Architecture et Système

Stefan Schwoon

Cours L3, 2015/16, ENS Cachan

Circuit pour multiplication

Soient $x = (x_{n-1} \cdots x_0)_2$ et $y = (y_{n-1} \cdots y_0)_2$ deux entiers naturels.

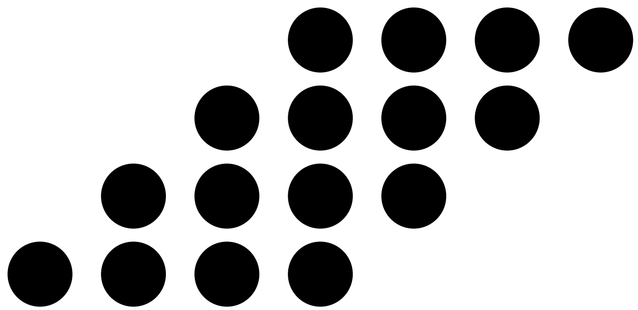
On souhaite construire un circuit efficace pour calculer $z = (z_{2n-1} \cdots z_0)_2$ tel que $z = x \cdot y$.

Technique de multiplication classique :

$$z = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (x_i \cdot y_j \cdot 2^{i+j})$$

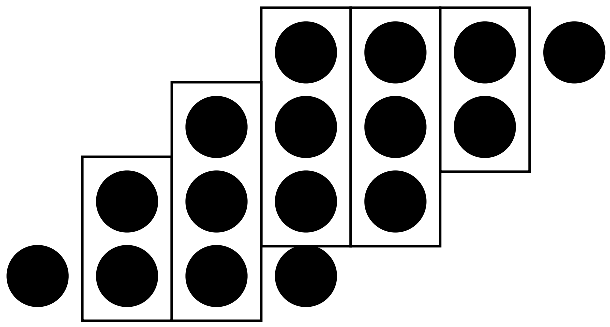
Notons que $x_i \cdot y_j \in \{0, 1\}$, du coup on obtient n^2 bits avec des “poids” différents.

Visualisation (pour $n = 4$), chaque boule reprsent un produit $x_i y_j$:

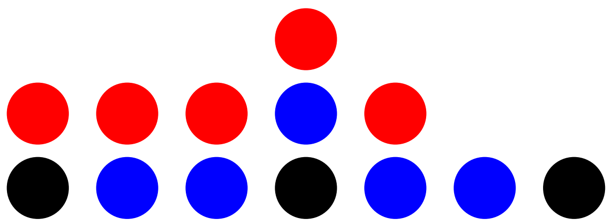


Ayant obtenu les n^2 bits, il faut donc faire l'addition dans les $2n$ colonnes (tout en propagant les retenues). Le nombre maximal de “boules” dans une colonne est de n.

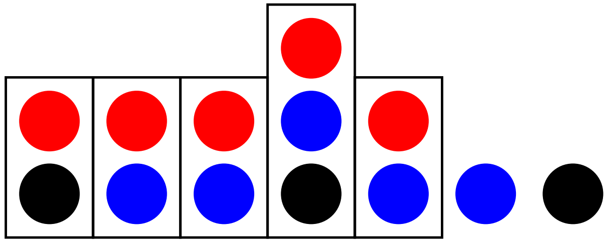
Principe : Successivement réduire les colonnes. On trouve des groupes de deux ou trois “boules” et on fait passer tous les groupes (en parallèle) par des DA ou AC.



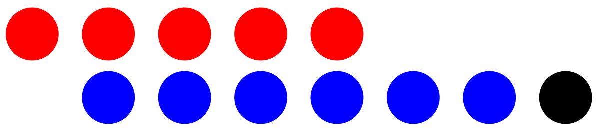
Ensuite on distribue les deux bits résultants (somme et retenue) aux bonnes colonnes :



Et on répète : identifier des groupes de deux ou trois . . .



. . . ce qui donne :



Quand il ne reste qu'au maximum deux boules par colonne, on exècute on addition de deux entiers.

Analyse du multiplicateur

Il faut n^2 portes ET pour obtenir les produits de départ.

Ensuite, on réduit la hauteur des colonnes par un facteur d'environ $2/3$ par étape.

> Une analyse précise montre qu'après $\log_{3/2} n$ étapes on obtient une hauteur d'au plus 4.
>
> Une fois la hauteur réduite à 4, trois étapes supplémentairs sont suffisants.
>
> Du coup, la ˚eduction se fait en profondeur $\mathcal{O}(\log n)$.

Le circuit pour l'addition finale est de taille $\mathcal{O}(n)$ et de profondeur $\mathcal{O}(\log n)$

Du coup, la multiplication entière se fait en taille $\mathcal{O}(n^2)$ et profondeur $\mathcal{O}(\log n)$.

Fonctions logiques sur les mots

Les ordinateurs permettent typiquement d'appliquer un opérateur logique sur tous les bits de deux mots en parallèle :

$$x \circ y = (x_{n-1} \circ y_{n-1}, \cdots, x_0 \circ y_0)_2$$

Exemples :

cas $\circ = \wedge$: souvent utilisé avec des constants, p.ex. $x \wedge 16$ sert à isoler le 4ème bit de x.

cas $\circ = \vee$: fait l'union des bits qui sont 1, p.ex. $x := x \vee 16$ met le quatrième bit de x à 1.

Multiplexeur

Supposons qu'on possède k valeurs en entrée (ou $k = 2^n$), pour $n \geq 1$), ou chacun des valeurs est un mot de m bits. Appelons $x_0, \ldots, x_{k-1}$ ces valeurs.

On souhaite sélectionner l'un de ces valeurs par son indice, p.ex. étant donné la valeur s (codé par n bits), on souhait sortir x_s.

Applications:

Lire un mot dans la mémoire en spécifiant son adresse.

Choisir entre plusieurs sources de données.

Obtenir une valeur dans un tableaux pré-calculé.

Multiplexeur : Exemple

Supposons $m = n = 1$, du coup on possède deux bits x_0, x_1 et un bit de sélection s_0.

Solution: on réalise la fonction $(x_0 \wedge \neg s_0) \vee (x_1 \wedge s_0)$.

Supposons $m = 1, n = 2$, du coup on posséde $x_0, \cdots, x_3$ et $s_1 s_0$:

Dans un premier temps, on évalue s_0 pour sélectionner (en parallèle) entre x_0, x_1 et x_2, x_3.

Dans un deuxième temps, on évalue s_1 pour sélectionner parmi les deux bits choisis avant.

Le problème se géneralise pour $n > 2$ avec un circuit de profondeur $\mathcal{O}(n)$.

Pour $m > 1$: Faire le tout en parallèle pour tous les bits en entrée.

Décodeur

On possède une valeur s codé par n bits et 2^n sorties $y_0, \ldots, y_{2^n-1}$.

On souhaite sortir $y_s = 1$ et $y_{s'} = 0$ pour tout $s' \neq s$.

Solution (facile) : Pour tout $0 \leq s' < 2^n$, on construit la conjonction de n bits.

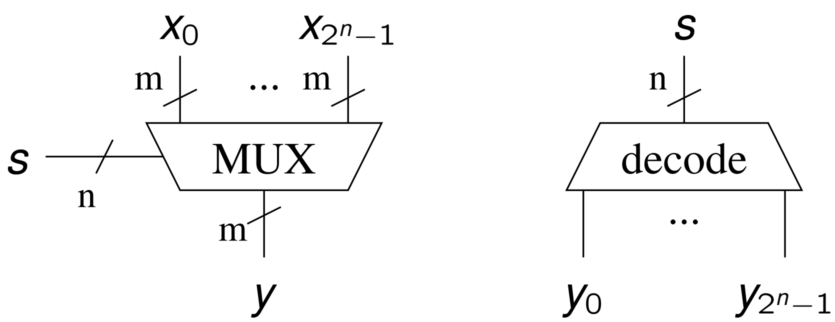
Applications :

Sélectionner une cellule spécifique dans la mémoier pour y mettre une valeur.

Sélectionner un périphérique (parmi plusieurs) pour y transmettre une valeur.

Symboles

Les symboles typiquement utilisés pour les multiplexeurs et décodeurs :



Ici, les barres diagonales symbolisent que le fil transfère un mot de la taille indiquée à côté.

Mémoire

Verrous et bascules

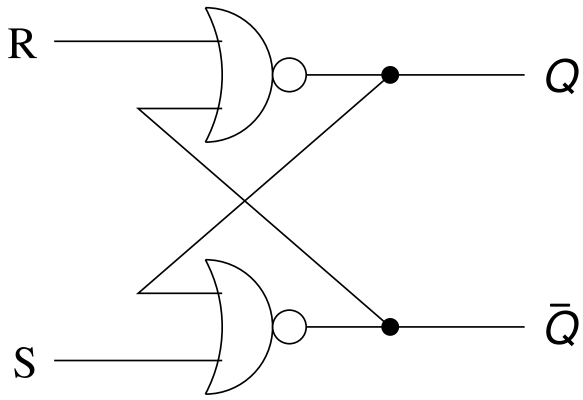
Circuits utilisés pour stocker des bits:

- les circuits non-temporisés sont appelés verrous (*latch* en anglais)
- les circuits temporisés sont appelés bascules (*flip-flop* en anglais)

Il y a plusieurs types de verrous et bascules, selon des besoins spécifiques.
Quelques éléments typiques :

- contrôles : aucun effet quand tous les fils de contrôle sont 0
- signal d'horloge (bascules seulement)
- deux sorties Q et $\bar{Q}$, la valeur du bit et son complément

Verrou RS



controls R (reset) and S (set):

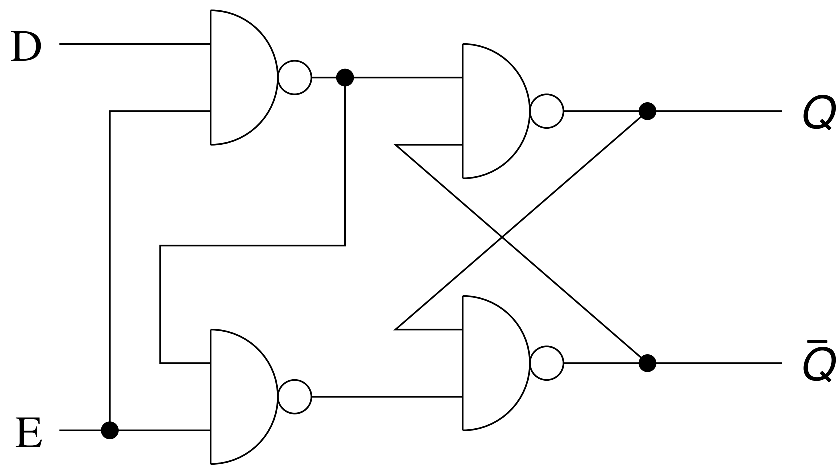
$R = S = 0$: pas de changement

$R = 1$, $S = 0$: sortie Q devient 0

$R = 0$, $S = 1$: sortie Q devient 1

$R = S = 1$: 0 pour les deux sorties, non-defini quand on revient sur $R = S = 0$

Verrou D



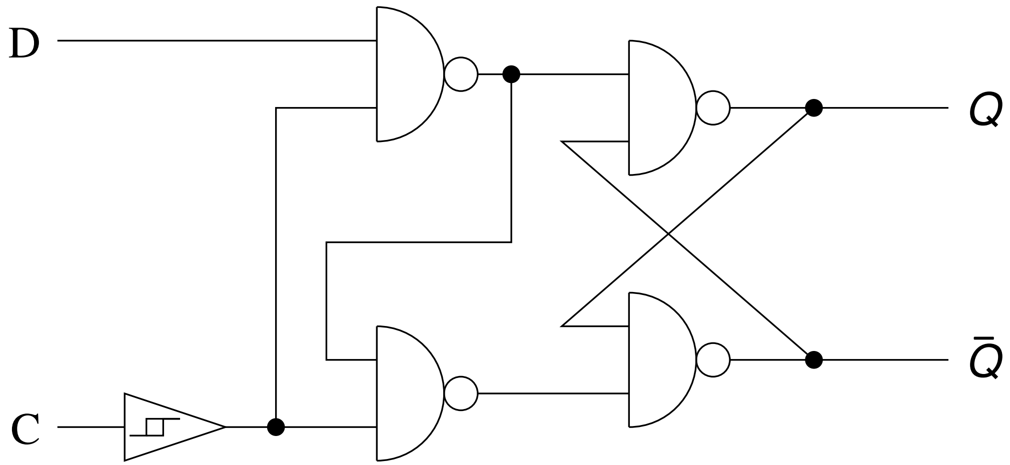
contrôles D (data) et E (enable):

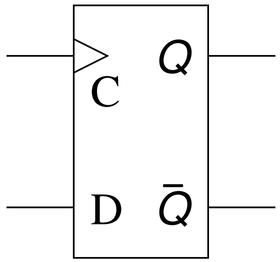
$E = 0$: stable

$E = 1$: Q devient D

Bascule D

Reprenons le verrou D, mais en rempla**c**cant *E* par un signal oscillant. Dans ce cas, *D* est pris en compte dans certains moments spécifiques.





Dans les diagrams, *C* est typiquement indiqué par un triangle.

Quand le signal C est partagé parmi tous les bascules du système, ceci permet de synchroniser les calculs dans le système entier.

Le délai entre les “`1`” de C doit être suffisamment long pour que tous les calculs en parallèle puissent réussir.

En plus, les “`1`” doivent être suffisamment court pour éviter qu’un valeur intermédiaire et pris en compte.