Addition Matérielle

Alain GUYOT

Concurrent Integrated Systems
TIMA

(33) 04 76 57 46 16

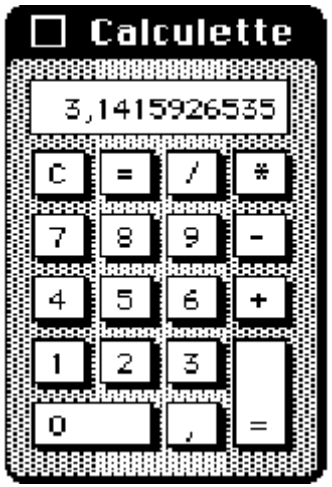
Alain.Guyot@imag.fr

http://tima-cmp.imag.fr/Homepages/guyot

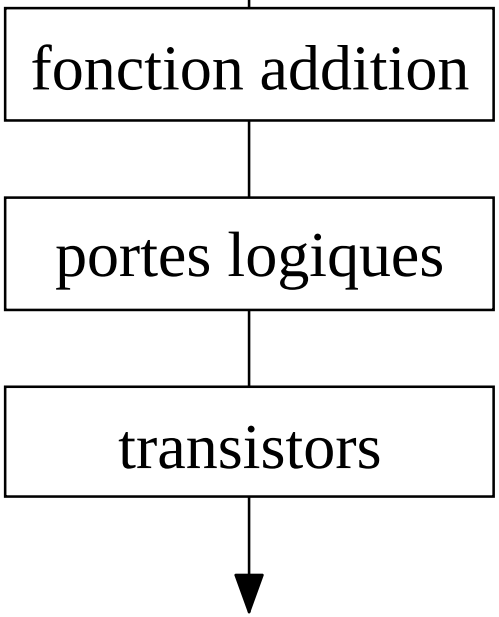
Techniques de l'Informatique et de la Microélectronique
pour l'Architecture. Unité associée au C.N.R.S. n° B0706

But
Réaliser des additionneurs combinatoires.
Optimiser le coût et/ou la vitesse
et/ou la consommation.

Problème
- Propagation de
  la retenue



Contraintes:
délai max
consommation max
surface max.



Remarque:
l'addition est l'opération
arithmétique la plus commune.

Rappels sur l'écriture des entiers en base 2

Digital

Entiers positifs

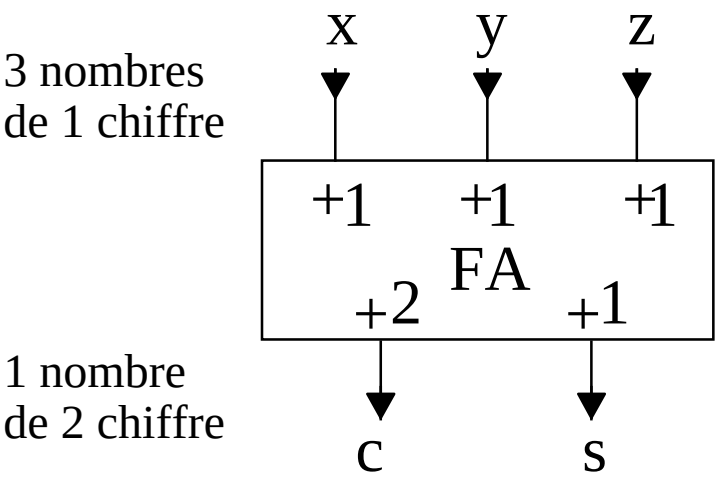
$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad a_i \in \{ 0 , 1 \} \qquad A \in [0, 2^n - 1]$$

Notation de position imitée de la notation décimale adoptée en Europe au XI^{ème} siècle. La valeur d'un nombre est la somme pondérée de ses chiffres.

Entiers relatifs

$$B = -b_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i \, 2^i \qquad b_i \in \{ 0 , 1 \} \qquad B \in [-2^{n-1}, +2^{n-1} - 1]$$

Fonction "Full Adder" (FA)



x	y	z	Σ	c	s
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	2	1	0
1	0	0	1	0	1
1	0	1	2	1	0
1	1	0	2	1	0
1	1	1	3	1	1

$x, y, z, c, s \in \{0 , 1\}$

$x + y + z \equiv 2*c + s$

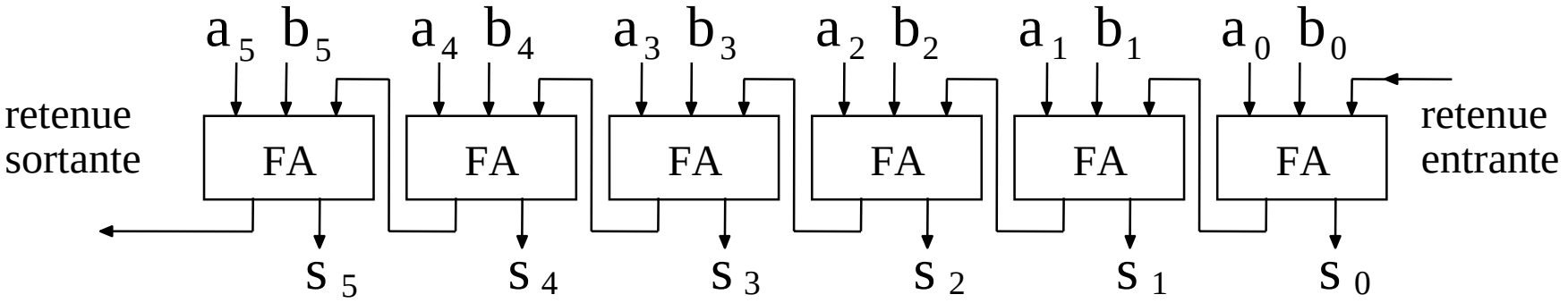
$s = x \oplus y \oplus z$ *somme modulo 2*

$c = \text{majorité}(x,y,z) = x \wedge y \vee x \wedge z \vee y \wedge z$

La somme pondérée de ce qui sort du “FA”est égale à la somme pondérée de ce qui entre dans le “FA”

Addition de Nombres ≥ 0

$$A = \sum_{i=0}^{5} a_i 2^i \qquad B = \sum_{i=0}^{5} b_i 2^i \qquad S = \sum_{i=0}^{5} s_i 2^i \qquad a_i, b_i, s_i \in \{0 , 1\}$$



Tout assemblage cohérent de “FA” conserve la propriété: La somme pondérée de ce qui sort est égale à la somme pondérée de ce qui entre

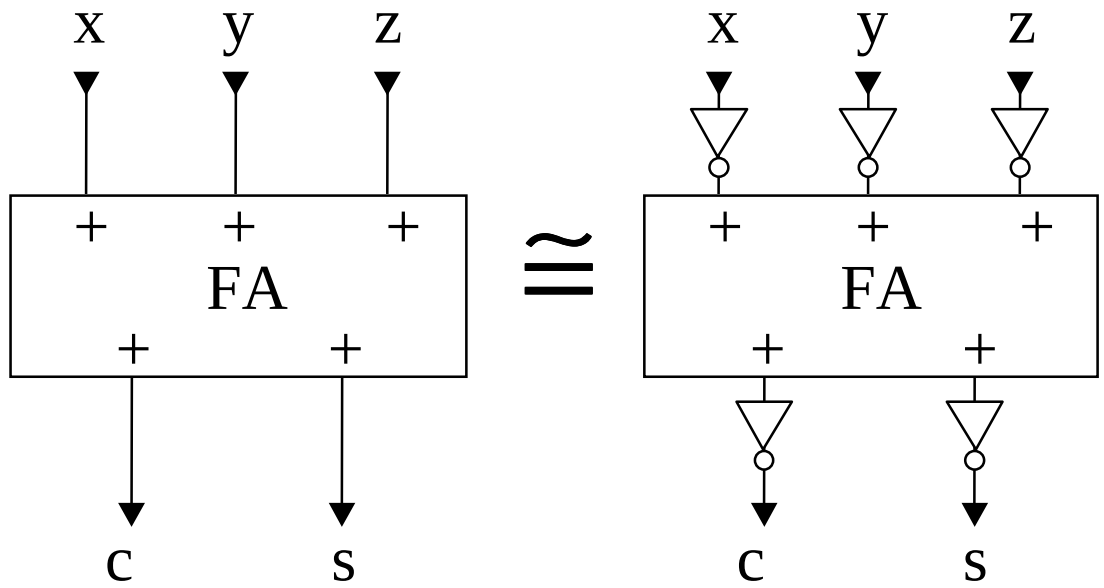
Si on ignore la retenue sortante:

$$S = \left(A + B + \text{retenue entrante} \right)_{\text{modulo } 2^6}$$

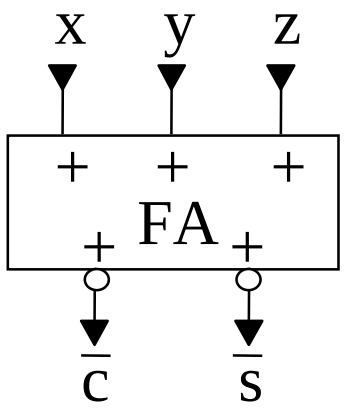
Dans ce cas l’opérateur accepte 2 conventions.

Le "Full Adder" est Autodual
(propriété générale des additionneurs)

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



"Full Adder" (FA) symétrique

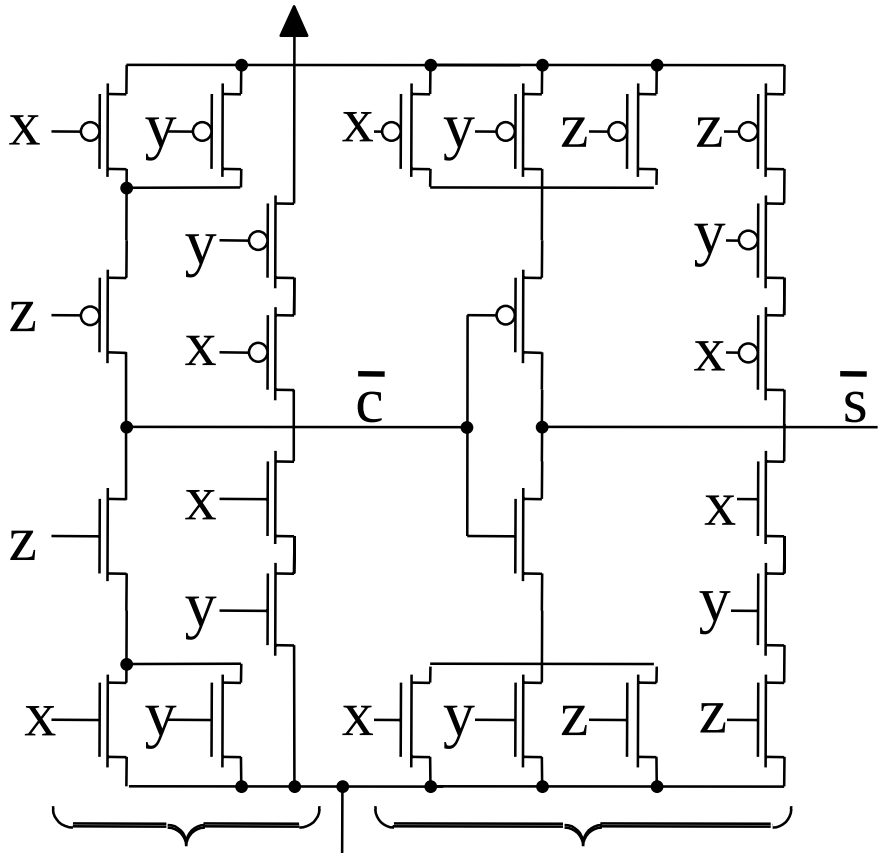


c = majorité (x,y,z)

s = **si** c **alors** (x∧y∧z) **sinon** (x∨y∨z)

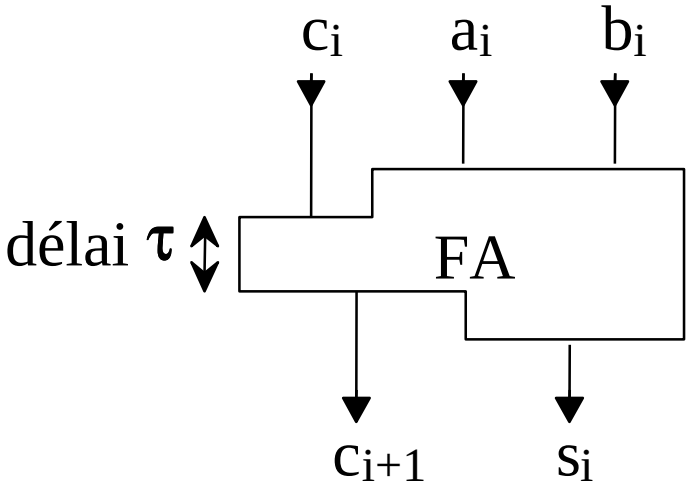
s = (x∧y∧z) ∨ $\overline{c}$∧(x∨y∨z)

	∧	M	∨	c	s
000	0	0	0	0	0
001	0	0	1	0	1
011	0	1	1	1	0
111	1	1	1	1	1



Circuit « miroir »

"Full Adder" (FA) dissymétrique



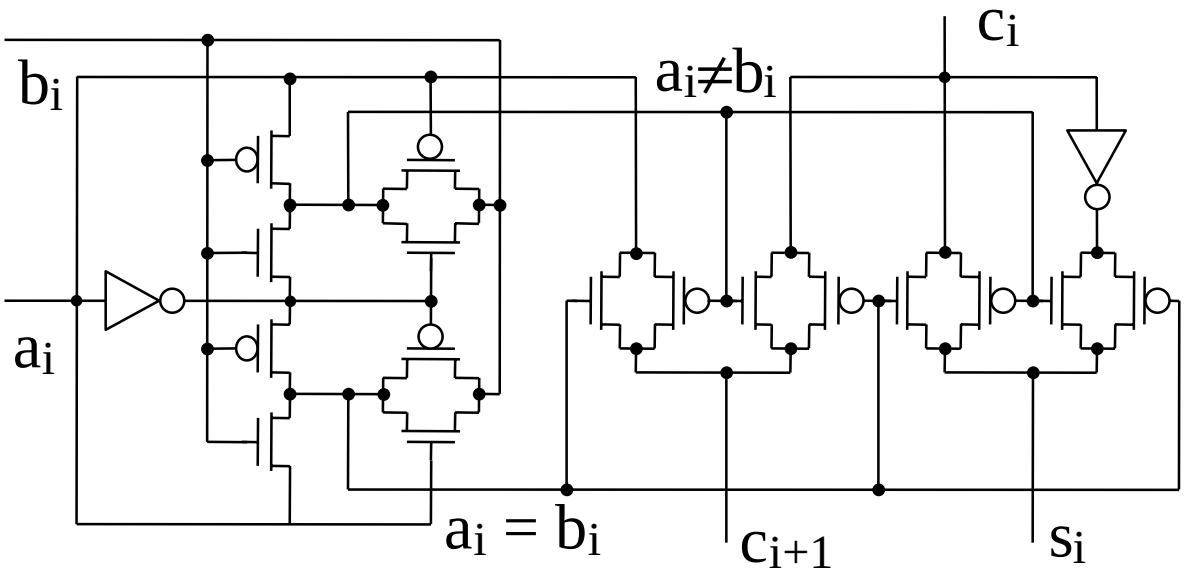
minimiser le délai τ entre
c_i et c_{i+1} (au dépens des autres)

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

a_i	b_i	c_{i+1}	s_i
0	0	0	c_i
0	1	c_i	$\overline{c_i}$
1	0	c_i	$\overline{c_i}$
1	1	1	c_i

a_i	b_i	c_{i+1}	s_i
$a_i = b_i$		a_i	c_i
$a_i \neq b_i$		c_i	$\overline{c_i}$

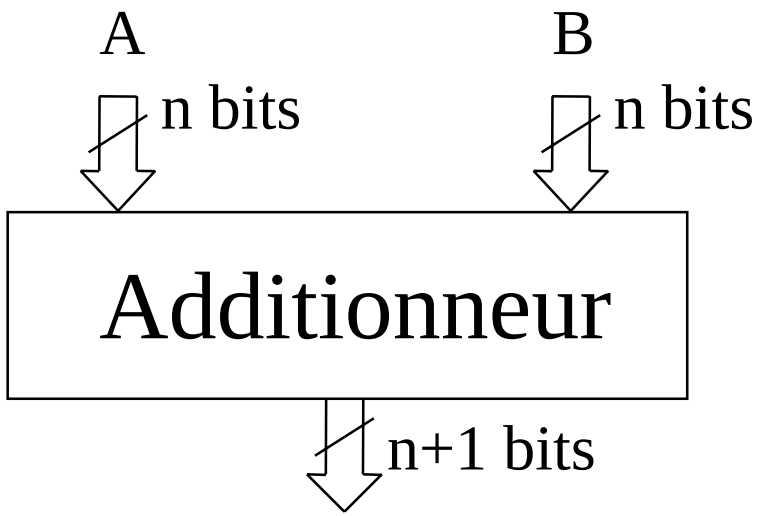
"Full Adder" (FA) à porte de transmission



si $a_i = b_i$ **alors** $c_{i+1} := a_i$, $s_i := \overline{c_i}$

si $a_i \neq b_i$ **alors** $c_{i+1} := c_i$, $s_i := \overline{c_i}$

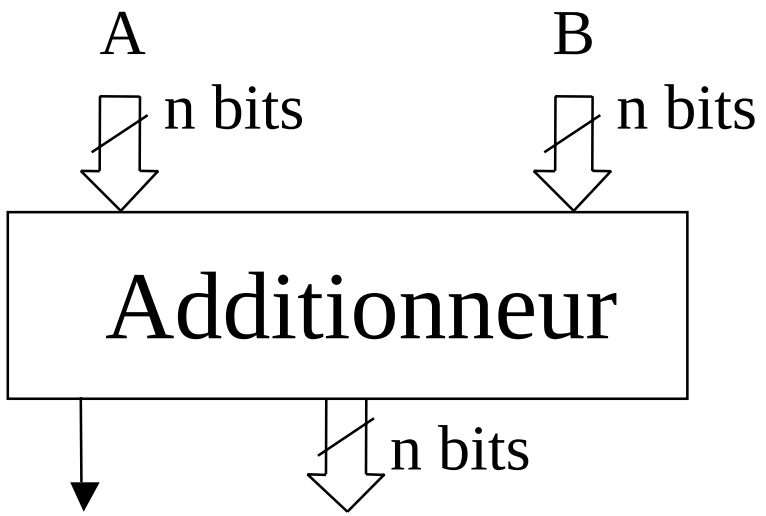
Débordement d'entiers positifs



$S = A + B$

La somme pondérée de ce qui sort est égale à la somme pondérée de ce qui entre

Solution 1



débordement
$S \geq 2^n$, ne tient pas sur n bits

$S \in [0, 2^n - 1]$

$S = (A + B) \text{ modulo } 2^n$

La somme pondérée de ce qui sort n'est pas égale à la somme pondérée de ce qui entre

Solution 2

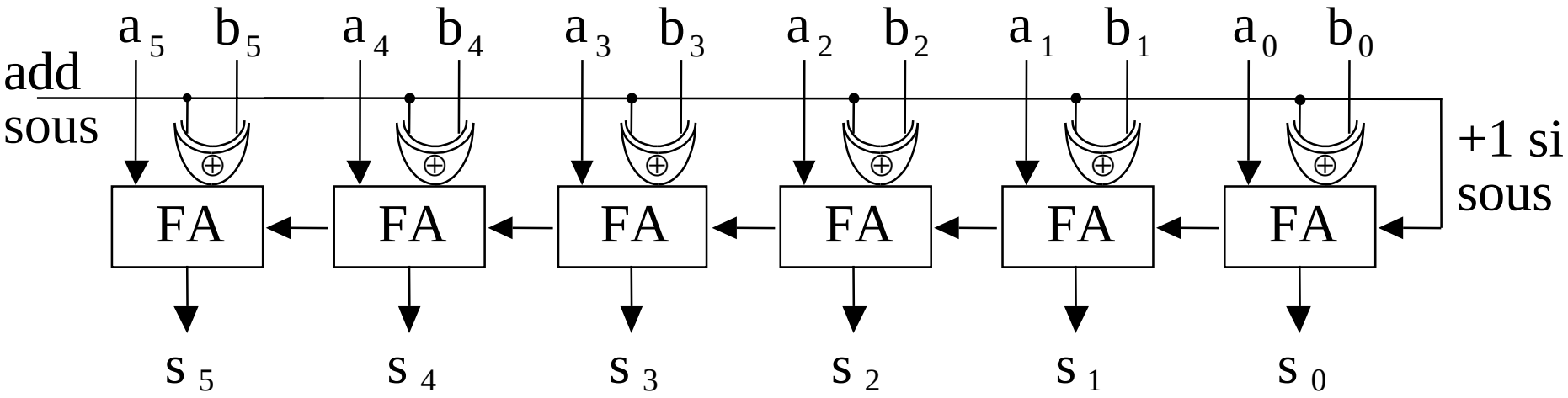
Entiers relatifs

Définition de l'opposé à partir de l'additionneur modulo 2^n

$$B \cong -A \leftrightarrow (A + B) = 0 \ (2^n_{\bmod 2^n})$$

$$A + \overline{A} = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \qquad A + (\overline{A} + 1) = 2^n = 0 \qquad -A = (\overline{A} + 1)$$

Additionneur/soustracteur



$$S = A \pm B$$

Notation des entiers relatifs

a_3	a_2	a_1	a_0	A		
0	0	0	0	0	0	-0+0
0	0	0	1	+1	+1	-0+1
0	0	1	0	+2	+2	-0+2
0	0	1	1	+3	+3	-0+3
0	1	0	0	+4	+4	-0+4
0	1	0	1	+5	+5	-0+5
0	1	1	0	+6	+6	-0+6
0	1	1	1	+7	+7	-0+7
1	0	0	0	-8	-8	-8+0
1	0	0	1	-7	-7	-8+1
1	0	1	0	-6	-6	-8+2
1	0	1	1	-5	-5	-8+3
1	1	0	0	-4	-4	-8+4
1	1	0	1	-3	-3	-8+5
1	1	1	0	-2	-2	-8+6
1	1	1	1	-1	-1	-8+7

est son propre opposé

n'a pas d'opposé

(débordement détectable)

$-A = (\overline{A} + 1)$

$$A = -a_3 2^3 + \sum_{i=0}^{2} a_i 2^i$$

Notation en complément à 2

Dite en complément à 2, en fait à 2^n

Le bit poids fort est négatif, les autres positifs

La valeur d'un nombre est la somme pondérée de ses bits

Le bit poids fort indique le signe du nombre ($0 \Leftrightarrow \geq 0$, $1 \Leftrightarrow < 0$)

Alors 0 est positif

Le plus grand nombre négatif n'a pas d'opposé

Le changement de signe provoque une propagation de retenue

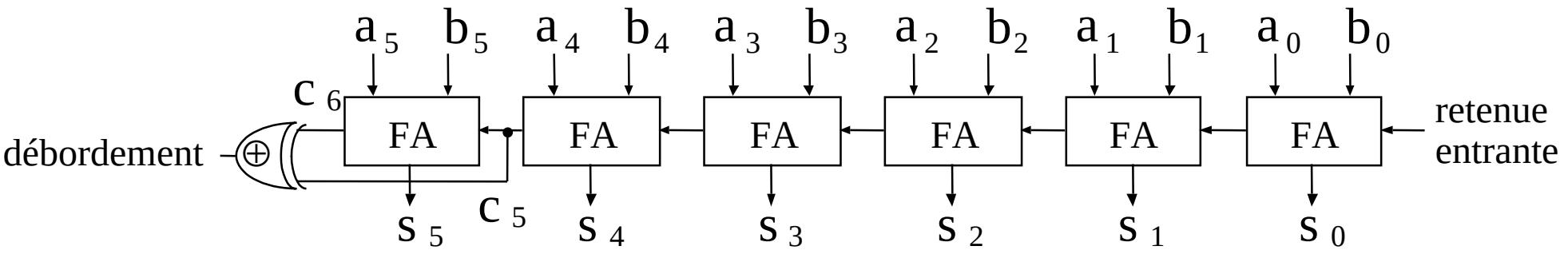
Il y a d'autres systèmes (peu usités)

Débordement de l'addition d'entiers relatifs

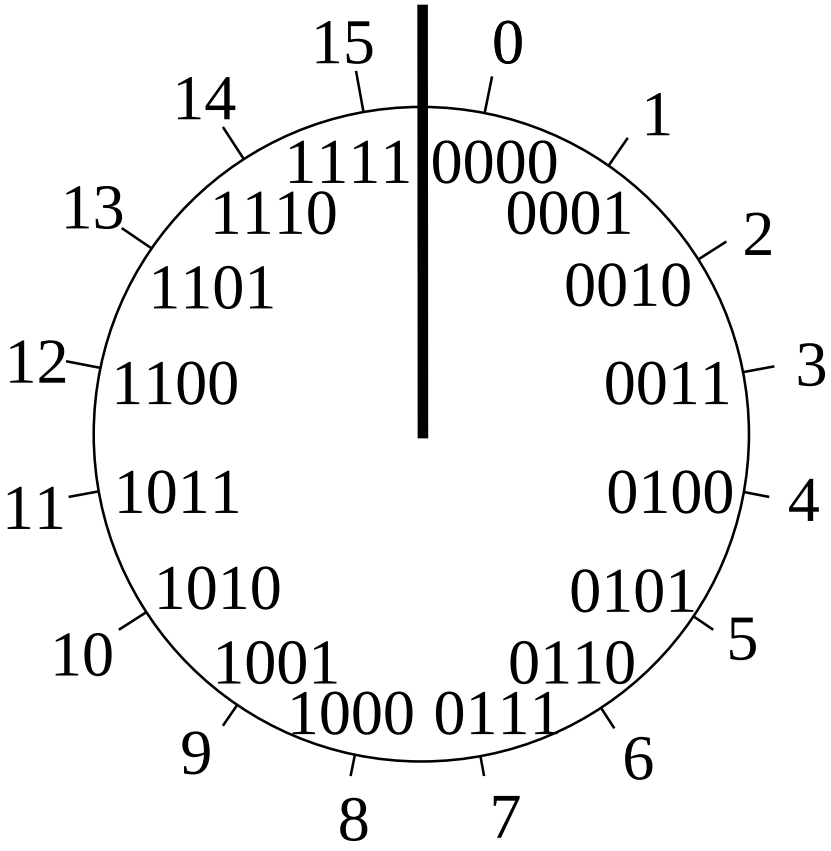
A	B	S=A+B
+	+	+
+	+	-
+	-	+
+	-	-
-	+	+
-	+	-
-	-	+
-	-	-

cas de débordement

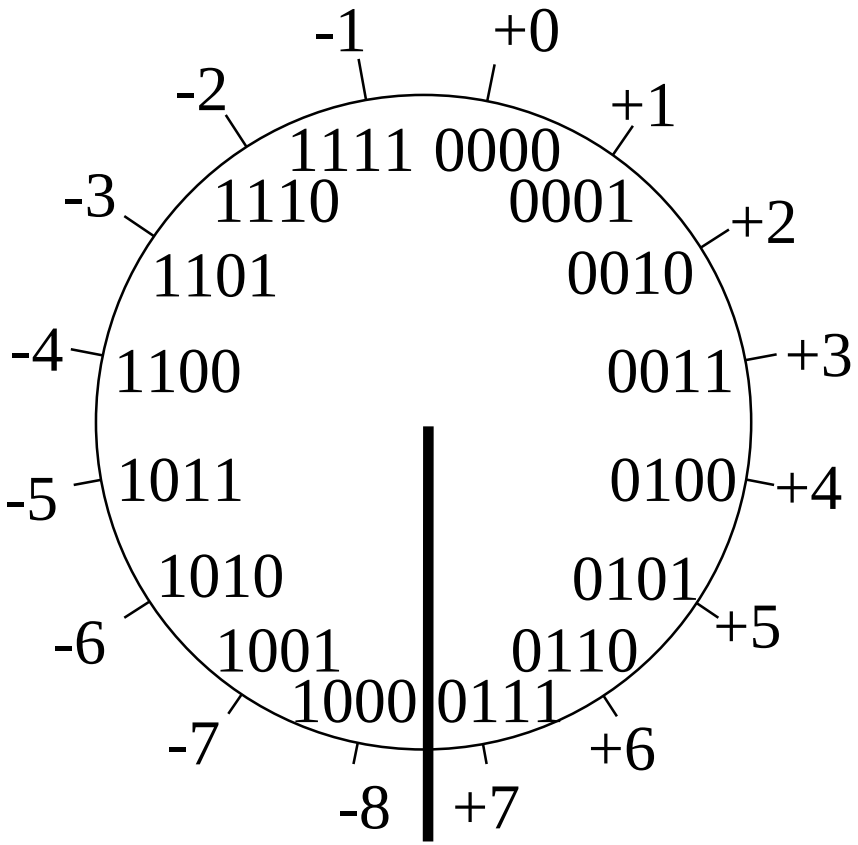
a_{n-1}	b_{n-1}	c_{n-1}		c_n	s_{n-1}
+	+	0	=	0	+
+	+	1	≠	0	-
+	-	0	=	0	-
+	-	1	=	1	+
-	+	0	=	0	-
-	+	1	=	1	+
-	-	0	≠	1	+
-	-	1	=	1	-



Débordement (exemple sur 4 bits)



Entiers positifs



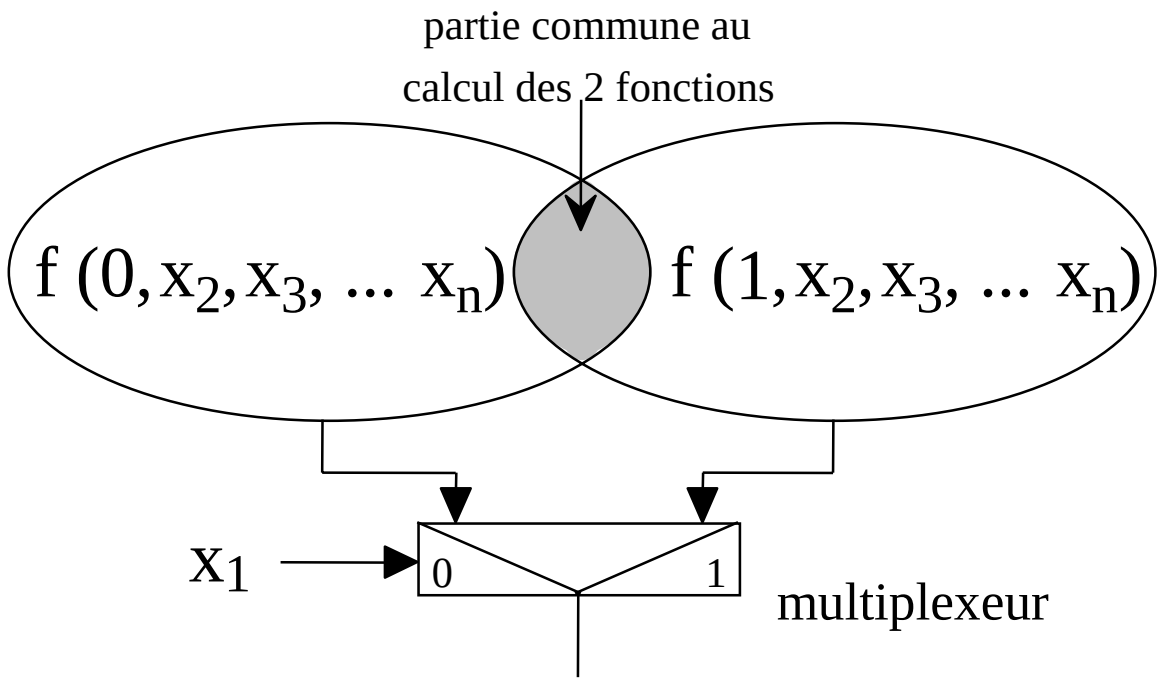
Entiers relatifs

Écriture de Shannon

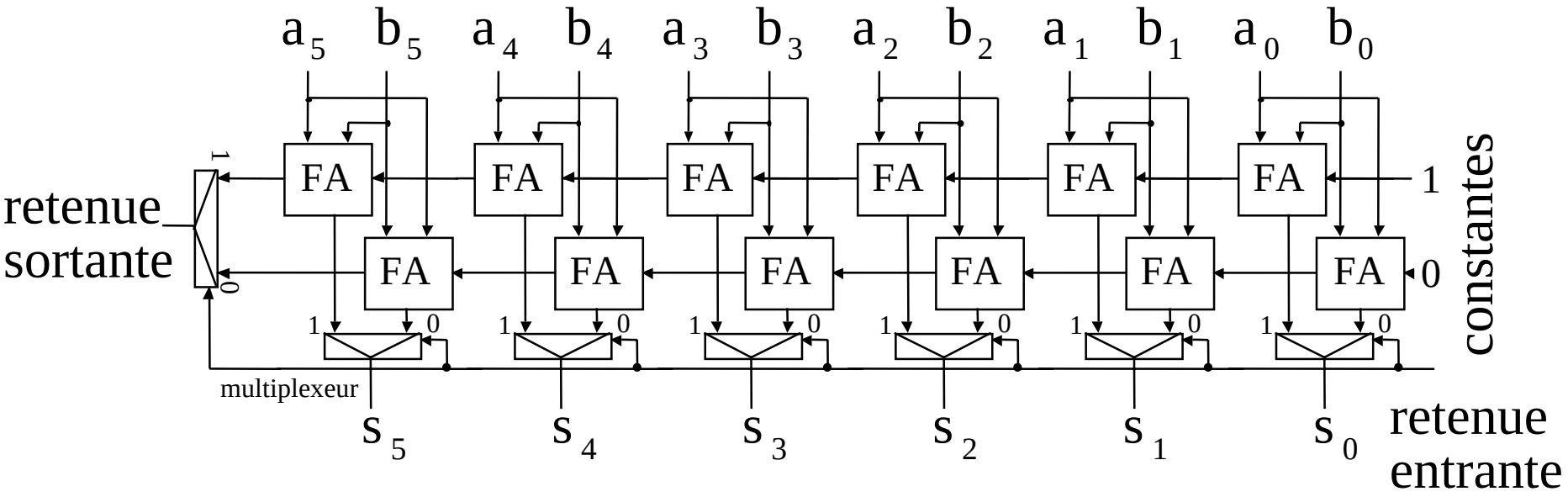
On a à réaliser $f(x_1, x_2, x_3, \ldots x_n)$

on veut disposer de temps pour calculer x_1

$\Rightarrow$ on précalcule $f(0, x_2, x_3, \ldots x_n)$ et $f(1, x_2, x_3, \ldots x_n)$



“Carry select adder”

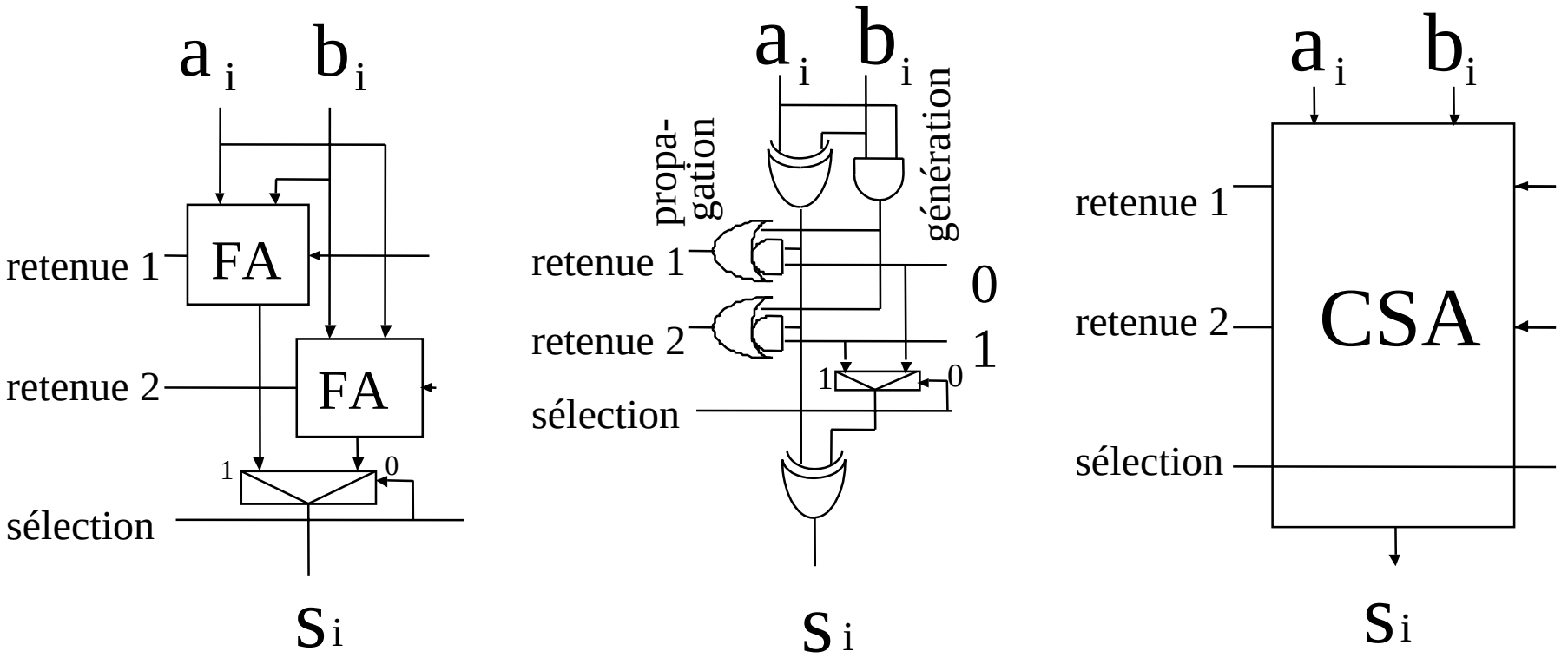


La retenue entrante ne se propage pas à travers les FA

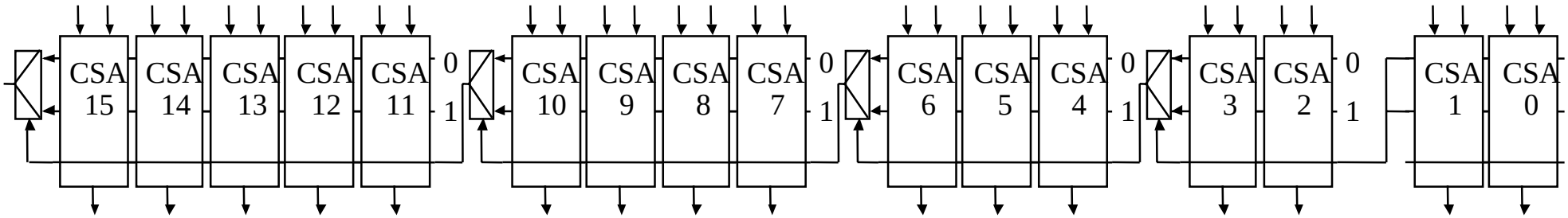
⇒ on dispose de temps pour la calculer

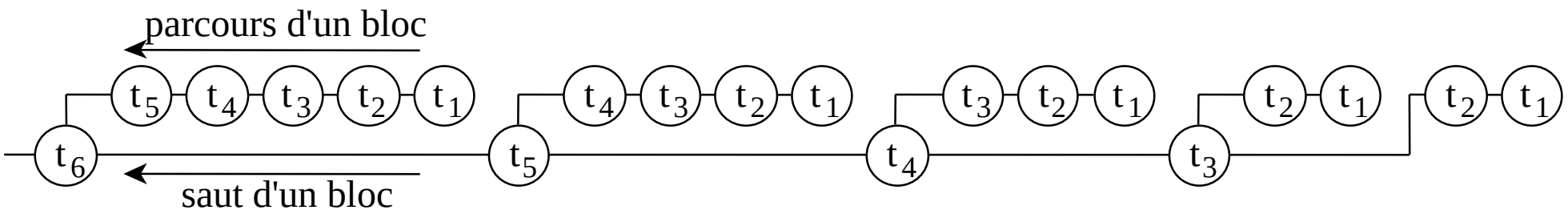
⇒ à mettre en poids forts la où la retenue est en retard

Cellule de carry select adder



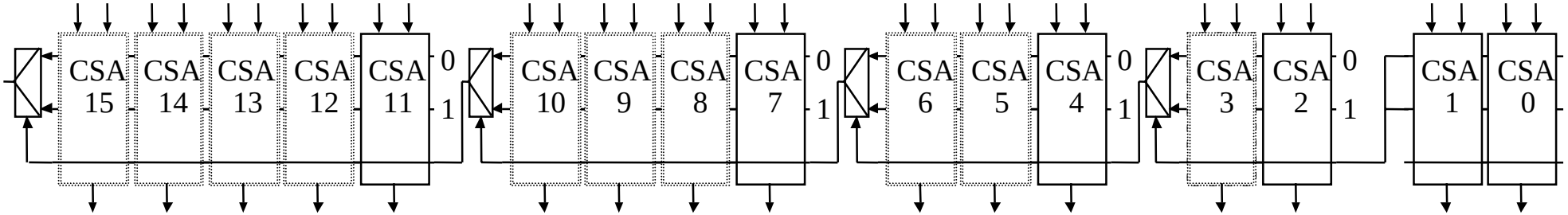
Additionneur en temps $\sqrt{\ }\overline{n}$

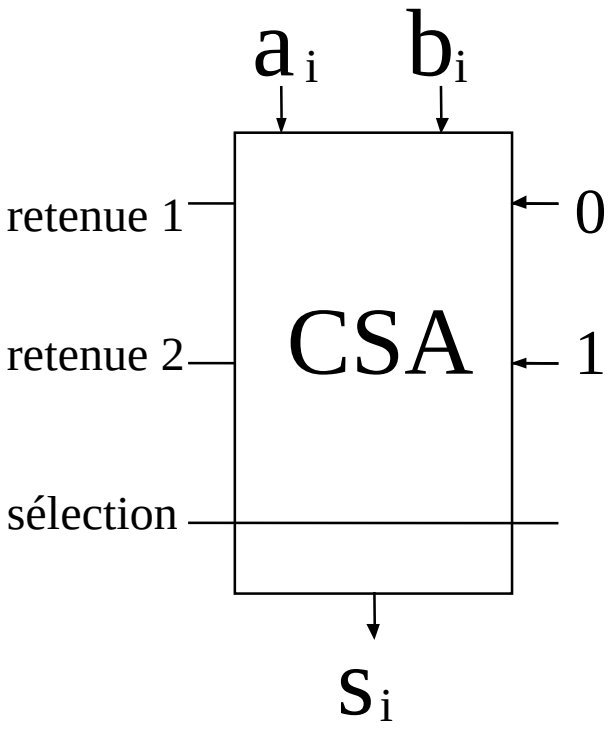


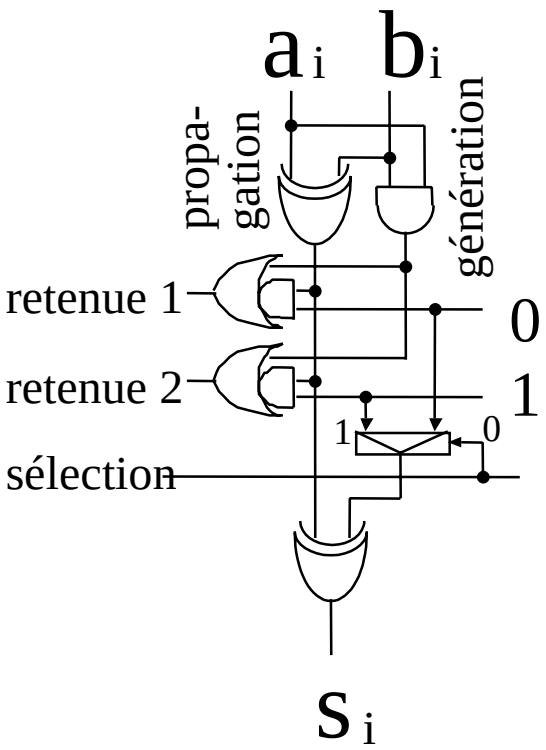


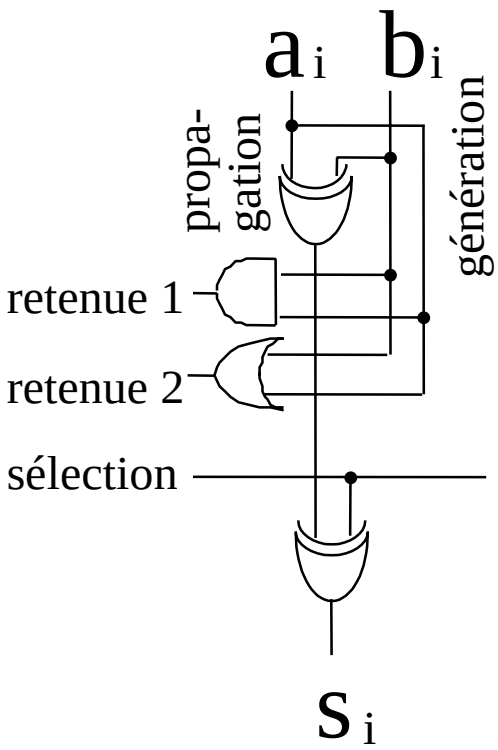
$$n = \sum_{i=0}^{\tau-2} i = \frac{\tau(\tau+1)}{2} + 1 \qquad 2n = \tau^2 + \tau + 2 \Rightarrow \tau = \frac{\sqrt{8n-7}-1}{2} \approx \sqrt{2n} \approx 1{,}4\sqrt{n}$$

Première cellule d'un bloc de CSA









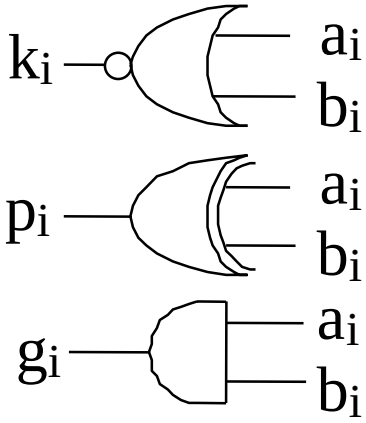
Génération et propagation de la retenue

a_i	b_i	c_{i+1}
0	0	0
0	1	c_i
1	0	c_i
1	1	1

k_i (Absorption) $k_i = \overline{a_i \vee b_i}$

p_i (Propagation) $p_i = a_i \oplus b_i$

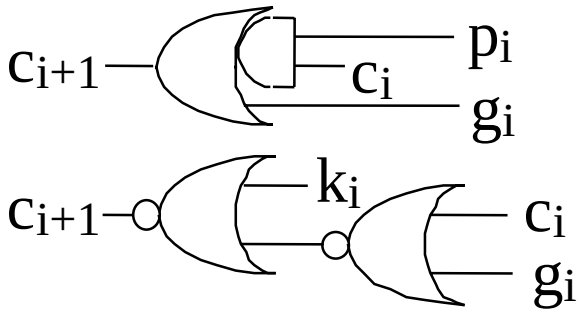
g_i (Génération) $g_i = a_i \wedge b_i$



A	1	0	1	0	0	0	1	0	1	0	1	0	1	1	0	0
+B	0	1	1	0	0	1	0	1	1	0	1	0	0	0	1	0
	p	p	g	k	k	p	p	p	g	k	g	k	p	p	p	k
C	1	1	1	0	0	1	1	1	1	0	1	0	0	0	0	

$c_{i+1} = (p_i \wedge c_i) \vee g_i = (\overline{k_i} \wedge c_i) \vee g_i$

$\overline{c_{i+1}} = (p_i \wedge \overline{c_i}) \vee k_i = (g_i \wedge \overline{c_i}) \vee k_i$



Génération et propagation de la retenue (2)

Définitions

$p_i = a_i \oplus b_i$ propagation: $c_{i+1} = c_i$

$g_i = a_i \wedge b_i$ génération: $c_{i+1} = 1$

$k_i = \overline{a_i} \wedge \overline{b_i}$ absorption: $c_{i+1} = 0$

$\mathbf{P}_{i,j} = \prod_{n=i}^{j} p_n$ propagation: $c_{i+1} = c_j \quad i \geq j$

$\mathbf{G}_{i,j} = g_i \vee \sum_{n=i+1}^{j} (\mathbf{P}_{i,n} \wedge g_n)$ $c_{i+1} = 1$

$\mathbf{K}_{i,j} = k_i \vee \sum_{n=i+1}^{j} (\mathbf{P}_{i,n} \wedge k_n)$ $c_{i+1} = 0$

$\mathbf{P}_{i,j}$ = la retenue a été propagée du rang j au rang i+1

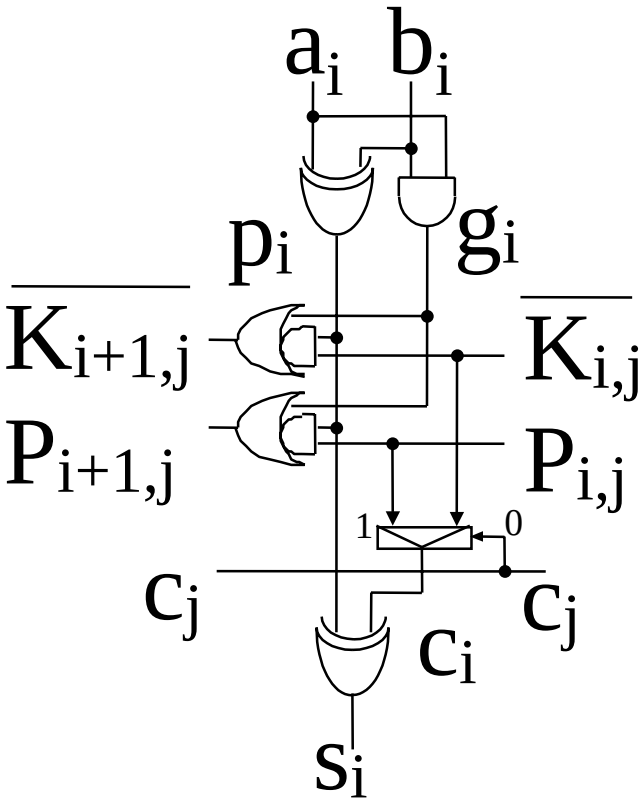
$\mathbf{G}_{i,j}$ = la retenue a été générée entre i et j et propagée jusqu'à i+1

$\mathbf{K}_{i,j}$ = la retenue a été absorbée entre i et j et propagée jusqu'à i+1

Exercice

Simplification du CSA

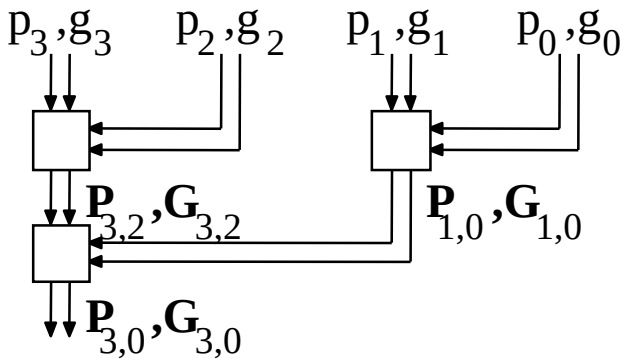
1- Simplifier les équations logiques de la cellule de “Carry-Select Adder” en exprimant la relation entre $P_{i,j}$ et $K_{i,j}$

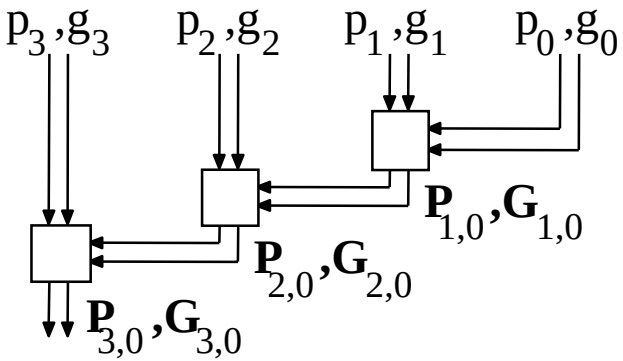


Génération et propagation de groupe

Définissons g_i, p_i, $\mathbf{G}_{i,j}$ et $\mathbf{P}_{i,j}$ de la façon récursive suivante :

$\mathbf{G}_{i,i} = g_i = a_i \wedge b_i$	*génération de retenue au rang i*
$\mathbf{P}_{i,i} = p_i = a_i \oplus b_i$	*propagation de retenue au rang i*
$\mathbf{G}_{i,k} = \mathbf{G}_{i,j} \vee \mathbf{P}_{i,j} \wedge \mathbf{G}_{j-1,k}$	*génération de retenue entre le rang i et le rang k* $(n \geq i \geq j > k \geq 0)$
$\mathbf{P}_{i,k} = \mathbf{P}_{i,j} \wedge \mathbf{P}_{j-1,k}$	*propagation de la retenue du rang k au rang i*
$c_{i+1} = \mathbf{G}_{i,0} \vee \mathbf{P}_{i,0} \wedge c_0$	*retenue au rang i+1, ce que l'on cherche à obtenir*



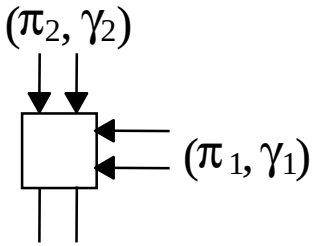


Propriété: La cellule est associative.

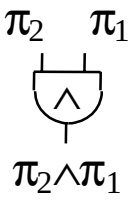
Conséquence: De nombreux assemblages sont possibles

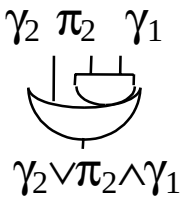
Règle d'assemblage: Toute sortie de rang i dépends des entrées de rang 0 à i.

Cellule de Brent et Kung pour calculer les "propagation de groupe" et "génération de groupe"

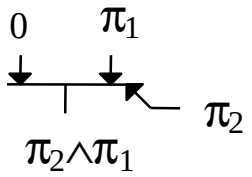


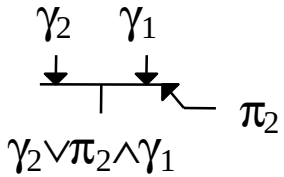
$(\pi_2 \wedge \pi_1 , \gamma_2 \vee \pi_2 \wedge \gamma_1)$





Réalisation à portes logiques

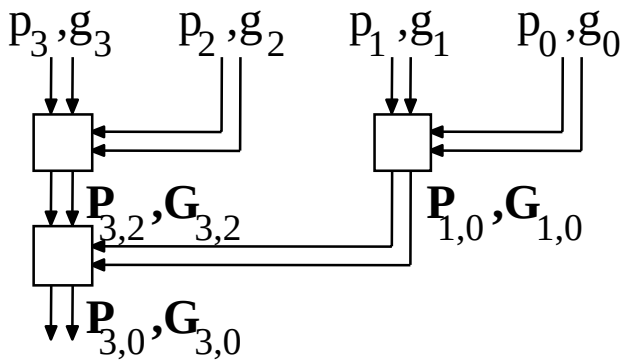


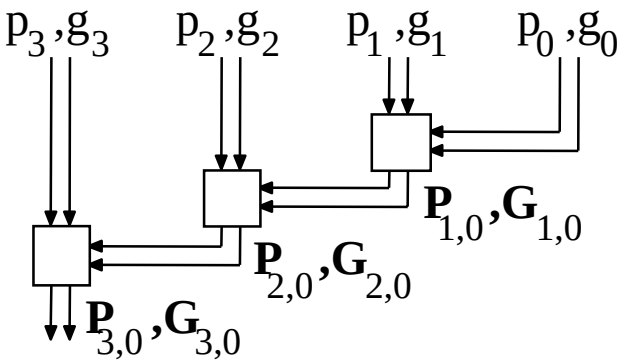


Réalisation à multiplexeurs

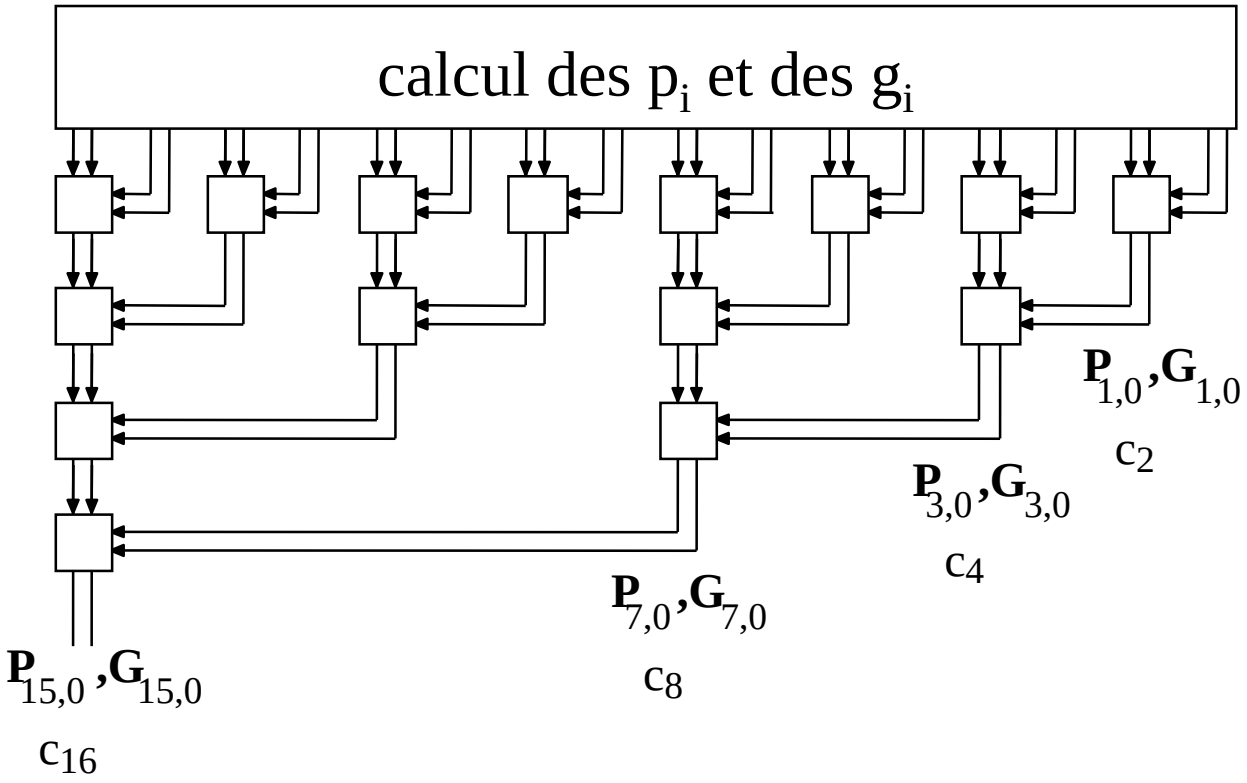
- <u>Associative</u>
- Non commutative
- Idempotente
- Non décroissante (inverseurs)

$$\mathbf{G}_{i,k} = \mathbf{G}_{i,j} \vee \mathbf{P}_{i,j} \wedge \mathbf{G}_{j-1,k}$$
$$\mathbf{P}_{i,k} = \mathbf{P}_{i,j} \wedge \mathbf{P}_{j-1,k}$$





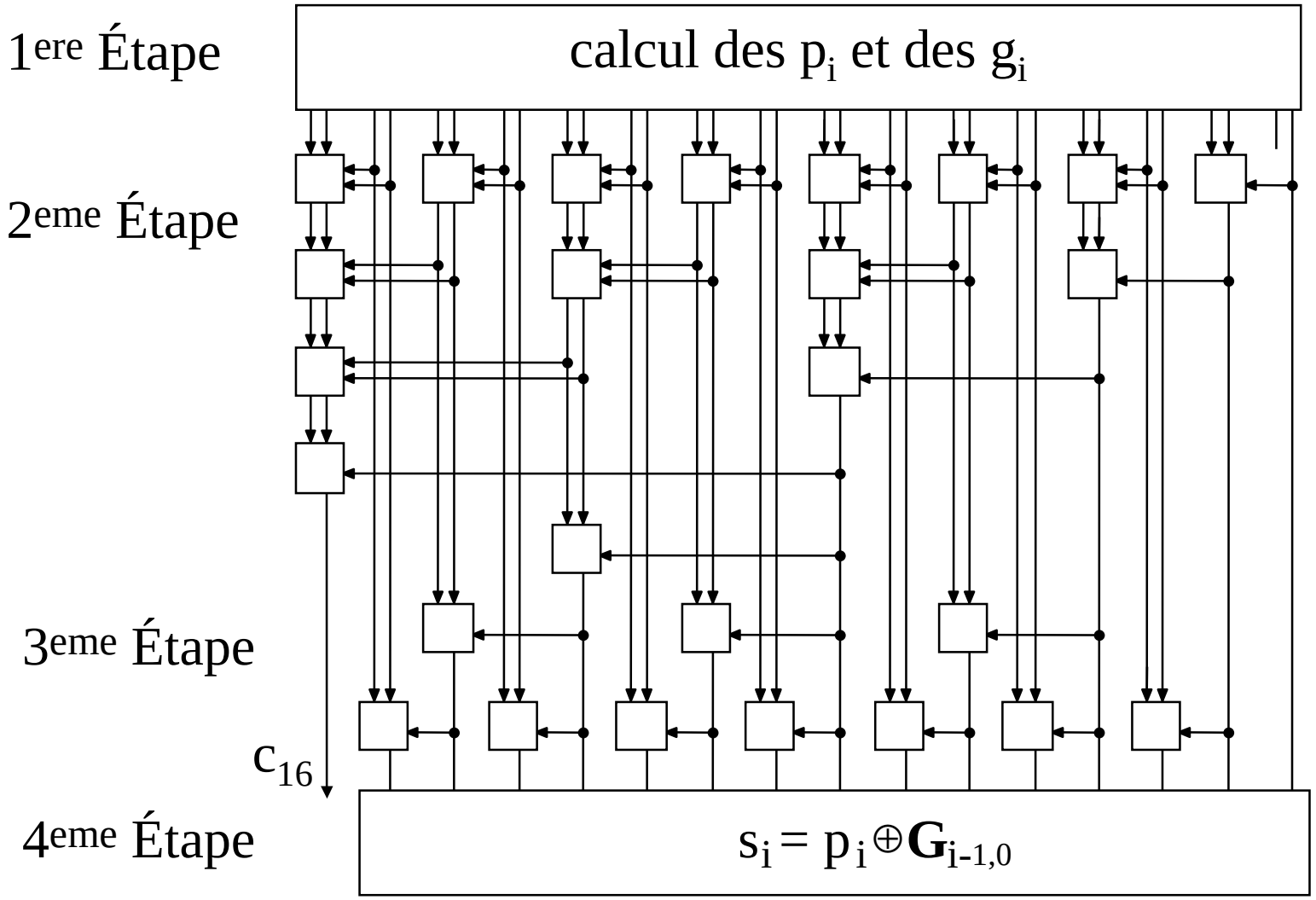
Tous les additionneurs en temps $\log_2(n)$ commence par un calcul arborescent



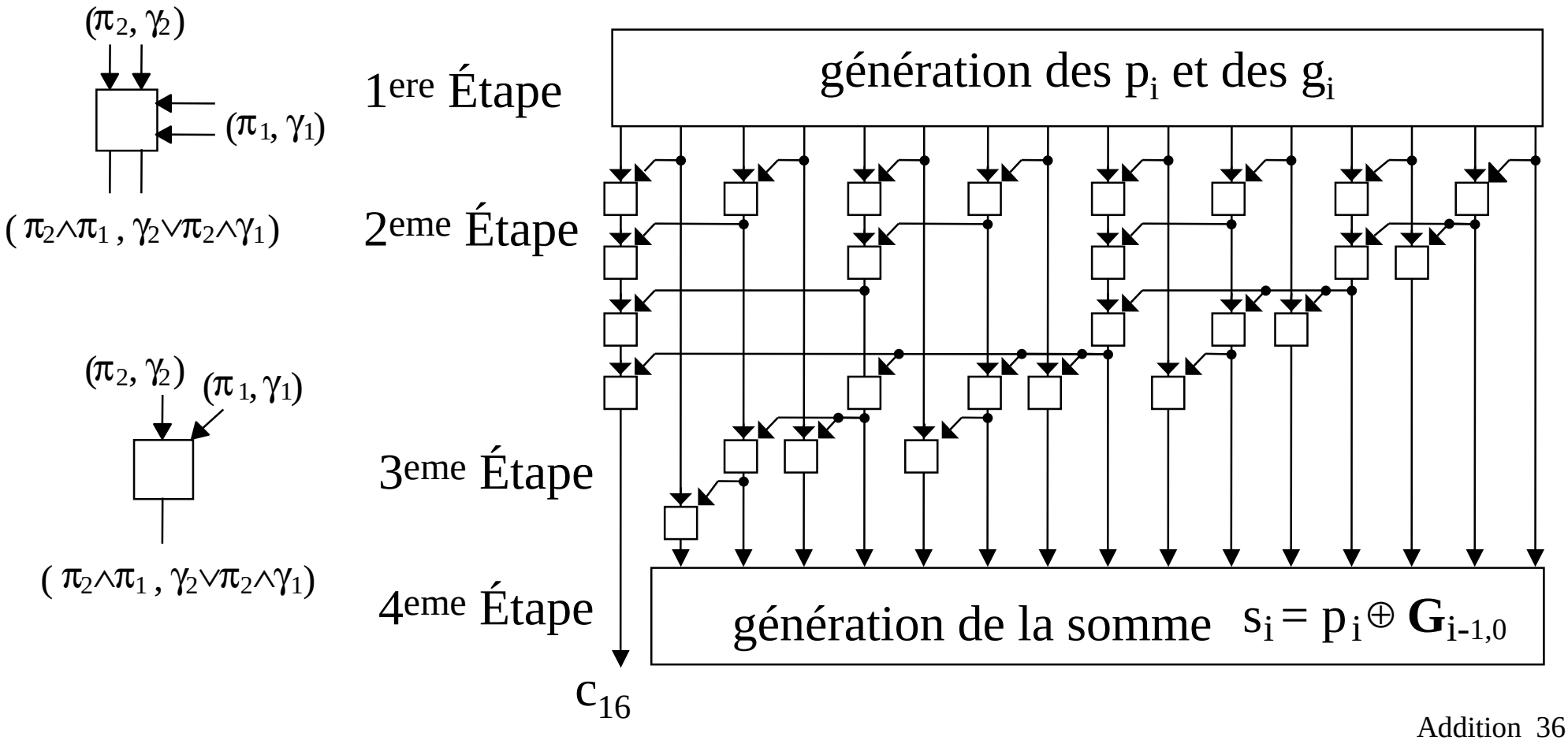
Le calcul des autres retenues est:

- 1 coût minimum
- 2 rapide à “fan out” variable
- 3 rapide à fan-out fixe

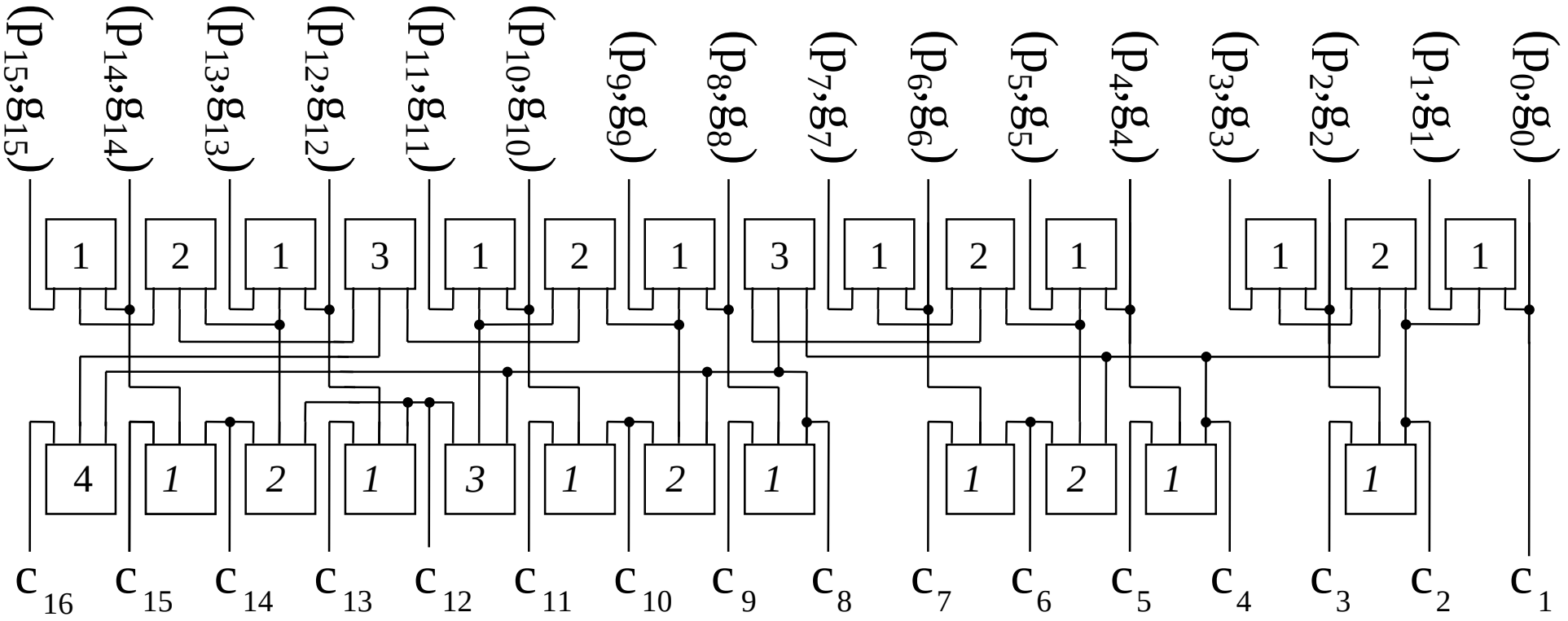
Additionneur de Brent et Kung en temps $\log_2(n)$



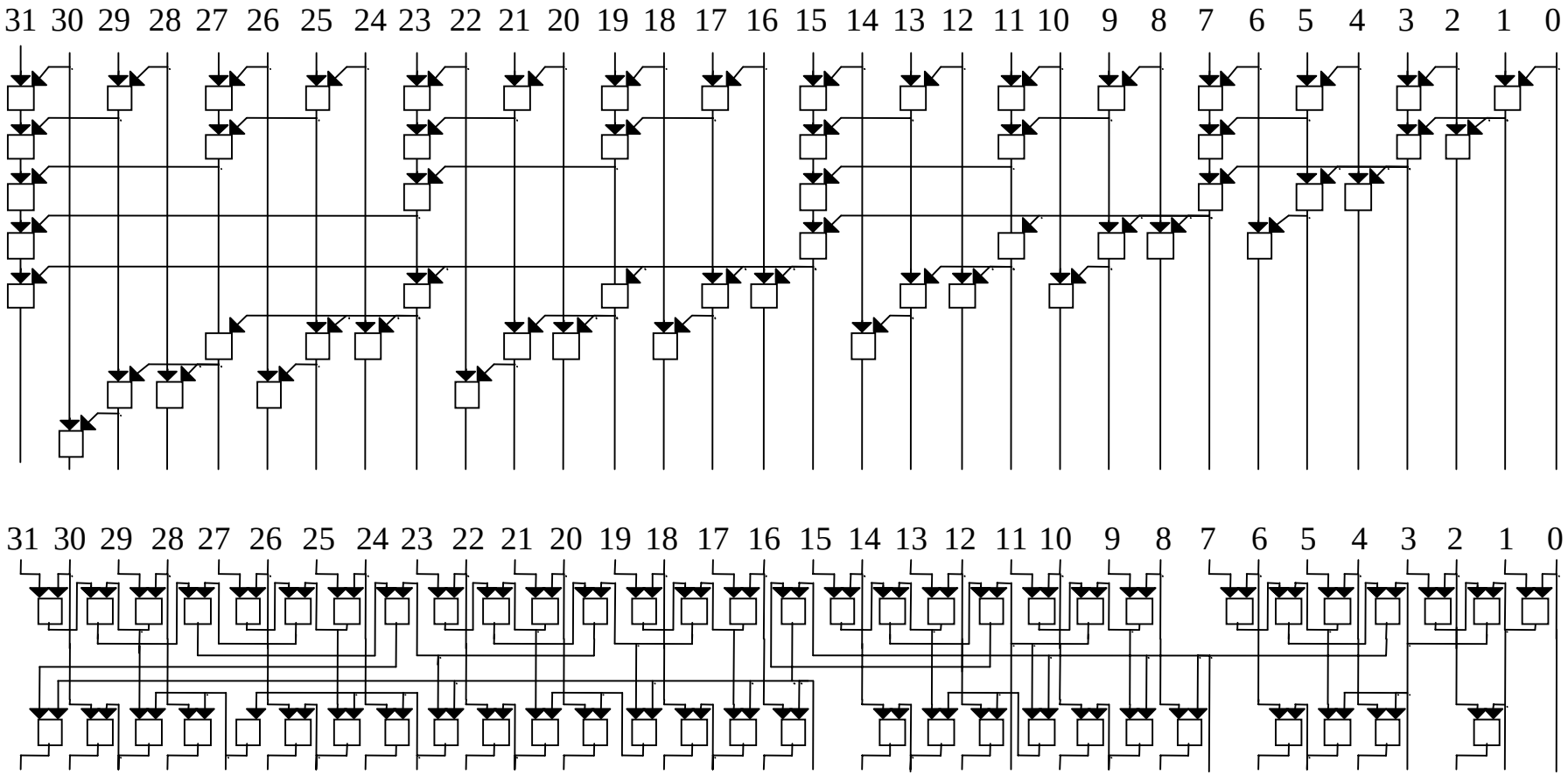
Additionneur de Brent et Kung en temps $\log_2(n)$ (2)



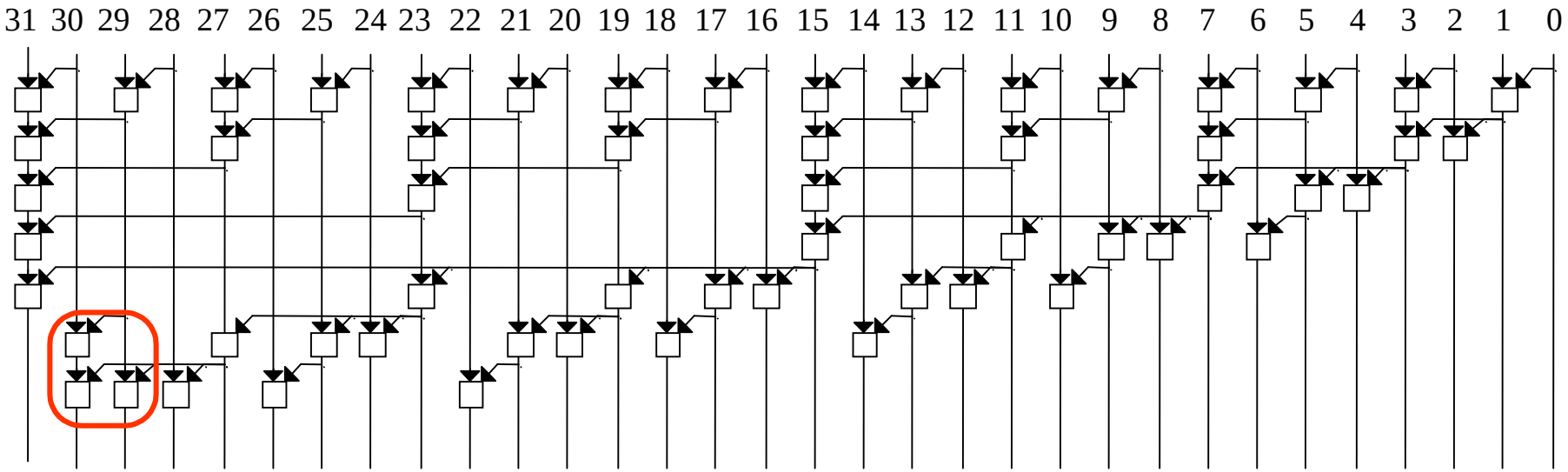
Mise à plat des deux arbres binaires (16 bits)



Additionneur de Brent et Kung



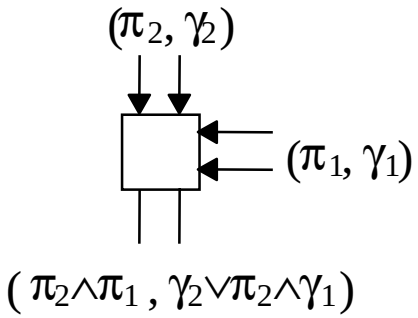
Additionneur de Brent et Kung modifié

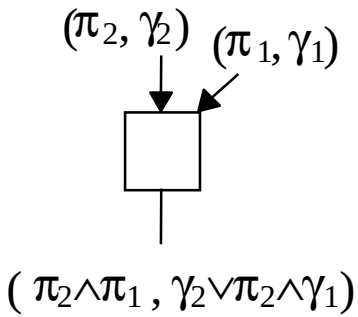


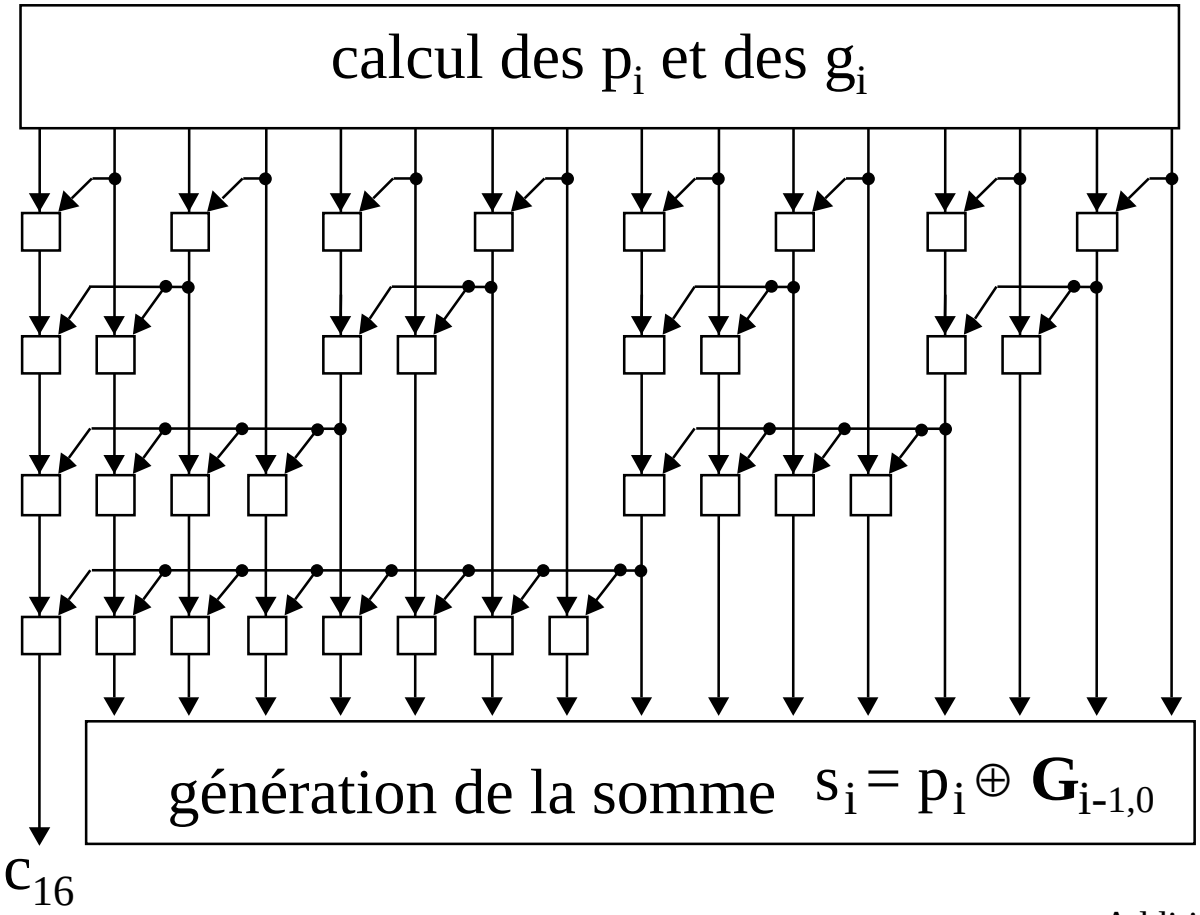
Modification de J.P. Fishburn (1990)

Une cellule de plus (+ 2%) décroît le chemin critique de 8 à 7 cellules

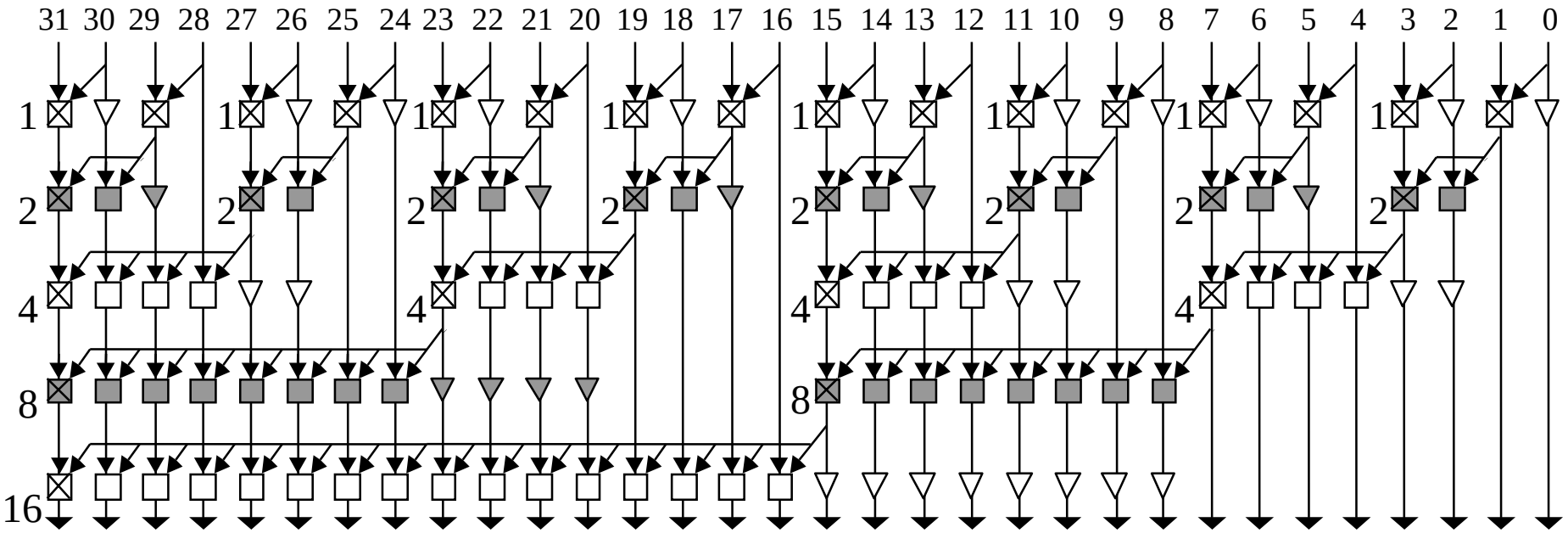
Additionneur de Sklansky en temps $\log_2(n)$

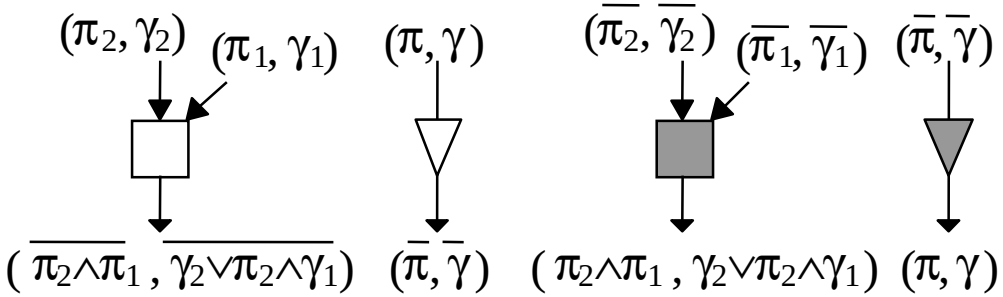




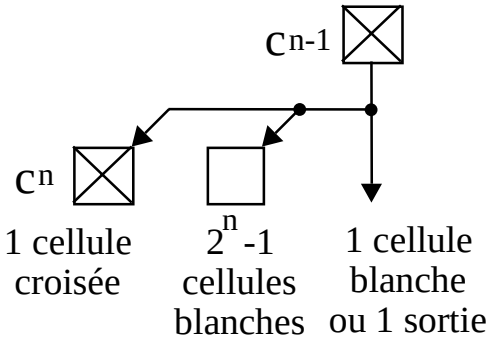


Prise en compte des aspects électriques





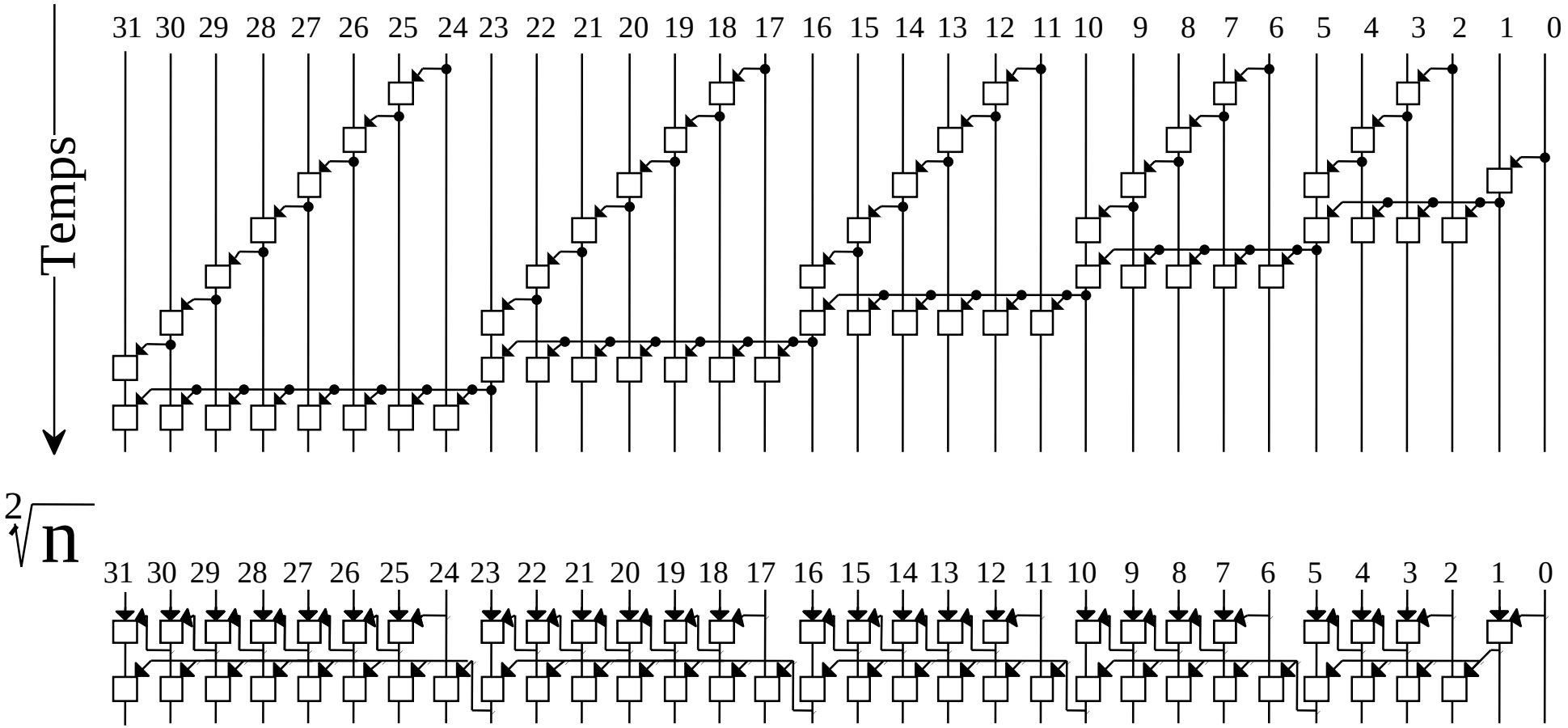
Prise en compte des inversions



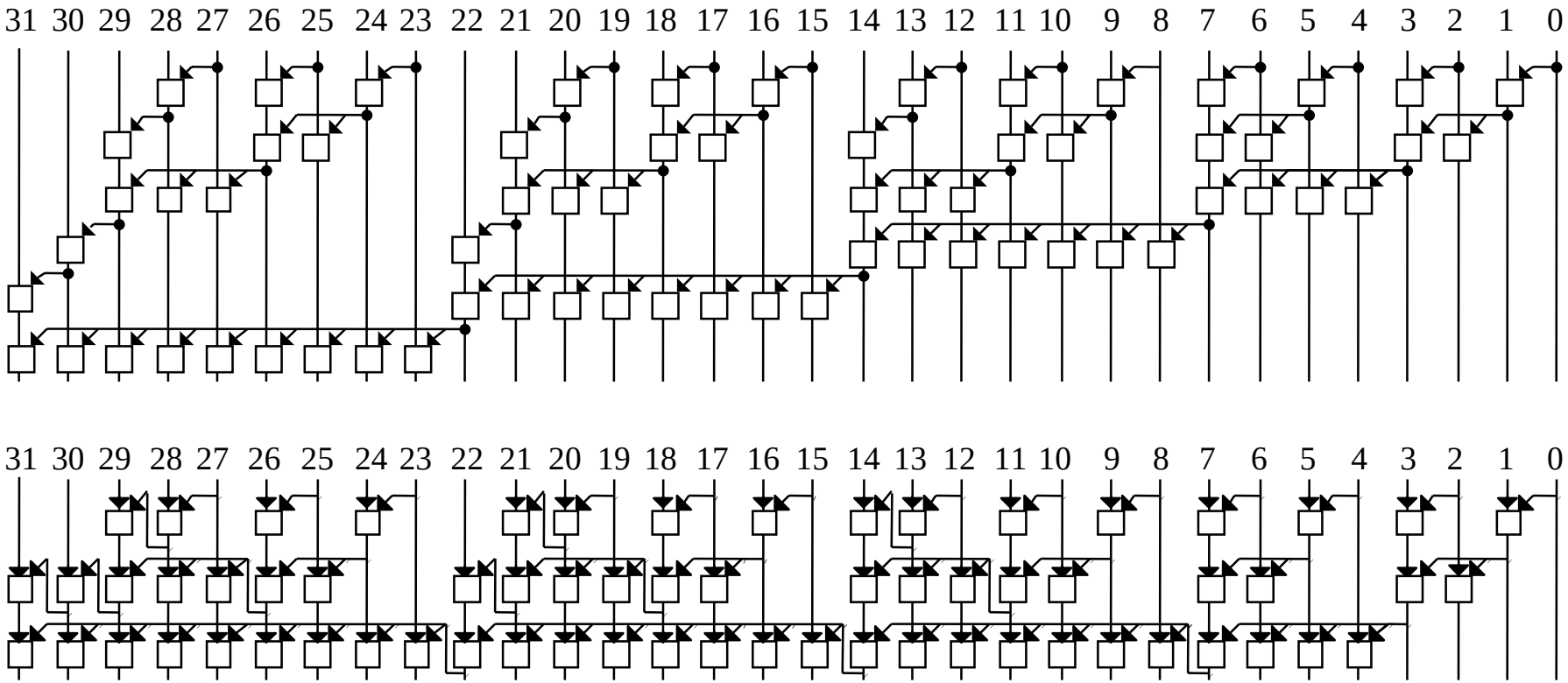
$C_{n-1} = k\,(C_n + 2^n - 1 + 1)$

dimensionnement

Additionneur en temps $\sqrt{n}$

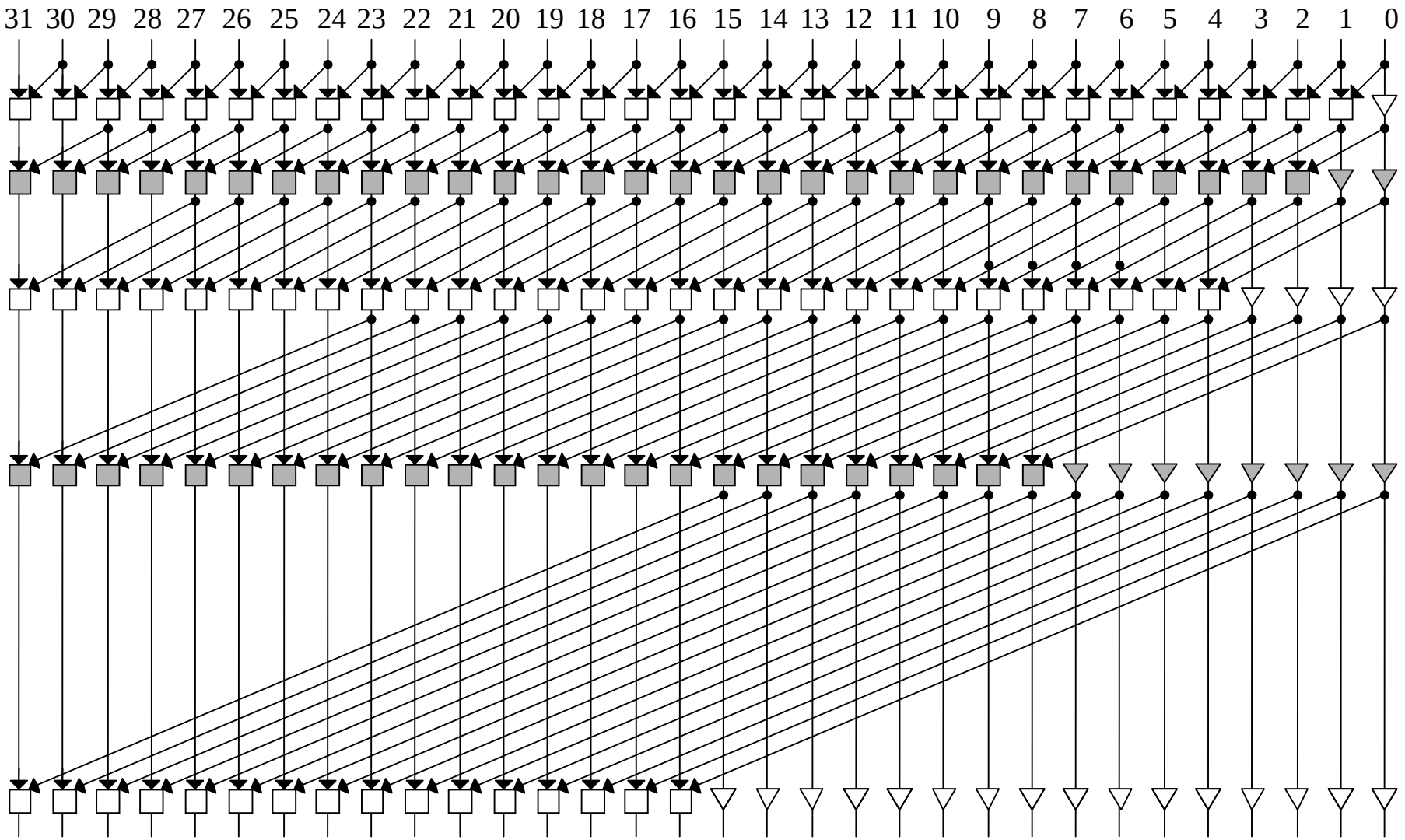


Additionneur en temps $\sqrt[3]{n}$



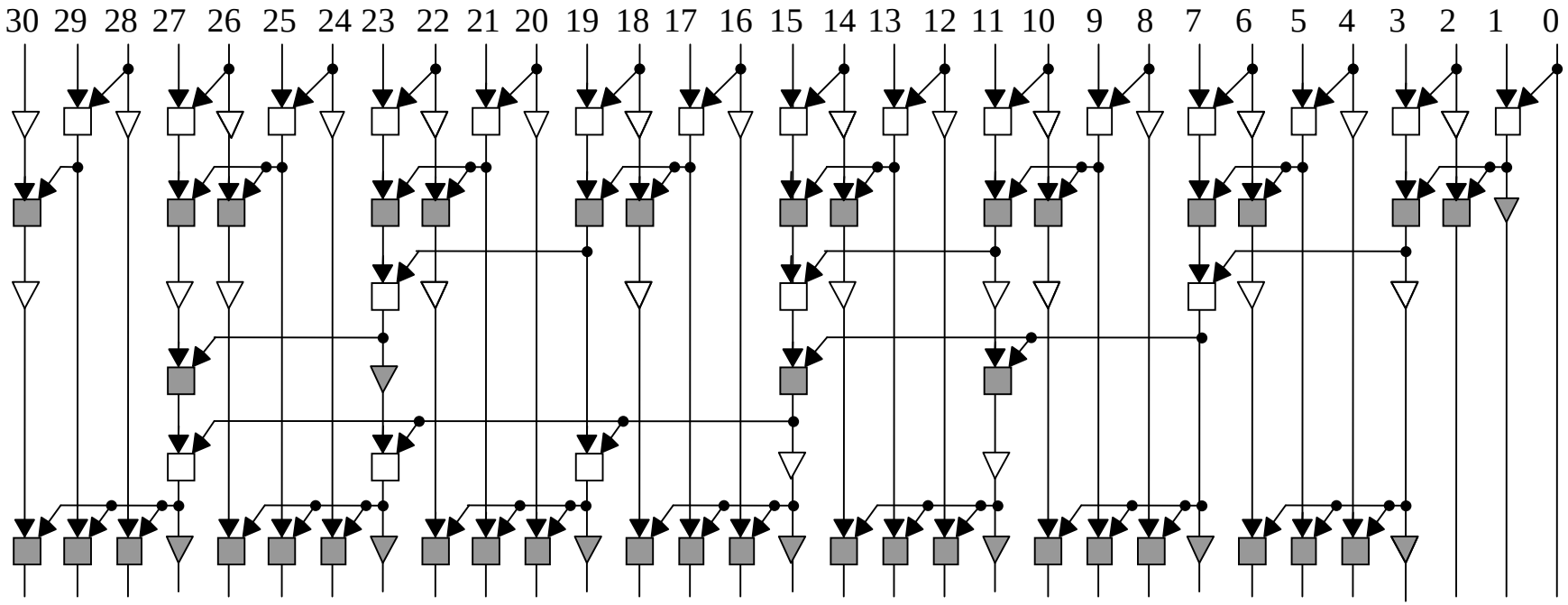
Pour un délai τ le nombre n de bits est : $\sum_{i=1}^{\tau} \sum_{j=1}^{i} j \Rightarrow \quad \tau = \sqrt[3]{n}$

Additionneur de Kogge et Stone



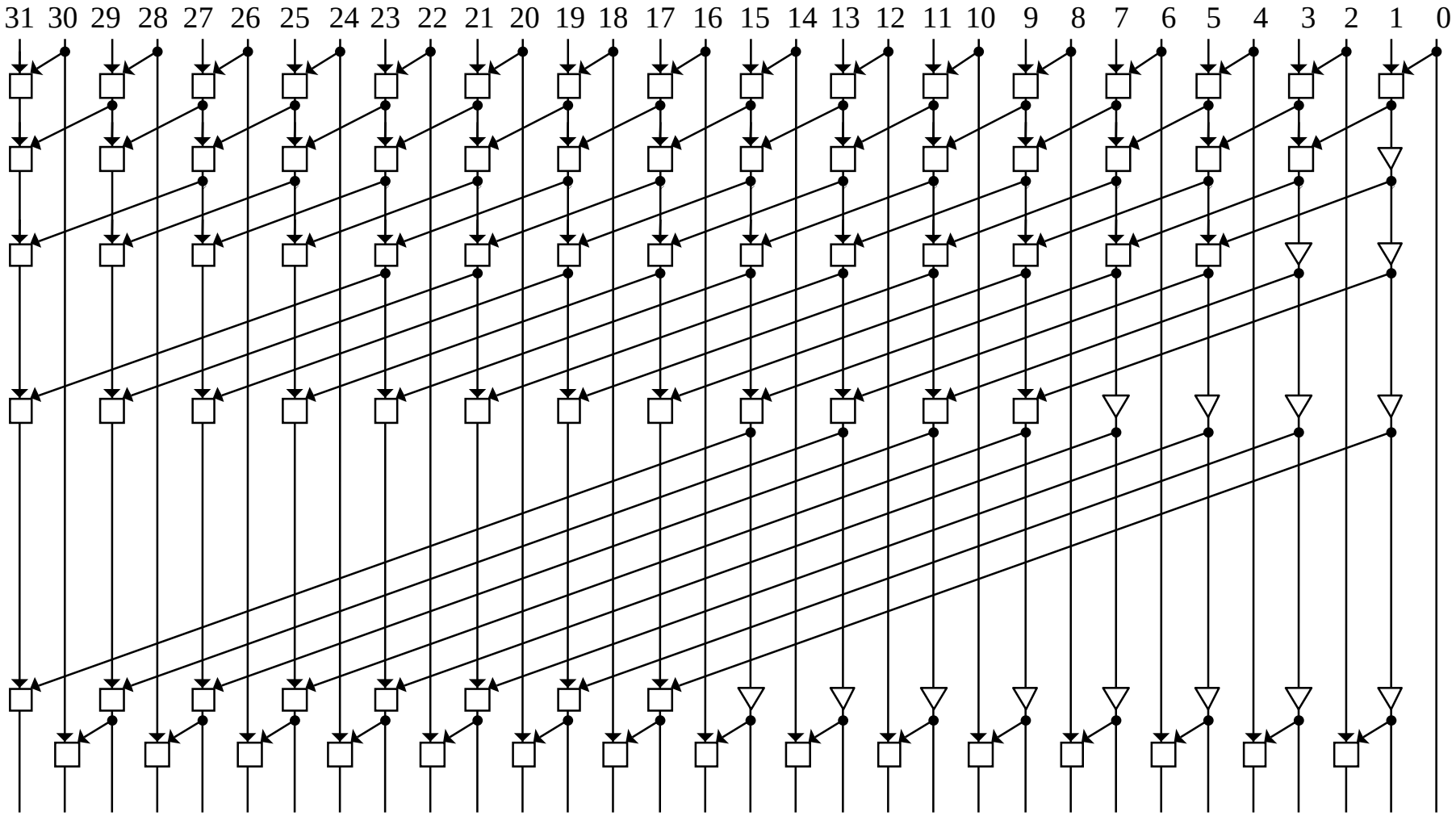
Le "fan-out" est toujours 2

Additionneur hybride
(arbre de Brent & Kung / Carry Select)



Pendant que l'arbre calcule 1 retenue sur 4,
les CSA propagent la retenue sur 4 positions.

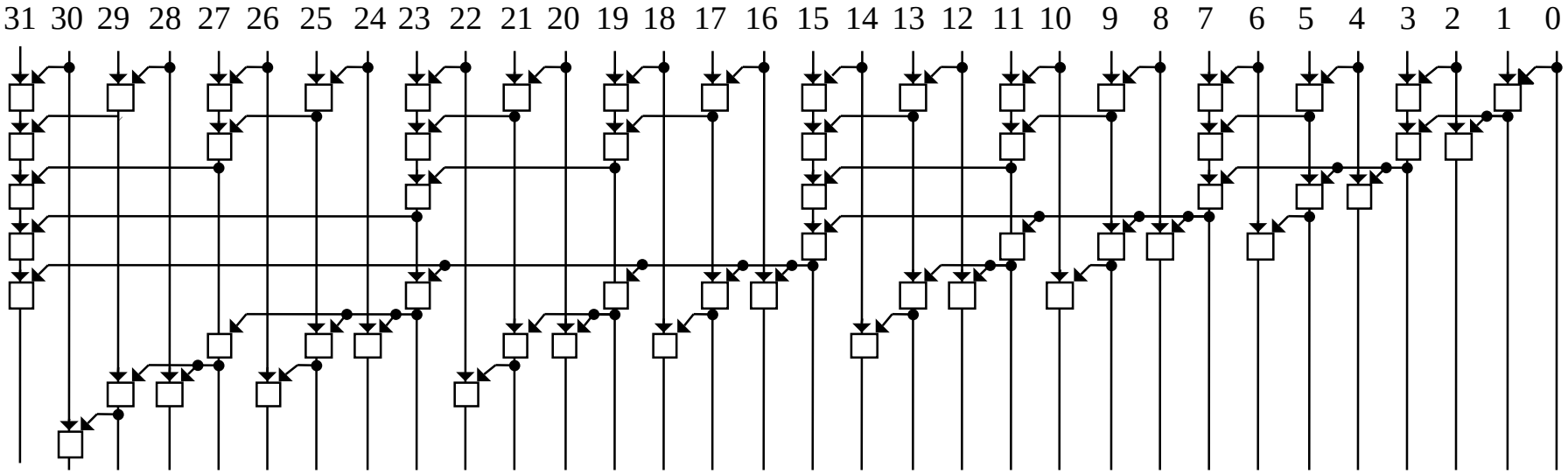
Additionneur de Han et Carlson



Résumé sur les additionneurs à cellule de Brent et Kung (Δ -cell)

Type d'addition	# de Δ-cells	Délai (Δ-cell)	Max. fan-out	Exemple n = 32 bits		
Propagation	$n-1$	$n-1$	2	31	31	2
2-level carry select	$\lceil 2n-\sqrt{2n} \rceil$	$\lceil \sqrt{2n} \rceil$	$\lceil \sqrt{2n} \rceil$	54	8	6
3-level carry select	$3n$??	$\lceil \sqrt[3]{6n} \rceil$	$\lceil \sqrt[3]{6n} \rceil$	66	6	9
Brent-Kung	$\lceil 2n- \log_2(n) \rceil$	$\lceil 2 \log_2(n) -2 \rceil$	$\lceil 2 \log_2(n) -2 \rceil$	57	8	5
Variante du BK	ci-dessus +1	ci-dessus -1	$\lceil 2 \log_2(n) -2 \rceil$	58	7	5
Sklanski	$\lceil n/2 \log_2(n) \rceil$	$\lceil \log_2(n) \rceil$	$n/2$	80	5	16
Kogge and Stone	$\lceil n (\log_2(n)-1) \rceil$	$\lceil \log_2(n) \rceil$	2	129	5	2
Han and Carlson	$\lceil n/2 \log_2(n) \rceil$	$\lceil \log_2(n) \rceil +1$	2	80	6	2
Hybrid CS-VN	$\lceil 2.5n -\sqrt{2n} \rceil$	$\lceil 1+\sqrt{n} \rceil$	$n/2$	65	6	16

Exercice



1- De quel type est cet additionneur ?

2- Quel est le délai (en nombre de cellules sur le chemin critique) de cet additionneur ?

3- Proposer une modification (ajout de une cellule) qui fasse décroître la longueur du chemin critique (de une cellule)

Additionneur de Ling

Un des additionneurs les plus rapides a été décrit par H. Ling, d'IBM, en 1981. La contribution de Ling est la "pseudo retenue". Prenons par exemple le calcul du 6^{eme} bit de la somme.

$s_6 = p_6 \oplus \mathbf{G}_{5,0}$ ($\mathbf{G}_{i,0}$ est la retenue au rang i+1)

$\mathbf{G}_{5,0} = \mathbf{G}_{5,3} \vee \mathbf{P}_{5,3}\, \mathbf{G}_{2,0}$

$\mathbf{G}_{5,3} = g_5 \vee \overline{k_5}\, g_4 \vee \overline{k_5}\ \overline{k_4}\, g_3$

$\mathbf{G}_{2,0} = g_2 \vee \overline{k_2}\, g_1 \vee \overline{k_2}\ \overline{k_1}\, g_0$

$\mathbf{P}_{5,3} = \overline{k_5}\ \overline{k_4}\ \overline{k_3}$

Le délai de s_6 est déterminé par $\mathbf{G}_{5,0}$ Tous les termes de $\mathbf{G}_{5,0}$ contiennent $\overline{k_5}$ sauf le premier qui est g_5. $\mathbf{G}_{5,0}$ peut être simplifié en notant que $g_i = \overline{k_i} \wedge g_i$. Si on remplace g_5 par $\overline{k_5} \wedge g_5$ on peut mettre $\overline{k_5}$ en facteur : $s_6 = p_6 \oplus (\overline{k_5}\, \boldsymbol{G}_{5,0})$. On note $\boldsymbol{G}_{i,0}$ la pseudo retenue au rang i+1.

$\boldsymbol{G}_{5,0} = \boldsymbol{G}_{5,3} \vee \boldsymbol{P}_{5,3}\, \boldsymbol{G}_{2,0}$. (se vérifie à partir de $\mathbf{G}_{5,0} = \mathbf{G}_{5,3} \vee \mathbf{P}_{5,3}\, \mathbf{G}_{2,0}$)

$\boldsymbol{G}_{5,3} = g_5 \vee g_4 \vee \overline{k_4}\, g_3$

$\boldsymbol{G}_{2,0} = g_2 \vee g_1 \vee \overline{k_1}\, g_0$

$\boldsymbol{P}_{5,3} = \overline{k_4}\ \overline{k_3}\ \overline{k_2}$

Pour s_6 on précalcule p_6 et $p_6 \oplus \overline{k_5}$.

On vérifie que le calcul de $\boldsymbol{G}_{5,3}$ est plus rapide que le calcul de $\mathbf{G}_{5,3}$.

$\mathbf{G}_{5,3} = a_5\, b_5 \vee (a_5 \vee b_5)\, a_4\, b_4 \vee (a_5 \vee b_5)\, (a_4 \vee b_4)\, a_3\, b_3$

$\boldsymbol{G}_{5,3} = a_5\, b_5 \vee a_4\, b_4 \vee (a_4 \vee b_4)\, a_3\, b_3$

Additionneur de Ling (2)

La performance repose sur un arbre ternaire (maximum raisonnable d'entrées par porte), sur un calcul direct à partir des a_i et b_i (sans passer explicitement par g_i, p_i, $\overline{k_i}$), su un arbre plus simple et sur le précalcul de la somme modulo 2 pour le résultat S.
On vérifie facilement que:

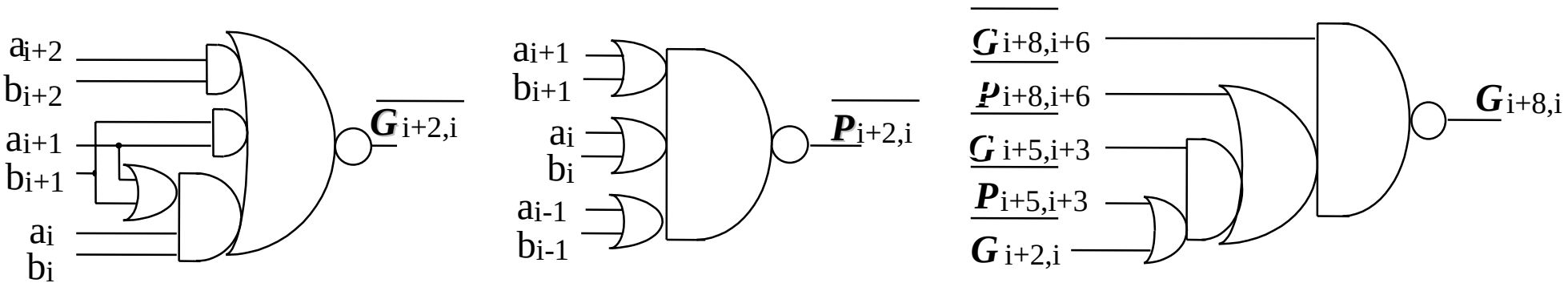
$\mathbf{G}_{i+2,i} = \overline{k}_{i+2}\, \boldsymbol{G}_{i+2,i}$

$\mathbf{P}_{i+2,i}\ \overline{k}_{i-1} = \overline{k}_{i+2}\, \boldsymbol{P}_{i+2,i}$

ce qui permet de vérifier que: $\boldsymbol{G}_{5,0} = \boldsymbol{G}_{5,3} \vee \boldsymbol{P}_{5,3}\, \boldsymbol{G}_{2,0}$. (voir transparent précédent)
A partir de cette propriété on calcule

$\boldsymbol{G}_{i+8,i} = \boldsymbol{G}_{i+8,i+6} \vee \boldsymbol{P}_{i+8,i+6}\boldsymbol{G}_{i+5,i+3} \vee \boldsymbol{P}_{i+8,i+6}\, \boldsymbol{P}_{i+5,i+3}\, \boldsymbol{G}_{i+2,i}$.

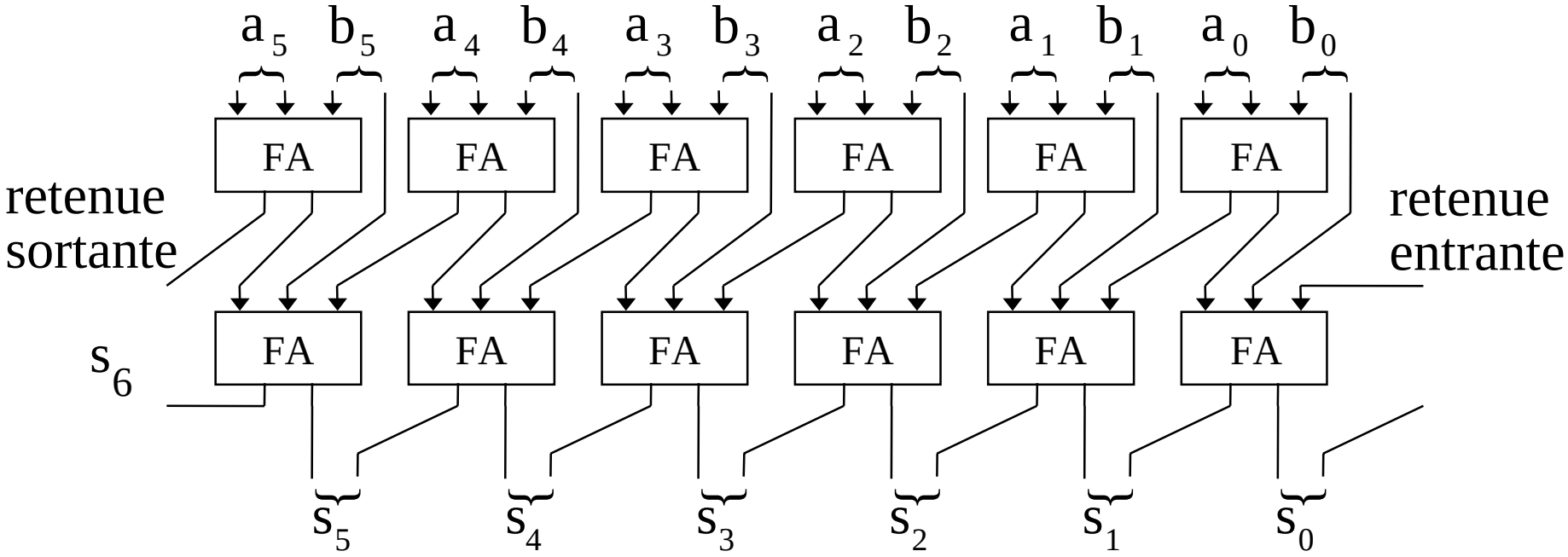
$\boldsymbol{P}_{+8,i} = \boldsymbol{P}_{i+8,i+6}\, \boldsymbol{P}_{i+5,i+3}\, \boldsymbol{P}_{i+2,i}$.



Aucune de ces portes complexes n'a plus de 3 transistors série entre sortie et alimentations

Addition parallèle sans propagation de la retenue CS

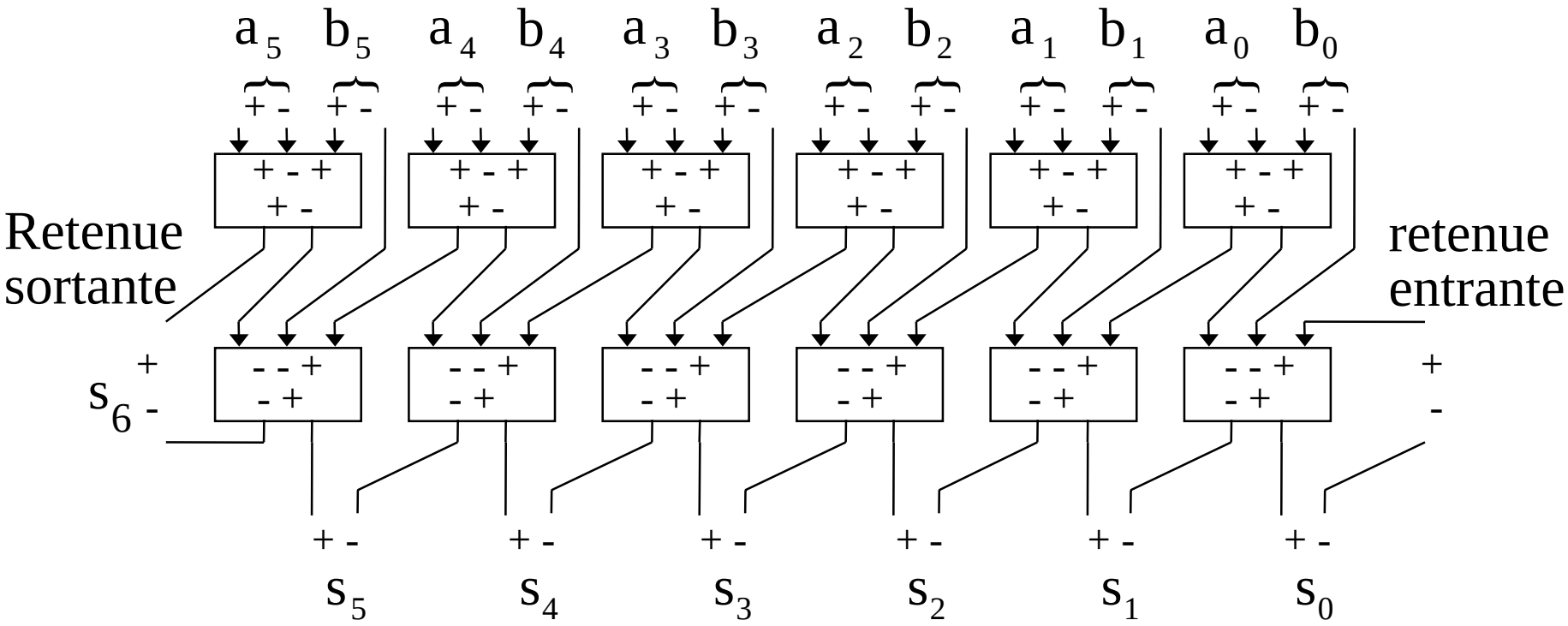
$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad B = \sum_{i=0}^{n-1} b_i\, 2^i \qquad S = \sum_{i=0}^{n} s_i\, 2^i \qquad a_i, b_i, s_i \in \{0,1,2\}$$



La somme pondérée des bits qui entrent est égale à la somme pondérée des bits qui sortent !

Addition parallèle sans propagation de la retenue BS

$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad B = \sum_{i=0}^{n-1} b_i\, 2^i \qquad S = \sum_{i=0}^{n} a_i\, 2^i \qquad a_i, b_i, s_i \in \{-1, 0, 1\}$$



Nombre, Chiffre et Bit

Un nombre est un ensemble ordonné de chiffres (en général).
La valeur d'un nombre est (en général) la somme pondérée de ses chiffres.
Les poids sont (en général) des puissances de la base de numération.
La base de numération est (souvent) une puissance de 2 (2,4,8,16)

$$A=\sum_{i=0}^{n-1} a_i\, 2^i \qquad A=\sum_{i=1}^{n} (1+2^{-i})a^i \qquad A=\sum_{i=0}^{n-1} a_i \log (1+2^{-i})$$

Si un chiffre a 2 valeurs, il est codé sur 1 bit. Ces deux valeurs sont généralement 0 et 1
il y a alors confusion entre chiffre et bit
Si un chiffre a plus de 2 valeurs,
sa valeur est (commodément) la somme pondérée des bits qui le codent.

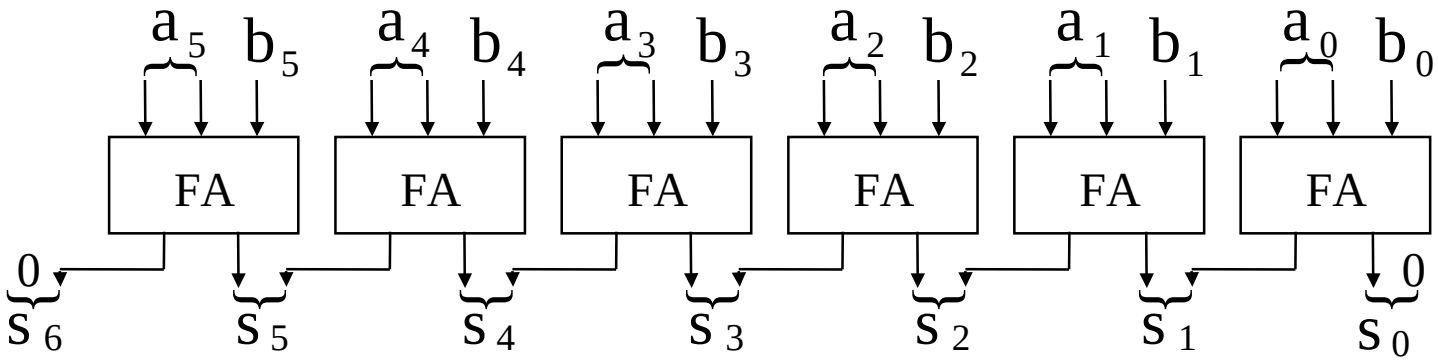
$a_i \in \{0,1\}$ $a_i \in \{-1,1\}$ $a_i \in \{0,1,2\}$ $a_i \in \{0,1,2,3\}$ $a_i \in \{-1,0,1\}$

Des contre-exemples sont donnés.

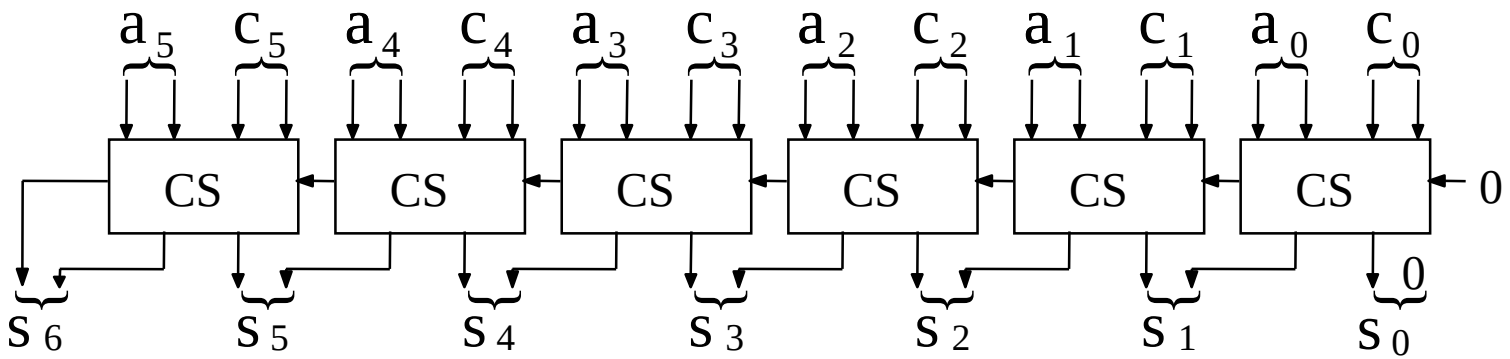
Additions sans propagation

$$A = \sum_{i=0}^{n-1} a_i\, 2^i \qquad C = \sum_{i=0}^{n-1} c_i\, 2^i \qquad S = \sum_{i=0}^{n} s_i\, 2^i \qquad a_i, c_i, s_i \in \{0,1,2\}$$

1- Hybride: Carry-save + Conventionnel → Carry-save

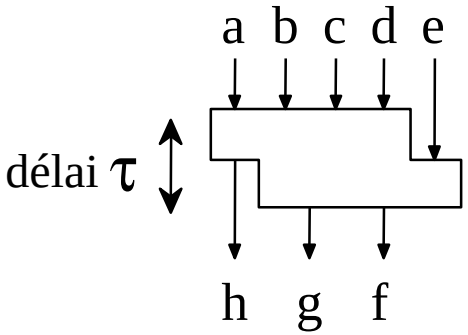


2- Binaire: Carry-save + Carry-save → Carry-save

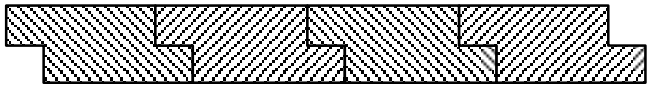


Variantes de cellules de CS

Modèle de délai



h sort avant
que e rentre



Cette cellule est également appelée “4 donne 2”

$a + b + c + d + e = f + 2*g + 2*h$

h ne dépend pas de e

a	b	c	d	Σ	f	g	h	
0	0	0	0	0	e	0	0	
0	0	0	1	1	$\bar{e}$	e	0	
0	0	1	0	1	$\bar{e}$	e	0	
0	0	1	1	2	e	0/1	1/0	← 2
0	1	0	0	1	$\bar{e}$	e	0	
0	1	0	1	2	e	0/1	1/0	← 4
0	1	1	0	2	e	0/1	1/0	← 8
0	1	1	1	3	$\bar{e}$	e	1	
1	0	0	0	1	$\bar{e}$	e	0	
1	0	0	1	2	e	0/1	1/0	← 16
1	0	1	0	2	e	0/1	1/0	← 32
1	0	1	1	3	$\bar{e}$	e	1	
1	1	0	0	2	e	0/1	1/0	← 64
1	1	0	1	3	$\bar{e}$	e	1	
1	1	1	0	3	$\bar{e}$	e	1	
1	1	1	1	4	e	1	1	

Optimisation de cellules de CS