Additionneurs

Ou comment les réaliser

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Février 2024

Demi additionneur (half adder)

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- $S = \overline{A}B + A\overline{B} = A \oplus B$
- $C_{out} = AB$

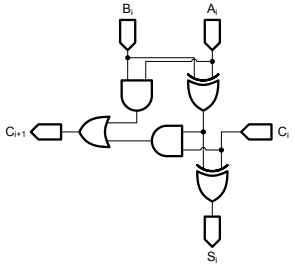
Additionneur complet (full adder)

A	B	C_{in}	C_{out}	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

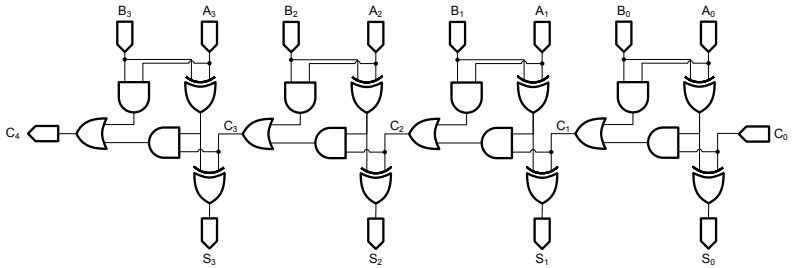
- $S = \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C_{in}} + ABC_{in} + A\overline{B}\overline{C_{in}}$
- $C_{out} = BC_{in} + AB + AC_{in}$

Additionneur complet: implémentation

$$
\begin{aligned}
S &= \overline{A}\overline{B}C_{in} + \overline{A}B\overline{C_{in}} + ABC_{in} + A\overline{B}\overline{C_{in}} \\
S &= \overline{A}\overline{B}C_{in} + ABC_{in} + \overline{A}B\overline{C_{in}} + A\overline{B}\overline{C_{in}} \\
S &= (\overline{A}\overline{B} + AB)C_{in} + (\overline{A}B + A\overline{B})\overline{C_{in}} \\
S &= (\overline{A \oplus B})C_{in} + (A \oplus B)\overline{C_{in}} \\
S &= A \oplus B \oplus C_{in} \\
C_{out} &= BC_{in} + AB + AC_{in}
\end{aligned}
$$



Additionneur 4 bits



- Délai combinatoire pour n bits: $2n + 1$
- Délai de retenue à retenue: $2n$

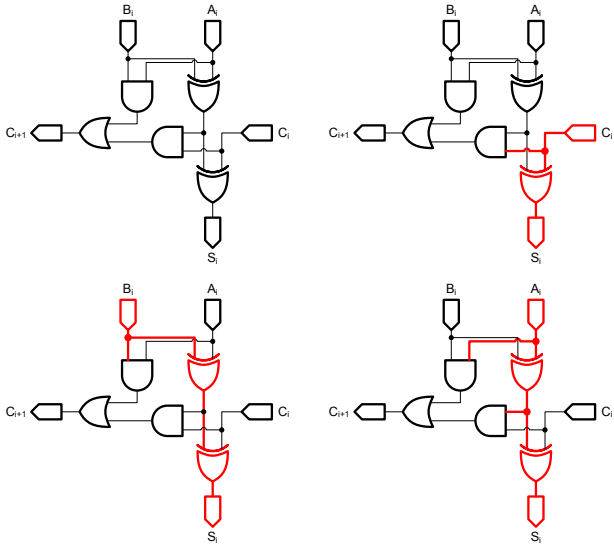
Additionneur à anticipation de retenue

- La propagation de retenue a un délai proportionnel au nombre de bits de l'additionneur...
- Est-il possible d'optimiser la structure?
- L'idée est de calculer les retenues C_i. Une fois qu'elles sont connues, le résultat est directement calculé.
- Pour un additionneur complet:

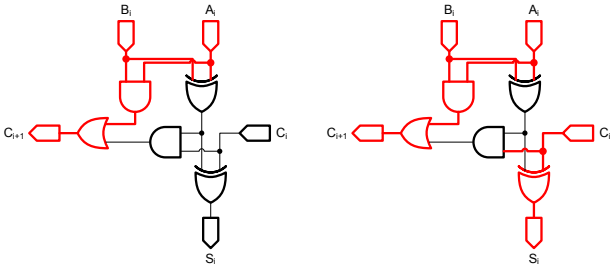
$$C_{out} = BC_{in} + AB + AC_{in}$$

$$C_{out} = \underbrace{AB}_{\text{génération}} + \underbrace{(A+B)\,C_{in}}_{\text{propagation}}$$

Ni génération ni propagation de retenue

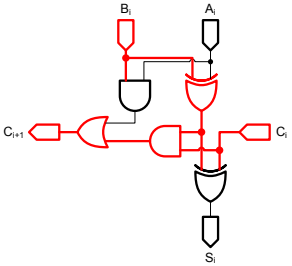


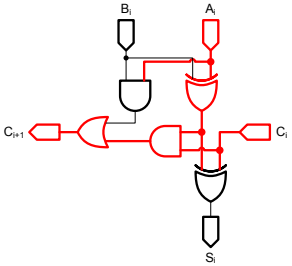
Génération de retenue

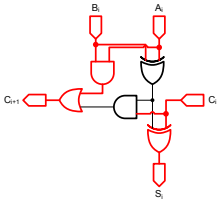


- On définit:
- $g_i = A_i B_i$

Propagation de retenue







- On définit:
- $p_i = A_i + B_i$

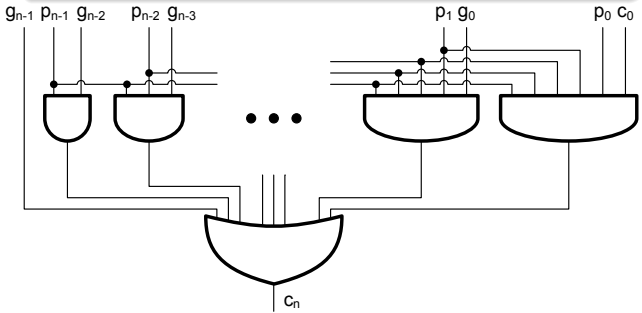
Propagation de retenue

- $g_i = a_i b_i$
- $p_i = a_i + b_i$
- $c_i = a_{i-1} b_{i-1} + (a_{i-1} + b_{i-1}) c_{i-1}$
- $c_i = g_{i-1} + (p_{i-1} c_{i-1})$
- $c_i = g_{i-1} + (p_{i-1}(g_{i-2} + (p_{i-2} c_{i-2})))$
- $c_i = g_{i-1} + (p_{i-1}(g_{i-2} + (p_{i-2}(g_{i-3} + (p_{i-3} c_{i-3})))))$

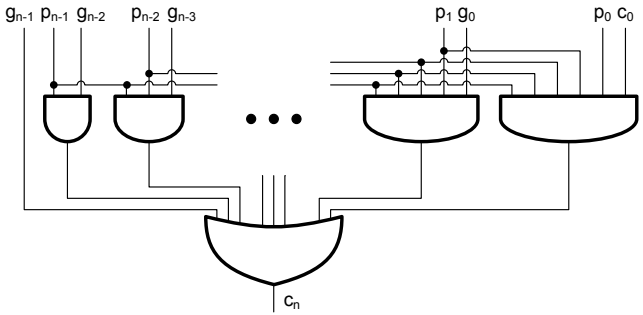
$$
\begin{aligned}
c_i \quad = \quad & g_{i-1} + (p_{i-1} g_{i-2}) \\
& + (p_{i-1} p_{i-2} g_{i-3}) \\
& + \cdots \\
& + (p_{i-1} p_{i-2} \cdots p_1 g_0) \\
& + (p_{i-1} p_{i-2} \cdots p_1 p_0 c_0)
\end{aligned}
$$

Anticipation de retenue: implémentation directe

$$
\begin{aligned}
c_i \quad = \quad & g_{i-1} + (p_{i-1} g_{i-2}) \\
& + (p_{i-1} p_{i-2} g_{i-3}) \\
& + \cdots \\
& + (p_{i-1} p_{i-2} \cdots p_1 g_0) \\
& + (p_{i-1} p_{i-2} \cdots p_1 p_0 c_0)
\end{aligned}
$$



Anticipation de retenue: implémentation directe



- Problèmes:
  - entrance de $n+1$ sur la porte OU et la porte ET de droite
  - p_{n-1} doit attaquer n portes
  - structure irrégulière, et beaucoup de logique nécessaire

Anticipation de retenue: meilleure solution

- Même idée de base, mais construction par étape:
- Une retenue sortant de la position 0 est générée si:
  - Une retenue est générée à la position 0, ou
  - Une retenue dans la position 0 est propagée

$$c_1 = g_0 + c_0 p_0$$

- Une retenue sortant de la position 1 est générée si:
  - Une retenue est générée à la sortie du bloc constitué des deux premiers bits (G_{01}), ou
  - Une retenue est propagée par ce bloc (P_{01})

$$\begin{aligned} c_2 &= G_{01} + P_{01} c_0 \\ G_{01} &= g_1 + p_1 g_0 \\ P_{01} &= p_1 p_0 \end{aligned}$$

Anticipation de retenue: meilleure solution

- De manière générale, pour $i < j, j + 1 < k$:

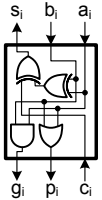
$$
\begin{aligned}
c_{k+1} &= G_{ik} + P_{ik} c_i \\
G_{ik} &= G_{j+1,k} + P_{j+1,k} G_{ij} \\
P_{ik} &= P_{ij} P_{j+1,k} \\
P_{ii} &= p_i \\
G_{ii} &= g_i
\end{aligned}
$$

- P_{ik} indique si le bloc ik propage la retenue
- G_{ik} indique si le bloc ik génère une retenue

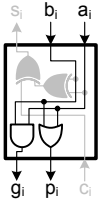
Réalisation

- Deux blocs sont nécessaires
- Le premier génère p et g, et calcul le résultat

Bloc de calcul



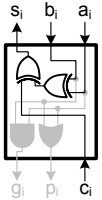
calcul de p et g



$$p_i = a_i + b_i$$

$$g_i = a_i b_i$$

Calcul du résultat

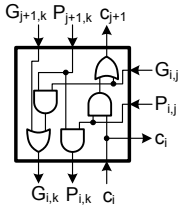


$$s_i = a_i \oplus b_i \oplus c_i$$

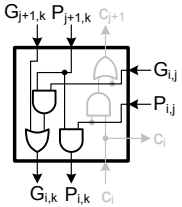
Réalisation

- Deux blocs sont nécessaires
- Le deuxième calcule les P et G, ainsi que les retenues

Bloc de calcul



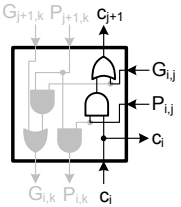
Calcul de P et G



$$P_{i,k} = P_{i,j}P_{j+1,k}$$

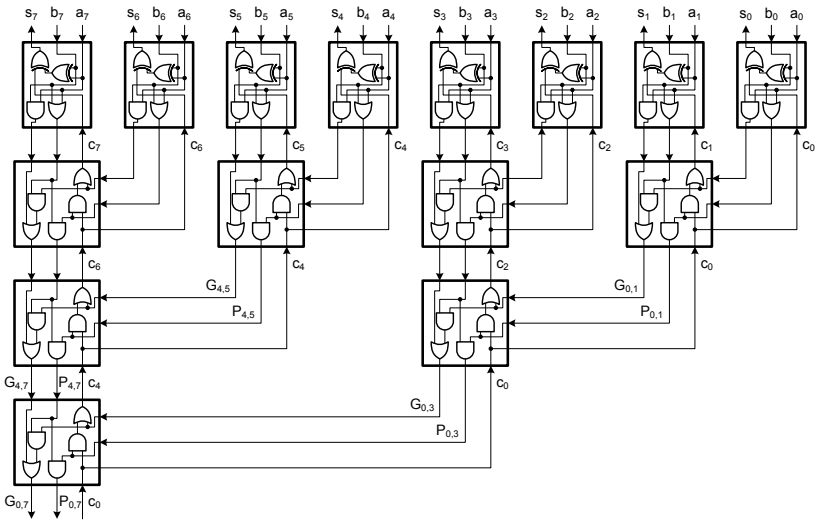
$$G_{i,k} = G_{j+1,k} + P_{j+1,k}G_{i,j}$$

Calcul de la retenue

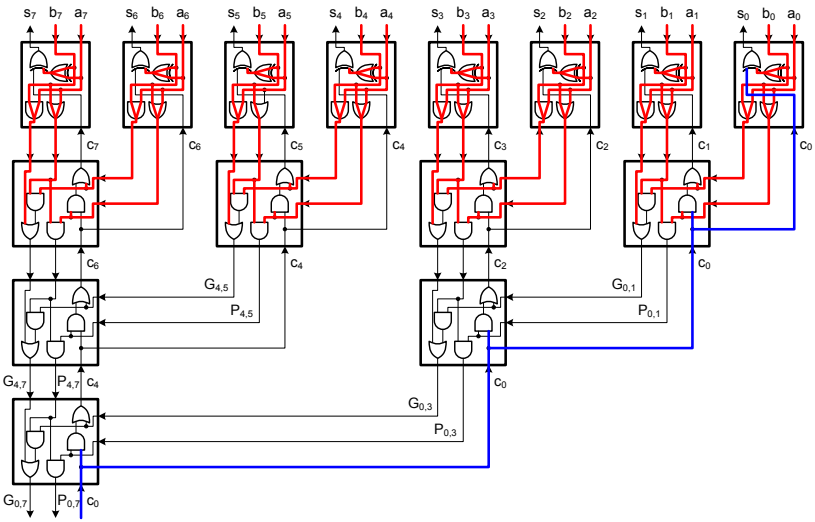


$$c_{j+1} = G_{i,j} + P_{i,j}c_i$$

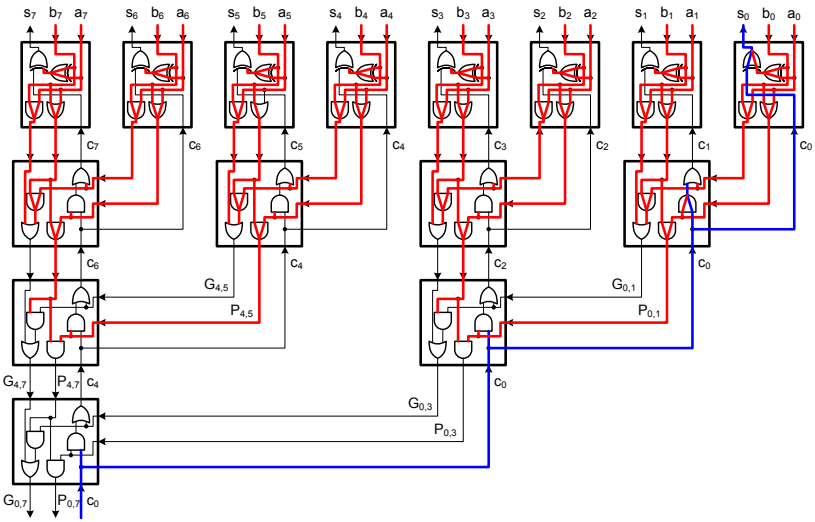
Implémentation



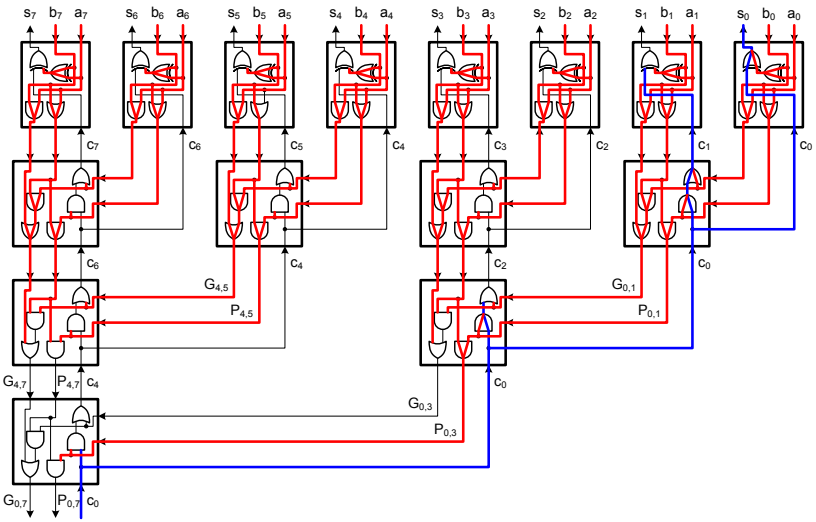
Implémentation: temps 1



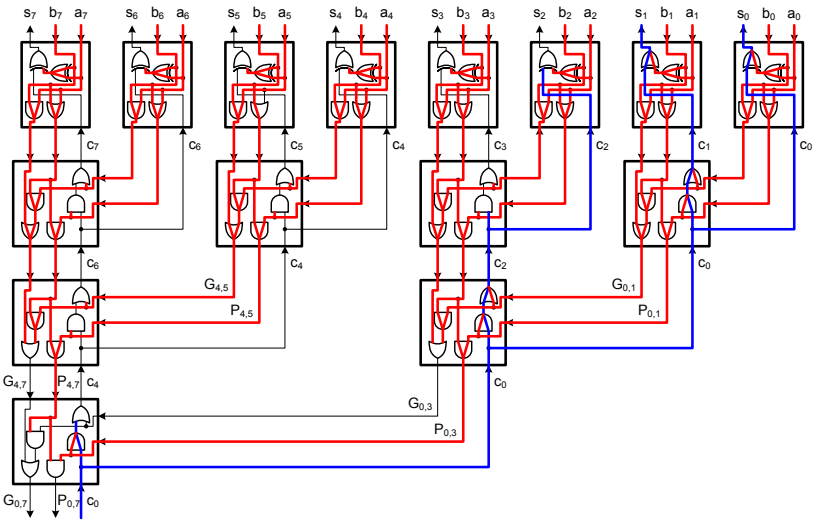
Implémentation: temps 2



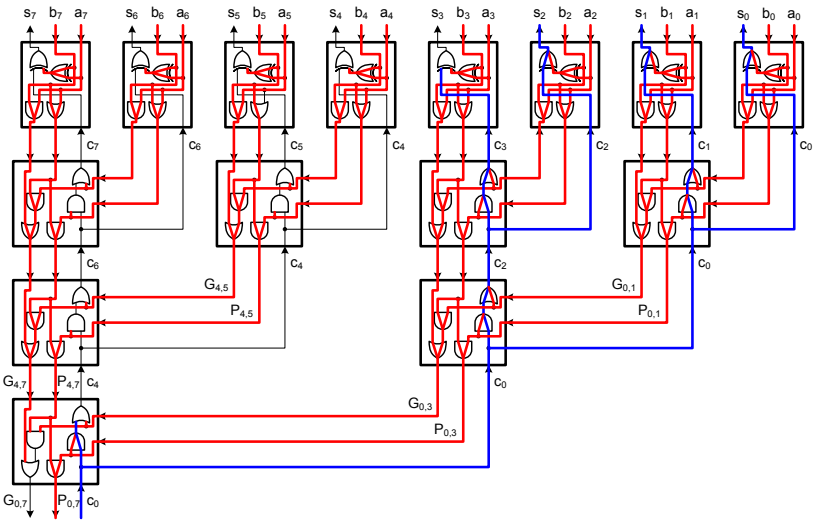
Implémentation: temps 3



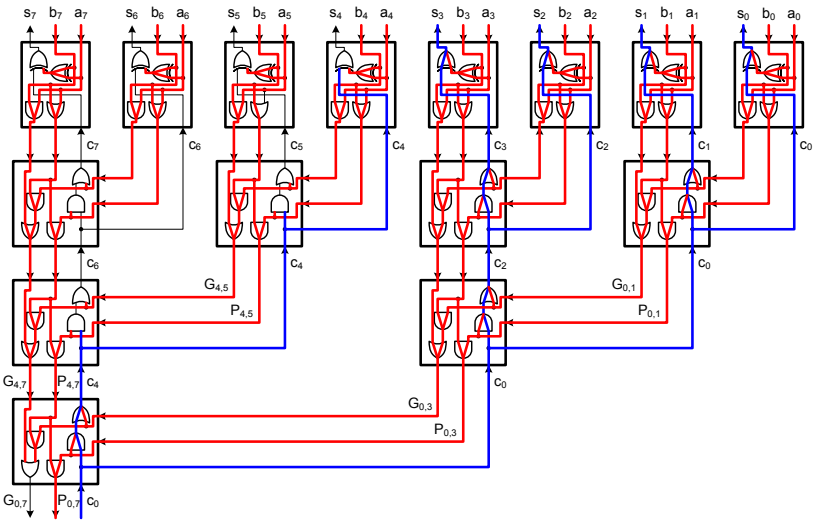
Implémentation: temps 4



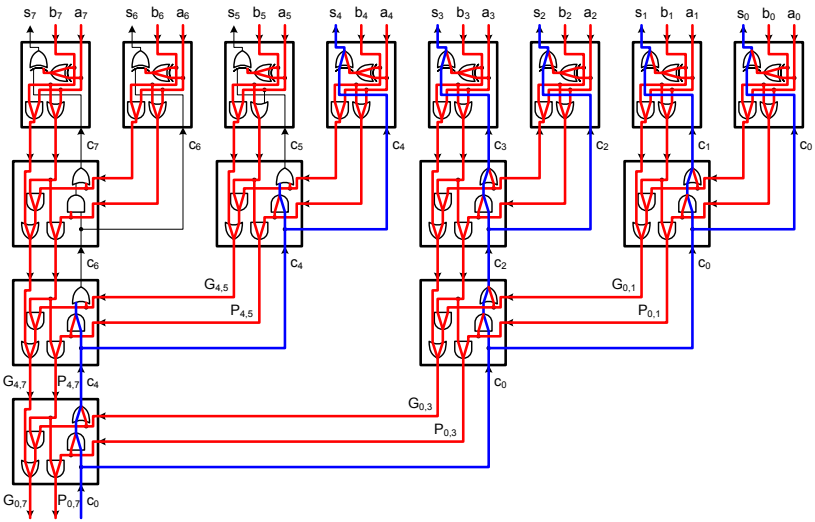
Implémentation: temps 5



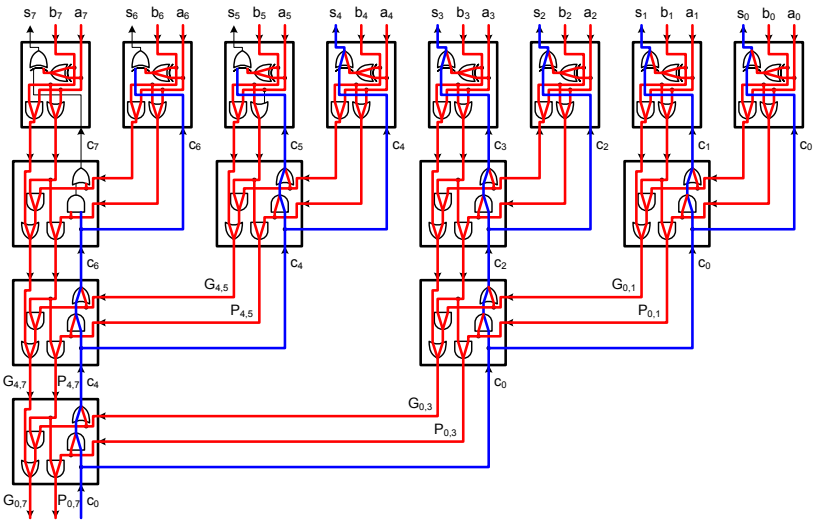
Implémentation: temps 6



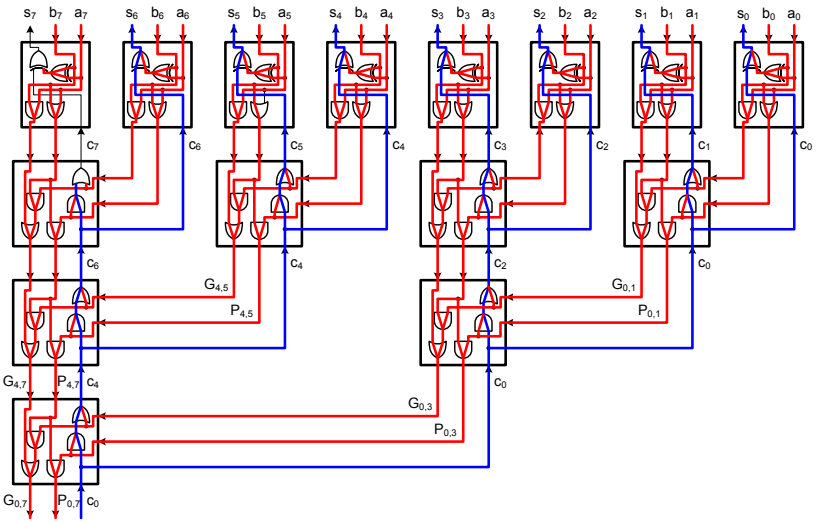
Implémentation: temps 7



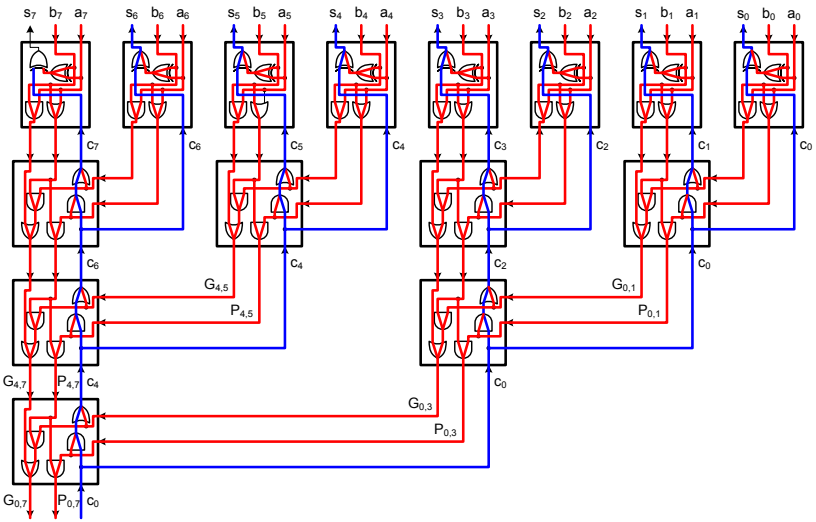
Implémentation: temps 8



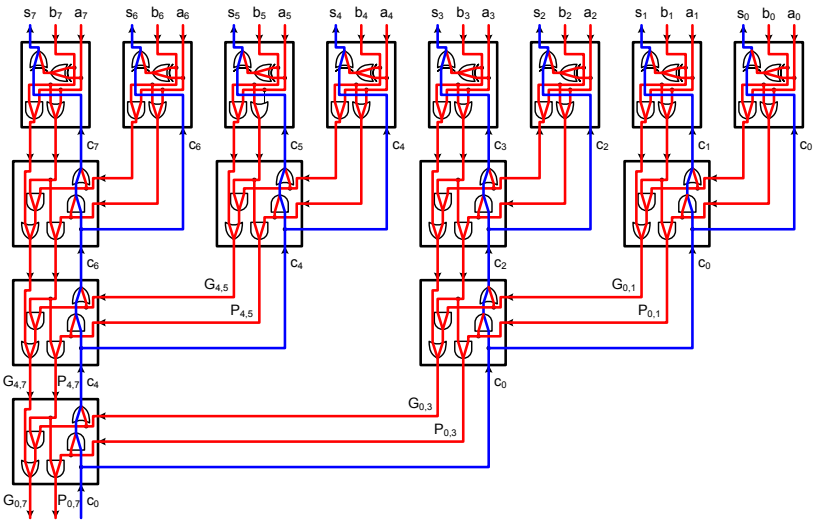
Implémentation: temps 9



Implémentation: temps 10



Implémentation: temps 11



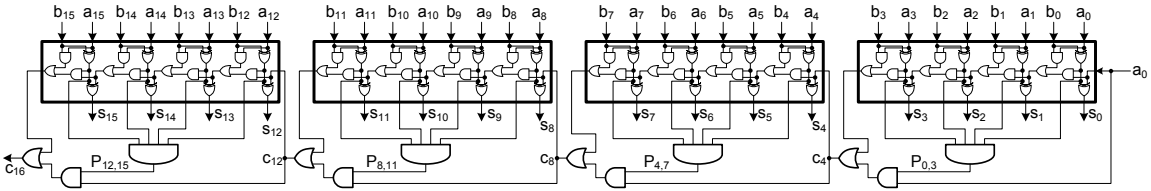
Comparaison

- Comparaison des délais entre deux additionneurs

Bits	Ripple Carry	Look-ahead Carry
1	2	2
4	8	6
8	16	10
12	24	10
16	32	10
20	40	14
24	48	14
32	64	14
64	128	14

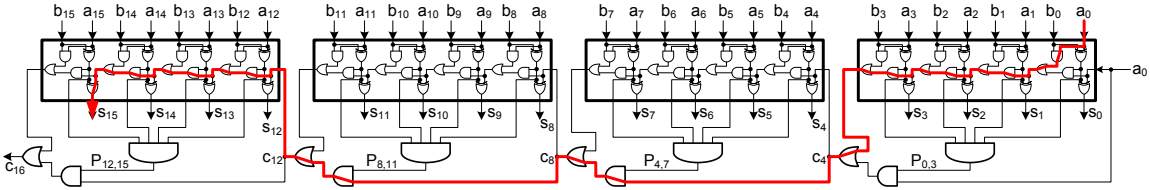
Additionneur à saut de retenue

- Principe:
  - La propagation de retenue est plus rapide à calculer que la génération



Additionneur à saut de retenue

- Chemin le plus long

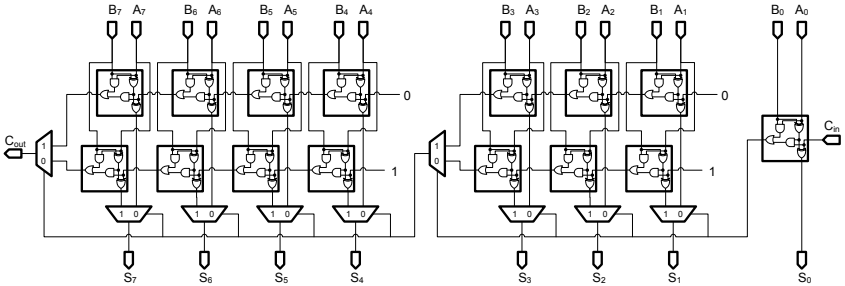


Meilleure solution

- Idée: blocs de taille variable
- Exemple: additionneur de 20 bits
- Blocs de taille 4: délai proportionnel à $4 + (20/4 - 2) + 4 = 11$ unités de temps
- Blocs variables (2-5-6-5-2): délai proportionnel à 9 unités de temps

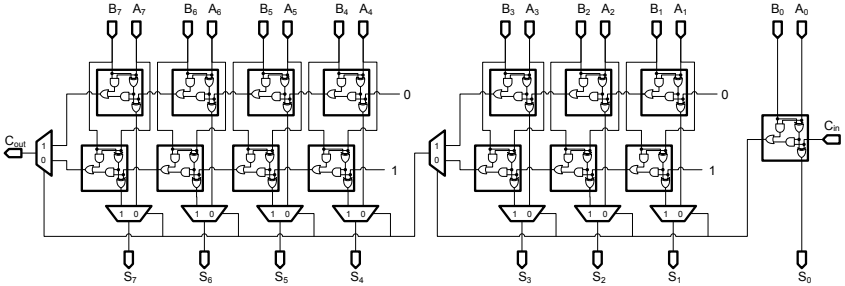
Sélection de retenue

- Principe:
  - Calculer en parallèle avec une retenue de 1 ou 0, puis sélectionner le résultat



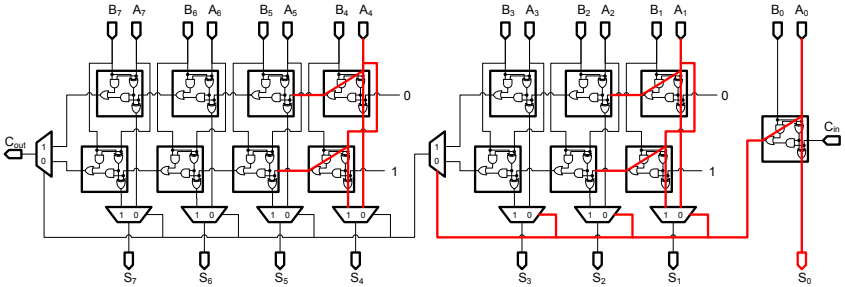
Carry select

- *Temps* = 0



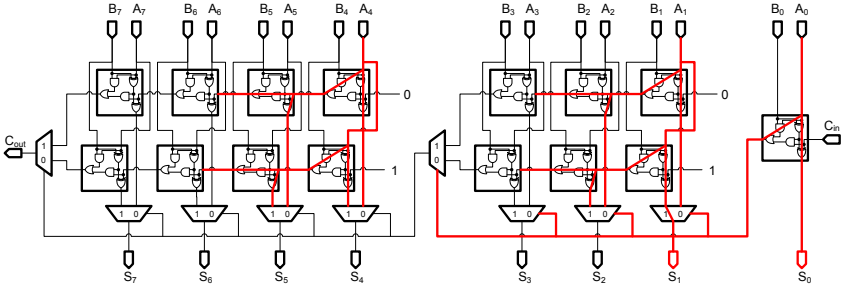
Carry select

- *Temps* $= 3$



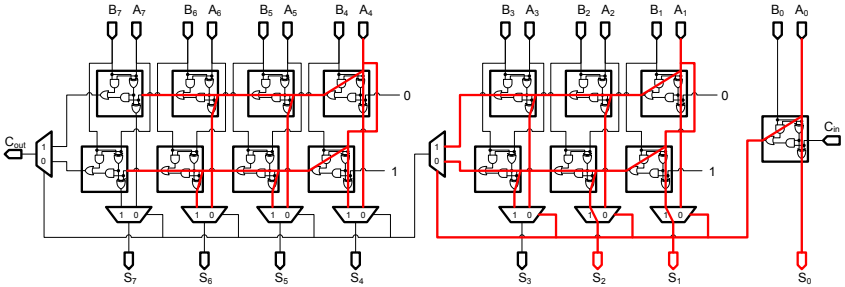
Carry select

- *Temps* $= 5$



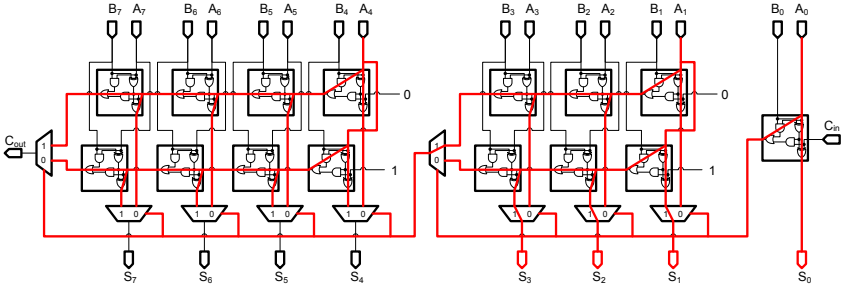
Carry select

- *Temps* $= 7$



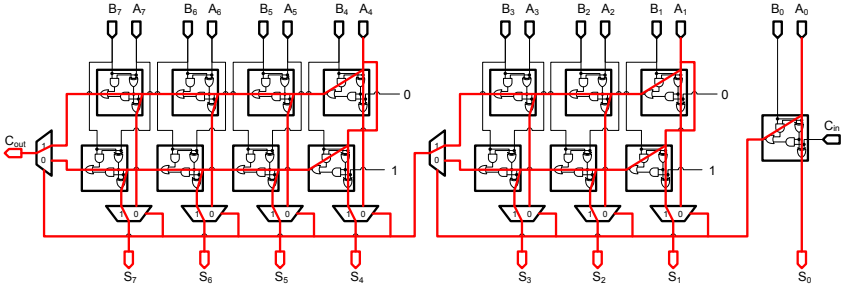
Carry select

- *Temps* = 9



Carry select

- *Temps* = 11



Comparaison

Type	**Temps**	**Espace**
Propagation simple	$O(n)$	$O(n)$
Retenue anticipée	$O(\log n)$	$O(n \log n)$
Saut de retenue	$O(\sqrt{n})$	$O(n)$
Sélection de retenue	$O(\sqrt{n})$	$O(n)$