Systèmes numériques : de l'algorithme aux circuits

Logique CMOS & Logique Combinatoire

Hadrien Barral (prenom.nom@ens.psl.eu)

(Basé sur les slides de Systèmes Numériques d'Yves Mathieu)

Version : 1.0

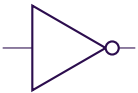
Construction d'une porte logique

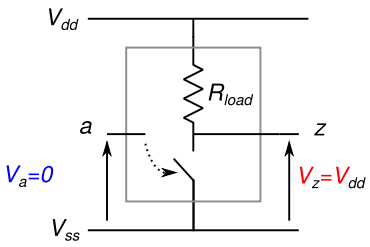
De quoi avons-nous eu besoin ?

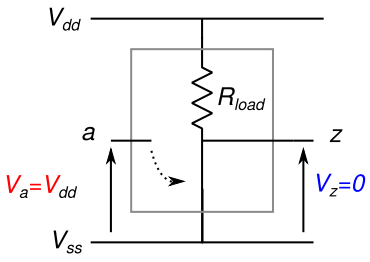
- Une référence de masse : V_{ss}
- Une source d'alimentation : V_{dd}
- Une équivalence électrique pour les niveaux logiques : $0 \equiv V_{ss}$, $1 \equiv V_{dd}$
- Une charge résistive : R_{load}
- Un interrupteur contrôlé par une tension (référencée à V_{ss})
  - $V_{control} = 0$: Interrupteur Ouvert
  - $V_{control} = V_{dd}$: Interrupteur Fermé

Porte NOT

Version schématique

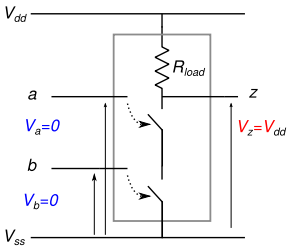


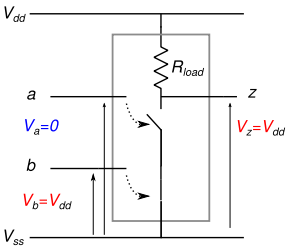


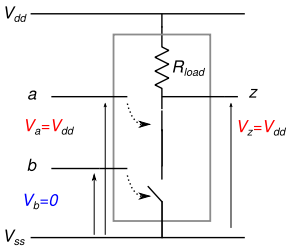


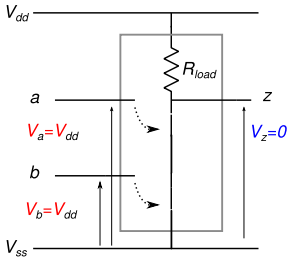
Porte NAND

Version schématique



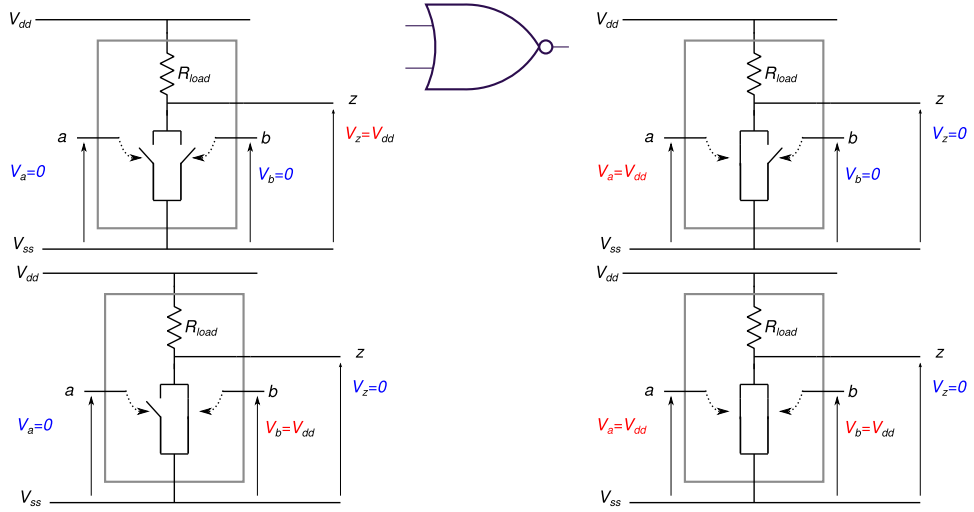






Porte NOR

Version schématique



Limitations des logiques RTL/DTL/PMOS/NMOS/...

Problèmes :

- Un courant permanent traverse la porte quand la sortie est à 0 :
  - On aimerait disposer d'une porte qui ne consomme de l'énergie que lorsqu'elle change d'état...
- Les physiciens ne savent pas réaliser des interrupteurs idéaux (à des températures et des pressions raisonnables) :
  - La sortie de la porte n'atteint pas le niveau minimal V_{ss}
  - Il n'est pas garanti qu'un assemblage de telles portes logiques soit fonctionnel

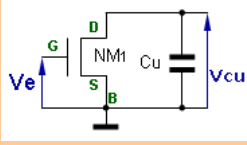
Efficacité des interrupteurs MOS

Portes CMOS : les charges capacitives parasites

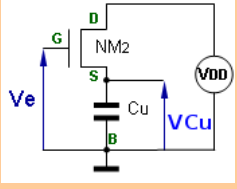
- Les nœuds internes d'une porte logique CMOS ont des capacités parasites :
  - Capacité grille-source ou grille-drain (oxyde de grille)
  - Capacité source-substrat ou drain-substrat (diodes polarisées en inverse)
- La sortie d'une porte logique est connectée à :
  - Des lignes métalliques de connexion (capacité parasite avec la masse, l'alimentation, entre nœuds)
  - Des entrées d'autres portes logiques (grilles de transistors)
- Tous ces éléments peuvent être modélisés par une capacité parasite équivalente unique C_{par} connectée entre la sortie de la porte et la masse
  - Qui doit être chargée pendant les transitions montantes de la sortie de la porte
  - Qui doit être déchargée pendant les transitions descendants de la sortie de la porte
- Le temps de calcul d'une porte logique est directement lié au temps de charge et de décharge de cette capacité.

Le transistor NMOS est il un interrupteur efficace ?

Transistor NMOS déchargeant une capacité C_u



Transistor NMOS chargeant une capacité C_u



Cf simulation

Efficacité des interrupteurs MOS

portes CMOS : les charges capacitives parasites

- La mobilité des charges est différente parce que les composants sont différents
- A dimensions et tensions (V_{gs}, V_{ds}) identiques, le courant entre l'émetteur et le collecteur (I_{ds}) change.
- Le transistor PMOS est moins efficace que le transistor NMOS

Interrupteurs empilés : l'effet de substrat

- La source des transistors empilés n'est pas connectée à V_{ss}
- La tension de seuil du transistor croît avec V_{sb} (la différence de potentiel entre la source et le substrat)
- Les transistors empilés sont moins efficaces...

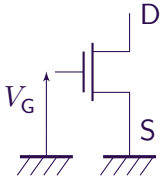
Efficacité des interrupteurs MOS

Quelques règles d'ingéniérie... :

- Les transistors NMOS sont de préférence utilisés pour décharger des capacités (génération du 0 logique).
- Les transistors PMOS sont de préférence utilisés pour charger des capacités (génération du 1 logique).
- Pour une efficacité identique les transistors PMOS doivent être plus larges que les transistors NMOS.
- Les empilements de transistors doivent être limités à 3 or 4 transistors.

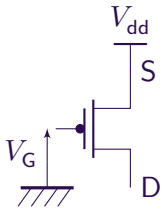
Transistors MOS et niveaux logiques

Deux interrupteurs non idéaux



Un transistor NMOS dont la Source est connectée à la masse.

- $V_G = V_{ss}$
  - interrupteur ouvert
- $V_G = V_{dd}$
  - interrupteur fermé



Un transistor PMOS dont la Source est connectée à l'alimentation

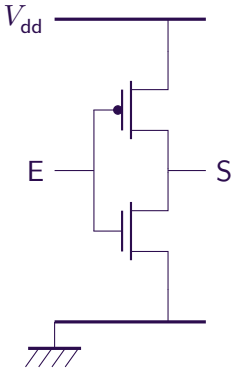
- $V_G = V_{ss}$
  - interrupteur fermé
- $V_G = V_{dd}$
  - interrupteur ouvert

Inverseur CMOS

- Valeur booléenne de l'entrée $E = 0$
  - $V_E = 0$
    - NMOS bloqué
    - PMOS conducteur
  - $V_S = V_{dd}$

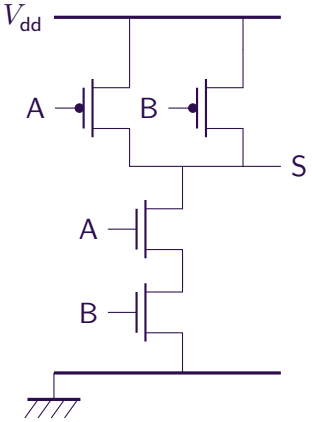
  $\rightarrow$ Valeur booléenne de la sortie $S = 1$
- Valeur booléenne de l'entrée $E = 1$
  - $V_E = V_{dd}$
    - NMOS conducteur
    - PMOS bloqué
  - $V_S = 0$

  $\rightarrow$ Valeur booléenne de la sortie $S = 0$



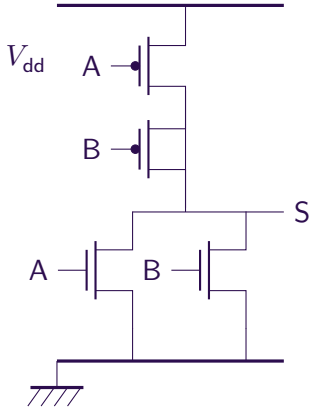
En première approximation : pas de consommation statique.

CMOS : Porte NAND



CMOS : Porte NOR

Le dual de NAND

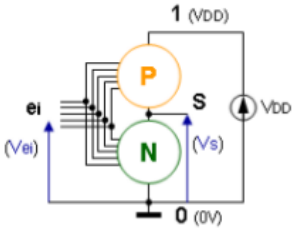


Logique CMOS

Généralisation à d'autres fonctions booléennes

Étant donnée la table de vérité d'une fonction booléenne :

- Un réseau de transistors PMOS est utilisé pour générer les 1 de la table de vérité
- Un réseau de transistors NMOS est utilisé pour générer les 0 de la table de vérité
- Les fonctions logiques implémentables sont de la forme
  $F(x_0, x_1, \ldots, x_n) = \overline{\sum \prod x_i}$
  $\rightarrow$ À savoir quoi ?
- La porte contient un nombre de transistors PMOS identique au nombre de transistors NMOS
- Les autres fonctions booléennes seront réalisées en assemblant ces portes CMOS primitives.



Méthode de construction de fonctions logiques de la forme $F = \overline{\sum \prod x_i}$

Réseau NMOS

En premier, construire le réseau NMOS :

- Exprimer la fonction $\overline{F(x_0, x_1, ..., x_n)}$ sous la forme $\sum \prod x_i$
- Réaliser toutes les simplifications et factorisations possibles
- Les $\prod$ restants correspondent à des séries de transistors NMOS (ou des séries de réseaux de transistors NMOS)
- Les $\sum$ restants correspondent à des transistors NMOS (ou des réseaux de transistors NMOS) connectés en parallèle.

Méthode de construction de fonctions logiques de la forme $F = \overline{\sum \prod x_i}$

Réseau PMOS

En second, construire le réseau PMOS :

- Exprimer la fonction $F(x_0, x_1, ..., x_n)$ sous la forme $\sum \prod \overline{x_i}$
- Réaliser toutes les simplifications et factorisations possibles
- Les $\prod$ restants correspondent à des séries de transistors PMOS (ou des séries de réseaux de transistors PMOS)
- Les $\sum$ restants correspondent à des transistors PMOS (ou des réseaux de transistors PMOS) connectés en parallèle.

Méthode de construction de fonctions logiques de la forme $F = \overline{\sum \prod x_i}$

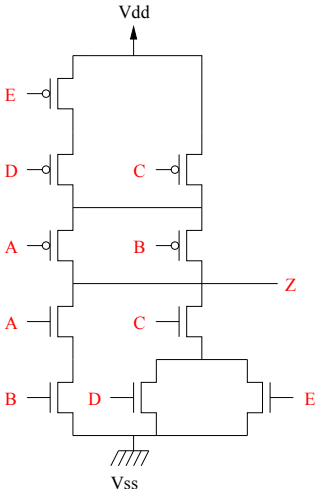
Remarques

Quelques remarques :

- Les portes CMOS les plus simples sont des portes inverseuses (le `NAND` est plus *petit* que le `AND`)
- On peut utiliser 2 réseaux PMOS et NMOS de topologies duales :
  - Utiliser des réseaux de transistors PMOS en parallèle quand des réseaux de transistors NMOS sont en série
  - Utiliser des réseaux de transistors PMOS en série quand des réseaux de transistors NMOS sont en parallèle
- Cette dernière méthode conduit souvent à des structures sous-optimales (d'un point de vue électrique)

Logique CMOS : analyse d'une porte

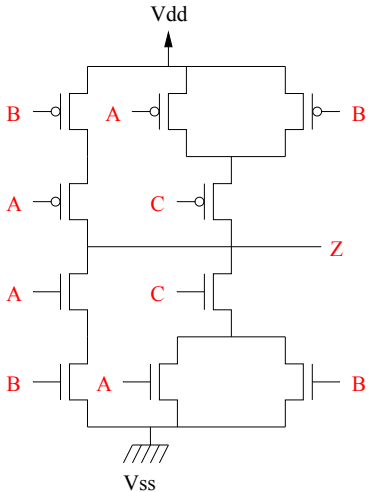
$$Z = \overline{A \cdot B + C \cdot (D + E)}$$



Exercice

Réseau optimal : $F = \overline{A \cdot B + B \cdot C + A \cdot C}$

1. Construire la porte en générant le réseau NMOS et le réseau PMOS de manière indépendante (C sera mis en facteur)
2. Échanger le transistor NMOS C avec le réseau de transistors NMOS A/B : y a-t-il une différence ?
3. Construire la porte en utilisant des réseaux PMOS et NMOS duaux.
4. Quels sont les inconvénients d'une telle structure ?



Petit/Grand nombre d'entrés : l'exemple du AND6

- Exemple de la porte AND6, procédé 28nm, tensions d'alimentation de 1V, tous les transistors NMOS sont de tailles identiques, $w_p/w_n = 1.6$, pas de charge en sortie.
- Cas testés : (1) Toutes les entrées changent en même temps , (2) Une seule entrée change.

Temps de propagation en fonction de l'implémentation :

NAND6+INV

$TP_1^R = 53$ ps
$TP_1^F = 18$ ps
$TP_2^R = 33$ ps
$TP_2^F = 11$ ps

NAND3+NOR2

$TP_1^R = 19$ ps
$TP_1^F = 5$ ps
$TP_2^R = 19$ ps
$TP_2^F = 9$ ps

NAND2+NOR3

$TP_1^R = 27$ ps
$TP_1^F = 4$ ps
$TP_2^R = 18$ ps
$TP_2^F = 9$ ps

Petit/Grand nombre d'entrés : l'exemple du AND6

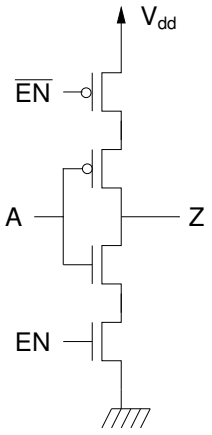
- Exercice pour ceux qui aiment la physique : définir un modèle simpliste du temps de propagation en utilisant un modèle résistif simple.
- Outre le temps le propagation, la consommation nécessite une étude similaire.
  → Voir *Low-Power Digital VLSI Design : Circuits and Systems* section 4.5.4.1.

Réseaux PMOS/NMOS non complémentaires

Cas bloqué

- Les réseaux NMOS et PMOS peuvent être simultanément bloqués.
- Cas typique d'utilisation : Entrées/Sorties sur bus externe au circuit.
- Exemple de circuit : inverseur 3-états.

EN	A	Z
0	0	hi-Z (haute impédance)
0	1	hi-Z (haute impédance)
1	0	1
1	1	0

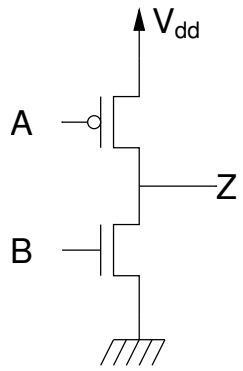


Réseaux PMOS/NMOS non complémentaires

Cas passant

- Les réseaux NMOS et PMOS peuvent être simultanément passants.
- Contrainte sur les entrées autorisées pour éviter les courts-circuits.
- Habituellement non géré par les outils automatiques
- Exemple de circuit :

EN	A	Z
0	0	1
0	1	invalide (interdit)
1	0	hi-Z (haute impédance)
1	1	0



Efficacité de la logique CMOS : critères de performance

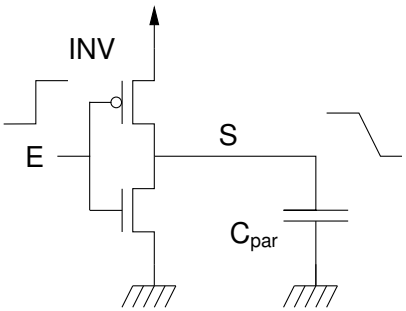
Dual du cours précédent

- **Surface** & **Coût** :
  Plus la puce est petite, plus l'efficacité de production est élevée, et plus le coût de fabrication est faible.
  - Utilisation de transistors plus petits (évolution technologique)
  - Utilisation d'un nombre réduit de transistors (choix architecturaux)
- **Vitesse** :
  Des portes logiques plus rapides impliquent une plus grande puissance de traitement.
  - Comment augmenter la fréquence d'horloge ?
- **Consommation** :
  Le calcul implique une consommation électrique.
  - Comment minimiser cette consommation d'énergie ? (e.g. embarqué)
  - Comment évacuer la chaleur dissipée ? (e.g. datacenter)

Délai d'une porte logique

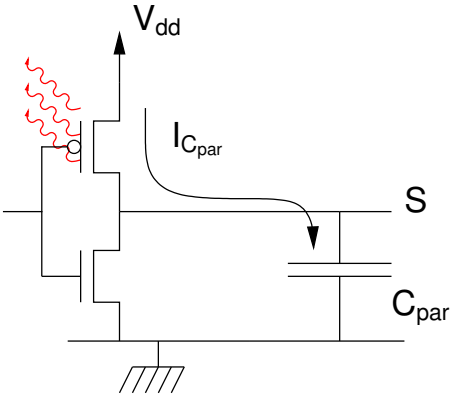
Un modèle simple appliqué au cas simple d'un front montant sur l'entrée d'un inverseur.

- Hypothèse 1 : le front montant est de durée nulle
- Hypothèse 2 : la seule capacité parasite est celle de la porte logique
- Hypothèse 3 : le courant circulant à travers les transistors pour la charge ou la décharge de la capacité parasite C_{par} est approximativement égal au courant qui traverse le transistor

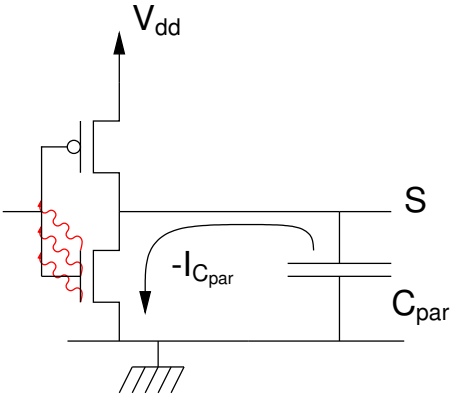


Consommation électrique de la logique CMOS

Énergie dissipée v.s. énergie stockée



Front montant (*rising edge*)



Front descendant (*falling edge*)

Consommation électrique de la logique CMOS

Bilan énergétique

Charge : l'énergie provient de l'alimentation

$$E_{V_{dd}} = C_{par} \int_0^{V_{dd}} V_{dd} dV_S = C_{par} {V_{dd}}^2$$

Décharge : énergie stockée dans la capacité

$$E_{C_{par}} = C_{par} \int_0^{V_{dd}} V_S dV_S = C_{par} \frac{{V_{dd}}^2}{2}$$

Dissipation de $C_{par} \frac{{V_{dd}}^2}{2}$ (dans les 2 sens)

Consommation électrique de la logique CMOS

Consommation d'une puce complète

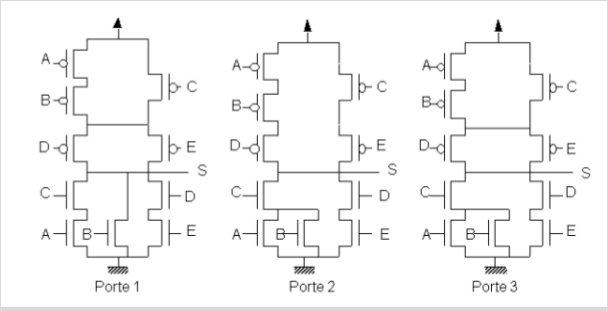
- Soit C_{chip} la capacité parasite totale de la puce
- Soit F_{clk} la fréquence de fonctionnement de l'horloge de la puce (logique synchrone)
- Soit T_{act} la probabilité moyenne de transition des signaux pendant un seul cycle de l'horloge ($T_{\text{act}} \approx 0.3$).

Consommation de la puce complète

$$P_{\text{circuit}} \approx T_{\text{act}} F_{\text{clk}} C_{\text{chip}} {V_{\text{dd}}}^2$$

Exercice

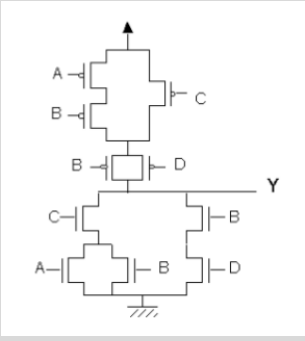
Le juste circuit CMOS



1. Quels sont les montages CMOS corrects ?
2. Pour les montages CMOS corrects, donnez la fonction logique.
3. Pour les autres, donnez un jeu d'entrées menant à un problème.

Exercice

Circuit mystère CMOS



1. Que calcule le côté NMOS de ce circuit ? Et le côté PMOS ?
2. Conclure.

Exercice

Circuit minorité en CMOS

1. Synthétisez la fonction XOR : $A \cdot \overline{B} + \overline{A} \cdot B$. Quel est le nombre minimum de transistors nécessaires ?
2. Synthétisez la fonction minorité a 3 entrées : $\text{Min}(A, B, C) = \overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C} + \overline{B} \cdot \overline{C}$. Trouvez une structure minimisant le nombre de transistors.
3. Montrez que $\text{Min}(A, B, C) = \text{Max}(\overline{A}, \overline{B}, \overline{C})$, avec Maj la fonction majorité à 3 entrées.
4. Déduisez-en une structure symétrique entre les réseaux P et N.

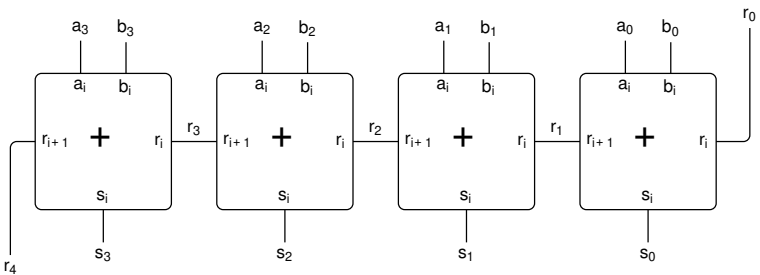
Addition

Revenons en primaire

Comment faire une addition de 2 nombres de 4 bits ?

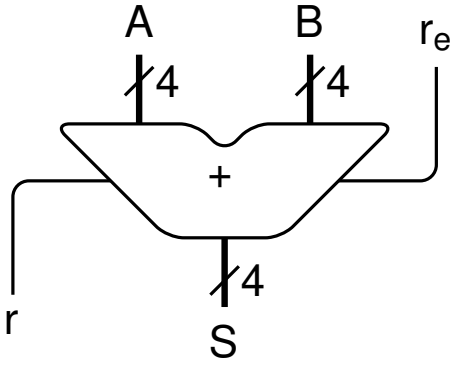
Décomposition de l'addition

- L'addition peut être décomposée en plusieurs additions élémentaires sur 1 bit.



Additionneur à propagation de retenue

Retenue en entrée et en sortie.



Additionneur sur 1 bit (1/2)

Arithmétiquement

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

Table de vérité

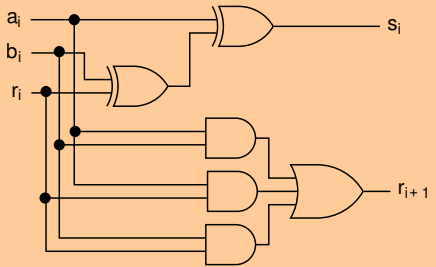
a_i	b_i	r_i	r_{i+1}	s_i	**Décimal**
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

Additionneur sur 1 bit (2/2)

Équations booléennes

$$s_i = a_i \oplus b_i \oplus r_i$$
$$r_{i+1} = a_i \cdot b_i + a_i \cdot r_i + b_i \cdot r_i$$

Schéma



(Une autre solution avec "half-adder" sera vue en TD.)

Soustraction

Dans la quasi-totalité des cas, inutile de faire un surtout de soustraction dédié.
Hypothèse (globalement[1] vraie de nos jours) : on code les nombres en complément à 2.

[1] : *voir C11 vs C23 (section, 6.2.6.2)*

- Soit B un nombre codé en complément à 2 sur n bits.
- On a $-B = \overline{B} + 1$.
- Pour faire $A - B$, il suffit alors de calculer $A + (-B) = A + \overline{B} + 1$.
  - On réutilise l'additionneur,
  - on ajoute des portes `NOT` pour faire $\overline{B}$,
  - le 1 peut se rajouter directement dans r_e.

Exercice

Vous avez 4 secondes

1. On se place dans l'arithmétique en complément à 2 sur 4 bits.
   Soit $X = -8$. Combien vaut $-X$?

Dynamique de l'addition

Dépassement de capacité pour l'addition non-signée

Nombres positifs

Pour deux nombres A et B représentés sur n bits nous avons :

$$A \leq 2^n - 1$$
$$B \leq 2^n - 1$$
$$A + B \leq 2^{n+1} - 2 < 2^{n+1}$$

$A + B$ est toujours représentable sur $n + 1$ bits.

Dynamique de l'addition

Dépassement de capacité pour l'addition signée

Nombres en complément à 2

Pour deux nombres A et B représentés en complément à 2 sur n bits nous avons :

$$\begin{aligned} -2^{n-1} \leq \quad A \quad &\leq 2^{n-1}-1 \\ -2^{n-1} \leq \quad B \quad &\leq 2^{n-1}-1 \\ -2^{n} \leq A+B &\leq 2^{n}-2 < 2^{n} \end{aligned}$$

$A + B$ est toujours représentable sur $n + 1$ bits.

Dynamique de l'addition

Interprétation de la retenue 1/2

Nombres en complément à 2

					non signé	CA2
		1	1	1	7	− 1
+		0	0	1	1	1
=	1	0	0	0	8	0 ou − 8?

				non signé	CA2
	0	1	1	3	3
+	0	0	1	1	1
=	1	0	0	4	− 4

				non signé	CA2
	1	1	1	7	− 1
+	1	0	0	4	− 4
= 1	0	1	1	11	+ 3 + retenue?

En complément à 2, l'interprétation de la retenue n'est pas la même que pour les nombres non signés.

Dynamique de l'addition

Interprétation de la retenue 2/2

En complément à 2, une solution simple pour toujours avoir le bon résultat est de d'abord étendre les nombres sur un bit de plus puis faire la somme. La retenue produite au delà du bit ajouté n'est pas prise en compte.

Nombres en complément à 2

		1	1	1	1	− 1
+		1	1	0	0	− 4
=	✗	1	0	1	1	− 5

		1	1	1	1	− 1
+		0	0	0	1	1
=	✗	0	0	0	0	0

Exercice

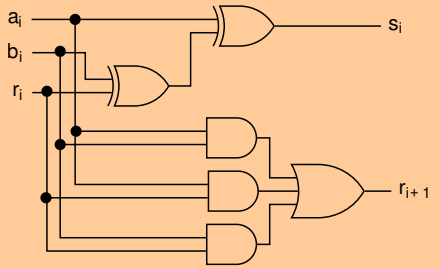
Full-adder

1. Estimez le nombre de transistors nécessaires à la fabrication de la structure sur le schéma ci-contre.
2. Vérifiez que s_i peut s'exprimer sous la forme :
   $s_i = a_i \cdot b_i \cdot r_i + \overline{r_{i+1}} \cdot (a_i + b_i + r_i)$
3. En déduire une nouvelle construction (sous forme de transistors)
4. Comparez les deux solutions

Schéma

$$s_i = a_i \oplus b_i \oplus r_i$$

$$r_{i+1} = a_i \cdot b_i + a_i \cdot r_i + b_i \cdot r_i$$

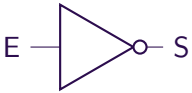


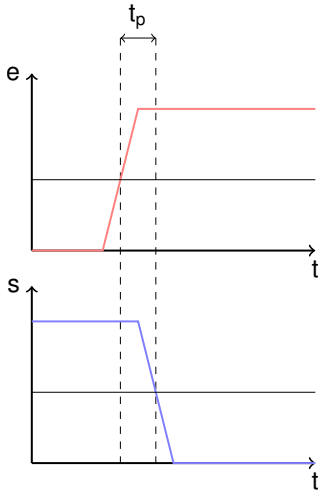
Temps de propagation d'une porte

- Les portes logiques respectent les lois de la physique. Les changement d'état ne peuvent pas être instantanés.
- Le temps de propagation est le temps entre le changement des entrées d'une porte et la stabilisation de la valeur de sa sortie.
- La valeur de la sortie n'est valide qu'après ce temps.

Temps de propagation d'une porte sur un inverseur

Avec un modèle physique simple

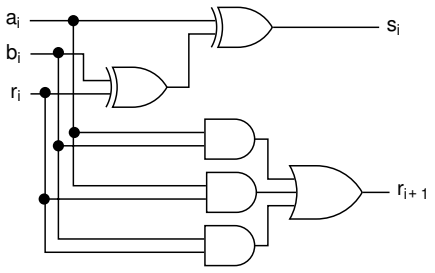




Temps de calcul d'une porte complexe

- Les portes logiques de base sont pré-caractérisées.
- Pour une technologie particulière, on connaît le temps de propagation des portes de base.
- À partir de ces tables on calcule le temps de propagation des portes complexes en faisant la somme des temps individuels des portes qui se suivent.
- Le temps de calcul d'une porte complexe est le temps de propagation du chemin le plus long.

Temps de calcul : exemple sur un full-adder

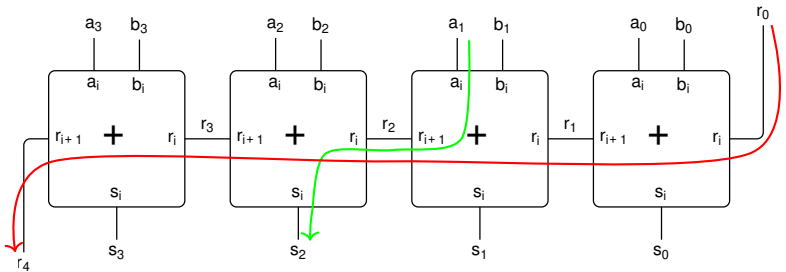


- Considérons que le temps de propagation est de :
  - 1ns pour les portes `AND` et `OR`
  - 2ns pour les portes `XOR`
- Quel est le temps de propagation des entrées vers chaque sortie ?

Chemin critique

Définition

Le **chemin critique** est le plus long chemin possible en terme de temps de propagation.



Exemple sur un additionneur 4-bits.

Modèles du temps de propagation

On peut regarder des modèles de temps de propagation plus fins, e.g.

Un modèle plus précis de temps de propagation

On peut distinguer :

- Le temps de propagation d'une entrée vers la sortie pour une transition montante de la sortie.
- Le temps de propagation d'une entrée vers la sortie pour une transition descendante de la sortie.

Un autre modèle plus précis de temps de propagation

On peut définir un temps de propagation par type de porte **et** fan-out (nombre d'entrées branchées à la sortie).

Exercice

Comparateur

On désire réaliser un opérateur capable d'effectuer la comparaison de 2 nombres positifs A et B codés sur n bits. La sortie S de l'opérateur vaut 1 si A est strictement inférieur à B, 0 sinon.

1. Proposer une solution à l'aide d'un soustracteur.
2. Proposer une solution en comparant bit à bit les nombres A et B en commençant par les bits de poids forts.
3. Construire un opérateur élémentaire à 2 entrées a_i et b_i et 2 sorties I_i (Inférieur) et E_i (Égal), pouvant servir à implémenter la solution précédente.
4. En utilisant l'opérateur construit précédemment, proposer le schéma complet du comparateur.
5. Conclure sur l'intérêt par rapport la la solution à base de soustracteur.

Une pluralité de méthodes pour faire un additionneur

Il existe de nombreux types d'additionneurs.

- Ripple-carry adder *(vu précédement)*
- Carry-lookahead adder *(sera vu en TD)*
- Brent–Kung adder
- Kogge–Stone adder
- Carry-save adder
- Carry-select adder
- Carry-skip adder
- Conditional Sum Adders
- ...

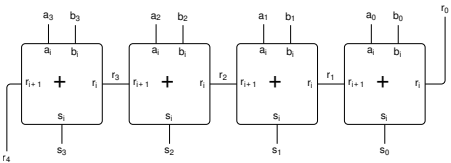
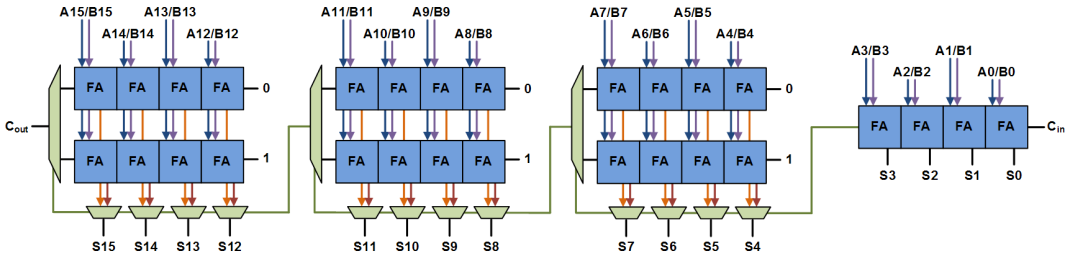


Schéma pour un *Ripple-carry adder*.
Quel est le meilleur temps de propagation possible pour un additionneur ?

Carry-select adder

Version à taille fixe

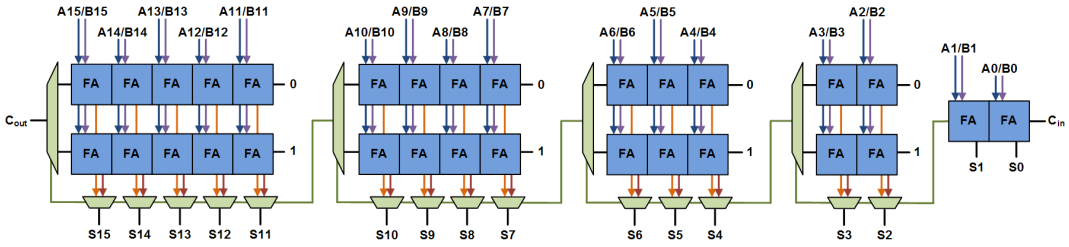


Source : Wikipedia

Complexité : $O(\sqrt{n})$

Carry-select adder

Version à taille variable



Source : Wikipedia

Délai optimal avec l'hypothèse (douteuse) que la propagation d'un `MUX` et d'un *full-adder* sont identiques.

Exemple réel sur un additionneur 32-bit (0.8µm CMOS, 3.3V) : 8-9-7-4-4.

Carry-skip adder

Au tableau

Multiplieur

Différentes approches

Tout comme pour les additionneurs, il existe de nombreux types de multiplieurs.

- Braun multiplier
- Baugh-Wooley multiplier
- Booth multiplier & encoding
- Wallace tree
- Dadda multiplier
- ...

Revenons (encore !) en primaire

$$
\begin{array}{r}
1\,7\,9\,4 \\
\times\ 1\,9\,9\,9 \\
\hline
1\,6\,1\,4\,6 \\
1\,6\,1\,4\,6\ \ \\
1\,6\,1\,4\,6\ \ \ \\
1\,7\,9\,4\ \ \ \ \\
\hline
3\,5\,8\,6\,2\,0\,6
\end{array}
$$

Multiplieur de Braun

- On veut calculer $P = X \cdot Y$.
- On a : $P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 2^{i+j} \cdot x_i \cdot y_i$
- Attention : ne fonctionne que sur des nombres signés.

Schéma au tableau

Multiplieur de Baugh-Wooley 1/2

- Avantage : le multiplieur de Baugh-Wooley gère les nombres non-signés.

$$\begin{aligned} P &= X \cdot Y \\ &= \left(-2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i\right)\left(-y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} 2^i y_i\right) \\ &= 2^{2n-2}x_{n-1}y_{n-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} 2^{i+j}x_i y_j \\ &\quad - x_{n-1}\sum_{i=0}^{n-2} 2^{n+i-1}y_i - y_{n-1}\sum_{i=0}^{n-2} 2^{n+i-1}x_i \end{aligned}$$

On enlève les soustractions :

$$-x_{n-1}\sum_{i=0}^{n-2} 2^{n+i-1}y_i = x_{n-1}\left(-2^{2n-2} + 2^{n-1} + \sum_{i=0}^{n-2} 2^{n+i-1}\overline{y}_i\right)$$

Multiplieur de Baugh-Wooley 2/2

Formule finale : *(~~à apprendre par cœur pour l'examen~~)*

$$\begin{aligned} P \;=\; & -2^{2n-1} + 2^{2n-2}\left(\overline{x}_{n-1} + \overline{y}_{n-1} + x_{n-1}y_{n-1}\right) \\ & + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} 2^{i+j} x_i y_j + 2^{n-1}\left(x_{n-1} + y_{n-1}\right) \\ & + x_{n-1}\sum_{i=0}^{n-2} 2^{n+i-1}\overline{y}_i + y_{n-1}\sum_{i=0}^{n-2} 2^{n+i-1}\overline{x}_i \end{aligned}$$

Schéma au tableau

Diviseur

Version lente

Encore un peu de primaire ?

```
 3217941 | 42
-294     | 7661
 277
-252
 259
-252
  74
 -42
  32
```

On veut calculer A/D. Soit $Q = q_{n-1} \dots q_i \dots q_1 q_0$ le dividende sur n bits et R le reste.
Rappel trivial : $A = Q \cdot D + R$.

One recurrence to rule them all :

$$R_{i+1} = B \cdot R_i - q_{n-1-i} \cdot D$$

avec :

- B : base utilisée (ici 2)
- R_i : i-ème reste partiel de la division
- $R_0 = A$, $R = R_n$

Diviseur

Algorithme

Restoring division

```
1  def r_divide(A, D):
2    Q = [None] * n
3    R = A
4    D = D << n
5    for i in reversed(range(n)):
6      R = 2 * R - D
7      if R >= 0:
8        Q[i] = 1
9      else:
10       Q[i] = 0
11     R = R + D
12   return Concat(*Q), R
```

Comment choisir n ?

Non-restoring division

```
1  def nr_divide(A, D):
2    Q = [None] * n
3    R = A
4    D = D << n
5    for i in reversed(range(n)):
6      if R >= 0:
7        Q[i] = 1
8        R = 2 * R - D
9      else:
10       Q[i] = -1
11       R = 2 * R + D
12   return FixUp(Concat(*Q)), R
```

Comment réparer la sortie ?

Diviseur

Version rapide

Pour aller plus loin :

- Newton–Raphson division
- Goldschmidt division
- Division par une constante (je ne sais plus si c'est vu en cours de compilation)