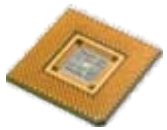
Master ACSI

Architectures pour le Traitement Numérique

Cours Habib MEHREZ

Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie

4, place Jussieu, 75252 Paris cedex 5

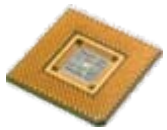


ADDITION

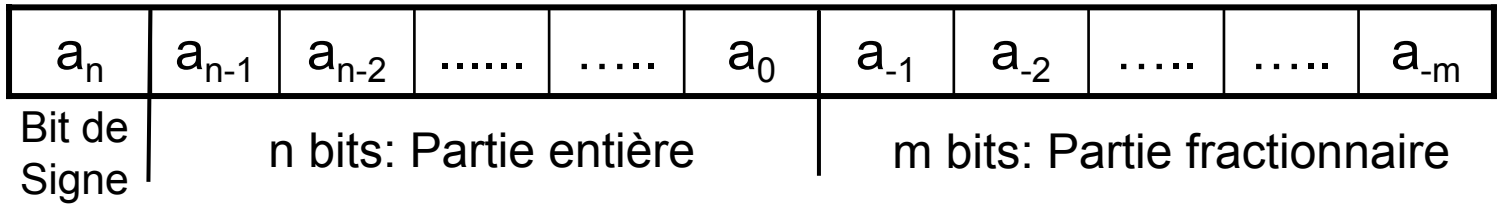
- Opérateurs arithmétiques
- Algorithmes et architectures pour l'addition
- Additif: optimisation des chemins de données arithmétiques par l'utilisation de l'arithmétique redondante

ADDITION

Opérateurs arithmétiques



Codage des nombres



- **Signe et grandeur**

$$A = (-1)^{a_n} \sum_{i=-m}^{i=n-1} a_i 2^i$$

- **Complément à 1**

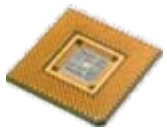
$$A = -a_n \sum_{i=-m}^{i=n-1} (1-a_i) 2^i + (1-a_n) \sum_{i=-m}^{i=n-1} a_i 2^i$$

- **Complément à 2**

$$A = -a_n 2^n + \sum_{i=-m}^{i=n-1} a_i 2^i$$

Codage des nombres (ex sur n=3 bits)

Signe et grandeur	Complément à 1	Complément à 2
+3=011 +2=010 +1=001 +0=000 -0=100 -1=101 -2=110 -3=111	+3=011 +2=010 +1=001 +0=000 -0=111 -1=110 -2=101 -3=100	+3=011 +2=010 +1=001 +0=000 -1=111 -2=110 -3=101 -4=100
$-2^{n-1}+1<=A<=2^{n-1}-1$	$-2^{n-1}+1<=A<=2^{n-1}-1$	$-2^{n-1}<=A<=2^{n-1}-1$



Codage des nombres

Passage d'une représentation à une autre

- **Signe et grandeur**

$A = 0\ a_{n-1}\ a_{n-2} \cdots\cdots\ a_1\ a_0\ a_{-1}\ a_{-2} \cdots\cdots a_{-m}$

$-A = 1\ a_{n-1}\ a_{n-2} \cdots\cdots\ a_1\ a_0\ a_{-1}\ a_{-2} \cdots\cdots a_{-m}$

- **Complément à 1**

$A_1 = 0\ a_{n-1} \cdots\cdots\ a_0 \cdots\cdots a_{-m}$

$-A_1 = 1\ (1-a_{n-1}) \cdots\cdots (1-a_0) \cdots\cdots (1-a_{-m})$

- **Complément à 2**

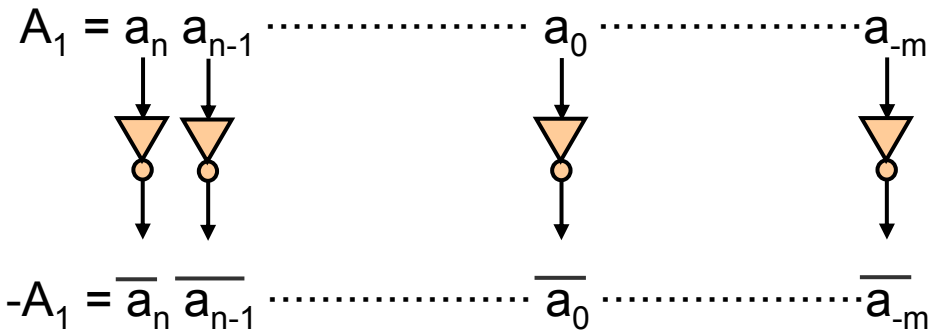
$A_2 = 0\ a_{n-1} \cdots\cdots\ a_0 \cdots\cdots a_{-m}$

$-A_2 = 1\ (1-a_{n-1}) \cdots\cdots (1-a_0) \cdots\cdots (1-a_{-m}) + 2^{-m}$

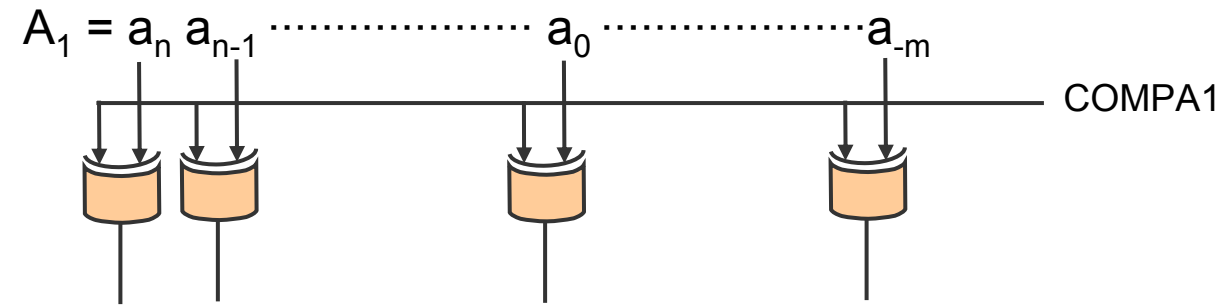
➔ **$-A_2 = -A_1 + 2^{-m}$**

Opérateur de complémentation à 1

- **Opérateur simple**

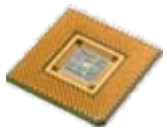


- **Complémenteur à 1 avec commande**



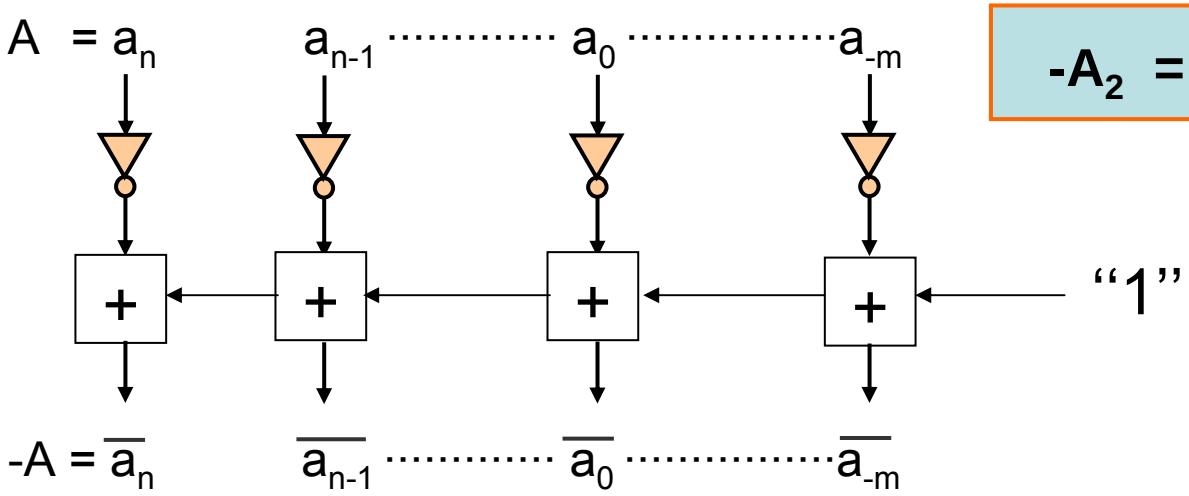
SORTIE= $-A_1 = \overline{a_n}\ \overline{a_{n-1}} \cdots \overline{a_0} \cdots \overline{a_{-m}}$ ···si COMPA1=1

SORTIE= $A_1 = a_n\ a_{n-1} \cdots a_0 \cdots a_{-m}$ ···si COMPA1=0



Opérateurs de complémentation à 2

Premier algorithme



$$-A_2 = -A_1 + 2^{-m}$$

Deuxième algorithme

Exemple:

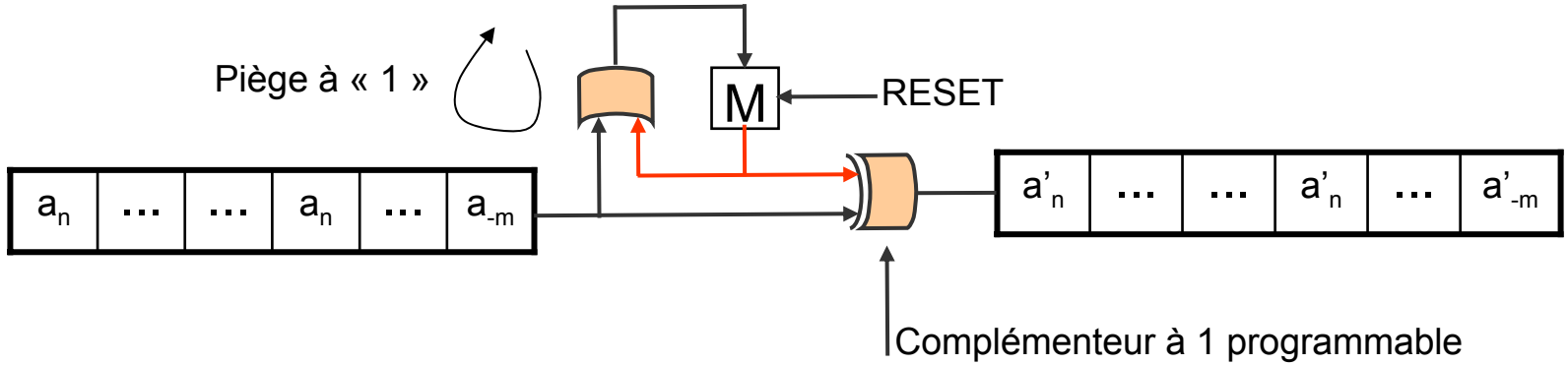
A = 4 : 0 1 0 0

-A = -4 : 1 1 0 0

En commençant par les LSB, on garde tous les bits jusqu'au premier « 1 » rencontré inclus, puis on complémente tous les bits suivants.

Opérateurs de complémentation à 2

Complémenteur à 2 Série



- M est initialisé à « 0 »
- Tous les a_i=0 passent, le premier « 1 » = 1
- Le premier « 1 » sera piégé => complémenter tous les a_i suivants

$M_{i+1} \leftarrow M_i$ OR a_i

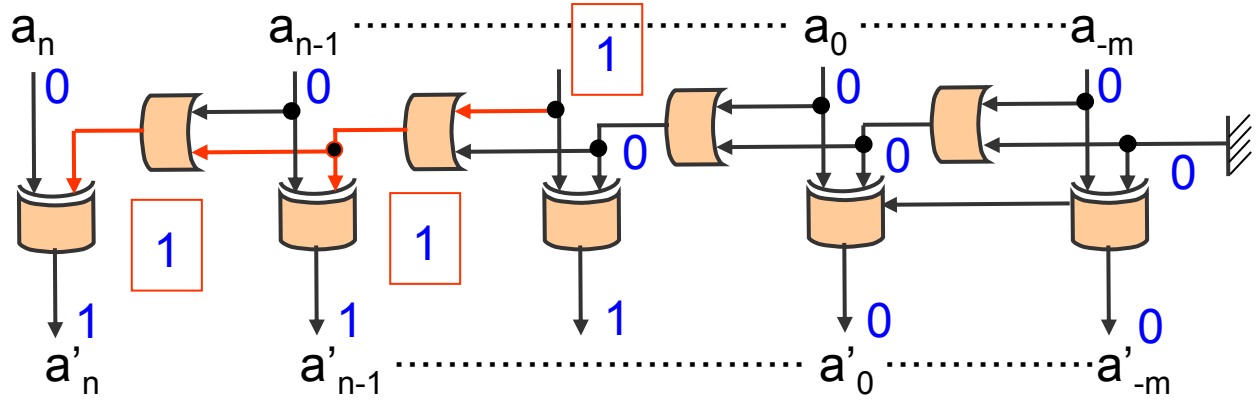
Si Mi ←1 alors $M_{i+1} \leftarrow 1$ qlq a_i

Opérateurs de complémentation à 2

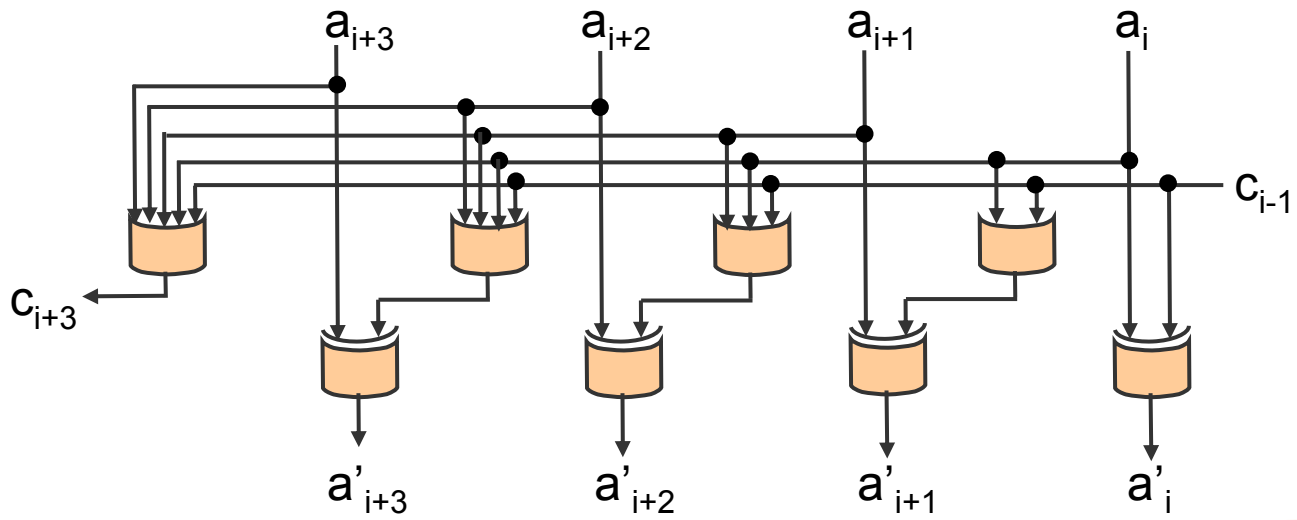
Complémenteur à 2 Parallèle à propagation

Surface ~ O(N)

Temps ~ O(N)



Complémenteur à 2 Rapide (anticipation des commandes)

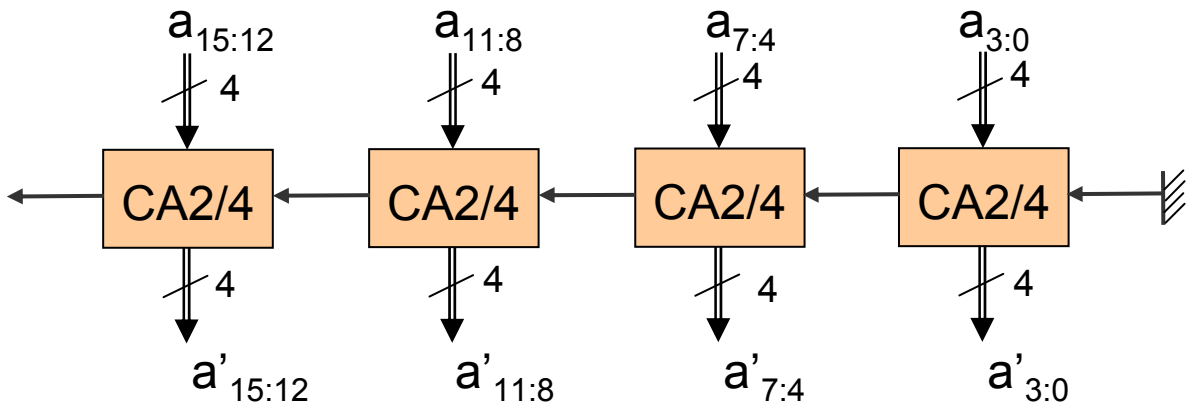


Opérateurs de complémentation à 2

- **Complémentaire à 2 sur N bits**

Surface ~ O(N)

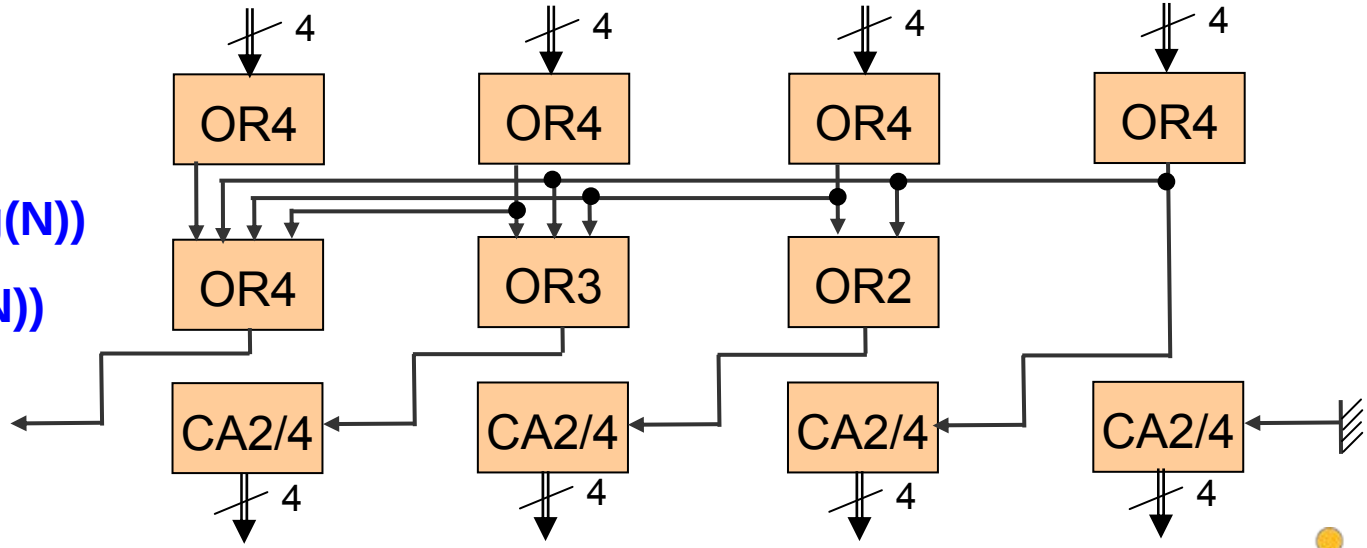
Temps ~ O(N)



- **Complémentaire à 2 version plus rapide**

Surface ~ O(N.Log(N))

Temps ~ O(Log(N))



Opérateurs arithmétiques

Additionneur en complément à 2

Les 2 opérandes doivent être codés sur le même nombre de bits, sinon, on fait une extension de signe.

Extension de signe

Exemple:

A = 0011 -> 3

B = 1100 -> -4

A+B	0	0	1	1
	1	1	0	0
-1 =	1	1	1	1

A = 00011 -> 5bits

B = 1100 -> 4bits

A+B	0	0	0	1	1
	?	1	1	0	0
+15 =	0	1	1	1	1

Résultat Faux !!

	0	0	0	1	1
	1	1	1	0	0
-1 =	1	1	1	1	1

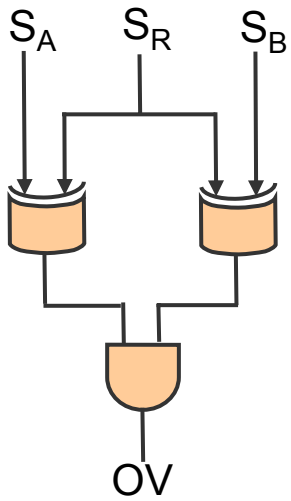
Résultat Bon !!

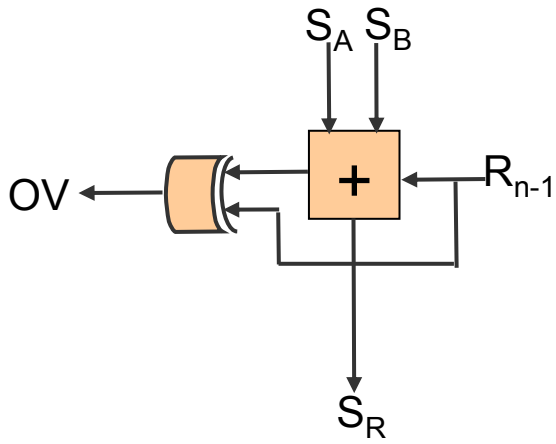
Opérateurs arithmétiques

- **Problème de dépassement de capacité (overflow)**

A										
+	1	0	0	1	-7	0	1	0	0	4
B	1	0	0	0	-8	0	1	0	1	5
=S	0	0	0	1	+1	1	0	0	1	- 7

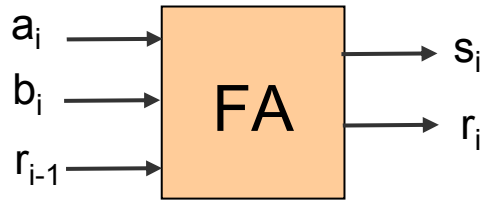
$$OV = S_A S_B S'_R + S'_A S'_B S_R = R_{n-1} \oplus R_n$$





Opérateurs d'addition

Additionneur complet



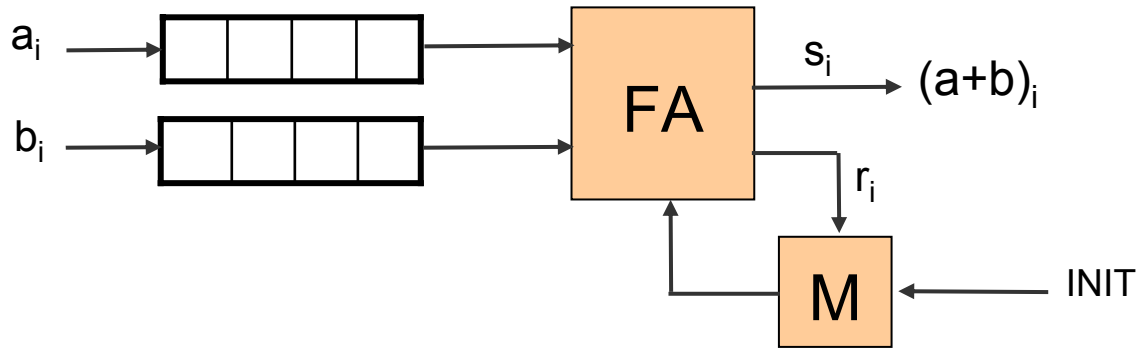
$$a_i + b_i + r_{i-1} = s_i + 2r_i$$

$$s_i = (a_i \oplus b_i) \oplus r_{i-1}$$

$$r_i = (a_i \oplus b_i) r_{i-1} + a_i b_i$$

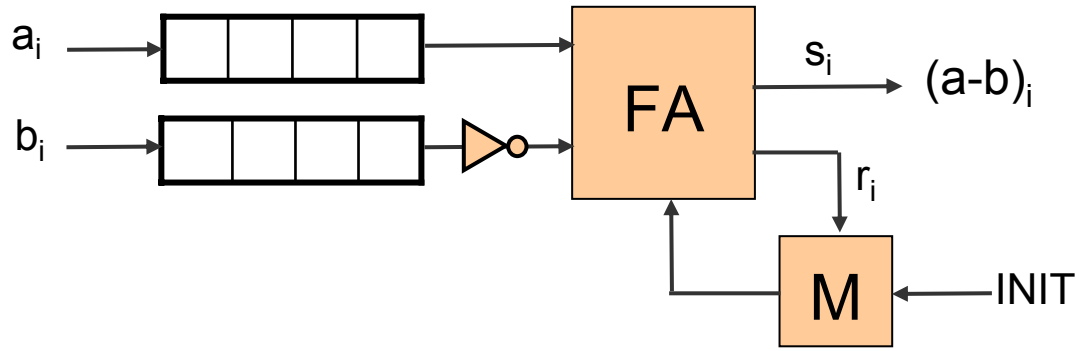
$$r_i = a_i . r_{i-1} + b_i . r_{i-1} + a_i . b_i$$

Additionneur série à retenue sauvegardée (CSA)

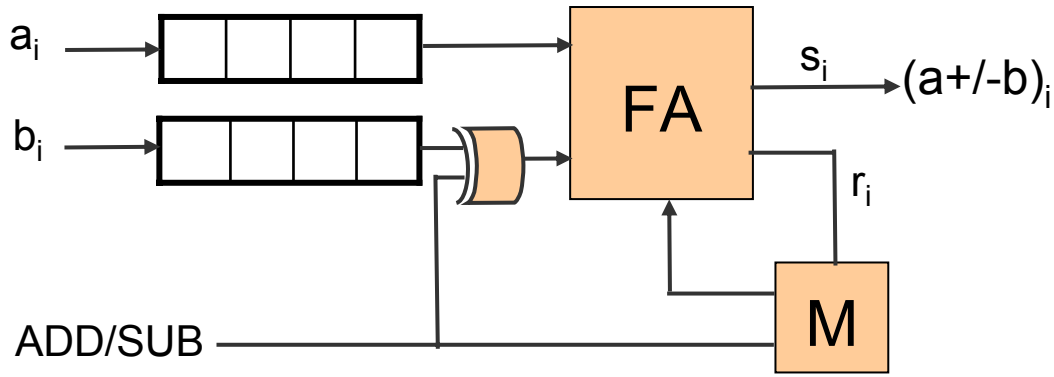


Opérateurs d'addition

- **Subtracteur série**



- **Additionneur/Subtracteur série**

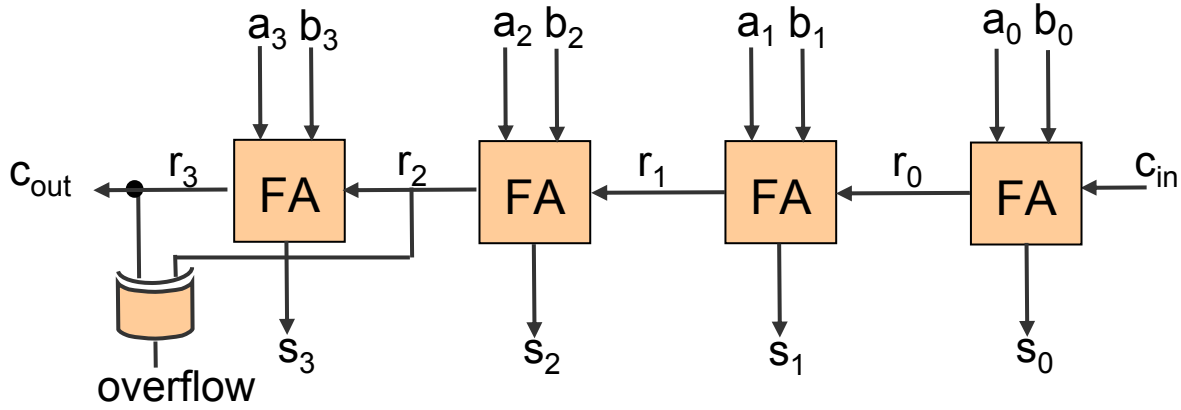


ADDITION

Algorithmes et architectures

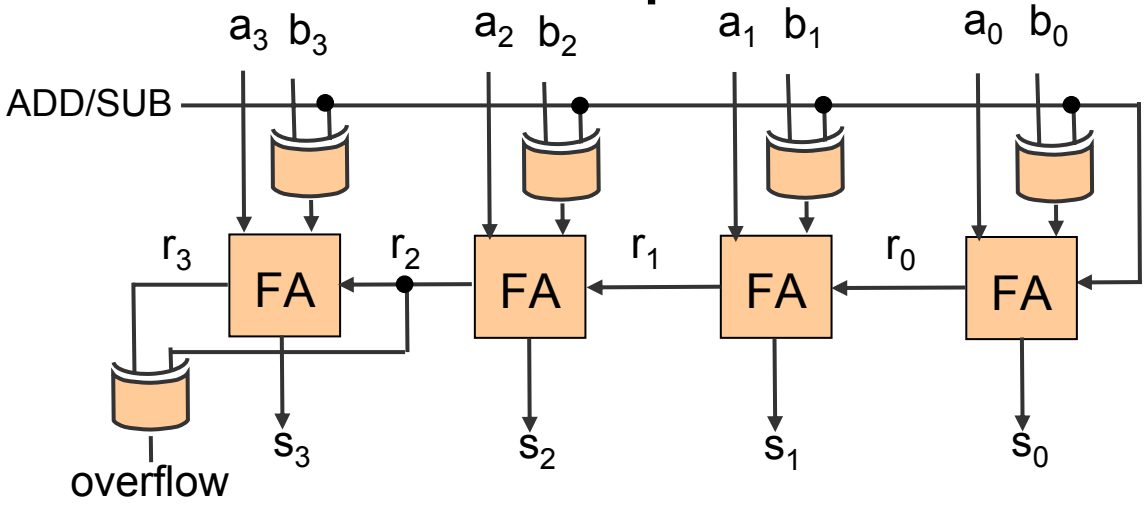
Addition à propagation de retenue

- **Additionneur parallèle à retenues propagées « Ripple carry »**



Temps	O(N)
Surface	O(N)

- **Additionneur/Subtracteur parallèle**



Addition à reports anticipes

$$S_i = (A_i \oplus B_i) \oplus R_{i-1} \qquad R_i = (A_i \oplus B_i)R_{i-1} + A_iB_i$$

On pose: $P_i = A_i \oplus B_i$ $\qquad G_i = A_i \cdot B_i$

D'où: $\qquad S_i = P_i \oplus R_{i-1}$ $\qquad R_i = G_i + P_iR_{i-1}$

- On va exprimer les R_i en fonction des P_i et$_i$ G_i

$S_{i+1} = P_i \oplus R_i$

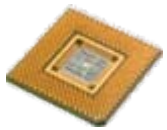
$R_{i+1} = G_{i+1} + P_{i+1}R_i = G_{i+1} + P_{i+1}(G_i + P_iR_{i-1}) = G_{i+1} + P_{i+1}G_i + P_{i+1}P_iR_{i-1}$

$S_{i+2} = P_{i+1} \oplus R_{i+1}$

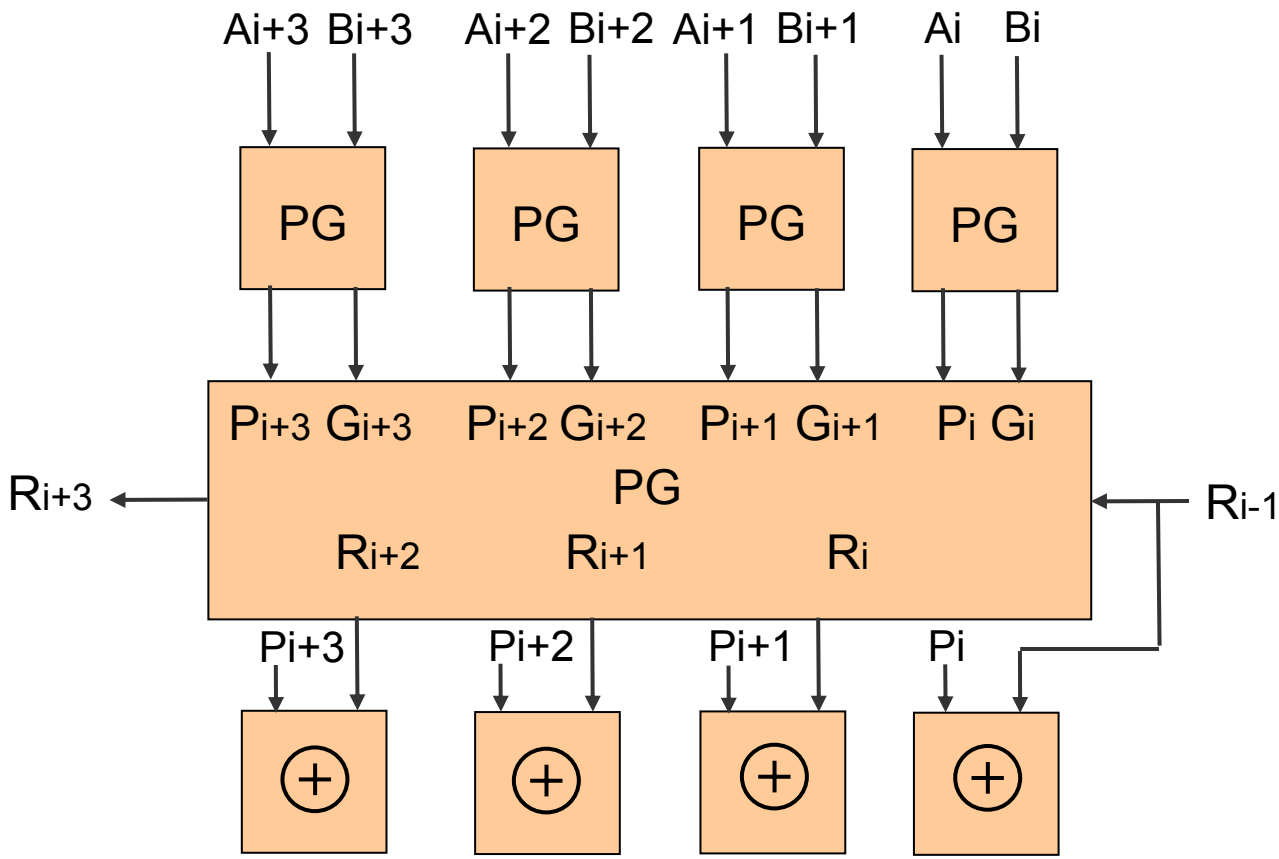
$R_{i+2} = G_{i+2} + P_{i+2}R_{i+1} = G_{i+2} + P_{i+2}(G_{i+1} + P_{i+1}R_i) = G_{i+2} + P_{i+2}G_{i+1} + P_{i+2}P_{i+1}G_{i+} + P_{i+2}P_{i+1}R_i\,R_{i-1}$

$S_{i+3} = P_{i+2} \oplus R_{i+2}$

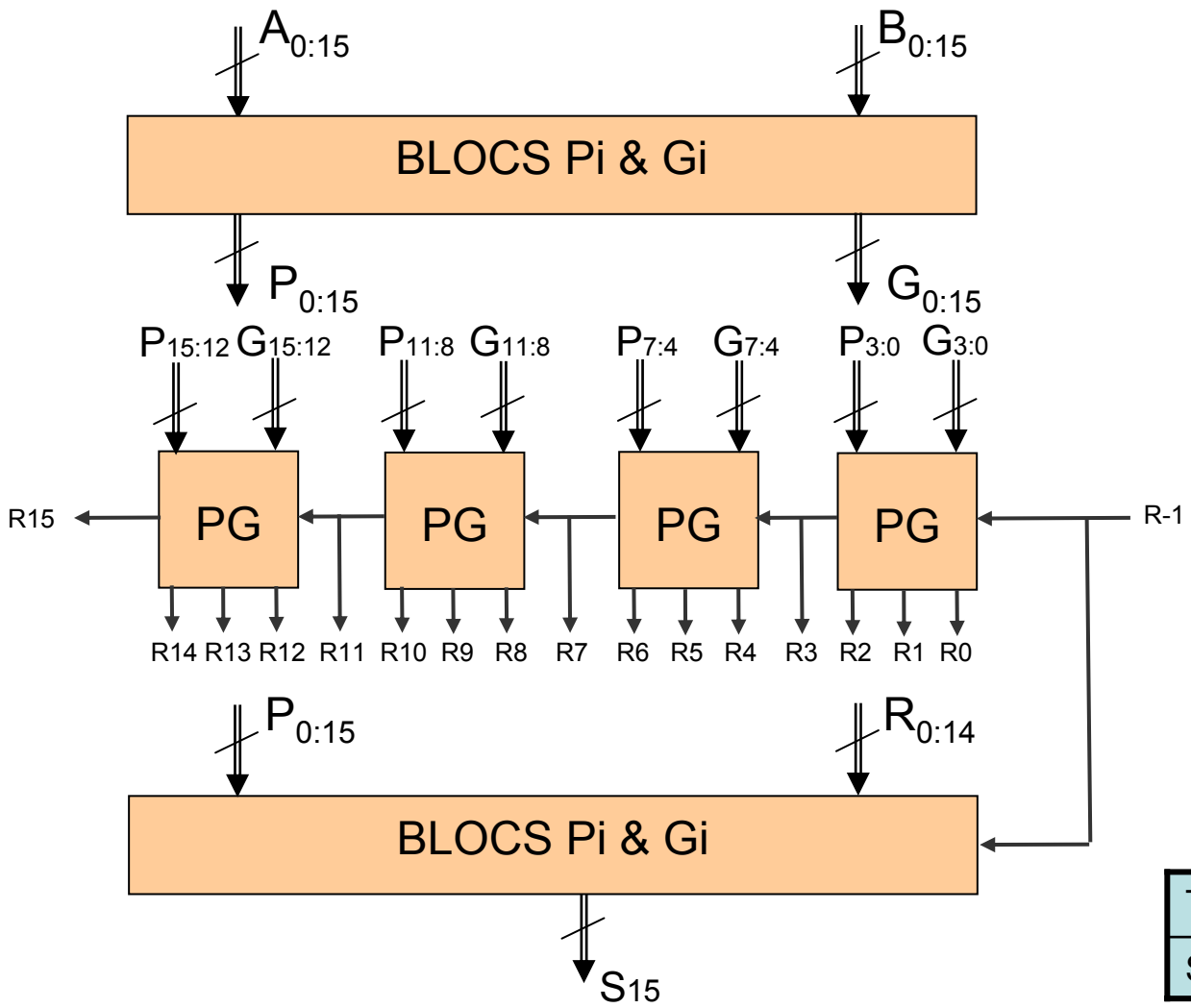
$R_{i+2} = G_{i+3} + P_{i+3}R_{i+2} = G_{i+3} + P_{i+3}G_{i+2} + P_{i+3}P_{i+2}RG_{i+1} + P_{i+3}P_{i+2}P_{i+1}G_i + P_{i+3}P_{i+2}P_{i+1}P_iR_{i-1}$



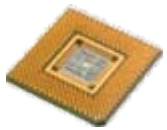
Addition à reports anticipes



Addition à reports anticipes sur N bits séries



Temps	O(n)
Surface	O(n)



Addition à reports anticipes

$S_{i+3} = P_{i+2} \oplus R_{i+2}$

$R_{i+3} = G_{i+3} + P_{i+3}R_{i+2} = G_{i+3} + P_{i+3}G_{i+2} + P_{i+3}P_{i+2}G_{i+1} + P_{i+3}P_{i+2}P_{i+1}G_i + P_{i+3}P_{i+2}P_{i+1}P_iR_{i-1}$

$R_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0R_{-1}$

$R_7 = G_7 + P_7G_6 + P_7P_6G_5 + P_7P_6P_5G_4 + P_7P_6P_5P_4R_3$

$R_{11} = G_{11} + P_{11}G_{10} + P_{11}P_{10}G_9 + P_{11}P_{10}P_9G_8 + P_{11}P_{10}P_9P_8R_7$

$R_{15} = G_{15} + P_{15}G_{14} + P_{15}P_{14}G_{13} + P_{15}P_{14}P_{13}G_{12} + P_{15}P_{14}P_{13}P_{12}R_{11}$

$R_3 = G_{3,0} + P_{3,0}R_{-1}$

$R_7 = G_{7,4} + P_{7,4}R_3$

$R_{11} = G_{11,8} + P_{11,8}R_7$

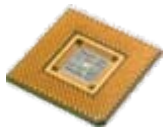
$R_{15} = G_{15,12} + P_{15,12}R_{11}$

$R_3 = G_{3,0} + P_{3,0}R_{-1}$

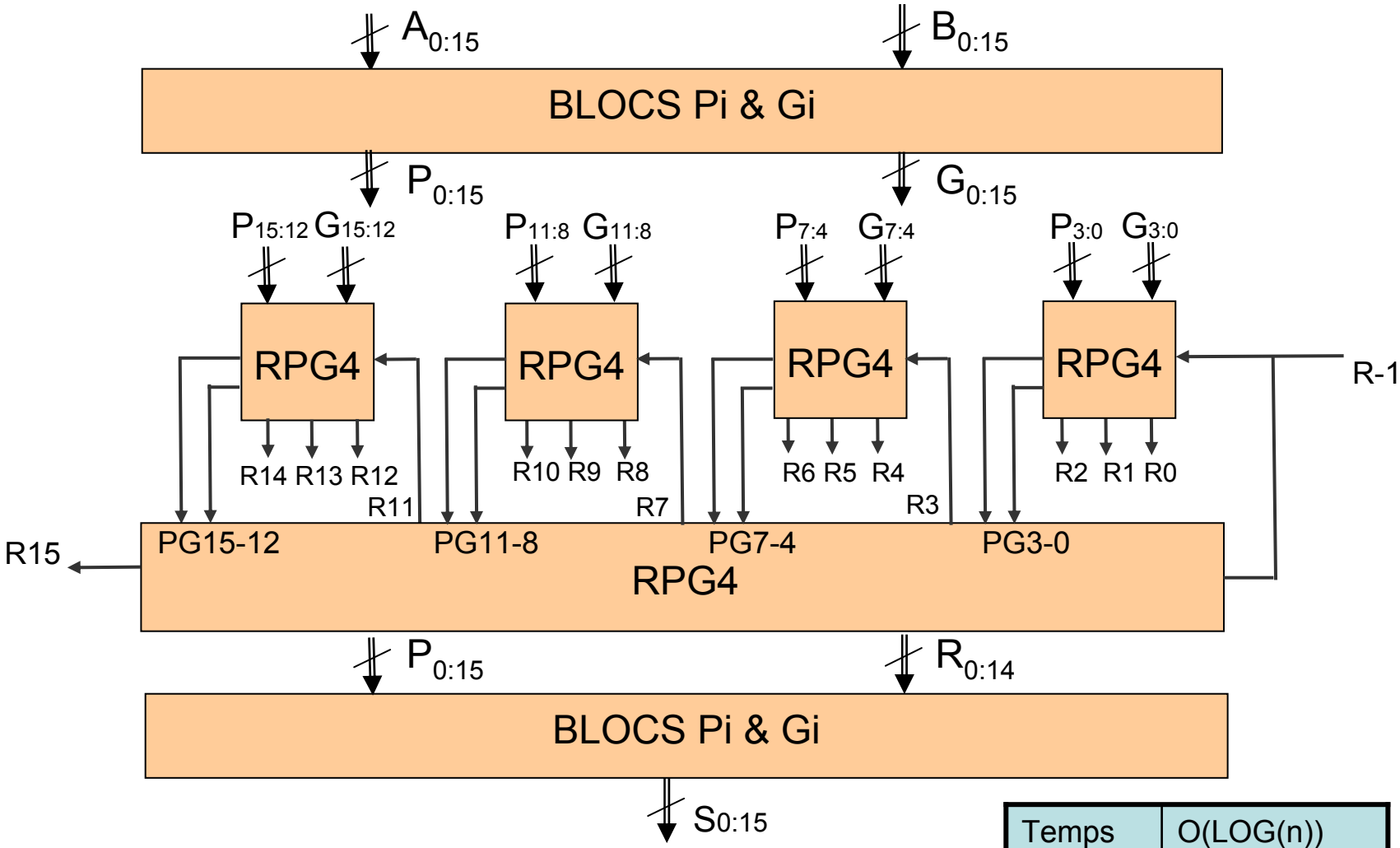
$R_7 = G_{7,4} + P_{7,4}(G_{3,0} + P_{3,0}R_{-1})$

$R_{11} = G_{11,8} + P_{11,8}(G_{7,4} + P_{7,4}(G_{3,0} + P_{3,0}R_{-1}))$

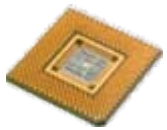
$R_{15} = G_{15,12} + P_{15,12}(G_{11,8} + P_{11,8}(G_{7,4} + P_{7,4}(G_{3,0} + P_{3,0}R_{-1})))$



Addition à reports anticipes sur N bits parallèles

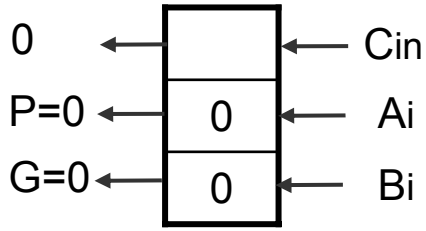


Temps	O(LOG(n))
Surface	O(n)

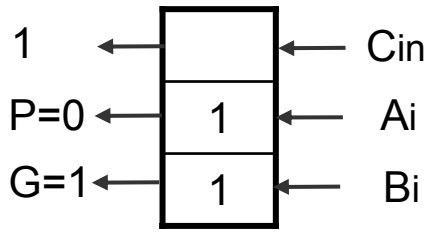


Accéleration de la retenue

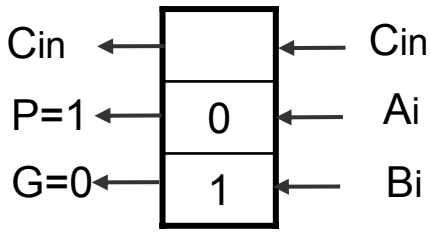
- **Absorption**

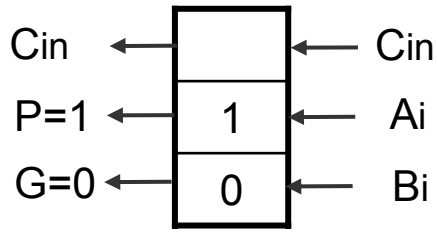


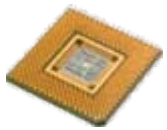
- **Generation**



- **Propagation**

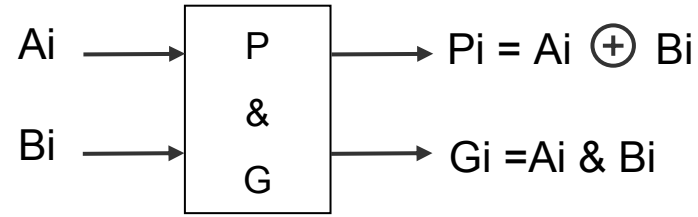




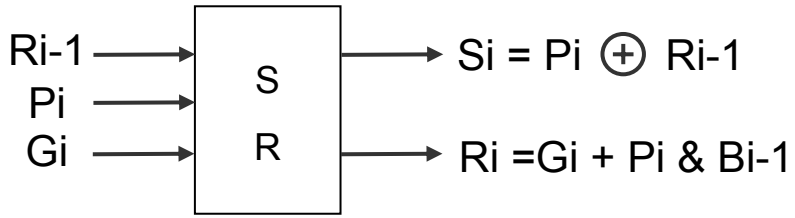


Accéleration de la retenue

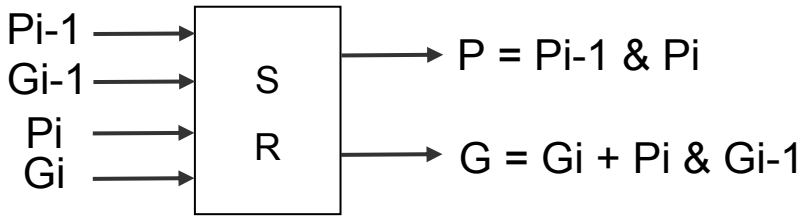
- Calcul de P&G en fonction de deux bits de même poids

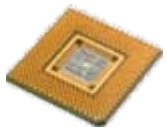


- Calcul de la somme et de la retenue à partir de PiGi

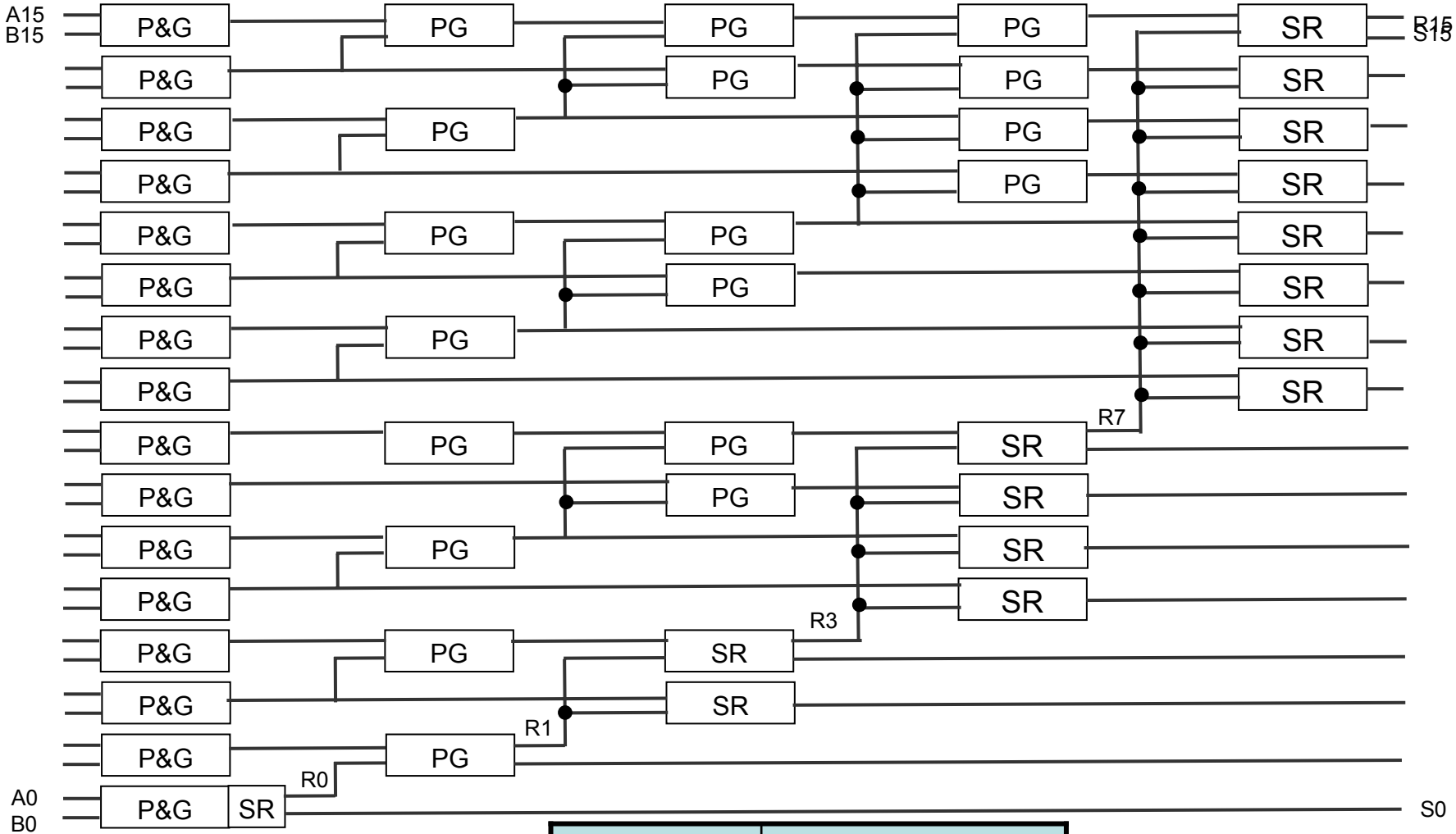


- **Calcul de P&G en fonction de Pi & Gi**





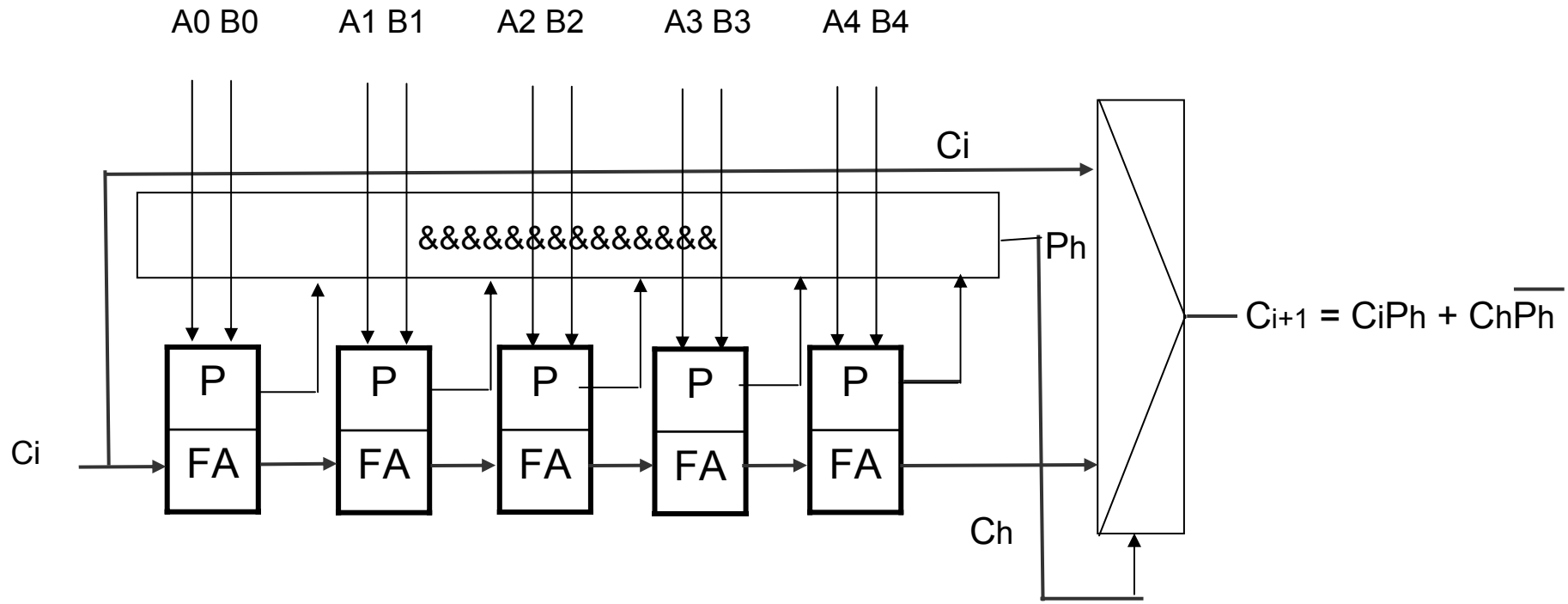
Recurrence SOLVER (Sklansky)



Temps	O(LOG(n))
Surface	O(n LOG(n))

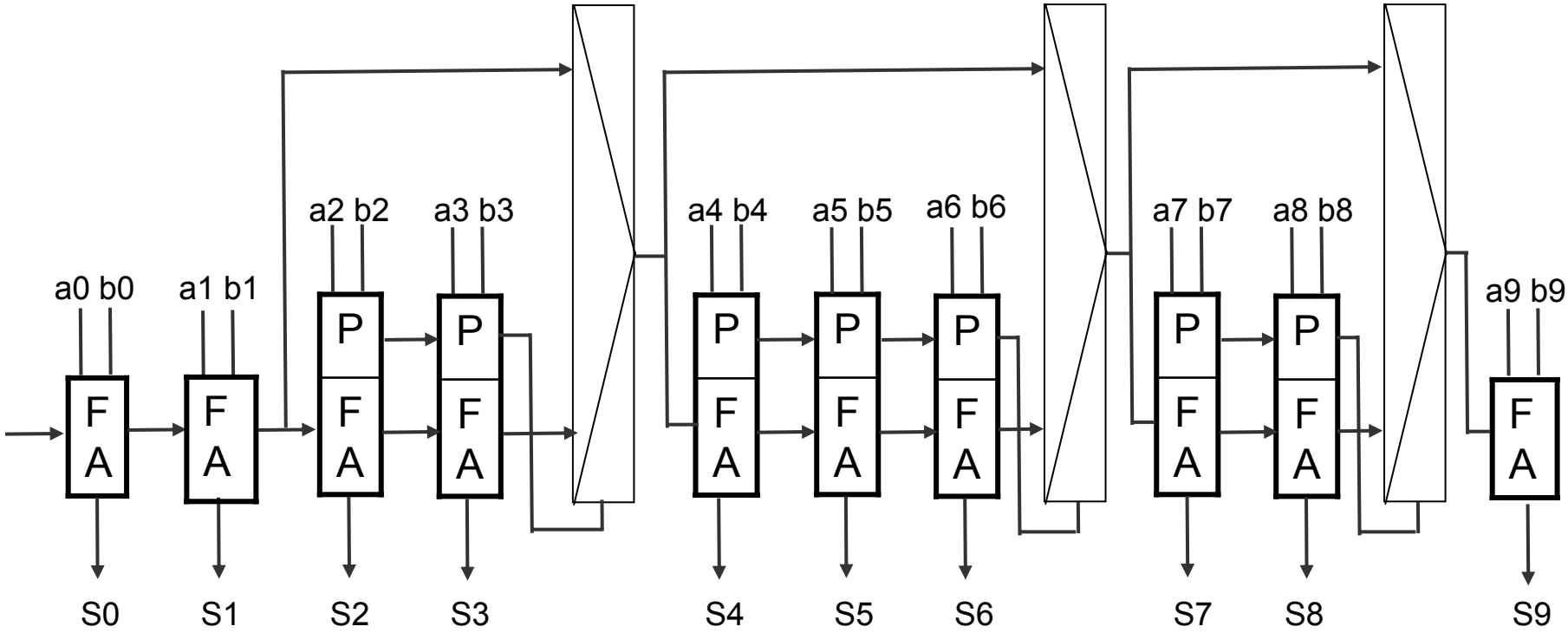
Additionneur à saut de retenue

- Principe de l'additionneur à saut de retenue



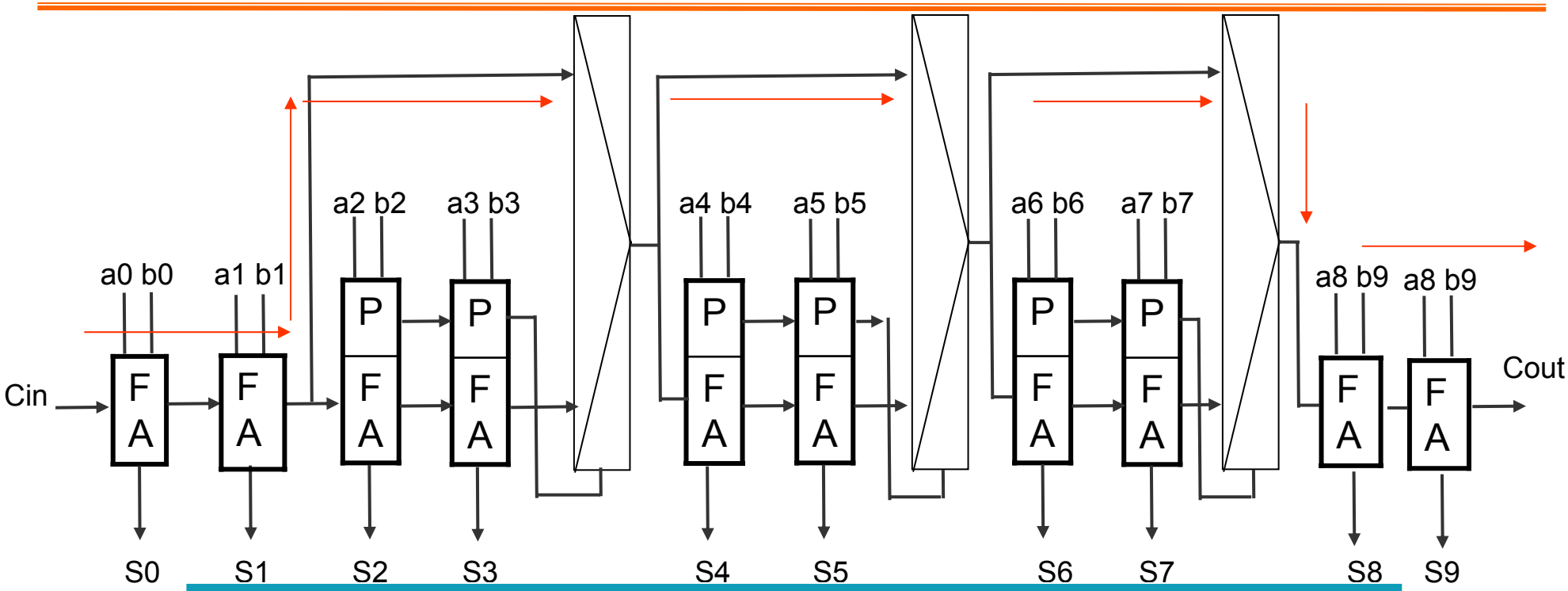
Ph: Propagation du bloc d'additionneur

Additionneur à saut de retenue



Temps	O(√n)
Surface	O(n)

Additionneur à saut de retenue



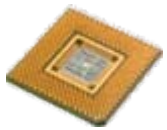
On considère p blocs sur k bits, soit N= p*k

$Tp=k.tp_{FA} + (p-2)tp_{Mux}+ k\ tp_{FA}= 2.k.tp_{FA} + (p-2)tp_{Mux}$ Or p=N/k

➲ Tp= 2.k.tpFA + (N/k -2)tpMux ➲ $Tp'(k)= 2.tpFA - N/k^2. tpMux$

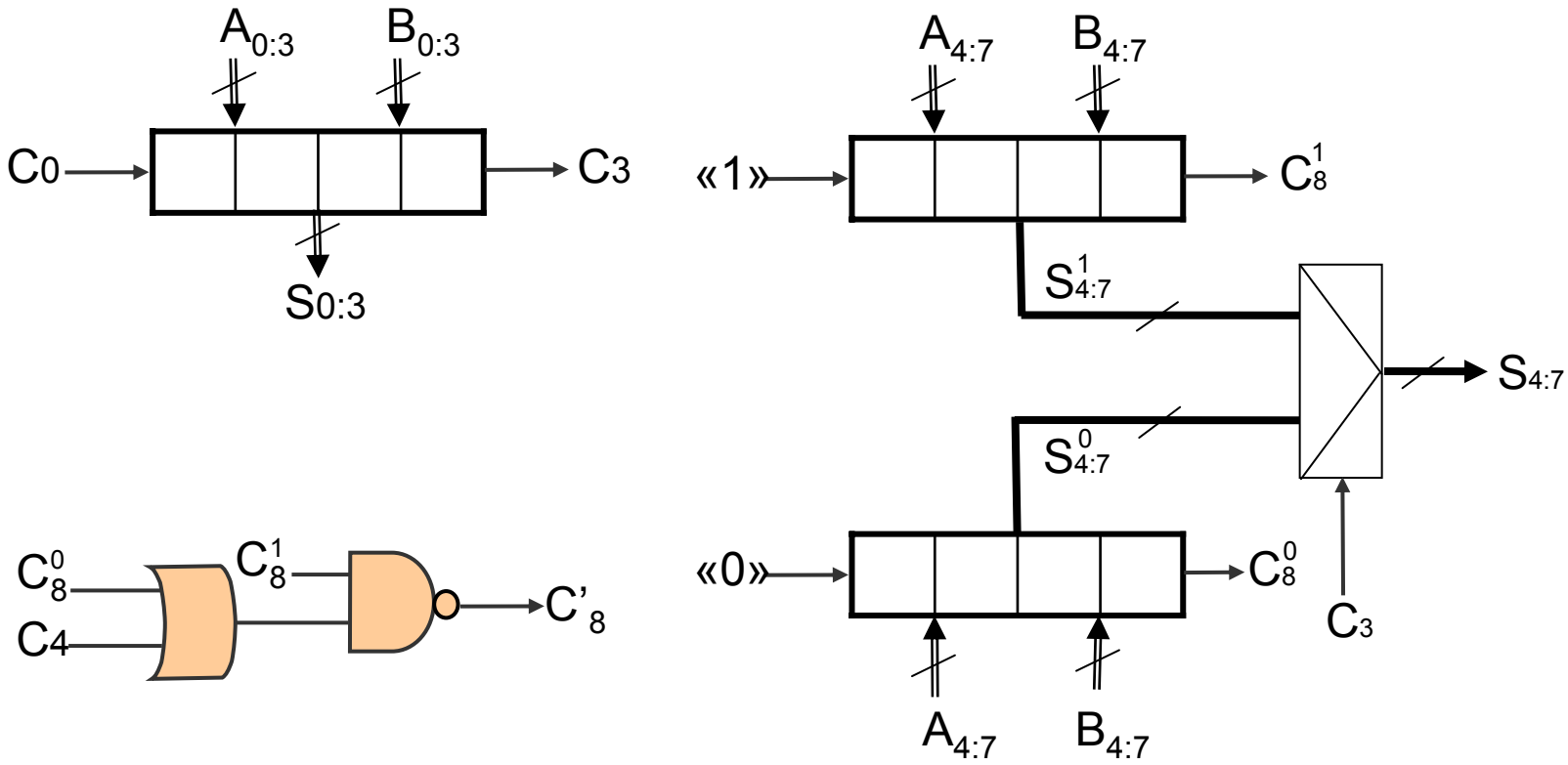
Tp'(k)= 0 ➲ k0= sqrt(N. . tpMux / 2.tpFA)

➲ Tpmin= 2.k0.tpFA + (N/k0 -2)tpMux



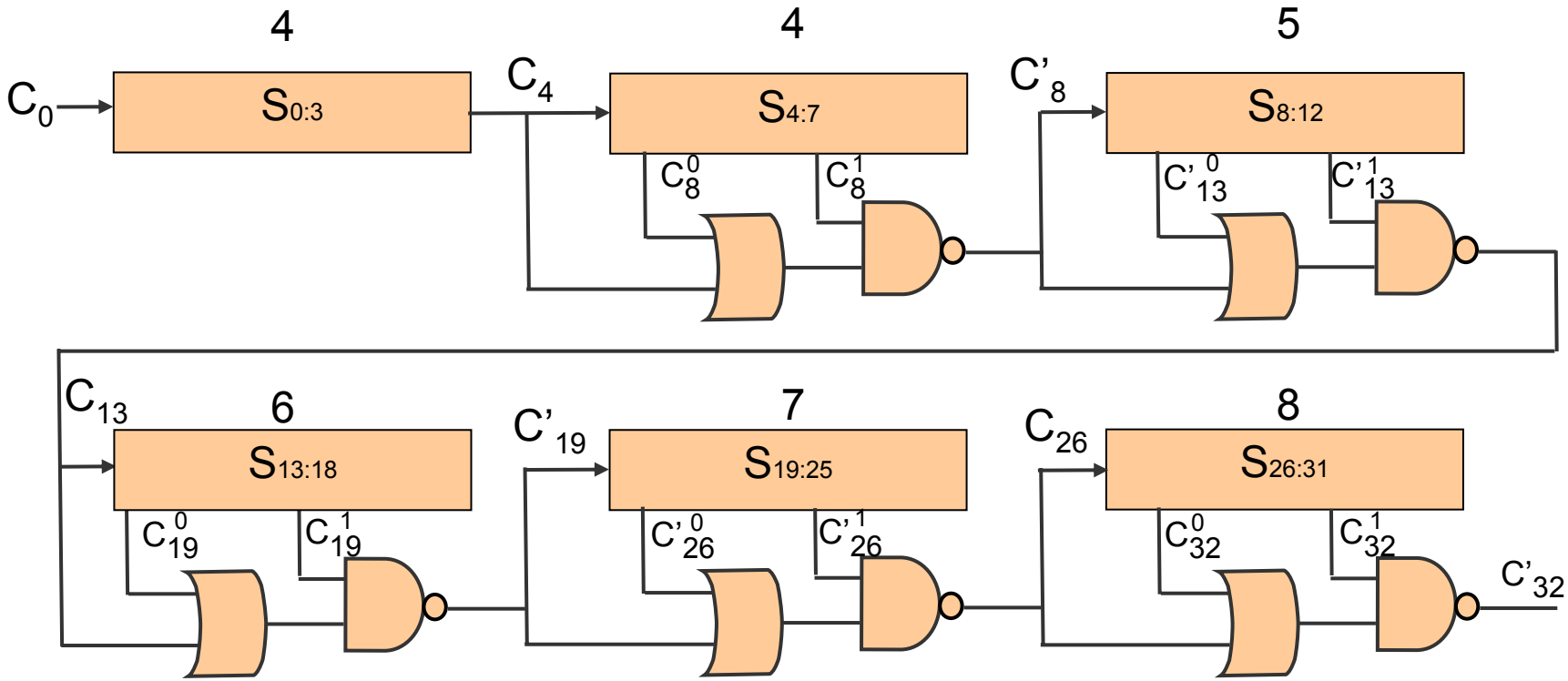
Addition à retenue conditionnelle *(Carry select adder)*

- Principe

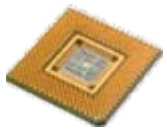


Addition à retenue conditionnelle *(Carry select adder)*

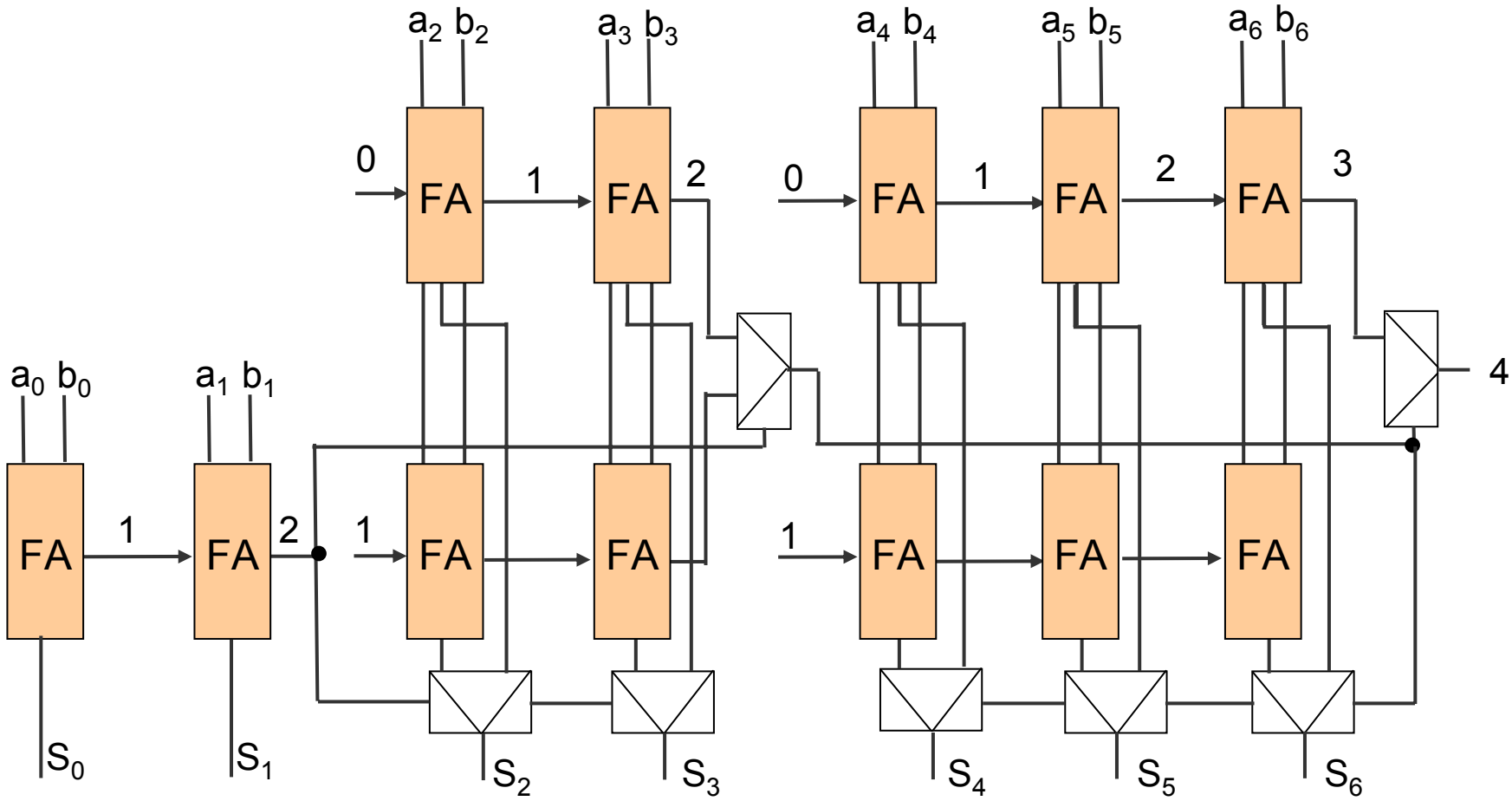
- Additionneur 32 bits



Temps	O(√n)
Surface	O(n)

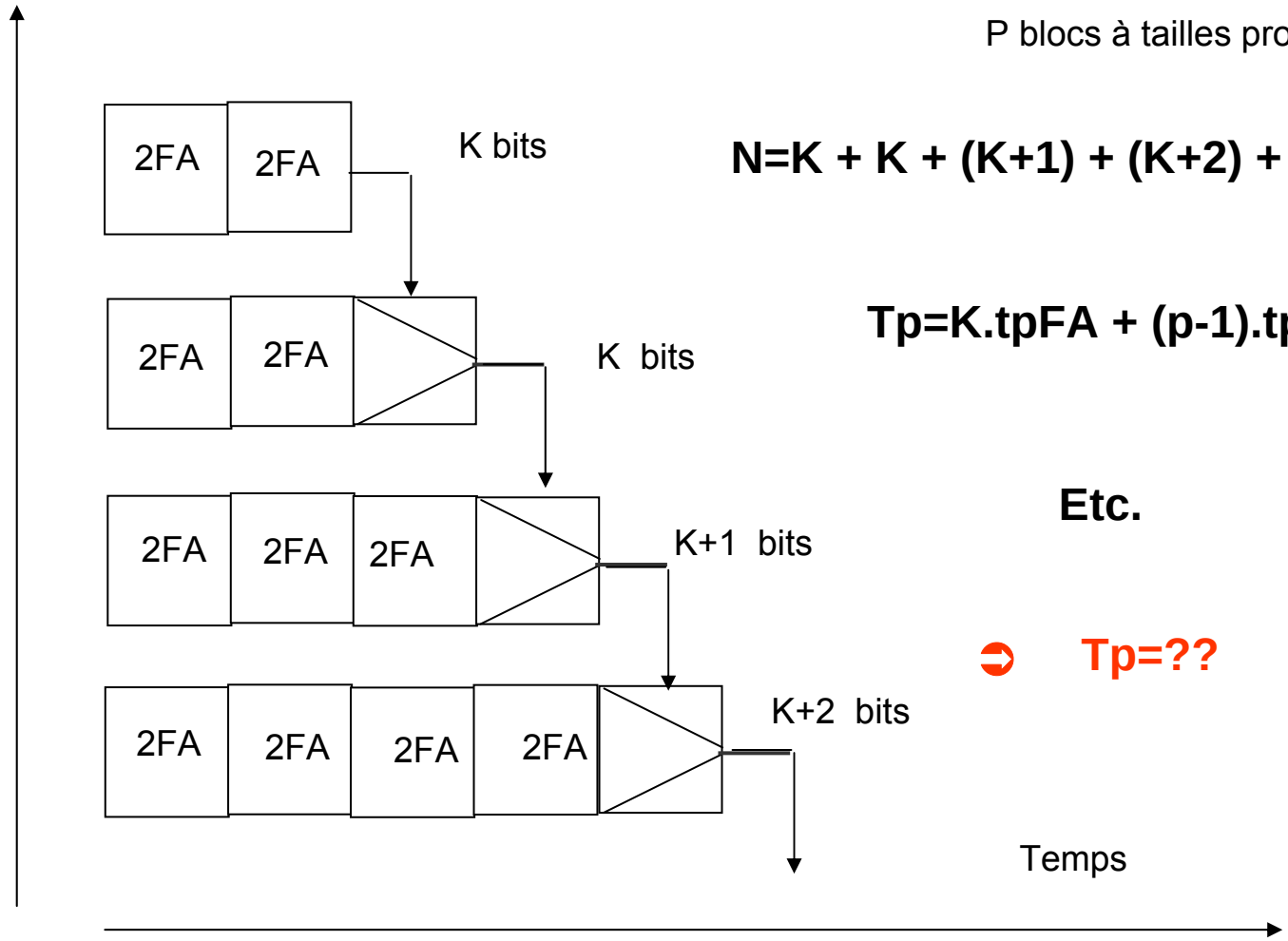


Addition à selection de retenue



Temps	O(√n)
Surface	O(N)

Addition à selection de retenue



P blocs à tailles progressives

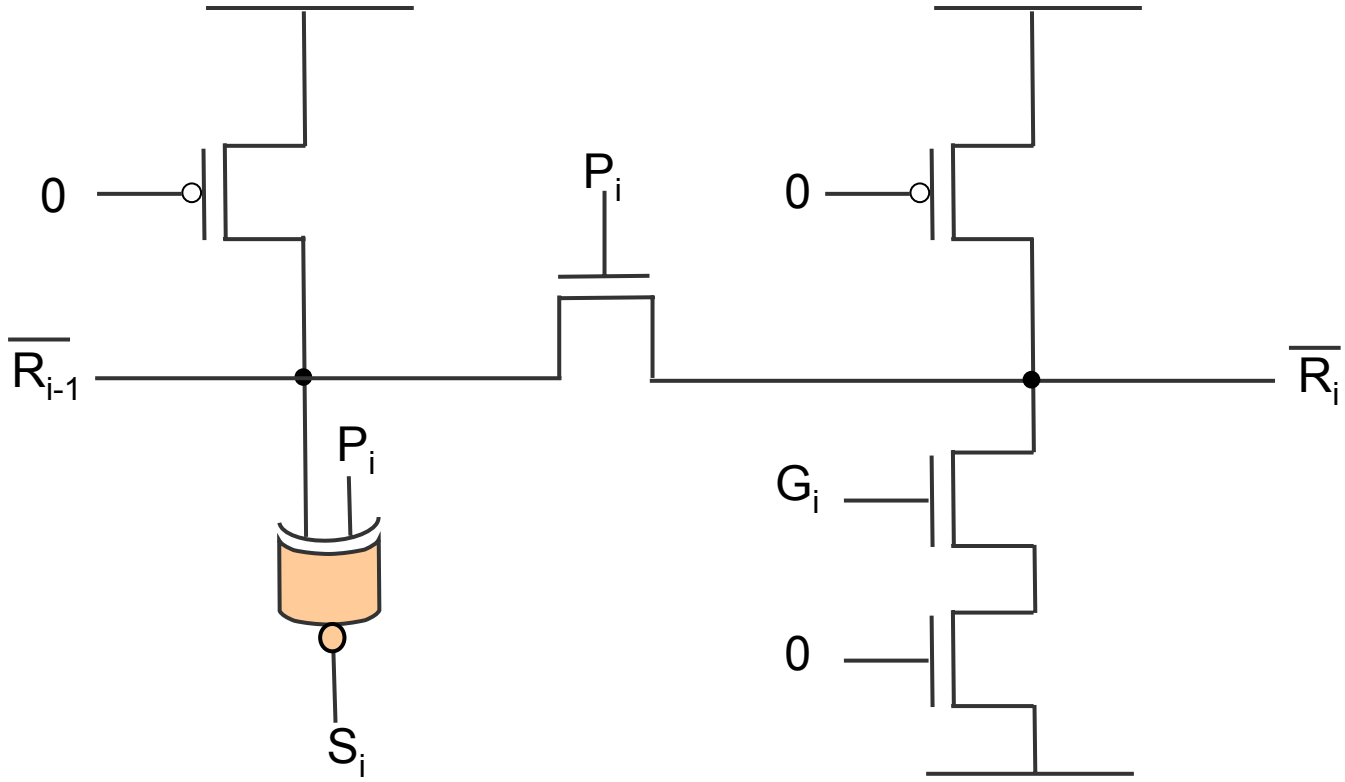
N=K + K + (K+1) + (K+2) + (K+3) + … (K+p-2)

Tp=K.tpFA + (p-1).tpMux

Etc.

➲ **Tp=??**

Propagation de type MANCHESTER

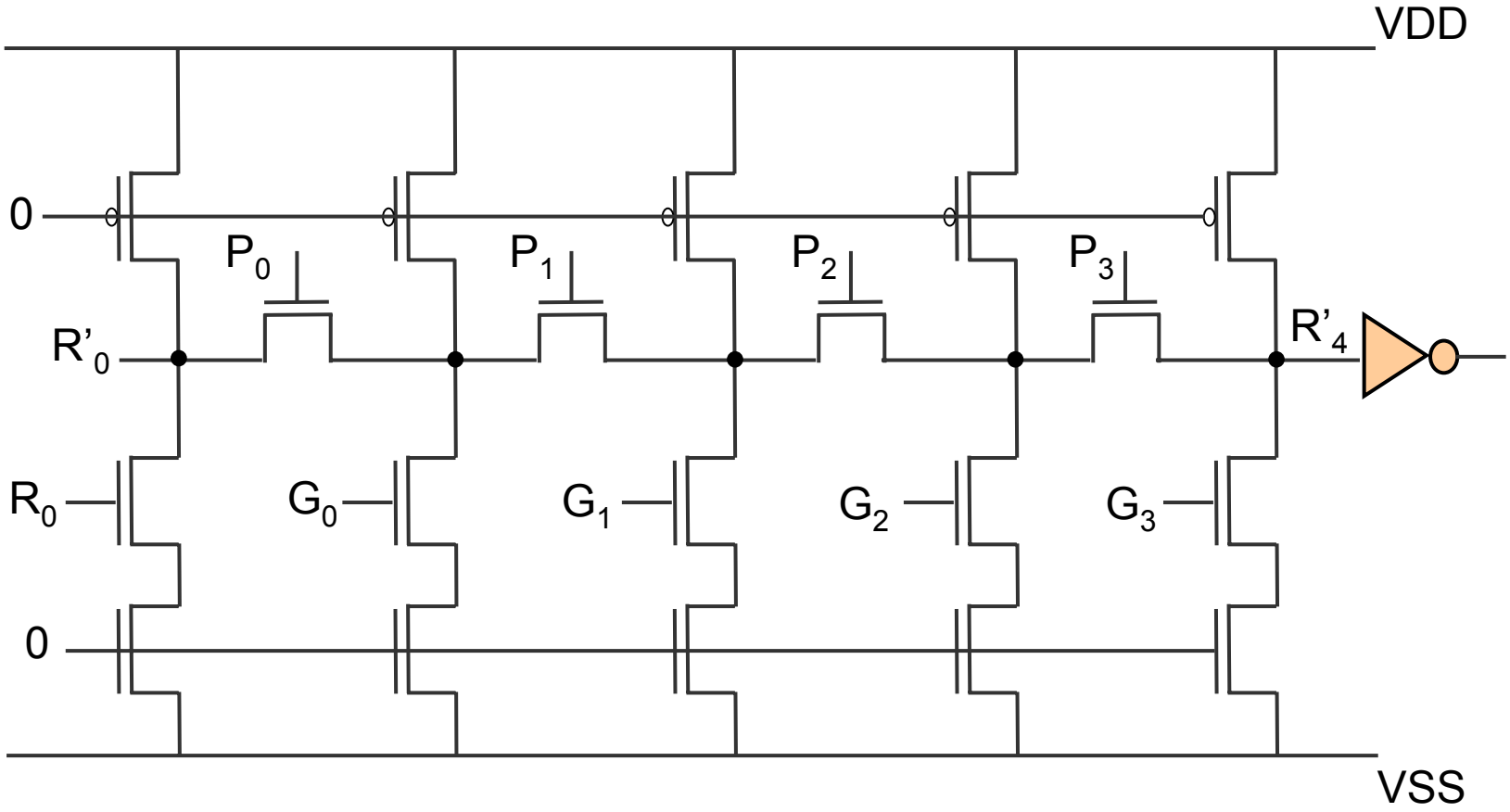


$$\overline{\overline{R_i}} = R_i + P_i\overline{R_{i-1}}$$

$$\overline{\overline{S_i}} = P_i \oplus \overline{R_{i-1}}$$

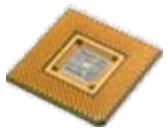
Propagation de type MANCHESTER

- Exemple sur 4 bits



Conclusions

Additionneur	Temps	Surface
Additionneur à propagation de retenue	O(n)	O(n)
Additionneur à sélection de retenue	O(√n)	O(n)
Additionneur à saut de retenue	O(√n)	O(n)
Additionneur recurrence SOLVER (Sklansky)	O(log(n))	O(n*log(n))
Additionneur à report anticipe série	O(n)	O(n)
Additionneur à report anticipe parallèle	O(log(n))	O(n)



ADDITIF

Optimisation des chemins de données arithmétiques par l'utilsation de l'archimétique redondante

Arithmétique mixte : Systèmes classiques de numération (NR)

Numération simple de position

- Nombre non signé uniquement
- en base 2 : un chiffre => un bit
- un nombre *A* sur N chiffres est tel que :

$$A = \sum_{i=0}^{i=N-1} x_i * 2^i$$

$$A \in \{0,...,2^N - 1\} \; et \; x_i \in \{0,1\}$$

Complément à la base

- Nombre signé
- en base 2 : un chiffre => un bit
- un nombre *A* sur N chiffres est tel que :

$$A = -x_{N-1} * 2^{N-1} + \sum_{i=0}^{i=N-2} x_i * 2^i$$

$$A \in \{-2^{N-1},...,2^{N-1} - 1\} \; et \; x_i \in \{0,1\}$$

- **Systèmes autonomes**
- **A chaque nombre correspond un code unique et compact**
- **Opérateurs, fonctions élémentaires et primitives de contrôle déjà existants**

Arithmétique mixte : Systèmes redondants de numération (R)

Notation Carry Save (CS)

Systèmes de numération en base 2 non signé ou signé

- un chiffre $\in$ {0,1,2} => deux bits
- un nombre *CS* sur N chiffres est tel que :

$$CS = \sum_{i=0}^{i=N-1} cs_i^0 * 2^i + \sum_{i=0}^{i=N-1} cs_i^1 * 2^i \quad avec \quad cs_i^0 + cs_i^1 = cs_i \in \{0,1,2\}$$

en non signée : $CS \in \{0,...,2^{N+1} - 2\}$

en signée : $CS \in \{-2^N,...,2^N - 2\}$

- **Systèmes non autonomes, plusieurs codages pour un même nombre**
- **Notation équivalente à une addition non encore effectuée en arithmétique classique**
- **Opérateur : addition en temps constant**

Notation Borrow Save (BS)

Systèmes de numération en base 2 non signé ou signé

- un chiffre $\in$ {-1,0,1} => deux bits
- un nombre *BS* sur N chiffres est tel que :

$$BS = \sum_{i=0}^{i=N-1} bs_i^+ * 2^i - \sum_{i=0}^{i=N-1} bs_i^- * 2^i \quad avec \quad bs_i^+ - bs_i^- = bs_i \in \{-1,0,1\}$$

en non signé et en signé : $BS \in \{2^N + 1, ..., 2^N - 1\}$

- **Systèmes non autonomes, plusieurs codages pour un même nombre**
- **Notation équivalente à une soustraction non encore effectuée**
- **Opérateurs : addition en temps constant**

De quoi s'agit il?

Notation Classique (NR)

en complément à 2

N bits

Ex: +5=0101

Notation redondante

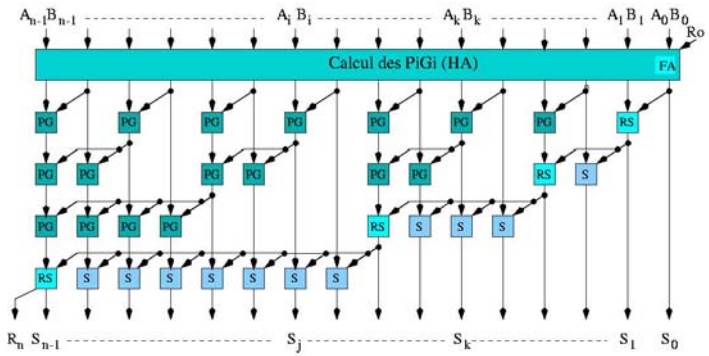
2N bits

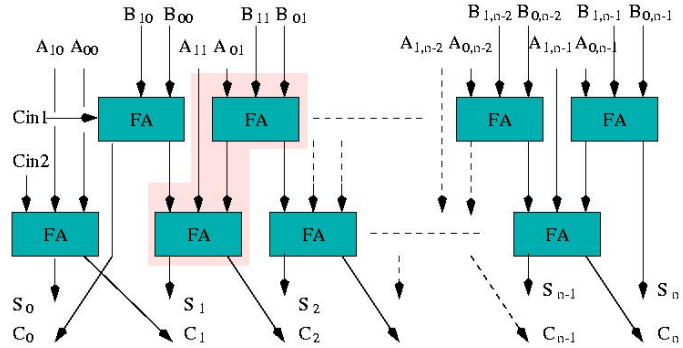
CS: +5= 0100 0011

+ 0001 + 0010

BS: +5= 0111 0110

- 0010 - 0001



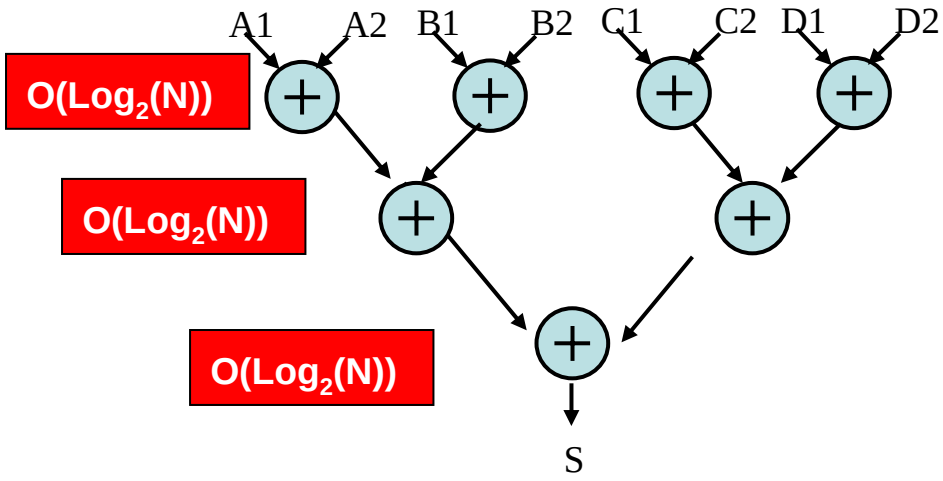


O(Log$_2$(N))

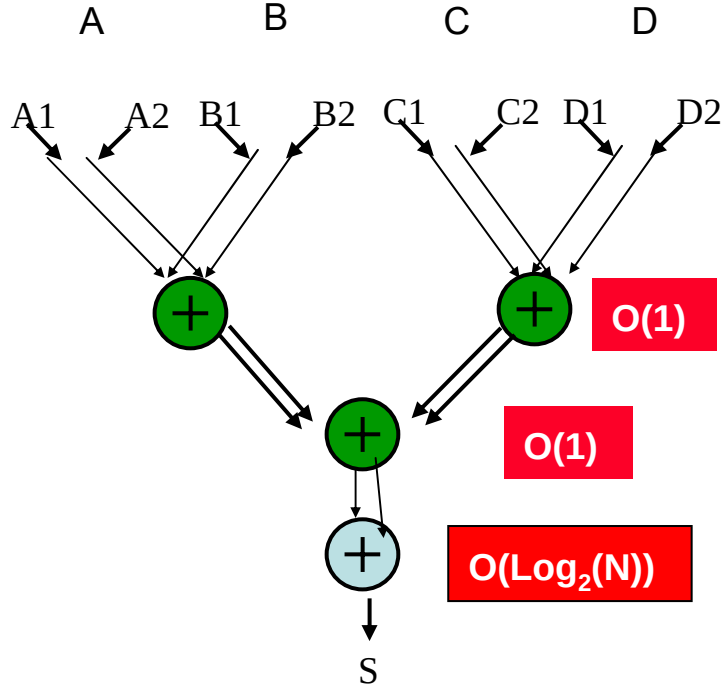
O(1)

Implications

Notation Classique (NR)



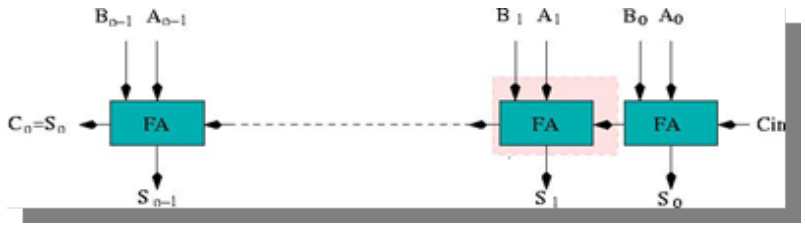
Notation Redondante CS



Gains importants en vitesse, surface et consommation

Architectures des additionneurs classiques

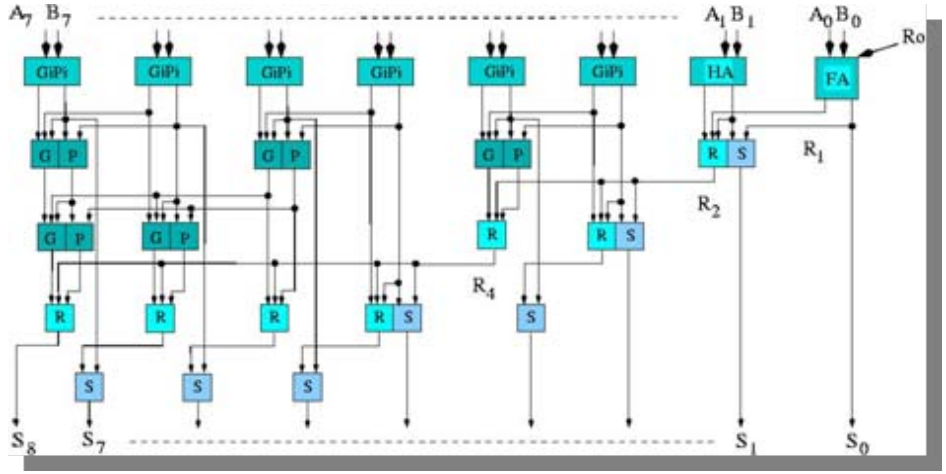
Additionneur « *carry ripple adder* »



1 Couche de Full-Adders, avec propagation de retenues

- ➲ Surface en O(N)
- ➲ Délai en O(N)

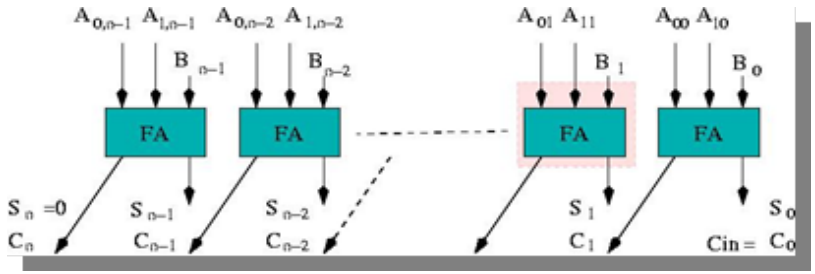
Additionneurs « carry *lookahead adder* »



- ➲ Surface en O(N.log(N))
- ➲ Délai en O(log(N))

Architectures des additionneurs mixtes et redondants

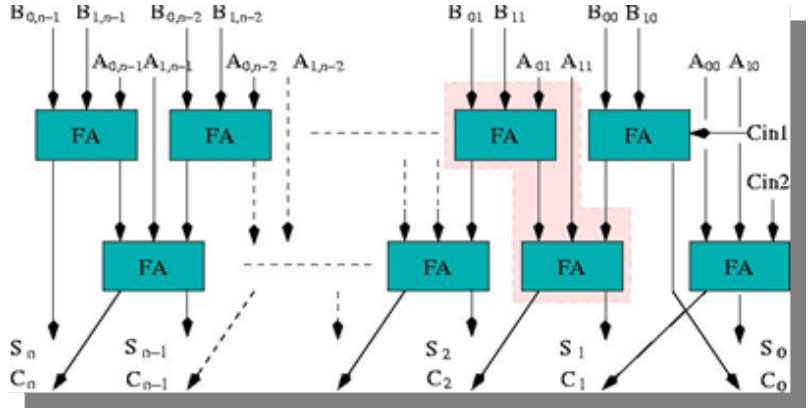
Additionneurs mixtes



1 Couche de Full-Adders sans propagation de retenue

- ➲ Surface en O(N)
- ➲ Délai en O(1)

Additionneurs redondants



2 Couches de Full-Adders sans propagation de retenue

- ➲ Surface en O(N)
- ➲ Délai en O(1)

Additionneurs : comparatif en délai

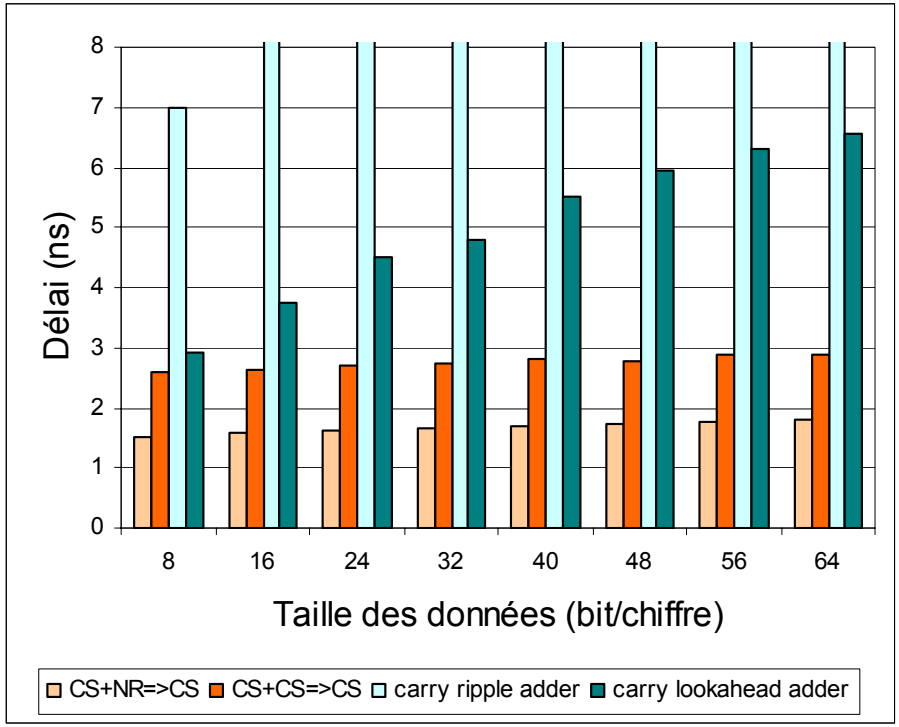
Référence : carry lookahead adder

Additionneurs mixtes :

- ➲ temps constant
- ➲ réduction de 47% à 73%

Additionneurs redondants :

- ➲ temps constant
- ➲ réduction de 10% à 56%



➲Additionneurs mixtes et redondants systématiquement plus performants en délai

➲Le gain croit avec la dynamique

Additionneurs : comparatif en surface

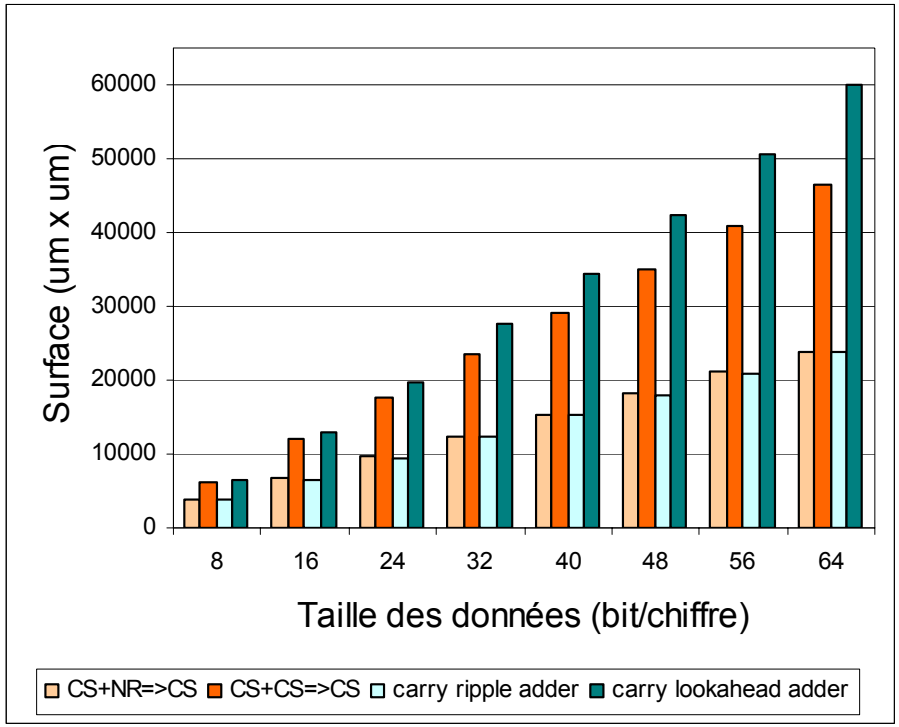
Référence : carry lookahead adder

Additionneurs mixtes :

- ➲ réduction de 40% à 69%
- ➲ identique au carry ripple adder

Additionneurs redondants :

- ➲ réduction de 1% à 22%
- ➲ 2 fois le carry ripple adder



➲Surfaces des additionneurs mixtes et redondants inférieures à la référence

➲Le gain croit avec la dynamique

Additionneurs : comparatif en consommation

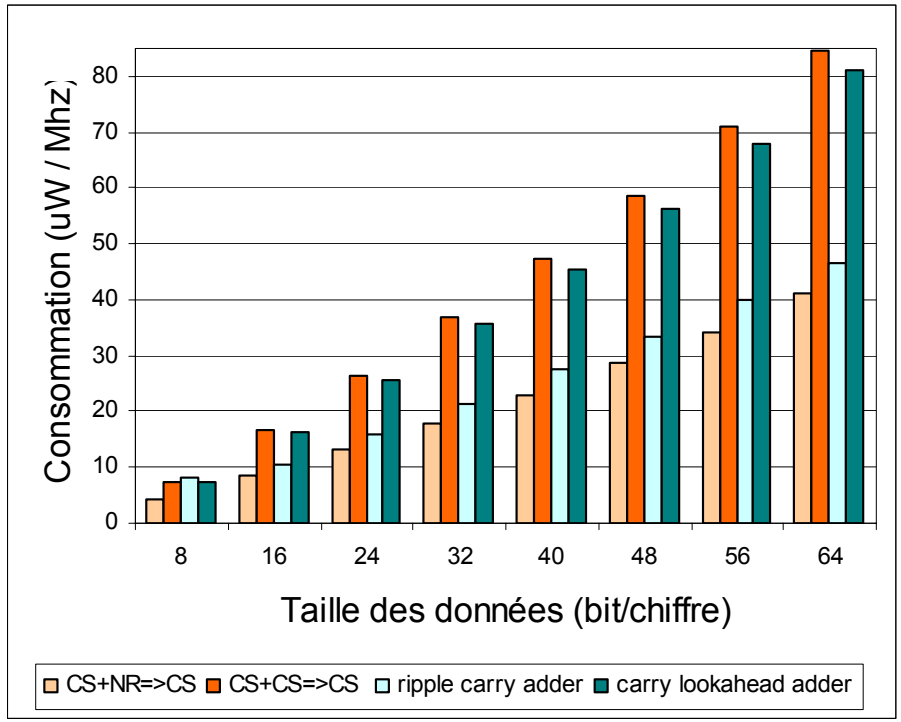
Référence : carry lookahead adder

Additionneurs mixtes :

- ➲ réduction de 43% à 49%
- ➲ inférieure au carry ripple adder

Additionneurs redondants :

- ➲ augmentation de 2% à 4%
- ➲ < 2 fois carry ripple adder



➲Consommation des additionneurs mixtes et redondants de *«très inférieures à égales»*

➲rapport de consommation quasi constant pour une même architecture