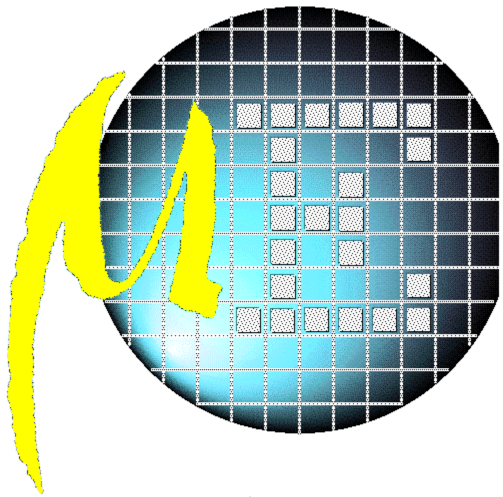
Addition sur Silicium



Alain GUYOT

TIMA

☎(33) 04 76 57 46 16

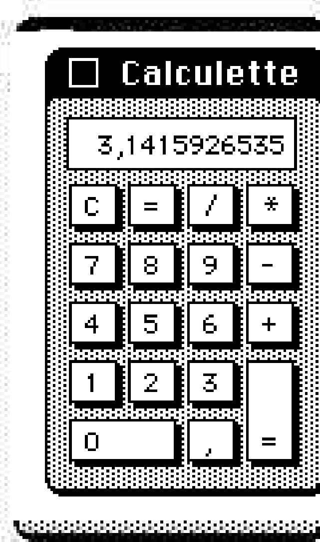
Alain.Guyot@imag.fr

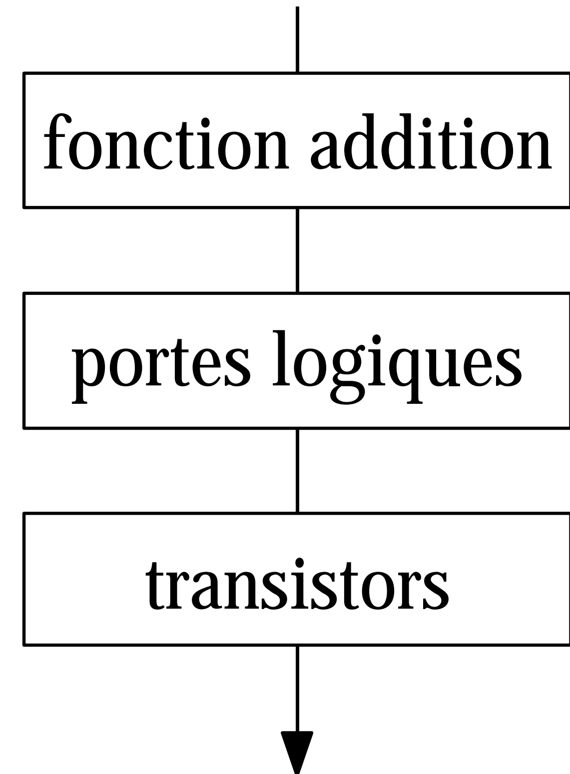
http://tima-cmp.imag.fr/~guyot

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 la surface et/ou la vitesse

Moyens
Associativité,
commutativité,
distributivité

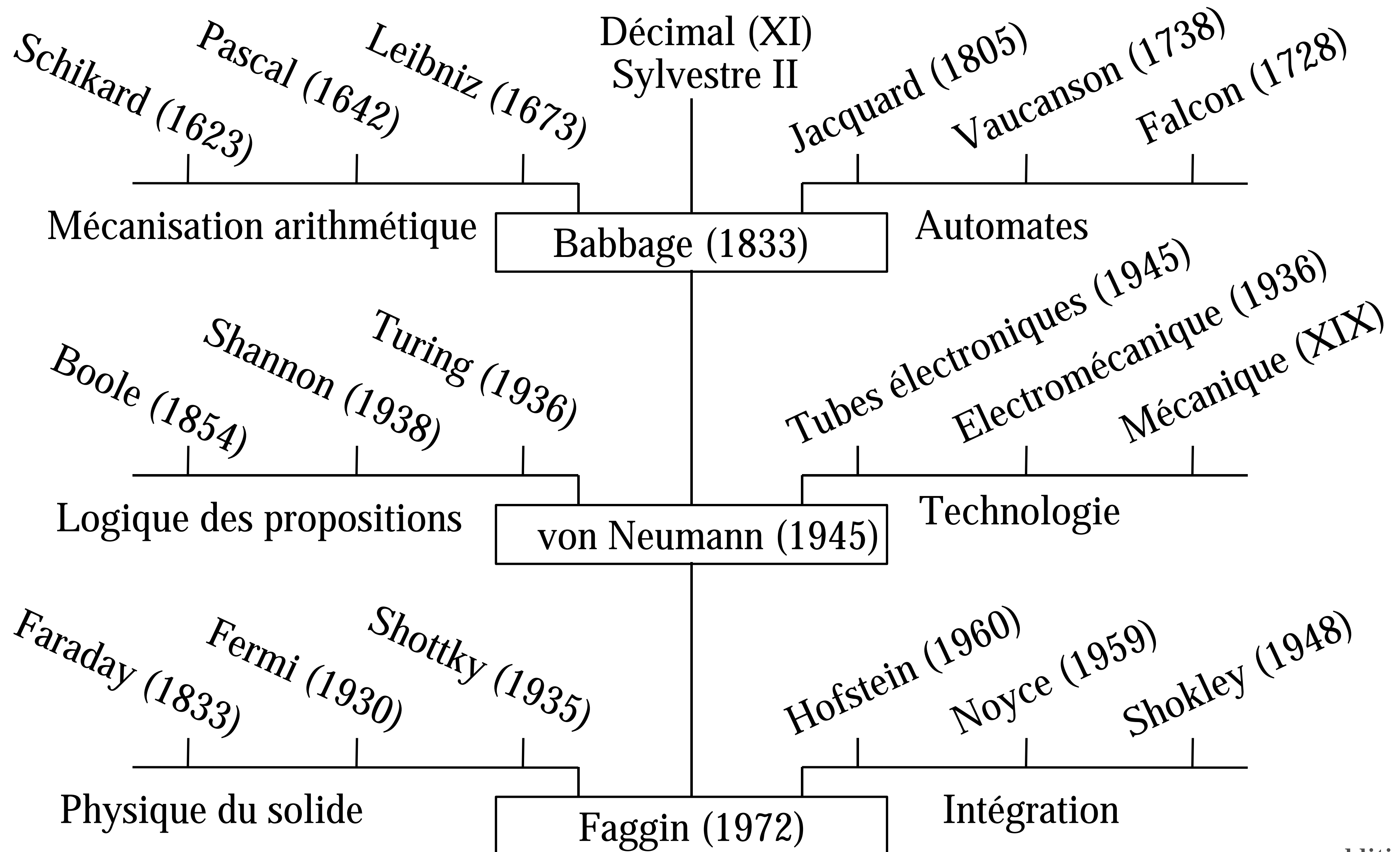




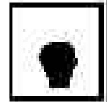
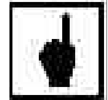
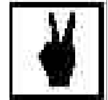
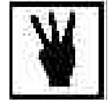
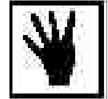
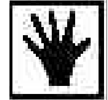
Problèmes de l'addition
- Propagation de la retenue

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

La mécanisation du calcul



Rappels sur l'écriture des entiers en base 2

Digital

Entiers positifs

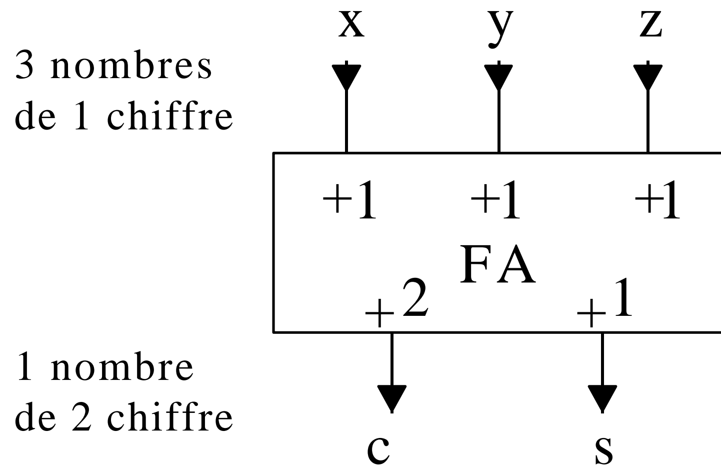
$$A = \sum_{i=0}^{n-1} a_i \, 2^i \qquad a_i \in \{ 0 , 1 \} \qquad A \in [0, 2^n - 1]$$

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

Entiers relatifs

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

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

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

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

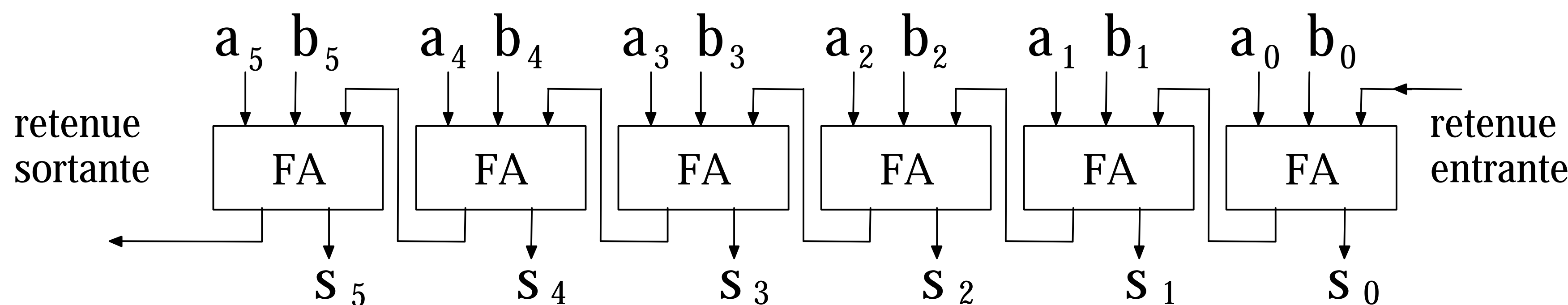
$s = x \oplus y \oplus z$ *somme modulo 2*

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

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

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



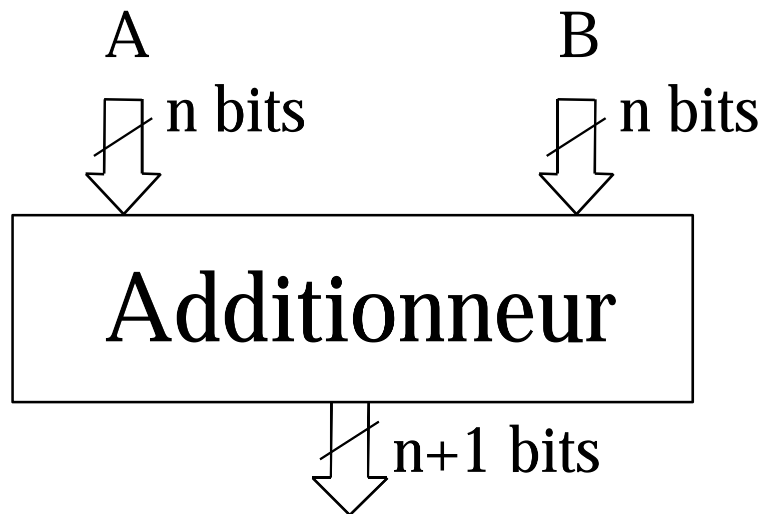
Tout assemblage cohérent de “FA” conserve la propriété: La somme pondérée de ce qui sort est égale à la somme pondérée de ce qui entre

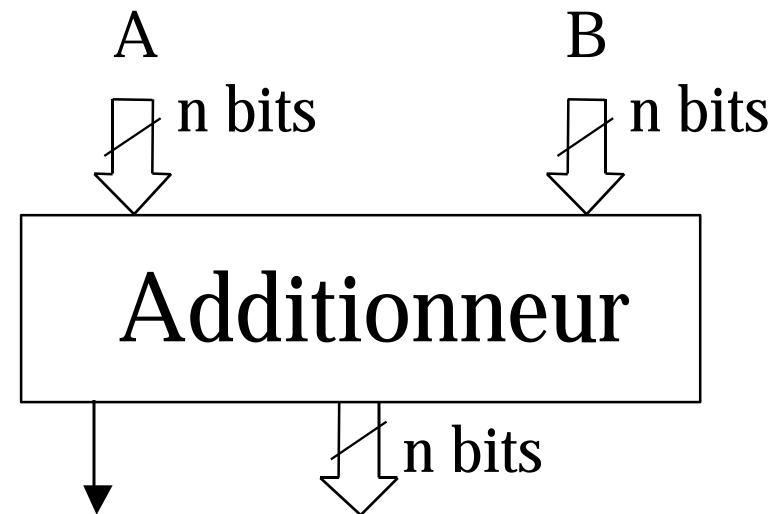
Si on ignore la retenue sortante:

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

Dans ce cas l’opérateur accepte 2 conventions.

Débordement d'entiers positifs





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

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

Solution 1

Solution 2

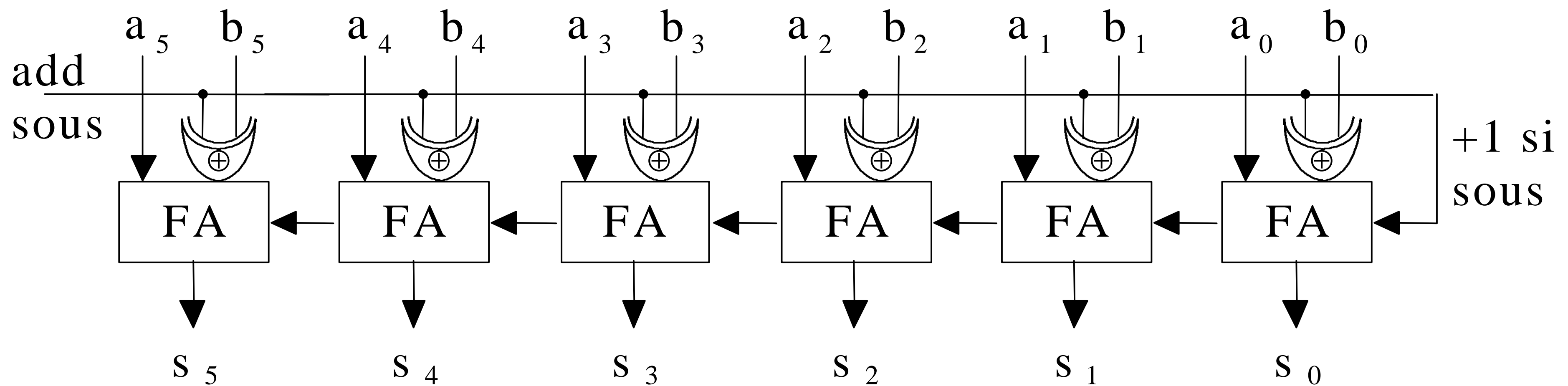
Entiers relatifs

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

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

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

Additionneur/soustracteur



$$S = A \pm B$$

Générateur VHDL : ADD_SUB paramétrable

```
library IEEE; use IEEE.std_logic_1164.all, IEEE.std_logic_components.all;

entity ADD_SUB is
      generic (   N        : positive := 6);
      port    (   A, B     : in    std_logic_vector(N-1 downto 0);
                  AS_IN    : in    std_logic;
                  AS_OUT   : out   std_logic;
                  S        : out   std_logic_vector(N-1 downto 0));
end ADD_SUB;

architecture STRUCTURAL of ADD_SUB is
   component FULL_ADDER
      port (X, Y, Z : in std_logic; S, C : out std_logic );
   end component;
   signal BAS  : std_logic_vector (N-1 downto 0) ;
   signal AS   : std_logic_vector (N-2 downto 0) ;
begin
L1: block begin
  I1: XORGATE    port map (Input(1)=>AS_IN, Input(2)=>B(0), Output=>BAS(0));
  I2: FULL_ADDER port map (A(0), BAS(0), AS_IN, S(0), AS(0));
  L2: for I in 1 to N-2 generate
      I3: XORGATE    port map (Input(1)=>AS_IN, Input(2)=>B(I), Output=>BAS(I));
      I4: FULL_ADDER port map (A(I), BAS(I), AS(I-1), S(I), AS(I));
     end generate;
  I5: XORGATE    port map (Input(1)=>AS_IN, Input(2)=>B(N-1), Output=>BAS(N-1));
  I6: FULL_ADDER port map (A(N-1), BAS(N-1), AS(N-2), S(N-1), AS_OUT);
end block L1;
end STRUCTURAL;

configuration CFG_ADD_SUB_STRUCTURAL of ADD_SUB is
for STRUCTURAL
 for L1 for all: XORGATE use CONFIGURATION IEEE.CFG_XORGATE_BI; end for;
        for all: FULL_ADDER use CONFIGURATION WORK.CFG_FULL_ADDER_STRUCTURAL; end for;
 for L2 for I3: XORGATE    use CONFIGURATION IEEE.CFG_XORGATE_BI; end for;
        for I4: FULL_ADDER use CONFIGURATION WORK.CFG_FULL_ADDER_STRUCTURAL;  end for;
 end for; -- L2 -- end for; -- L1 --
end for; -- STRUCTURAL
end CFG_ADD_SUB_STRUCTURAL;
```

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

(erreur détectable)

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

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

Notation en complément à 2^n

Dite en complément à 2, en fait à 2^n

Le bit poids fort est négatif, les autres positifs

Tous les bits se traitent de la même façon dans l'addition

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

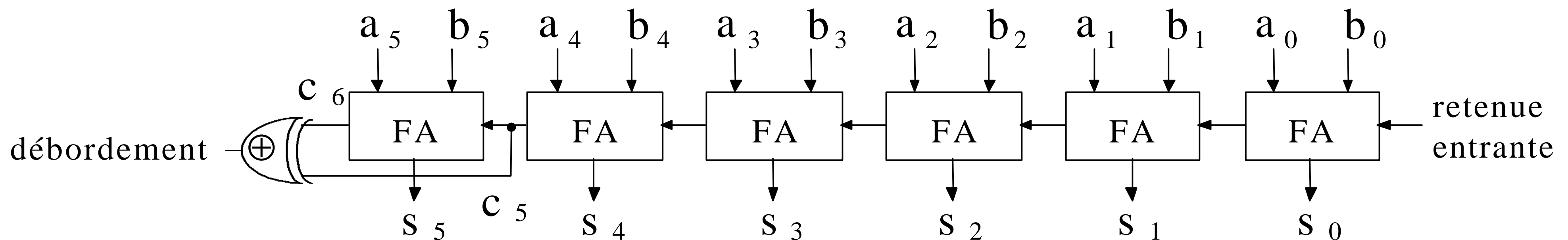
Il y a d'autres systèmes (peu usités)

Débordement de l'addition d'entiers relatifs

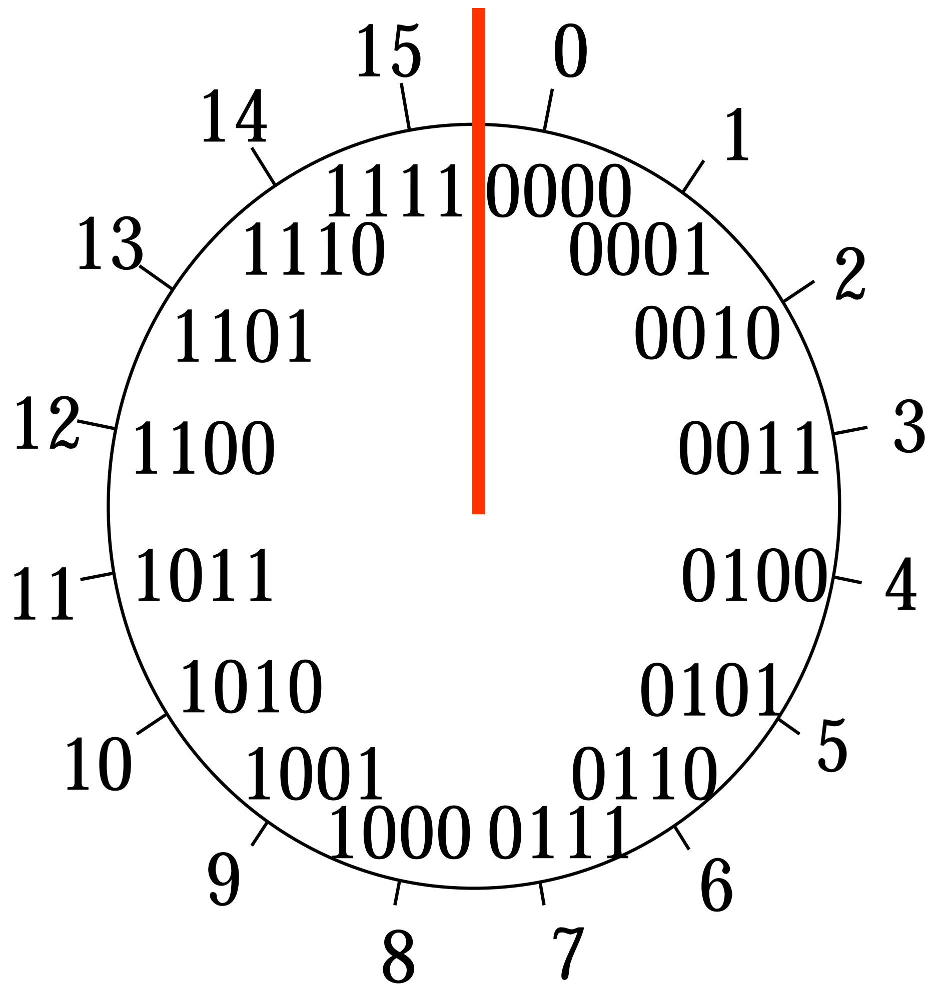
A	B	S=A+B
+	+	+
+	+	-
+	-	+
+	-	-
-	+	+
-	+	-
-	-	+
-	-	-

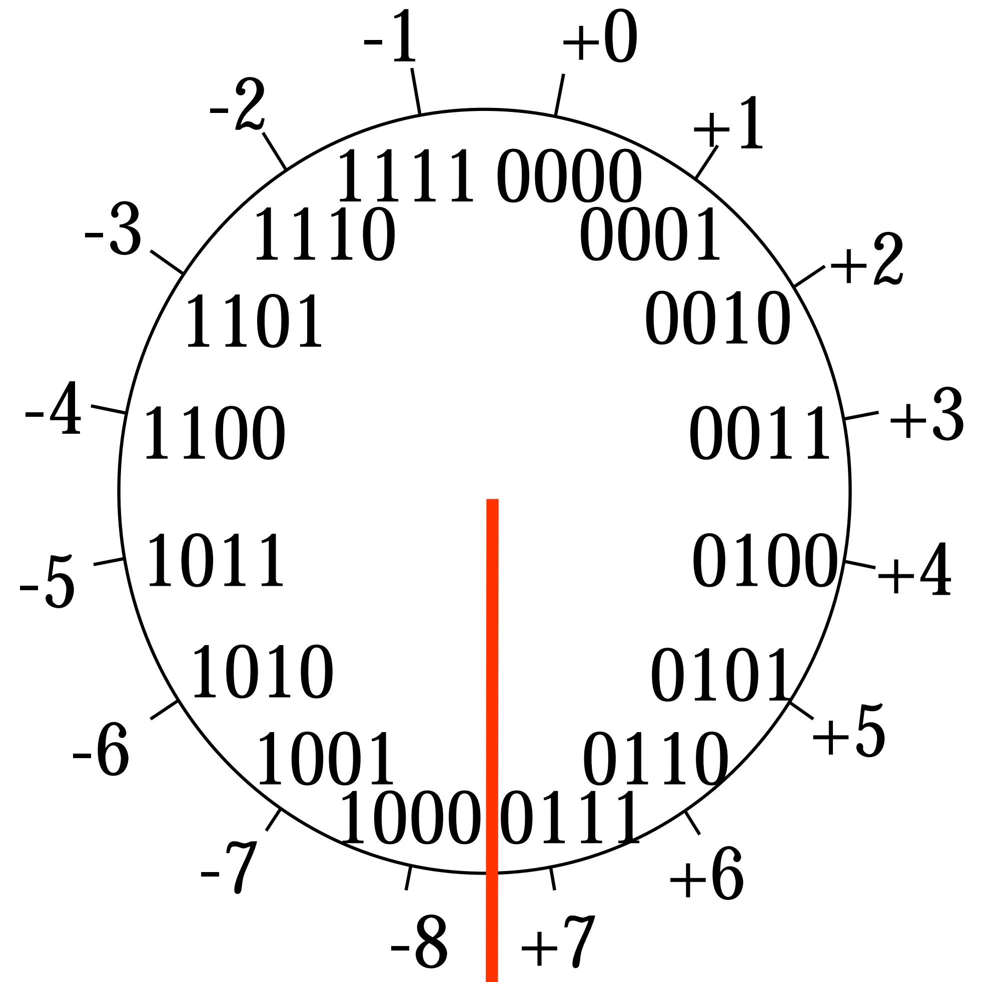
cas de débordement

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	-



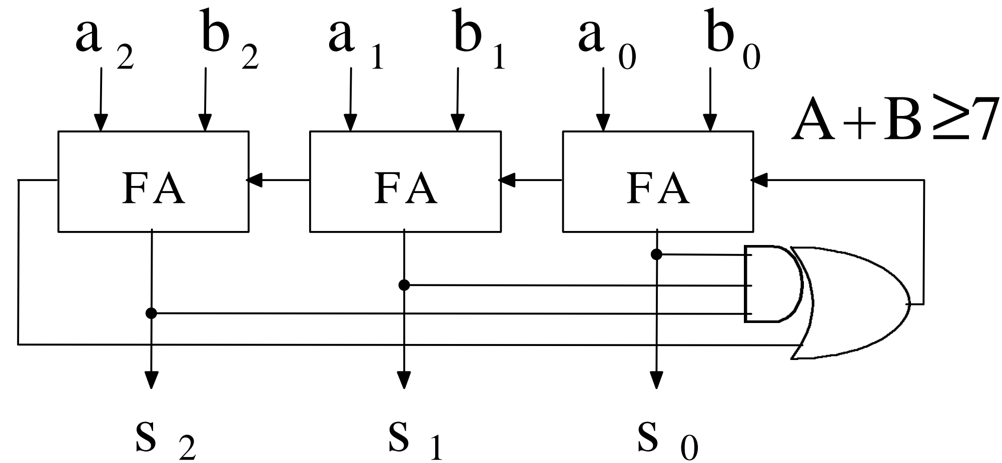
Débordement (exemple sur 4 bits)





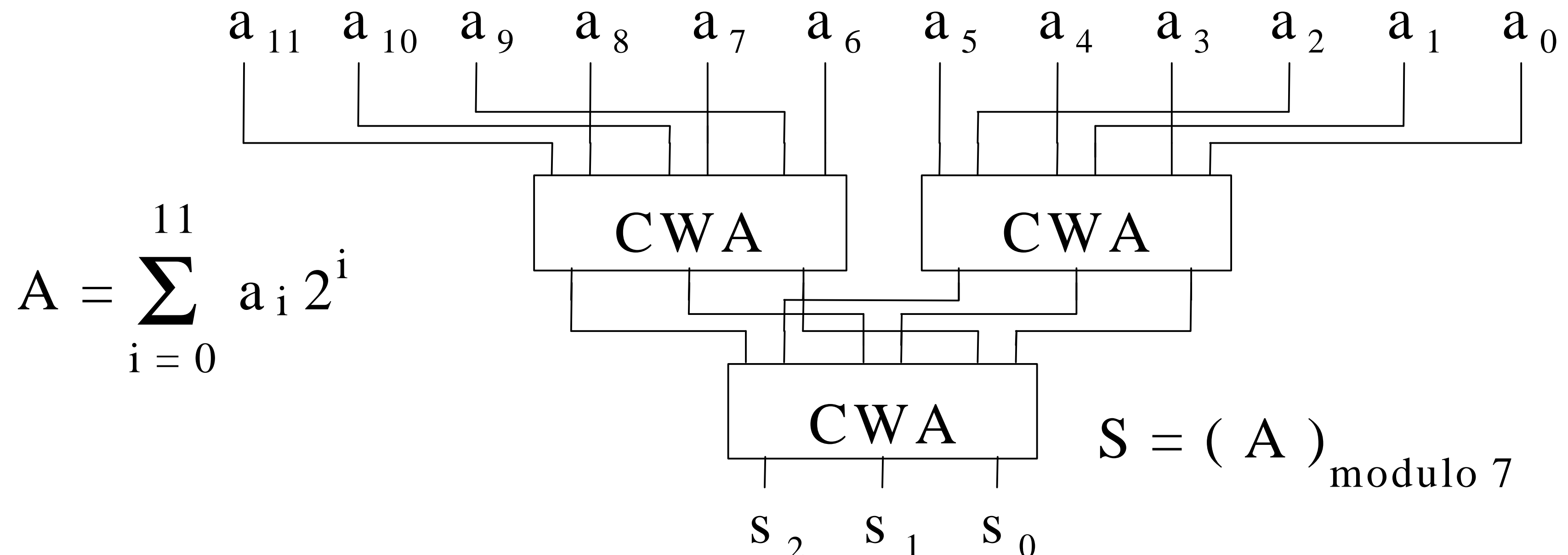
Modulo 2^n -1

Carry wrap around adder



$$S = (A + B)_{\text{modulo } 7}$$

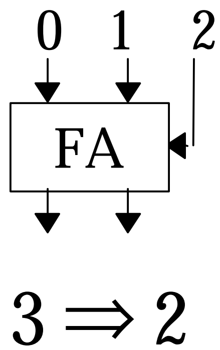
$$A = \sum_{i=0}^{11} a_i 2^i$$

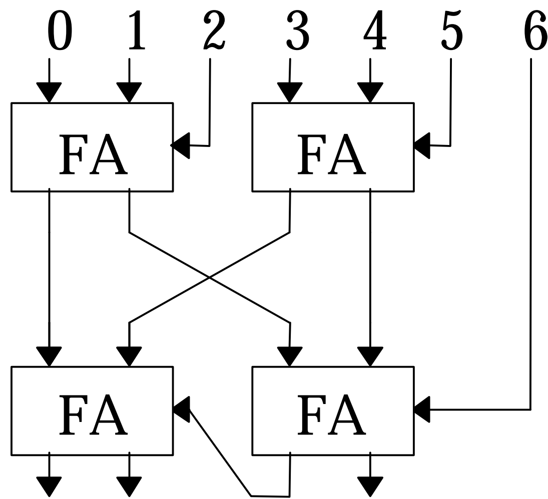


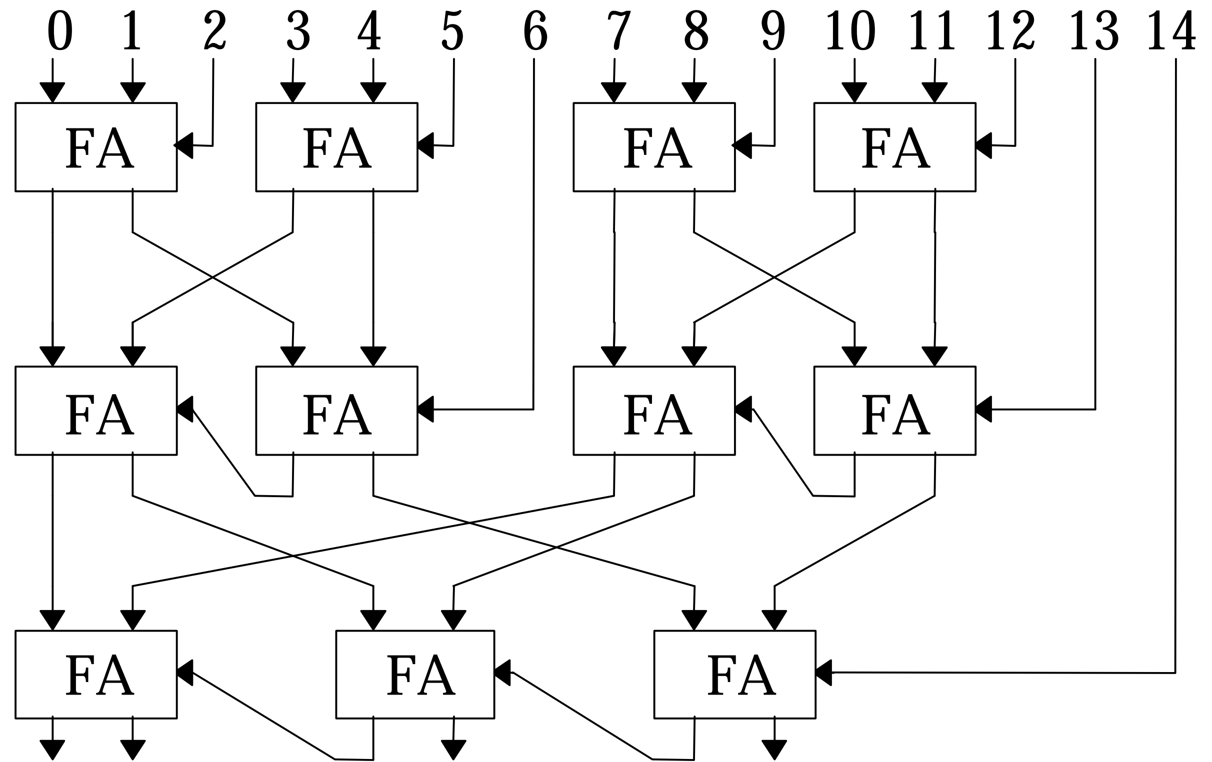
$$S = (A)_{\text{modulo } 7}$$

Arbre de Wallace

Compter les bits à 1 dans une chaîne



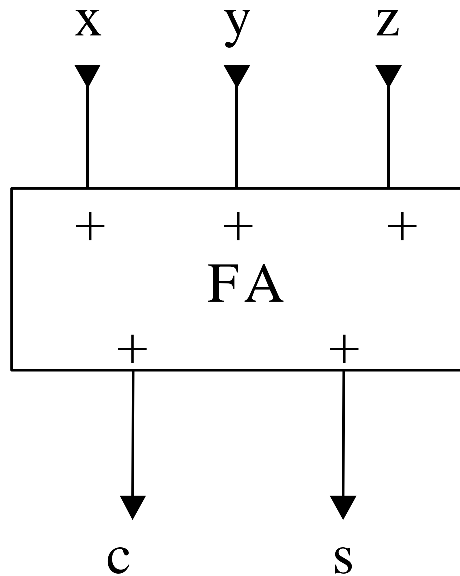




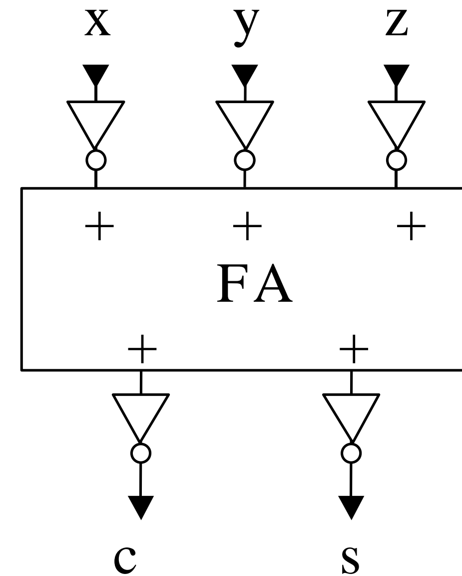
Le "Full Adder" est Autodual

(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



$\cong$

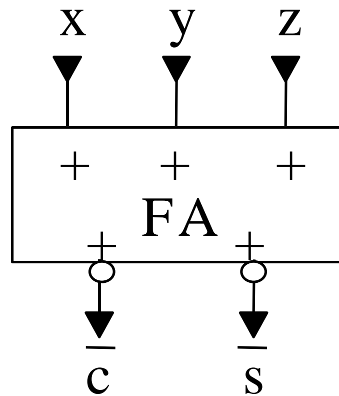


$$\bar{x} = 1 - x$$

$$x + y + z = 2c + s$$

$$\bar{x} + \bar{y} + \bar{z} = 3 - x - y - z = 2(1\text{-}c) + (1\text{-}s) = 2\bar{c} + \bar{s}$$

"Full Adder" (FA) symétrique

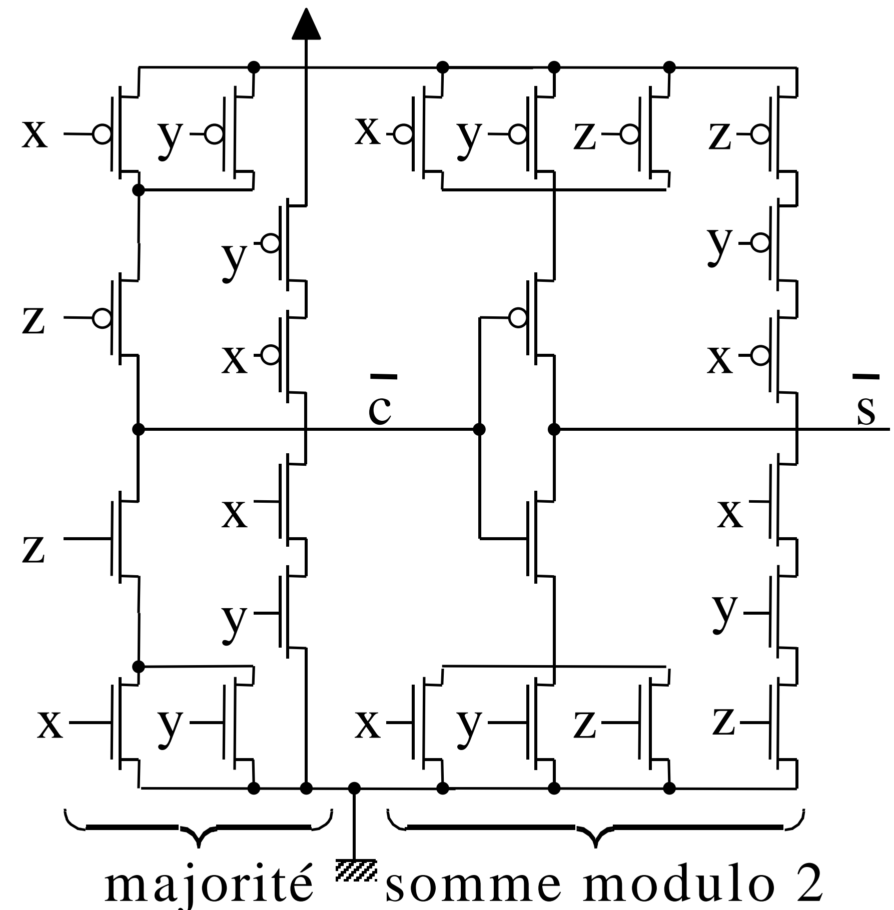


c = majorité (x,y,z)

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

s = (x∧y∧z) ∨ $\overline{c}$∧(x∨y∨z)

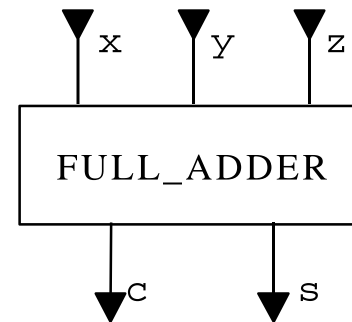
	∧	M	∨	c	s
000	0	0	0	0	0
001	0	0	1	0	1
011	0	1	1	1	0
111	1	1	1	1	1

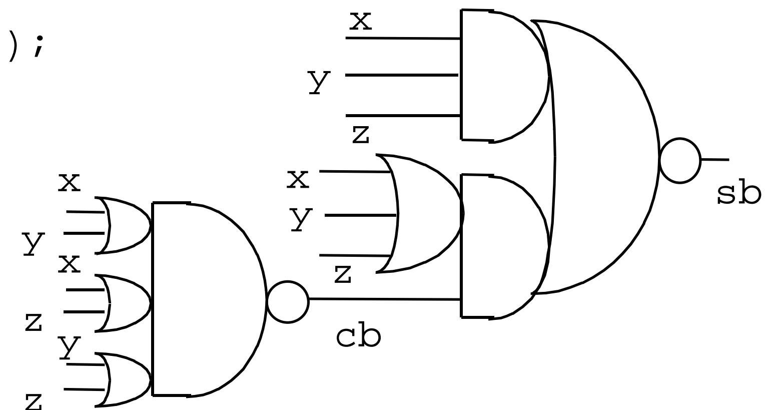


Circuit « miroir »

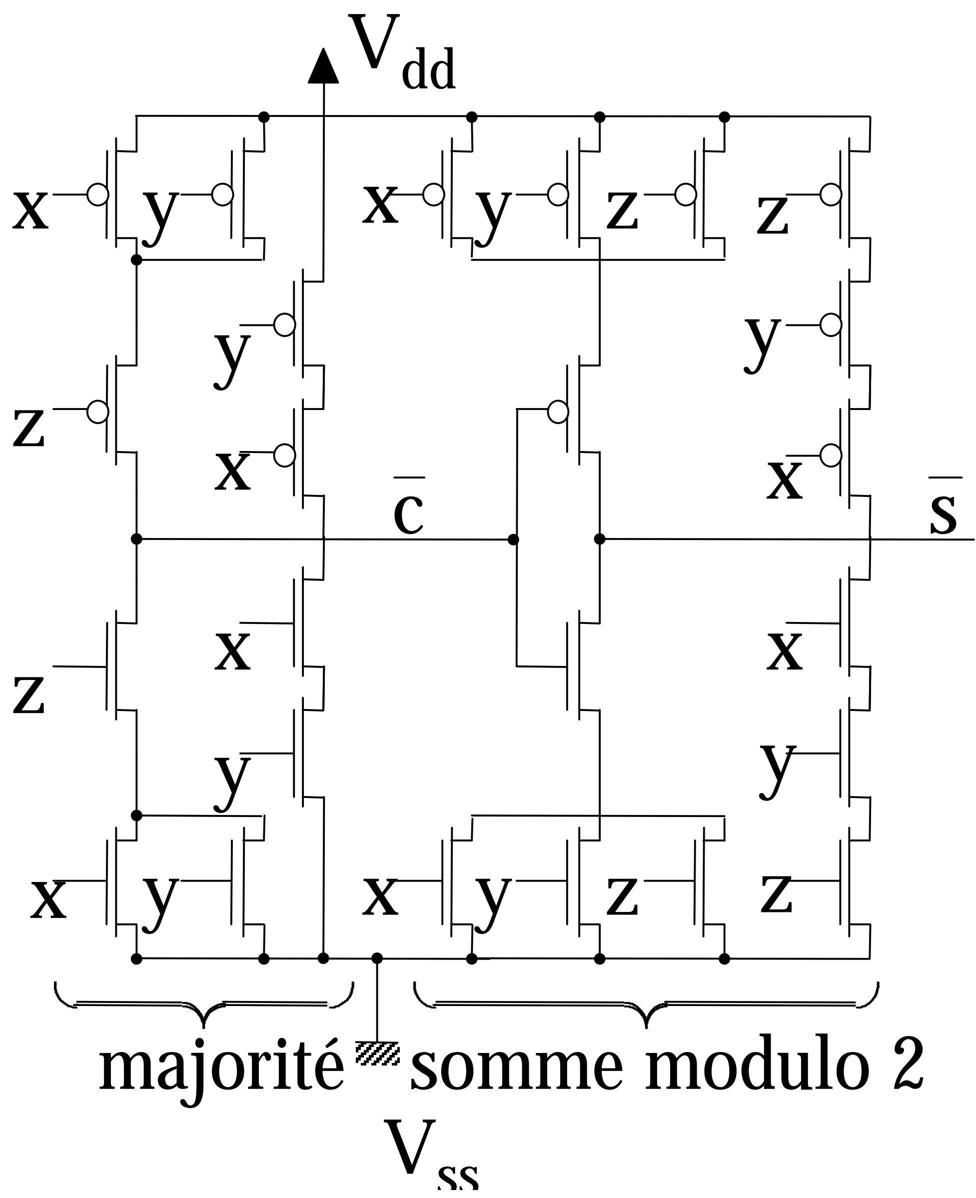
Description VHDL dataflow du "Full Adder"

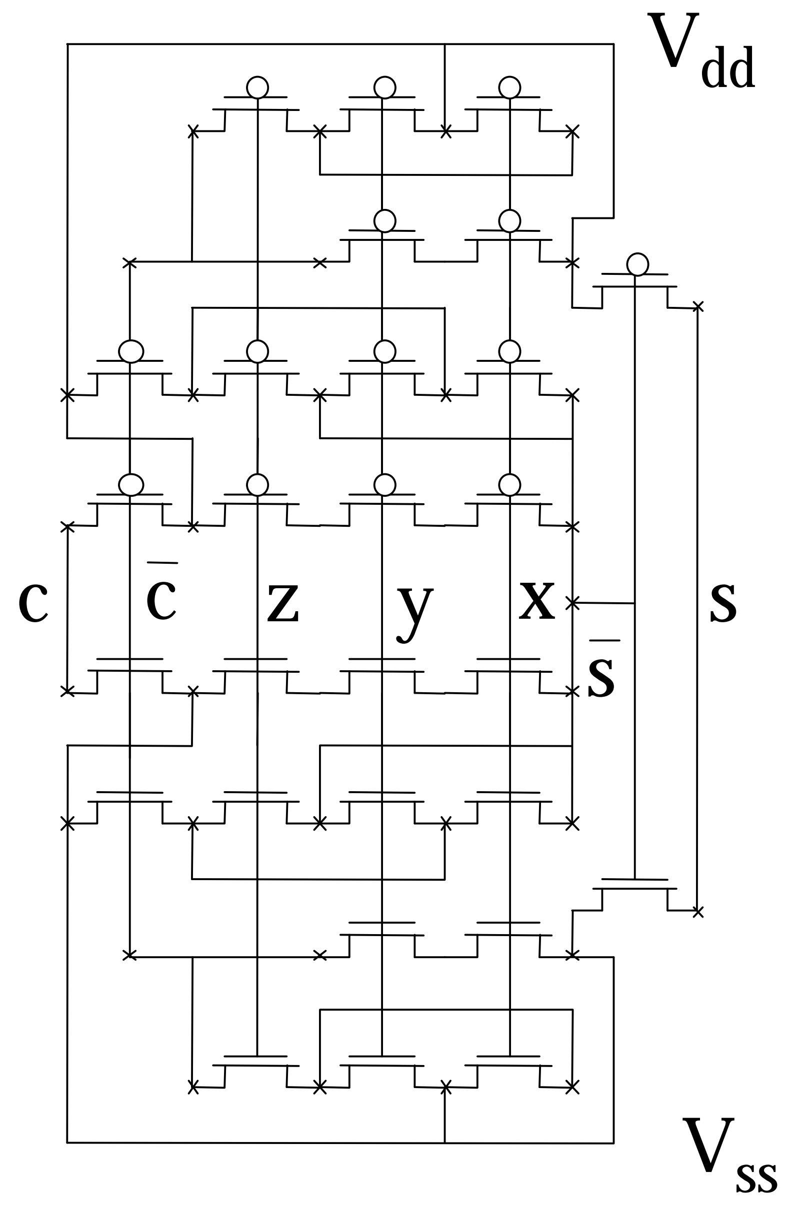
```
library IEEE; use IEEE.std_logic_1164.all,
        IEEE.std_logic_components.all;

entity FULL_ADDER is
      Port (    X, Y, Z : in     std_logic;
                C, S    : out    std_logic );
end FULL_ADDER;

architecture DATAFLOW of FULL_ADDER is
signal CB, SB : std_logic;
begin
 CB <= not ((X and Y) or (X and Z) or (Y and Z));
 SB <= not ((X and Y and Z) or ( (X or Y or Z) and CB));
 C <= not CB ;
 S <= not SB ;
end DATAFLOW;

configuration CFG_FULL_ADDER_DATAFLOW of FULL_ADDER is
   for DATAFLOW
   end for;
end CFG_FULL_ADDER_DATAFLOW;
```



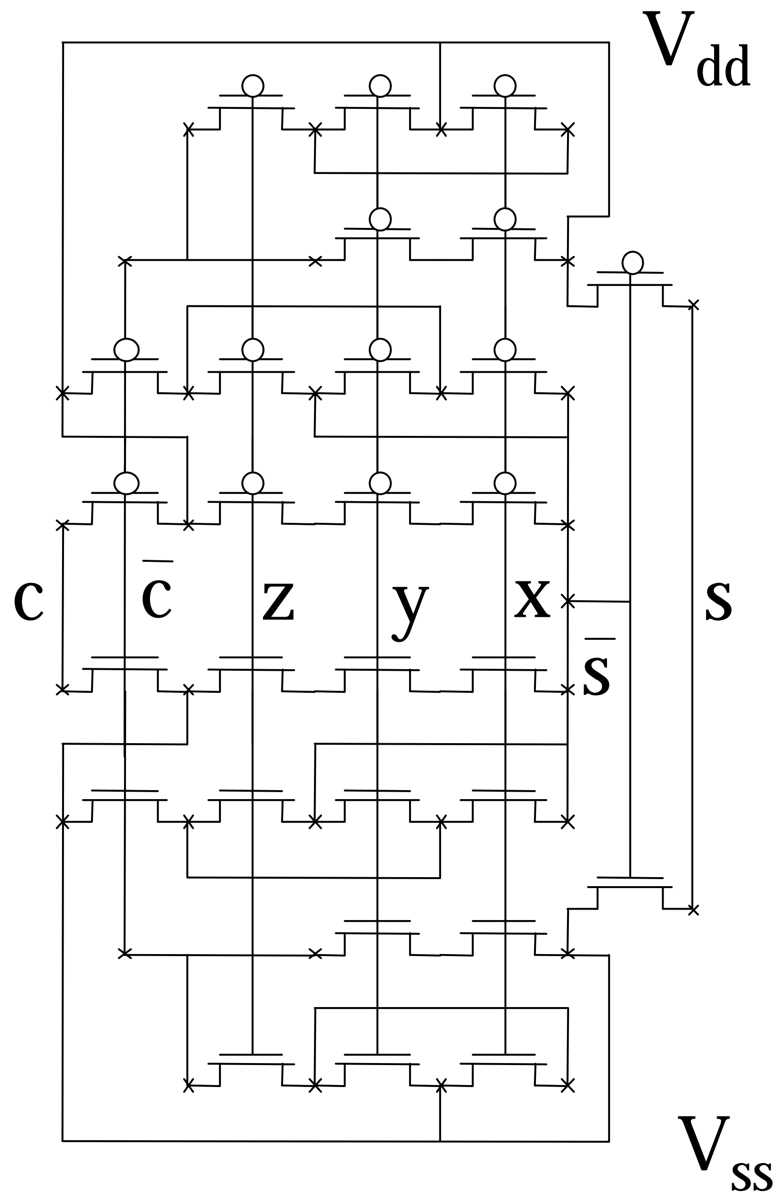


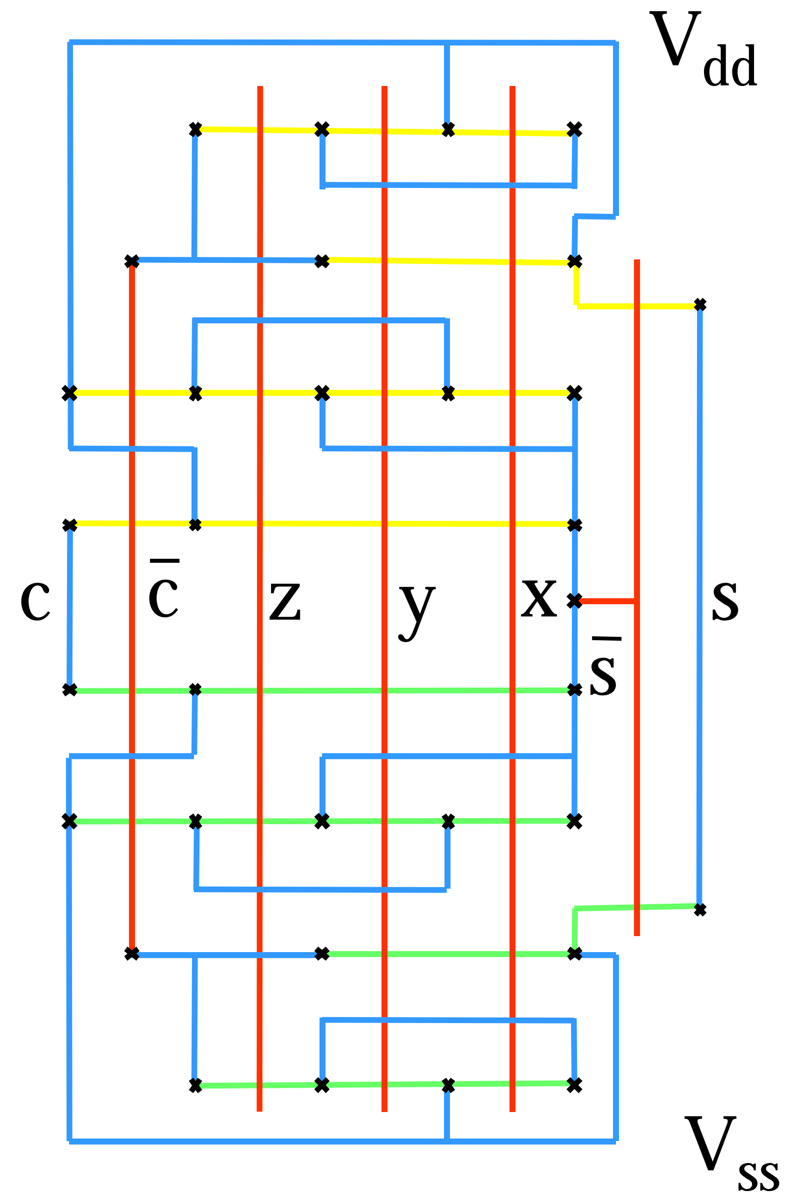
dessin du "Full Adder"



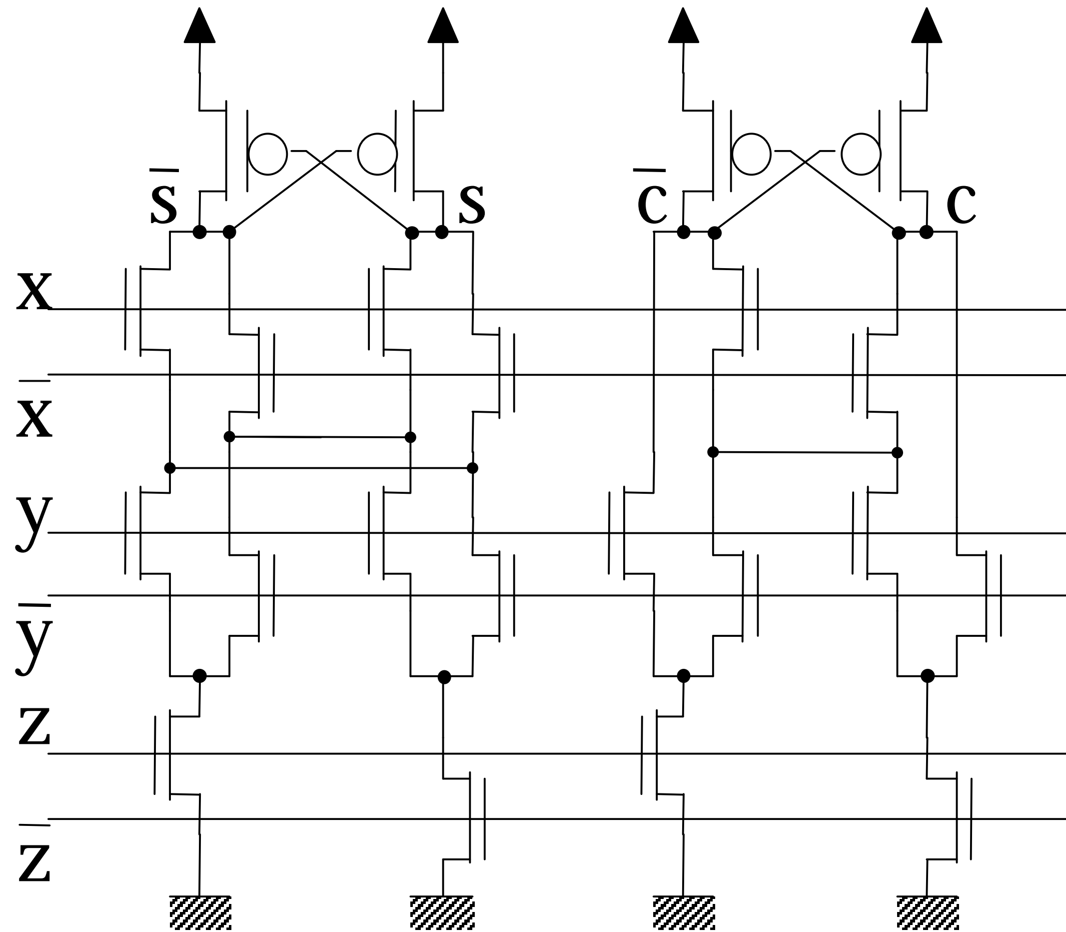


dessin bâton du full adder





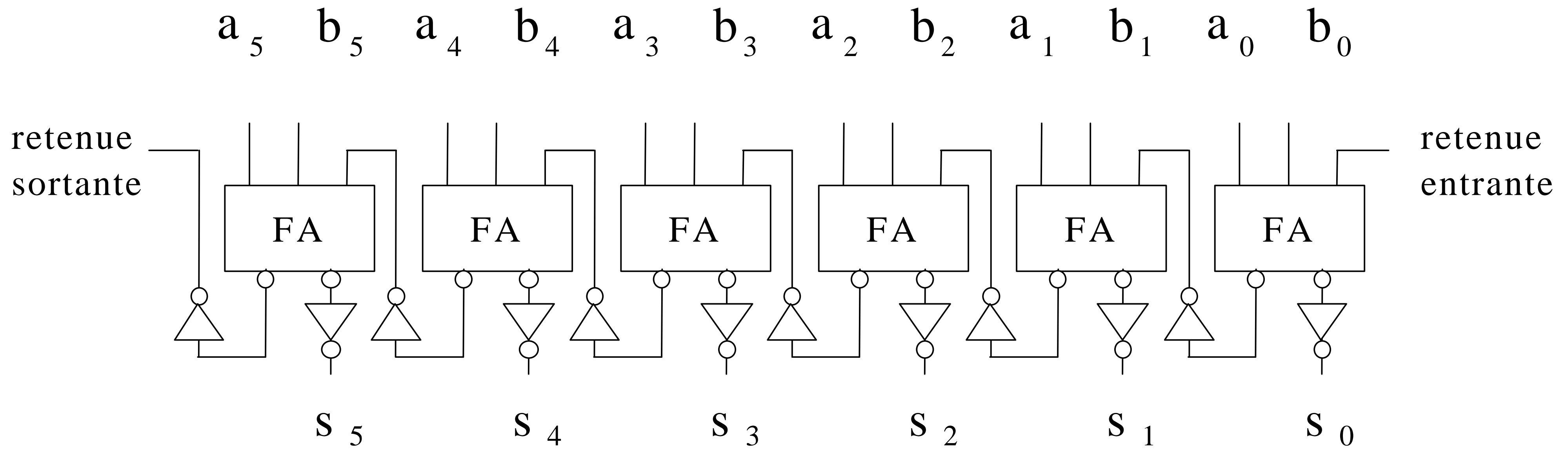
"Full Adder" (FA) en cascode différentiel

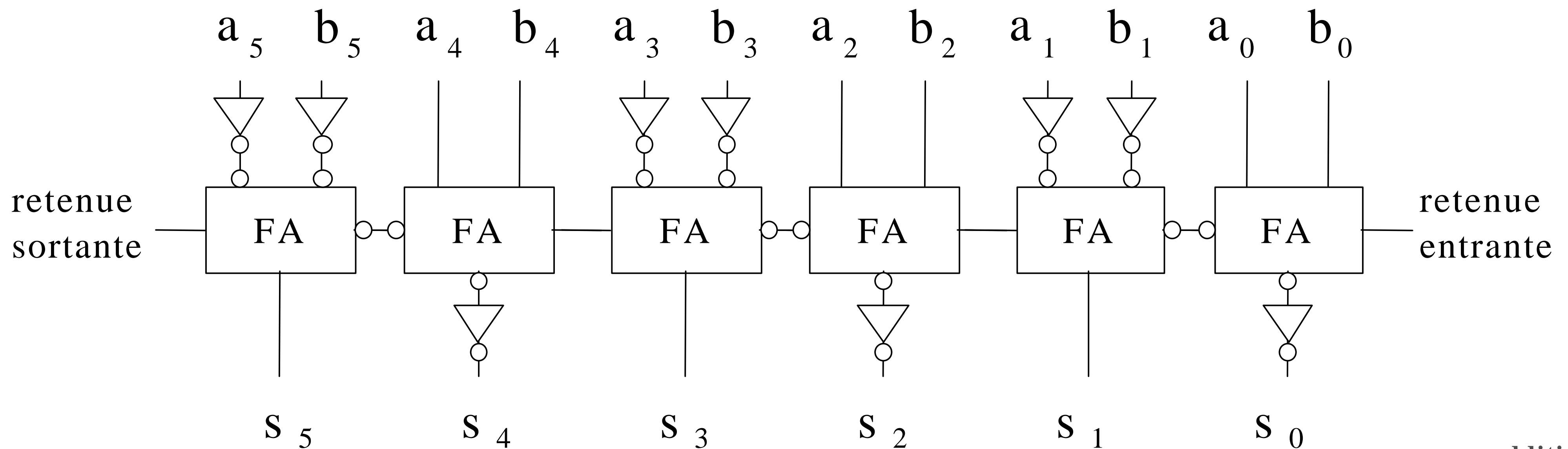


$$s = x \oplus y \oplus z = x \wedge y \wedge z \vee x \wedge \bar{y} \wedge \bar{z} \vee \bar{x} \wedge \bar{y} \wedge z \vee \bar{x} \wedge y \wedge \bar{z}$$

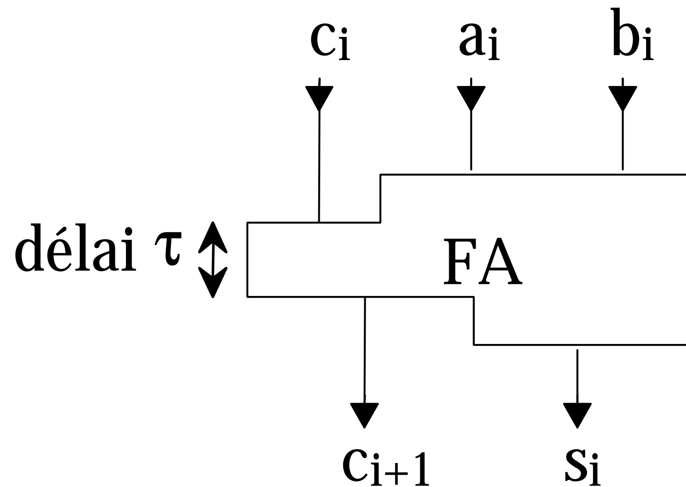
$$c = \text{majorité}(x,y,z) = y \wedge z \vee x \wedge \bar{y} \wedge z \vee x \wedge y \wedge \bar{z}$$

"Full Adder" (FA) symétrique





"Full Adder" (FA) dissymétrique



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

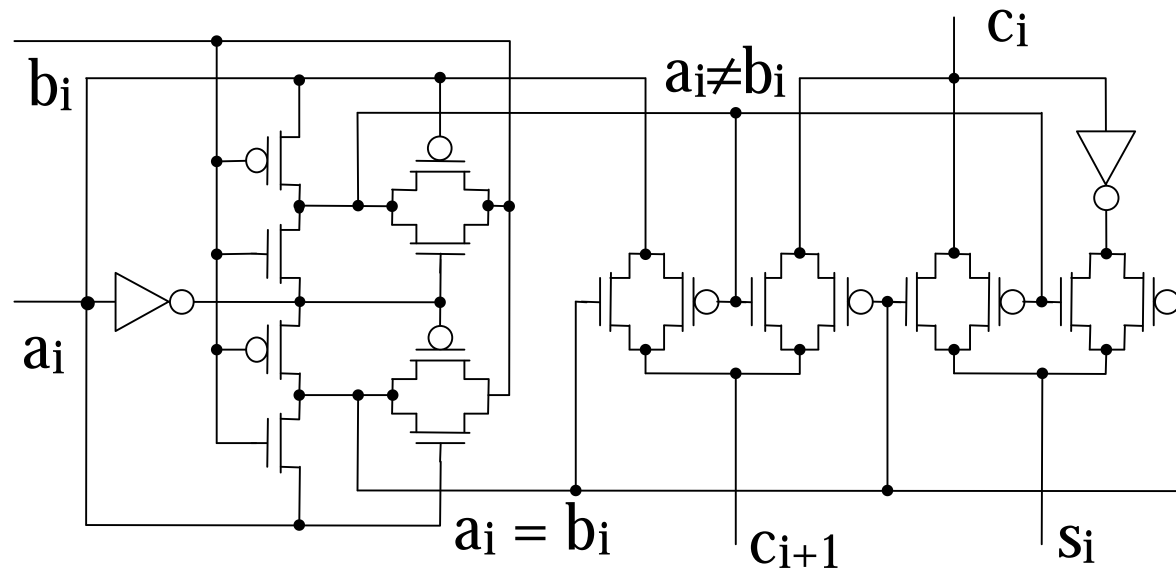
a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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	$\overline{c_i}$

Méthode pour synthèse à porte de transmission: faire passer les variables à transmettre dans la table de vérité

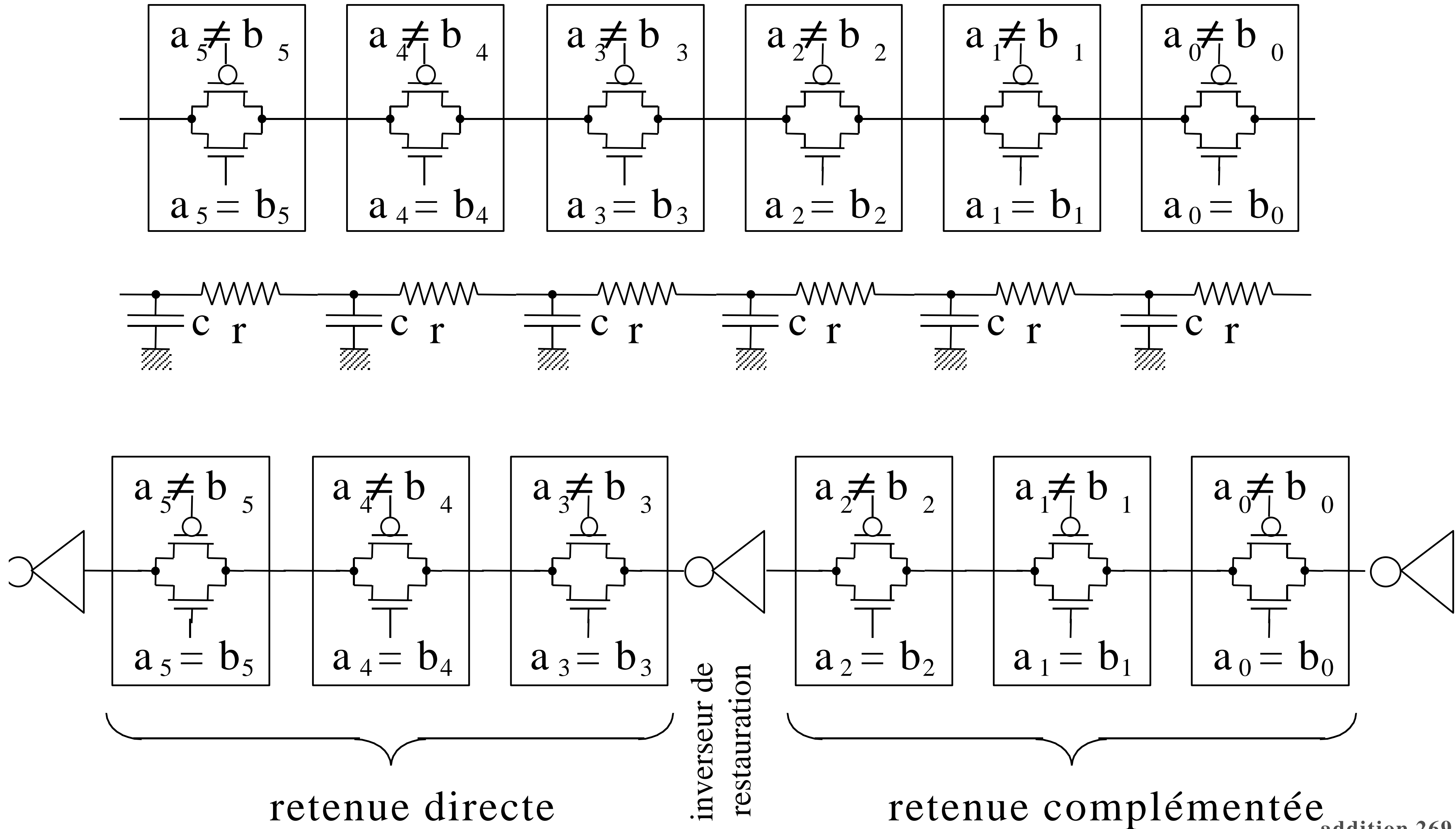
"Full Adder" (FA) à porte de transmission



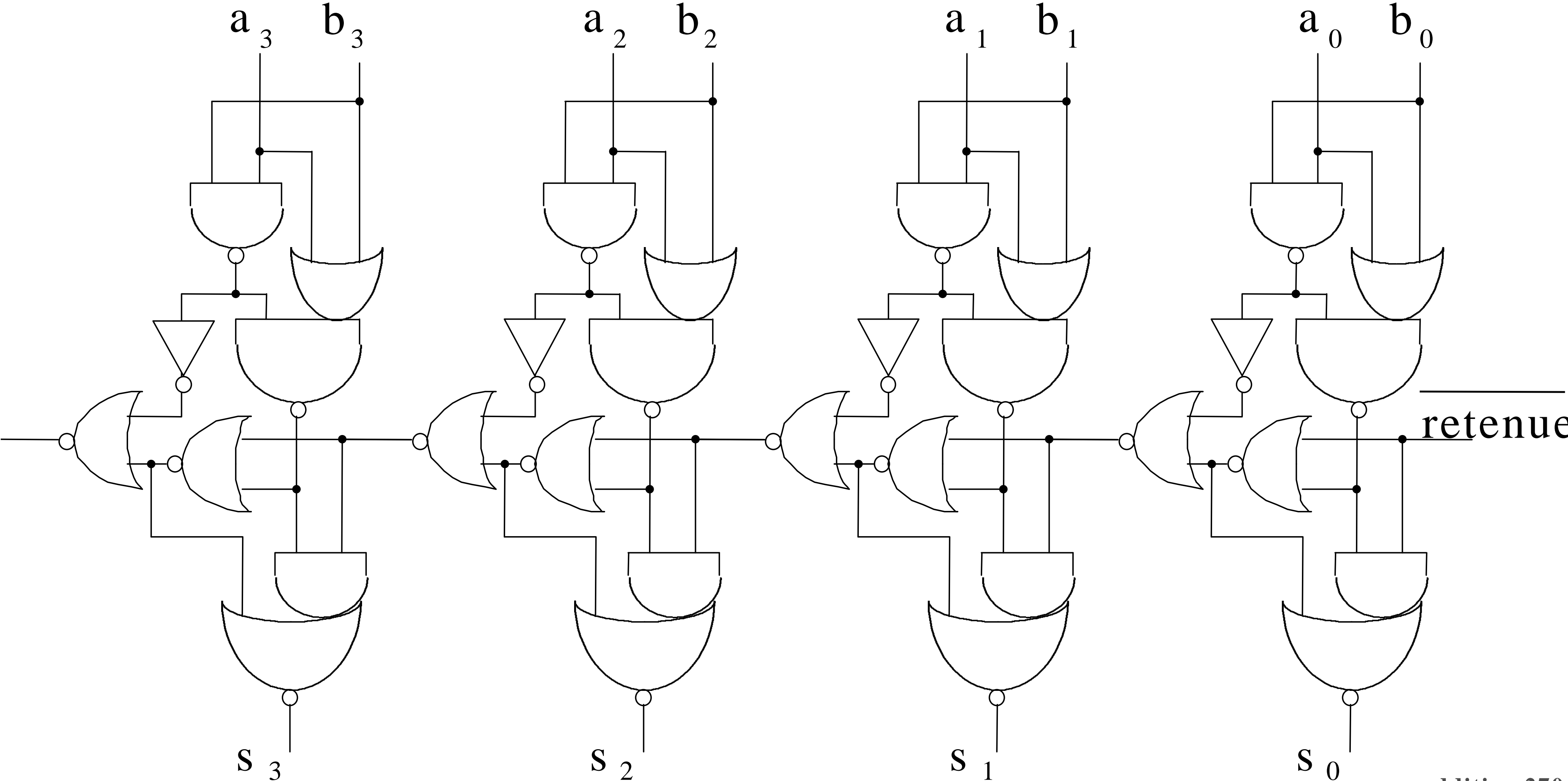
si $a_i = b_i$ **alors** $c_{i+1} := a_i$, $s_i := c_i$

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

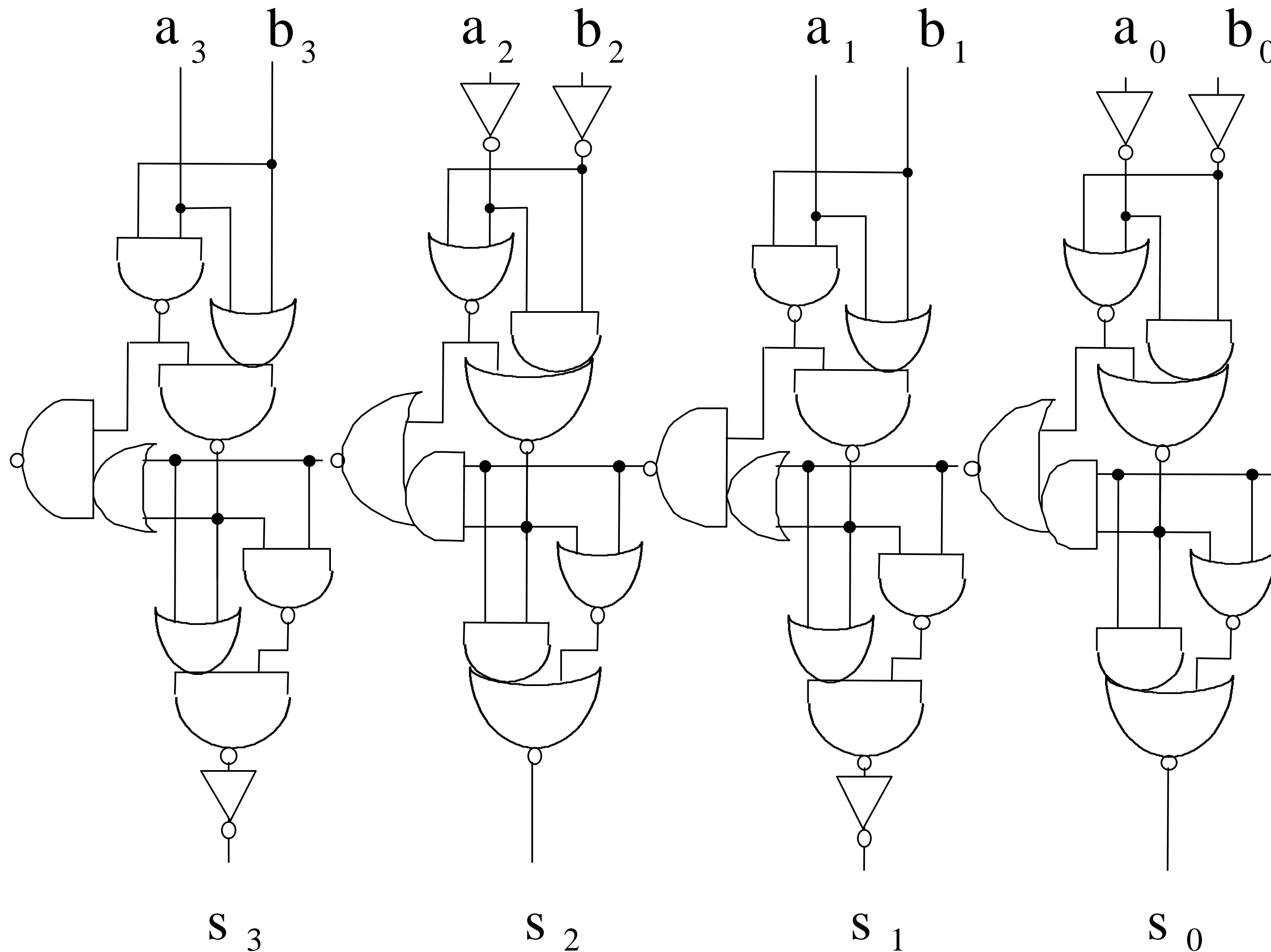
Amélioration du délai de l'addition à porte de transmission



Additionneur à 2 portes sur le chemin de la retenu



Additionneur à 1 porte sur le chemin de la retenue



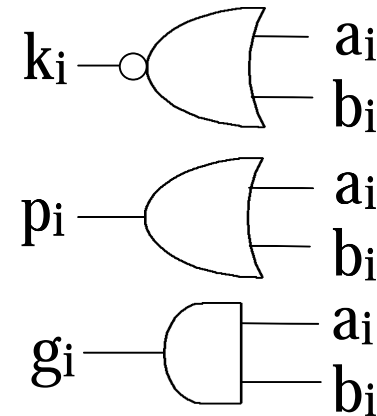
Génération et propagation de la retenue

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

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

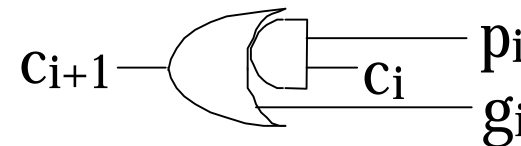
p_i (Propagation) $p_i = a_i \oplus b_i$

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

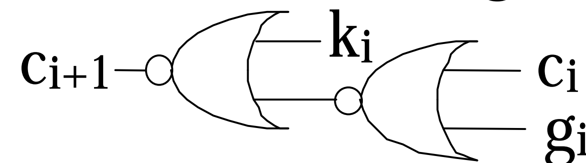


A	1 0 1 0 0 0 1 0 1 0 1 0 1 1 0 0
+B	0 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0
	p p g k k p p p g k g k p p p k
C	1 1 1 0 0 1 1 1 1 0 1 0 0 0 0 0

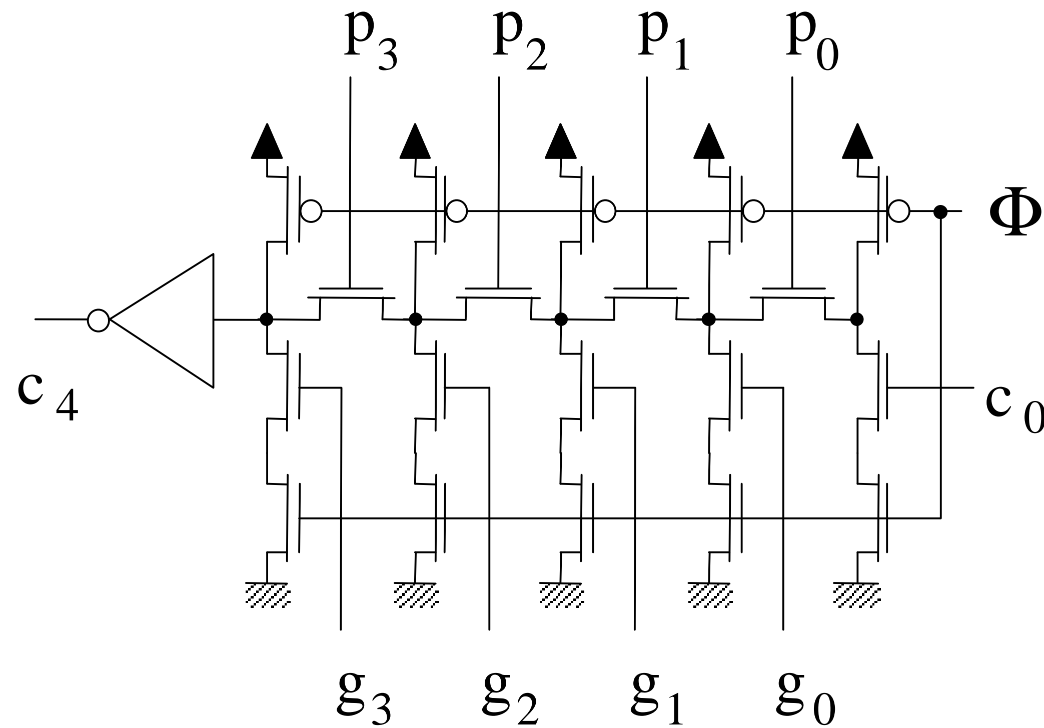
$$c_{i+1} = (p_i \wedge c_i) \vee g_i = (\overline{k_i} \wedge c_i) \vee g_i$$



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



Propagation "Manchester"



Anticipation du calcul de la retenue (1)

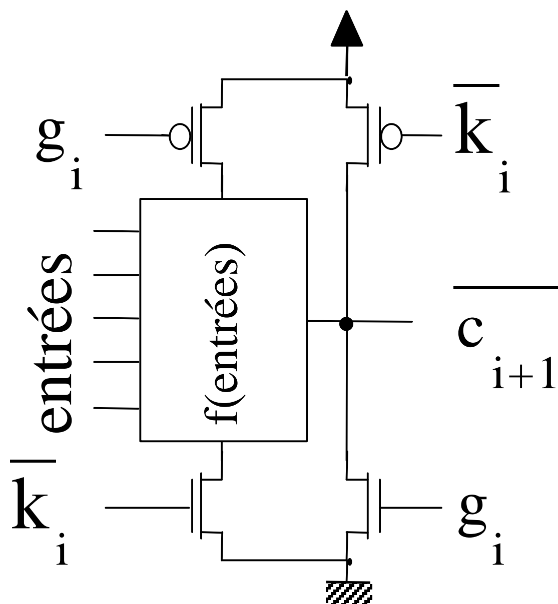
Définissons g_i et p_i de la façon suivante:

$g_i = a_i \wedge b_i$ *génération de retenue au rang i*

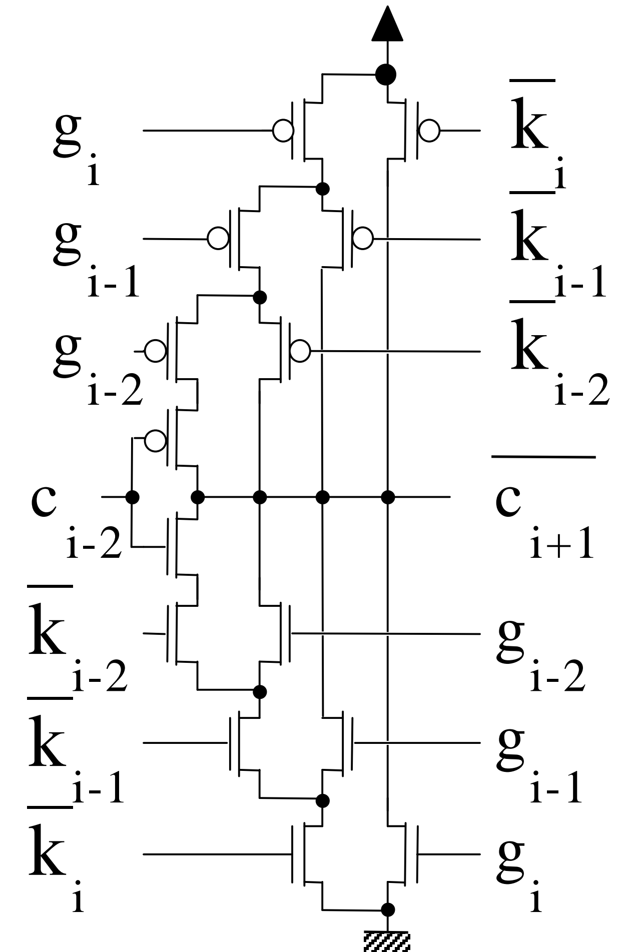
$\overline{k}_i = a_i \vee b_i$ *propagation de retenue au rang i*

$c_{i+1} = g_i \vee \overline{k}_i \wedge (g_{i-1} \vee \overline{k}_{i-1} \wedge (g_{i-2} \vee \overline{k}_{i-2} \wedge c_{i-2}))$

$c_{i+1} = g_i \vee \overline{k}_i \wedge g_{i-1} \vee \overline{k}_i \wedge \overline{k}_{i-1} \wedge g_{i-2} \vee \overline{k}_i \wedge \overline{k}_{i-1} \wedge \overline{k}_{i-2} \wedge c_{i-2}$



a_i	b_i	$\overline{k}_i$	g_i	$\overline{c_{i+1}}$
0	0	0	0	1
0	1	1	0	f(entrées)
1	0	1	0	f(entrées)
1	1	1	1	0

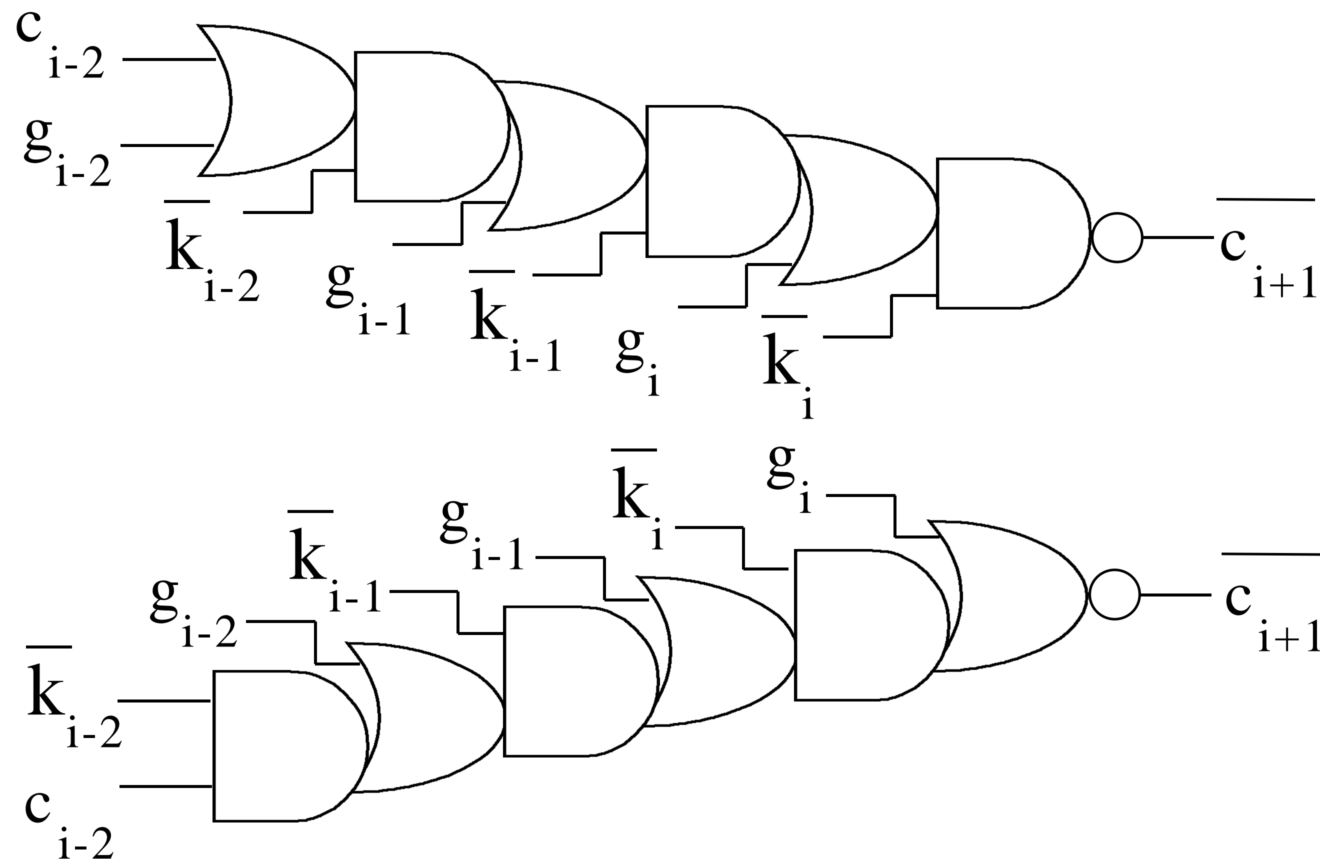


Application récursive

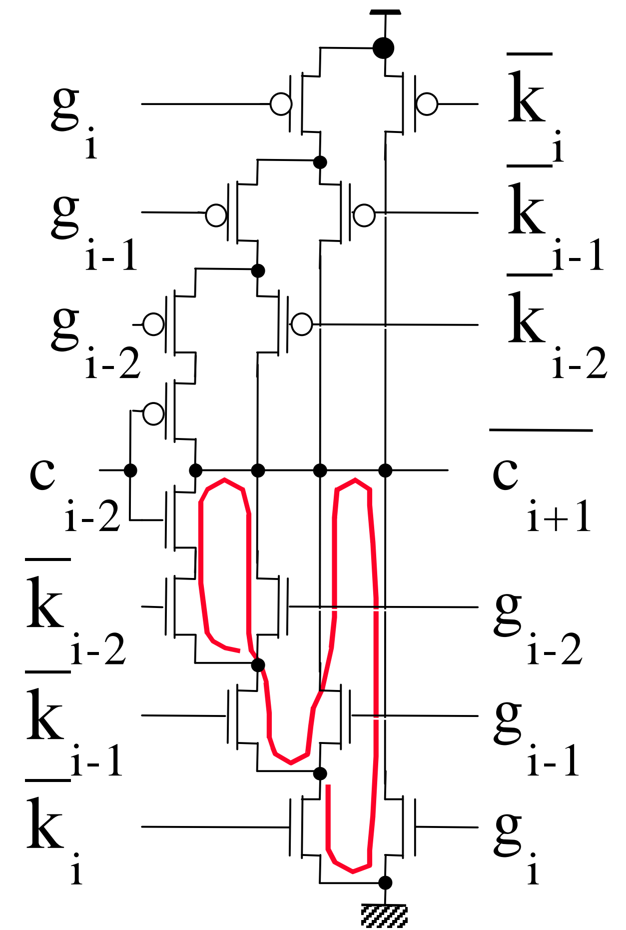
f(entrées) = c_i

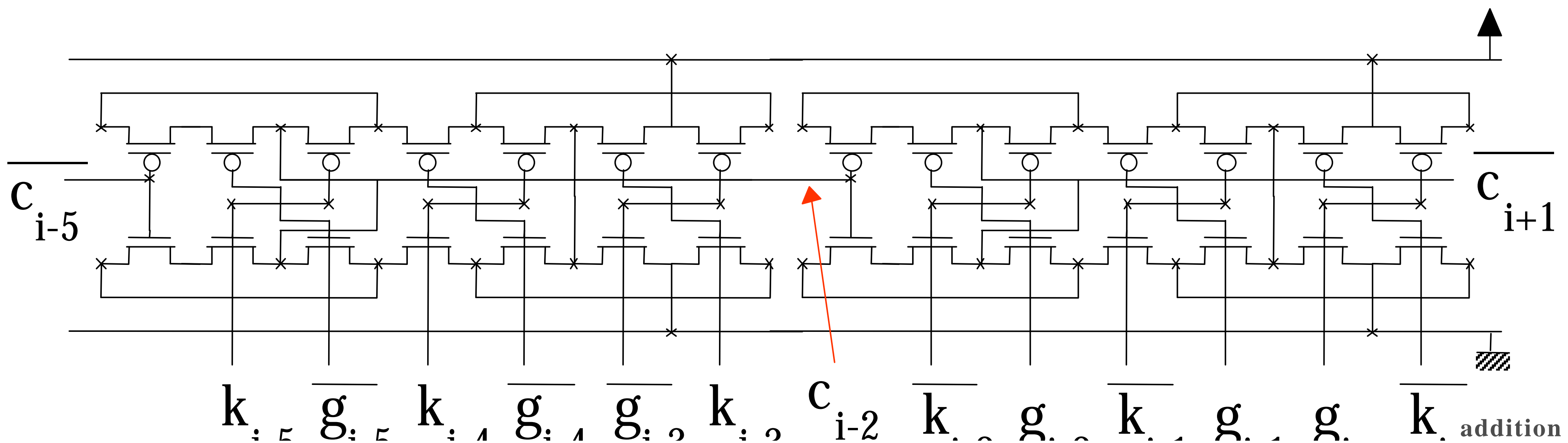
Anticipation du calcul de la retenue (2)

$$\overline{k}_i \geq g_i \Rightarrow c_{i+1} = g_i \vee \left(\overline{k}_i \wedge c_i\right) = \overline{k}_i \wedge \left(g_i \vee c_i\right)$$



Mise en ligne des transistors de l'anticipation



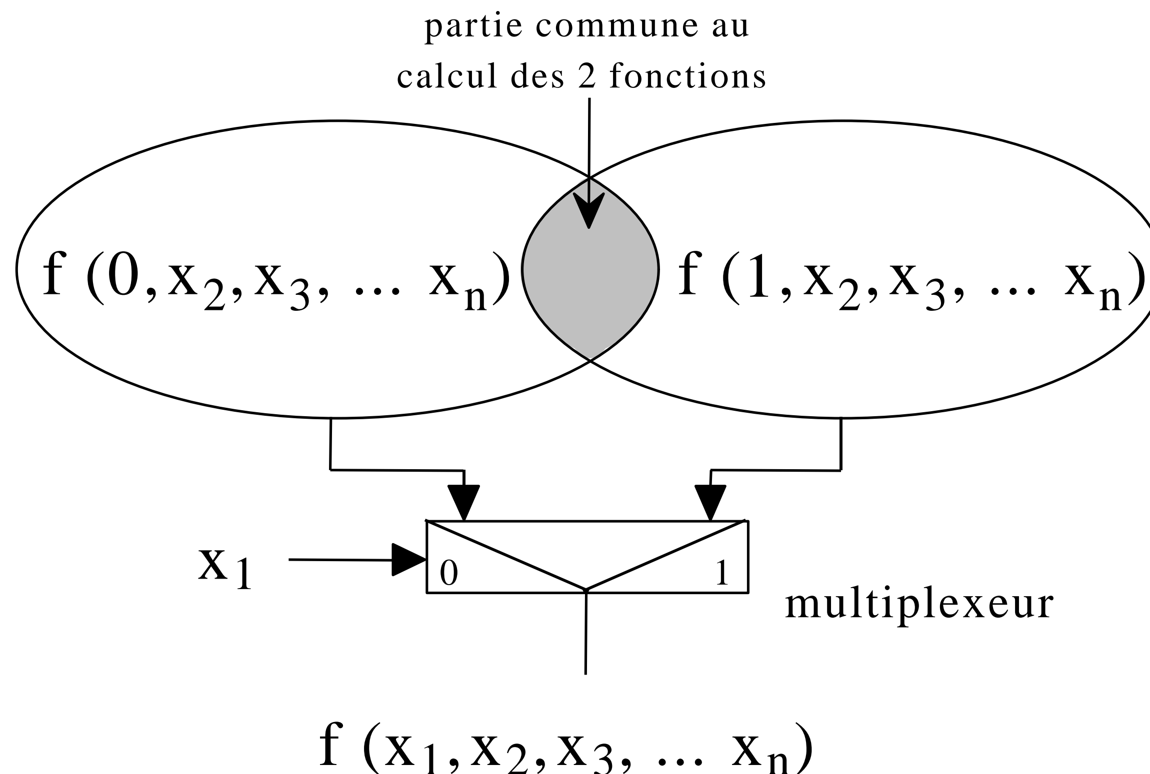


Écriture de Shannon

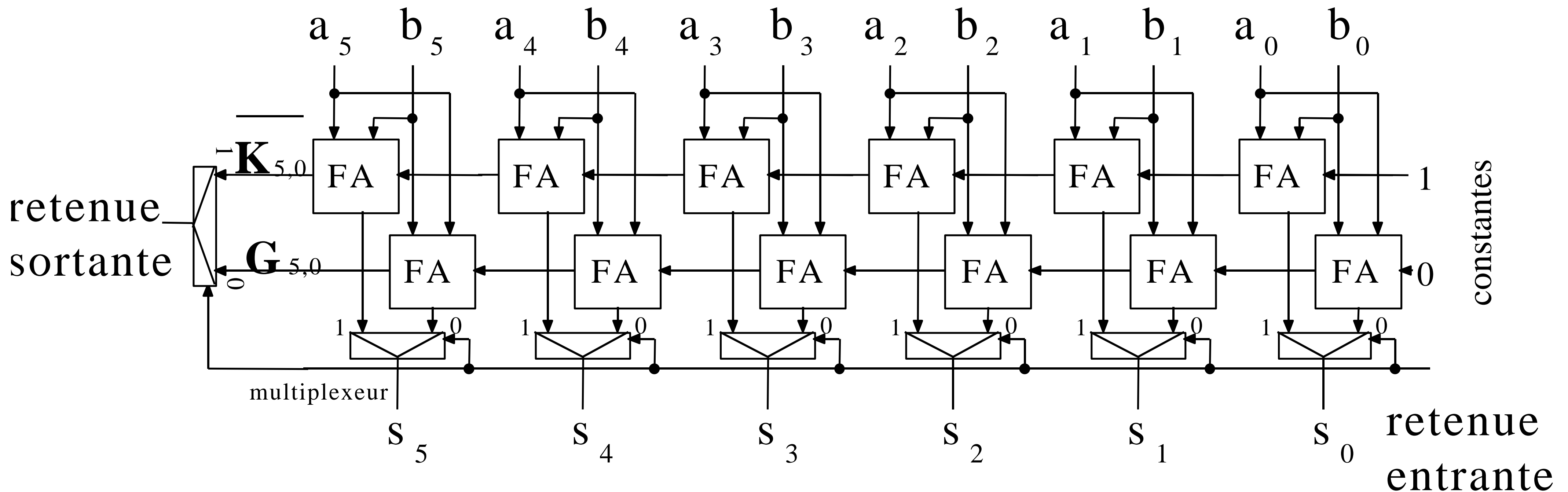
On doit réaliser $f(x_1, x_2, x_3, \ldots x_n)$

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

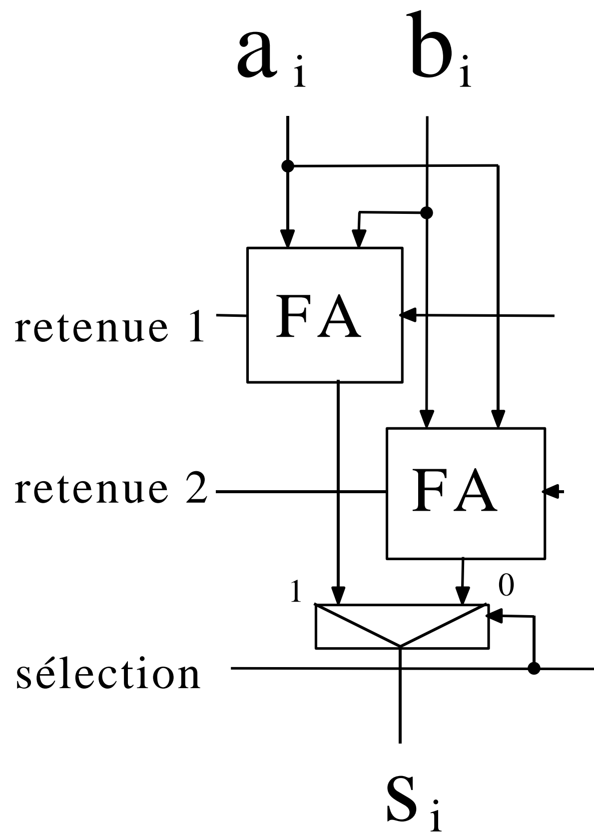
⇒ on dispose de temps pour la calculer

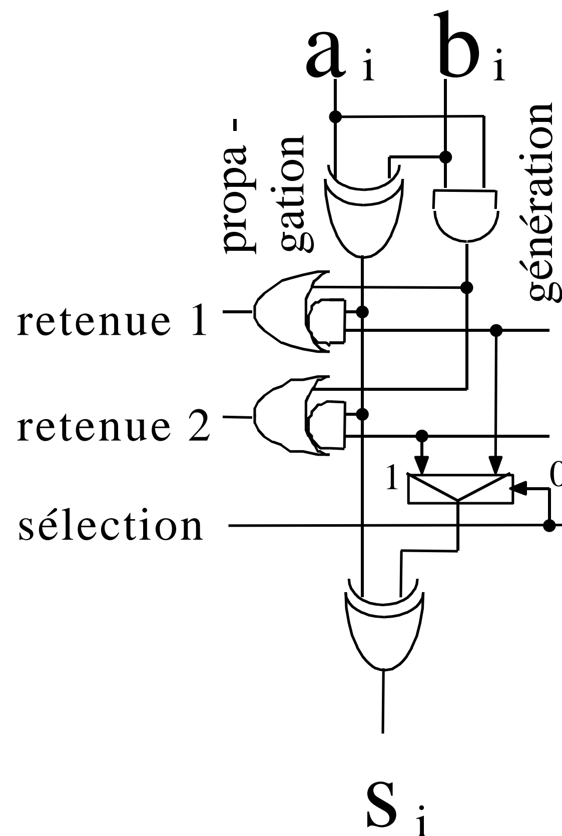
Définissons $\mathbf{G}_{i,j}$ et $\mathbf{K}_{i,j}$ de la façon suivante

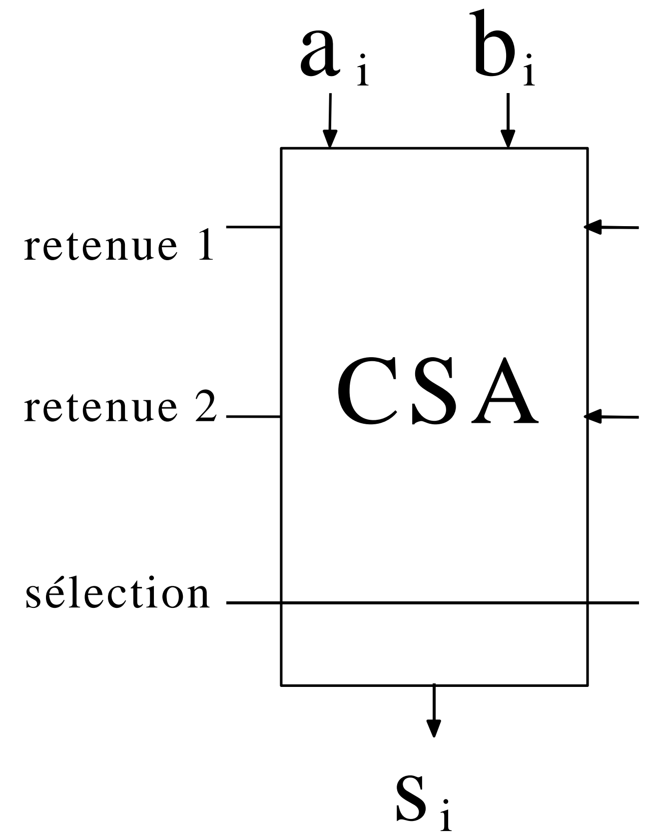
$\mathbf{G}_{i,j}$ = *génération de la retenue i entre le rang i et le rang k*

$\mathbf{K}_{i,j}$ = *destruction de la retenue i entre le rang i et le rang k*

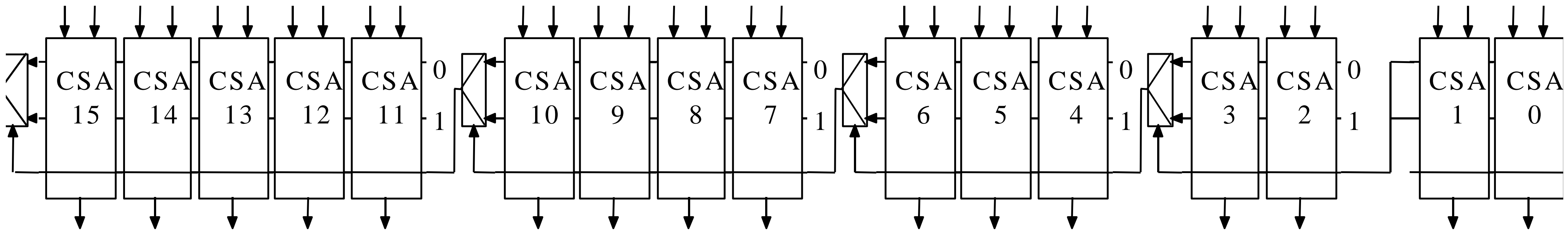
Cellule de carry select adder

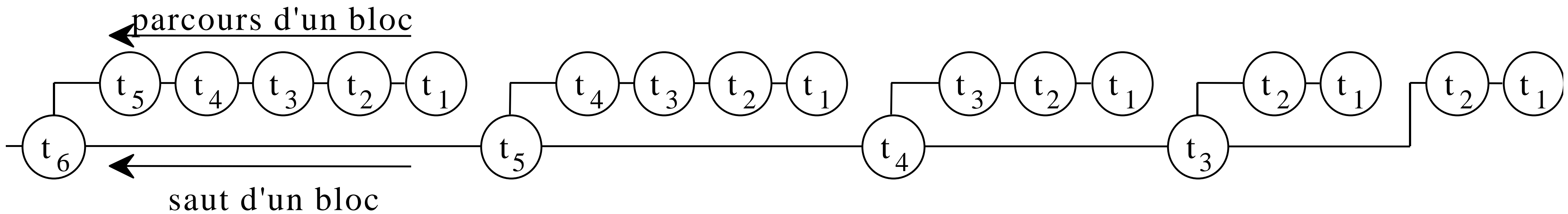






Additionneur en temps $\sqrt{n}$

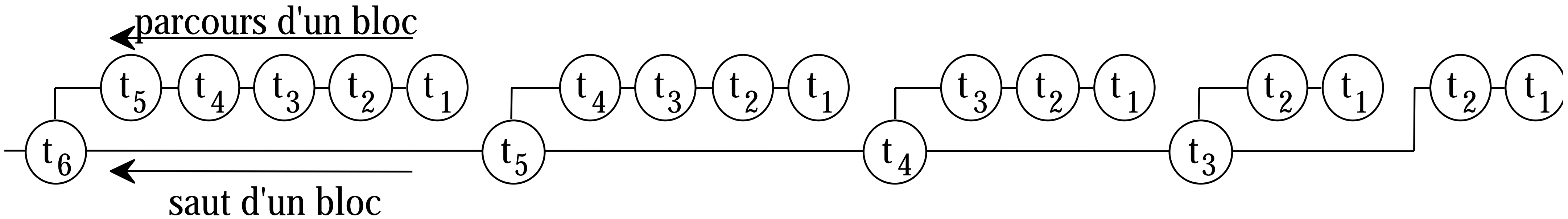


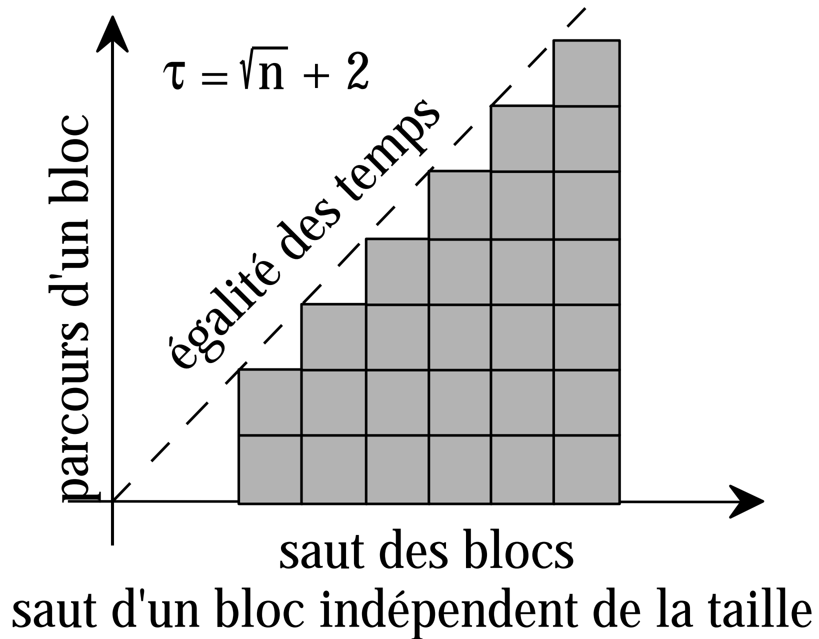


$$\sum_{i=1}^{\tau-2} i = \frac{(\tau-2)*(\tau-3)}{2} \approx \frac{(\tau-2)^2}{2}$$

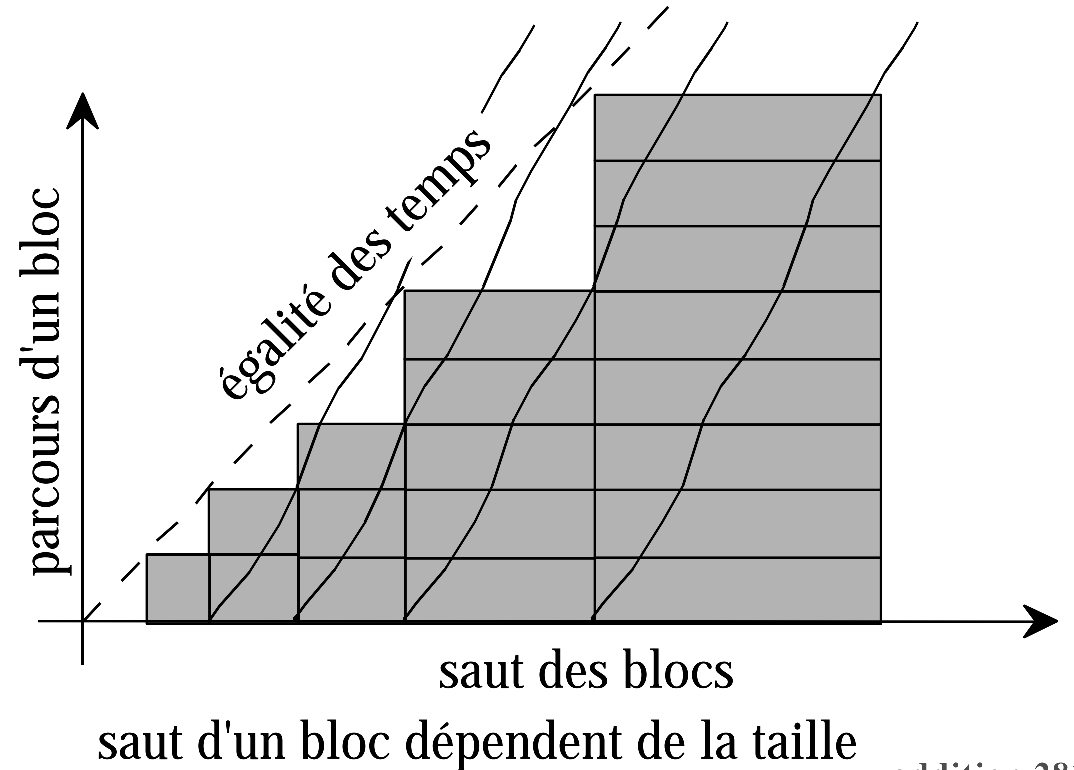
$$\tau = \sqrt{n} + 2$$

Méthode graphique pour équilibrer les délais d'un CSA



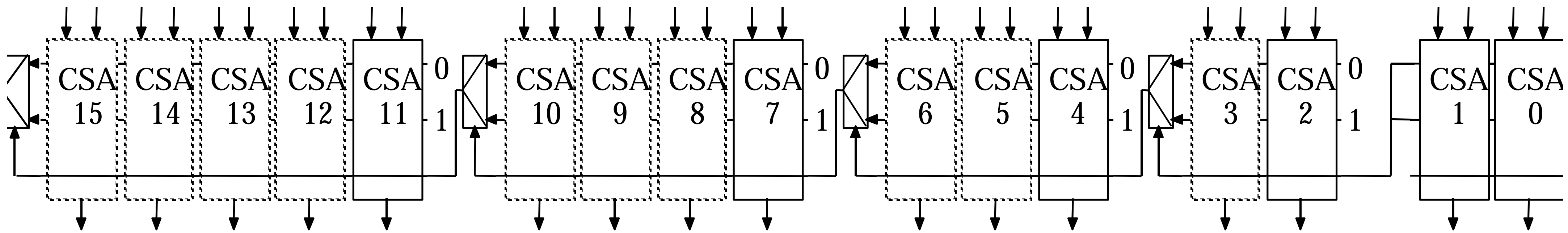


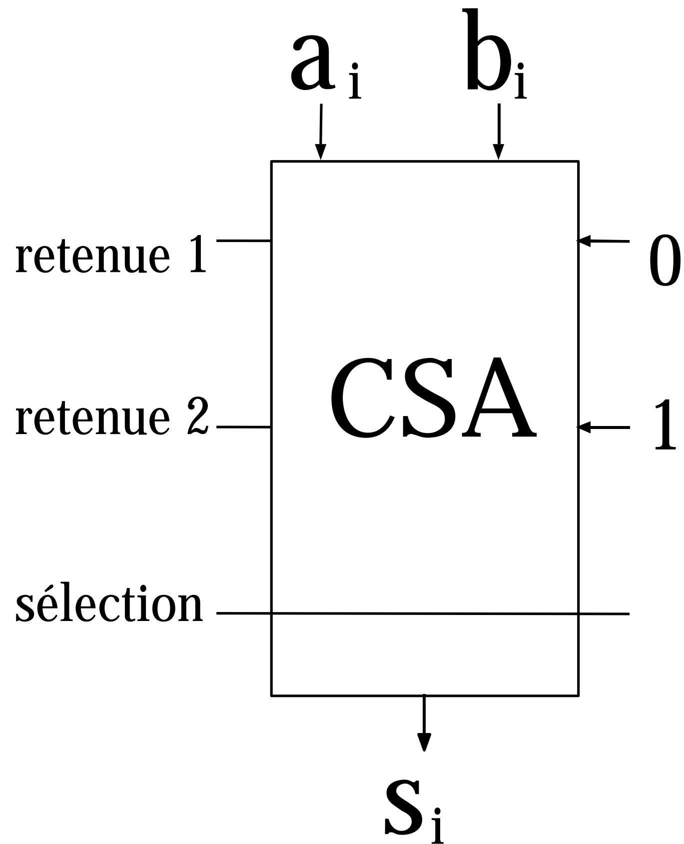
saut d'un bloc indépendent de la taille

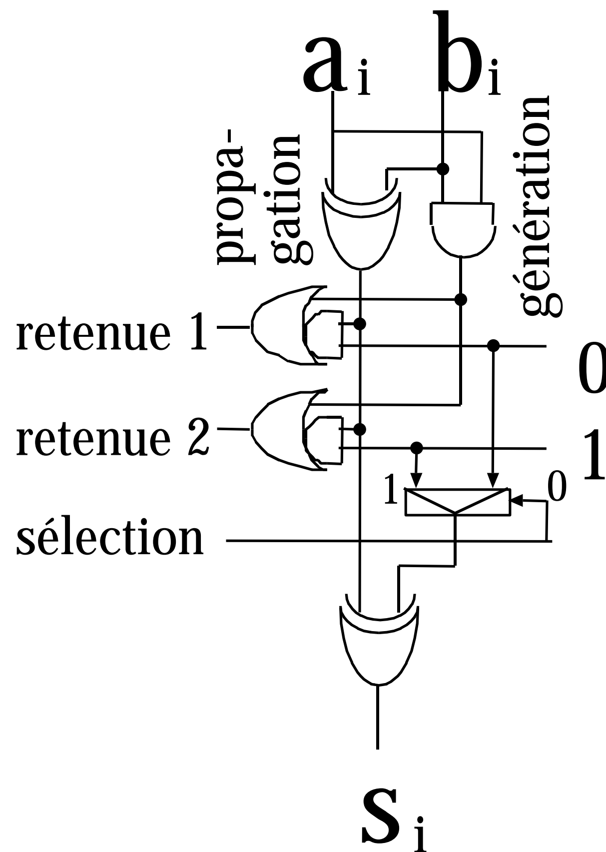


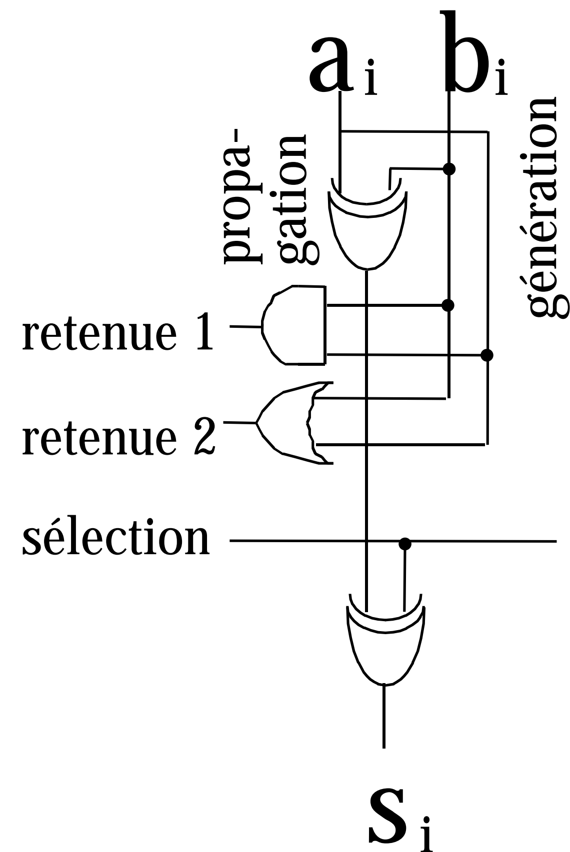
saut d'un bloc dépendent de la taille

Première cellule d'un bloc de CSA





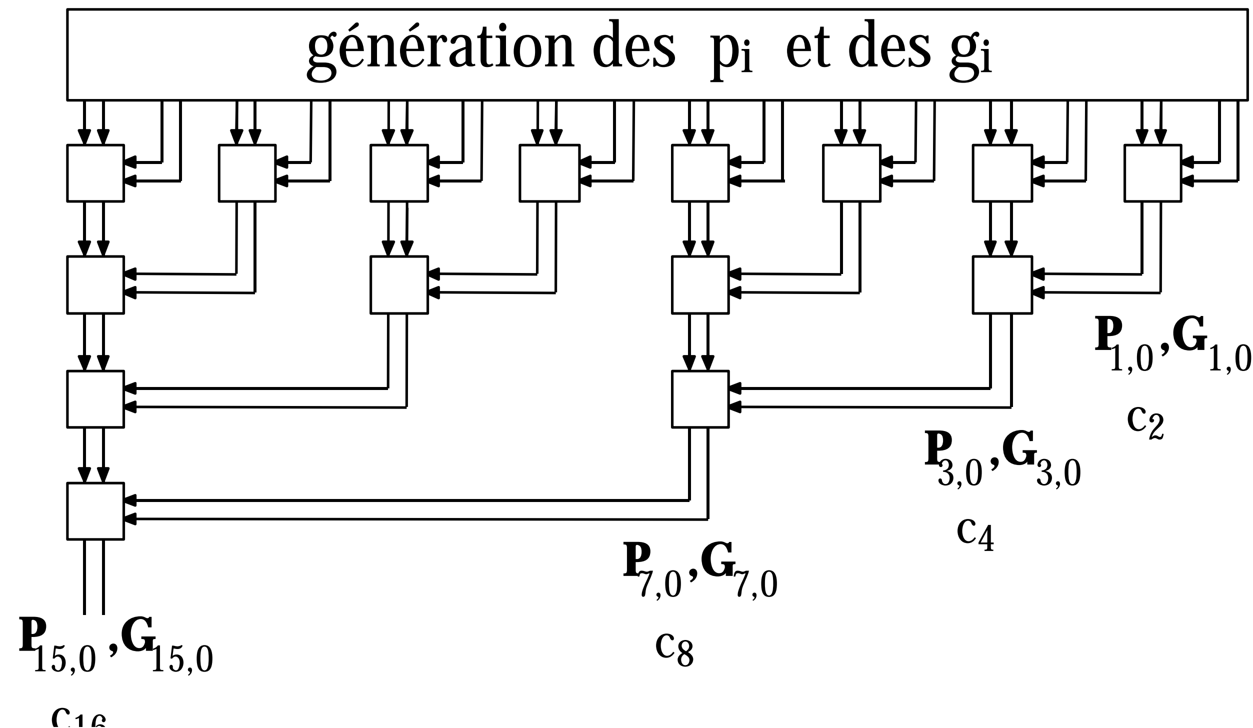




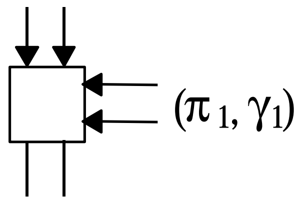
Additionneur en temps $\log_2 (n)$

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

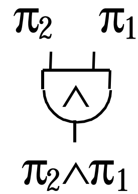
$\mathbf{G}_{i,i} = g_i = a_i \wedge b_i$	*génération de retenue au rang i*
$\mathbf{P}_{i,i} = p_i = a_i \oplus b_i$	*propagation de retenue au rang i*
$\mathbf{G}_{i,k} = \mathbf{G}_{i,j} \vee \mathbf{P}_{i,j} \wedge \mathbf{G}_{j-1,k}$	*génération de retenue entre le rang i et le rang k* $(n \geq i \geq j > k \geq 0)$
$\mathbf{P}_{i,k} = \mathbf{P}_{i,j} \wedge \mathbf{P}_{j-1,k}$	*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*

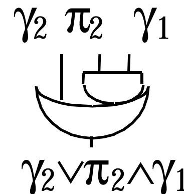


Cellule de Brent et Kung pour calculer les "propagation de groupe" et "génération de groupe"

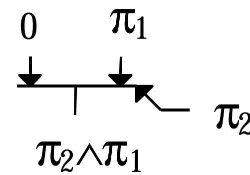


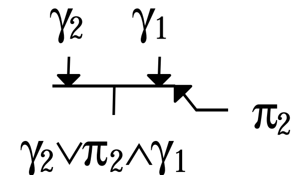
$(\pi_2 \wedge \pi_1 , \gamma_2 \vee \pi_2 \wedge \gamma_1)$





Réalisation à portes logiques

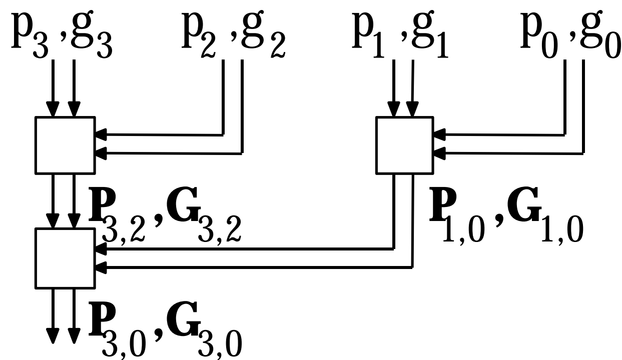


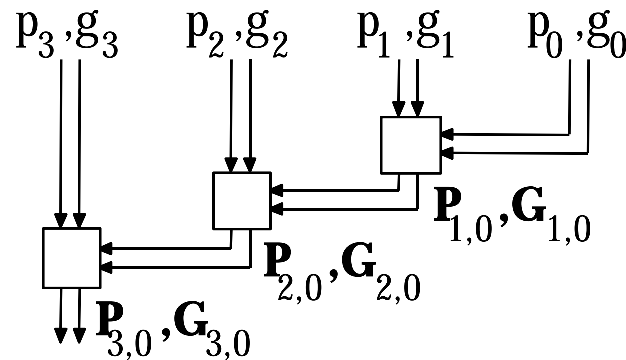


Réalisation à multiplexeurs

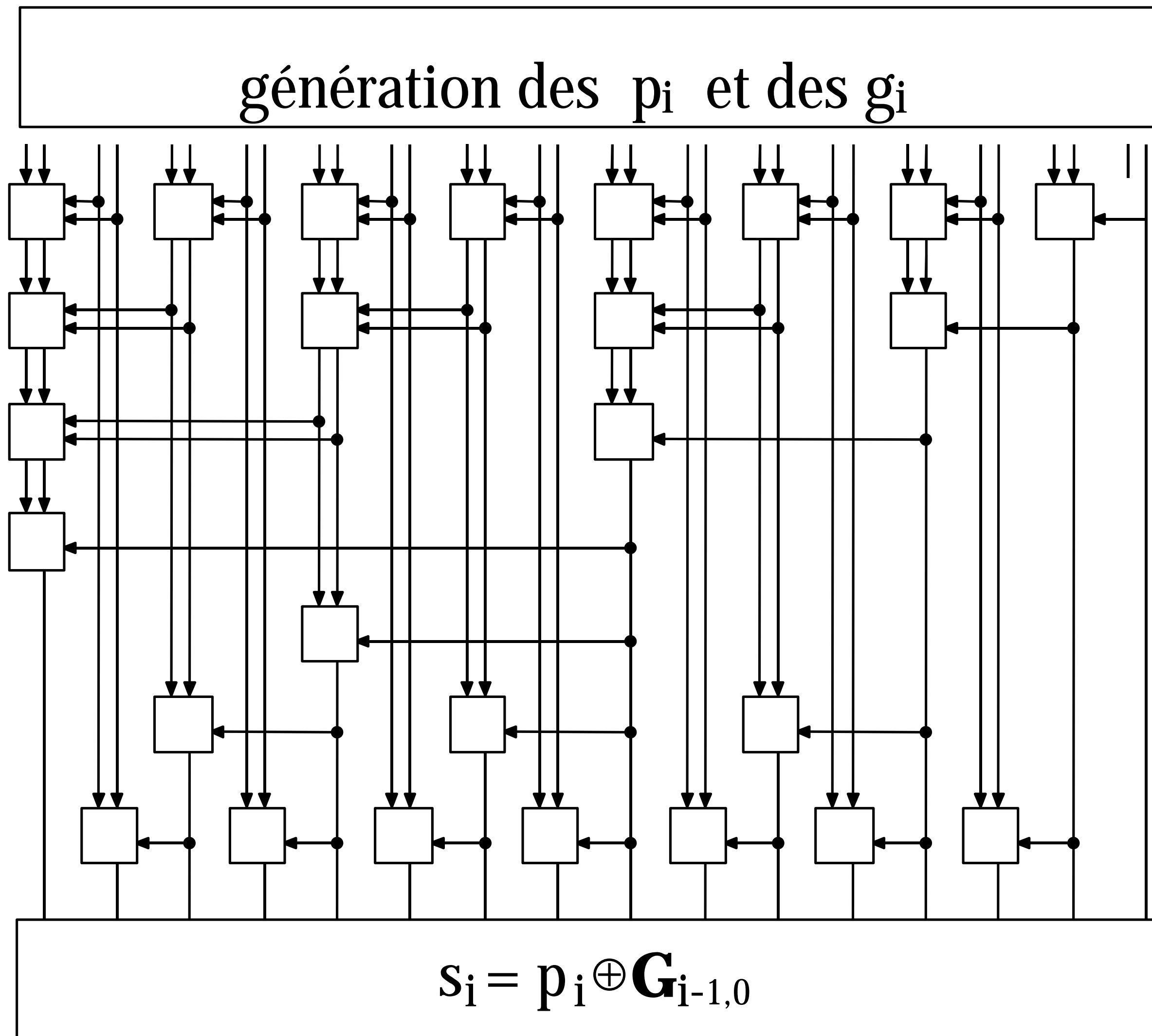
- Associative
- Non commutative
- Idempotente
- Non décroissante (inverseurs)

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

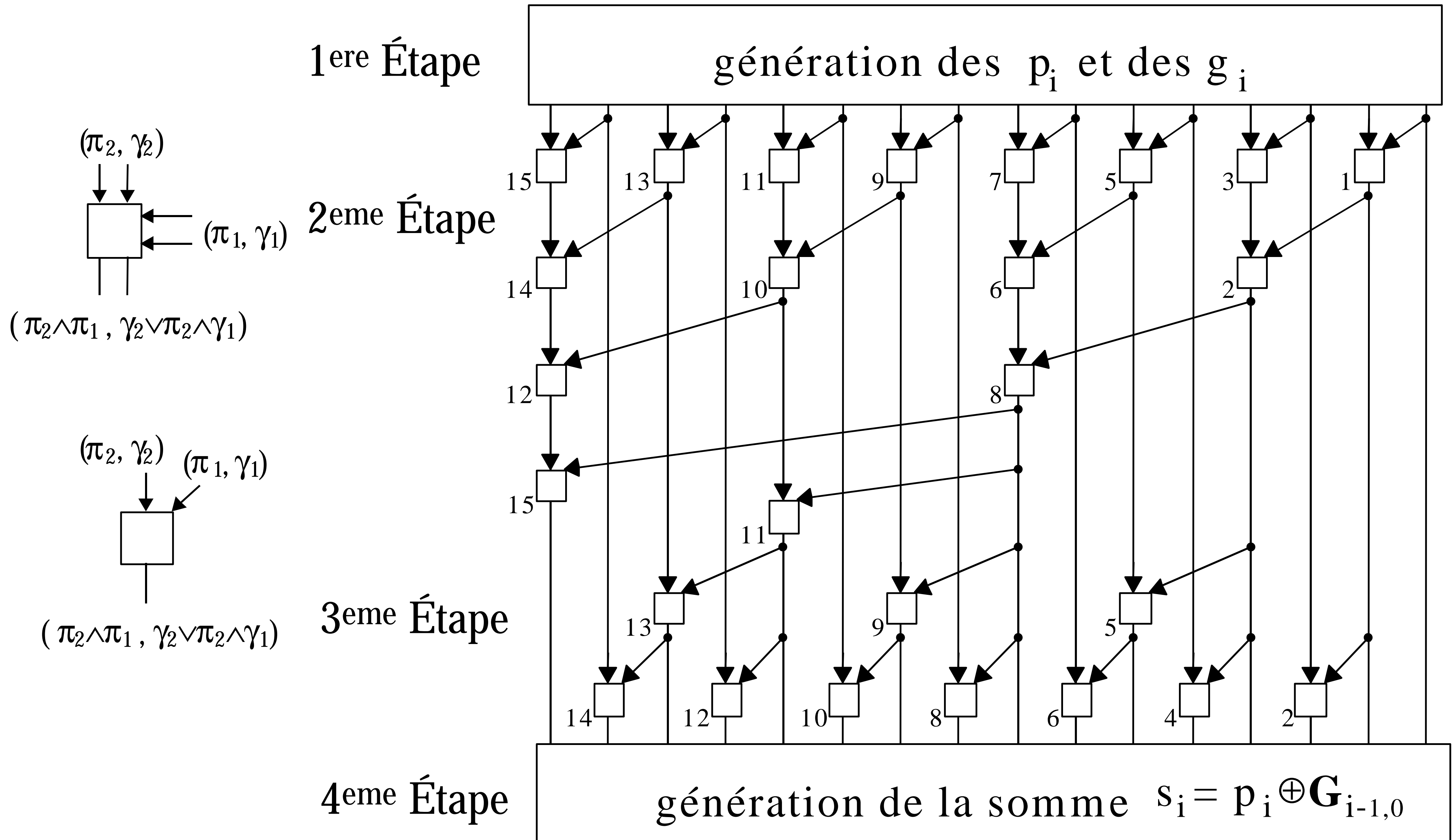




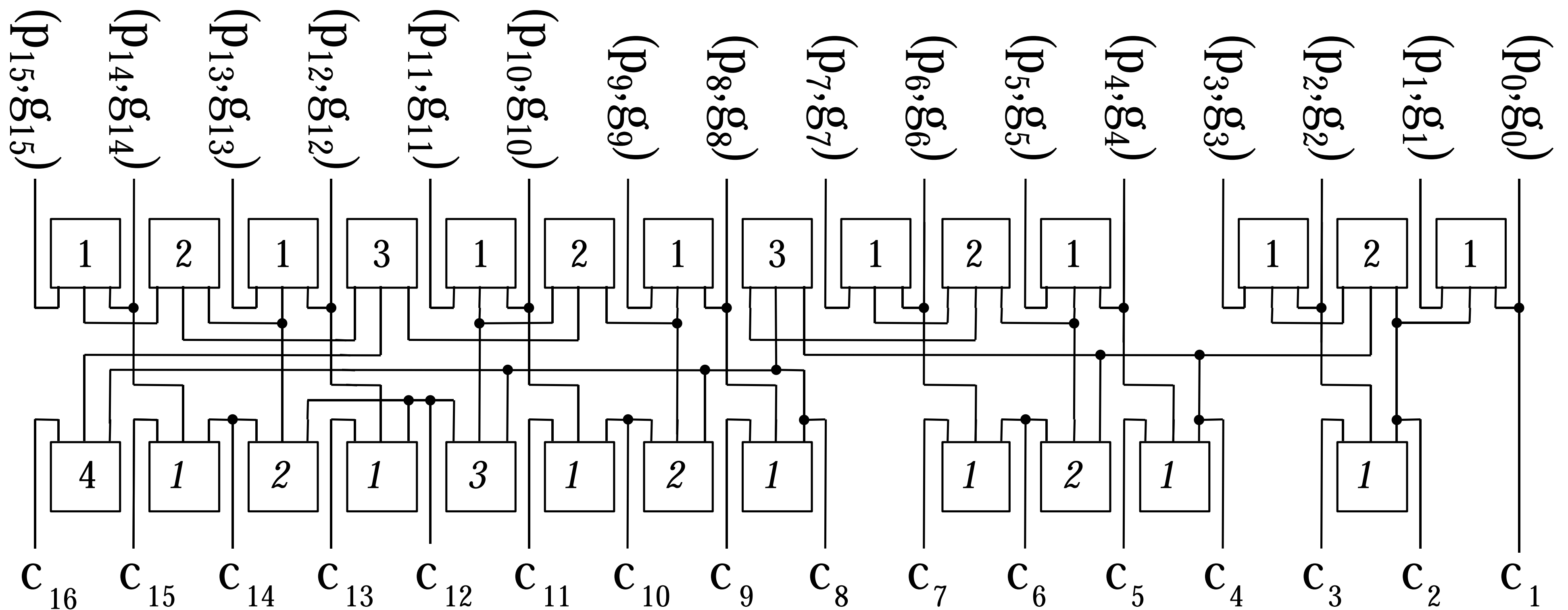
Additionneur de Brent et Kung en temps $\log_2(n)$



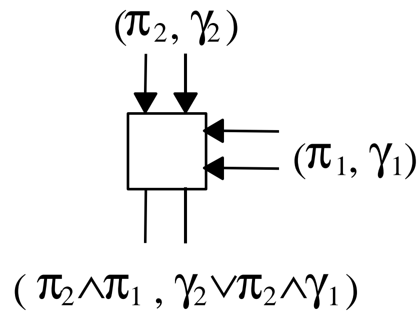
Additionneur de Brent et Kung

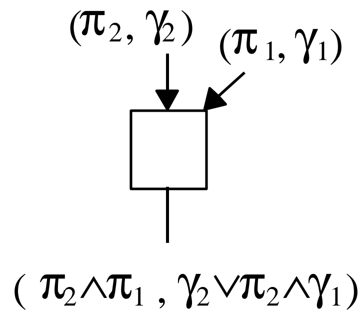


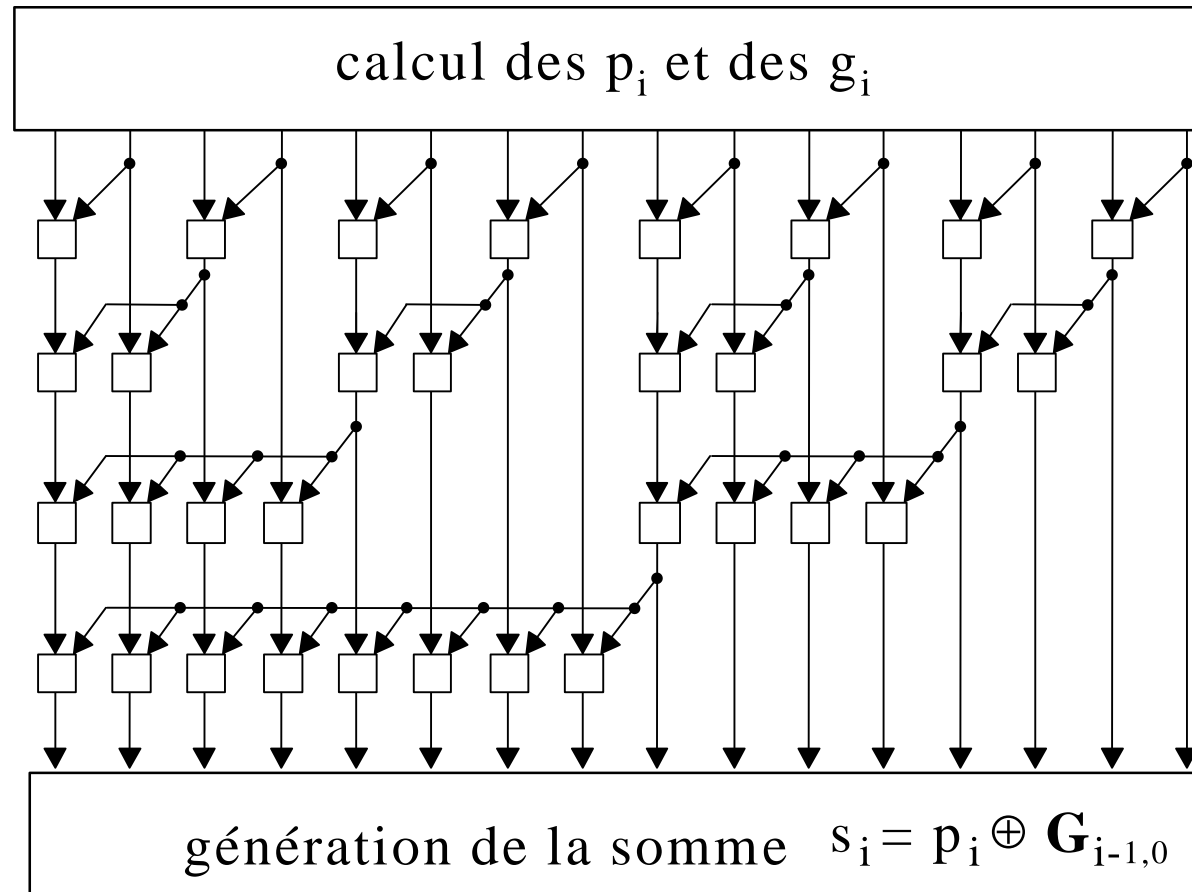
Mise à plat des deux arbres binaires (16 bits)



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





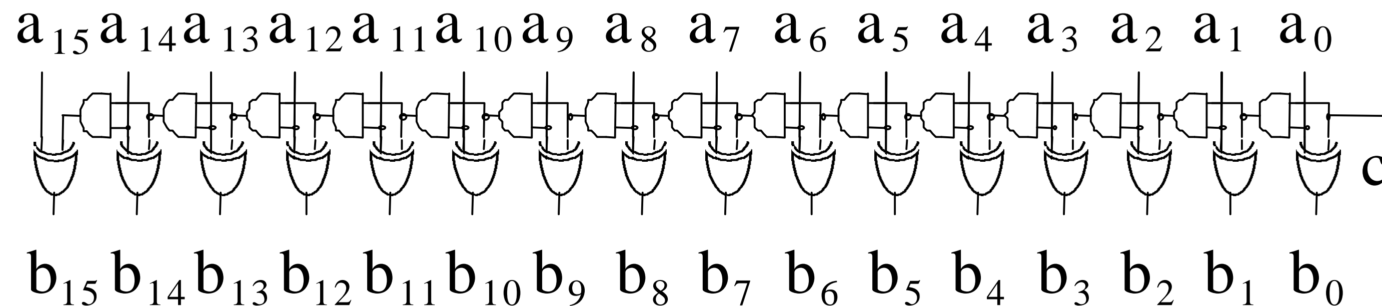


Incrémenteur en temps $\log_2(n)$

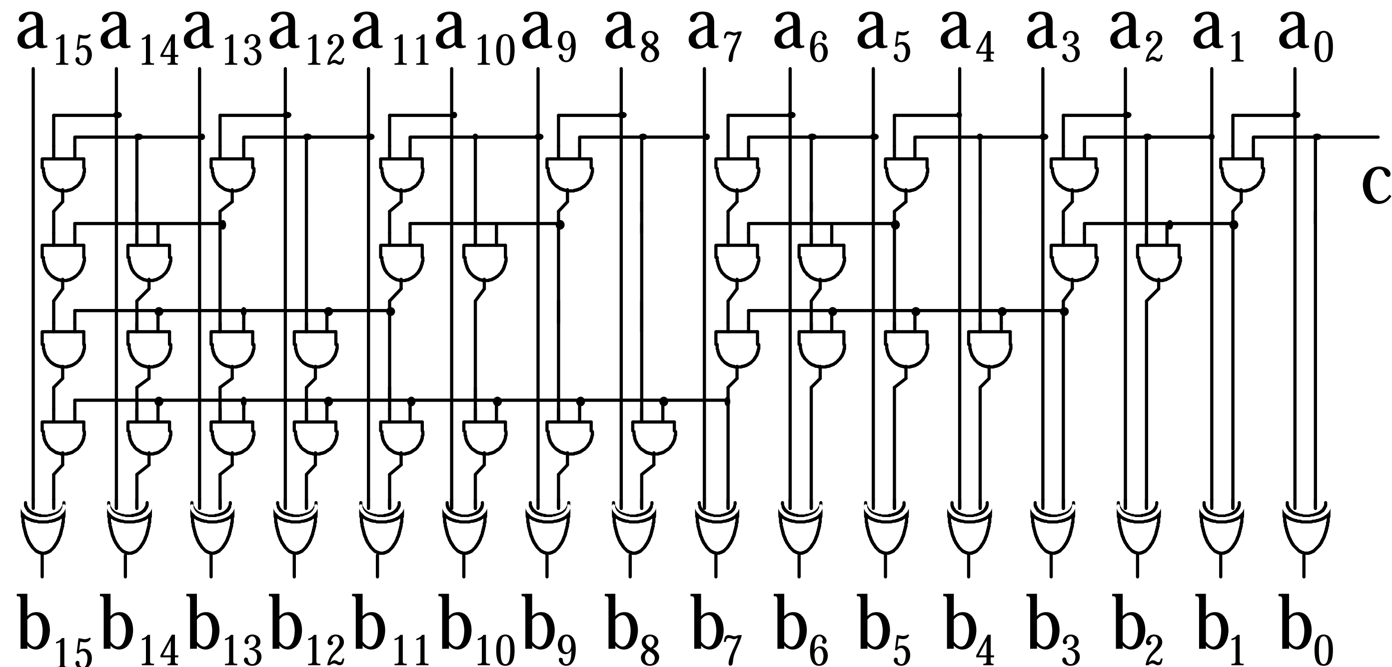
Dans un incrémenteur B := A + c,

les génération **G** sont toujours 0. On ne calcule que les **P**

incrémenteur à propagation

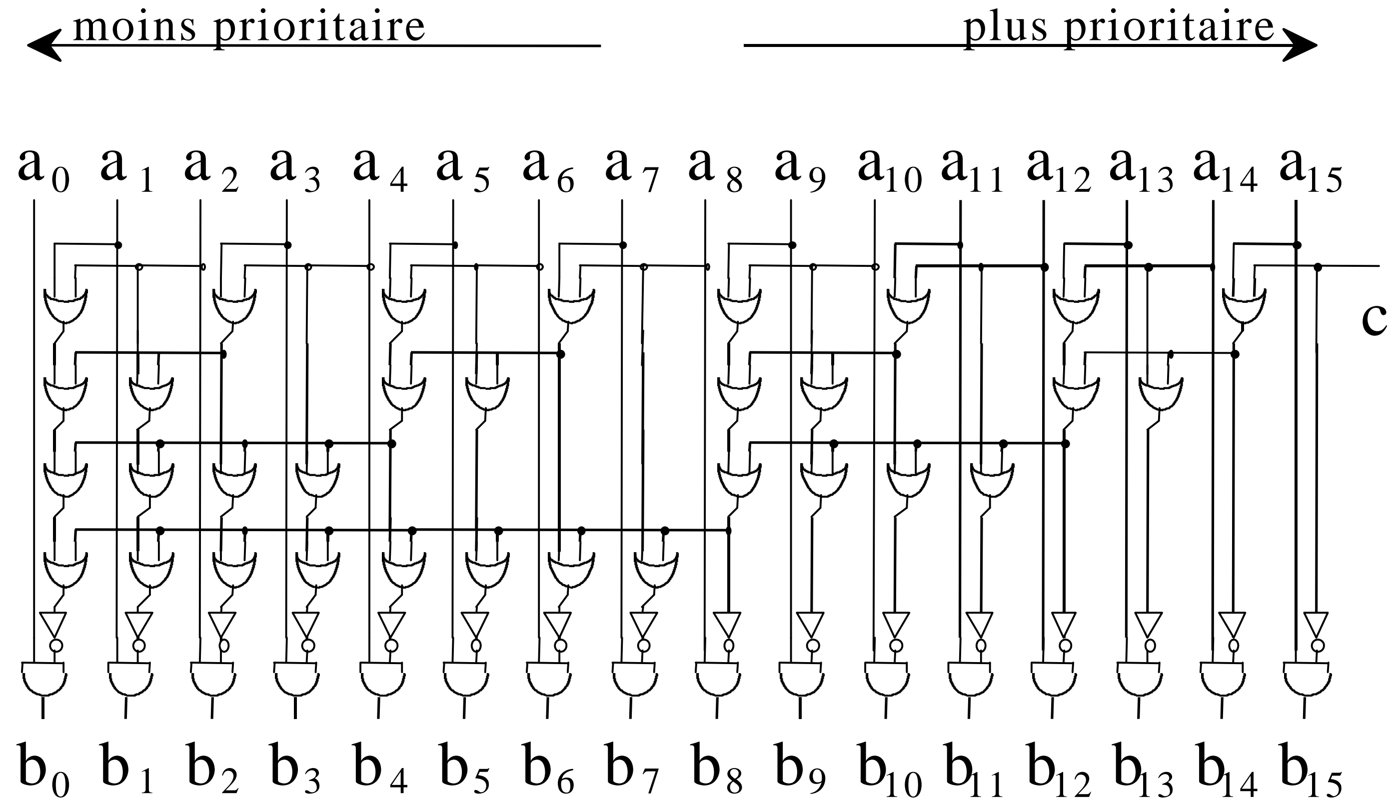


calcul des **P** inspiré du précédent



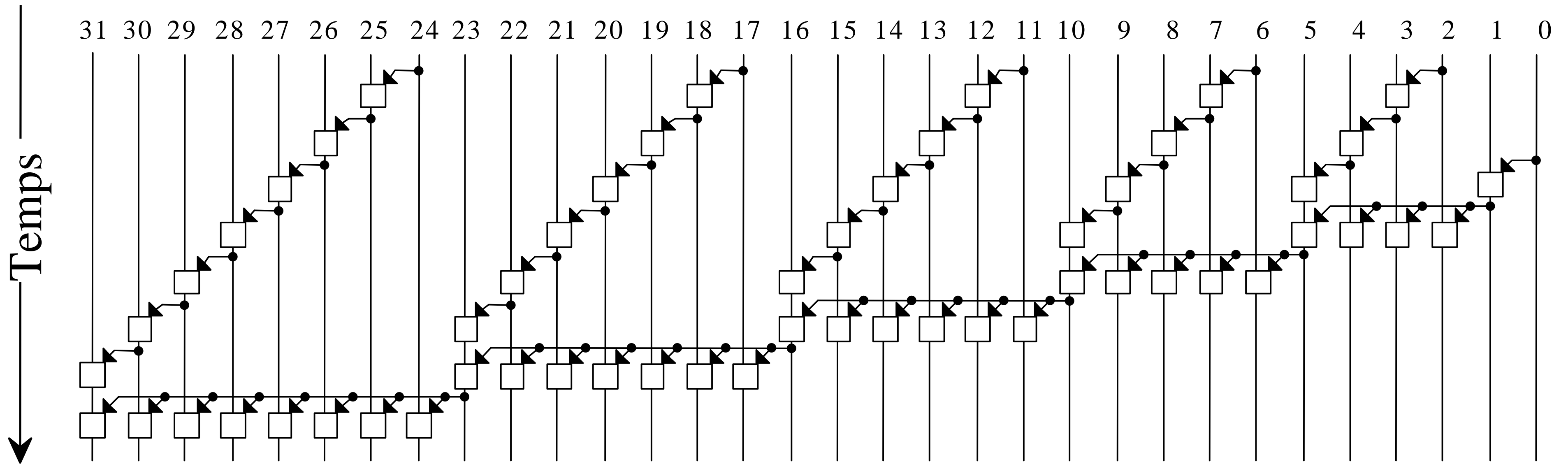
Priorité en temps $\log_2$ (n)

calcul du masque inspiré du calcul des **P** précédent

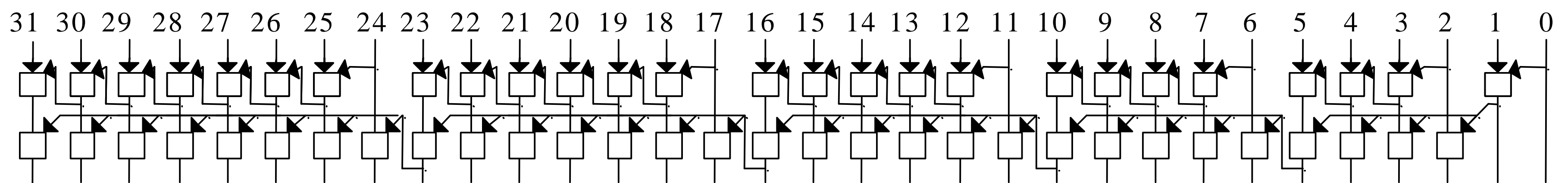


Attention: l'ordre des bits a été inversé sur ce schéma.
Ce circuit est utilisé dans les opérateurs virgule flottante pour déterminer le chiffe le plus significatif d'un nombre en vue d'éliminer les zéros en poids forts (non significatifs).

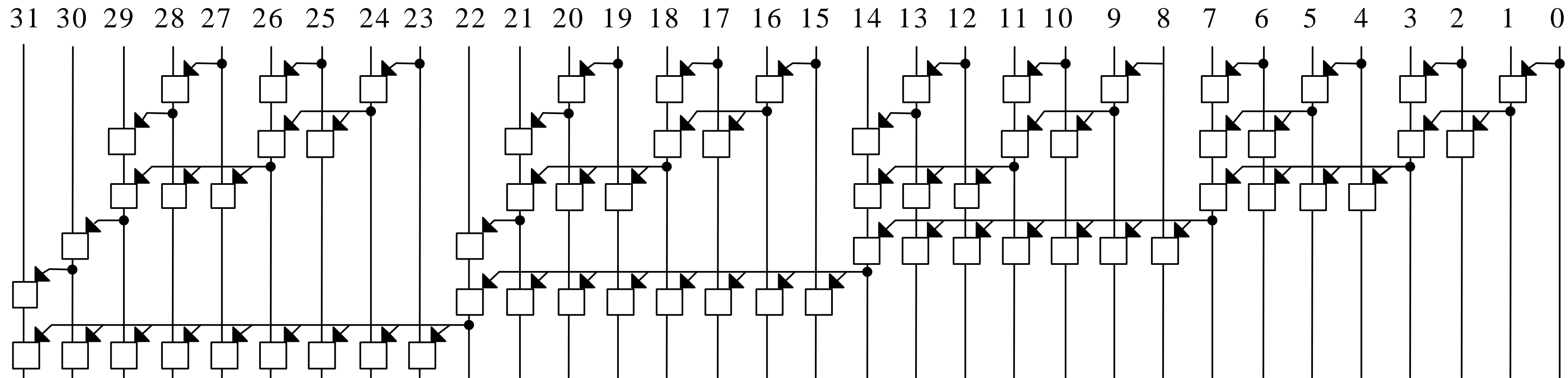
Additionneur en temps $\sqrt{n}$

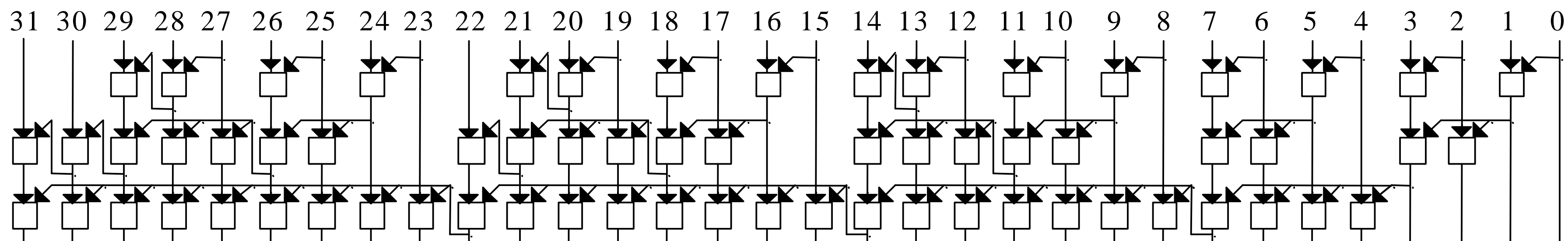


$\sqrt[2]{n}$



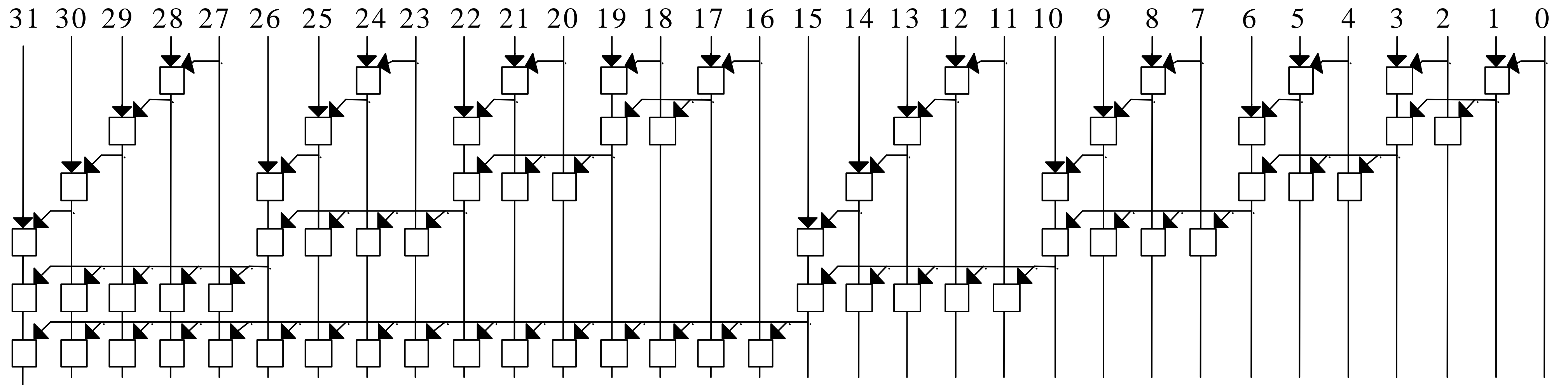
Additionneur en temps $\sqrt[3]{n}$

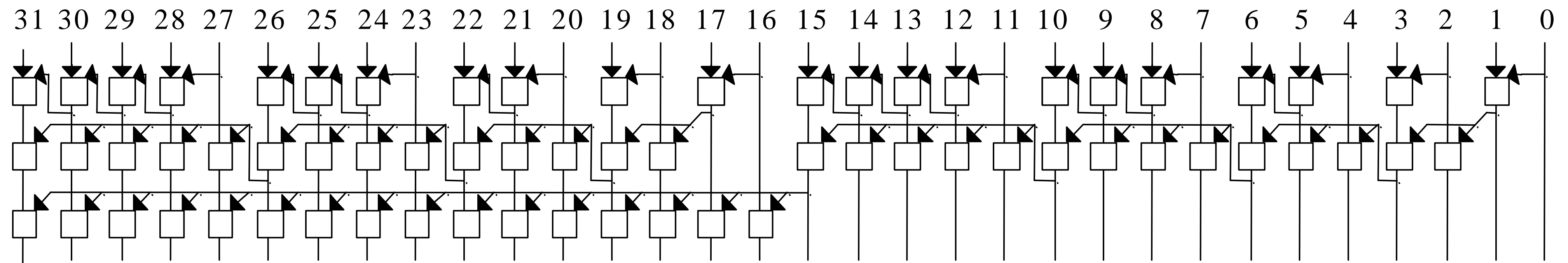




Pour un délai τ le nombre n de bits est : $\sum_{i=1}^{\tau} \sum_{j=1}^{i} j \Rightarrow \quad \tau = \sqrt[3]{n}$

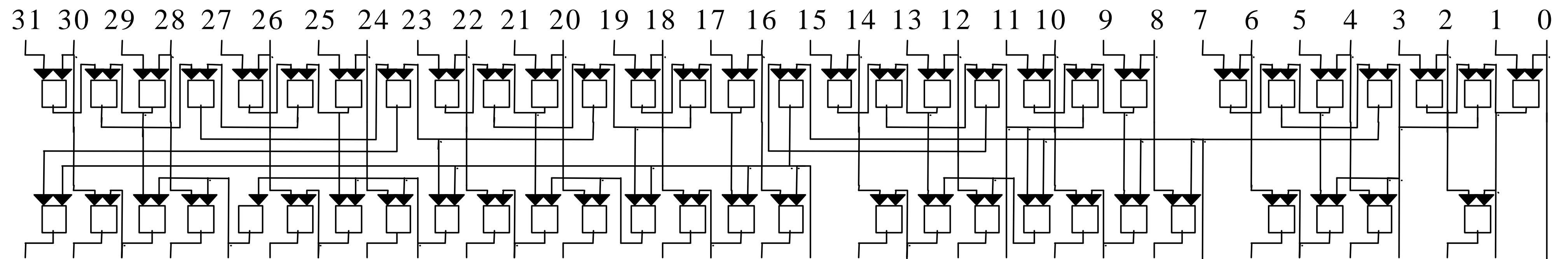
Additionneur mixte racine-log



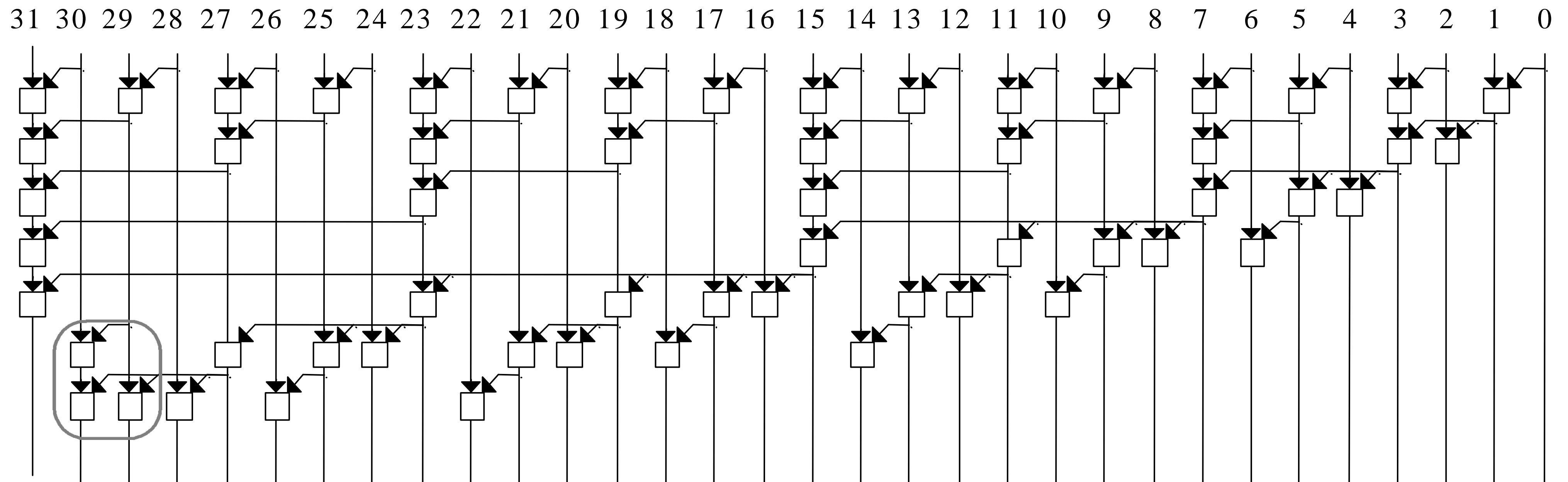


Additionneur de Brent et Kung

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

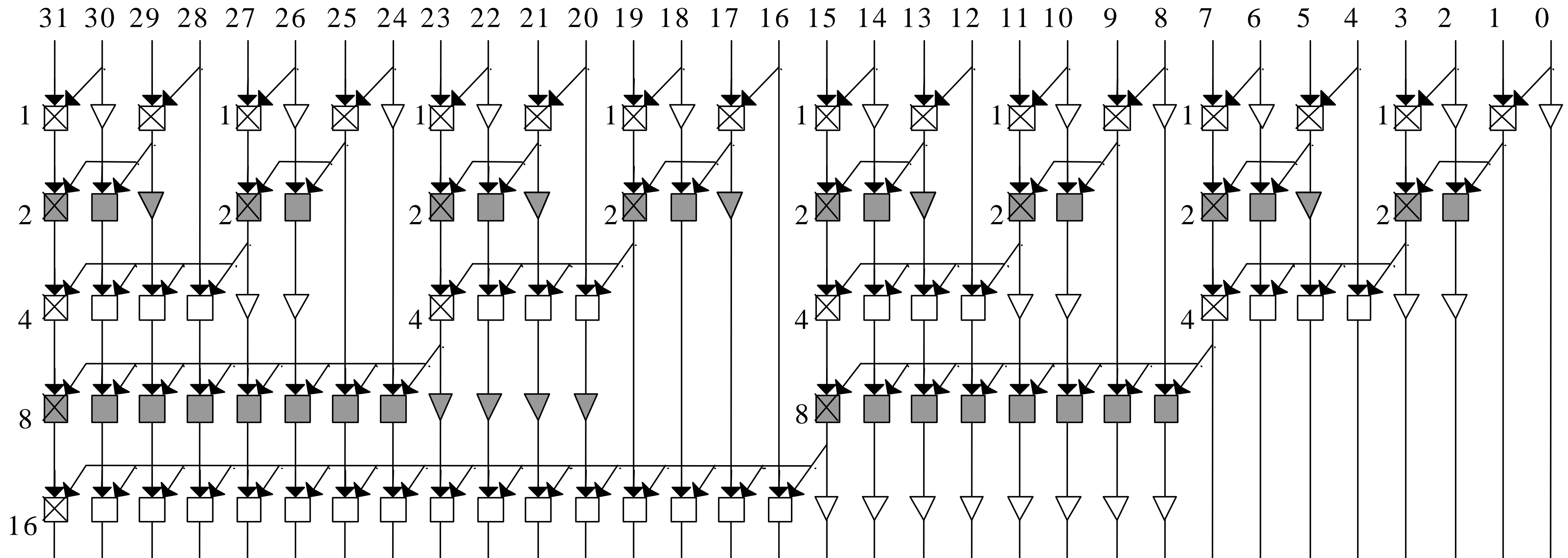


Additionneur de Brent et Kung modifié



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

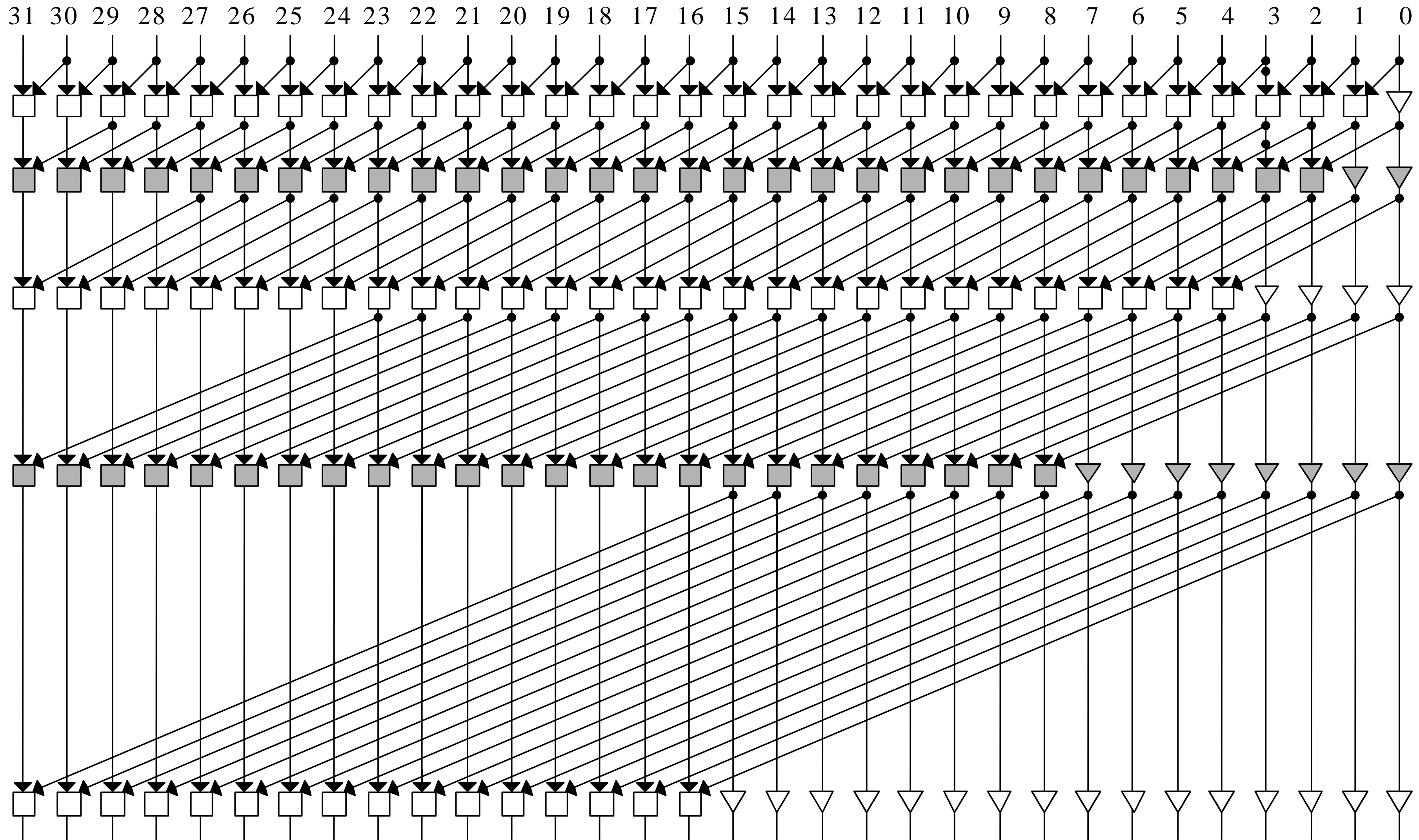
Dimensionnement de l'additionneur de Sklanski



Le "fan-out" d'une cellule avec un "fan-in" de K est

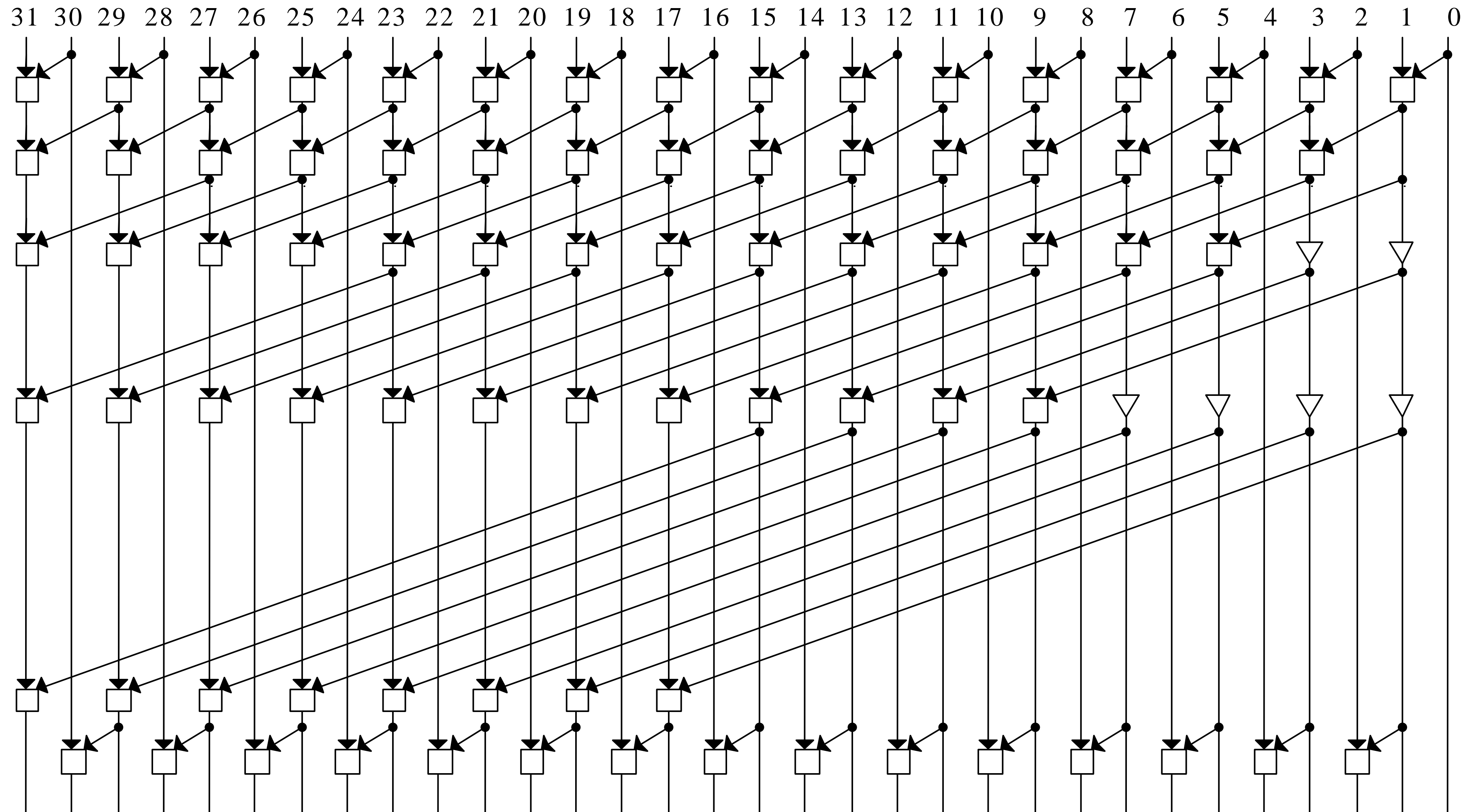
2K fan-in 1 + 1 fan-in K = 4 K fan-in

Additionneur de Kogge et Stone



Le "fan-out" est toujours 2

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	$/2 \quad +1$	ci-dessus -1	$\lceil 2\log_2(n) - 2 \rceil$	58	7	5
Sklansky	$\lceil n/2 \log_2(n) \rceil$	$\lceil \log_2(n) \rceil$	$n/2$	80	5	16
Kogge and Stone	$n(\log_2(n)-1)\rceil$	$\lceil \log_2(n) \rceil$	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

Additionneur en temps $\log_2(n)$ à arbre unique

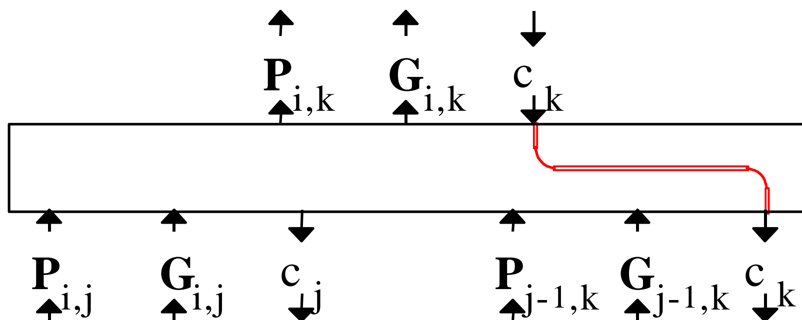
$P_{7,0}$ $G_{7,0}$ c_0

$P_{7,4}$ $G_{7,4}$ c_4 $P_{3,0}$ $G_{3,0}$ c_0

$P_{7,6}$ $G_{7,6}$ c_6 $P_{5,4}$ $G_{5,4}$ c_4 $P_{3,2}$ $G_{3,2}$ c_2 $P_{1,0}$ $G_{1,0}$ c_0

p_7 g_7 c_7 p_6 g_6 c_6 p_5 g_5 c_5 p_4 g_4 c_4 p_3 g_3 c_3 p_2 g_2 c_2 p_1 g_1 c_{71} p_0 g_0 c_0

s_7 a_7 b_7 s_6 a_6 b_6 s_5 a_5 b_5 s_4 a_4 b_4 s_3 a_3 b_3 s_2 a_2 b_2 s_1 a_1 b_{71} s_0 a_0 b_0



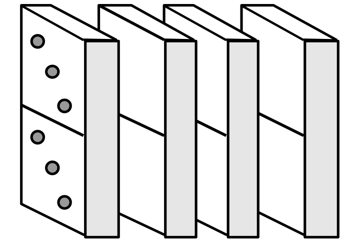
$$G_{i,k} = G_{i,j} \vee P_{i,j} \wedge G_{j-1,k} \qquad (n \geq i \geq j > k \geq 0)$$

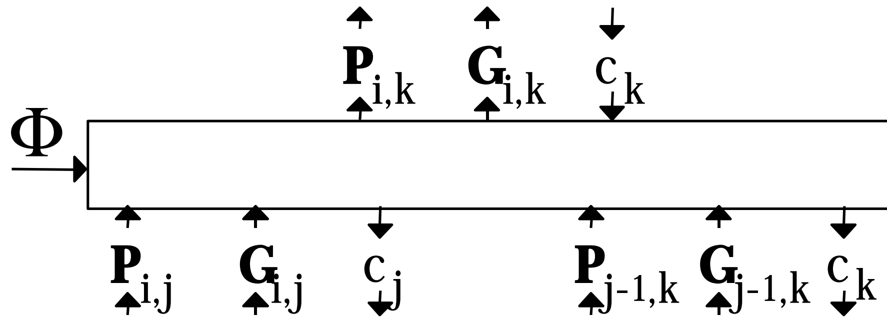
$$P_{i,k} = P_{i,j} \wedge P_{j-1,k}$$

$$c_j = G_{j-1,k} \vee P_{j-1,k} \wedge c_k$$

c_k restauré

Additionneur en temps $\log_2(n)$
Réalisation en Domino

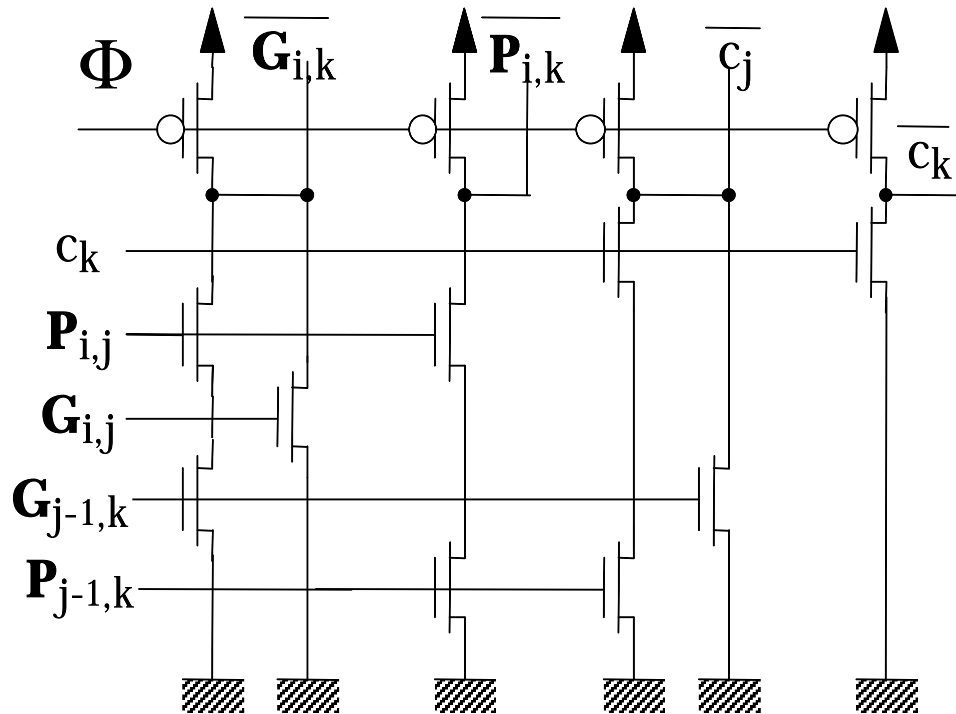


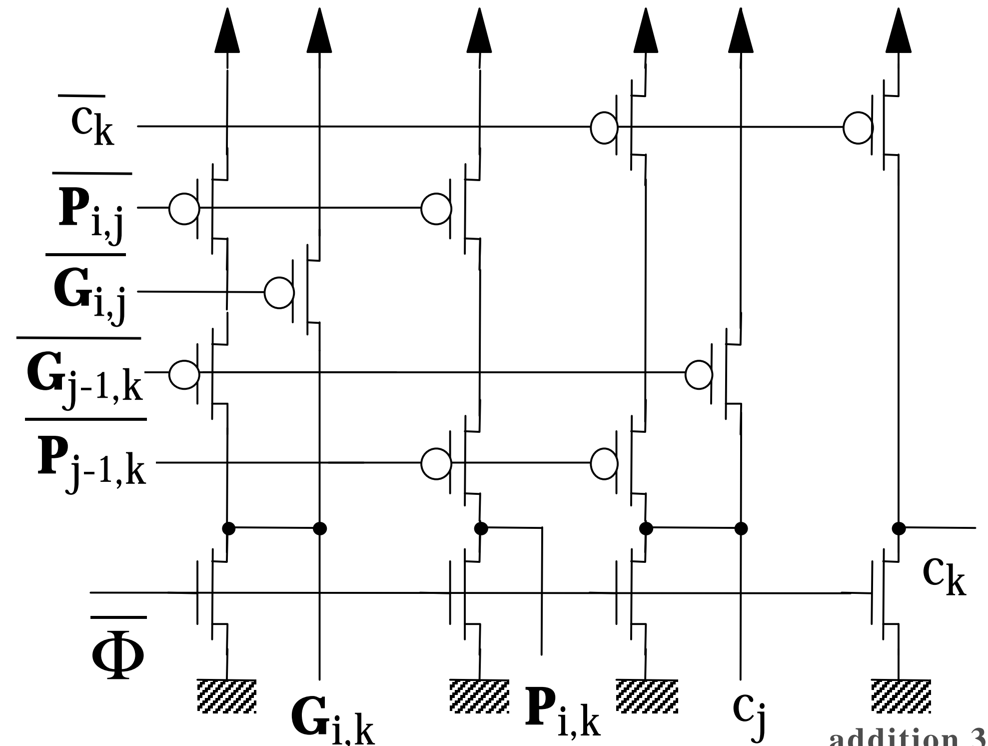


$$G_{i,k} = G_{i,j} \vee P_{i,j} \wedge G_{j-1,k}$$

$$P_{i,k} = P_{i,j} \wedge P_{j-1,k}$$

$$c_j = G_{j-1,k} \vee P_{j-1,k} \wedge c_k$$





Additionneur en temps $\log_2(n)$
Réalisation en Domino

$c_7 = \mathbf{G}_{7,0} \vee \mathbf{P}_{7,0} \wedge c_0$

si $\mathbf{P}_{7,0}$ alors S = $c_0 - 1$

$\overline{\mathbf{P}_{7,0}}$ $\overline{\mathbf{G}_{7,0}}$ c_0

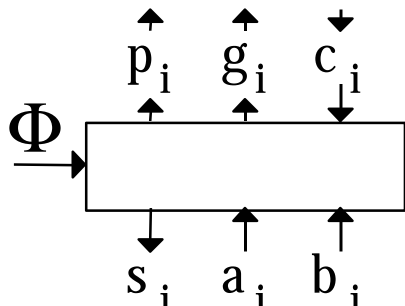
Φ type n

$\overline{\Phi}$ type p type p

Φ type n type n type n type n

Φ

$s_7\ a_7\ b_7$ $s_6\ a_6\ b_6$ $s_5\ a_5\ b_5$ $s_4\ a_4\ b_4$ $s_3\ a_3\ b_3$ $s_2\ a_2\ b_2$ $s_1\ a_1\ b_{71}$ $s_0\ a_0\ b_0$



$g_i = a_i \wedge b_i \wedge \Phi$

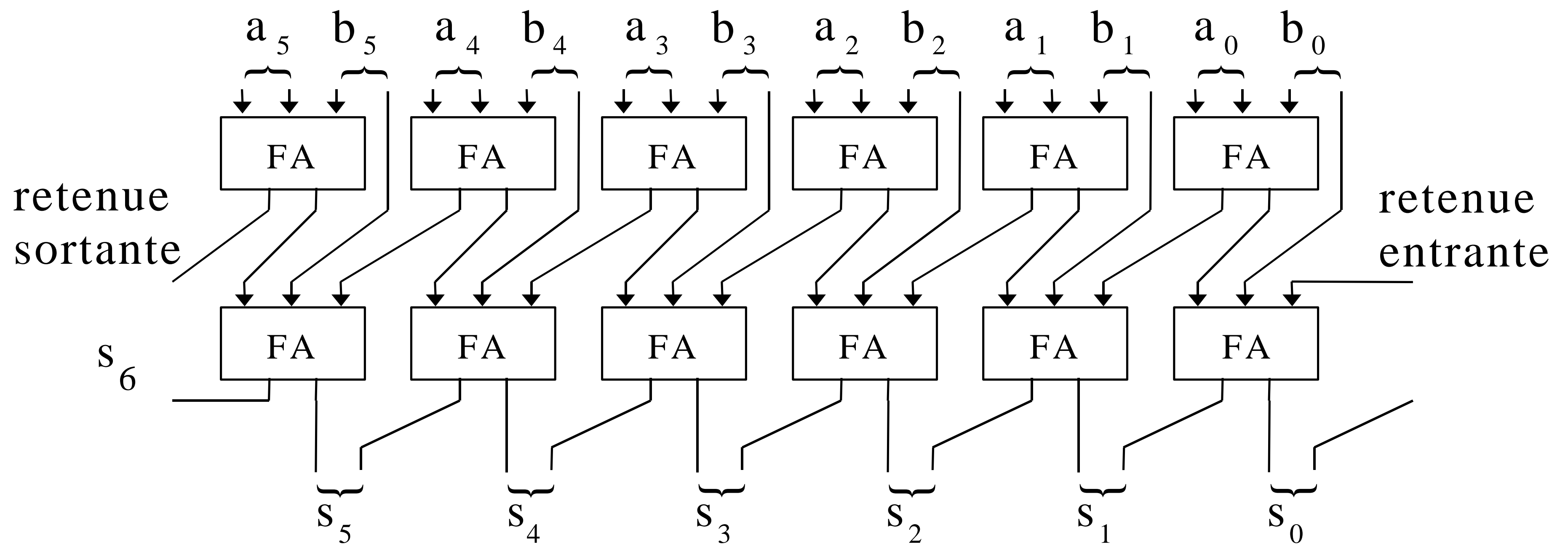
$p_i = a_i \oplus b_i \wedge \Phi$

Conditionné par Φ

$s_i = p_i \oplus c_i$

Addition parallèle sans propagation de la retenue CS

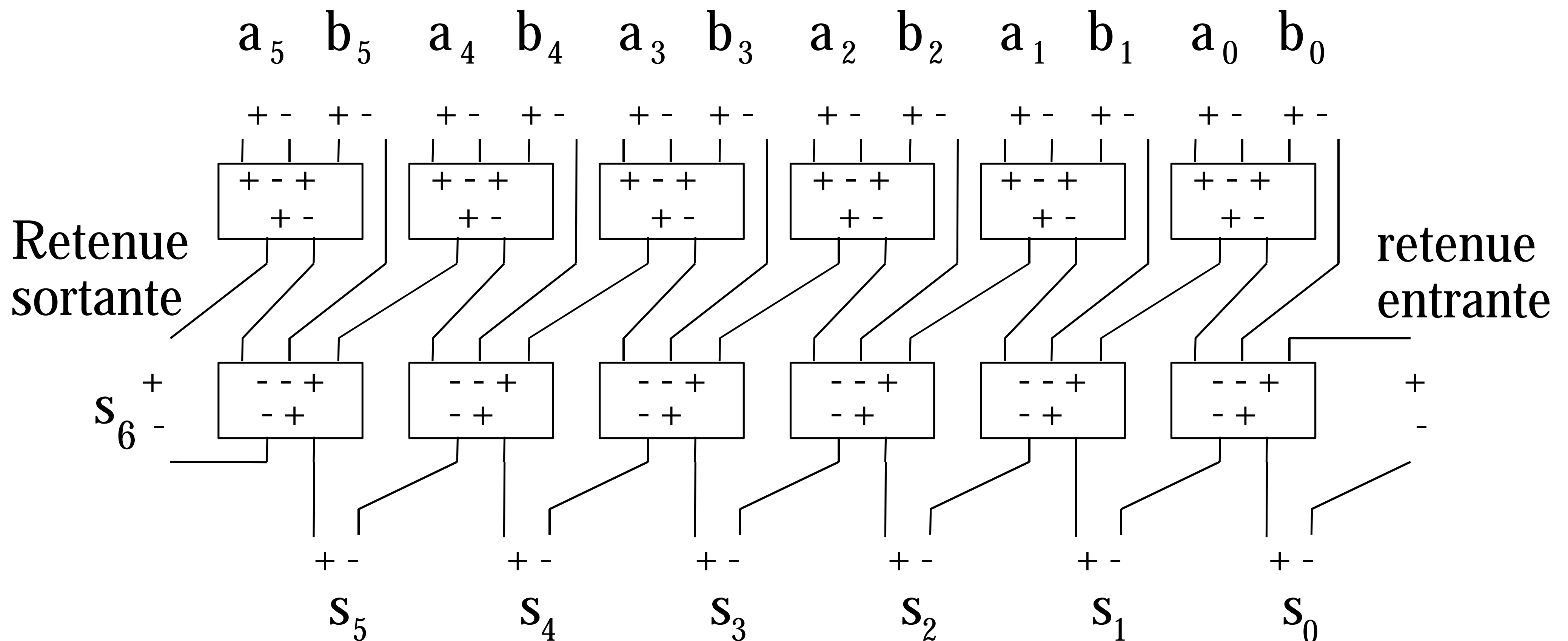
$$A = \sum_{i=0}^{n-1} a_i\, 2^i \quad B = \sum_{i=0}^{n-1} b_i\, 2^i \quad S = \sum_{i=0}^{n} s_i\, 2^i \qquad a_i, b_i, s_i \in \{0,1,2\}$$



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

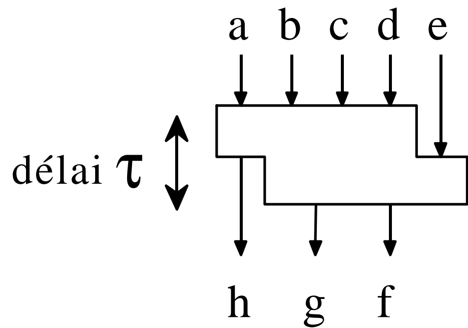
Addition parallèle sans propagation de la retenue BS

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

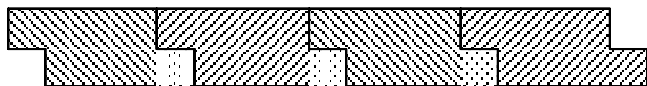


Variantes de cellules de CS

Modèle de délai



h sort avant
que e rentre



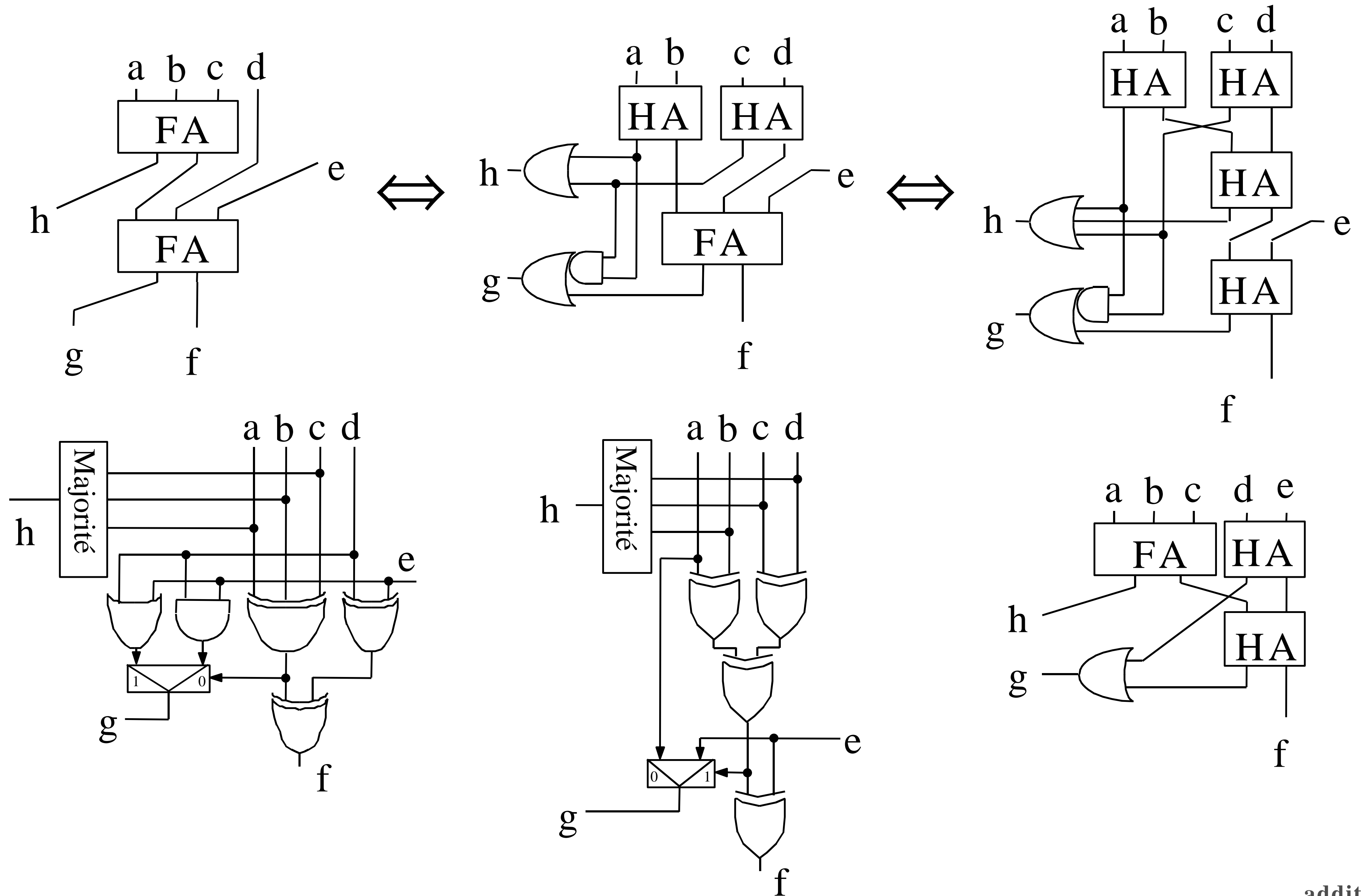
Cette cellule est également appelée “4 donne 2”

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

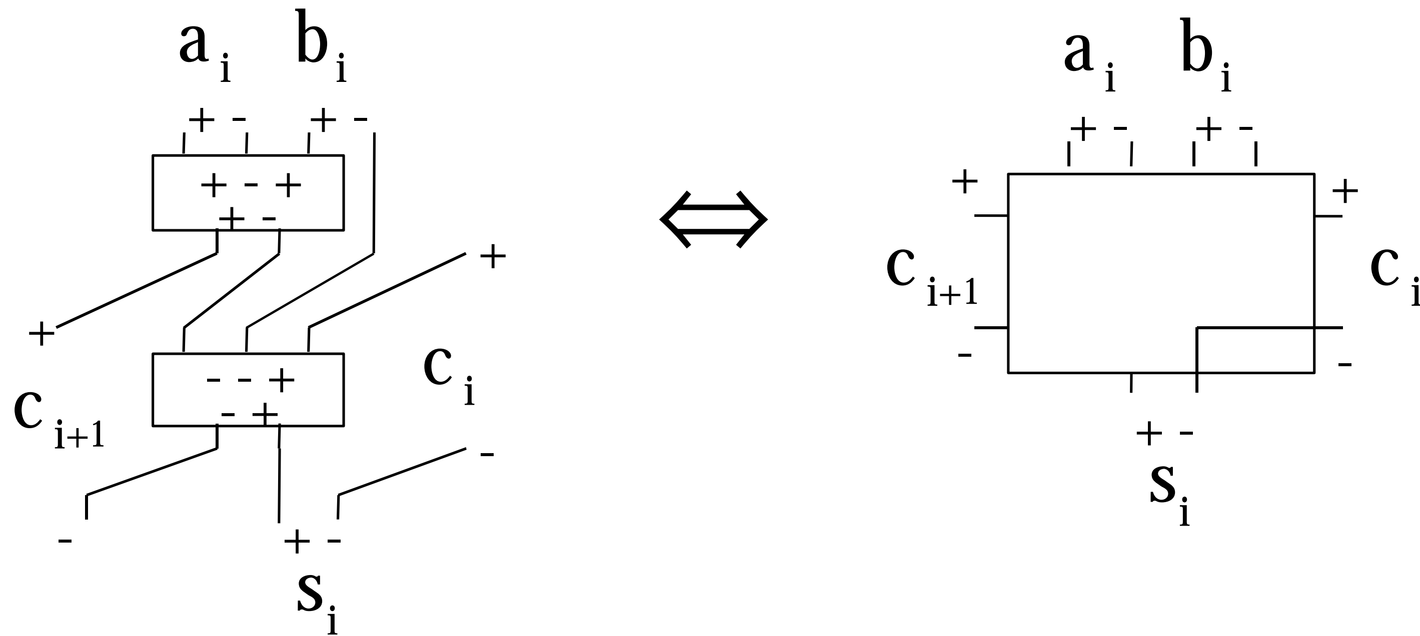
h ne dépend pas de e

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 BS et CS



Optimisation de la cellule d'additionneur "borrow save"



$$x = a_i^+ \oplus a_i^- \oplus b_i^+ \oplus b_i^-$$

$$c_{i+1}^+ = (a_i^+ \wedge \overline{a_i^-}) \vee (b_i^+ \wedge \overline{b_i^-})$$

$$c_{i+1}^- = (\overline{a_i^+} \wedge a_i^- \wedge (b_i^+ \oplus b_i^-)) \vee (\overline{b_i^+} \wedge b_i^- \wedge (a_i^+ \oplus a_i^-)) \vee (c_i^- \wedge x)$$

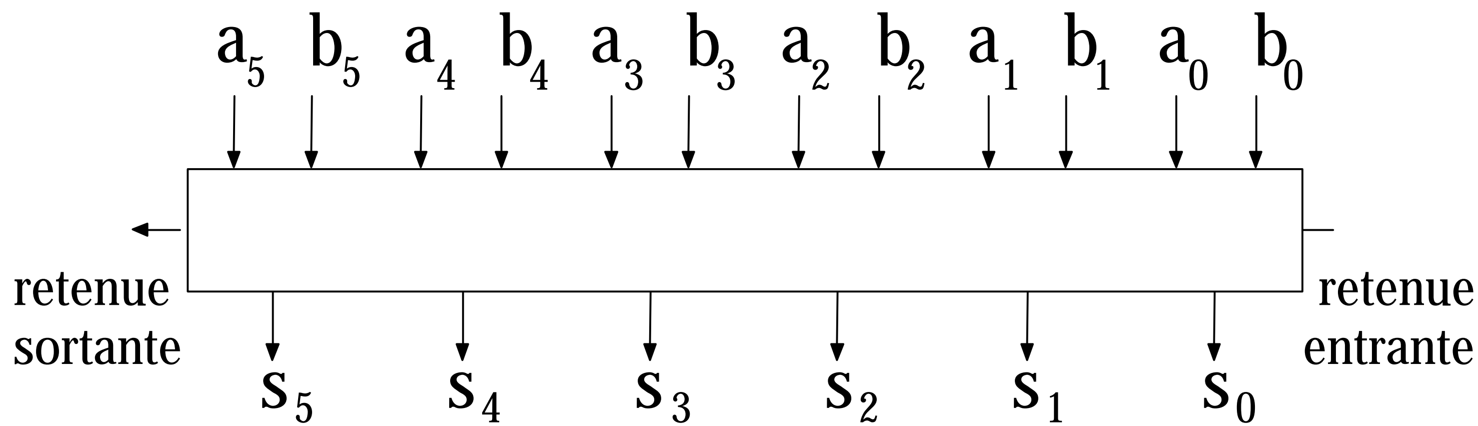
$$s_i^- = c_i^-$$

$$s_i^+ = x \oplus c_i^+$$

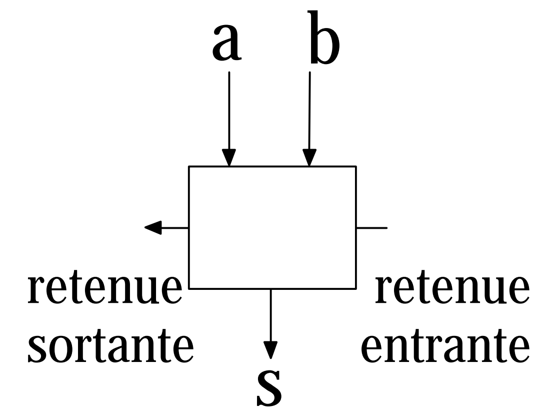
Résumé sur les additionneurs

type	délai	surface	régularité
transmission	n^2	n	très bonne
propagation	n	n	bonne
sélection par retenue	$\sqrt{n}$	n	bonne
retenue bondissante	$\sqrt{n}$	n	moyenne
arbre binaire	$\log n$	$n(\alpha+\beta \log n)$	moyenne
arbre ternaire	$\log n$	$n \log n$	mauvaise
sans propagation	constant	n	très bonne

Test exhaustif des additionneurs

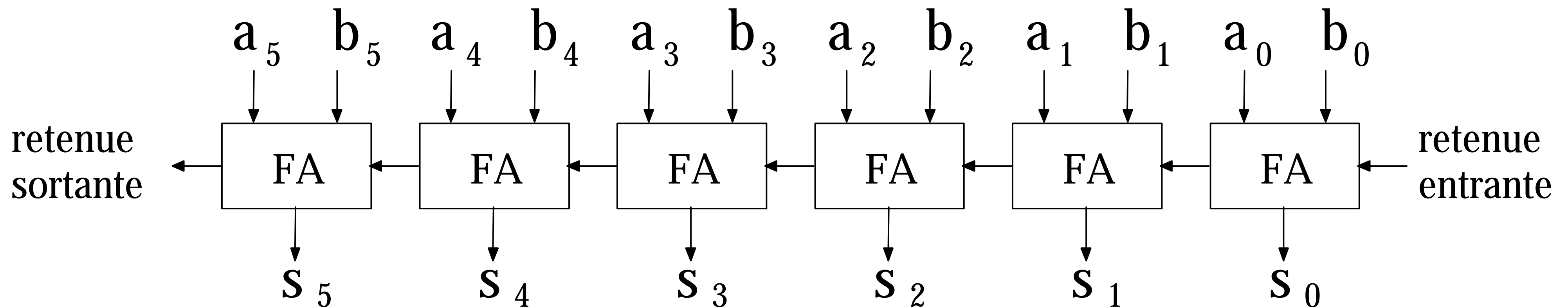


2^{2n+1}



2^3

Test des additionneurs



a_5	b_5	a_4	b_4	a_3	b_3	a_2	b_2	a_1	b_1	a_0	b_0	retenue entrante	
0	0	0	0	0	0	0	0	0	0	0	0	0	test des
0	0	1	1	0	0	1	1	0	0	1	1	0	cellules impaires
1	1	0	0	1	1	0	0	1	1	0	0	1	test des
0	1	0	1	0	1	0	1	0	1	0	1	0	cellules paires
0	1	0	1	0	1	0	1	0	1	0	1	1	
1	0	1	0	1	0	1	0	1	0	1	0	0	
1	0	1	0	1	0	1	0	1	0	1	0	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	