Opérateurs arithmétiques matériels

Arnaud Tisserand
INRIA LIP Arénaire

ARCHI03
Roscoff, 31 mars / 04 avril 2003




Plan de l'exposé

1. Du transistor aux portes élémentaires
2. Addition
3. Multiplication
4. Division

Adéquation arithmétique–architecture

Seule une bonne connaissance à la fois en arithmétique des ordinateurs et en architecture des ordinateurs permet de concevoir des opérateurs de calcul performants !
— un grand savant, un jour. . .

Il faut concilier :

- représentation des nombres
- algorithmes pour les opérations
- propriétés mathématiques
- structure des circuits
- comportement électrique
- performances (vitesse, surface, consommation d'énergie)

Partie 1

Du transistor aux portes élémentaires. . .

Points abordés dans cette partie

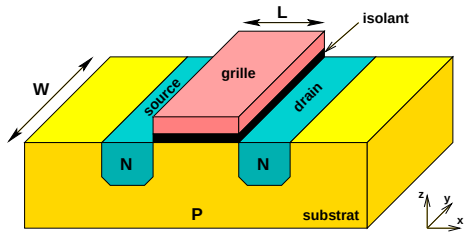
- Fonctionnement de base des transistors
- Codage physique des niveaux logiques
- Éléments ou portes de base
- Mise en lumière de certaines limitations
- Consommation d'énergie
- Ordre de grandeur de caractéristiques classiques

Structure des transistors MOS

Il existe deux types de transistors MOS (*metal oxide semiconductor*) :
↪ les transistors N et les transistors P

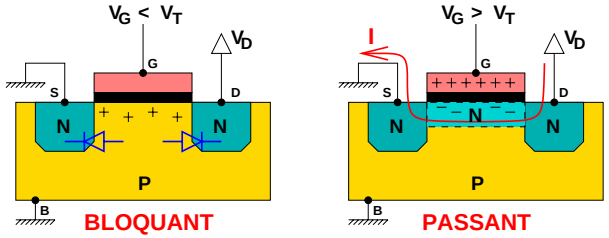
Un transistor de type N est composé de :

- substrat (Si) dopé P
- drain et source dopés N
- couche isolante
- grille



Dans une zone dopée N, les porteurs de charge majoritaires sont des électrons (ce sont des trous dans le cas d'une zone dopée P). Pour un transistor de type P on inverse les dopages.

Fonctionnement du transistor N



Le transistor change d'état en fonction de la tension de grille V_G (commande) par rapport à la tension de seuil V_T (cst techno) :

- Cas bloquant $V_G < V_T$: la source et le drain sont isolés par les jonctions PN (diodes).
- Cas passant $V_G > V_T$: des électrons libres du substrat (porteurs minoritaires) sont attirés sous la grille $\longrightarrow$ inversion de polarité $\longrightarrow$ circulation du courant I dans la couche de type N entre le drain et la source.

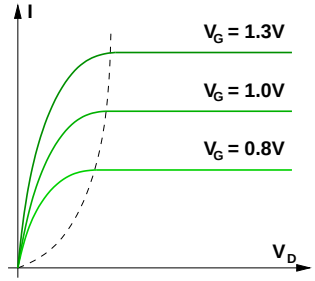
Modèles du transistor

Caractéristique de sortie d'un transistor :

$$I = \begin{cases} 0 & V_G < V_T \\ \beta\left((V_G - V_T)V_D - \frac{{V_D}^2}{2}\right) & 0 < V_D < V_G - V_T \\ \frac{\beta}{2}(V_G - V_T)^2 & 0 < V_G - V_T < V_D \end{cases}$$

où

$$\beta = C_{\text{techno}} \times \frac{W}{L}$$



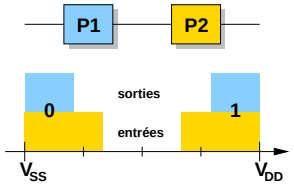
Modèle logique :

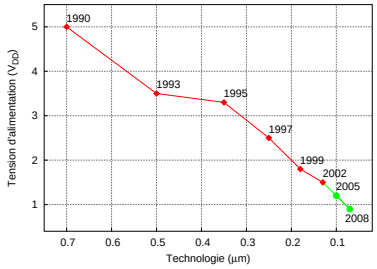
	Transistor N	Transistor P
		
G à 0	bloquant	passant
G à 1	passant	bloquant

Valeurs logiques

- Le zéro logique (0) est codé par la tension de référence (masse) notée V_{SS} ou ⏚.
- Le un logique (1) est codé par la tension d'alimentation (positive) notée V_{DD} ou ⍋.

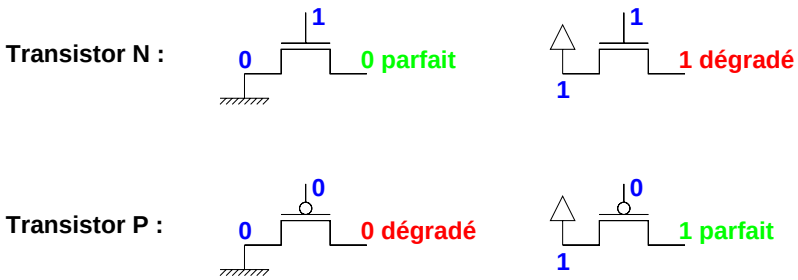
Il faut adopter un codage des tensions permettant un bon fonctionnement en présence de bruit modéré ⟶ état = plage de tensions :





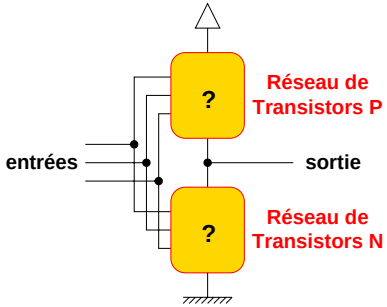
Problème de transmission de certaines valeurs

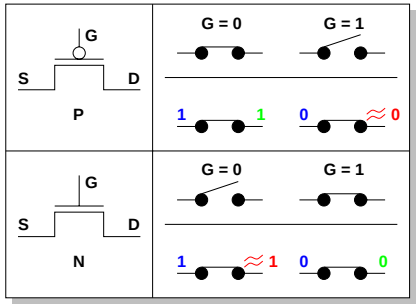
Du fait des tensions de seuil, les transistors ne laissent pas passer correctement toutes les valeurs :



Portes logiques CMOS

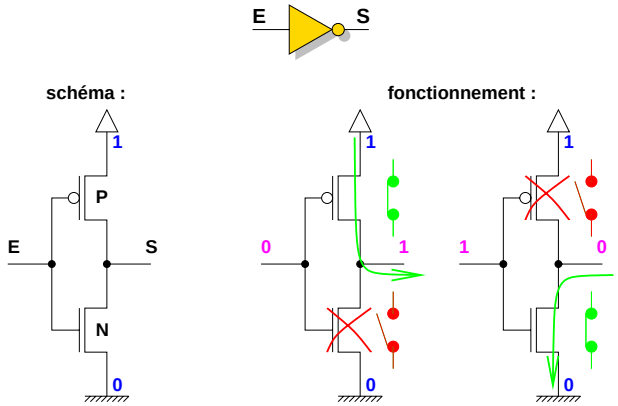
Il existe de nombreuses solutions pour faire des portes logiques à partir des transistors. La plus utilisée aujourd'hui est la logique CMOS (*complementary MOS*). On utilise au mieux les deux types de transistors : N et P.



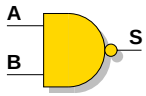


Inverseur

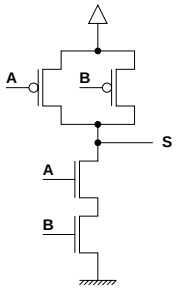
C'est la porte la plus simple : juste deux transistors (1 N et 1 P).



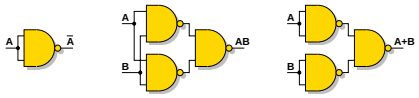
Porte NAND (non–et)



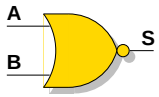
A	B	S
0	0	1
0	1	1
1	0	1
1	1	0



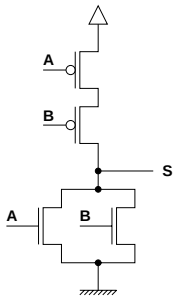
La porte NAND est universelle. On peut faire toutes les autres portes logiques avec des combinaisons de portes NAND.



Porte NOR (non–ou)

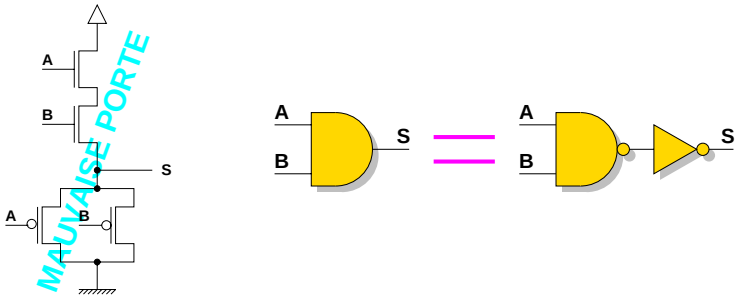


A	B	S
0	0	1
0	1	0
1	0	0
1	1	0



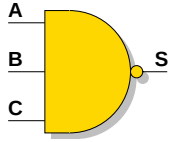
Comment faire une porte AND ?

Il y a une mauvaise solution et une bonne. . .

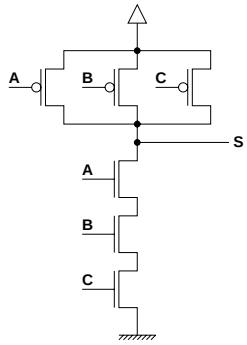


- Les niveaux en sortie sont de mauvaise qualité.
- Solution : $AB = \overline{\overline{AB}}$ (porte à 6 transistors)

Porte NAND à 3 entrées



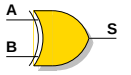
A	B	C	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Le nombre de transistors en série dans les réseaux N et P est limité (de 3 à 5 en général). Au dessus de cette limite il faut réaliser la fonction par composition de plusieurs portes.

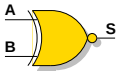
Autres portes élémentaires pour l'arithmétique

XOR (ou exclusif)



A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

XNOR



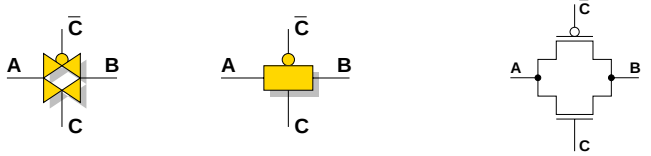
A	B	S
0	0	1
0	1	0
1	0	0
1	1	1

MUX (multiplexeur)

C	S
0	A
1	B

Porte de transfert

Le robinet pour circuits. Lorsque la commande C est à 1, les nœuds A et B sont connectés sinon ils sont isolés.



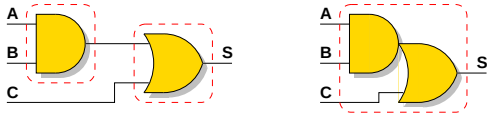
Dans une porte de transfert ou de passage (*pass gate*), les états sur les nœuds A et B ne sont pas forcés via une borne d'alimentation (V_{DD} ou V_{SS}). Le signal se dégrade lors du passage dans une suite de telles portes (prévoir une régénération régulière).

Cellules complexes

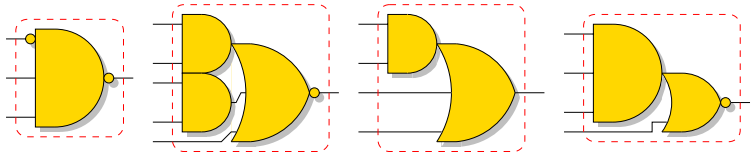
Il est facile de réaliser des portes donnant des combinaisons plus ou moins complexes de AND-OR en CMOS.

Exemples :

- Porte AO21 (gain de 25% en surface et de 40% en vitesse)

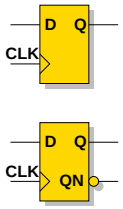


- Quelques autres portes classiques



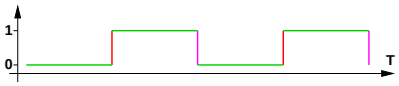
Éléments de mémorisation

Il existe une multitude d'éléments de mémorisation dans les bibliothèques de portes. Dans la suite, nous allons utiliser principalement des bascules D.



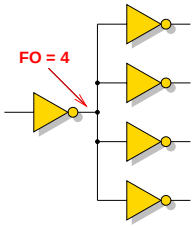
CLK	D	Q(t+1)	QN(t+1)
1	X	Q(t)	QN(t)
0	X	Q(t)	QN(t)
↑	0	0	1
↑	1	1	0

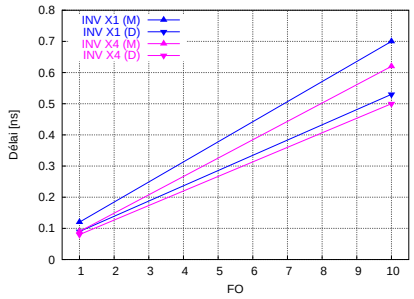
Remarque :
↑ est le front montant d'horloge.

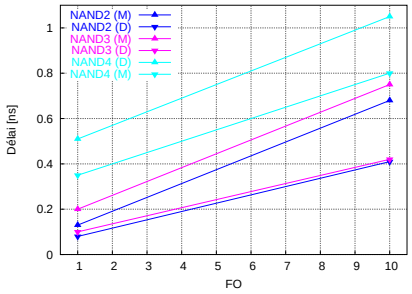


Sortance

Le délai d'une porte (temps pour changer d'état) dépend de sa charge en sortie. La sortance (fan-out) donne une indication de cette charge. On la mesure comme le nombre d'entrées de portes reliées à la sortie (normalisé en nombre d'entrées d'un inverseur de base en général).

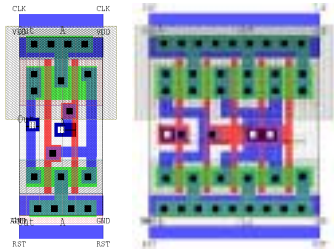


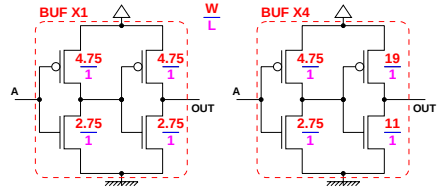




Régénération du signal par *buffer*

La fonction du *buffer* est juste de régénérer le signal sans rien changer à sa valeur ($f(x) = x$). On le réalise en mettant en série deux inverseurs.





caractéristique	BUF X1	BUF X4
taille (h×l) [λ]	53 × 25	53 × 50
capacité A [fF]	5.89	5.89
$T_{0\to1}$	$11 + 439 \times C_{out}$	$17 + 132 \times C_{out}$
$T_{1\to0}$	$12 + 318 \times C_{out}$	$21 + 137 \times C_{out}$

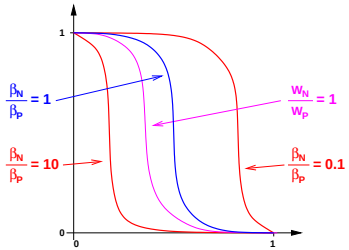
Dimensionnement des transistors

Le gain d'un transistor est composé d'un facteur technologique constant C et d'un facteur géometrique :

$$\beta = C \times \frac{W}{L}$$

La mobilité des électrons et des trous n'est pas la même. Le facteur technologique est différent pour les transistors N et les transistors P (rapport entre 2 et 3 environ).

Dimensionnement inverseur :
- $\frac{\beta_N}{\beta_P} = 1$ ($T_{0\to1} \approx T_{1\to0}$)
- L minimal
- $W = f(FO)$



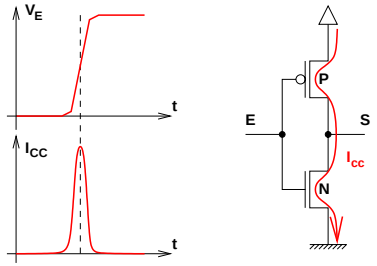
Consommation d'énergie

On peut décomposer la consommation d'énergie d'un circuit intégré en plusieurs contributions de natures différentes :

- Consommation statique : problèmes essentiellement technologiques
  - Fuites dans les jonctions PN (courant de diode en inverse)
  - Fuite sous-seuil du canal
- Consommation dynamique :
  - Court-circuits lors des transitions.
    ↪ Conception des cellules et du circuit
  - Charge et décharge des nœuds (capacités) du circuit.
    ↪ Diminuer la taille des capacités et l'activité du circuit

Court-circuits dans les portes CMOS

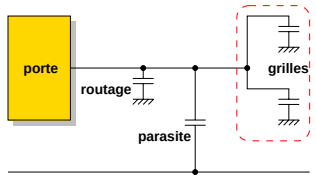
Lors d'une transition, les réseaux de transistors N et P sont passant simultanément pendant un bref instant.



Solution : faire des transitions les plus courtes possibles. . .

Charge et décharge des nœuds du circuit

La structure du circuit se comporte comme une multitude de capacités qu'il faut charger et décharger. Ces capacités sont partout : grilles des transistors, structures de routage, structures parasites. . .



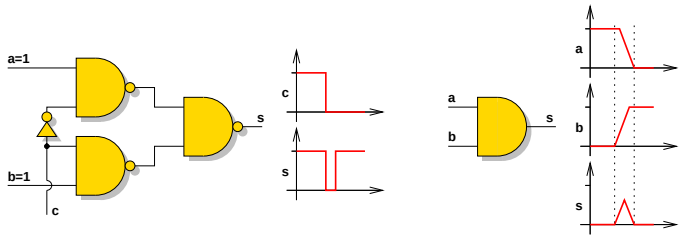
Solutions :

- faire des petits circuits
- diminuer l'activité (algorithmes, codage des données)
- diminuer V_{DD} (mais sans pénaliser la vitesse)

Les bonnes et les mauvaises transitions

Il y a deux types de transitions :

- les transitions porteuses d'information (nécessaires)
- les transitions redondantes ou parasites (dues aux imperfections temporelles)



Un peu de modélisation

Hypothèse : seule la consommation due aux charges et décharges des capacités est prise en compte (0 fuite, 0 court-circuit).

Puissance dynamique moyenne consommée par un circuit :

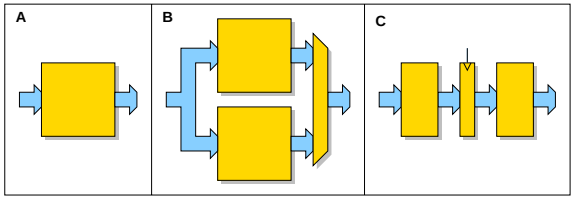
$$P = \alpha \times C \times f \times V_{DD}^2$$

où α est le facteur d'activité, C la capacité globale représentant le circuit et f la fréquence.

Remarque : délai d'une porte $d = \gamma \times \frac{C \times V_{DD}}{(V_{DD}-V_T)^2} \approx \frac{1}{V_{DD}}$.

Parallélisme

Comment diminuer la tension d'alimentation sans perdre en vitesse ? ? ?



solution	capacité	tension	fréquence	consommation
standard (A)	C	V	f	$P_A = CV^2f$
parallèle (B)	$2.2C$	$0.6V$	$0.5f$	$P_B = 2.2C(0.6V)^2 0.5f = 0.396P_A$
pipeline (C)	$1.2C$	$0.6V$	f	$P_C = 1.2C(0.6V)^2 f = 0.432P_A$

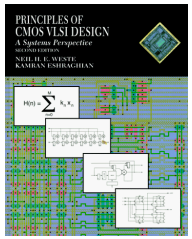
Exemple de caractéristiques de quelques portes [1]

porte	surface [μm^2]	capacité entrées [pF]	délais [ns] ↗ C_L	↗ $10C_L$	↘ C_L	↘ $10C_L$	conso. [μW/MHz]
INV 1X	36	0.007	0.12	0.70	0.09	0.53	0.23
INV 2X	55	0.010	0.10	0.63	0.09	0.51	0.47
INV 4X	73	0.021	0.09	0.62	0.08	0.50	0.83
BUF 1X	73	0.006	0.18	0.73	0.19	0.62	0.41
NAND2 1X	55	0.007	0.13	0.68	0.08	0.42	0.28
AND2 1X	73	0.004	0.22	0.77	0.27	0.72	0.42
NOR2 1X	55	0.009	0.14	0.66	0.15	0.58	0.39
OR2 1X	73	0.009	0.21	0.76	0.21	0.66	0.47
NAND3 1X	91	0.007	0.20	0.75	0.10	0.41	0.43
NOR3 1X	91	0.012	0.13	0.65	0.15	0.59	0.38
XOR2 1X	146	0.019	0.28	0.79	0.14	0.49	0.74
XOR3 1X	273	0.019	0.27	0.78	0.14	0.49	0.68
MUX 1X (A)	127	0.006	0.27	0.82	0.31	0.77	0.50
(C)		0.012	0.24	0.79	0.36	0.82	

[1] Données extraites de la bibliothèque de portes de AMS (austriamicrosystems) : 0.35μm, $V_{DD} = 3.3$V $T = 25°$C, $C_L = 0.015$pF.

Pour en savoir plus. . .

Principles of CMOS VLSI Design
A Systems Perspective
Neil H. E. Weste et Kamran Eshraghian
Seconde édition, octobre 1994
Addison-Wesley
ISBN : 0–201–53376–6



Partie 2

Addition

Points abordés dans cette partie

- Représentations des nombres (rappels)
- Cellules de base pour l'addition
- Additionneurs simples
- Additionneurs rapides
- Additionneurs redondants
- Additionneurs de plusieurs nombres

Représentations des nombres utilisées dans la suite

Nous allons essentiellement utiliser des entiers naturels représentés en numération simple de position[2] en base 2. On note n la taille des nombres utilisés.

$$A = a_{n-1}a_{n-2}\cdots a_1a_0 = \sum_{i=0}^{n-1} a_i 2^i$$

Pour les entiers relatifs, nous utiliserons soit la représentation en complément à deux, soit la représentation en signe et valeur absolue.

$$A = a_{n-1}a_{n-2}\cdots a_1a_0 = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$A = sa_{n-2}\cdots a_1a_0 = (-1)^s \sum_{i=0}^{n-2} a_i 2^i$$

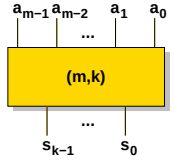
[2] C'est la représentation usuelle des nombres.

Cellules de base pour l'addition

En plus des portes logiques précédentes, nous allons utiliser des nouvelles portes présentant une propriété arithmétique bien utile : le comptage de 1.

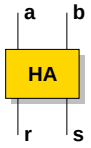
Un compteur (m,k) est une cellule, élémentaire ou non, qui compte le nombre de 1 présents sur ses m entrées et donne le résultat en numération simple de position sur k bits en sortie.

$$\sum_{i=0}^{m-1} a_i = \sum_{j=0}^{k-1} s_j 2^j$$



La cellule demi-additionneur (*half-adder* ou HA) est un compteur (2,2) tandis que la cellule d'addition complète (*full-adder* ou FA) est un compteur (3,2). Ces deux portes sont largement utilisées dans les opérateurs arithmétiques.

La cellule HA



a	b	r	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

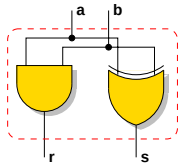
Équation arithmétique :

$$2r + s = a + b$$

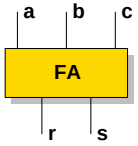
Équations logiques :

$$s = a \oplus b$$
$$r = ab$$

Réalisation pratique du HA :



La cellule FA



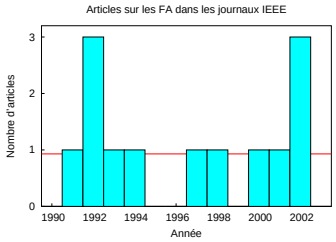
a	b	c	r	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Équation arithmétique :

$$2r + s = a + b + c$$

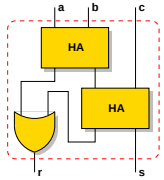
Équations logiques :

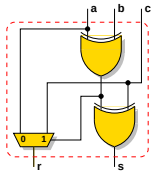
$$\begin{aligned} s &= a \oplus b \oplus c \\ r &= ab + ac + bc \end{aligned}$$

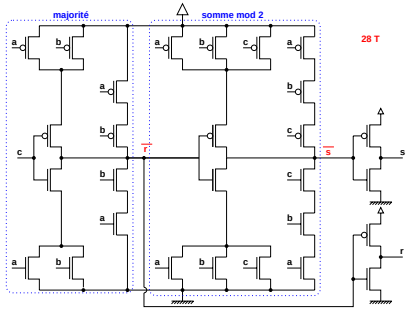


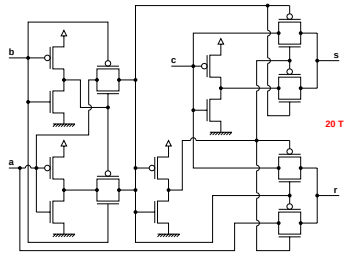
Il existe de nombreuses réalisations pratiques de la cellule FA.

Quelques implantations possibles de la cellule FA



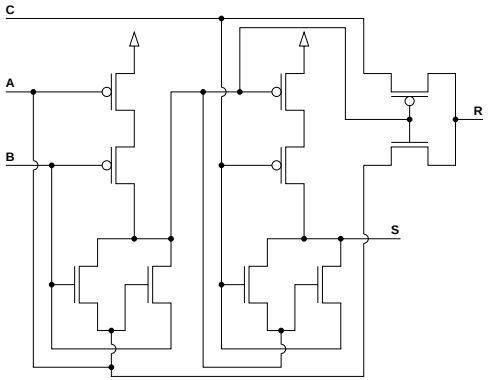






La cellule FA du jour

Solution en 10 transistors[3] :



A	B	C	R	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0_w	1
0	1	1	1	0
1	0	0	0_w	1
1	0	1	1_w	0
1	1	0	1	0
1	1	1	1_w	1_w

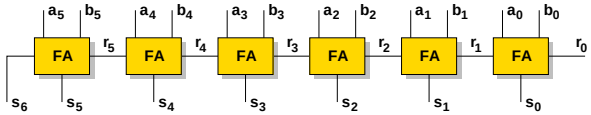
[3] H. T. Bui, Y. Wang et Y. Jiang. *Design and analysis of low-power 10-transistor full adders using novel XOR–XNOR gates*. IEEE Trans. CaS, jan. 2002.

Exemple de caractéristiques de portes pour l'arithmétique

porte	surface	capacité entrées	délais [ns]				conso.	chemin
			↗		↘			
	[μm^2]	[pF]	C_L	$10C_L$	C_L	$10C_L$	[μW/MHz]	
HA 1X	164	0.015	0.26	0.81	0.33	0.80	0.93	A → R
			0.35	0.90	0.35	0.79		A → S
		0.015	0.26	0.82	0.35	0.82		B → R
			0.28	0.83	0.35	0.79		B → S
FA 1X	364	0.030	0.39	0.96	0.45	0.95	1.28	A → R
			0.44	1.01	0.59	1.08		A → S
		0.028	0.40	0.97	0.42	0.94		B → R
			0.44	1.01	0.59	1.08		B → S
		0.025	0.34	0.90	0.38	0.89		C → R
			0.47	1.03	0.54	1.01		C → S
INV 1X	36	0.007	0.12	0.70	0.09	0.53	0.23	
NAND2 1X	55	0.007	0.13	0.68	0.08	0.42	0.28	
AND2 1X	73	0.004	0.22	0.77	0.27	0.72	0.42	
NAND3 1X	91	0.007	0.20	0.75	0.10	0.41	0.43	
XOR2 1X	146	0.019	0.28	0.79	0.14	0.49	0.74	
MUX 1X	127	0.006	0.27	0.82	0.31	0.77	0.50	A,B → S
		0.012	0.24	0.79	0.36	0.82		C → S

Additionneur séquentiel

C'est l'additionneur le plus simple. Il est composé de n cellules FA connectées en série.



Dans la littérature, on le trouve sous le nom de *Ripple-Carry Adder* (RCA) ou parfois de *Carry-Propagate Adder* (CPA)[4].

	complexité
délai	$O(n)$
surface	$O(n)$

[4] Attention : CPA peut aussi désigner un additionneur non redondant (propage mais ne conserve pas).

Propagation et génération de retenue

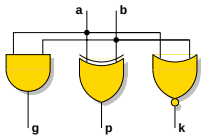
Autres façons de voir la table de vérité de la cellule FA :

a	b	r	s	remarque
0	0	0	c	génération
0	1	c	$\overline{c}$	propagation
1	0	c	$\overline{c}$	propagation
1	1	1	c	génération

a, b	r	s	remarque
$a = b$	a	c	génération
$a \neq b$	c	$\overline{c}$	propagation

Dans certaines références bibliographiques, on parle d'absorption (*kill*) dans le cas particulier de la génération d'une retenue sortante à 0 ($a = b = 0$).

$$\begin{aligned} p &= a \oplus b \\ g &= ab \\ k &= \overline{a}\overline{b} = \overline{a+b} \end{aligned}$$



Chaînes de retenues dans l'additionneur séquentiel

Du fait de la dépendance linéaire entre les cellules, il y a une grande variabilité du temps de calcul en fonction de la longueur de la plus grande chaîne de propagation de retenue.

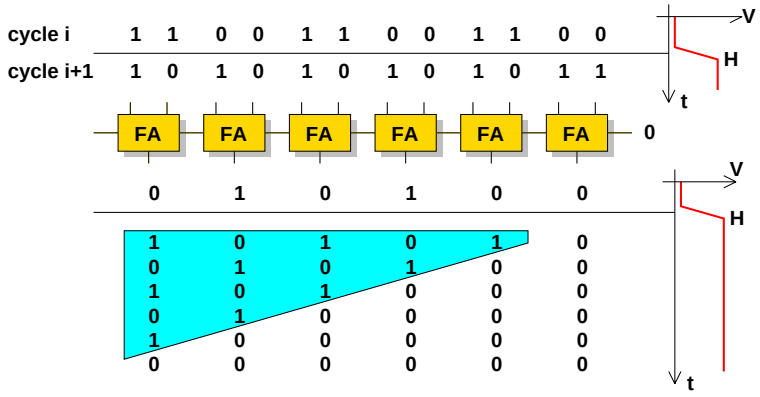
- pire cas en n propagations (ex : $000000 + 111111$)
- meilleur cas en 0 propagation (ex : $111111 + 111111$)
- cas moyen ?

En 1946, A. Burks, H. Goldstine et J. Von Neumann présentent dans "*preliminary discusion of the logical design of an electronic instrument*" l'étude du temps de calcul moyen d'un additionneur séquentiel.

Soit $L(n)$ la *longueur moyenne de la plus grande chaîne de propagation de retenue* dans un additionneur séquentiel de n bits avec une distribution uniforme et équiprobable des entrées. On a :

$$L(n) \leq \log_2 n$$

Activité d'un additionneur séquentiel



Activité pire cas en $n^2/2$, activité moyenne en $\frac{3n}{2}$ (seulement $\frac{n}{2}$ utile).

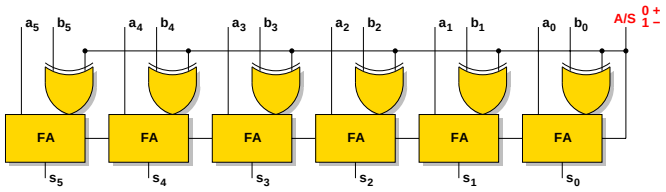
Soustraction

En complément à 2 on va faire :

$$A - B = A + (-B)$$

Pour avoir l'opposé de B :

$$\begin{aligned} B + \overline{B} &= \underbrace{111\ldots111}_{n \text{ bits}} = -2^{n-1} + \sum_{i=0}^{n-2} 2^i = -1 \\ -B &= \overline{B} + 1 \end{aligned}$$



Dépassements de capacité

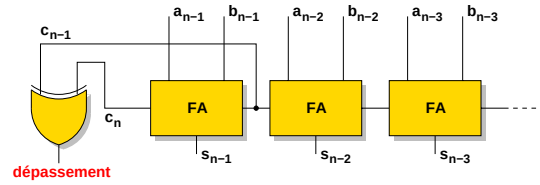
Il y a deux solutions :

- n bits $\pm$ n bits $\longrightarrow$ $n+1$ bits alors aucun risque de dépassement
- n bits $\pm$ n bits $\longrightarrow$ n bits et il faut tester les dépassements

Détection d'un dépassement de capacité en complément à 2 :

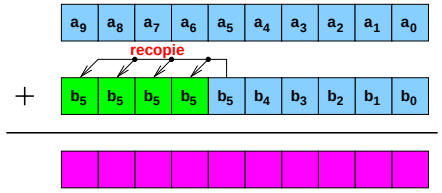
- Opérandes de signes opposés : aucun risque
- Opérandes de même signe : il y a dépassement ssi le signe du résultat est différent de celui des opérandes

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	−



Extension de signe

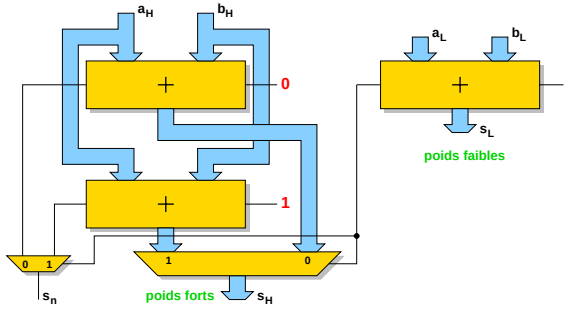
Lors de l'addition de nombres de tailles différentes en complément à 2, il faut faire une extension de signe.



Il faut faire attention au problème de sortance et à l'ordre des calculs lors de la somme de plus de 2 nombres.

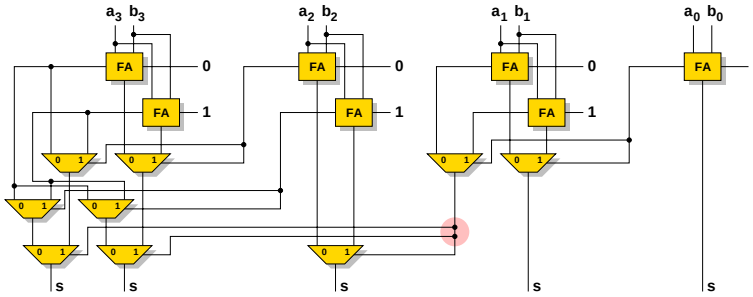
Additionneur à sélection de retenue

Idée : couper en deux et calculer le bloc des poids forts avec les deux retenues entrantes possibles et sélectionner la bonne sortie avec la retenue sortante des poids faibles (*Carry-Select Adder* ou *Conditional-Sum Adder*).



Version récursive $\longrightarrow$ délai en $O(\log n)$ mais. . .

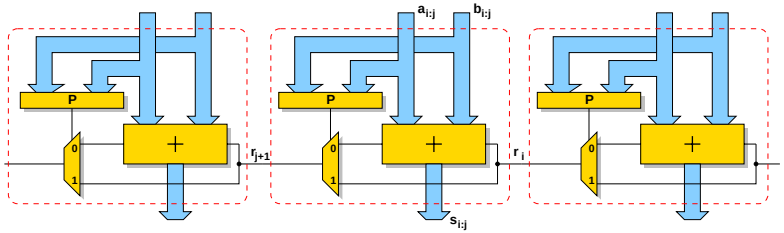
Il y a un petit problème de sortance



La sortance de certains nœuds n'est pas bornée. On peut ajouter des portes (inverseurs ou *buffers*) pour regénérer le signal mais cela augmente le délai car ces nœuds sont sur le chemin critique.

Additionneur à retenue bondissante

Idée : découper en blocs où chaque bloc permet de détecter rapidement une propagation sur tout le bloc (*Carry-Skip Adder*).



Questions :

- blocs de même taille ?
- taille optimale des blocs ?

Cas des blocs de même taille

Le pire cas d'un additionneur de n bits découpé en blocs de k bits se produit lorsque l'on doit propager du bit 1 jusqu'au bit $n-2$ (générations de retenues aux bits 0 et $n-1$ et propagation au milieu). Soit τ_1 le temps de propagation sur un 1 bit et τ_2 le temps de saut d'un bloc de k bits ($\tau_2 < \tau_1$).

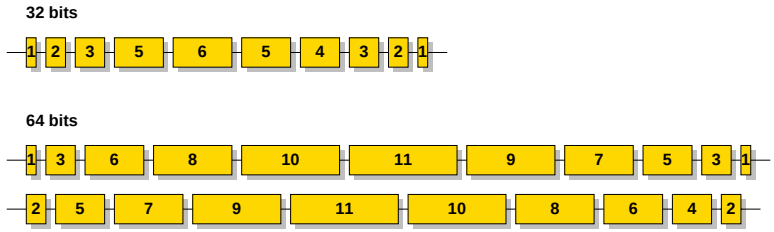
$$\begin{aligned} T(k) &= 2(k-1)\tau_1 + (\frac{n}{k} - 2)\tau_2 \\ T'(k) &= 2\tau_1 - \frac{n\tau_2}{k^2} \\ T''(k) &= \frac{2n\tau_2}{k^3} \end{aligned}$$

$$k_{opt} = \sqrt{\frac{n\tau_2}{2\tau_1}} \quad \longrightarrow \quad T(k_{opt}) = O(\sqrt{n})$$

Cas des blocs de tailles variables

Le problème est compliqué dans le cas général, mais de nombreuses solutions (heuristiques) ont été proposées pour les cas se présentant en pratique.

Exemple de solutions dans *A Simple Strategy for Optimized Design of One-Level Carry-Skip Adders*. M. Alioto et G. Palumbo. IEEE Trans. CaS I, jan. 03.



Additionneur à retenues anticipées

Jusqu'ici on a essayé de propager au plus vite les retenues. On va maintenant essayer de les calculer au plus vite.

L'étage de rang i reçoit une retenue entrante c_i égale à 1 dans les seuls cas suivants[5] :

- l'étage $i-1$ génère une retenue
  $\hookrightarrow g_{i-1} = 1$
- l'étage $i-1$ propage une retenue générée à l'étage $i-2$
  $\hookrightarrow p_{i-1} = g_{i-2} = 1$
- les étages $i-1$ et $i-2$ propagent une retenue générée à l'étage $i-3$
  $\hookrightarrow p_{i-1} = p_{i-2} = g_{i-3} = 1$
  ⋮
- les étages de $i-1$ à 0 propagent la retenue entrante c_0 à 1
  $\hookrightarrow p_{i-1} = p_{i-2} = \ldots = p_1 = p_0 = c_0 = 1$

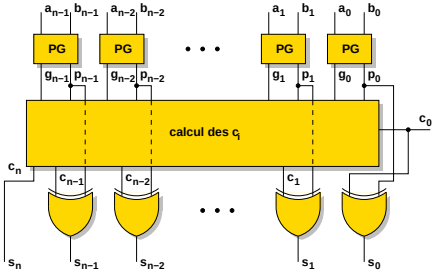
[5] Rappels : $p_i = a_i \oplus b_i$, $g_i = a_i b_i$ et $k_i = \overline{a_i + b_i}$.

On peut donc calculer les retenues avec la relation[6] suivante :

$$c_i = g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3} + \ldots + p_{i-1}\cdots p_1 g_0 + p_{i-1}\cdots p_0 c_0$$

Le principe des additionneurs à retenues anticipées (*Carry Look Ahead*) consiste à évaluer de manière totalement parallèle :

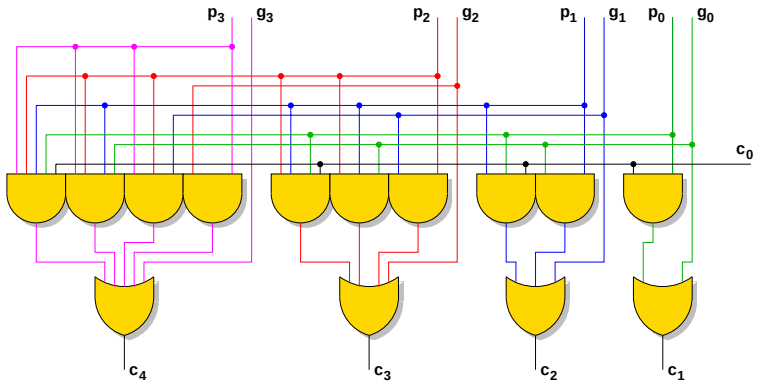
- les couples (g_i, p_i)
- les retenues c_i à l'aide de la relation ci-dessus
- les sommes $s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$



[6] qui se démontre par récurrence en utilisant $c_i = g_{i-1} + c_{i-1}p_{i-1}$.

Calcul des retenues pour un CLA 4 bits

$$c_1 = g_0 + p_0 c_0 \qquad c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0 \qquad c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$



Remarques sur les CLA

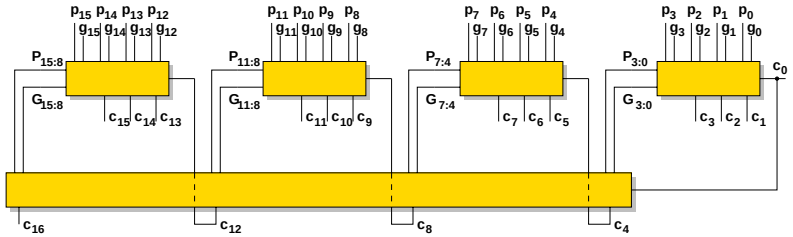
- Problème de sortance non bornée.
- Problème d'entrance[7] non bornée pour certaines portes.
- Additionneur théoriquement en temps constant mais logarithmique en pratique du fait de l'ajout de portes supplémentaires pour résoudre les problèmes d'entrance et de sortance.
- Les CLA à 4 bits servent souvent comme petites briques de base (élémentaires ou pas) pour concevoir de plus gros additionneurs (CLA ou autres).

[7] Nombre d'entrées limité dans une porte.

CLA cascadé

Le bloc de k bits, rangs i à j ($i > j$ et $i - j + 1 = k$), s'occupe des calculs :

- retenues c_i, c_{i-1}. . . à c_j
- propagation sur le bloc $P_{i:j} = p_i p_{i-1} \cdots p_j$
- génération dans le bloc $G_{i:j} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + p_i \cdots p_{j+1} g_j$

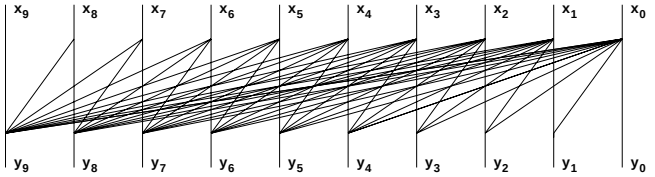


Délai en $O(log_k n)$ mais structure complexe.

Résolution de problèmes par préfixe parallèle

Les n sorties (y_{n-1}, y_{n-2},$\cdots$, y_0) sont calculées à partir des n entrées (x_{n-1}, x_{n-2},$\cdots$, x_0) en utilisant un opérateur associatif $\square$ avec :

$$\begin{aligned} y_0 &= x_0 \\ y_1 &= x_1 \square x_0 \\ y_2 &= x_2 \square x_1 \square x_0 \\ &\vdots \\ y_{n-1} &= x_{n-1} \square x_{n-2} \square \cdots \square x_1 \square x_0 \end{aligned}$$



Addition par préfixe parallèle

❶ Calcul des générations et propagations des entrées :

$$g_i = a_i b_i \quad \text{et} \quad p_i = a_i \oplus b_i \qquad \forall i = 0, 1, \ldots, n-1$$

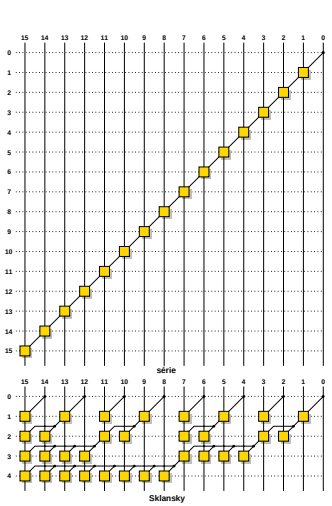
❷ Calcul des retenues c_i par préfixe parallèle à m niveaux :

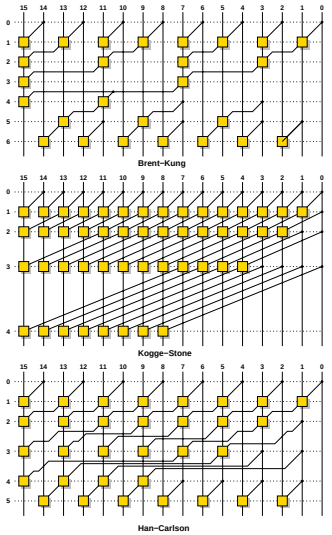
$$\begin{aligned} (G^0_{i:i}, P^0_{i:i}) &= (g_i, p_i) \\ (G^l_{i:k}, P^l_{i:k}) &= (G^{l-1}_{i:j}, P^{l-1}_{i:j}) \square (G^{l-1}_{j:k}, P^{l-1}_{j:k}) \qquad k \le j \le i \text{ et } l = 1, \ldots, m \\ &= (G^{l-1}_{i:j} + P^{l-1}_{i:j} G^{l-1}_{j:k}, P^{l-1}_{i:j} P^{l-1}_{j:k}) \\ c_{i+1} &= G^m_{i:0} + P^m_{i:0} c_0 \qquad \forall i = 0, 1, \ldots, n-1 \end{aligned}$$

❸ Calcul des sommes :

$$s_i = p_i \oplus c_i \qquad \forall i = 0, 1, \ldots, n-1$$

Quelques additionneurs à préfixe parallèle



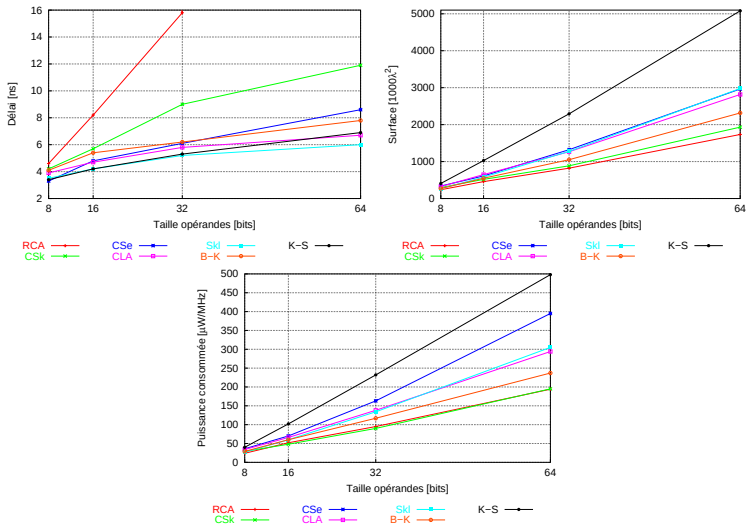


Caractéristiques des additionneurs à préfixe parallèle

Valeurs de la partie calcul des retenues c_i seule (étape ❷, il faut ajouter en plus les cellules des étapes ❶ et ❸). $T_\square$ et $S_\square$ sont respectivement le délai et la surface en nombre de $\square$. HT est le nombre de lignes de routage horizontales et FO la sortance maximale.

additionneur	$T_\square$	$S_\square$	HT	FO
série	$n-1$	$n-1$	1	2
Sklansky	$\log n$	$\frac{1}{2}n\log n$	$\log n$	$\frac{1}{2}n$
Brent–Kung	$2\log n-2$	$2n-\log n-2$	$2\log n-1$	$\log n$
Kogge–Stone	$\log n$	$n\log n-n+1$	$n-1$	2
Han–Carlson	$\log n+1$	$\frac{1}{2}n\log n$	$\frac{1}{2}n+1$	2

Comparaison générale



Additionneurs redondants

Pour aller encore plus vite, on va "tricher" en conservant les retenues. Cela n'a un sens que si l'on effectue plusieurs additions successivement.

En 1961, Avizienis a suggéré de représenter les nombres en base β par l'ensemble de chiffres $\{-\alpha,-\alpha+1,\ldots,0,\ldots,\alpha-1,\alpha\}$ au lieu des chiffres de $\{0,1,2,\ldots,\beta-1\}$ avec $\alpha\leq\beta-1$.

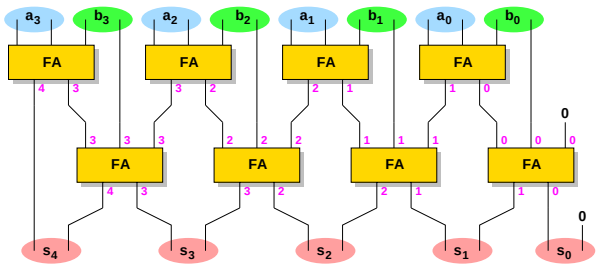
Dans ce système, si $2\alpha+1>\beta$ certains nombres ont plusieurs écritures possibles. Par exemple, le nombre 2345 (dans le système usuel) peut s'écrire en base 10 avec l'ensemble de chiffres $\{-5,-4,-3,-2,-1,0,1,2,3,4,5\}$ selon les codages 2345, 235(-5) ou 24(-5)(-5). On dit alors que le système est redondant.

L'intérêt des systèmes de représentation redondants est qu'il existe dans ces systèmes des algorithmes permettant d'effectuer des additions de façon totalement parallèle, c'est-à-dire sans propagation de retenues.

Additionneur *carry-save*

On représente le nombre A en base 2 avec les chiffres $a_i\in\{0,1,2\}$ sur deux fils tels que $a_i=a_{i,c}+a_{i,s}$ où $a_{i,c}\in\{0,1\}$ et $a_{i,s}\in\{0,1\}$.

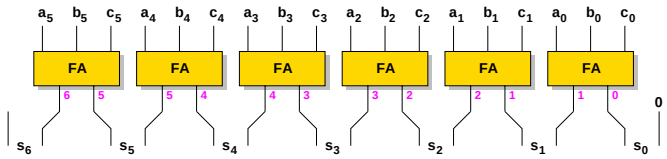
$$A=\sum_{i=0}^{n-1}a_i2^i=\sum_{i=0}^{n-1}(a_{i,c}+a_{i,s})2^i$$



Toutes les sommes sont obtenues dans le délai de 2 cellules FA.

Addition *carry-save* de $k \geq 3$ nombres standards

Soient A, B et C trois nombres en notation non-redondante. On obtient leur somme S en représentation *carry-save* avec l'opérateur suivant ($k = 3$).



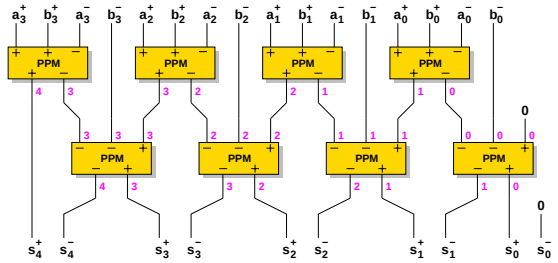
Plus généralement, un arbre *carry-save* à h niveaux permet de réduire au plus $n(h)$ entrées non-redondantes selon le tableau ci-dessous. On a $n(h) = \lfloor 3n(h-1)/2 \rfloor$ et $n(0) = 2$.

h	1	2	3	4	5	6	7	8	9	10	11
$n(h)$	3	4	6	9	13	19	28	42	63	94	141

Additionneur *borrow-save*

On représente le nombre A en base 2 avec les chiffres $a_i \in \{-1, 0, 1\}$ sur deux fils tels $a_i = a_i^+ - a_i^-$ où $a_i^+ \in \{0, 1\}$ et $a_i^- \in \{0, 1\}$.

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



Toutes les sommes sont obtenues dans le délai de 2 cellules PPM.

Cellule PPM

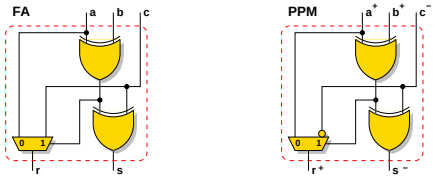
a^+	b^+	c^-	r^+	s^-
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Équation arithmétique :

$$2r^+ - s^- = a^+ + b^+ - c^-$$

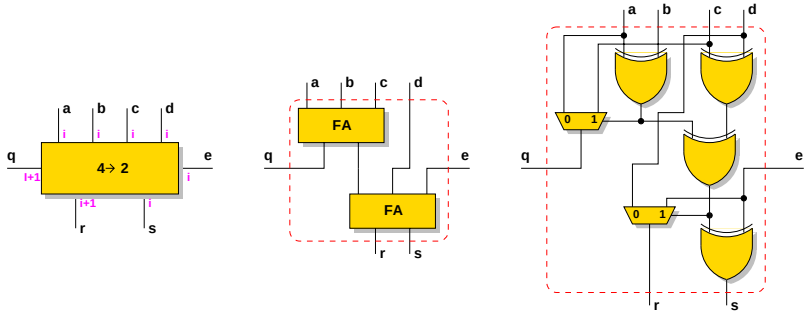
Équations logiques :

$$\begin{aligned} s &= a^+ \oplus b^+ \oplus c^- \\ r &= a^+ b^+ + a^+ \overline{c^-} + b^+ \overline{c^-} \end{aligned}$$



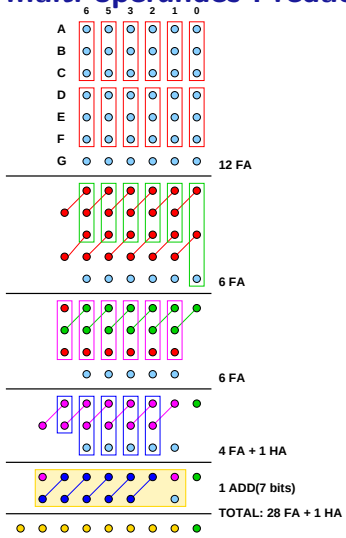
Cellule 4 donne 2

Equation arithmétique : $a + b + c + d + e = 2(r + q) + s$

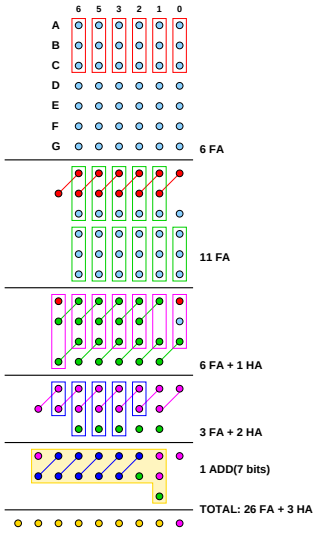


Intérêt : faire des arbres plus réguliers qu'avec la cellule FA (3 donne 2).

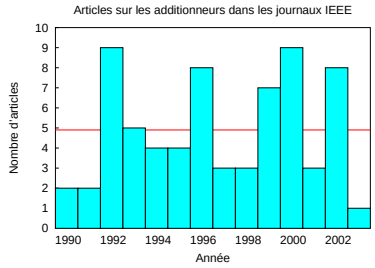
Additionneurs multi-opérandes : réduction en points



Et en utilisant la retenue entrante de l'additionneur final



La recherche sur les additionneurs



- 1994 : NEC, CMOS 0.4 μm et 3.3 V
  ALU basée sur un CLA 32 bits à 500 MHz.
- 2002 : Intel, CMOS 0.13 μm et 1.5 V
  ALU basée sur un Han-Carlson 32 bits à 5 GHz.

Pour en savoir plus. . .

Binary Adder Architectures for Cell-Based VLSI and their Synthesis

Reto Zimmermann

1998
Hartung–Gorre
ISBN : 3–89649–289–6

Arithmétique des ordinateurs

Jean–Michel Muller

1989
Masson
ISBN : 2–225–81689–1

Partie 3

Multiplication

Points abordés dans cette partie

- Multiplication par additions–décalages
- Recodage de Booth
- Multiplieurs cellulaires
- Multiplieurs arborescents
- Opérateurs dérivés de la multiplication

Multiplication par additions–décalages

Calculons le produit $P = A \times B$ avec des additions de deux nombres et des décalages à l'aide de l'algorithme (série–parallèle) suivant :

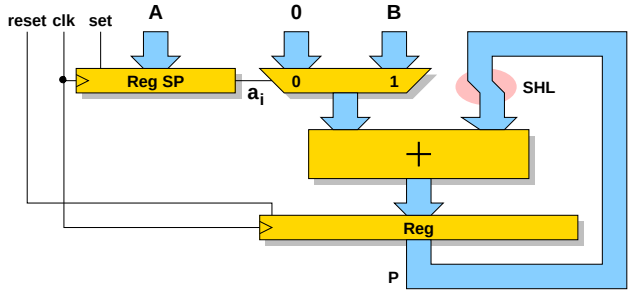
```
1  P ⟵ 0
2  for i from 0 to n − 1 do
3      P ⟵ P + a_i B 2^i
```

Utiliser directement cet algorithme nécessite un décaleur variable. On va donc plutôt utiliser une version modifiée de cet algorithme :

```
1  P ⟵ 0
2  for i from 0 to n − 1 do
3      P ⟵ (P + a_i B) × 2^−1
4  P ⟵ P 2^n
```

L'opération de la ligne 4 est virtuelle.

Implantation matérielle du multiplieur par additions–décalages



	complexité
délai	$O(n)$
surface	$O(n)$

Multiplieur ou multiplicateur ?

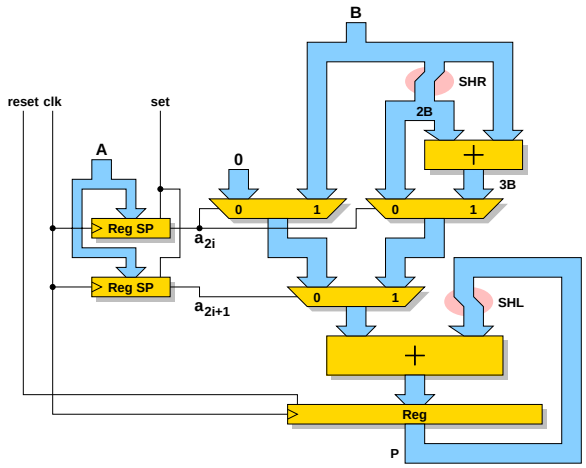
Dans “Le petit Larousse 2003” on trouve :

MULTIPLIEUR n.m. INFORM. Organe d'un calculateur analogique ou numérique permettant d'effectuer le produit de deux nombres.

MULTIPLICATEUR n.m. ARITHM. Nombre par lequel on multiplie un autre appelé *multiplicande*. (Dans 3 fois 4, égal à $4+4+4$, 3 est le multiplicateur.)

Accélération de la multiplication série–parallèle

Idée : utiliser des chiffres du multiplicateur dans une base plus grande.



Recodage de Booth

Idée de Booth en 1951 : augmenter le nombre de 0 dans le multiplicateur en le recodant avec l'ensemble de chiffres signés $\{-1=\overline{1},0,1\}$.

Utilisation de l'identité : $2^{i+k}+2^{i+k-1}+2^{i+k-2}+\cdots+2^i = 2^{i+k+1}-2^i$

Exemple : le nombre 60 peut s'écrire $00111100 = 0100\overline{1}00$.

Cette méthode permet de transformer les chaînes de 1 par une écriture avec plus de 0.

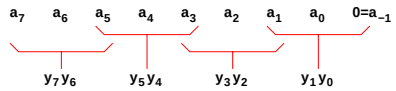
Mais dans certains cas on peut faire apparaître plus de 1 dans la chaîne recodée que dans la chaîne initiale !

Exemple : la chaîne 01010101 se recode par Booth en $\overline{1}1\overline{1}1\overline{1}1\overline{1}1$.

$\Longrightarrow$ utilisation du recodage de Booth modifié

Recodage de Booth modifié

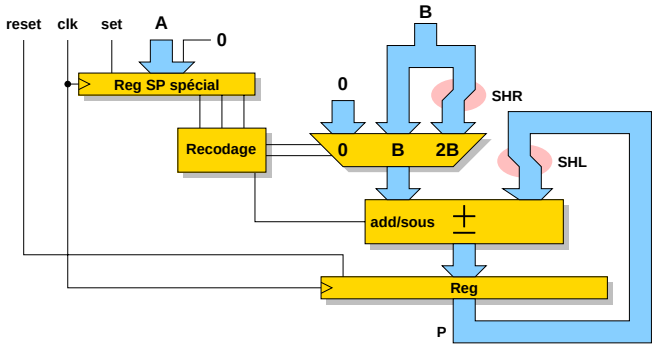
Idée : ne pas recoder les 1 isolés mais seulement les chaînes de 1.



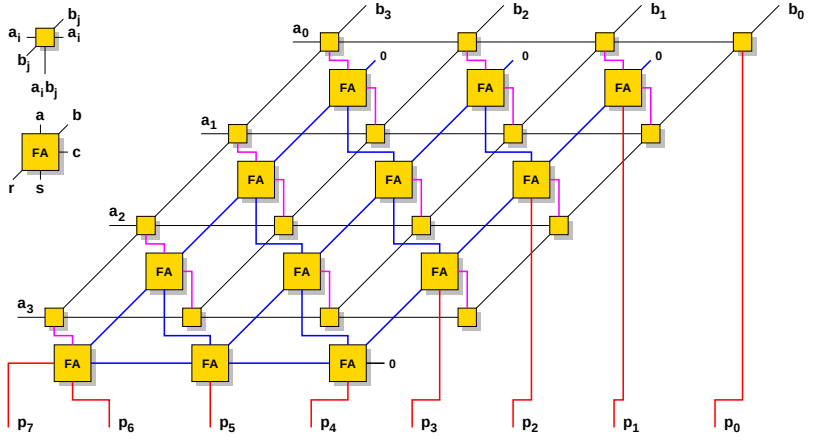
a_i	a_{i-1}	a_{i-2}	y_i	y_{i-1}	signification	opération
0	0	0	0	0	chaîne de 0	$+0$
0	0	1	0	1	fin chaîne de 1	$+B$
0	1	0	0	1	1 isolé	$+B$
0	1	1	1	0	fin chaîne de 1	$+2B$
1	0	0	$\overline{1}$	0	début chaîne de 1	$-2B$
1	0	1	1	$\overline{1}$	0 isolé	$-B$
1	1	0	0	$\overline{1}$	début chaîne de 1	$-B$
1	1	1	0	0	milieu chaîne de 1	$+0$

Avantage : faire un produit de 2 nombres de n bits en $\lfloor n/2 \rfloor + 1$ additions–décalages au plus.

Multiplieur série–parallèle avec recodage modifié de Booth



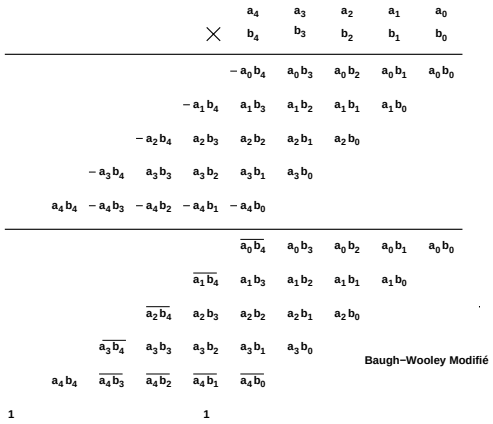
Multiplieur cellulaire de Braun



Temps de calcul en $O(n)$.

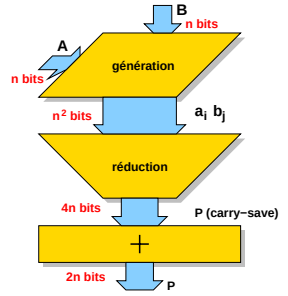
Produits de 2 nombres en complément à 2

Si A et B sont en complément à 2 alors certains produits partiels ont des poids négatifs.



Multiplieurs arborescents

1. Génération parallèle des produits partiels $a_i b_j$ avec ou sans recodage de Booth
   $\hookrightarrow$ temps en $O(1)$ (sortance a_i,b_j $O(\log n)$)
2. Calcul de la somme en *carry-save* des produits partiels à l'aide d'un arbre de réduction
   $\hookrightarrow$ temps en $O(\log n)$
3. Addition finale des 2 vecteurs de bits de la somme *carry-save* avec un additionneur rapide
   $\hookrightarrow$ temps en $O(\log n)$



On a donc une structure permettant la multiplication en un temps de l'ordre de $O(\log n)$.

Génération des produits partiels

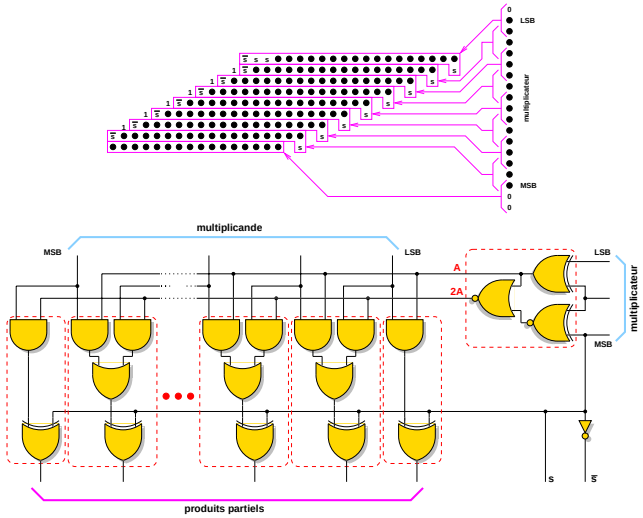
Dans le produit $P = A \times B$ où A et B sont des entiers de n bits, la génération des produits partiels nécessite n^2 cellules AND (ou $n^2 - 2n - 2$ AND et $2n - 2$ NAND dans le cas d'entrées en complément à deux).

Tous les produits partiels $a_i b_j$ peuvent être obtenus en même temps modulo le problème de sortance sur les entrées a_i et b_j.

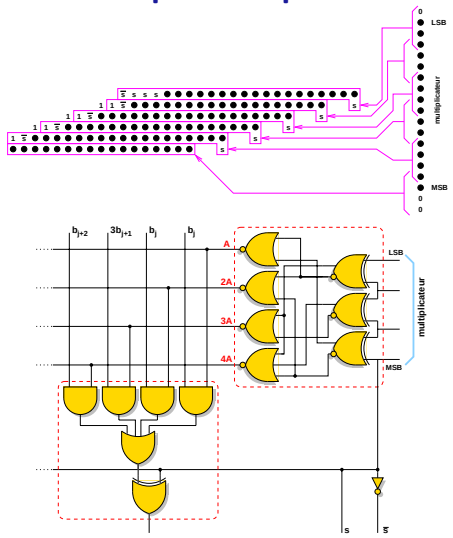
On peut utiliser un recodage de Booth modifié pour diminuer le nombre de produits partiels. Les solutions les plus courantes en terme de recodage de Booth modifié dans les multiplieurs arborescents sont :

- Booth 2 : recodage des a_i en base 4 avec l'ensemble de chiffres $\{0, \pm 1, \pm 2\}$.
- Booth 3 : recodage des a_i en base 8 avec l'ensemble de chiffres $\{0, \pm 1, \pm 2, \pm 3, \pm 4\}$.

Génération des produits partiels : Booth 2



Génération des produits partiels : Booth 3



Gain du recodage de Booth

Cas du recodage de Booth 2 :

- Vitesse : le gain d'un étage ou deux sur l'arbre de réduction des produits partiels est plus ou moins compensé par l'étage de recodage et génération des produits partiels beaucoup plus complexe.
- Surface : véritable gain (30 %) car on utilise les cellules de recodage pour régénérer le signal pour éviter les problèmes de sortance (chose qu'il faut faire sans le recodage avec des *buffers* supplémentaires) et les cellules de générations se réalisent bien en CMOS.

Cas du recodage de Booth 3 :

Rarement utilisé en pratique car les étages de génération des produits partiels deviennent beaucoup trop complexes (nombreuses portes et dont certaines avec 3 ou 4 entrées).

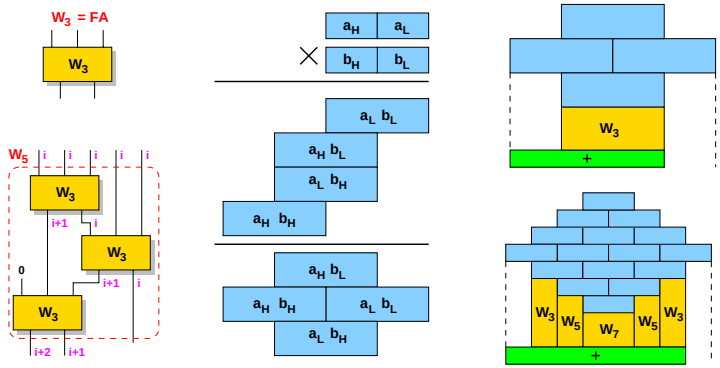
Arbres de réduction des produits partiels

Il faut maintenant calculer la somme *carry-save* des $n/2+1$ produits partiels. Pour cela, différents types d'arbres de réduction sont possibles :

- À base de cellules FA :
  - arbres de Wallace
  - arbres de Dadda
  - algorithmes de réduction rapide
- À base de cellules "4 donne 2"
- À base de "grands" compteurs

Arbres de Wallace

Les arbres de Wallace[8] sont des compteurs à p entrées ($\lceil log_2 p \rceil$ sorties). Un arbre de Wallace à $2^{p+1}-1$ se construit facilement à partir d'arbres de Wallace à 2^p-1 entrées (un arbre de Wallace à 3 entrées est une cellule FA).

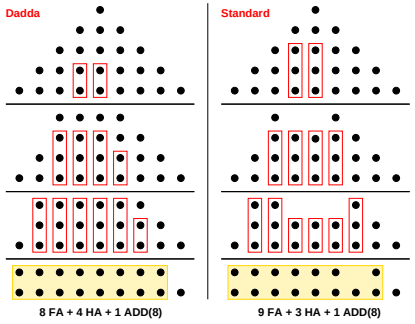


[8] C.S. Wallace. *A suggestion for a fast multiplier.* IEEE Transactions on Computers, février 1964.

Arbres de Dadda

Idée : faire le minimum de réduction à chaque étape pour gagner un niveau dans la suite $n(h) = \lfloor 3n(h-1)/2 \rfloor$ et $n(0) = 2$.

h	1	2	3	4	5	6	7	8	9	10	11
$n(h)$	3	4	6	9	13	19	28	42	63	94	141



Gain en surface pour les grands multiplieurs. Exemple : $n = 12$ bits $\Rightarrow$ 11% de portes en moins qu'avec un arbre de Wallace.

Algorithmes pour la génération automatique d'arbres de réduction optimisés

Il existe des généralisations et améliorations de la méthode de Dadda pour différents types de compteurs et utilisant des modèles de délai complexes (ex : bits de retenue plus rapides que ceux de somme).

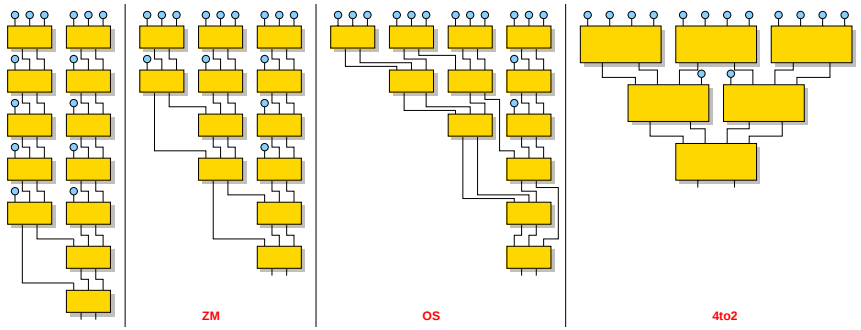
Par exemple, la méthode TDM[9] permet de gagner jusqu'a 30% en vitesse et 15% en surface par rapport à une solution utilisant un arbre de cellules "4 donne 2".

La génération automatique de multiplieurs est un problème très compliqué et fait intervenir différents aspects (souvent antagonistes) : vitesse, surface, régularité topologique, consommation d'énergie, besoins en cellules spéciales. . .

[9] V. Oklobdzija, D. Villeger et S. Liu. *A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach.*IEEE Transactions on Computers, mars 1996.

Problèmes topologiques au niveau cellules

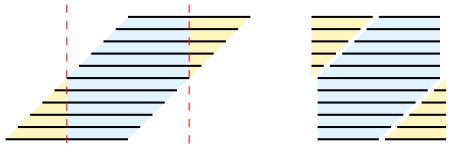
Quelle est la topologie la plus efficace et la plus simple à gérer ?



Réduction de 14 bits (sans mentionner les retenues latérales).

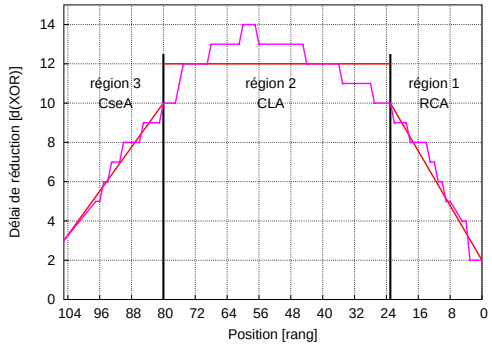
Problème de *layout*

On doit essayer de faire des circuits carrés.

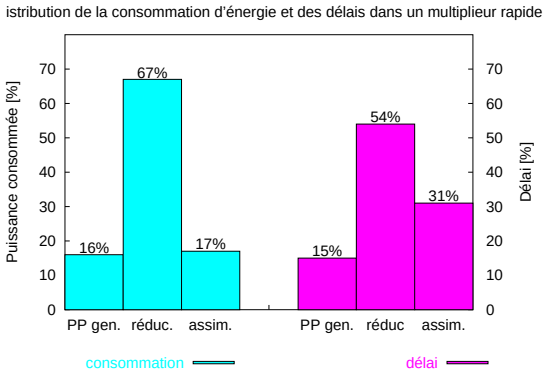


Addition finale pour multiplieurs

L'addition finale dans un multiplieur arborescent (assimilation des retenues) peut se faire avec un additionneur rapide. Mais de petites optimisations sont possibles en utilisant les temps d'arrivées relatifs des bits de la somme *carry-save* après réduction (adaptation du type d'additionneur suivant les régions).



Répartition de la consommation dans un multiplieur rapide

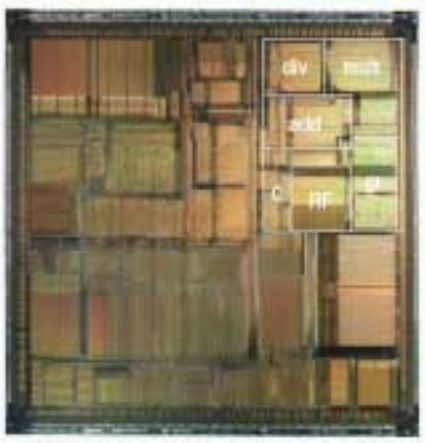


- 30% à 70% de transitions redondantes
- faire le placement et le routage avec les temps d'arrivées internes
- étage de *pipeline* à la "bonne" place

Exemple : multiplieur flottant de l'UltraSPARC

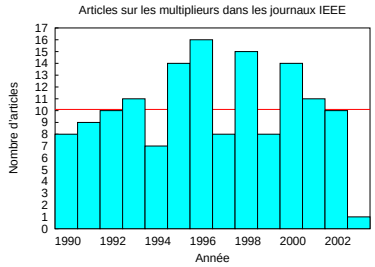
Multiplieur 53×53 bits :

- latence 3 cycles
  1. PP + réduction
  2. addition final
  3. arrondi
- débit $1\times$ par cycle
- recodage Booth 2
- cellule spéciale "4 donne 2"
- addition finale par *conditional-sum adder*
- 167 MHz, CMOS 0.5 μm



Données tirées des deux articles sur les unités flottantes de l'Ultra dans ARITH 12 (1995).

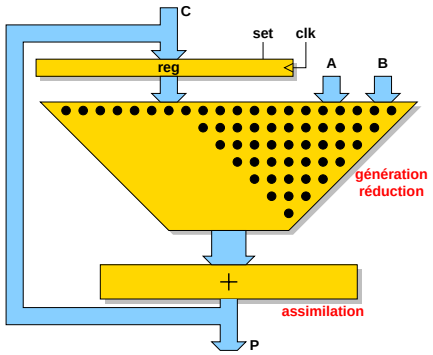
La recherche sur les multiplieurs



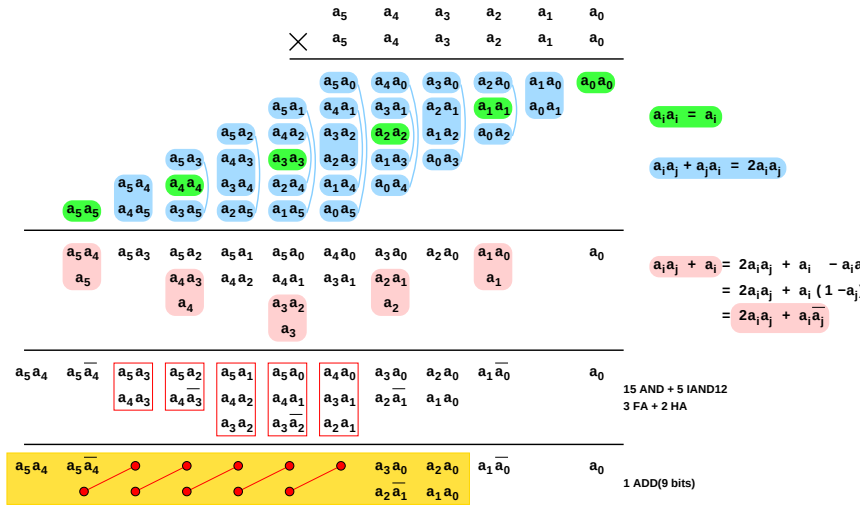
- 1991 : Toshiba, CMOS 0.5 μm et 3.3 V
  multiplieur 54 $\times$ 54 bits à 100 MHz, Booth 2, cellules "4 donne 2", addition CLA + CSeA.
- 2001 : Mitsubishi, CMOS 0.18 μm et 1.8 V
  multiplieur 54 $\times$ 54 bits à 600 MHz (2 cycles), Booth 2, cellules "4 donne 2", sans addition finale (produit en *carry-save*).

Multiplieur–accumulateur (MAC)

Calcul de $P(t) = A \times B + P(t-1)$. En général, A et B sur n bits et P sur m bits avec $m >> n$ ($16 \times 16 + 40 \longrightarrow 40$ sur certains DSP).



Calcul du carré par un opérateur dédié

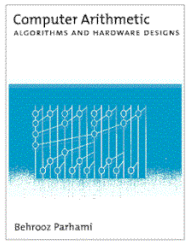


Pour en savoir plus. . .

Computer Arithmetic
Algorithms and Hardware Designs

Behrooz Parhami

2000
Oxford University Press
ISBN : 0–19–512583–5



Advanced Computer
Arithmetic Design

Micheal Flynn et Stuart Obermann

2001
Wiley–Interscience
ISBN : 0–471–41209–0

Partie 4

Division

Points abordés dans cette partie

- Division simple par additions–décalages
- Division SRT
- Méthode de Newton
- Extensions au calcul de la racine carrée

Division à la main

On cherche à trouver le quotient q de la division du dividende x par le diviseur d (le reste est r). On a : $x = q \times d + r$.

Exemple : calcul de $123/234$, (résultat exact $\frac{123}{234} = 0.5256410256\ldots$).

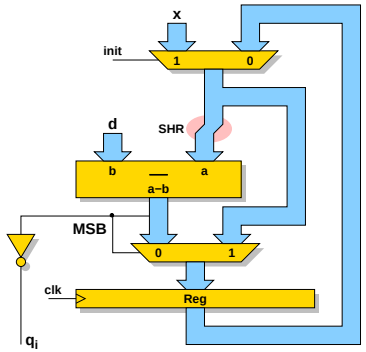
1	2	3	0			2	3	4
	6	0	0					
	1	3	2	0		. 5	2	5	6	4	···
		1	5	0	0			
				9	6	0		
					2	4		

1	x	234	=	234
2	x	234	=	468
3	x	234	=	702
4	x	234	=	936
5	x	234	=	1170
6	x	234	=	1404
7	x	234	=	1638
8	x	234	=	1872
9	x	234	=	2106
10	x	234	=	2340

On effectue donc l'itération : $x^{(i+1)} = 10x^{(i)} - q_{i+1}d$

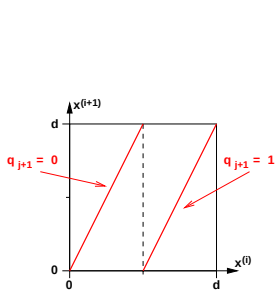
Division restaurante

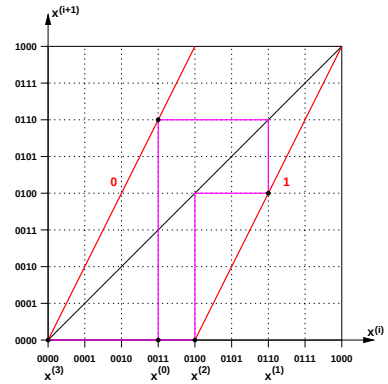
```
1  for i from 1 to n do
2      x ⟵ 2x
3      x ⟵ x − d
4      if x ≥ 0 then
5          q_i ⟵ 1
6      else
7          q_i ⟵ 0
8          x ⟵ x + y
```



Le caractère *restaurant* de cet algorithme vient de l'annulation de l'effet de la ligne 3 par la ligne 8 dans certains cas.

Diagramme de Robertson de la division restaurante





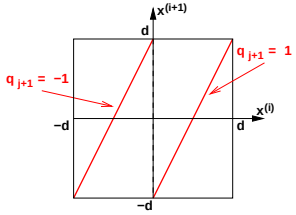
x = 11
d = 1000

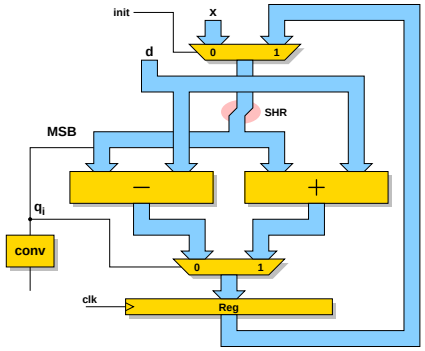
$x^{(0)} = 0011$	$q_0 = 0$
$x^{(1)} = 0110$	$q_1 = 1$
$x^{(2)} = 0100$	$q_2 = 1$
$x^{(3)} = 0000$	$q_3 = 0$

$\frac{3}{8} = 0.375$

Division non restaurante

```
1  for i from 1 to n do
2      x ⟵ 2x
3      if x ≥ 0 then
4          x ⟵ x − y
5          q_i ⟵ 1
6      else
7          x ⟵ x + y
8          q_i ⟵ −1
```



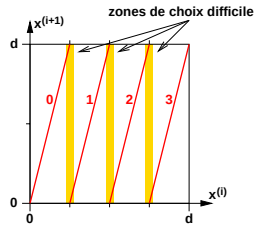


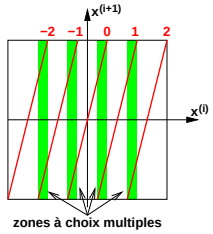
La conversion de $\{-1, 1\}$ vers $\{0, 1\}$ peut se faire à la volée (MSDF) avec un petit opérateur.

Comment aller plus vite ?

Idée : faire des itérations avec des q_i dans une base plus grande.

Problème : faire des comparaisons précises avec plusieurs multiples du diviseur.





Solution : utiliser une représentation redondante pour le quotient.
↪ faire des comparaisons approchées

Division SRT

La méthode SRT proposée par Sweeney, Robertson et Tocher en 1958 est basée sur :

- une représentation redondante des chiffres du quotient en base β (pour simplifier le choix des q_i).
- une représentation redondante des restes partiels (pour accélérer la soustraction, en *borrow-save* par exemple).
- une table permettant de déduire q_{i+1} à partir de quelques bits de poids forts de d et de $x^{(i)}$ (après conversion en non-redondant).

```
1  d' ⟵ trunc(d, k_d, MSB)
2  for i from 1 to n/log2 β do
3      x'^(i) ⟵ trunc(x^(i), k_x, MSB)
4      q_{i+1} ⟵ T(d', conv(x'^(i)))
5      x^(i+1) ⟵ β x^(i) − q_{i+1} × d
```


Architecture générale d'un diviseur SRT

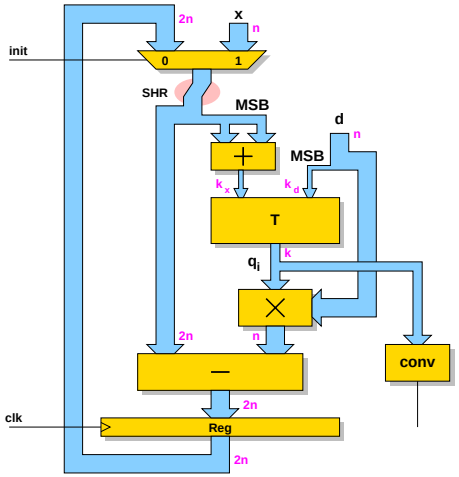
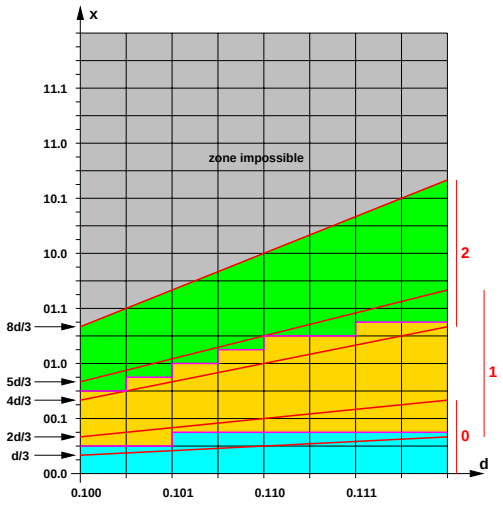


Table pour un diviseur SRT

Diagramme reste-diviseur :

- base $\beta = 4$
- $q_i \in \{-2, -1, 0, 1, 2\}$
- entrée :
  - d sur 3 bits
  - x sur 5 bits

Remarque : Le diagramme est symétrique par rapport à l'axe d.



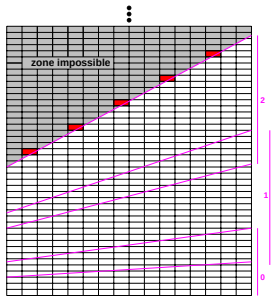
Le bug de la division du Pentium[10]

Exemples de problèmes en 1994 :

$$\frac{4195835.0}{3145727.0} = 1.333739068902\ldots \quad \text{sur le pentium}$$
$$= 1.333820449136\ldots \quad \text{exact}$$

Diviseur du Pentium : SRT base 4, chiffres du quotient $\{-2, -1, 0, 1, 2\}$, restes partiels en *carry-save*, table avec 5 bits d'entrée pour d et 7 bits pour $x^{(i)}$.

Problème : 5 cases fausses dans la table



[10] RR 95-06 de J.-M. Muller sur ce sujet à `http://www.ens-lyon.fr/LIP/`.

Extensions à la racine carrée

On peut calculer des racines carrées avec un algorithme à additions-décalages proche de celui pour la divison.

Exemple : on cherche $r = \sqrt{c}$ avec $x^{(0)} = c - 1$ et

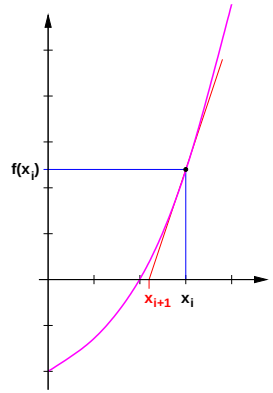
```
1 for i from 1 to n/log2 β do
2     x'^(i) ⟵ trunc(x^(i), k_x, MSB)
3     r'^(i) ⟵ trunc(r^(i), k_r, MSB)
4     r_{i+1} ⟵ T(conv(r'^(i)), conv(x'^(i)))
5     r^(i+1) ⟵ r^(i) + r_{i+1} β^{-i-1}
6     x^(i+1) ⟵ β x^(i) − 2 r^(i) r_{i+1} − r_{i+1}^2 β^{-i-1}
```


Méthode de Newton

On cherche une racine de l'équation $f(x) = 0$ (où f est supposée continument dérivable) en utilisant la suite (x_i) définie par :

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Si x_0 est suffisament proche d'une racine simple α de f, alors la suite (x_i) converge quadratiquement vers α (le nombre de chiffres significatifs double à chaque étape).



$f(x) = \frac{x^2}{2} - 2$

i	0	1	2	3	4
x_i	3	2.166666667	2.006410256	2.000010240	2.000000000

Méthode de Newton pour la division

Pour trouver le quotient $q = a/d$ on va procéder en 2 étapes :

1. calcul de $t = 1/d$ à l'aide de la fonction $f(x) = \frac{1}{x} - d$. On doit donc calculer l'itération suivante :

$$\begin{aligned} x_{i+1} &= x_i - \frac{\frac{1}{x_i} - d}{-\frac{1}{{x_i}^2}} \\ &= x_i + x_i - d{x_i}^2 \\ &= x_i(2 - {x_i}^2) \end{aligned}$$

   Le coût de chaque itération est de 2 multiplications et 1 addition.

   La valeur x_0 est obtenue par une lecture dans une petite table.

2. calcul de $q = t \times a$

Exemple de l'Itanium

Utilisation du multiplieur–accumulateur flottant sur des registres de 82 bits. Initialisation de la méthode de Newton par une lecture de table qui donne une approximation de $1/d$ à 8.886 bits près.

Exemple d'algorithme pour la simple précision (mantisse de 23 bits) :

```
1 y_0 ⟵ T(d)
2 q_0 ⟵ (a × y_0)_rn
3 e_0 ⟵ (1 − b × y_0)_rn
4 q_1 ⟵ (q_0 + e_0 × q_0)_rn
5 e_1 ⟵ (e_0 × e_0)_rn
6 q_2 ⟵ (q_1 + e_1 q_1)_rn
7 e_2 ⟵ (e_1 × e_1)_rn
8 q_3 ⟵ (q_2 + e_2 × q_2)_rn
9 q'_3 ⟵ round(q_3)
```

Méthode de Newton pour la racine carrée

Première idée : utiliser Newton avec $f(x) = x^2 - c$, on a alors :

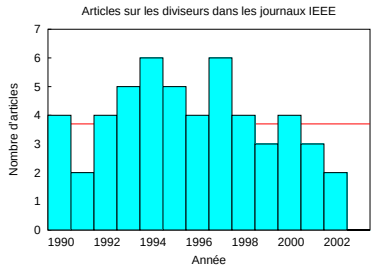
$$x_{i+1} = x_i - \frac{{x_i}^2 - c}{2x_i} = x_i - \frac{x_i}{2} + \frac{c}{2x_i} = \frac{1}{2}\left(x_i + \frac{c}{x_i}\right)$$

Seconde idée : utiliser Newton avec la fonction $f(x) = \frac{1}{x^2} - c$ qui a pour solution $\frac{1}{\sqrt{c}}$ (ensuite on multiplie par c pour avoir $\sqrt{c}$).

$$x_{i+1} = x_i - \frac{\frac{1}{x^2} - c}{-\frac{2}{{x_i}^3}} = x_i + \frac{x_i - c{x_i}^3}{2} = \frac{x_i}{2}\left(3 - c{x_i}^2\right)$$

Chaque itération fait intervenir 3 multiplications et une addition.

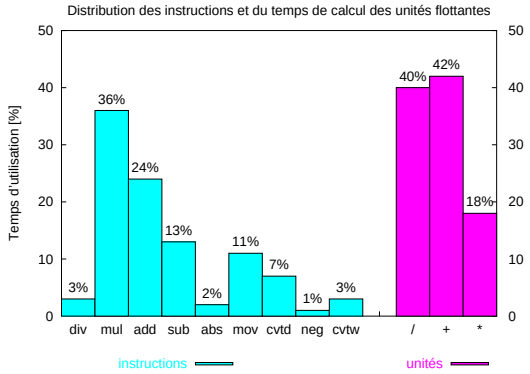
La recherche sur les diviseurs



- SRT base 4 dans le Pentium ou SRT base 8 dans l'Ultra (3 étages de base 2 par itération).
- Newton dans Itanium, Pentium 4 (table pour initialisation + routine logicielle).

La division, une opération peu utilisée ? Oui, mais. . .

Rapport technique[11] de S. Oberman et M. Flynn : "*Design issues in floating-point division*", CSL-TR-94-647 Stanford University.



[11] SPECfp92 sur DECstation avec un MIPS R3000 (latences : 2c add, 5c mul, 19c div), compil. O3.

Pour en savoir plus. . .

Division and Square Root : Digit-Recurrence Algorithms and Implementations

Milos Ercegovac et Tomas Lang

1994

Kluwer

ISBN : 0–7923–9438–0

Division Algorithms and Implementations

Stuart Oberman et Micheal Flynn

1997

IEEE Transactions on Computers

Vol. 46, No. 8, pp 833–854

Fin

Questions ?

Pour me contacter :

- arnaud.tisserand@ens-lyon.fr
- http://www.ens-lyon.fr/~atissera/ (va changer)
- Laboratoire LIP. ENS Lyon. 46 allée d'Italie. F-69364 Lyon cedex 07.

Merci.