Lucrarea 3

Sumatoare Ripple, Carry-Lookahead şi Carry Save

În această lucrare sunt introduse sumatoarele Ripple Carry, Carry Lookahead şi Carry Save. Apoi este prezentat cadrul în care se pot face evaluări cu privire la întârzierile care apar la sumatoarele paralele.

1 Dispozitive de adunare şi scădere

1.1 Sumatorul Serial (Serial Adder)

Sumatorul serial este un automat secvențial, a cărui comportare este dată de tabelul de stări. Avem două stări posibile din punct de vedere funcțional:

- S_0 - starea la care se adună la un moment dat 2 biți şi din starea anterioar nu a venit transport (carry);
- S_1 - starea la care se adună la un moment dat 2 biți şi avem carry.

Tabelul tranzițiilor este prezentat mai jos. La construirea sumatorului folosim bistabile de tip JK:

$$y(t+1) = J \cdot \overline{y(t)} + \overline{K} \cdot y(t) \qquad (3.1)$$

Ecuațiile corespunzătoare unei celule sumator complet sunt (ele sunt construite pe baza Tabelei 3.1):

$$\begin{cases} z_i = x_i \oplus y_i \oplus c_i \\ c_{i+1} = x_i \cdot y_i + x_i \cdot c_i + y_i \cdot c_i \end{cases} \qquad (3.2)$$

Inputs			Outputs	
x_i	y_i	c_i	z_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 3.1 Tabela de adevăr a sumatorului complet pe 1 bit.

Prin urmare, soluția de proiectare a sumatorului serial, care conține un sumator complet pe 1 bit (structură combinațională) şi un element de memorare – bistabil (element de logică secvențială) este prezentată în Figura 3.1.

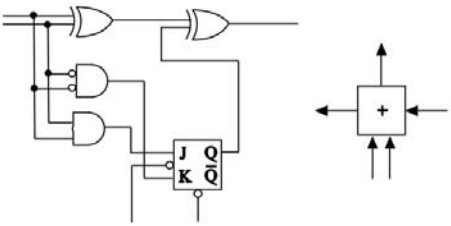


Figura 3.1 Sumatorul secvențial, proiectat cu un bistabil de tip JK.

Din nefericire un sumator serial execută suma într-un număr de impulsuri de clock egal cu *n* (dimensiune operanzilor) şi de accea, uzual, se apelează la sumatoare paralele.

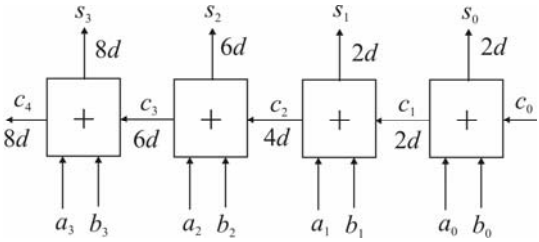


Figura 3.2 Sumatorul RCA pe 4 biți cu întârzierile corespunzătoare.

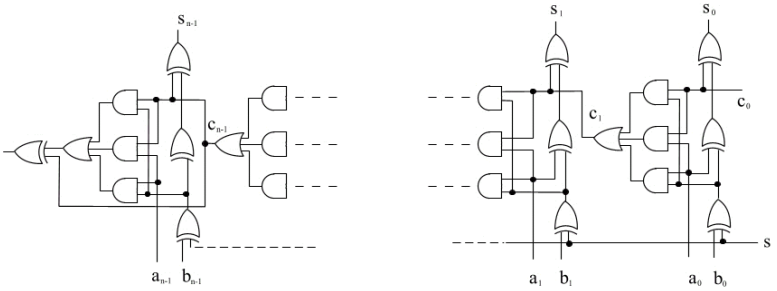


Figura 3.3 Prezentare detaliată a unui sumator/Scăzător binar, bazat pe soluția RCA.

2 Sumatoare Paralele

2.1 Ripple Carry Adder (RCA)

Un sumator cu transport propagat serial se obține prin legarea în serie (cascadă) a *n* celule sumator complet pe 1 bit, dacă avem operanzi pe *n* biți. În mod caracteristic, operația propriu-zisă se execută într-un singur impuls de clock, numai că perioada acestui clock trebuie să acopere intervalul de timp necesar propagării transportului în manieră serială prin întreaga schemă. Dacă „delay"-ul (întârzierea) printr-un circuit poartă *AND* sau *OR* este *d* (întârzierea pe o poartă *XOR* se consideră a fi 2*d*), atunci perioada clock-ului trebuie să fie $T \geq 2nd$. Figura 3.2 prezintă în detaliu soluția RCA pe 4 biți, la nivel de celulă de însumare pe 1 bit, precum şi întârzierile ce apar.

Operația de scădere se bazează pe adunarea complementului, şi pentru generarea complementului vom apela la o adăugare la schemă:

$$B+(A\oplus s),\text{ meaning }\begin{cases} s=0 & \Rightarrow \quad A+B \\ s=1 & \Rightarrow \quad B+\underbrace{\overline{A}+1}_{-A}=B-A \end{cases} \tag{3.3}$$

Pe 4 biți, un astfel de sumator va avea întârzierea RCA-ului corespunzător, 8*d*, la care se adaugă 2*d* de la porțile *XOR* suplimentare, per total 10*d*.

2.2 Carry lookahead adder –CLA (Sumator cu transport anticipat)

La anterioarele soluții, faptul că transportul trebuia propagat în manieră serială greva asupra performanței soluției, şi s-a plecat de la ideea îmbunătățirii timpului de propagare a transportului prin generarea în mod anticipat al transportului într-o manieră care să fie bazată pe „istorica valorică a operanzilor“:

- g_i - variabila de generare a transportului (generate);
- p_i - variabila de propagare a transportului (propagate);

În manieră recursivă avem ecuațiile unui CLA pe 4 biți:

$$\begin{cases} c_1 = a_0\cdot b_0 + a_0\cdot c_0 = \underbrace{a_0\cdot b_0}_{g_0} + \underbrace{(a_0+b_0)}_{p_0}\cdot c_0 \\ c_2 = \underbrace{a_1\cdot b_1}_{g_1} + \underbrace{(a_1+b_1)}_{p_1}\cdot c_1 = g_1 + p_1\cdot g_0 + p_1\cdot p_0\cdot c_0 \\ c_3 = g_2 + p_2\cdot g_1 + p_2\cdot p_1\cdot g_0 + p_2\cdot p_1\cdot p_0\cdot c_0 \\ c_4 = g_3 + p_3\cdot g_2 + p_3\cdot p_2\cdot g_1 + p_3\cdot p_2\cdot p_1\cdot g_0 + p_3\cdot p_2\cdot p_1\cdot p_0\cdot c_0 \end{cases} \tag{3.4}$$

Celula de însumare suferă o transformare în construcțiile CLA, ea se etichetează cu +*, şi deseori este denumită “celula degenerată de însumare”; Figura 3.4 o prezintă în detaliu.

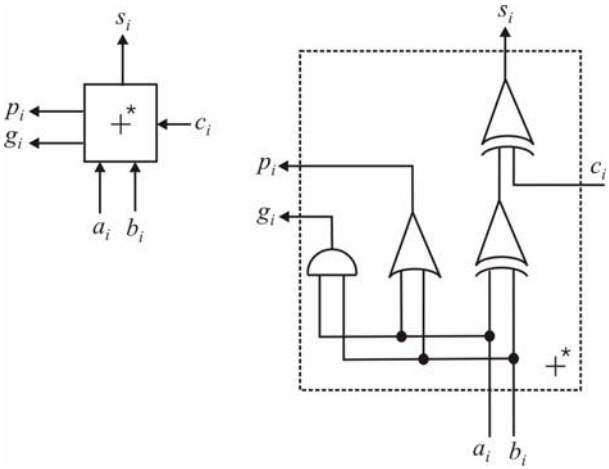


Figura 3.4 Celula de însumare modificată CLA.

Avantajele acestei construcții de tip CLA sunt:

- avem doar 5 niveluri logice față de 2*n* niveluri logice ca la ripple-carry-adder pentru obținerea sumei. Cele 5 niveluri logice sunt:
  - generarea lui *p* şi *g*;
  - 2,3-generarea lui carry;
  - 4,5-*XOR* pentru sumă ($\oplus$ se realizează pe 2 niveluri logice).

Dezavantajele sunt:

- creşte numărul intrărilor din porțile *AND* pentru carry şi *OR* (pentru bitul *i* avem *i*+1 intrări); creşte „fan-out"-ul *p*-urilor, adică numărul de circuite pe care le poate alimenta.

De exemplu „fan-out"-ul lui P_{n-1} este egal cu *n*; structura este neregulată şi creşte în complexitate – de aceea apar probleme legate de implementare. Tehnologiile actuale permit implementarea eficientă doar a celulelor CLA pe 4 biți, acestea fiind incluse în construcții special dedicate CLA-urilor pe mai multi biți. Un CLA pe 4 biți va avea o întârziere maximă de 3*d* pe semnalele de carry şi de 5*d* pe semnalele de sumă.

Aceste considerații au deschis o problemă a construcției CLA şi problema acestei construcții se bazează pe următorul raționament:

$$
\begin{aligned}
& c_1 = g_0 + p_0 \cdot c_0 \\
& c_0 = G_{0,1} + P_{0,1} c_0 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \bullet c_0 \qquad (3.5) \\
& \begin{cases} G_{0,1} = g_1 + p_1 \cdot g_0 \\ P_{0,1} = p_1 \cdot p_0 \end{cases}
\end{aligned}
$$

2.3 Carry Save Adder

Această soluție se caracterizează prin faptul că este alcătuită din mai multe niveluri de celule sumatoare complet disjuncte. O astfel de structură de sumatoare permite însumarea la fiecare nivel nu a doi operanzi ci a trei operanzi. Ea apare ca foarte utilă atunci când se încearcă accelerarea operației de înmulțire, care se bazează pe adunări repetate, prin pipelinizare aritmetică. Menționăm că pentru a obține suma corectă bazată pe CSA, la salvarea transportului este obligatoriu ca ultimul nivel de însumare să fie reprezentat de un sumator clasic (de exemplu un RCA).

Figura 3.5 prezintă un sumator CSA pentru însumarea a 4 numere binare pe 4 biți. Ecuația 3.6 explică modul în care se realizează această însumare în manieră CSA.

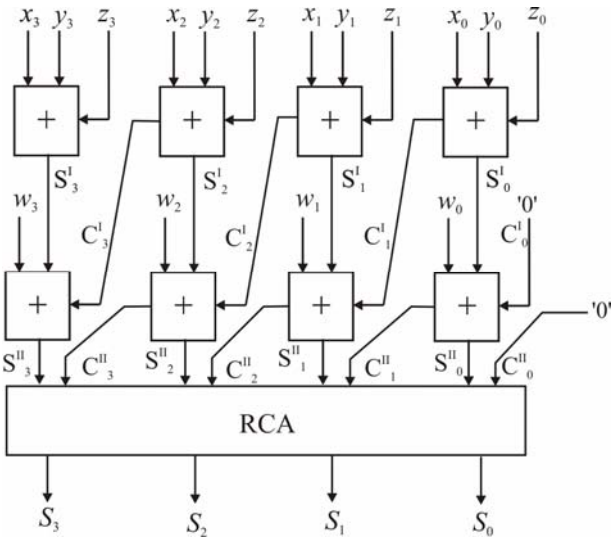


Figura 3.5 Carry Save Adder – exemplu pe 4 biți.

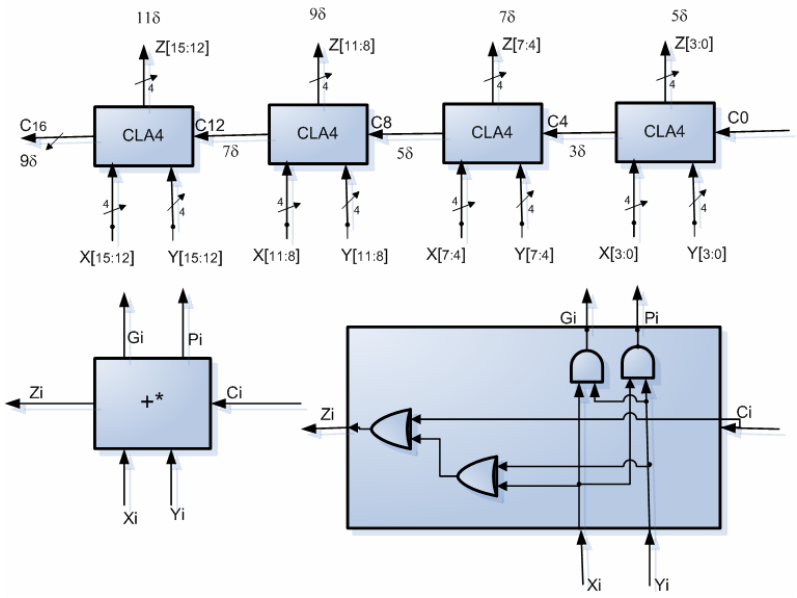
$$
\begin{array}{lll}
 & x = 0\,0\,1\,0 & + \\
 & y = 0\,1\,0\,0 & \\
 & \underline{z = 0\,0\,1\,1} & \\
 & s^{I} = 0\,1\,0\,1 & + \\
2\,x & c^{I} = 0\,1\,0\,0 & \qquad (3.6) \\
 & \underline{w = 0\,1\,0\,1} & \\
 & s^{II} = 0\,1\,0\,0 & + \\
2\,x & \underline{c^{II} = 1\,0\,1\,0} & \\
 & \boxed{S = 1\,1\,1\,0} &
\end{array}
$$

3 Aplicații

Problema 3.1 Să se construiască un sumator pe 16 biți, din 4 celule CLA pe 4 biți conectați în manieră RCA. Se cere şi estimarea penalității de

performanță a sumatorului proiectat, în termeni de τ, unde τ este întârzierea pe porțile *AND* şi *OR*, iar 2τ este întârzierea pe poarta *XOR*.

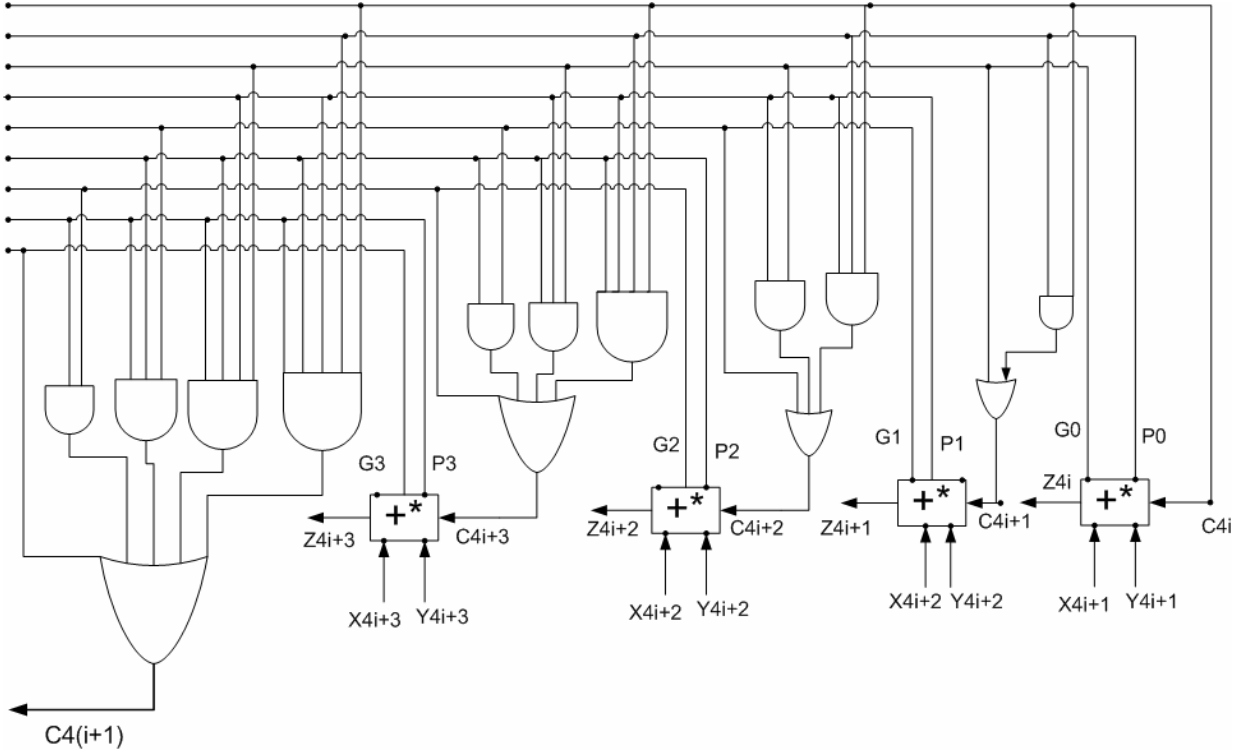
Se vor folosi celule de însumare ca în figura de mai jos: CLA pe 4 biți legate în manieră RCA şi celula degenerată CLA pe 1 bit, folosită pntru construcția celulelor CLA4.



Toate Q*i*-urile şi P*i*-urile au întârzieri de 1τ. Întârzierile pe carry-urile generate de logica de lookahead este de maxim 3τ, deoarece semnalele P*i* şi Q*i* alimentează doua niveluri succesive de porti.

Astfel C4 va avea întârziere de 3τ iar întârzierea maximă pe $Z[3:0] = Z_3Z_2Z_1Z_0$ va fi de 5τ.

Subsecvent se adaugă 2τ carry-urile care intră în celelalte celule, iar întârzierii pe Z-uri se mai adaugă 2τ. Figura de mai jos prezintă soluția de implementare a celulelor CLA pe 4 biți, care sunt legate în maniera RCA.


G3
P3
G2
P2
G1
P1
G0
P0
Z4i
+*
Z4i+3
C4i+3
Z4i+2
C4i+2
Z4i+1
C4i+1
C4i
X4i+3
Y4i+3
X4i+2
Y4i+2
X4i+2
Y4i+2
X4i+1
Y4i+1
C4(i+1)

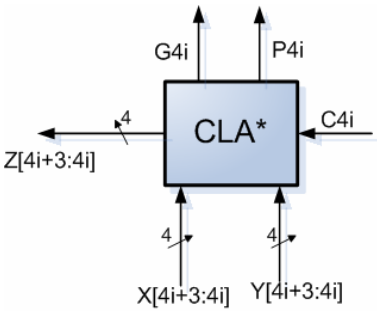
Problema 3.2

Să se construiască un sumator pe 16 biți din 4 celule CLA pe 4 biți, conectate în manieră CLA. Se cere şi estimarea penalității de performanță a sumatorului proiectat, în termeni de τ.

Carry Out-ul din prima celulă CLA4 nu se mai foloseşte, creându-se două noi semnale generate şi propagate: G0 şi P0.

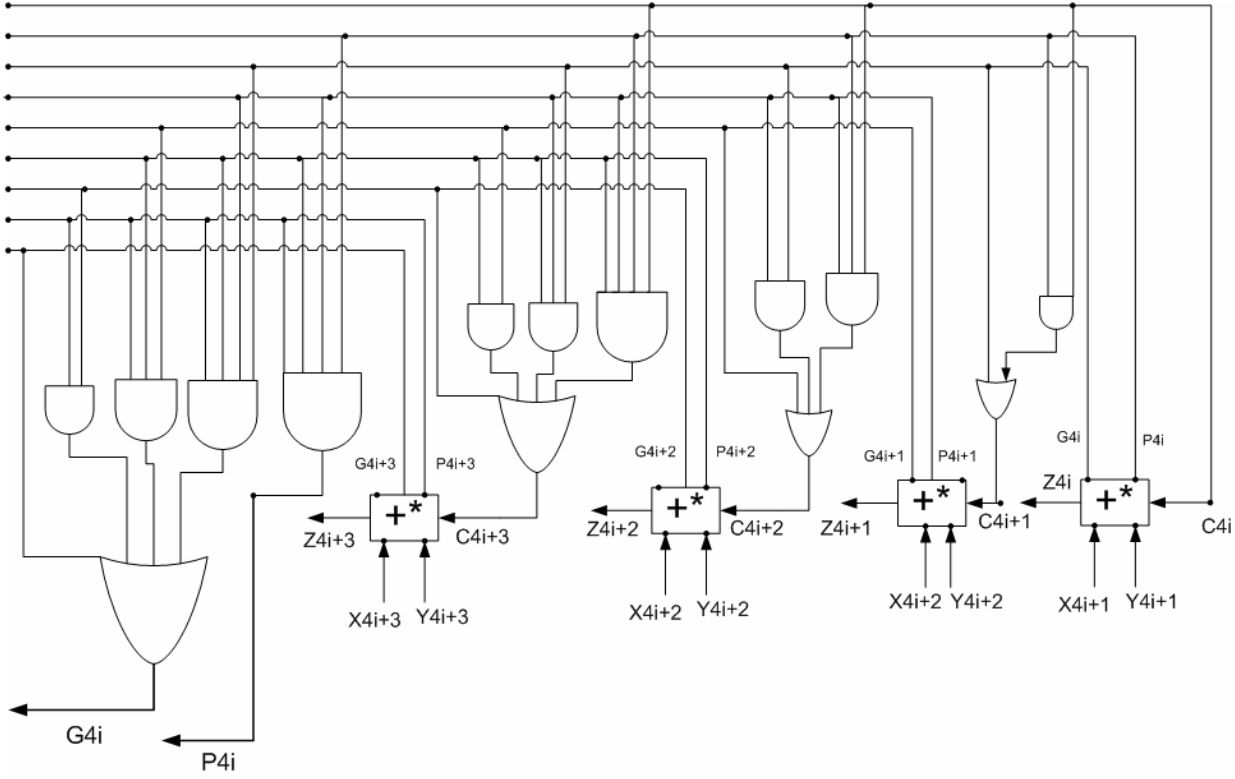
$$C_4 = G_0 + P_0 C_0 = c_4 = \underbrace{g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0}_{G_0} + \underbrace{p_3 p_2 p_1 p_0 c_0}_{P_0}$$

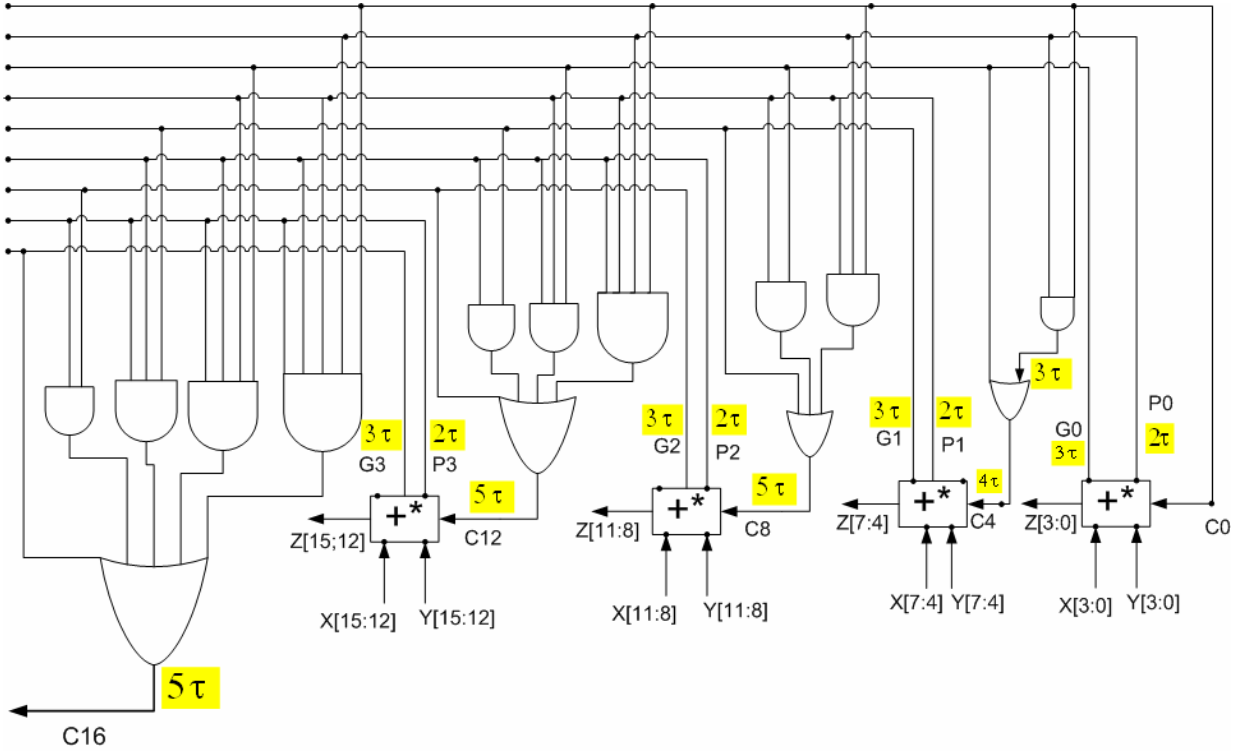
Noua celulă CLA, modificată, este prezentată în următoarea figură:



Detaliile acestei celule sunt prezentate în următoarea figură, iar apoi este prezentată construcția CLA cu celule CLA modificate. G4i si P4i au ca input-uri Pi ş Gi care au 1τ întârziere, rezultă P4i va avea 2τ întârziere, iar G4i va avea 3τ întârziere.

În ultima figură, C0 nu are întârziere, C4 - 4τ, C8, C12, C16 - 5τ. Prin urmare z[3:0] vor avea o întârziere maximă de 5τ, Z[7:4] - 8τ, iar Z[11:8], Z[15:12] - 9τ.


G4i+3
P4i+3
+*
Z4i+3
C4i+3
X4i+3
Y4i+3
G4i+2
P4i+2
+*
Z4i+2
C4i+2
X4i+2
Y4i+2
G4i+1
P4i+1
+*
Z4i+1
C4i+1
X4i+2
Y4i+2
G4i
P4i
Z4i
+*
C4i
X4i+1
Y4i+1
G4i
P4i


3 τ
G3
2τ
P3
5 τ
+*
Z[15;12]
C12
X[15:12]
Y[15:12]
3 τ
G2
2τ
P2
5 τ
+*
Z[11:8]
C8
X[11:8]
Y[11:8]
3 τ
G1
2τ
P1
4τ
+*
Z[7:4]
C4
X[7:4]
Y[7:4]
3 τ
G0
3τ
P0
2τ
+*
Z[3:0]
C0
X[3:0]
Y[3:0]
5 τ
C16

Problema 3.3

Să se implementeze un dispozitiv combinațional care efectuează adunarea/scăderea unor numere întregi reprezentate în semn-mărime pe 4 biți (1 semn + 3 mărime).

Soluție:

Pasul 1. Implementarea scăzătorului binar.

```
  10100   –
   1111
-------
  01111
```

x_i	y_i	b_i	z_i	b_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

z_i

b_i \ x_iy_i	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$z_i = \overline{x_i}\,\overline{y_i}b_i + x_i y_i b_i + \overline{x_i} y_i \overline{b_i} + x_i \overline{y_i}\,\overline{b_i}$$

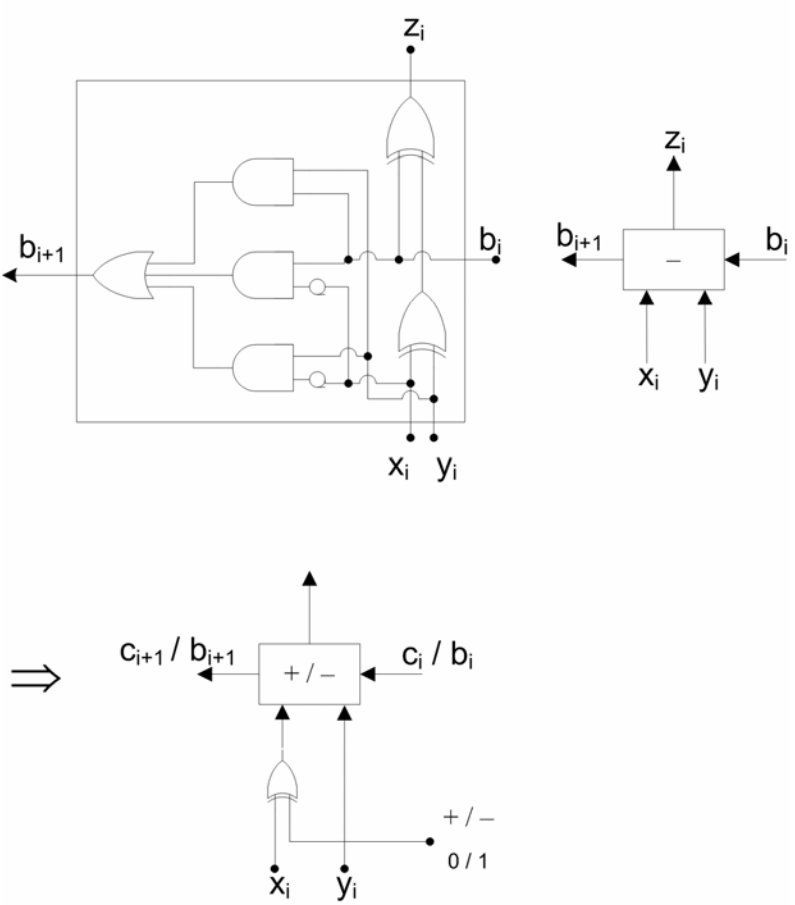
$$z_i = \overline{b_i}(x_i \oplus y_i) + b_i(\overline{x_i \oplus y_i})$$

$$z_i = x_i \oplus y_i \oplus b_i$$

b_{i+1}

b_i \ x_iy_i	00	01	11	10
0	0	1	0	0
1	1	1	1	0

$$b_{i+1} = \overline{x_i} y_i + \overline{x_i} b_i + y_i b_i$$

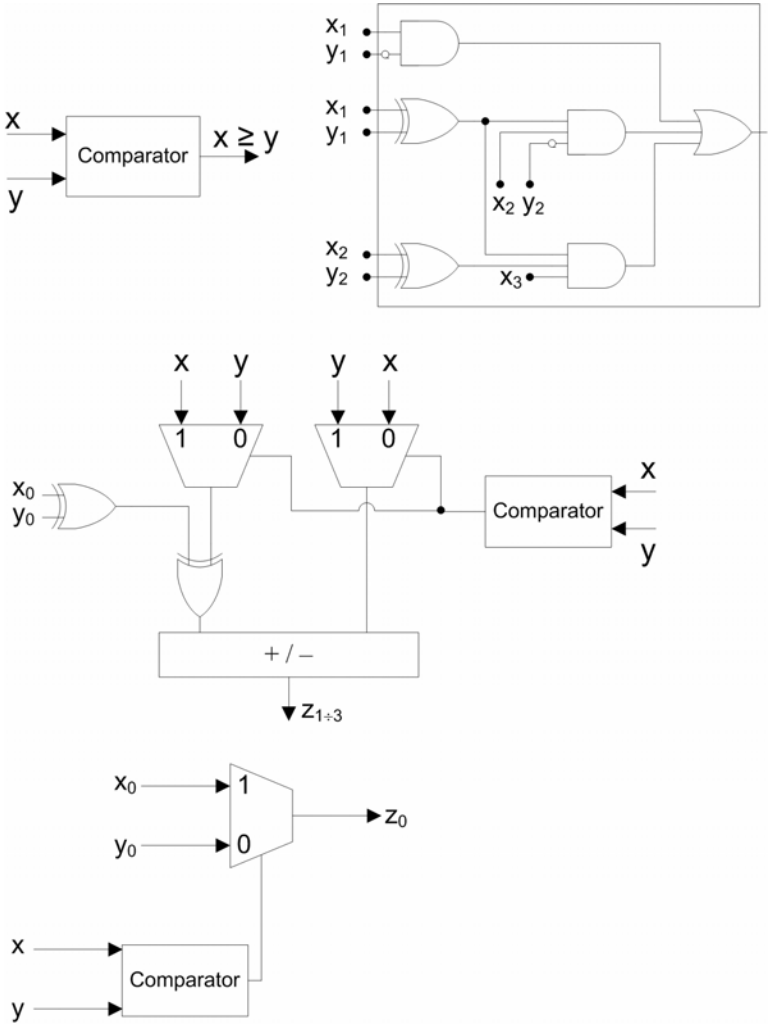


x, y pozitive / negative ⇒ adunare, altfel scădere
Dacă e scădere, semnul e impus de numărul mai mare în modul ⇒ implementarea unui comparator.

Pasul 2. Implementarea comparatorului.

x_1	x_2	x_3	y_1	y_2	y_3	$x \geq y$
1	x	x	0	x	x	yes
0	x	x	1	x	x	no
z_1	1	x	z_1	0	x	yes
z_1	0	x	z_1	1	x	no
z_1	z_2	1	z_1	z_2	x	yes
z_1	z_2	0	z_1	z_2	1	no

x_0	y_0	Semn rezultat
0	0	0
1	1	1
0	1	$x \geq y \Rightarrow x_0$, altfel y_0
1	0	



Problema 3.4 (propusă)
Să se realizeze sinteza unui dispozitiv de adunare/scădere în complement de 1, pentru operanzi pe 4 biți.