Opérateurs arithmétiques matériels
Additionneurs rapides

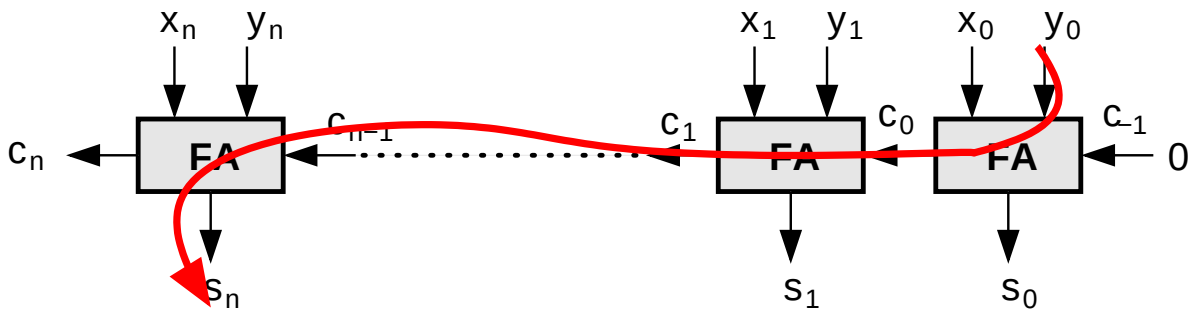
Florent de Dinechin

18 octobre 2005

Intro

J'espère que vous avez compris à ce stade

... que c'est la propagation de retenue qui est lente.



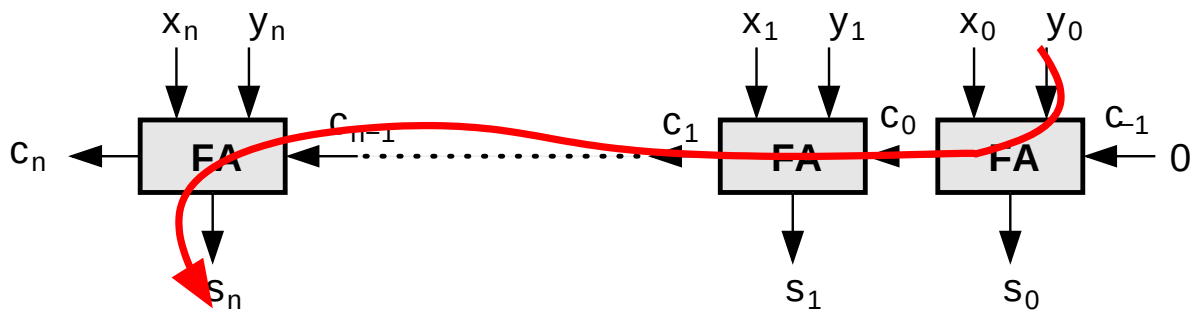
Quelques méthodes d'accélération :

- Soigner le matériel qui propage les retenues
- Construire les retenues par un genre d'arbre
- Essayer de passer du pire cas au cas moyen
- Changer de système de numération

Briques de bases

Le *full adder*

La brique de base principale qui permet de faire les quatre opérations...



- addition de trois nombres binaires a, b et c_{in} ;
- somme entre 0 et 3, exprimée sur deux bits : c_{out} (poids fort) et s (poids faible) ;
- $s = (a \oplus b) \oplus c_{in}$
- $c_{out} = a.b \ + \ c_{in}.(a \oplus b)$
- aussi appelé compteur *3-en-2* ;

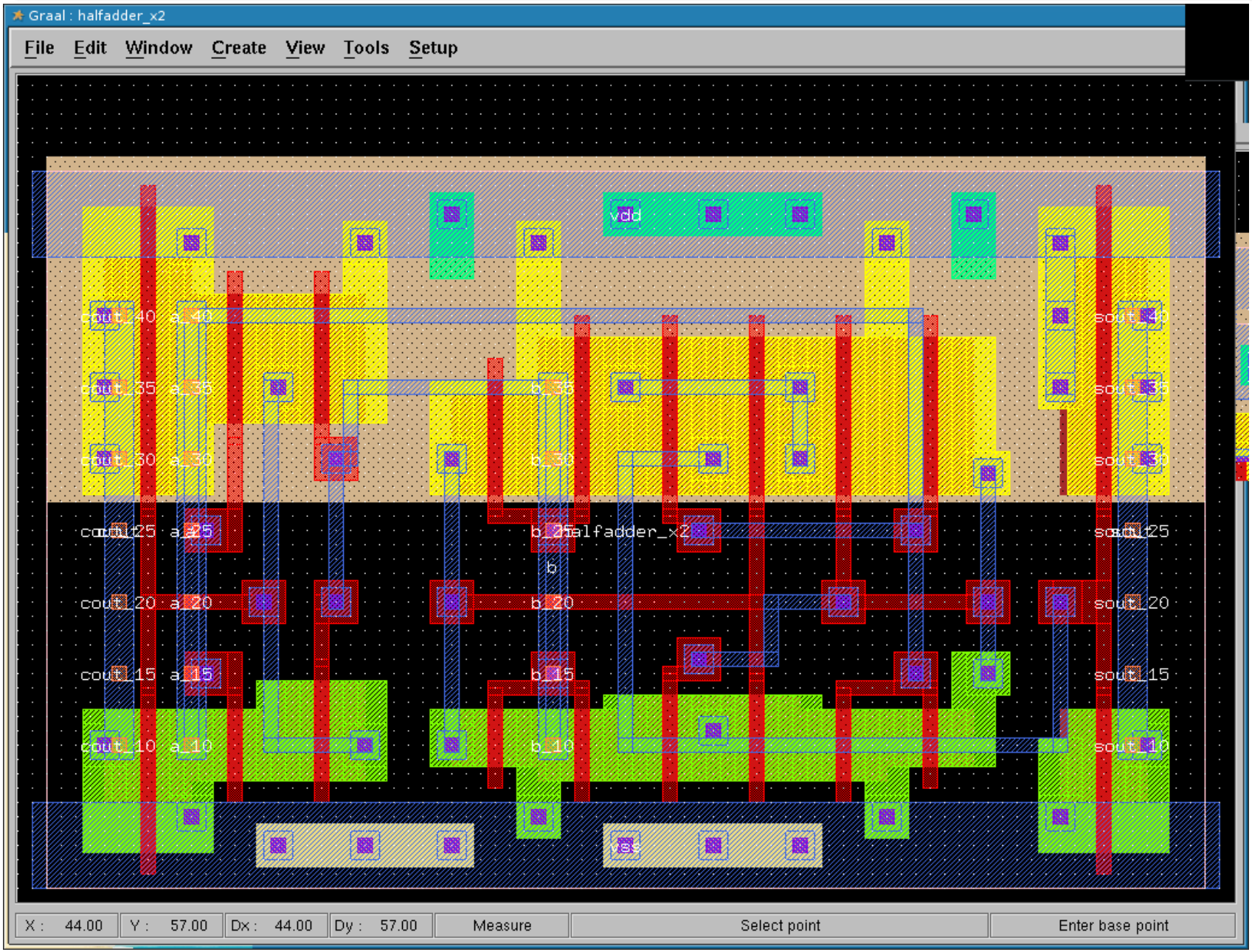
Le *half adder*

C'est juste quand on n'a besoin que de deux entrées

- addition de deux nombres binaires a et b ;
- somme entre 0 et 2, exprimée sur deux bits : c_{out} (poids fort) et s (poids faible) ;
- $s = a \oplus b$
- $c_{out} = a.b$
- il est tellement plus simple qu'on arrive à le reconnaitre dans Alliance...

Utilité : surtout les arbres, pas uniquement le bit zéro du CPA.
On verra...

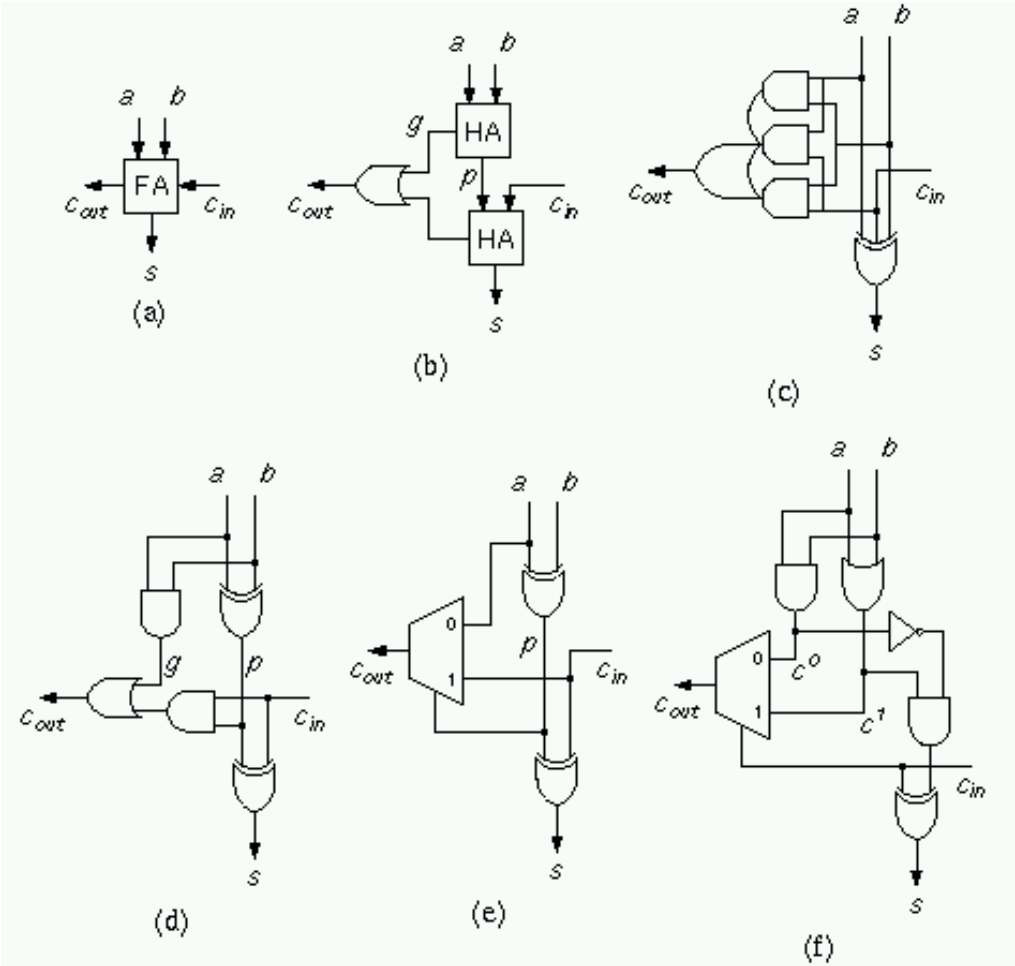
Un *half adder* en transistors



Revenons au *full adder*

- avec des portes
- avec des transistors

Avec des portes



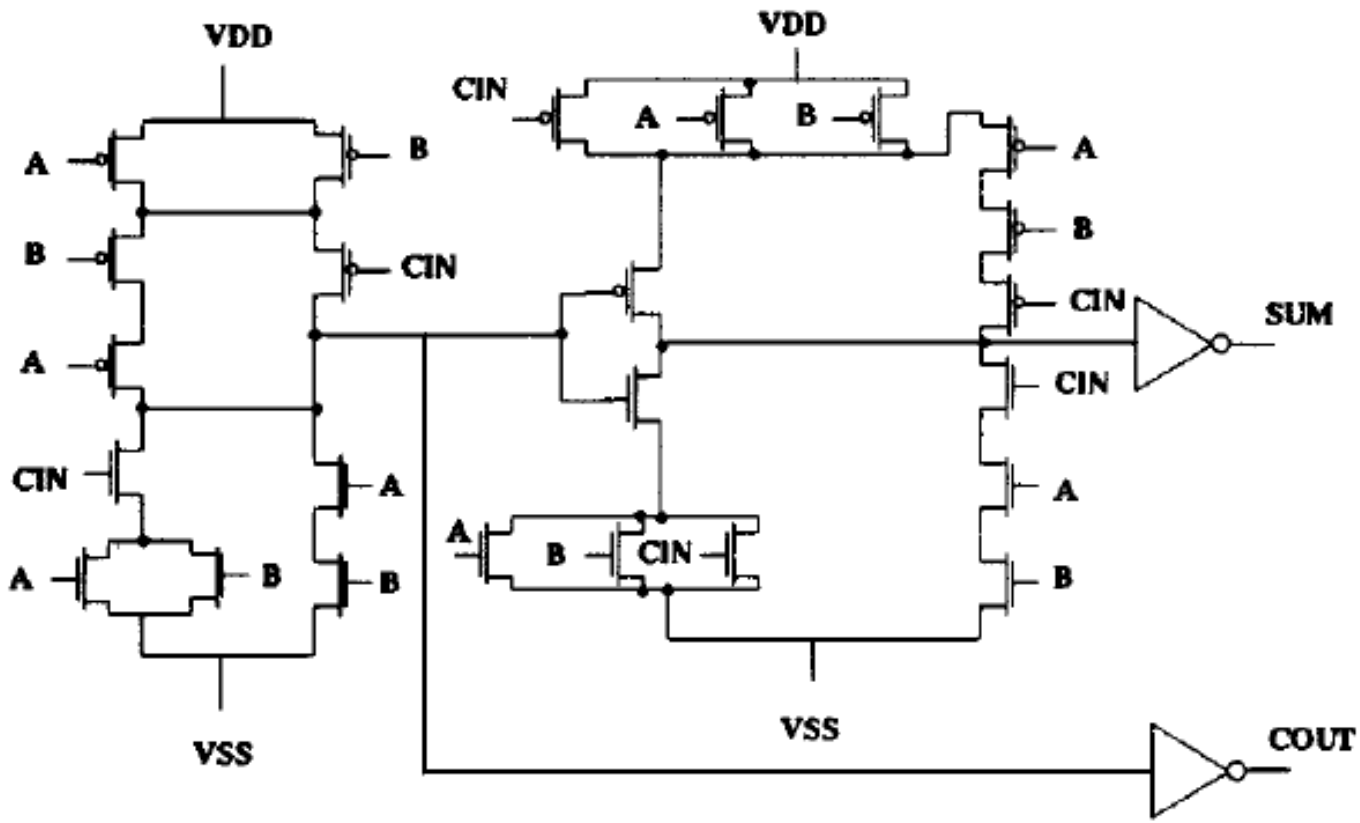
Choix entre ces solutions : (d)(e)(f) / (c)

- $T(a, b \rightarrow c_{out}) = 4/2$
- $T(c_{in} \rightarrow c_{out}) = 2$
- $T(a, b \rightarrow s) = 4$
- $T(c_{in} \rightarrow s) = 2/4$
- $A = 7/9$ portes

Avec des transistors

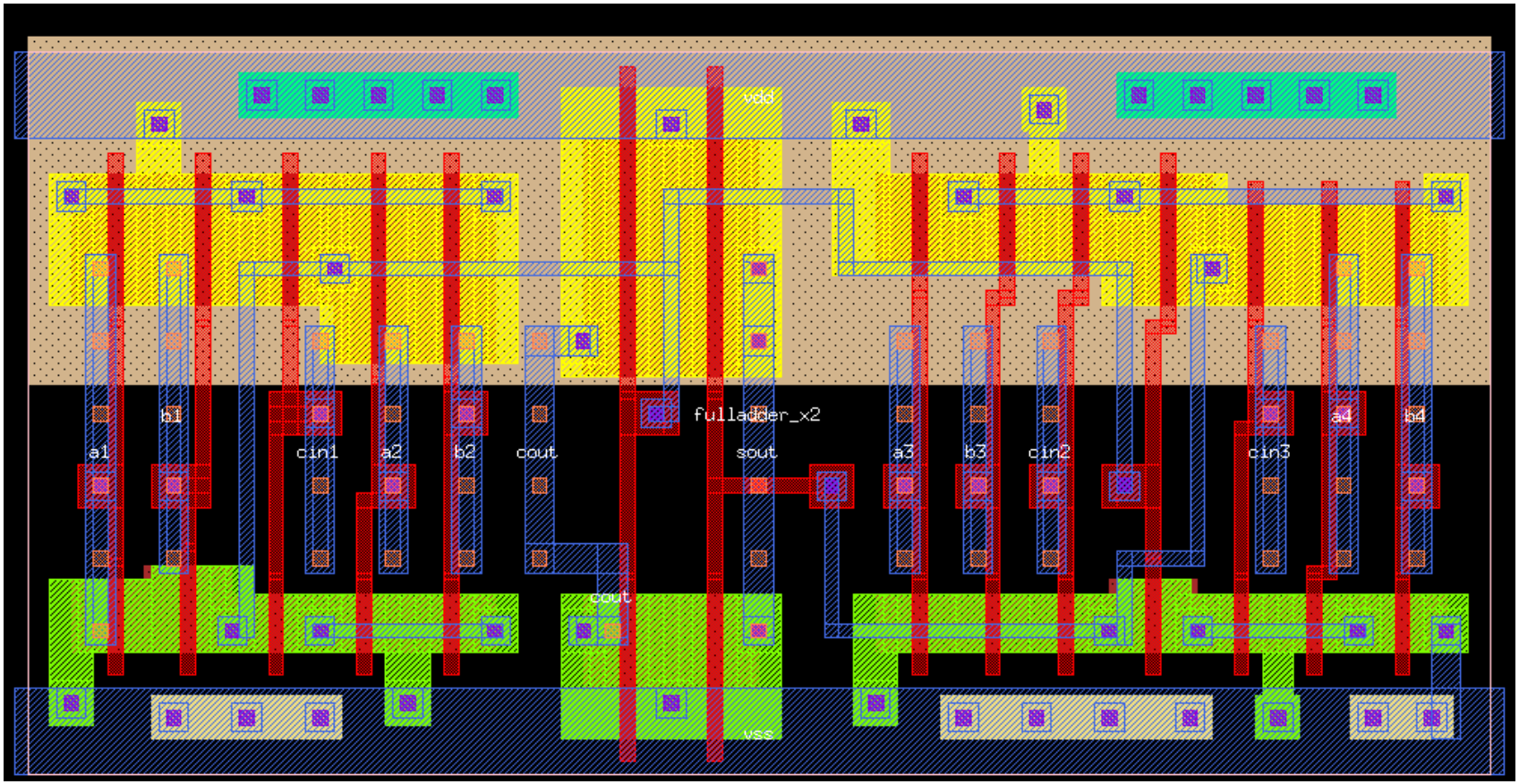
... Il y a des dizaines de papiers, des centaines de circuits

CMOS bien propre

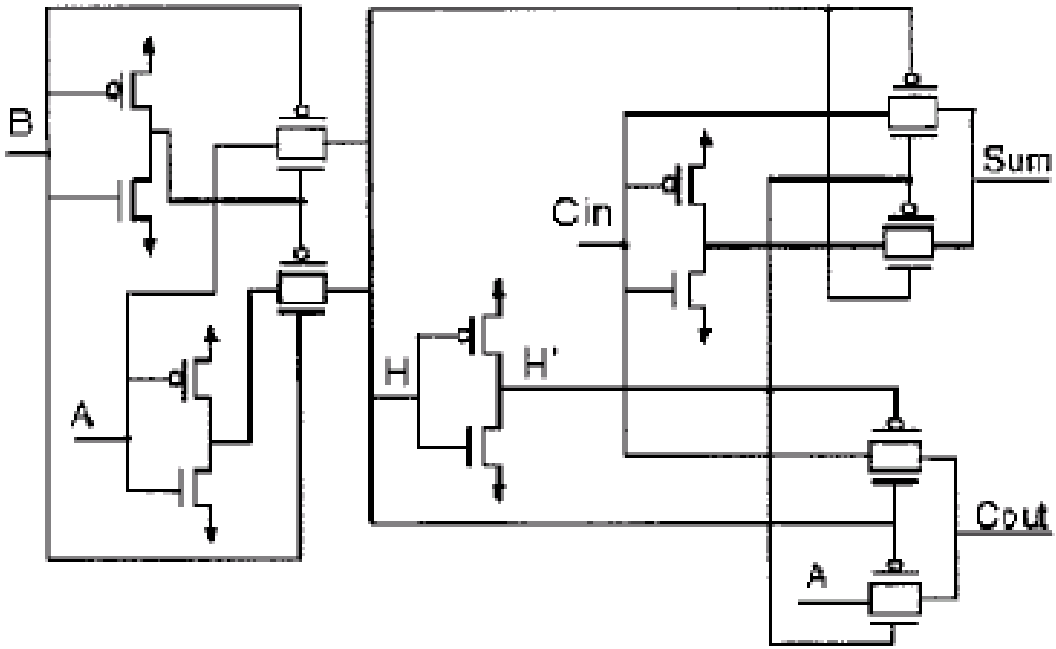


- Un NXor à gauche, le reste à droite
- 28 transistors
- Le signal de sortie est très joli
- Voyons celui de la bibliothèque standard d'Alliance

Un *full adder* en transistors



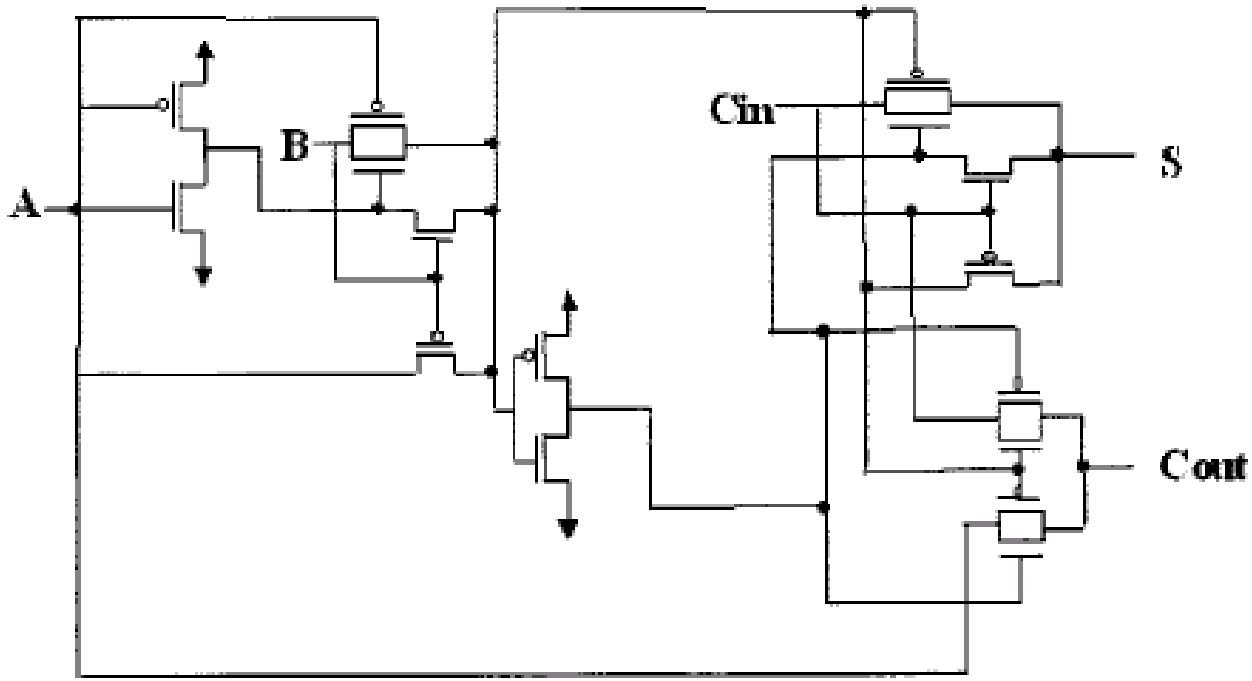
Circuit classique en transistors



$$
\begin{aligned}
H &= A \oplus B \\
S &= H \oplus C_{in} \\
C_{out} &= \text{si } H \text{ alors } C_{in} \text{ sinon } A
\end{aligned}
$$

- utilise des portes de transmission
- 20 transistors, 3 inverseurs
- Pouvez-vous faire un CPA avec ?

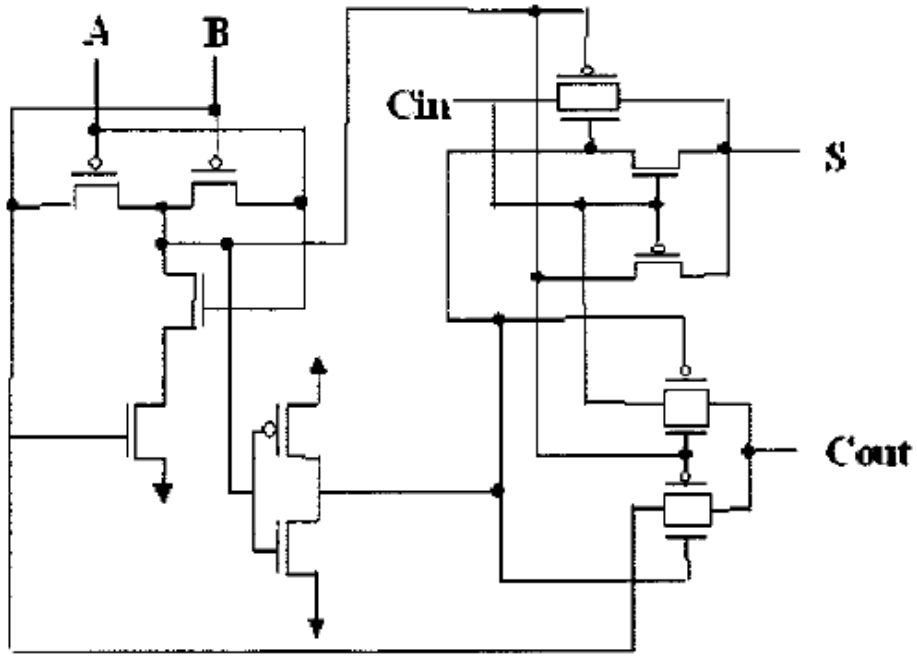
Constructions classiques (2)



- déduit de la théorie des fonctions de transmission ;
- 16 transistors, 2 inverseurs ;
- de savantes études ont montré qu'il était aussi plus rapide.

N. Zhuang and H. Wu, A new design of the CMOS full adder, *IEEE J. Solid-State Circuits*, vol. 27, pp. 840 844, May 1992.

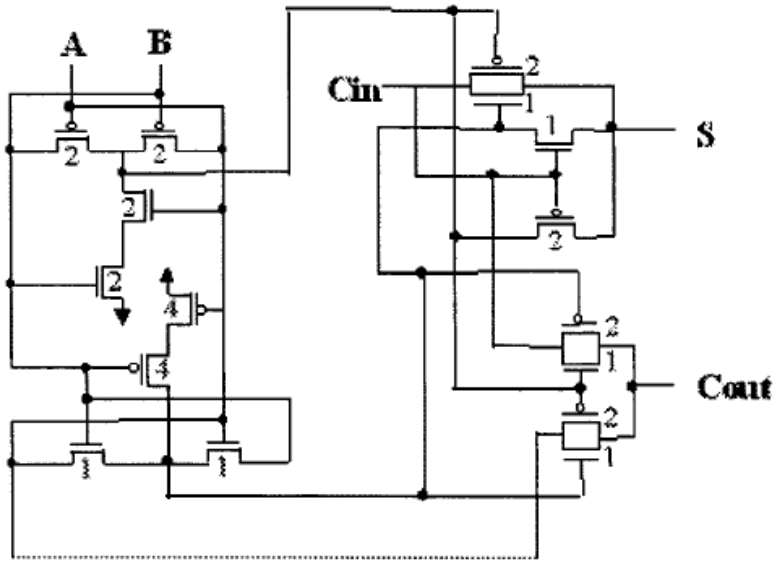
Encore moins de transistors



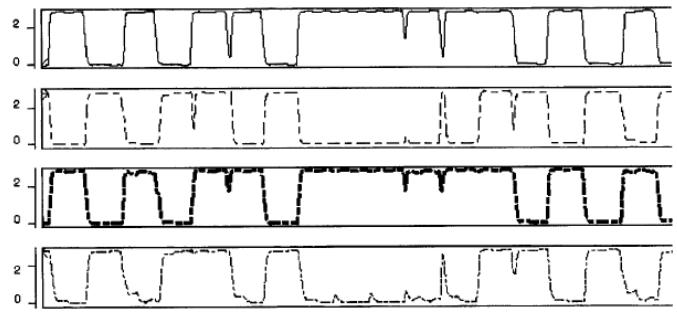
- pas très propre, tout cela (MOS mais plus CMOS) ;
- 14 transistors, 1 inverseur, mais il fuit un peut lorsque $A = B = 0$;
- faible consommation malgré tout.

E. Abu-Shama and M. Bayoumi, A new cell for low power adders, in *Proc. Int. Midwest Symp. Circuits and Systems*, 1995, pp. 1014 1017.

Plus mais mieux



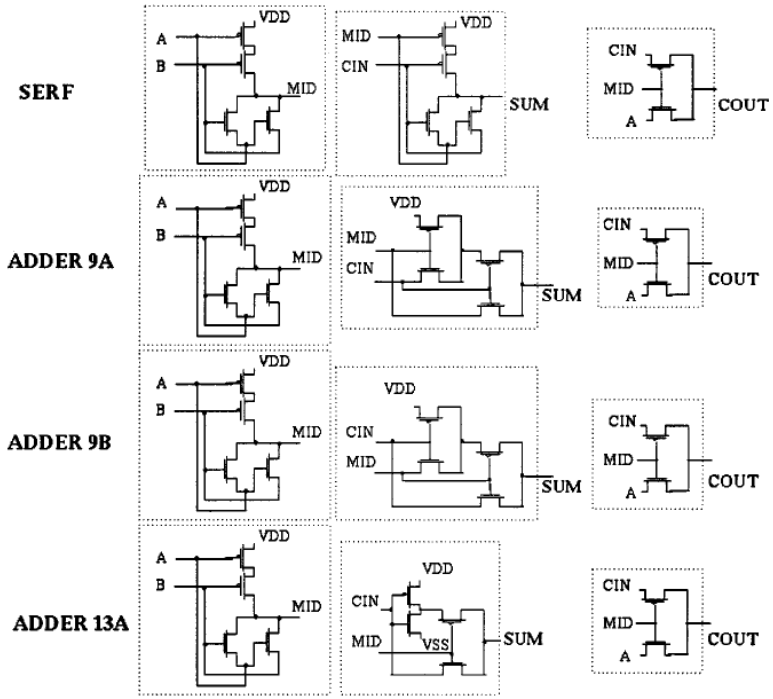
Transmisssion FA (en haut) vs celui-ci (en bas)



- 16 transistors, 0 inverseur (meilleure consommation)
- chemin critique réduit (plus d'inverseur)
- autres avantages en terme de consommation mais j'ai pas tout compris ;
- entre 5 et 20% d'amélioration sur les précédents suivant la métrique.

Ahmed M. Shams and Magdy A. Bayoumi, A Novel High-Performance CMOS 1-Bit Full-Adder Cell, *IEEE Transactions On Circuits And Systems II : Analog And Digital Signal Processing*, Vol. 47, No. 5, May 2000

Après j'arrête, promis



- Toute une famille à 10 transistors, obtenue sytématiquement
- Peut-on faire un CPA avec ?

Hung Tien Bui, Yuke Wang, and Yingtao Jiang, Design and Analysis of Low-Power 10-Transistor Full Adders Using Novel XORXNOR Gates, *IEEE Transactions On Circuits And Systems II : Analog And Digital Signal Processing*, Vol. 49, No. 1, January 2003

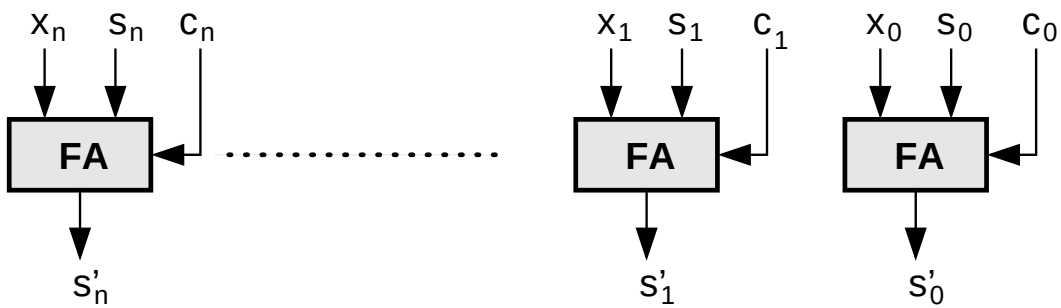
Moralité

- En l'an 2005 on bricole toujours la brique de base ;
- On n'a parlé ni du dimensionnement, ni du placement/routage de ces transistors ;
- Ce n'était qu'une des brique de base. Nous allons en voir d'autres
  (mais plus en transistor).

Additionneurs rapides

Prenons de la hauteur

- Si on connait toutes les retenues intermédiaires, le calcul de l'addition se fait en parallèle par les cellules *full adder*.



(En fait ce ne sont pas des FA puisqu'on économise tout le calcul de la retenue sortante)

Naissance, vie et mort des retenues

Considérons le *full adder* numéro i. Quelle que soit sa retenue entrante, il va

- créer *(generate)* une retenue si $x_i.y_i$,
- absorber *(kill)* la retenue entrante si $\overline{x_i}.\overline{y_i}$,
- propager *(propagate)* une retenue si $x_i \oplus y_i$.

Soit donc

$$\begin{cases} g_i &= x_i.y_i \\ k_i &= \overline{x_i}.\overline{y_i} \\ p_i &= x_i \oplus y_i \end{cases}$$

Avec cela on peut définir la retenue sortante par

$$\begin{cases} c_0 &= 0 \quad (\text{ou bien } c_{in}) \\ c_{i+1} &= g_i + p_i.c_i \end{cases}$$

Insistons lourdement

$$\begin{cases} g_i &= x_i.y_i \\ k_i &= \overline{x_i}.\overline{y_i} \\ p_i &= x_i \oplus y_i \end{cases}$$

Tiens, c'est étrange, ne devrait-on pas avoir not(propagate)=kill ! ? !

Relisons le transparent précédent :

- g_i c'est generate whatever the input carry
- k_i c'est kill whatever the input carry
- p_i c'est propagate whatever the input carry

Gardez cela à l'esprit dans la suite pour ne pas vous mélanger les pinceaux.

Switched carry-ripple (Manchester) adder

- Calculer les p_i en parallèle

$$\left\{ \begin{array}{lcl} p_i & = & x_i \oplus y_i \end{array} \right.$$

- Remplacer la chaîne $c_{in} \longrightarrow c_{out}$ à travers les *full adders* par une chaîne plus rapide (par exemple à travers des portes de transmission) commandées par les g_i

Temps toujours linéaire, mais avec une constante plus petite.

Si on utilise des portes de transmission, il faut regénérer le signal de temps en temps (typiquement toutes les 4 portes).

Carry-skip adder

- On calcule toujours les g_i, p_i et k_i en parallèle

$$\begin{cases} g_i & = & x_i.y_i \\ k_i & = & \overline{x_i}.\overline{y_i} \\ p_i & = & x_i \oplus y_i \end{cases}$$

- Les *full adder* sont regroupés en paquets de m :
  - un paquet est un additionneur séquentiel de m bits.
  - en plus de cette addition, chaque paquet calcule sont signal *propagate* (un gros ET logique sur ses p_i) ;
  - s'il ne propage pas, c'est qu'il y a eu un *kill* ou un *generate* dans le paquet, donc le paquet connait sa retenue sortante
  - Il y a donc un multiplexeur en sortie de chaque paquet, commandé par son *generate*, et choisissant entre son c_{in} et son c_{out}.
- Temps de calcul : faut regarder le chemin critique au pire cas.
- On doit trouver en gros $(2m-1)t_c + (n/m-1)t_{\text{mux}} + t_s$

Optimisation du carry-skip adder

- Le délai dépend de la taille m des groupes
- Pour avoir le m optimal, on dérive par rapport à m et on obtient une jolie formule.

$$m_{\text{opt}} = \sqrt{n.t_{\text{mux}}/2t_c}$$

- On peut aussi avoir des groupes de taille variable :
  - le chaînes de retenue qui partent du début sautent plus de blocs que les chaînes qui partent du milieu
  - on peut donc équilibrer les pires cas en faisant des groupes plus petits aux extrémités
  - pas de jolie formule...
- On peut aussi avoir des groupes de groupes (multi-level carry-skip)
- ...

Carry-select adder

- Une solution simple (et possiblement récursive) pour diminuer le délai :
  - faire en parallèle
    - l'addition des $n/2$ poids faibles,
    - l'addition des $n/2$ poids forts en supposant une retenue entrante de 0,
    - l'addition des $n/2$ poids forts en supposant une retenue entrante de 1,
  - Lorsque la retenue arrive de l'addition des poids faibles, choisir le résultat correspondant.
- Clairement, le délai diminue : $t_{\mathsf{csa}_2} = t_{\mathsf{add}_{n/2}} + t_{\mathsf{mux}}$
- (mais attention, le *fan-in* des bits de poids fort a augmenté !)
- au détriment de la surface... calculez vous-mêmes.
- Attention, le signal de commande du multiplexeur a un *fan-out* de $n/2$

Un peu plus général

- On peut couper en morceaux de m bits au lieu de $n/2$
- Il y a un genre de propagation de retenue entre les multiplexeurs
- Le temps de calcul est donc $t_{\mathsf{csa}_m} = t_{\mathsf{add}_m} + (\frac{n}{m} - 1)t_{\mathsf{mux}}$
- Le *fan-in* a seulement doublé.
- Le *fan-out* maximal est seulement m

La version récursive

Elle s'appelle *conditional-sum adder*

- On n'a que doublé (moins epsilon) le nombre des *full adder*
- mais on a un arbre de multiplexeurs
- et on a un problème de *fan-out* sur les signaux de commande des multiplexeurs

Additionneurs rapides à préfixe

Rappels

- *(generate)* $g_i = x_i.y_i$,
- *(kill)* $k_i = \overline{x_i}.\overline{y_i}$,
- *(propagate)* $p_i = x_i \oplus y_i$.
- $\begin{cases} c_0 & = & 0 \quad \text{(ou bien } c_{in}\text{)} \\ c_{i+1} & = & g_i + p_i.c_i \end{cases}$

Calculons c_{i+1} en fonction de c_{i-1} puis de c_{i-2} ...

La propagation de retenue est associative

- Considérons des *blocs de bits consécutifs*, notés $i : k$
  - retenue entrante c_k, retenue sortante c_{i+1}
  - generate du bloc noté $G_{i:k}$
  - propagate du bloc notés $P_{i:k}$
- On peut couper le bloc en deux à l'indice j :

$$\begin{aligned} G_{i:k} &= G_{i:j} + P_{i:j} G_{j:k} \\ P_{i:k} &= P_{i:j} P_{j:k} \end{aligned}$$

- Quand le bloc est composé d'un seul FA :

$$\begin{aligned} G_{i:i} &= g_i \\ P_{i:i} &= p_i \end{aligned}$$

- en plus concis : soit $I_{i:k} = (G_{i:k},\ P_{i:k})$

$$\begin{cases} I_{i:i} &= (g_i, p_i) \\ I_{i:k} &= I_{i:j} \bullet I_{j:k} \quad \forall k \leq j < i \end{cases}$$

- brique de base $\bullet$ associative composée de deux *et* et d'un *ou*.

Pour avoir des arbres de calcul des retenues

Il ne manque qu'un indice l comme *level* :

$$\begin{cases} (G^0_{i:i}, P^0_{i:i}) & = & (g_i, p_i) \\ (G^l_{i:k}\ ,\ P^l_{i:k}) & = & (G^{l-1}_{i:j} + P^{l-1}_{i:j} G^{l-1}_{j:k}\ ,\ P^{l-1}_{i:j} P^{l-1}_{j:k}) \quad \forall k \leq j < i \\ c_{i+1} & = & G^l_{i:0} \quad \text{(ou bien } G^l_{i:0} P^l_{i:0} c_{in}) \end{cases}$$

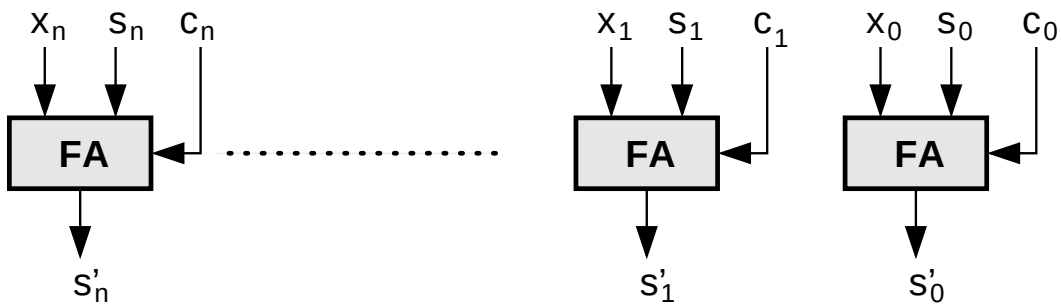
- en plus concis :

$$\begin{cases} I^0_{i:i} & = & (g_i, p_i) \\ I^l_{i:k} & = & I^{l-1}_{i:j} \bullet I^{l-1}_{j:k} \quad \forall k \leq j < i \end{cases}$$

- Il y a alors plein de manières de regrouper ces calculs en *arbres de préfixes*
  - choix des blocs $i : k$ pour chaque niveau l
- On va étudier certains de ces arbres du point de vue circuit.

Pourquoi on fait cela au fait ?

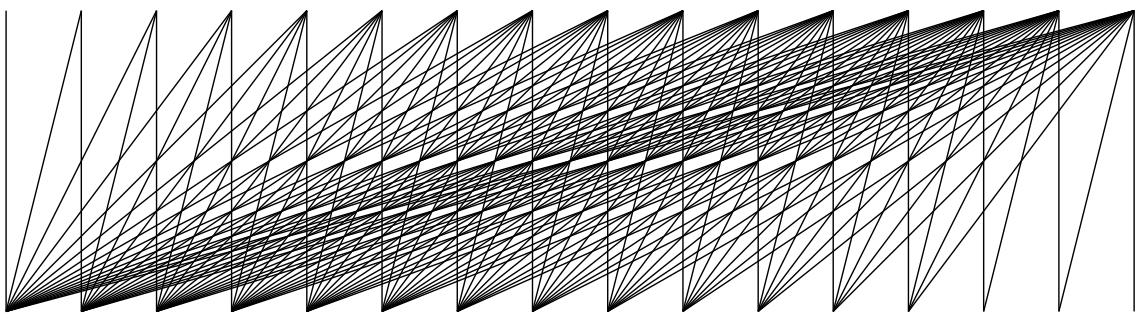
- (rappel) Si on connait toutes les retenues intermédiaires, le calcul de l'addition se fait en parallèle par les cellules FA.



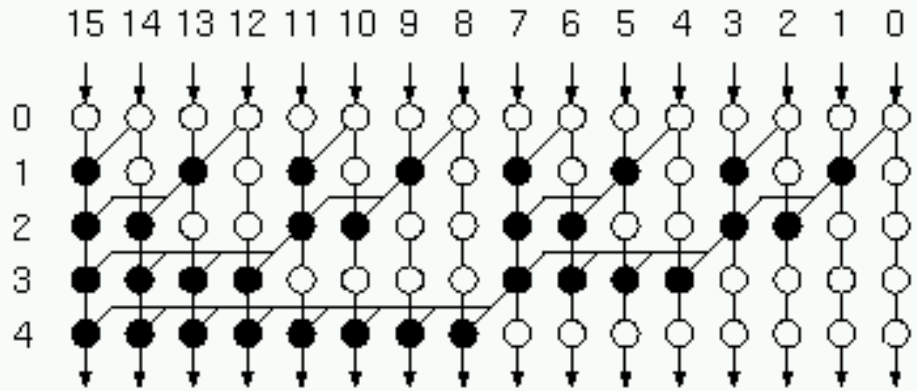
- Regardez le transparent précédent, il calcule toutes les retenues intermédiaires...

Arbres de préfixes

- On peut faire des arbres préfixes avec n'importe quel opérateur binaire associatif $\bullet$ pour lequel on veut calculer x_1, $x_1 \bullet x_2$, $x_1 \bullet x_2 \bullet x_3$, ...
- Tous les dessins qui suivent sont tirés de la thèse de Reto Zimmermann, entièrement consacrée aux additionneurs rapides
- Dans ces dessins, les ronds vides ne font rien (fils, ou buffers).



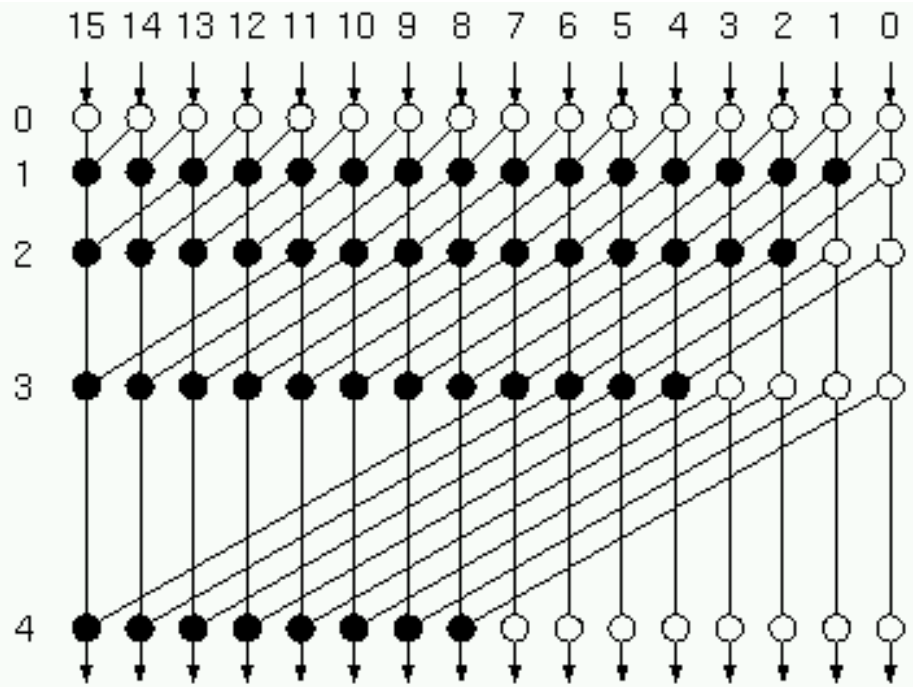
Arbre préfixe de Sklansky (1960)



$$
\begin{aligned}
T_{\bullet} &= \log n \\
A_{\bullet} &= \tfrac{1}{2} n \log n \\
A_{\bullet+\circ} &= n \log n \\
A_{\bullet} T_{\bullet} &\approx \tfrac{1}{2} n \log^2 n \\
A_{tracks} &= \log n \\
FO_{max} &= \tfrac{1}{2} n
\end{aligned}
$$

- Pas très réaliste à cause du *fan-out*
- À part cela, délai minimum
- Peu de fils horizontaux
- Mauvaise compaction verticale

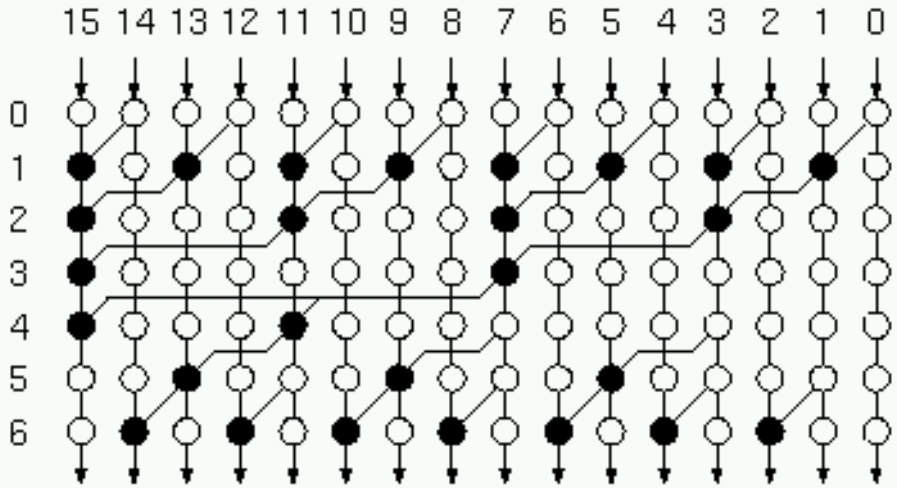
Arbre préfixe de Kogge-Stone (1973)



$$
\begin{aligned}
T_\bullet &= \log n \\
A_\bullet &= n \log n - n + 1 \\
A_{\bullet+\circ} &= n \log n \\
A_\bullet T_\bullet &\approx n \log^2 n \\
A_{tracks} &= n - 1 \\
FO_{max} &= 2
\end{aligned}
$$

- Plein d'arbres indépendants en parallèle
- Plus réaliste (trop) : *fan-out* de 2
- Délai minimal aussi mais le nombre de • a explosé
- Les fils prennent de la place

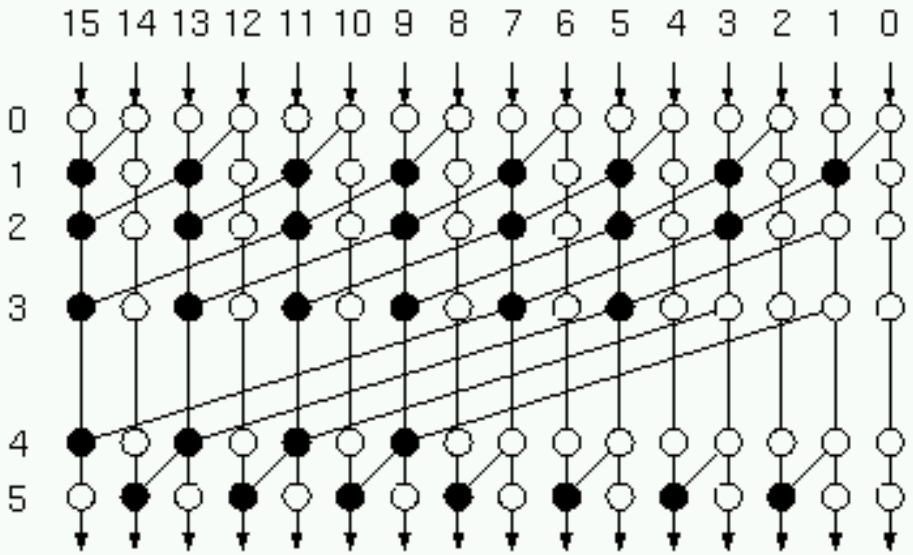
Arbre préfixe de Brent et Kung (1982)



$$
\begin{aligned}
T_{\bullet} &= 2\log n - 2 \\
A_{\bullet} &= 2n - \log n - 2 \\
A_{\bullet+\circ} &= 2n\log n - 2n \\
A_{\bullet}T_{\bullet} &\approx 4n\log n \\
A_{tracks} &= 2\log n - 1 \\
FO_{max} &= \log n
\end{aligned}
$$

- *Fan-out* de $\log_2 n$, et même 3 si les $\circ$ contiennent des buffers.
- Délai doublé
- Nombre de $\bullet$ en $O(n)$
- Possibilité de compaction verticale

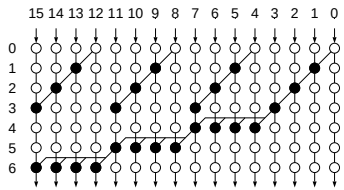
Arbre préfixe de Han-Carlson (1987)



$$
\begin{aligned}
T_{\bullet} &= \log n + 1 \\
A_{\bullet} &= \tfrac{1}{2} n \log n \\
A_{\bullet+\circ} &= n \log n + n \\
A_{\bullet} T_{\bullet} &\approx \tfrac{1}{2} n \log^2 n \\
A_{tracks} &= \tfrac{1}{2} n + 1 \\
FO_{max} &= 2
\end{aligned}
$$

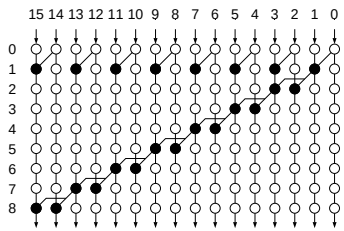
- Les avantages des deux précédents en les combinant : premier et dernier étages en Brent et Kung, milieu en Kogge-Stone.
- On peut aussi mettre plus d'étages de Brent et Kung.

Il y en a plein d'autres (cf aussi Arith 2001)



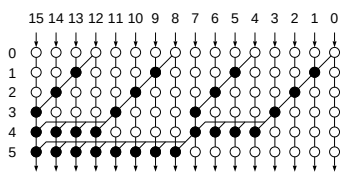
$$\begin{aligned} T_\bullet &= \tfrac{1}{4}n+2 \\ A_\bullet &= \tfrac{7}{4}n-4 \\ A_{\bullet+\circ} &= 2n \\ A_\bullet T_\bullet &\approx \tfrac{7}{16}n^2 \\ A_{tracks} &= 2 \\ FO_{max} &= \tfrac{1}{4}n \end{aligned}$$

Figure 3.21: *Group-prefix algorithm: 4 groups, 1-level parallel.*



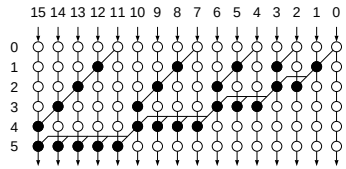
$$\begin{aligned} T_\bullet &= \tfrac{1}{m}n+m-2 \\ A_\bullet &= 2n-\tfrac{1}{m}n-m \\ A_{\bullet+\circ} &= 2n \\ A_\bullet T_\bullet &\approx \tfrac{2}{m}n^2 \\ A_{tracks} &= 2 \\ FO_{max} &= \tfrac{1}{m}n \end{aligned}$$

Figure 3.22: *Group-prefix algorithm: m (8) groups, 1-level parallel.*



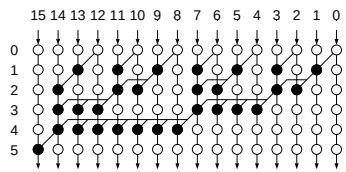
$$\begin{aligned} T_\bullet &= \tfrac{1}{4}n+1 \\ A_\bullet &= 2n-4 \\ A_{\bullet+\circ} &= 3n \\ A_\bullet T_\bullet &\approx \tfrac{1}{2}n^2 \\ A_{tracks} &= 3 \\ FO_{max} &= \tfrac{1}{2}n \end{aligned}$$

Figure 3.23: *Group-prefix algorithm: 2×2 groups, 2-level parallel.*



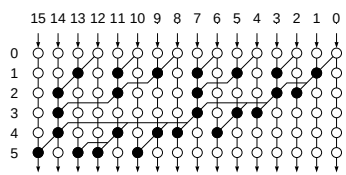
$$\begin{aligned} T_\bullet &\approx \sqrt{2n} \\ A_\bullet &\approx 2n-\sqrt{2n}-2 \\ A_{\bullet+\circ} &= 2n \\ A_\bullet T_\bullet &\approx 2n\sqrt{2n} \\ A_{tracks} &= 2 \\ FO_{max} &\approx \sqrt{2n} \end{aligned}$$

Figure 3.24: *Group-prefix algorithm: variable groups, 1-level parallel.*



$$\begin{aligned} T_\bullet &\approx \sqrt[3]{6n} \\ A_\bullet &\approx 3n \\ A_{\bullet+\circ} &= 3n \\ A_\bullet T_\bullet &\approx 3n\sqrt[3]{6n} \\ A_{tracks} &= 3 \\ FO_{max} &\approx (6n)^{2/3} \end{aligned}$$

Figure 3.25: *Group-prefix algorithm: variable groups, 2-level parallel.*



$$\begin{aligned} T_\bullet &\approx \sqrt[3]{6n}+1 \\ A_\bullet &\approx 2n \\ A_{\bullet+\circ} &= 2n \\ A_\bullet T_\bullet &\approx 2n\sqrt[3]{6n} \\ A_{tracks} &= 3 \\ FO_{max} &\approx \sqrt[3]{6n} \end{aligned}$$

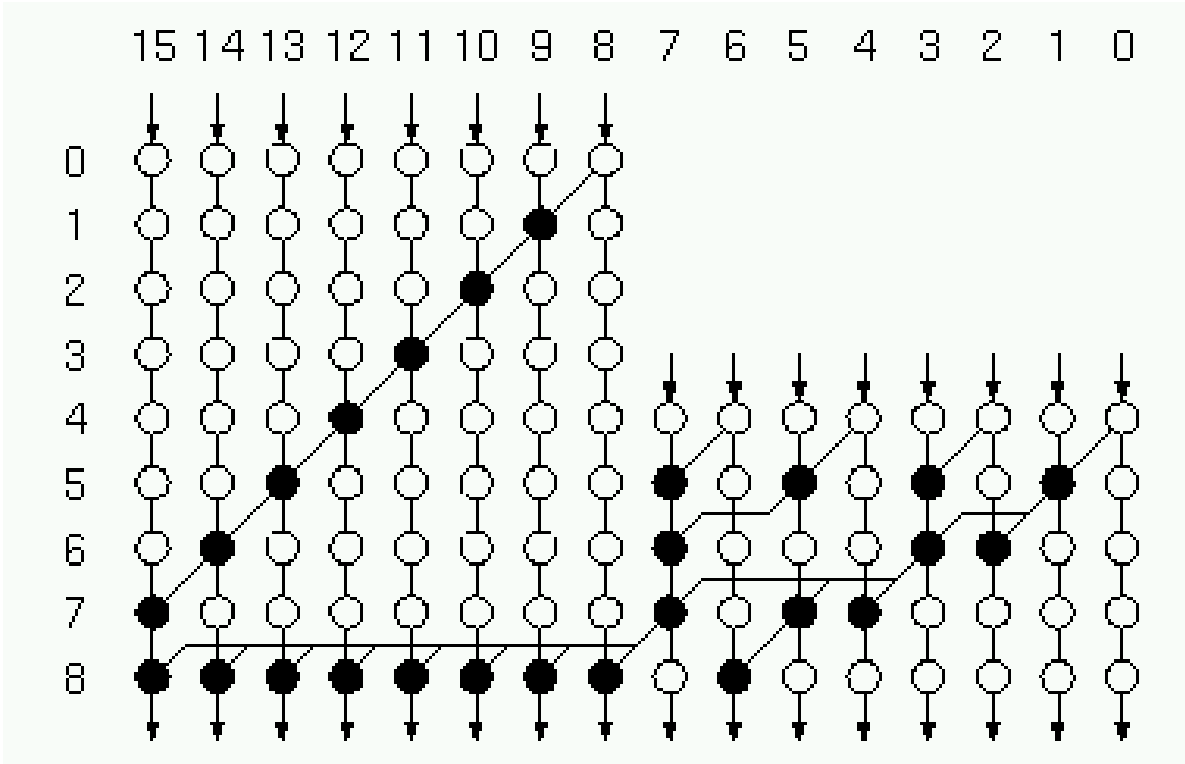
Figure 3.26: *Group-prefix algorithm: variable groups, 2-level parallel, optimized.*

Vous avez remarqué

... que toute la partie sur les *full adder* rapides ne me sert plus à rien ?

Autres possibilités

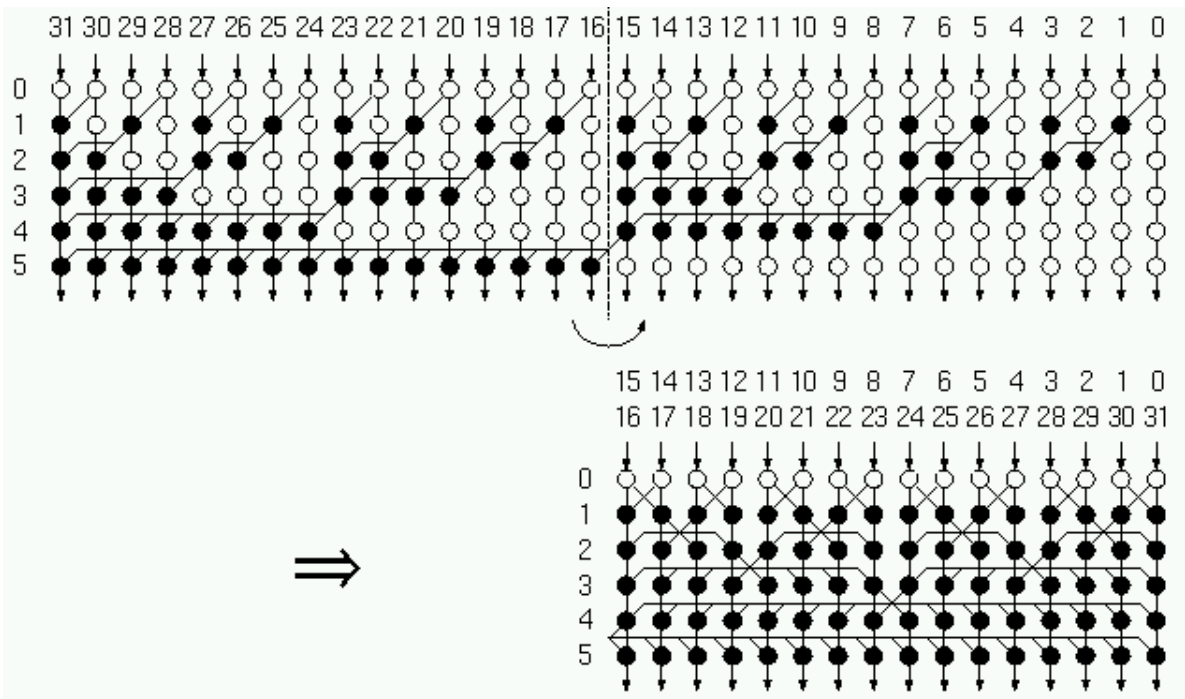
- Il arrive qu'une partie d'un mot arrive avant l'autre (exemple : adresse dans un contexte de mémoire virtuelle)



- pipelines, etc...

Un peu de placement

- On aime avoir un *layout* régulier
- Souvent les ∘ sont vides ⟶ perte de place
- Certains arbres préfixes (ici Sklansky) se prêtent au *repliage* :



Additionneurs parallèles

On a déjà presque tout vu

Rappels :

- *Carry-save* adders
- Additionneurs d'Avizienis

Reste juste l'additionneur binaire signé : voir le Muller Rose

Questions de granularité

Il n'y a pas que les bits dans la vie

- En passant de la base 2 à la base 4 ou 8, la plupart des idées précédentes fonctionnent encore.
- Intérêt : optimisation en transistors des briques de base
- Inconvénient : tout est plus compliqué.

Il y a un thésard à Lannion qui travaille en base 3 (3 niveaux électriques). Les Russes avaient étudié cela aussi.

Conclusions

Moralité

- En l'an 2002 on bricole toujours les additionneurs rapides.
- Et ce n'est pas fini : le formalisme des arbres de préfixes permet de naviguer sainement dans l'espace algorithmique pour choisir l'architecture la mieux adaptée à une technologie et à un problème donnés.
- Par ailleurs on peut encore combiner les différentes approches précédentes en fonction de ce qu'on recherche.
- La portée des fils devient de plus en plus un facteur à prendre en compte.
  Y a-t-il des repliages sioux ?
- ...

Amoralité

Tous ces additionneurs rapides, pour les FPGA, c'est comme pisser dans un violon.