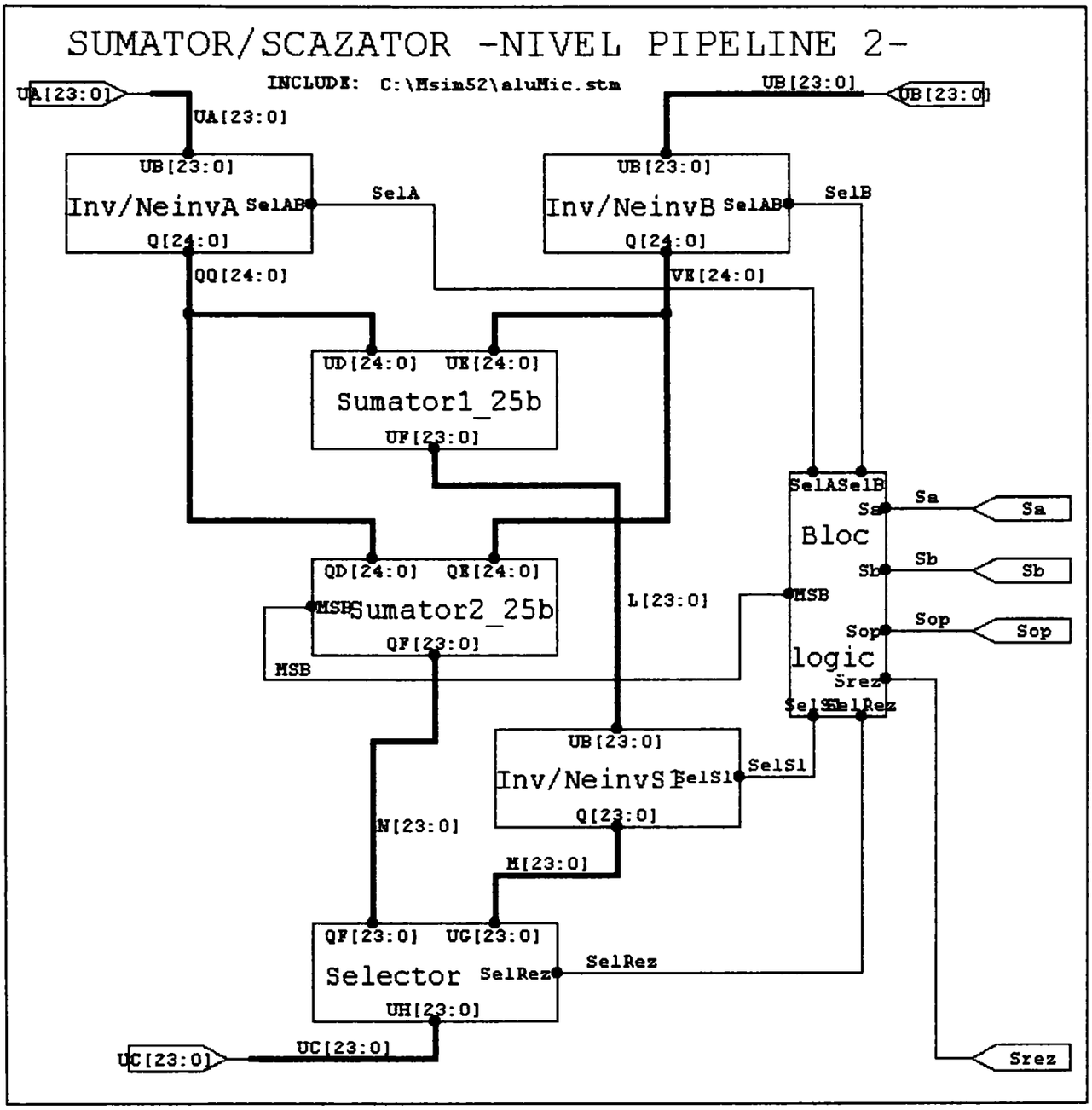

CIRCUIT DEPL. MANT. - NIVEL PIPELINE 2- VARIANTA 3
ExpB[7:0]
ExpA[7:0]
ManA[22:0]
ManB[22:0]
INCLUDE: C:\MS1M52\DEPLMAN.STH
InvB
InvA
ExpA-ExpB
ExpB-ExpA
SEL_MAN
Mlt
BS
SEL C-da
SelExp
SEL_ManA
SEL_ManB
Exp[7:0]
MA[23:0]
MB[23:0]


BM[2:0]
BM[2:0]
BM2
BM1
BM0
BN[2:0]
BN[2:0]
BN2
BN1
BN0
MSB
B0[5:0]
Bc
Ac
MBA
MBA
MBA
SelCOutC
SelCOutC
SelCOutC
B05
B04
B03
B0[5:0]
BMM[2:0]
BMM[2:0]
BMM2
BMM1
BMM0
BNN[2:0]
BNN[2:0]
BNN2
BNN1
BNN0
Bc
Ac
MBA
MBA
MBA
SelCOutC
SelCOutC
SelCOutC
B02
B01
B00
SEL_C-da

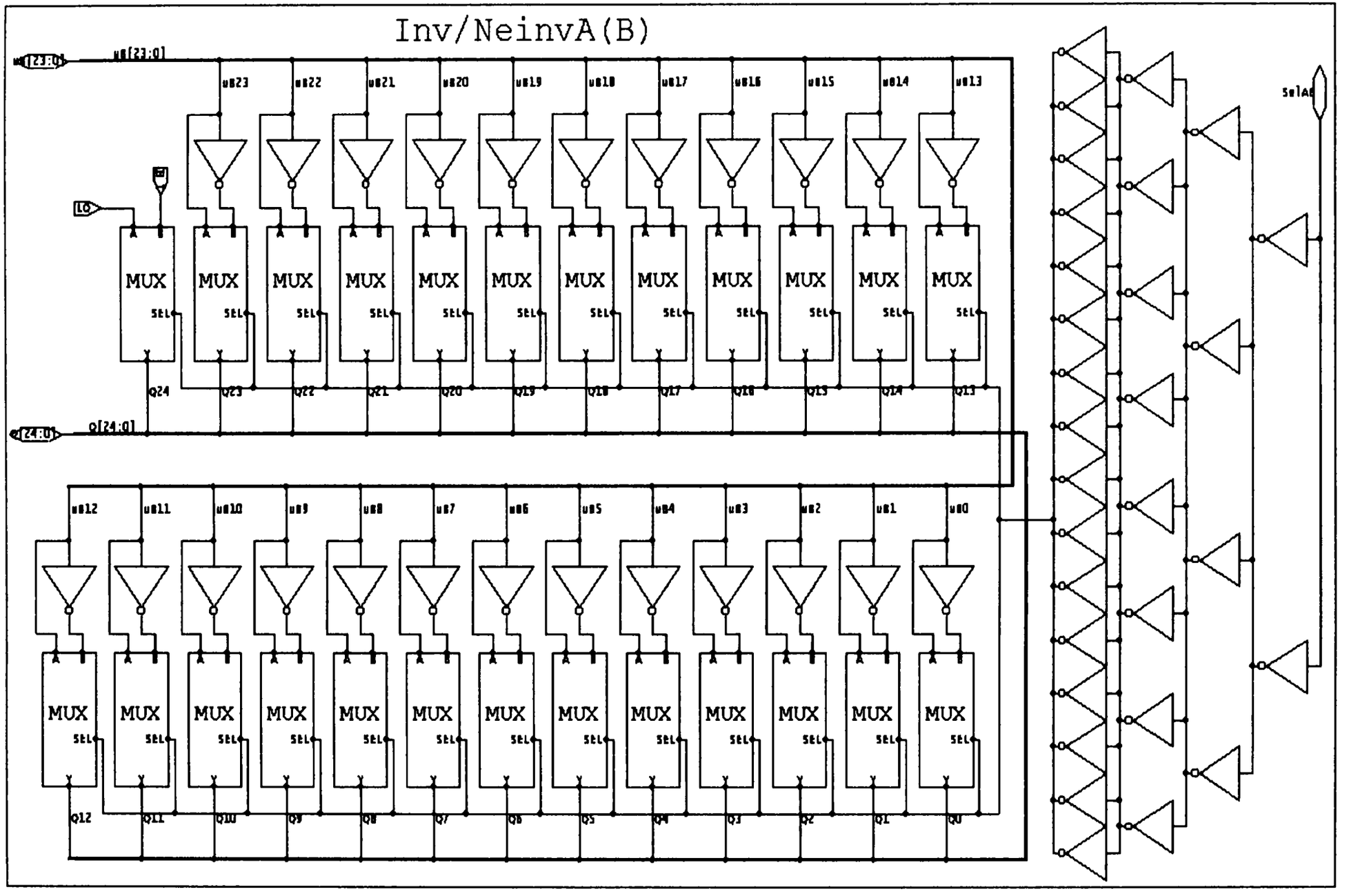
ANEXA 5


CT[22:0]
CV[22:0]
MSB
SEL_MAN
MuxB
SelBOutB
CT22
CT21
CT20
CT19
CT18
CT17
CT16
CT15
CT14
CT13
CT12
CT11
CT10
CT9
CT8
CT7
CT6
CT5
CT4
CT3
CT2
CT1
CT0
CV22
CV21
CV20
CV19
CV18
CV17
CV16
CV15
CV14
CV13
CV12
CV11
CV10
CV9
CV8
CV7
CV6
CV5
CV4
CV3
CV2
CV1
CV0

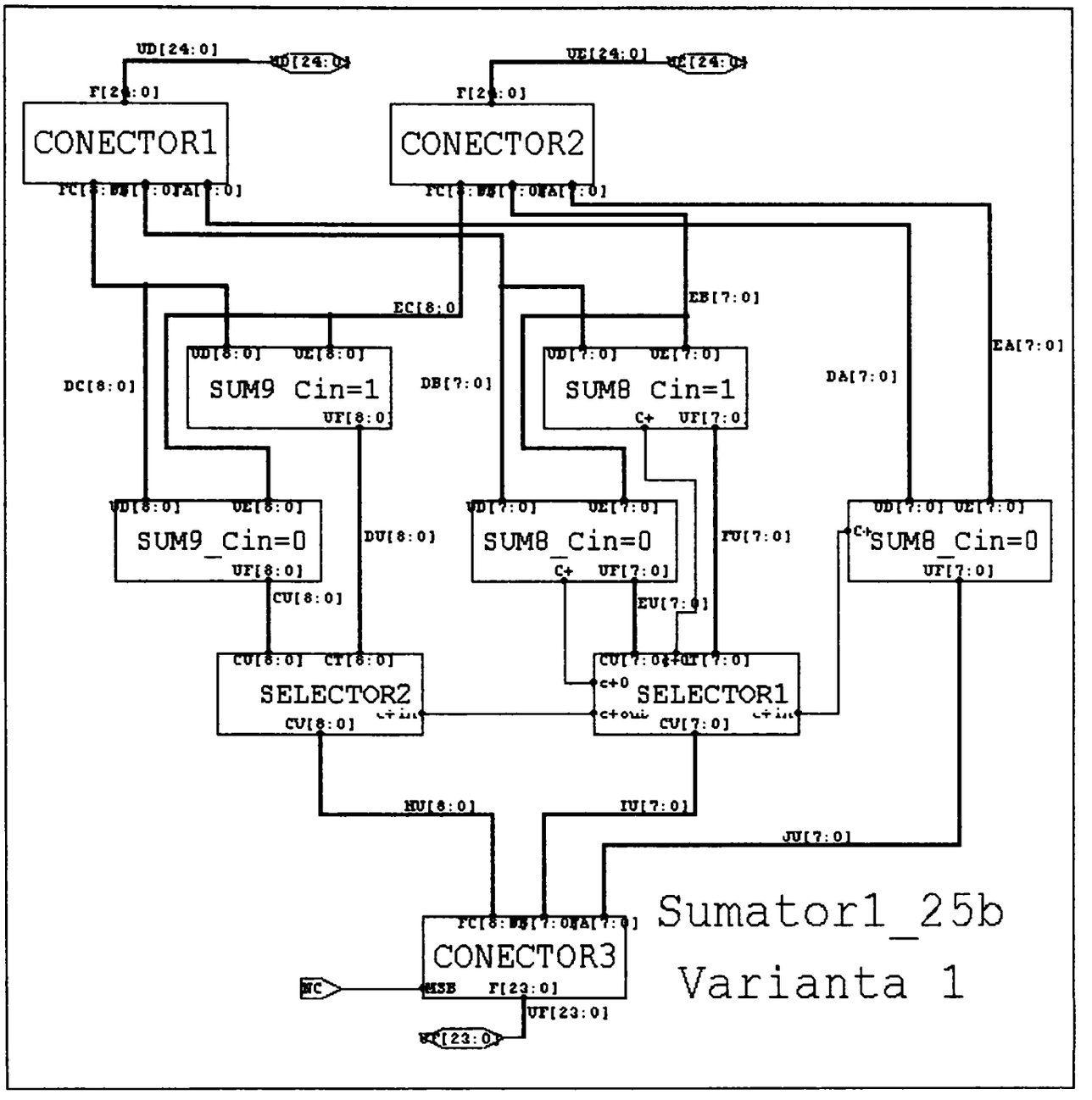


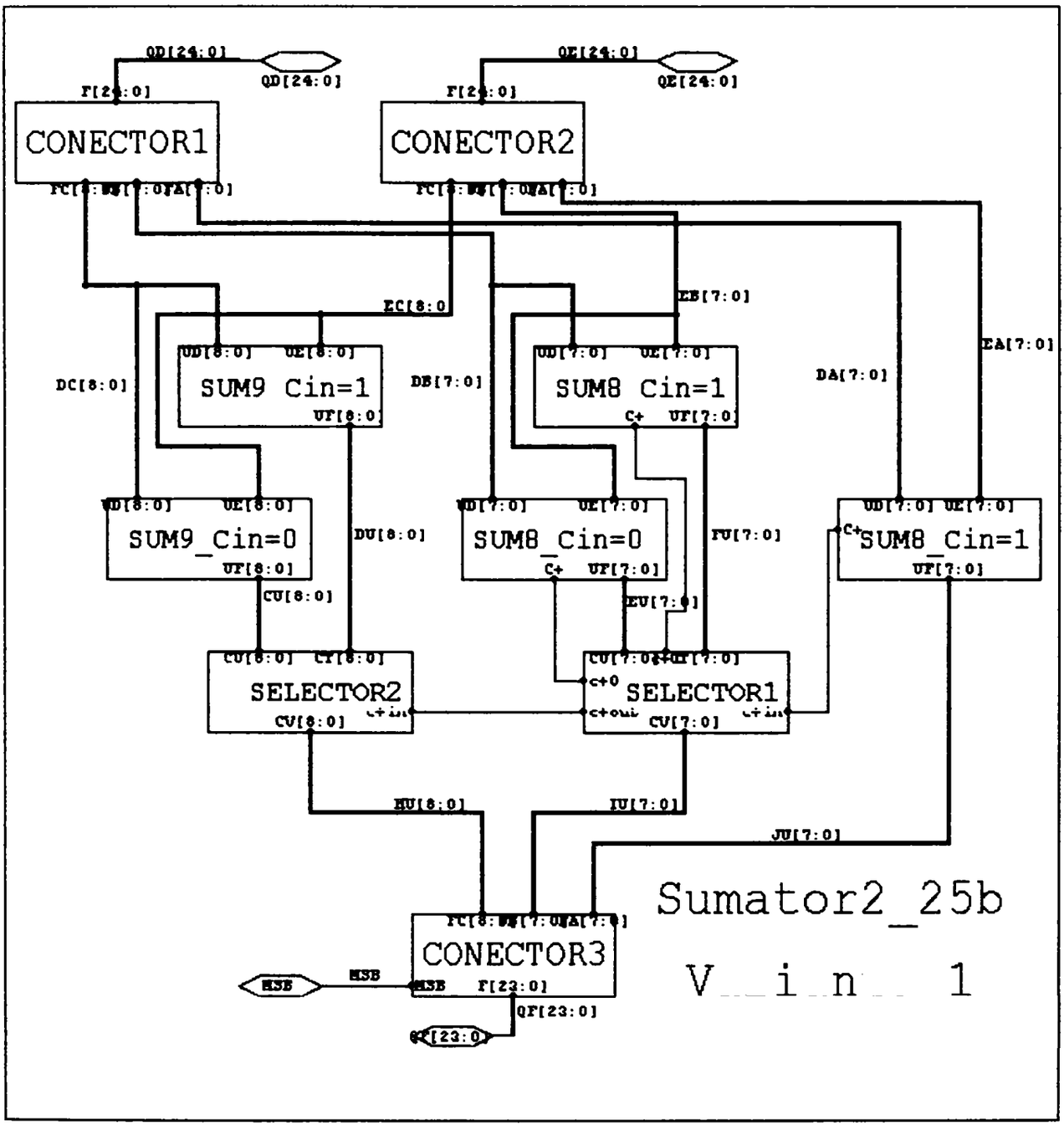

SUMATOR/SCAZATOR -NIVEL PIPELINE 2-
INCLUDE: C:\Msim52\aluMic.stm
UA[23:0]
UA[23:0]
UB[23:0]
UB[23:0]
UB[23:0]
Inv/NeinvA
SelAB
SelA
Q[24:0]
QQ[24:0]
UB[23:0]
Inv/NeinvB
SelAB
SelB
Q[24:0]
VE[24:0]
UD[24:0]
UE[24:0]
Sumator1_25b
UF[23:0]
SelA
SelB
Sa
Sa
Bloc
Sb
Sb
QD[24:0]
QE[24:0]
MSB
Sumator2_25b
QF[23:0]
L[23:0]
MSB
Sop
Sop
logic
Srez
SelS1
SelRez
UB[23:0]
Inv/NeinvS1
SelS1
Q[23:0]
N[23:0]
M[23:0]
QF[23:0]
UG[23:0]
Selector
SelRez
UH[23:0]
SelRez
UC[23:0]
UC[23:0]
Srez

ANEXA 6

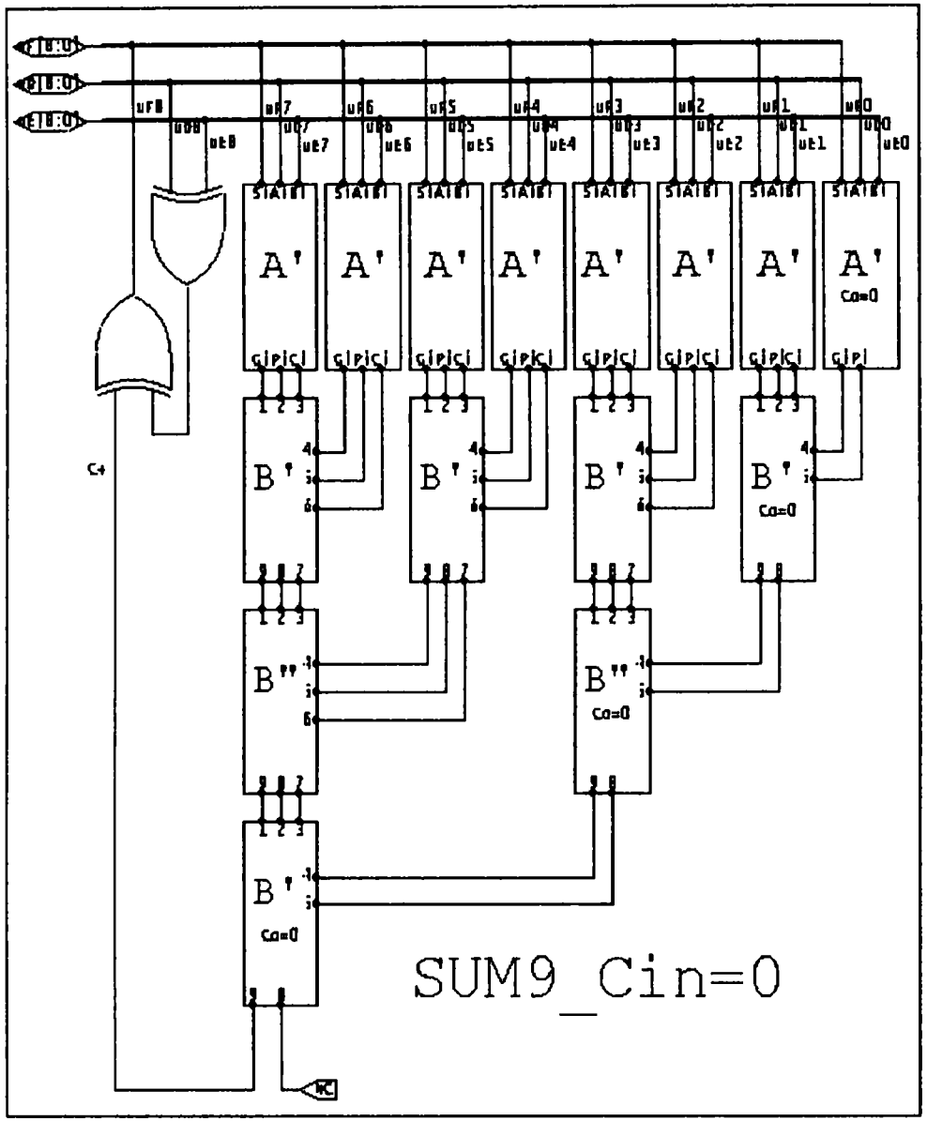


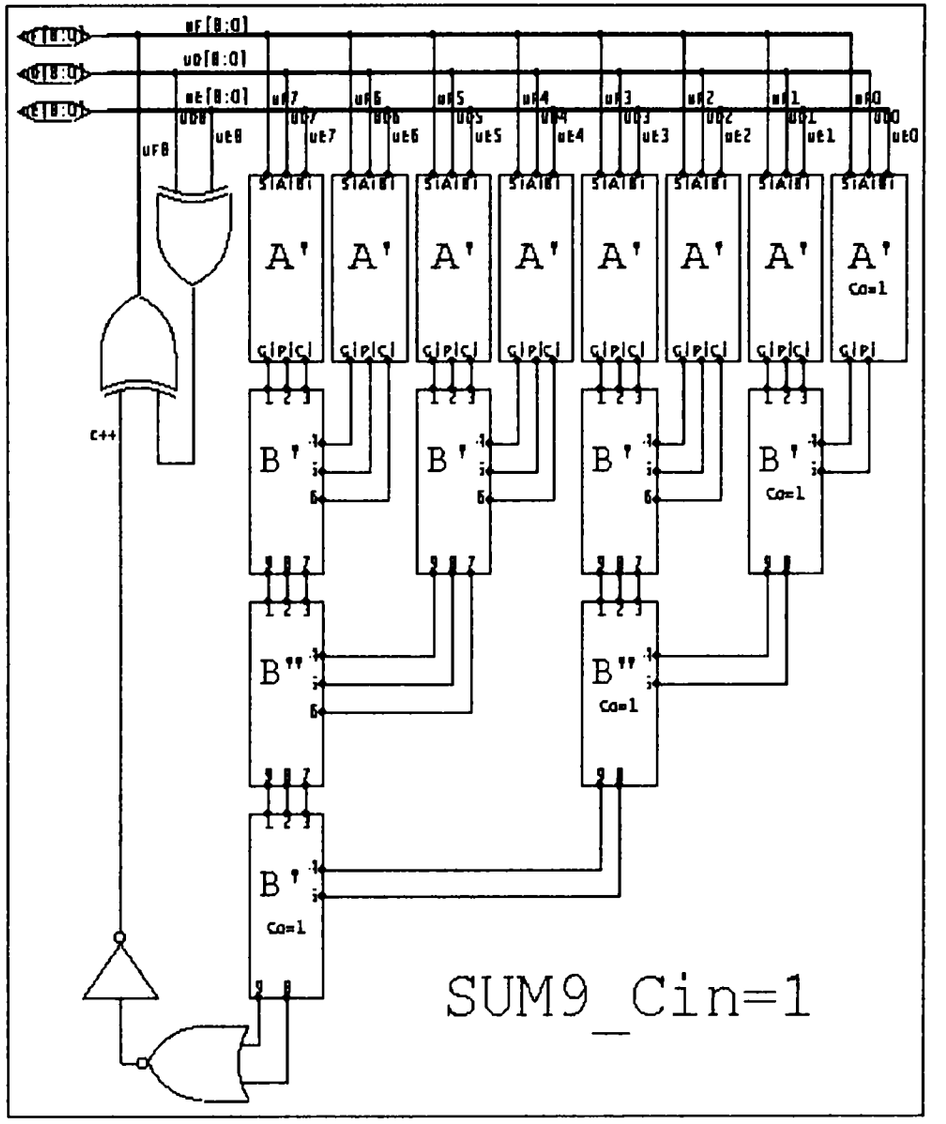
A6.2


UD[24:0]
UE[24:0]
CONECTOR1
CONECTOR2
EC[8:0]
EB[7:0]
EA[7:0]
DC[8:0]
DB[7:0]
DA[7:0]
SUM9 Cin=1
SUM8 Cin=1
SUM9_Cin=0
SUM8_Cin=0
SUM8_Cin=0
DU[8:0]
FU[7:0]
CU[8:0]
EU[7:0]
SELECTOR2
SELECTOR1
GU[8:0]
IU[7:0]
JU[7:0]
CONECTOR3
MSB
F[23:0]
Sumator1_25b
Varianta 1

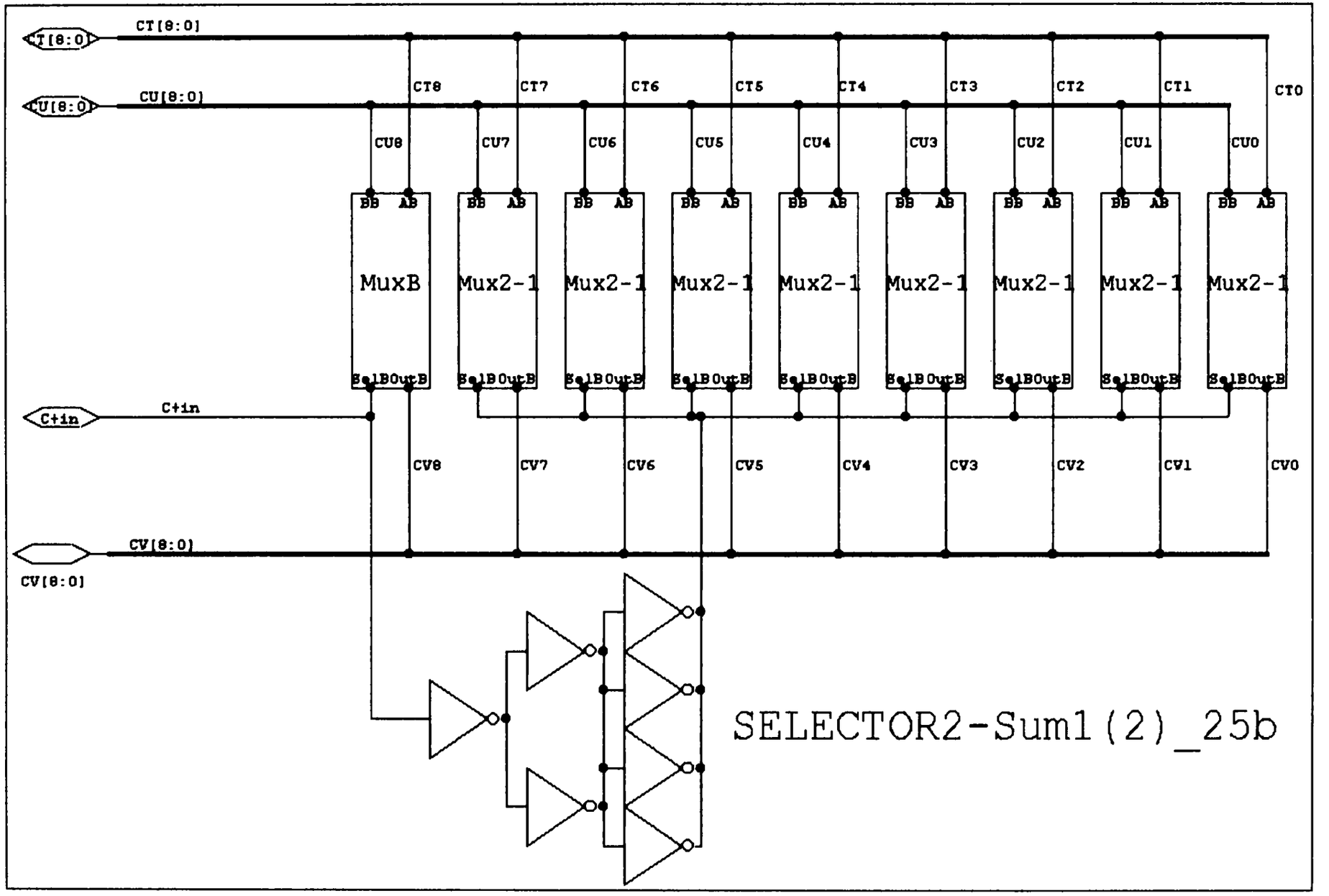

QD[24:0]
QD[24:0]
QE[24:0]
QE[24:0]
CONECTOR1
CONECTOR2
EC[8:0
EB[7:0]
EA[7:0]
DC[8:0]
DB[7:0]
DA[7:0]
SUM9 Cin=1
SUM8 Cin=1
SUM9_Cin=0
SUM8_Cin=0
SUM8_Cin=1
DU[8:0]
FU[7:0]
EU[7:0]
CU[8:0]
SELECTOR2
SELECTOR1
BU[8:0]
IU[7:0]
JU[7:0]
CONECTOR3
MSB
MSB
QF[23:0]
Sumator2_25b
V i n 1

ANEXA 6

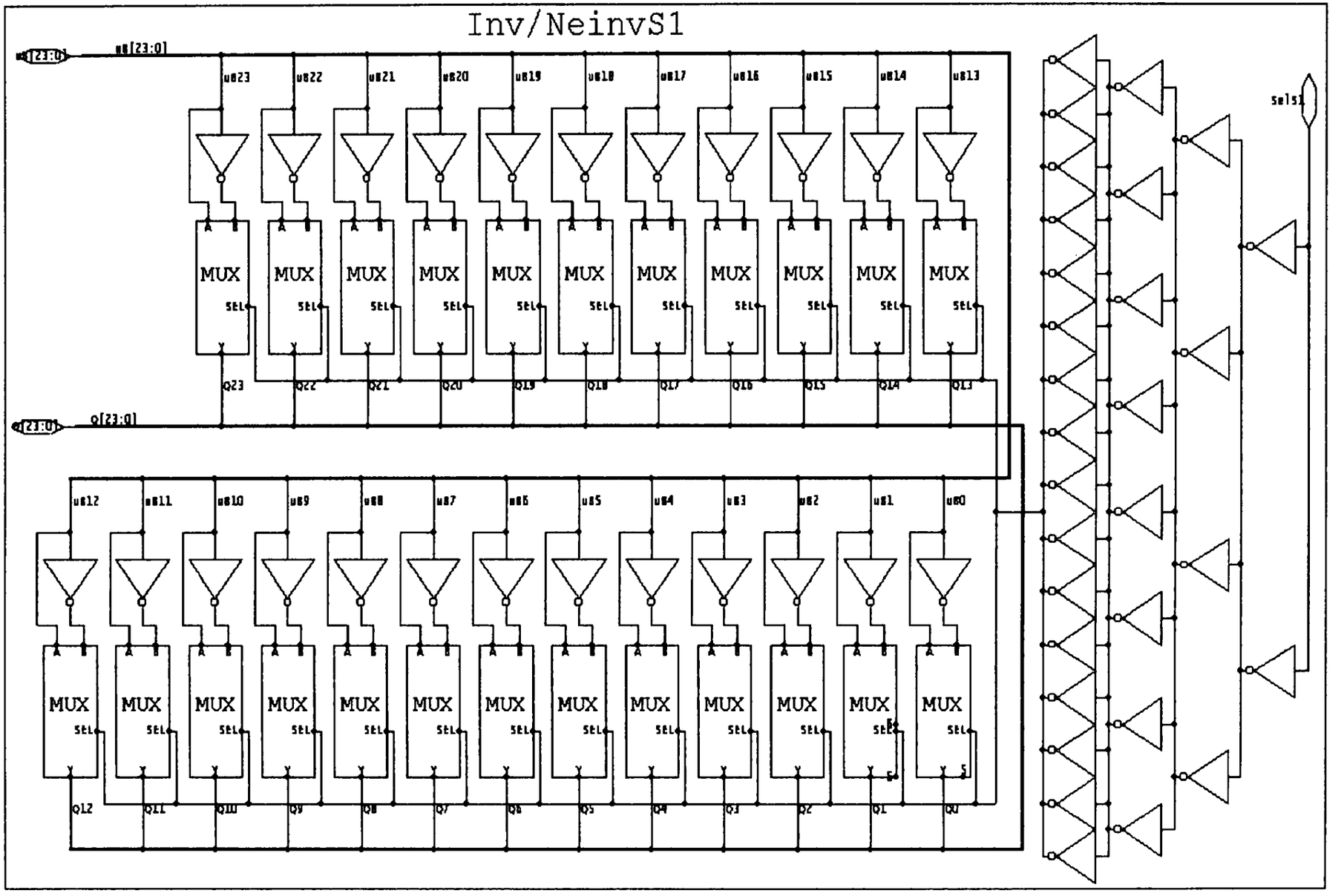




ANEXA 6

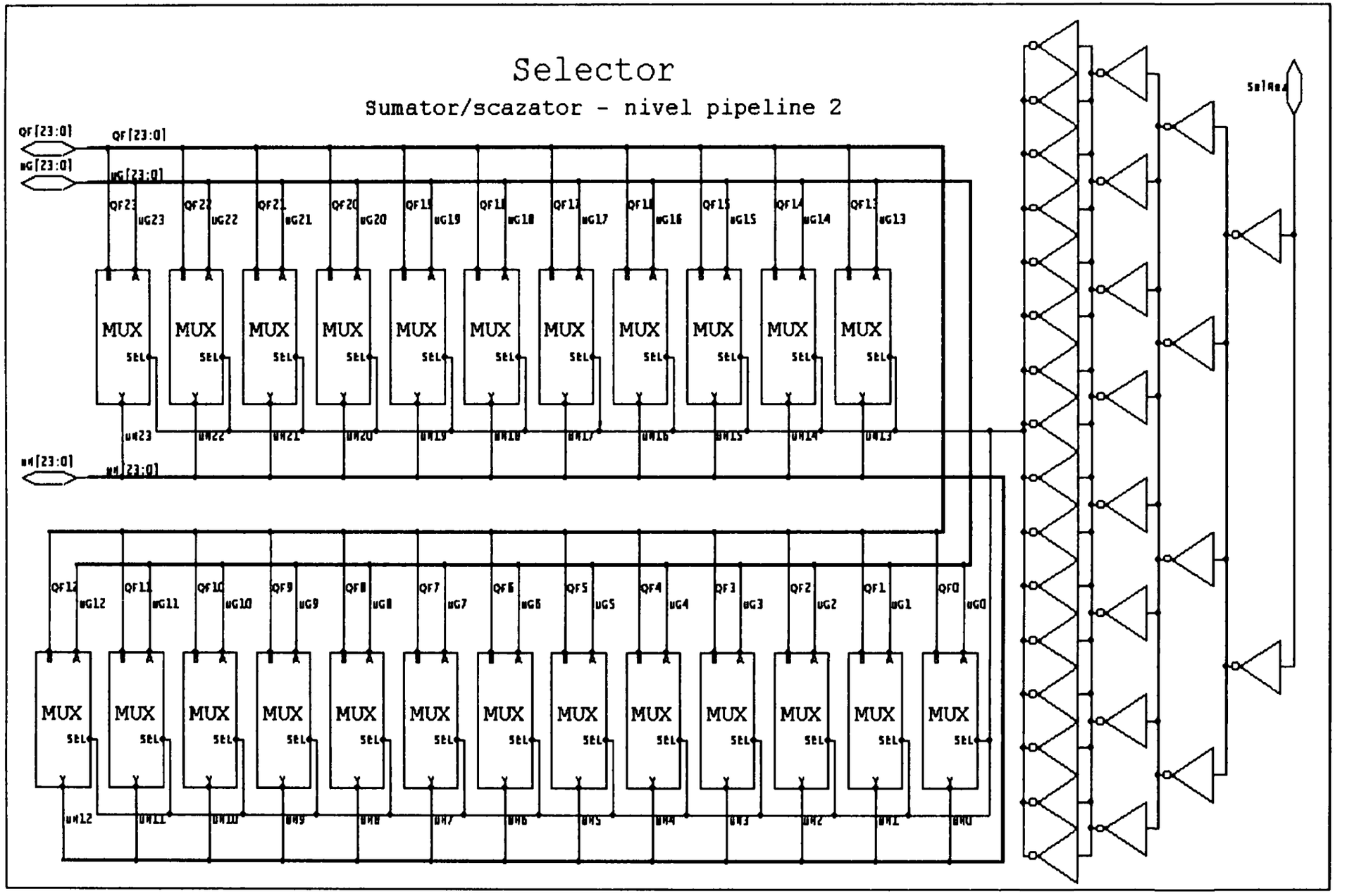

CT[8:0]
CU[8:0]
CT8
CT7
CT6
CT5
CT4
CT3
CT2
CT1
CT0
CU8
CU7
CU6
CU5
CU4
CU3
CU2
CU1
CU0
BB
AB
MuxB
Mux2-1
SelBOutB
C+in
CV8
CV7
CV6
CV5
CV4
CV3
CV2
CV1
CV0
CV[8:0]
SELECTOR2-Sum1(2)_25b

ANEXA 6

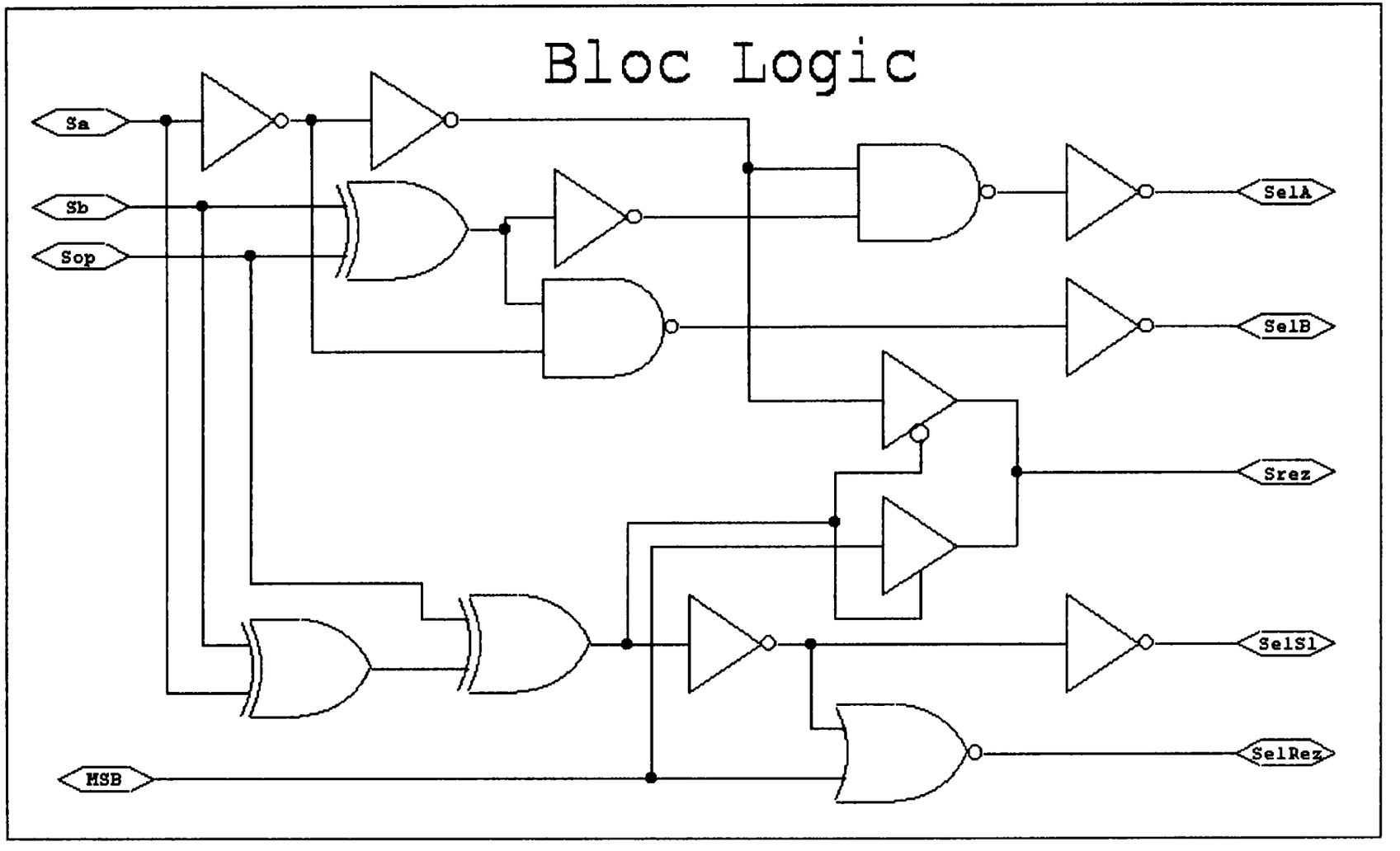


A6.7

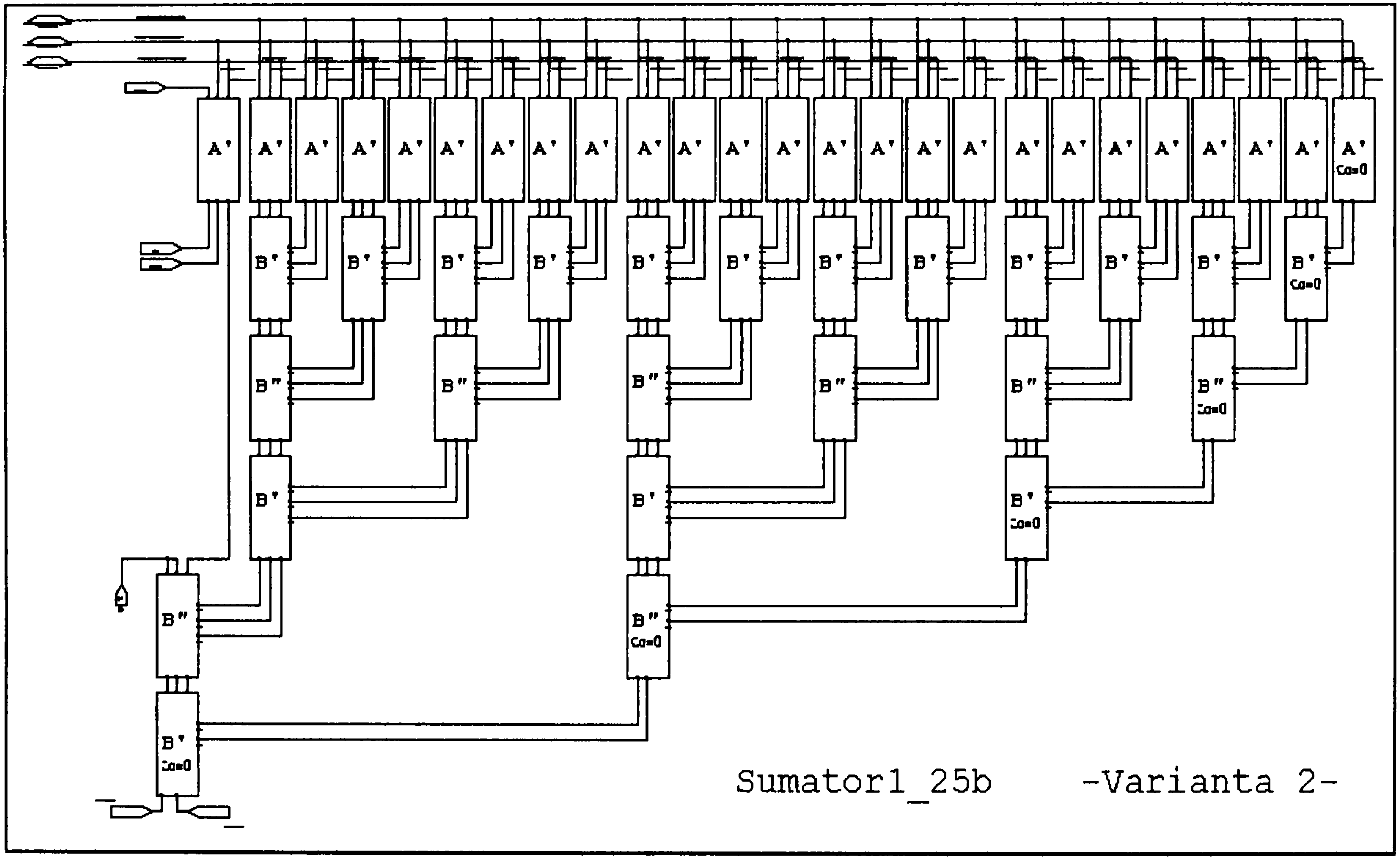
ANEXA 6


Selector
Sumator/scazator - nivel pipeline 2
QF[23:0]
uG[23:0]
u4[23:0]
MUX
SEL

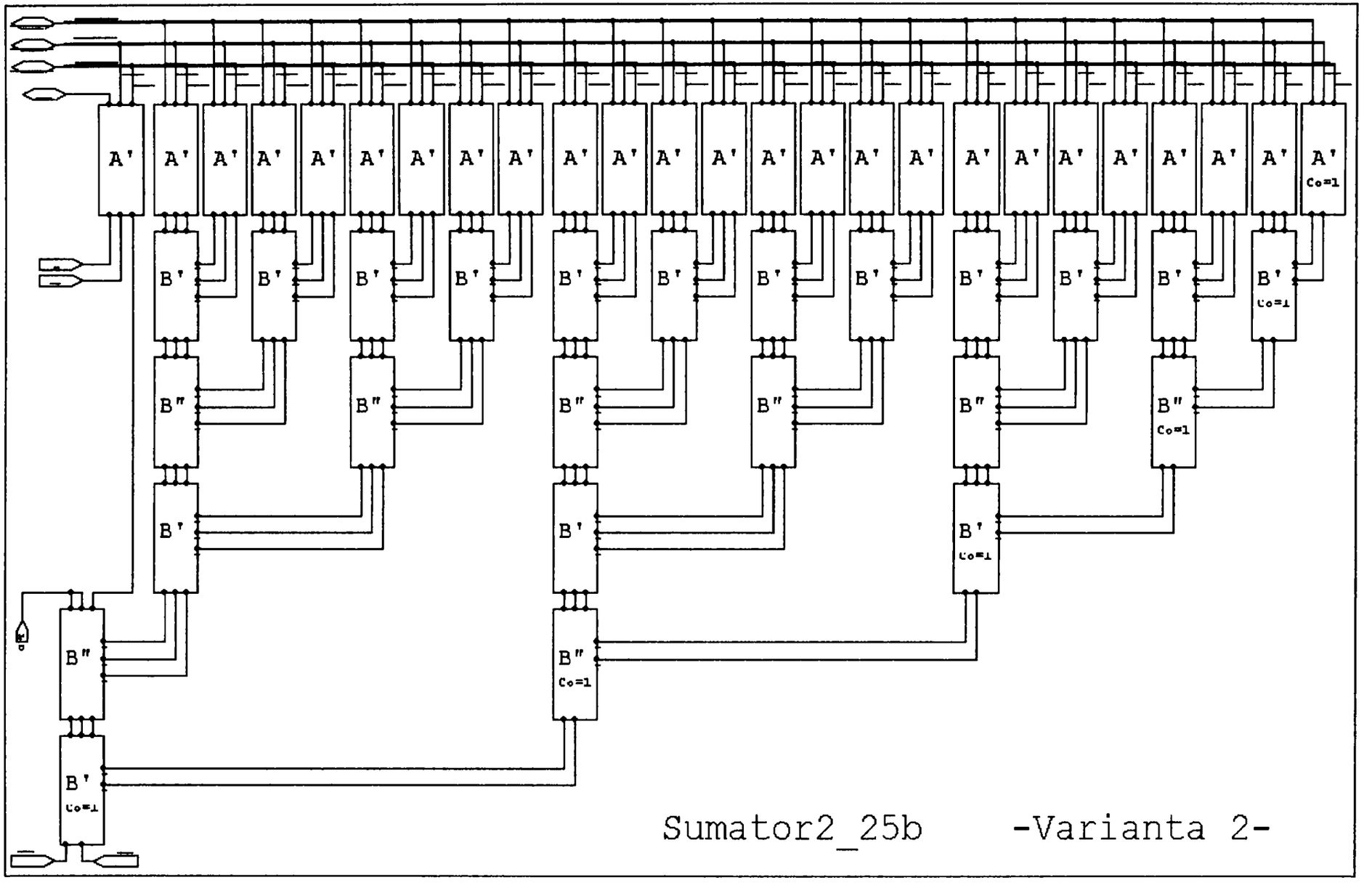
ANEXA 6

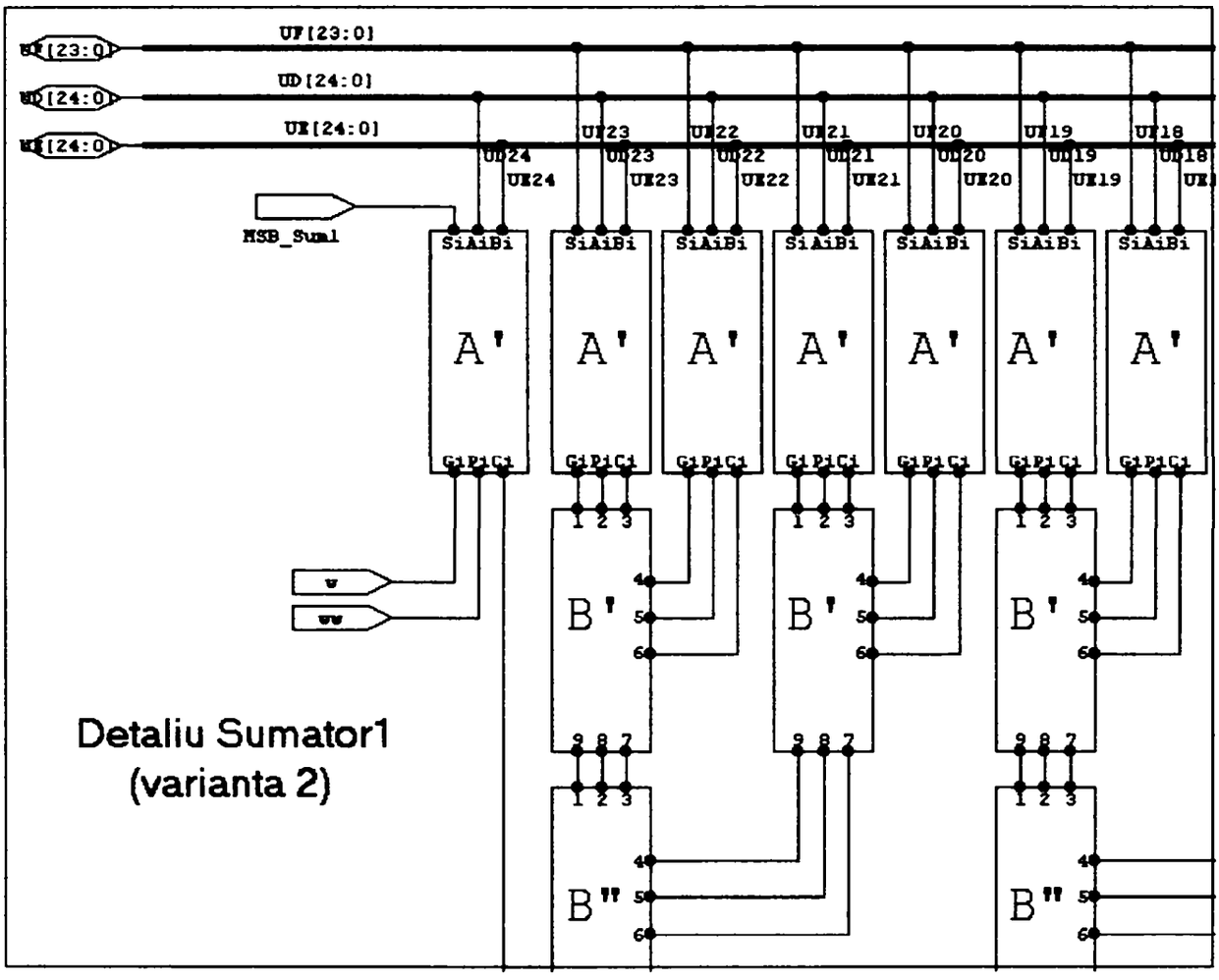


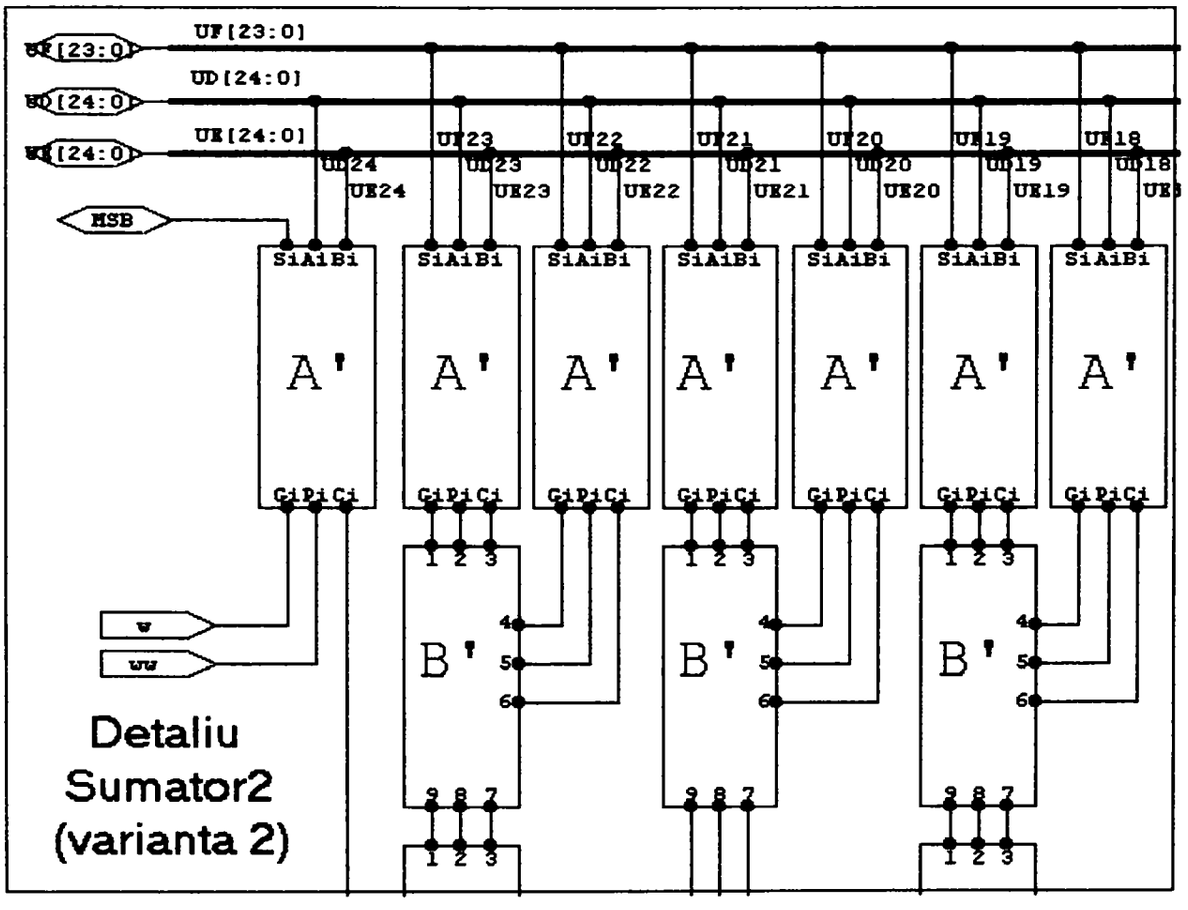
ANEXA 6



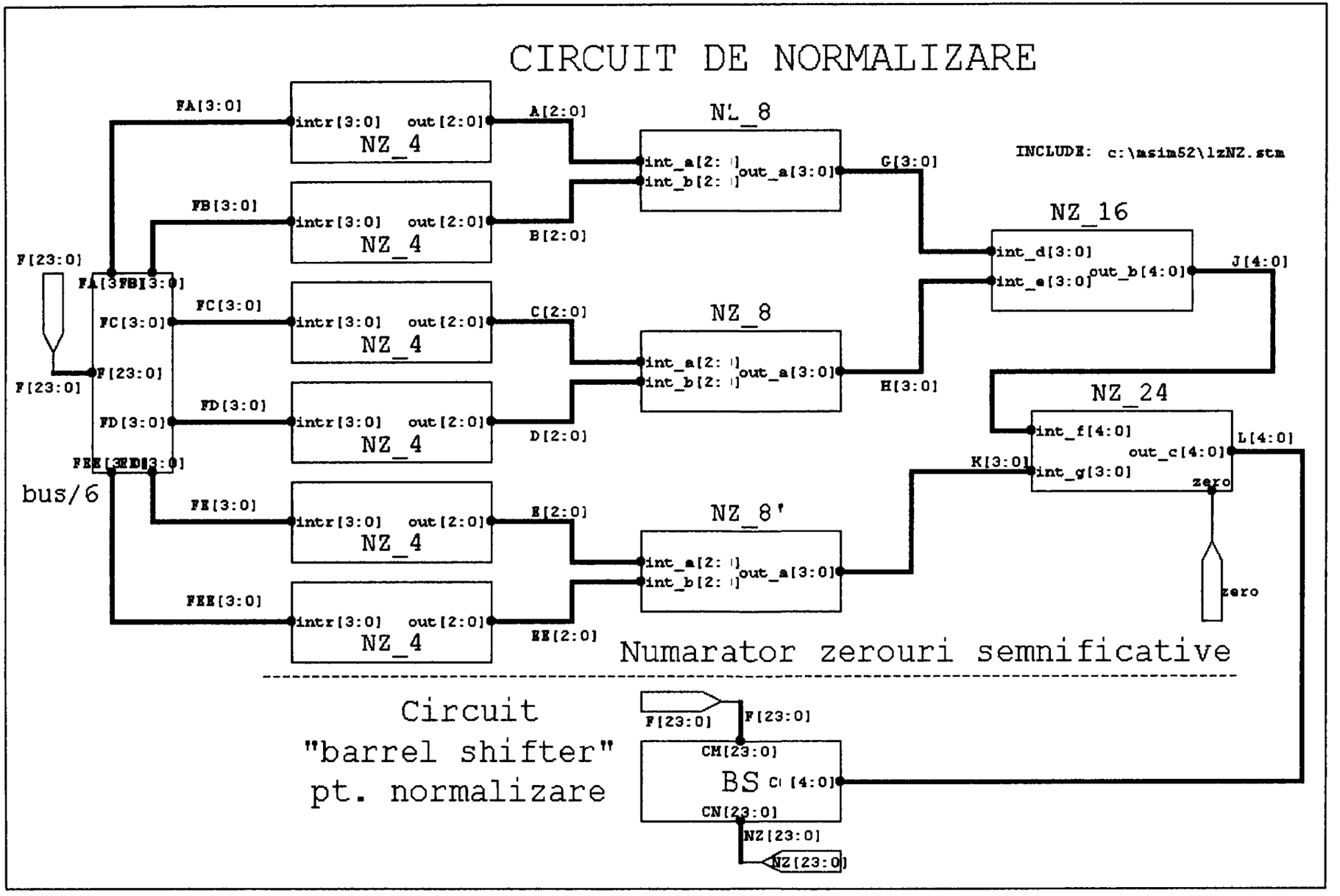
ANEXA 6



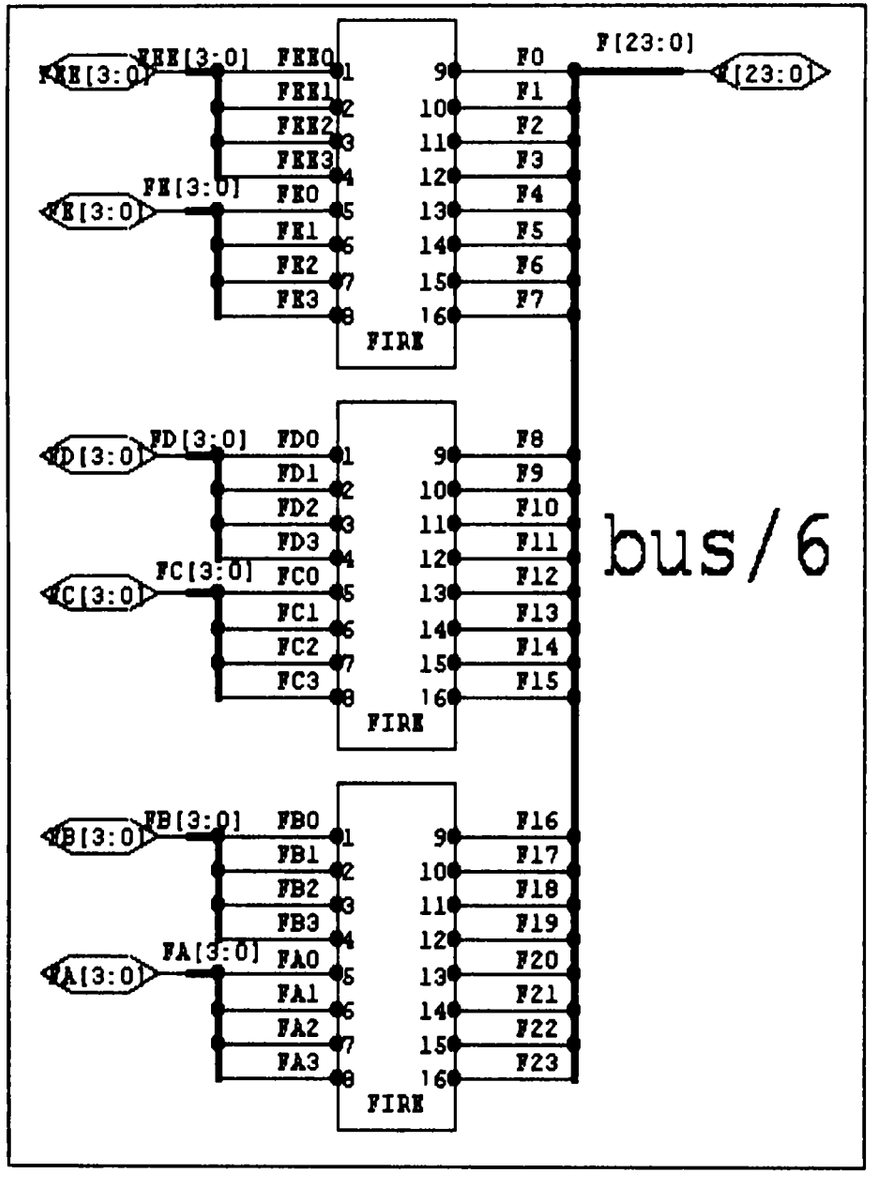

UF[23:0]
UD[24:0]
UE[24:0]
MSB_Sum1
SiAiBi
A'
GiPiCi
B'
B''
Detaliu Sumator1
(varianta 2)

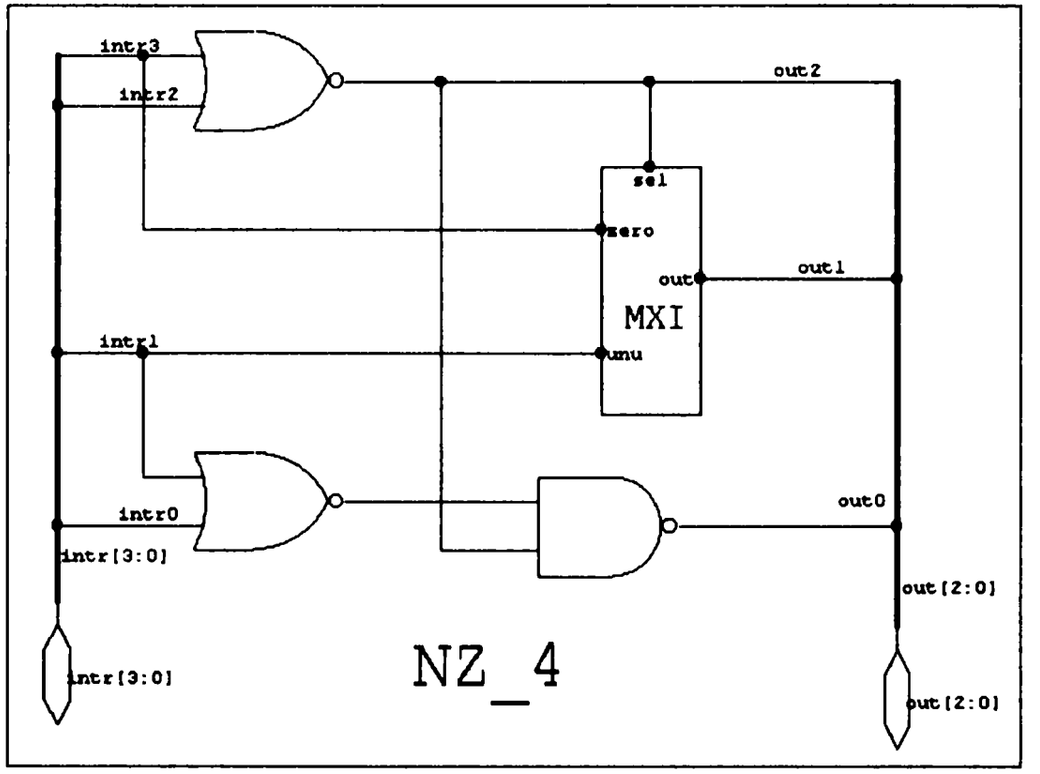

UF[23:0]
UD[24:0]
UE[24:0]
MSB
SiAiBi
A'
GiPiCi
B'
Detaliu
Sumator2
(varianta 2)

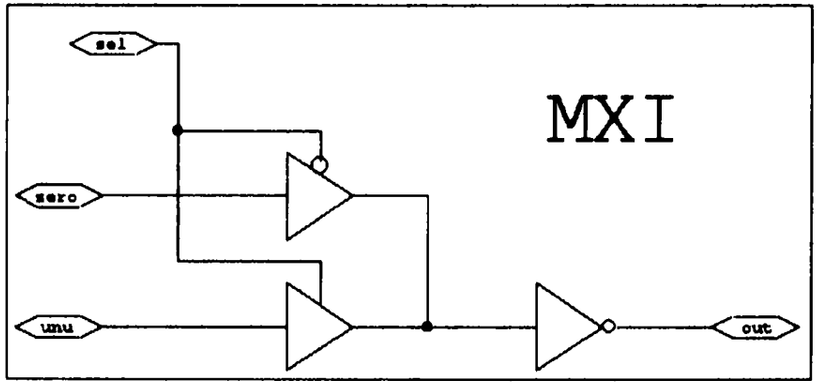
ANEXA 7



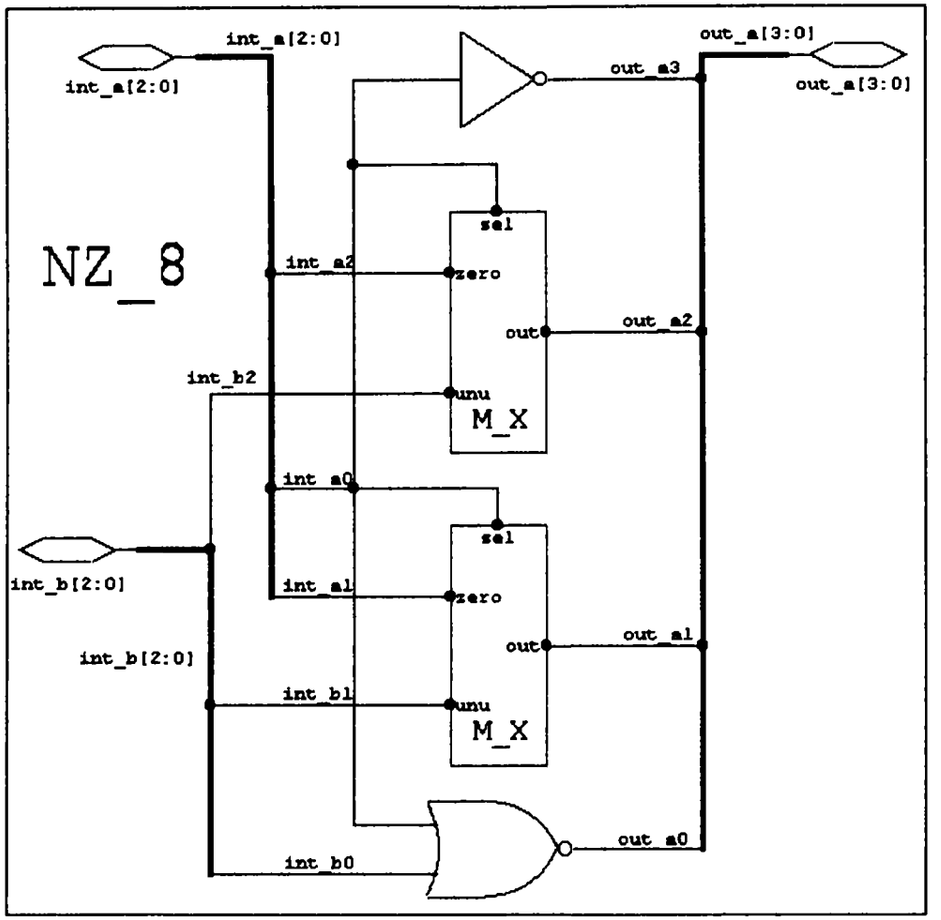
A7.1

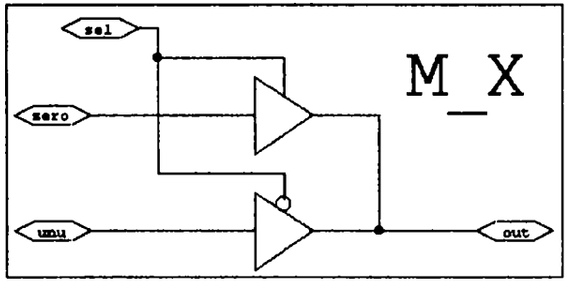

FEE[3:0]
FE[3:0]
FD[3:0]
FC[3:0]
FB[3:0]
FA[3:0]
FIRE
F[23:0]
bus/6

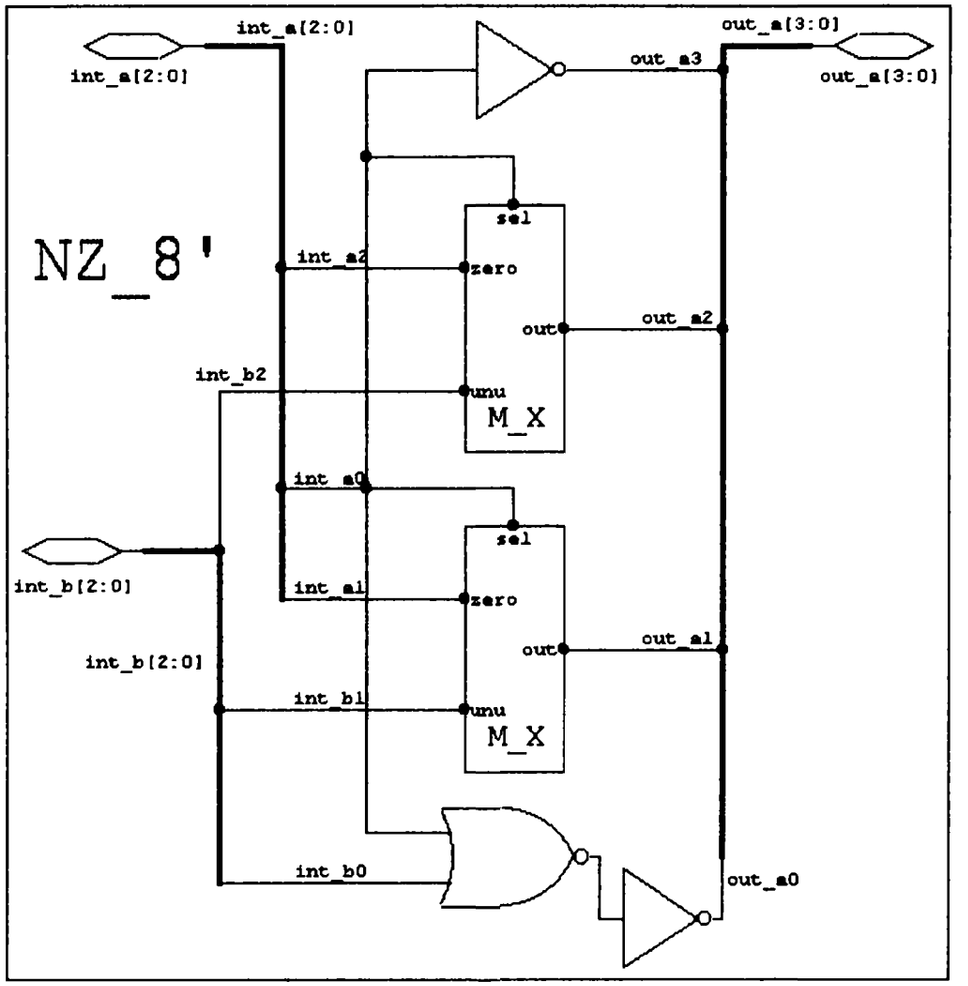

intr3
intr2
intr1
intr0
intr[3:0]
sel
zero
unu
out
MXI
out2
out1
out0
out[2:0]
NZ_4

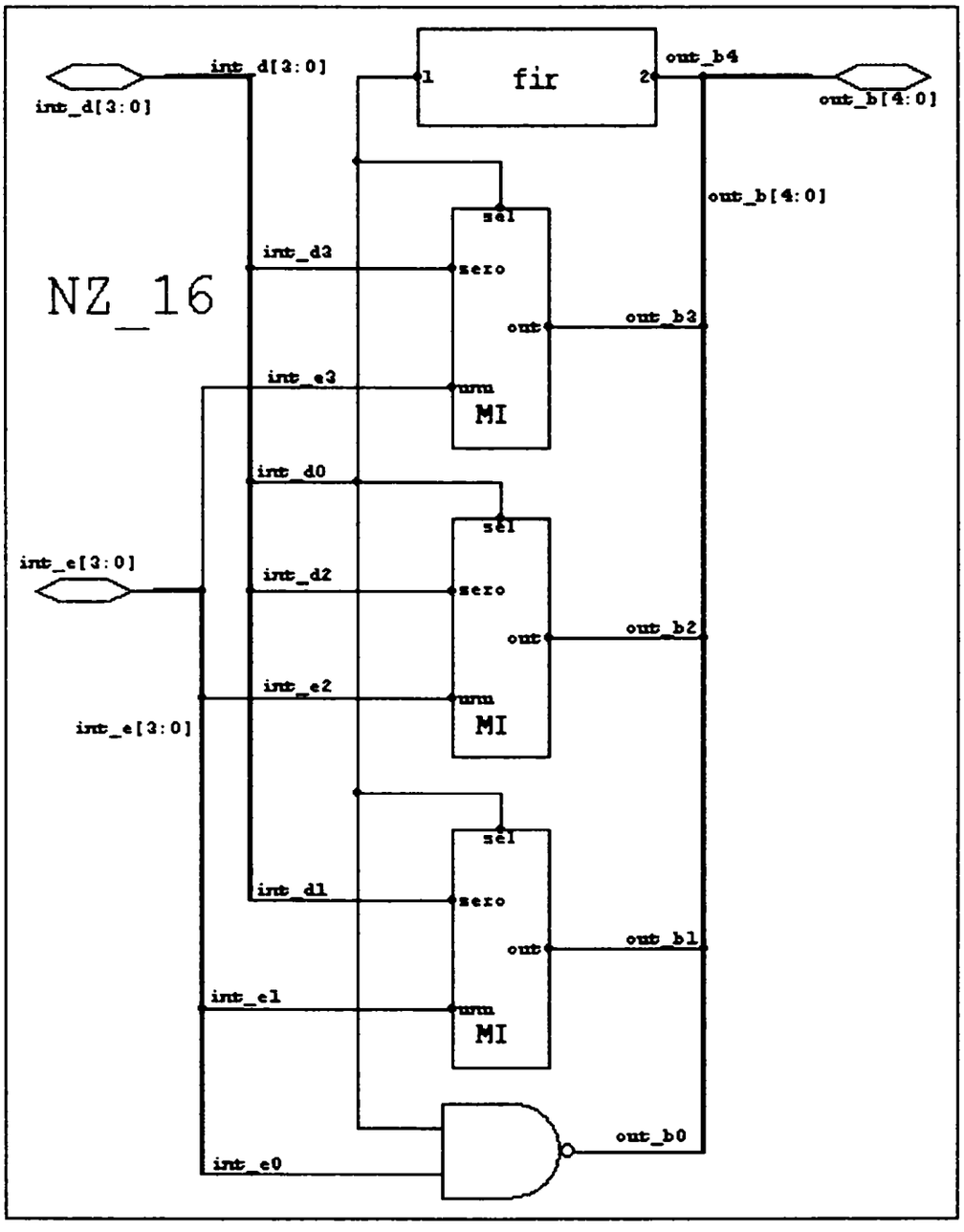

sel
zero
unu
out
MXI

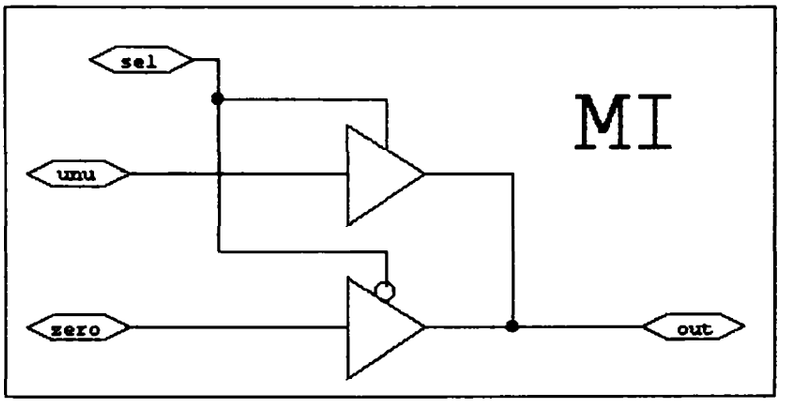
ANEXA 7

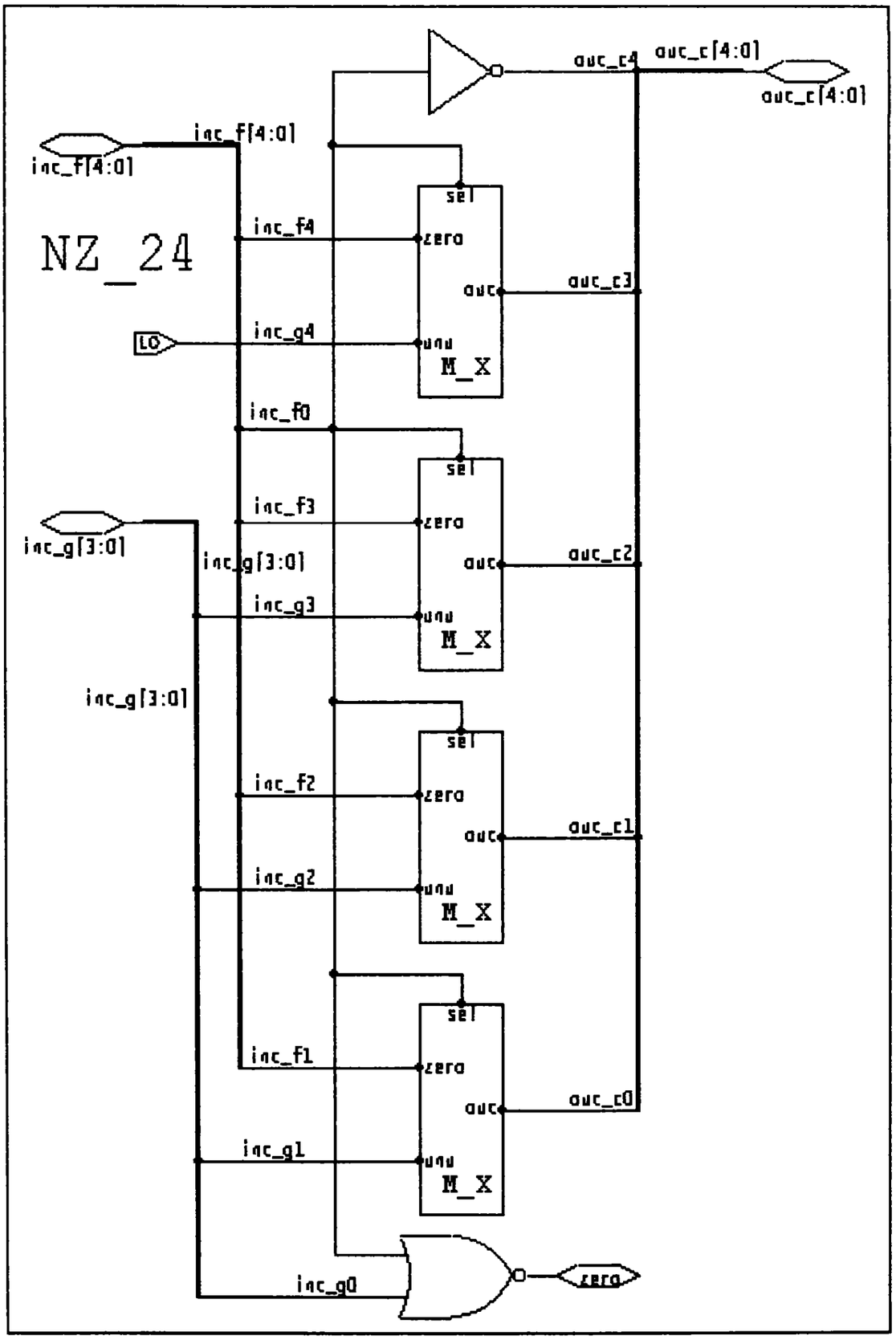

NZ_8
int_a[2:0]
int_a[2:0]
out_a[3:0]
out_a[3:0]
out_a3
sel
int_a2
zero
out
out_a2
int_b2
unu
M_X
int_a0
sel
int_b[2:0]
int_a1
zero
out
out_a1
int_b[2:0]
int_b1
unu
M_X
out_a0
int_b0

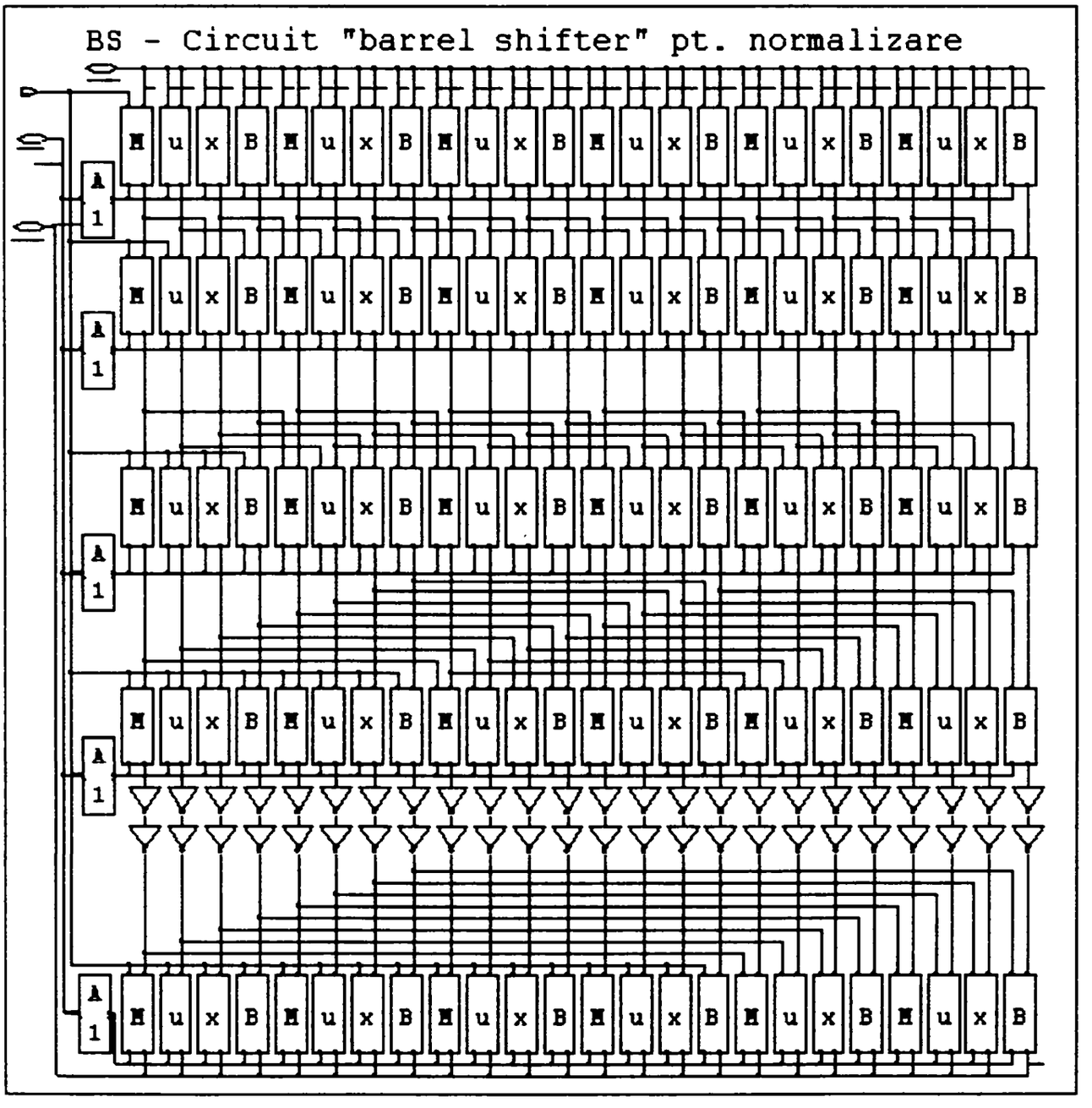

sel
M_X
zero
unu
out

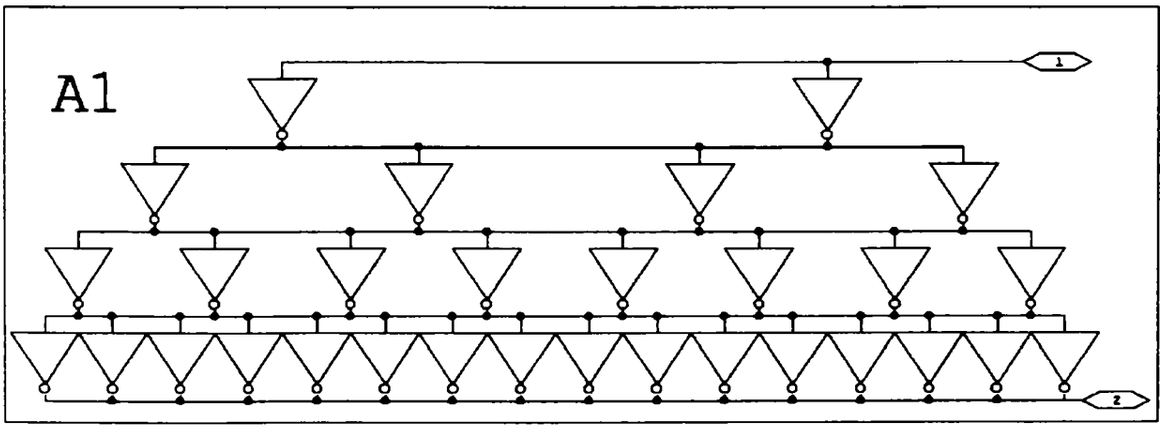

NZ_8'
int_a[2:0]
int_a[2:0]
out_a[3:0]
out_a[3:0]
out_a3
sel
int_a2
zero
out
out_a2
int_b2
unu
M_X
int_a0
sel
int_b[2:0]
int_a1
zero
out
out_a1
int_b[2:0]
int_b1
unu
M_X
int_b0
out_a0

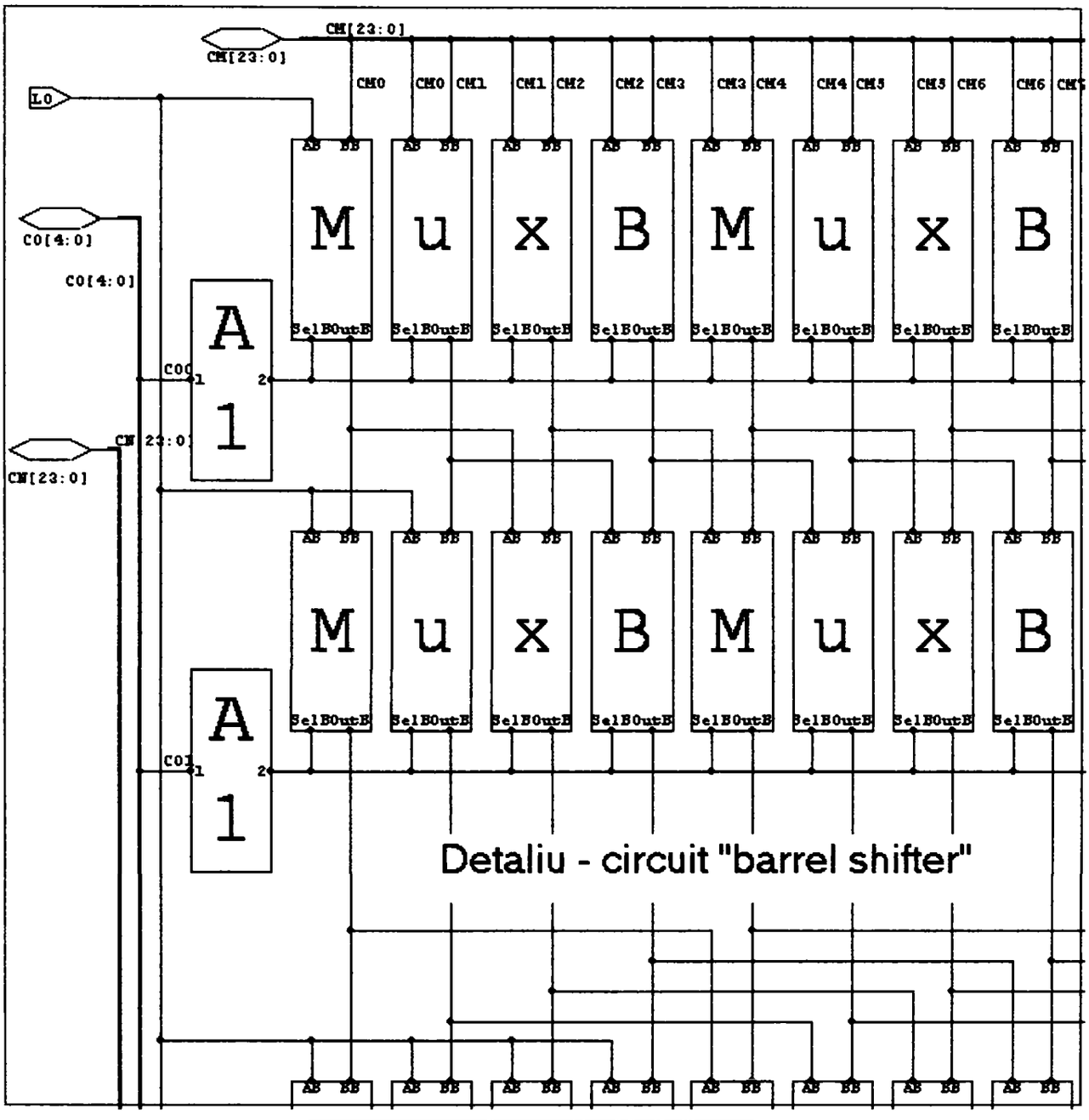

int_d[3:0]
int_d[3:0]
fir
1
2
out_b4
out_b[4:0]
out_b[4:0]
NZ_16
sel
int_d3
zero
out
out_b3
int_e3
unu
MI
int_d0
sel
int_e[3:0]
int_d2
zero
out
out_b2
int_e2
unu
int_e[3:0]
MI
sel
int_d1
zero
out
out_b1
int_e1
unu
MI
out_b0
int_e0

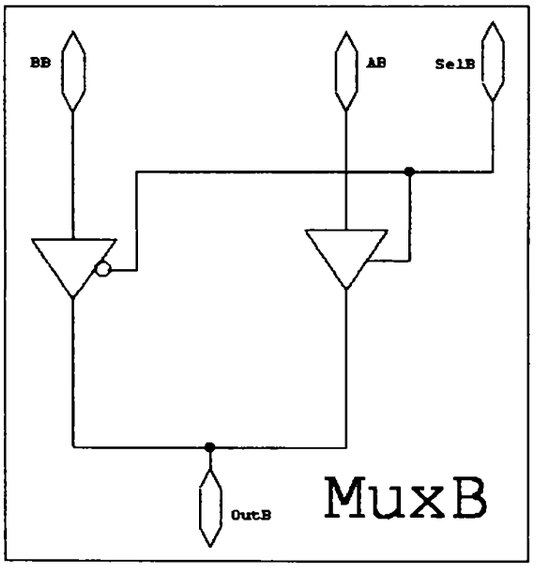

sel
MI
unu
zero
out


auc_c4
auc_c[4:0]
auc_c[4:0]
inc_f[4:0]
inc_f[4:0]
NZ_24
sel
inc_f4
zero
auc
auc_c3
LO
inc_g4
una
M_X
inc_f0
sel
inc_f3
zero
auc
auc_c2
inc_g[3:0]
inc_g[3:0]
inc_g3
una
M_X
inc_g[3:0]
sel
inc_f2
zero
auc
auc_c1
inc_g2
una
M_X
sel
inc_f1
zero
auc
auc_c0
inc_g1
una
M_X
inc_g0
zero





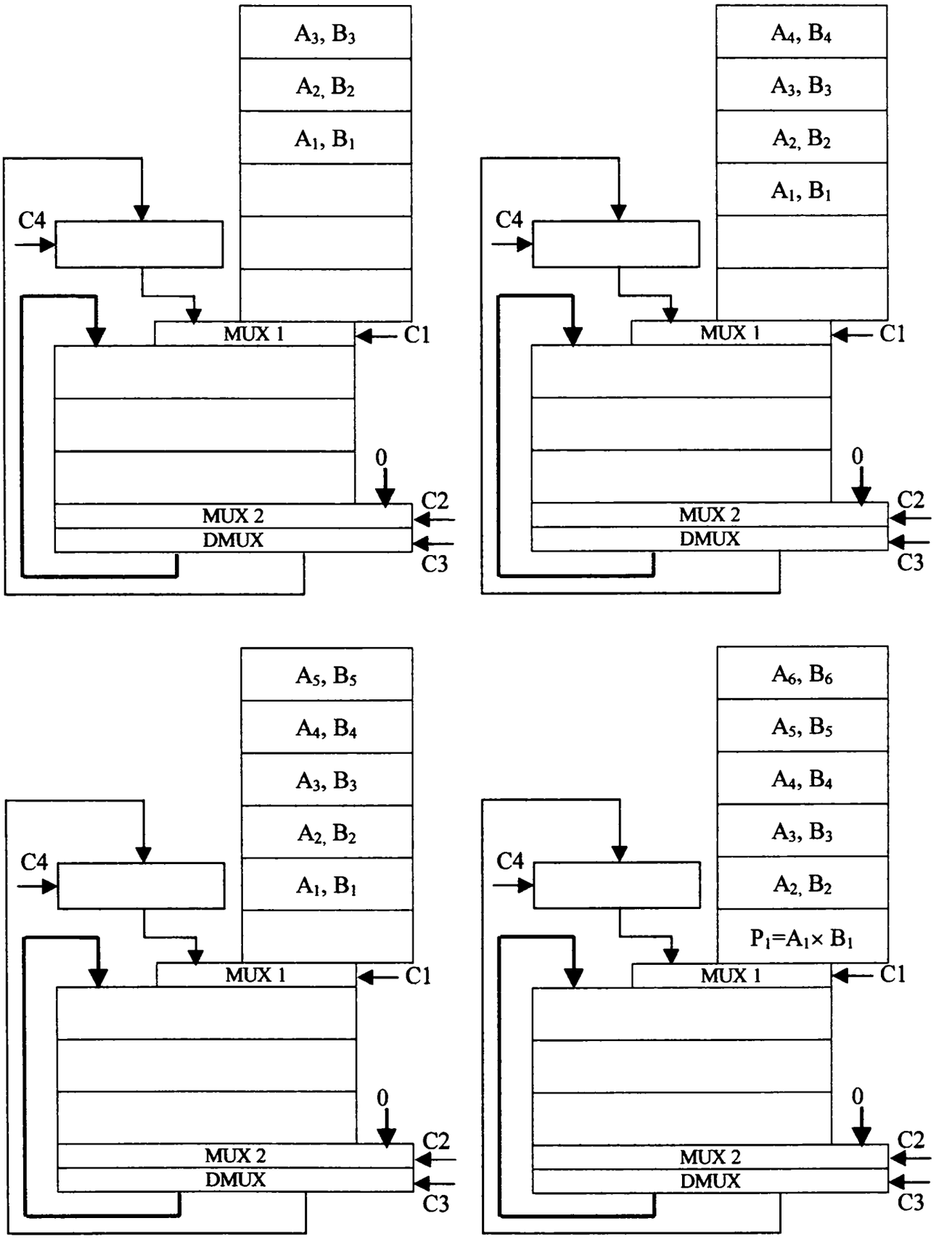

CM[23:0]
CM[23:0]
L0
CO[4:0]
CO[4:0]
CM[23:0]
CM0
CM1
CM2
CM3
CM4
CM5
CM6
AB
BB
M u x B
SelBOutB
A
1
2
Detaliu - circuit "barrel shifter"


BB
AB
SelB
OutB
MuxB

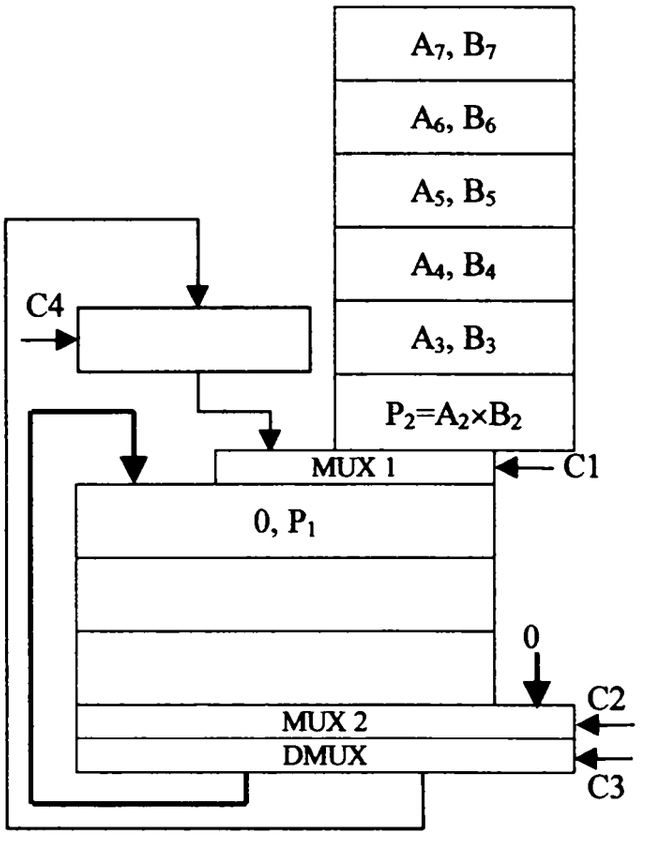
ANEXA 8

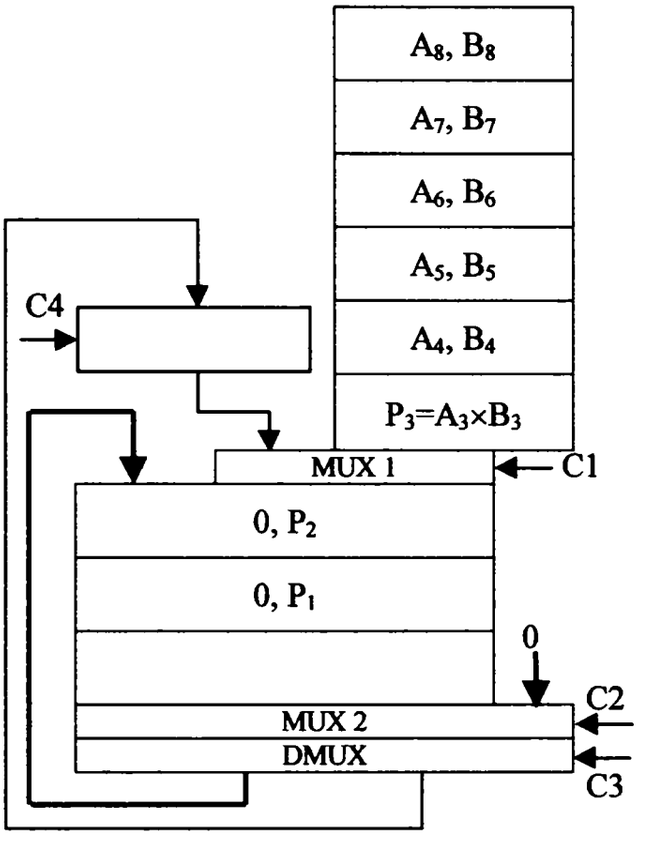
Blocurile MUX 1 şi respectiv MUX 2 şi DMUX se includ în nivelurile pipeline 6 din subunitatea logaritmică şi respectiv 3 din subunitatea în virgulă flotantă întrucât există rezerve de 0,4 ns în fiecare din ele. În acest exemplu procesorul aritmetic calculează următoarea expresie:

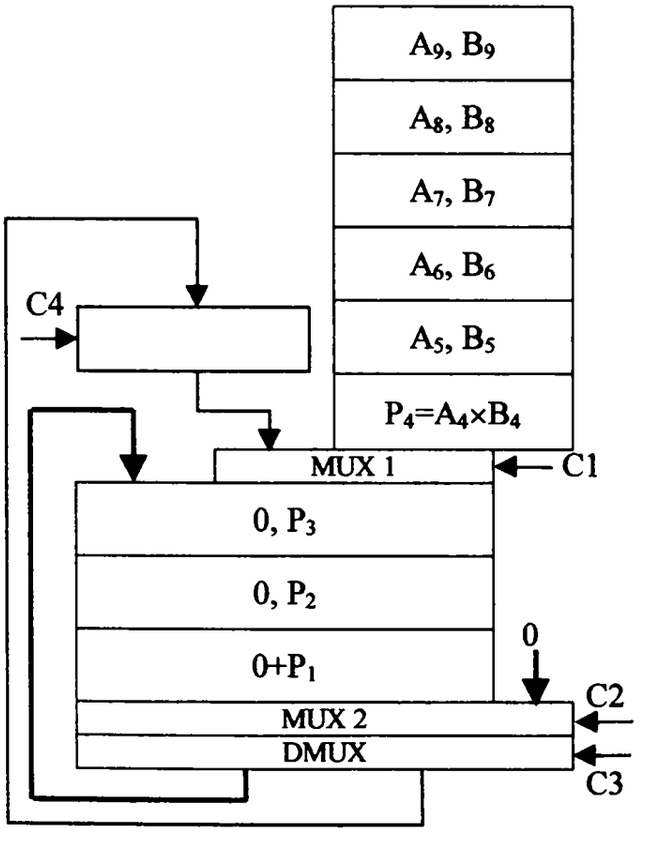
$$E = \sum_{i=1}^{20} A_i \times B_i = \sum_{i=1}^{20} P_i$$

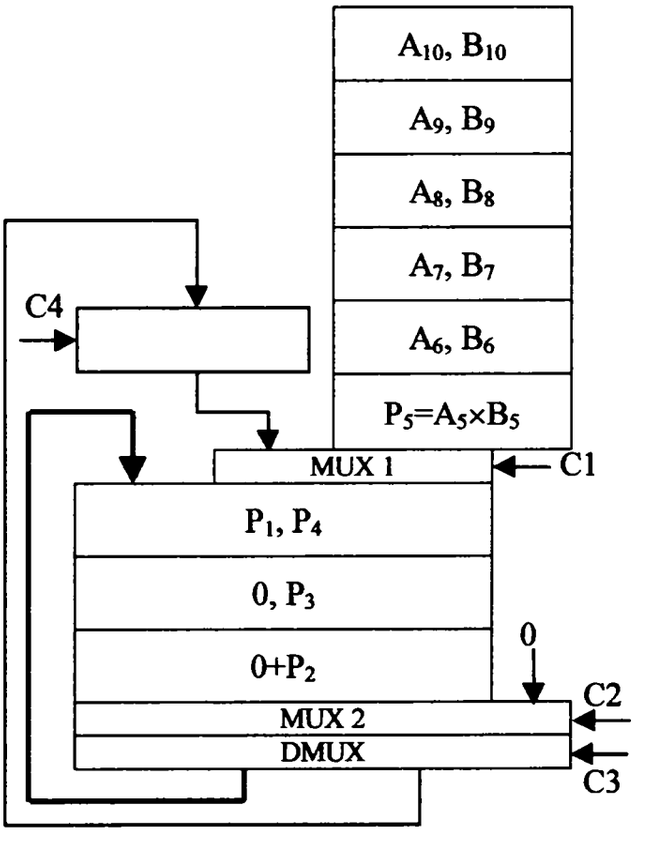


ANEXA 8

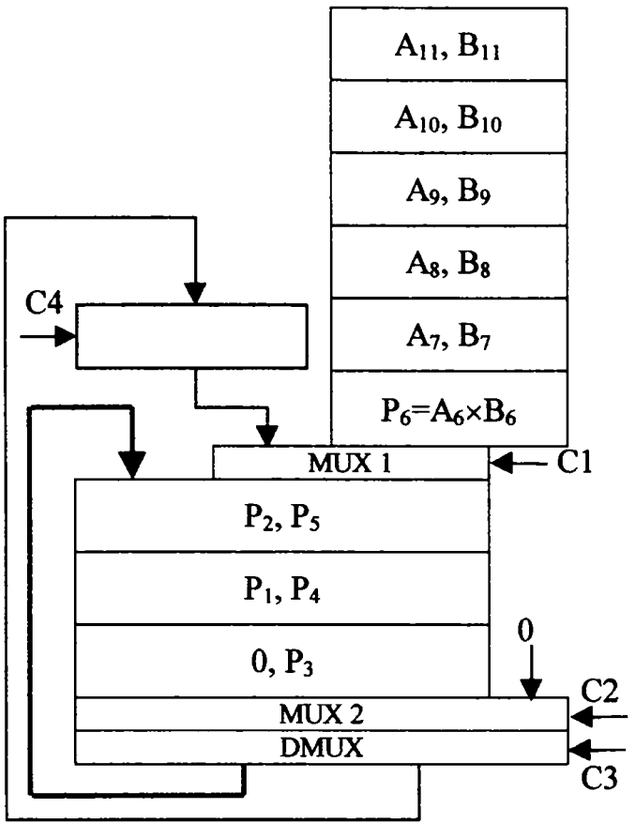

A7, B7
A6, B6
A5, B5
A4, B4
A3, B3
P2=A2×B2
C4
MUX 1
C1
0, P1
0
C2
MUX 2
DMUX
C3

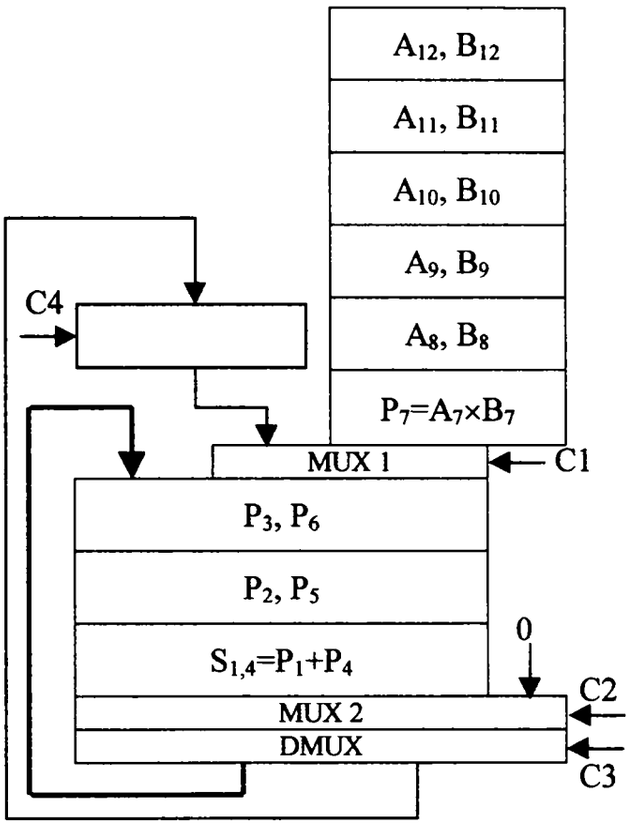

A8, B8
A7, B7
A6, B6
A5, B5
A4, B4
P3=A3×B3
C4
MUX 1
C1
0, P2
0, P1
0
C2
MUX 2
DMUX
C3

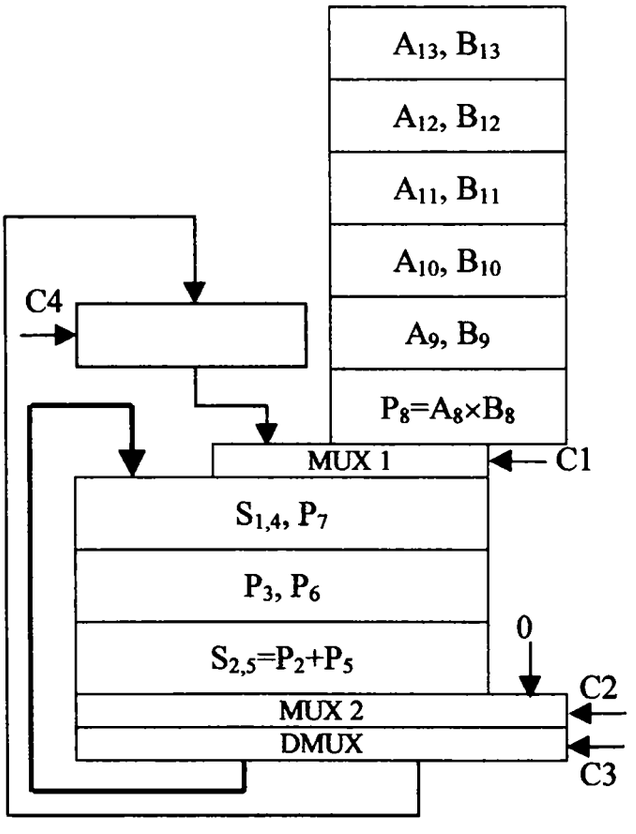

A9, B9
A8, B8
A7, B7
A6, B6
A5, B5
P4=A4×B4
C4
MUX 1
C1
0, P3
0, P2
0+P1
0
C2
MUX 2
DMUX
C3

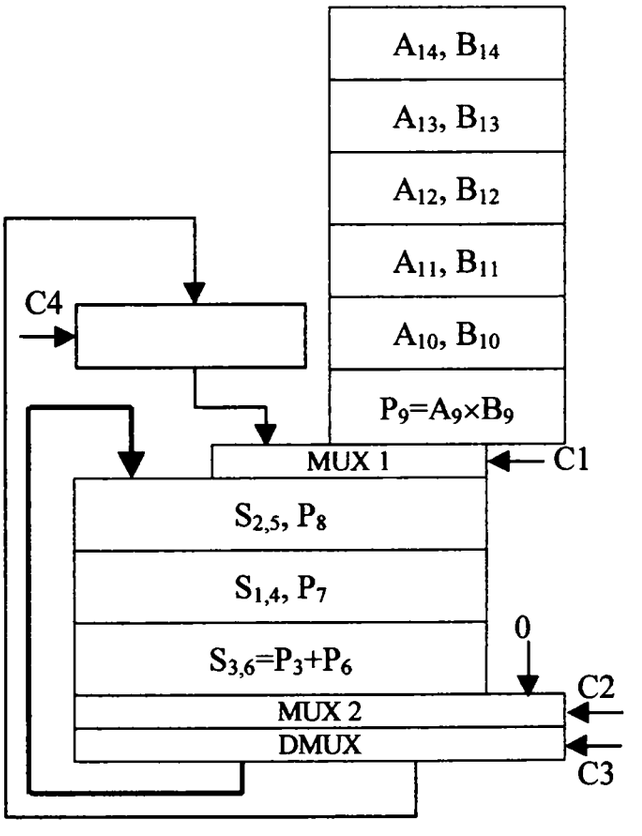

A10, B10
A9, B9
A8, B8
A7, B7
A6, B6
P5=A5×B5
C4
MUX 1
C1
P1, P4
0, P3
0+P2
0
C2
MUX 2
DMUX
C3

ANEXA 8

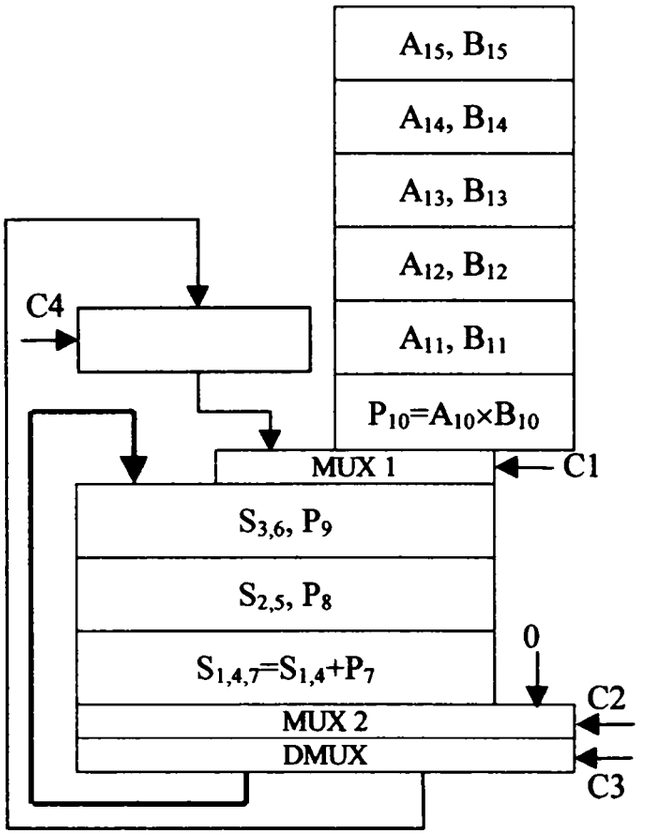

A11, B11
A10, B10
A9, B9
A8, B8
A7, B7
P6=A6×B6
C4
MUX 1
C1
P2, P5
P1, P4
0, P3
0
C2
MUX 2
DMUX
C3

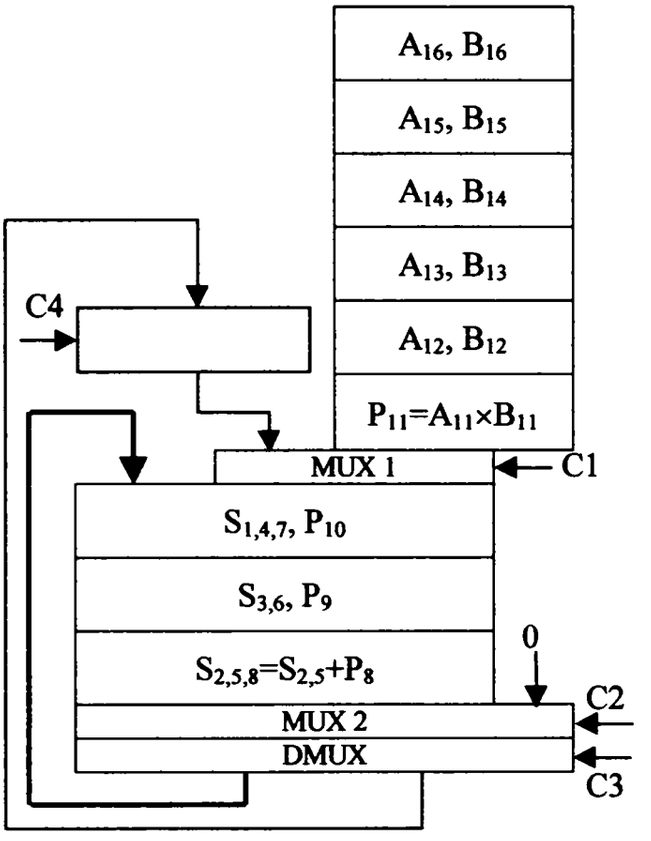

A12, B12
A11, B11
A10, B10
A9, B9
A8, B8
P7=A7×B7
C4
MUX 1
C1
P3, P6
P2, P5
S1,4=P1+P4
0
C2
MUX 2
DMUX
C3

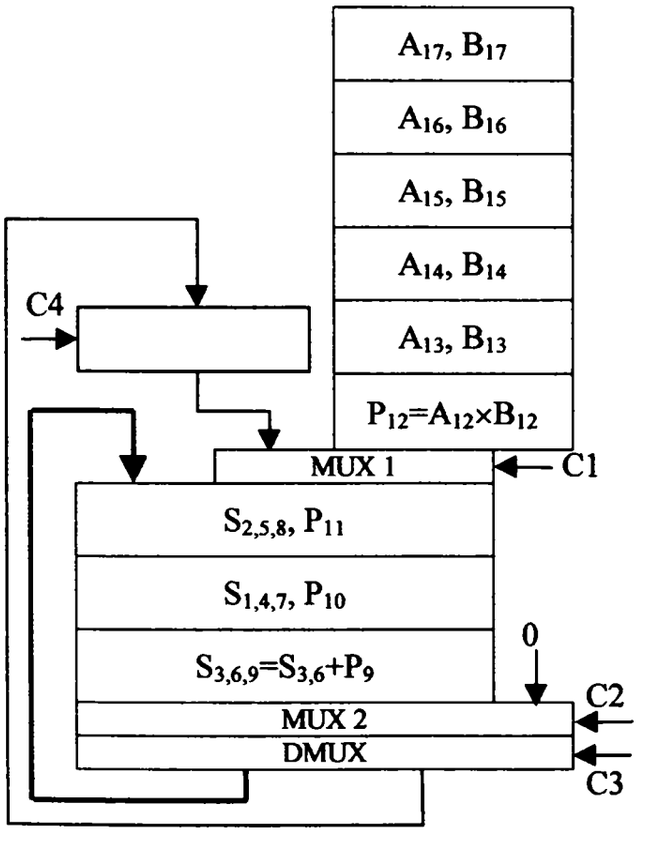

A13, B13
A12, B12
A11, B11
A10, B10
A9, B9
P8=A8×B8
C4
MUX 1
C1
S1,4, P7
P3, P6
S2,5=P2+P5
0
C2
MUX 2
DMUX
C3

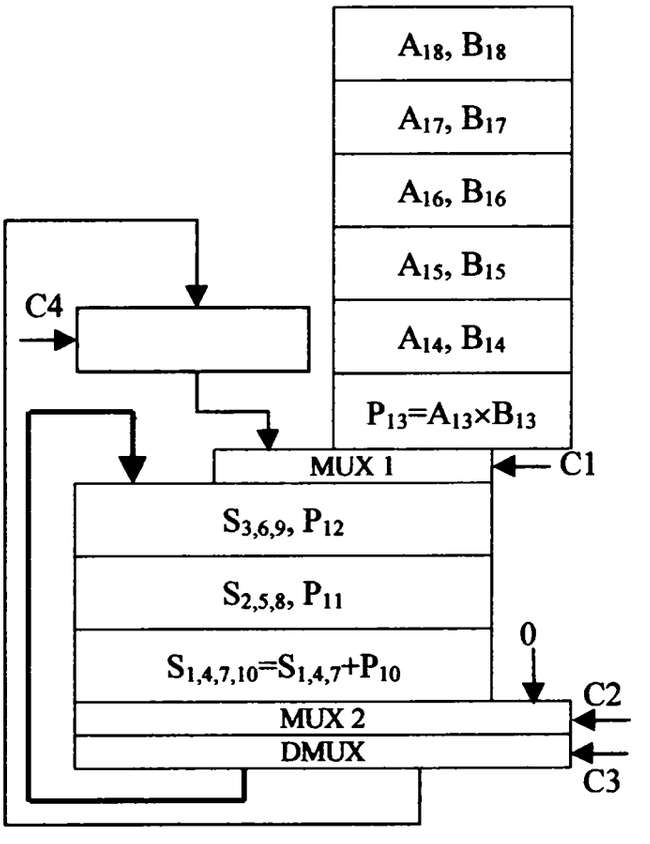

A14, B14
A13, B13
A12, B12
A11, B11
A10, B10
P9=A9×B9
C4
MUX 1
C1
S2,5, P8
S1,4, P7
S3,6=P3+P6
0
C2
MUX 2
DMUX
C3

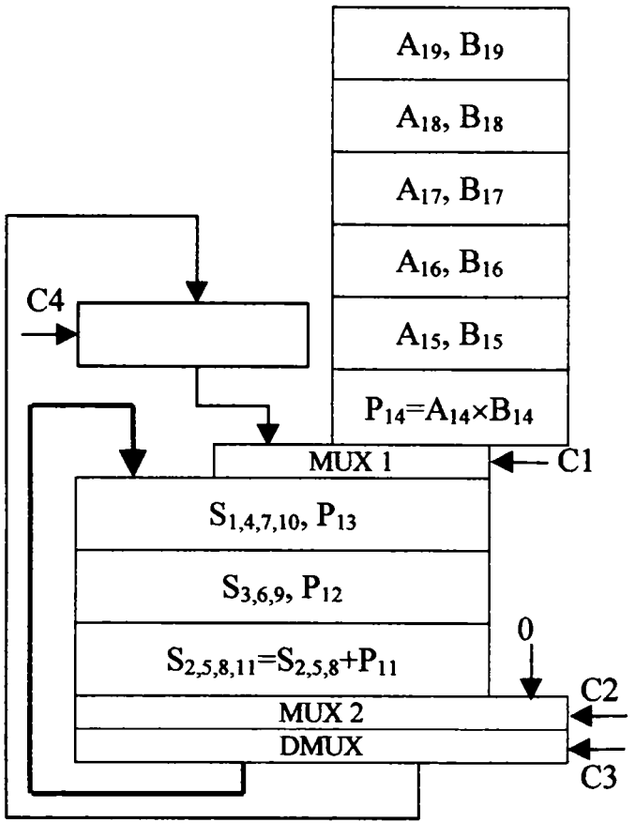
ANEXA 8

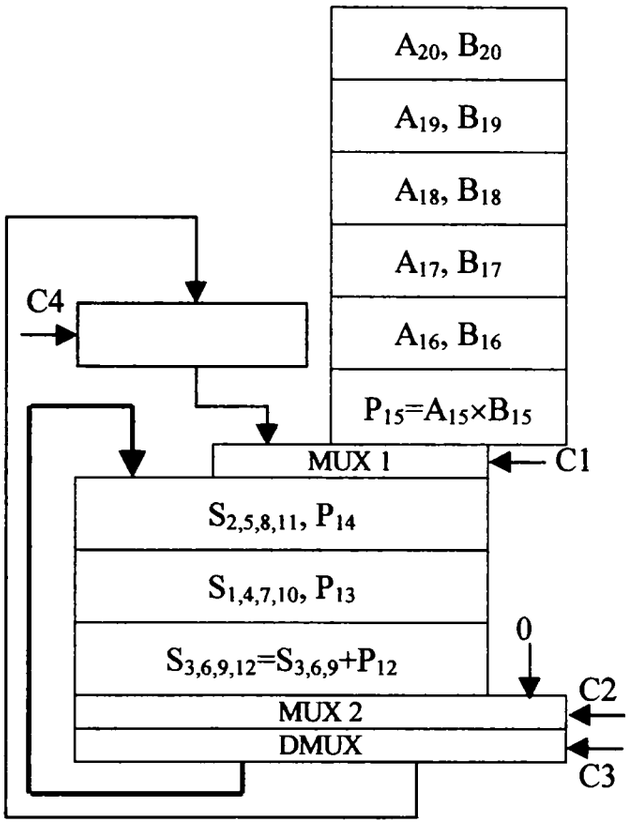

A15, B15
A14, B14
A13, B13
A12, B12
A11, B11
P10=A10×B10
C4
MUX 1
C1
S3,6, P9
S2,5, P8
S1,4,7=S1,4+P7
0
C2
MUX 2
DMUX
C3

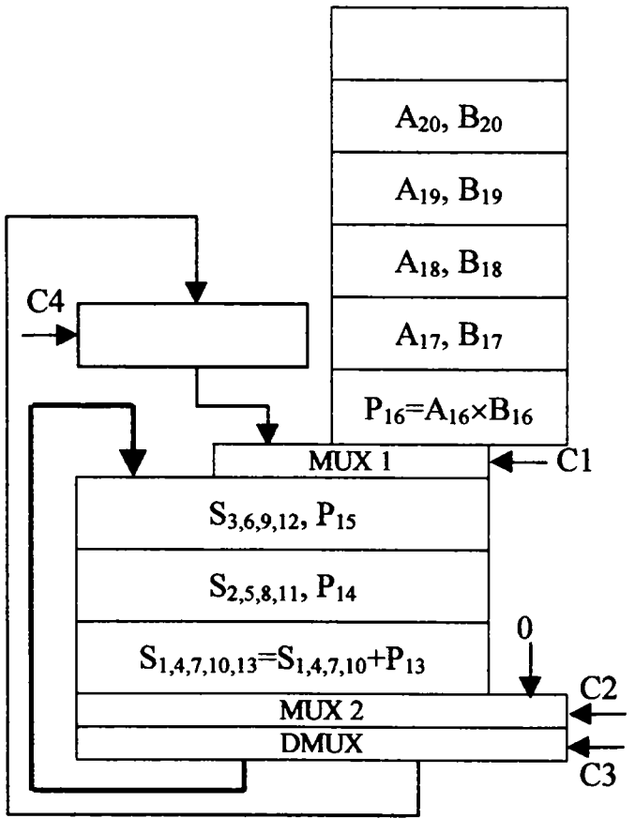

A16, B16
A15, B15
A14, B14
A13, B13
A12, B12
P11=A11×B11
C4
MUX 1
C1
S1,4,7, P10
S3,6, P9
S2,5,8=S2,5+P8
0
C2
MUX 2
DMUX
C3

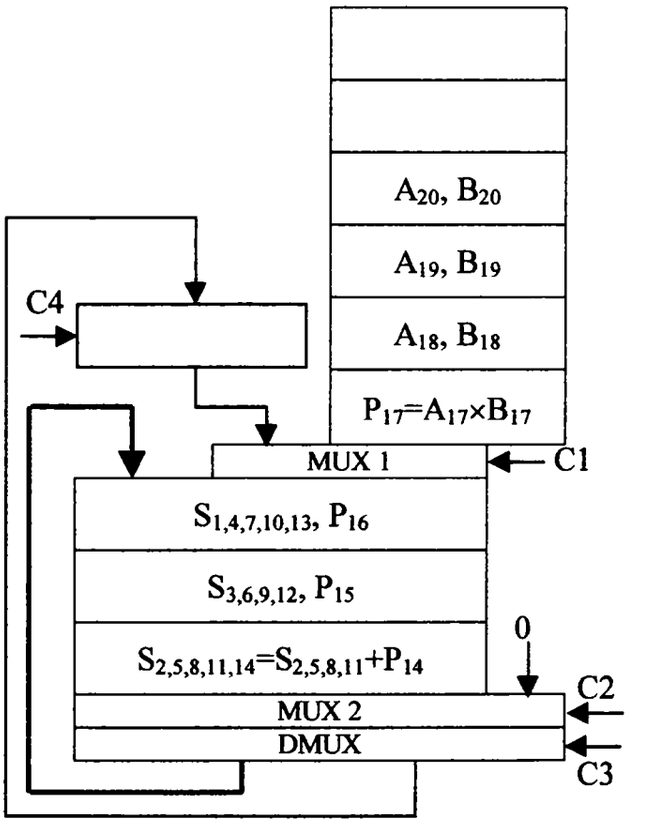

A17, B17
A16, B16
A15, B15
A14, B14
A13, B13
P12=A12×B12
C4
MUX 1
C1
S2,5,8, P11
S1,4,7, P10
S3,6,9=S3,6+P9
0
C2
MUX 2
DMUX
C3


A18, B18
A17, B17
A16, B16
A15, B15
A14, B14
P13=A13×B13
C4
MUX 1
C1
S3,6,9, P12
S2,5,8, P11
S1,4,7,10=S1,4,7+P10
0
C2
MUX 2
DMUX
C3

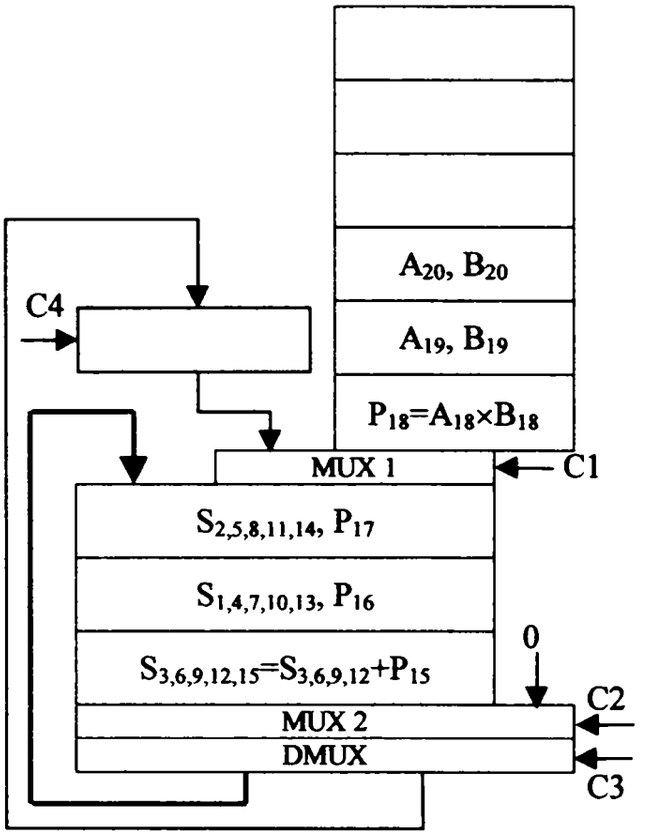

A19, B19
A18, B18
A17, B17
A16, B16
A15, B15
P14=A14×B14
C4
MUX 1
C1
S1,4,7,10, P13
S3,6,9, P12
S2,5,8,11=S2,5,8+P11
0
C2
MUX 2
DMUX
C3

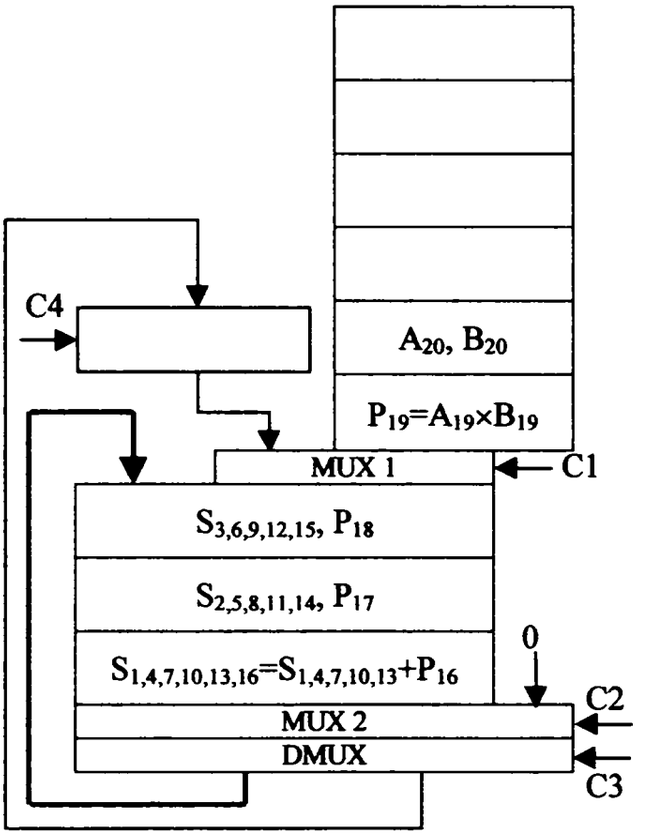

A20, B20
A19, B19
A18, B18
A17, B17
A16, B16
P15=A15×B15
C4
MUX 1
C1
S2,5,8,11, P14
S1,4,7,10, P13
S3,6,9,12=S3,6,9+P12
0
C2
MUX 2
DMUX
C3

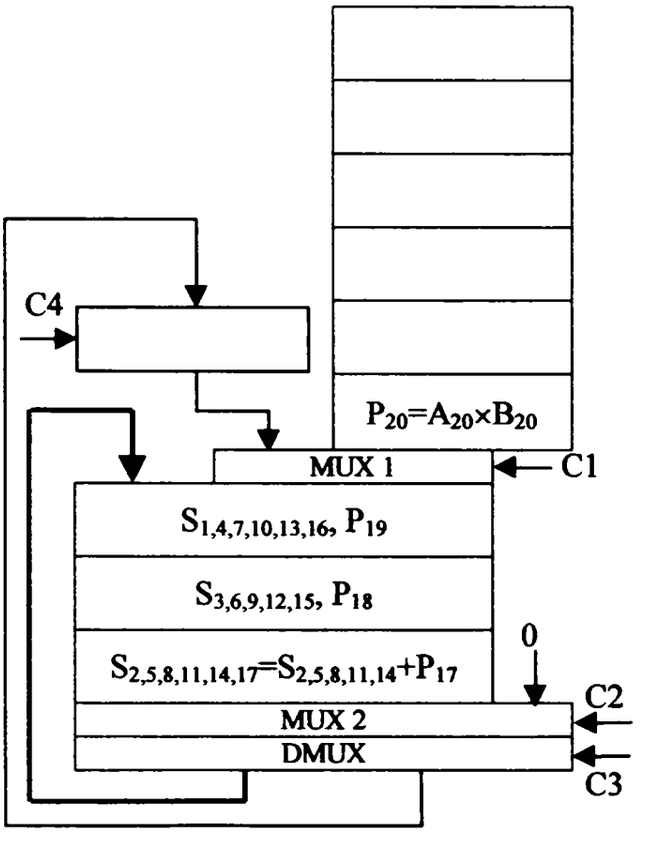

A20, B20
A19, B19
A18, B18
A17, B17
P16=A16×B16
C4
MUX 1
C1
S3,6,9,12, P15
S2,5,8,11, P14
S1,4,7,10,13=S1,4,7,10+P13
0
C2
MUX 2
DMUX
C3

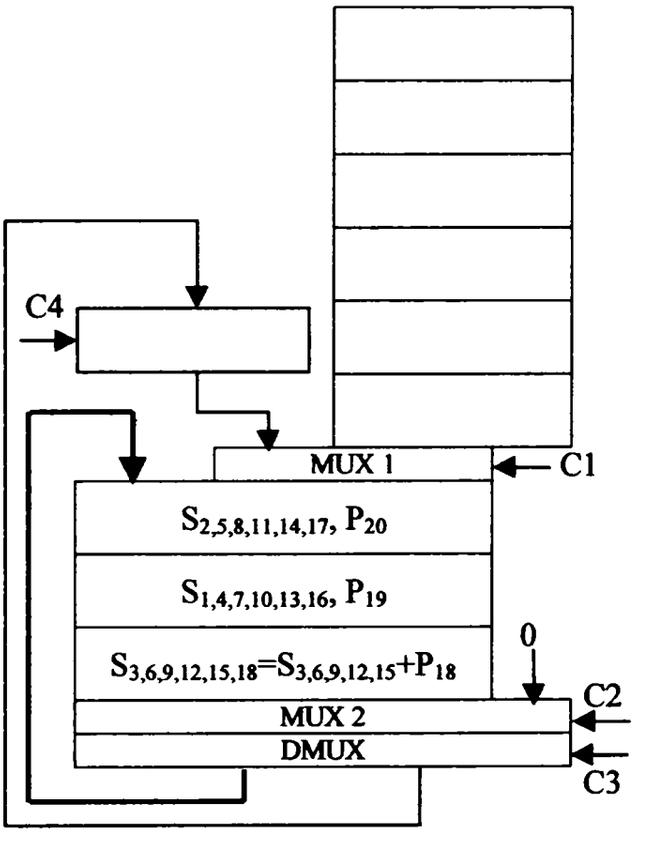

A20, B20
A19, B19
A18, B18
P17=A17×B17
C4
MUX 1
C1
S1,4,7,10,13, P16
S3,6,9,12, P15
S2,5,8,11,14=S2,5,8,11+P14
0
C2
MUX 2
DMUX
C3

ANEXA 8

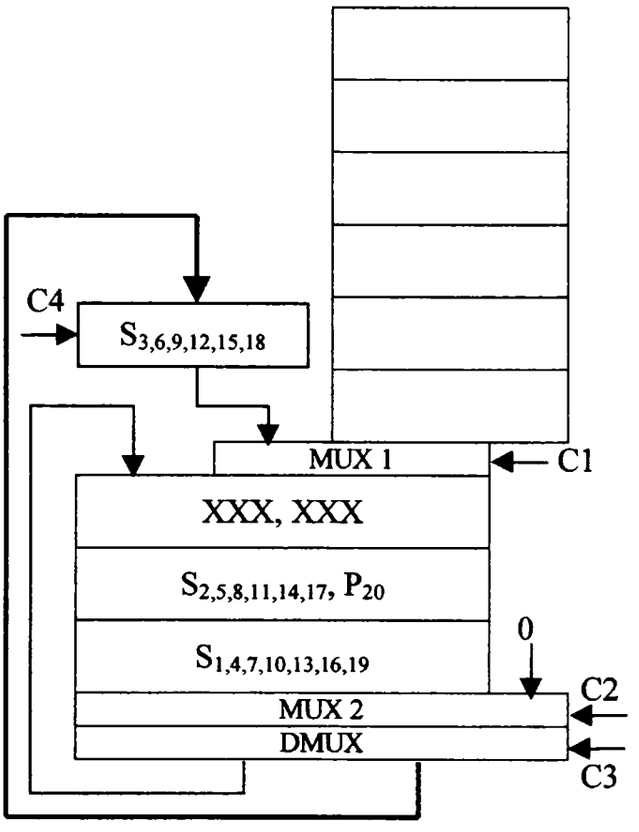

A20, B20
A19, B19
P18=A18×B18
C4
MUX 1
C1
S2,5,8,11,14, P17
S1,4,7,10,13, P16
S3,6,9,12,15=S3,6,9,12+P15
0
C2
MUX 2
DMUX
C3

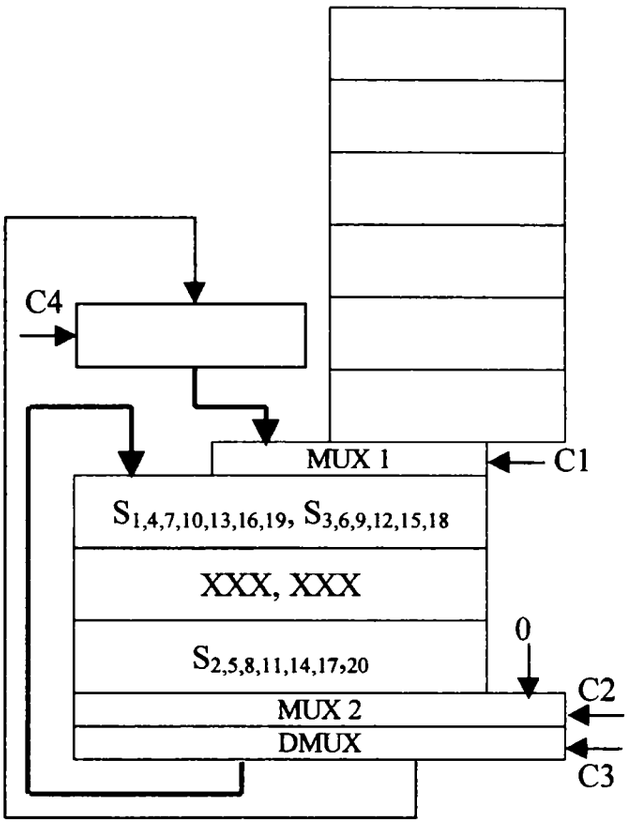

A20, B20
P19=A19×B19
C4
MUX 1
C1
S3,6,9,12,15, P18
S2,5,8,11,14, P17
S1,4,7,10,13,16=S1,4,7,10,13+P16
0
C2
MUX 2
DMUX
C3

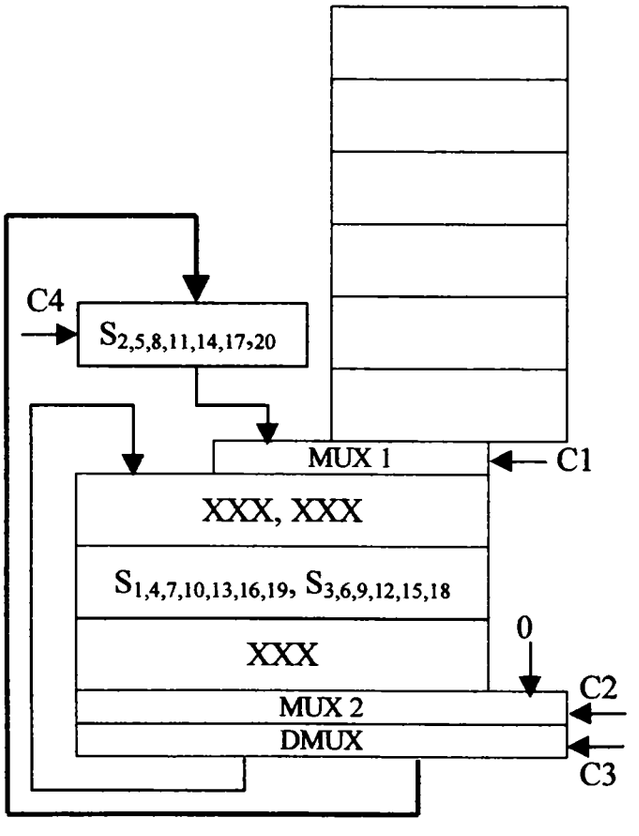

P20=A20×B20
C4
MUX 1
C1
S1,4,7,10,13,16, P19
S3,6,9,12,15, P18
S2,5,8,11,14,17=S2,5,8,11,14+P17
0
C2
MUX 2
DMUX
C3

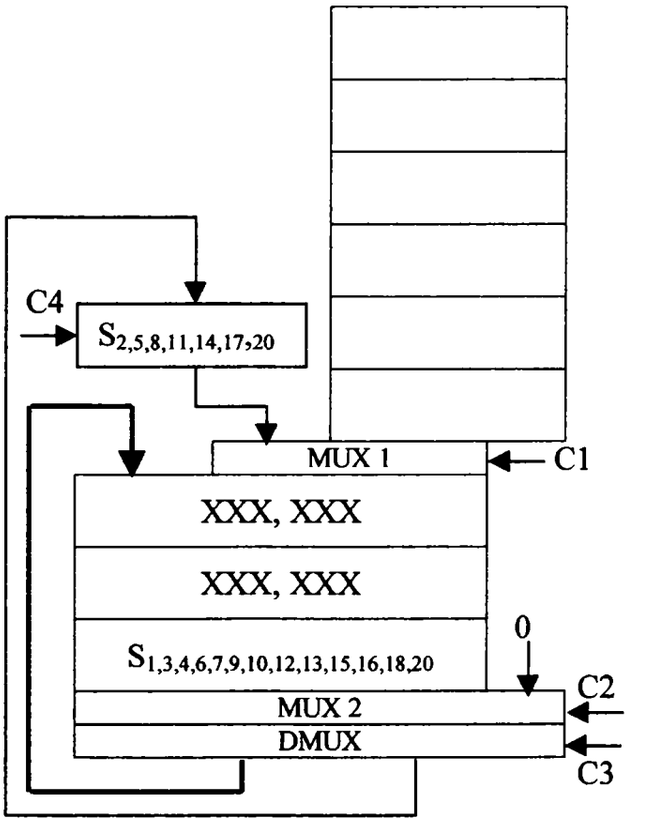

C4
MUX 1
C1
S2,5,8,11,14,17, P20
S1,4,7,10,13,16, P19
S3,6,9,12,15,18=S3,6,9,12,15+P18
0
C2
MUX 2
DMUX
C3

ANEXA 8

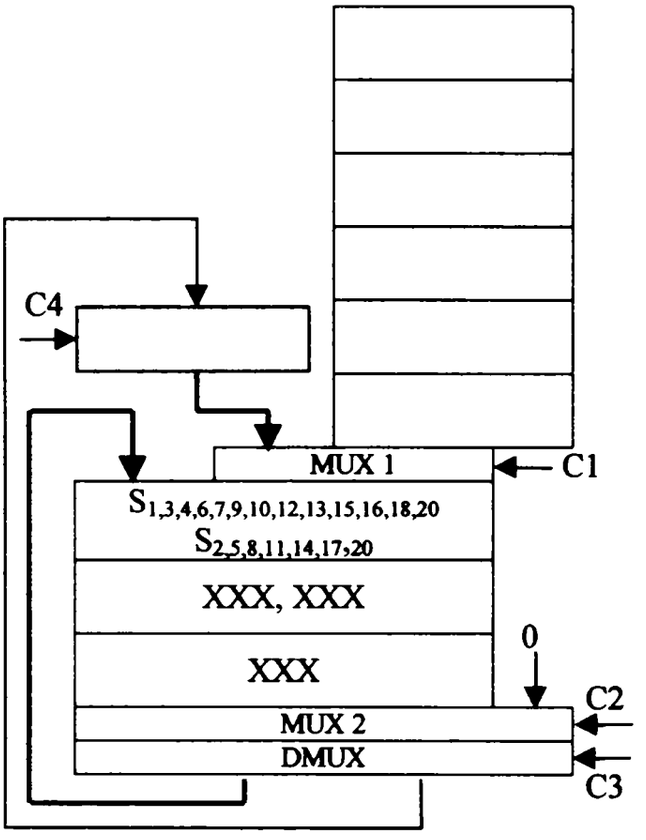

C4
$S_{3,6,9,12,15,18}$
MUX 1
C1
XXX, XXX
$S_{2,5,8,11,14,17}$, P_{20}
$S_{1,4,7,10,13,16,19}$
0
C2
MUX 2
DMUX
C3

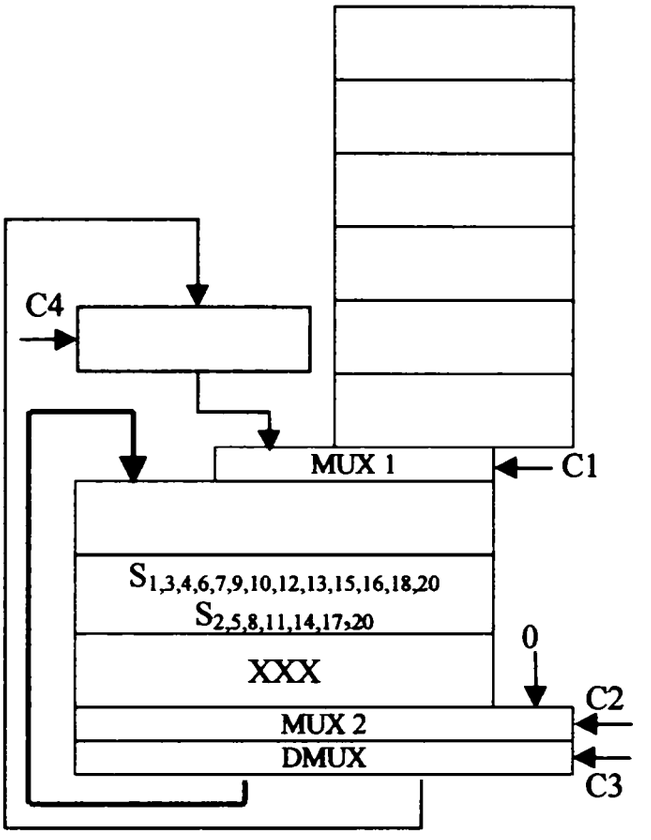

C4
MUX 1
C1
$S_{1,4,7,10,13,16,19}$, $S_{3,6,9,12,15,18}$
XXX, XXX
$S_{2,5,8,11,14,17,20}$
0
C2
MUX 2
DMUX
C3

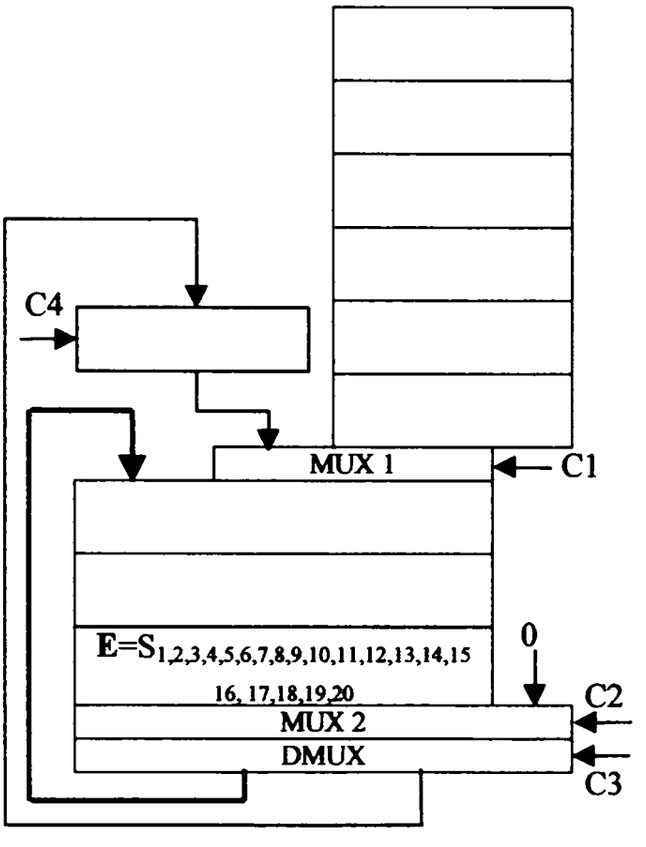

C4
$S_{2,5,8,11,14,17,20}$
MUX 1
C1
XXX, XXX
$S_{1,4,7,10,13,16,19}$, $S_{3,6,9,12,15,18}$
XXX
0
C2
MUX 2
DMUX
C3

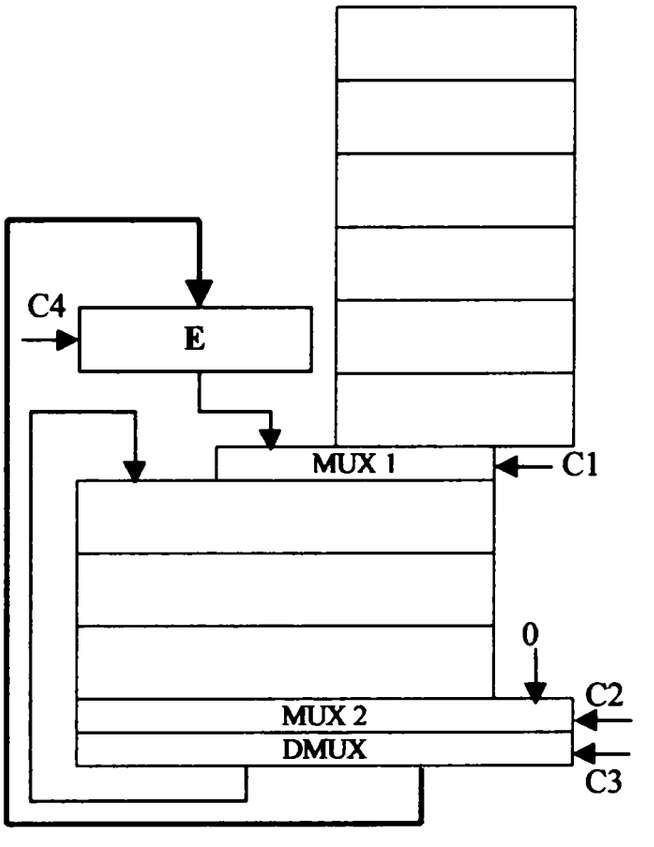

C4
$S_{2,5,8,11,14,17,20}$
MUX 1
C1
XXX, XXX
XXX, XXX
$S_{1,3,4,6,7,9,10,12,13,15,16,18,20}$
0
C2
MUX 2
DMUX
C3

ANEXA 8


C4
MUX 1
C1
$S_{1,3,4,6,7,9,10,12,13,15,16,18,20}$
$S_{2,5,8,11,14,17,20}$
XXX, XXX
XXX
0
C2
MUX 2
DMUX
C3


C4
MUX 1
C1
$S_{1,3,4,6,7,9,10,12,13,15,16,18,20}$
$S_{2,5,8,11,14,17,20}$
XXX
0
C2
MUX 2
DMUX
C3


C4
MUX 1
C1
$E=S_{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}$
16, 17,18,19,20
0
C2
MUX 2
DMUX
C3


C4
E
MUX 1
C1
0
C2
MUX 2
DMUX
C3

Index alfanumeric pentru circuitele prezentate în anexe