Architecture et Système

Stefan Schwoon

Cours L3, 2015/16, ENS Cachan

Circuits logiques

Plusieurs niveaux d'abstraction:

- couche physique (transistor)
- portes logiques
- registres
- processeur

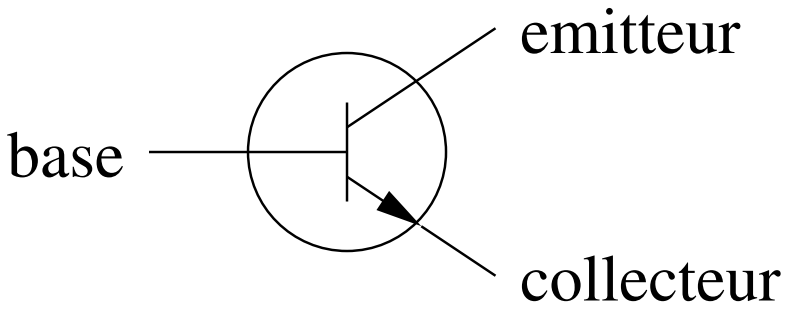
Aspects statiques et dynamiques (temps, stockage)

Couche physique

Éléments de base: plusieurs types de transistor.

Petit historique : développement à partir de 1925, utilisation dans les ordinateurs à partir des mi-1950s.

Exemple : transistor à effet de champ (en anglais : FET, field effect transistor):



Flux d'électrons possible entre émetteur et collecteur quand le voltage de base excède un certain seuil.

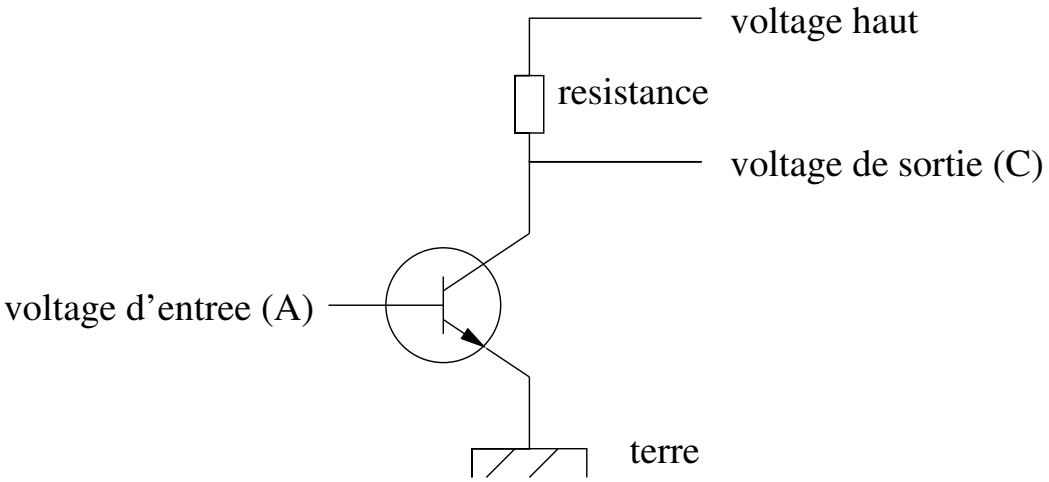
Transistor réalisant une manipulation logique

Le comportement du transistor permet de réaliser une logique binaire :

soit le voltage est en-dessous du seuil, alors le flux E$\rightarrow$C est interrompu;

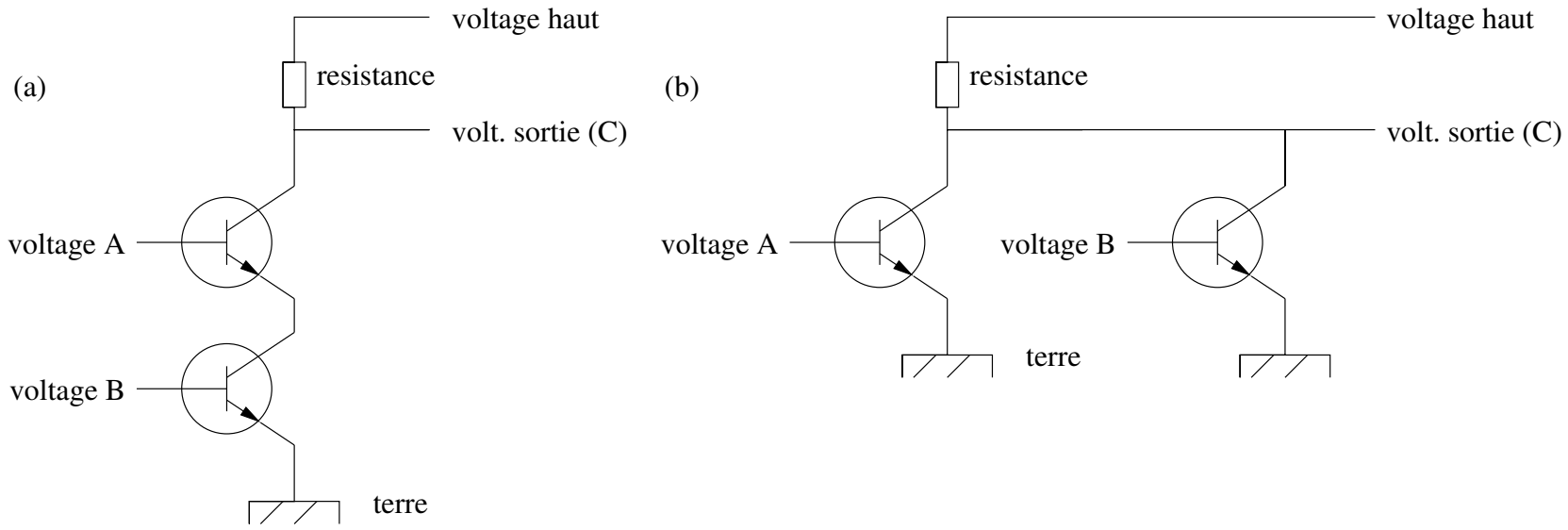
soit le voltage est en-dessouus du seuil, alors il y a un flux.

Utilisation d'un transistor pour la négation logique :



Opérations binaires

Portes NON-ET (a) et NON-OU (b) réaliser avec des transistors:



Importance de NON-ET et NON-OU

Les opérateurs NON-ET (NAND en anglais) et NON-OU (NOR en anglais) sont importants pour deux raisons :

Ils sont facilement réalisable avec des transistors.

Toute autre fonction logique peut être exprimé avec soit NON-ET, soit NON-OU :

$\neg A \equiv A \barwedge A$;

$A \wedge B \equiv (A \barwedge B) \barwedge (A \barwedge B)$;

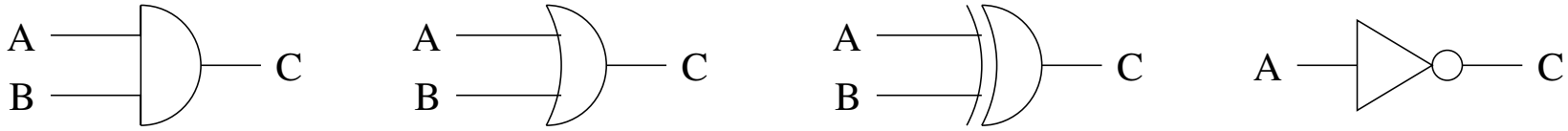
$A \vee B \equiv (A \barwedge A) \barwedge (B \barwedge B)$.

Désormais, pour faire abstraction des détails physiques, on représentera les circuits en forme de diagrammes avec des portes logiques qui traitent des bits avec valeurs 0 et 1. of individual bits.

Portes logiques

Diagrammes pour NON-ET (à gauche) et NON-OU (à droit):

Diagrammes derivés, réalise p.ex. par la combinaison de plusieurs portes NON-ET/OU.



ET OU XOR NON

Complexité des circuits

Dans un ordinateur, on traite des circuits logiques qui réalisent des fonction assez complexes et avec pas mal de bits en entrée (p.ex., l'addition).

On s'intéresse donc à optimiser les circuits par rapport à :

leur taille – minimiser le nombre de transistors utilisés = minimiser le coût du circuit ;

leur profondeur – le chemin le plus long (en nombre de transistors) qu'un signal doit traverser détermine le délai pour calculer le résultat, compte tenu du fait que chaque transistor dispose d'un certain délai pour mettre à jour le voltage en sortie après un changement du voltage en entrée.

Minimiser les deux au même temps - souvent contradictoire !

Mésures de complexité

Dans le suivant on s'intéressera à des fonctions avec n bits en entrée ou n est variable (mais en pratique souvent une puissance de 2).

On s'intéresse à la taille et complexité d'un circuit réalisant une certaine fonction logique pour n grand, c'est à dire la *complexité asymptotique*.

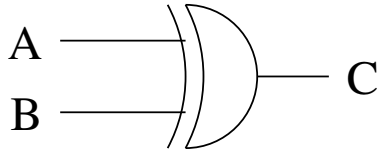
Objectif typique: taille $\mathcal{O}(n)$, profondeur $\mathcal{O}(\log n)$.

Remarques : l'analyse asymptotique nous permet certaines libertés en construisant les circuits.

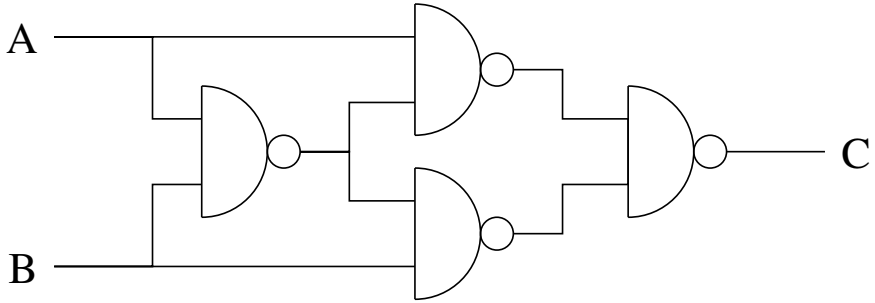
On se permettra des portes OU, ET, NON etc ; la complexité mésuree en nombre de transistor n'augmente que par un facteur constant.

Dans une même optique, on peut se permettre des porte ET/OU avec plus que deux valeurs en entrée tant que le nombre d'entrées reste indépendent de n.

Exemple : réalisation de OU exclusif (XOR)



Réalisable par plusieurs portes NON-ET :



Premier diagramme: taille/profondeur 1

Deuxième diagramme: taille 4, profondeur 3

Nombre de transistors: 8

Fonctions arithmetiques: demi-additionneur

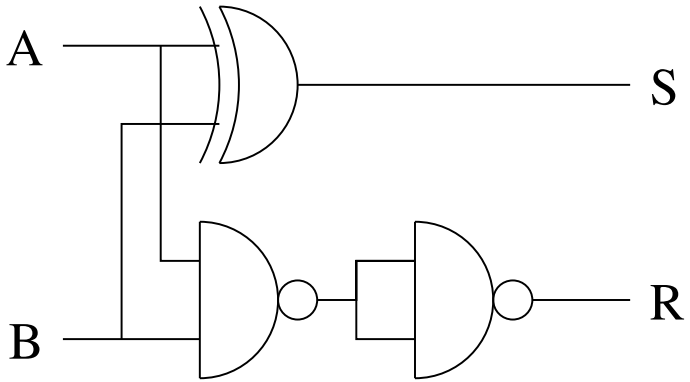
Fonction réalisée par un demi-additionneuradder:

deux bits en entrée, A et B;

deux bits en sortie, R (la *retenue*) et S (la *somme*);

résultat souhaité : $(R.S)_2 = A + B$ (ou . dénote la concaténation)

Réalisation potentielle :

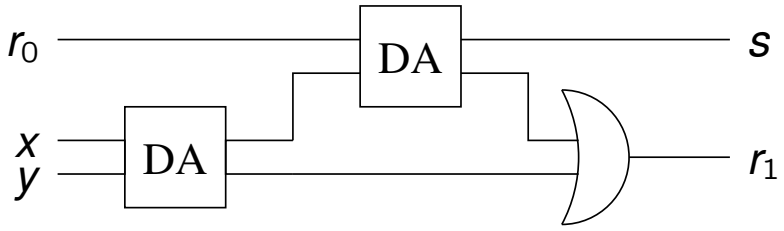


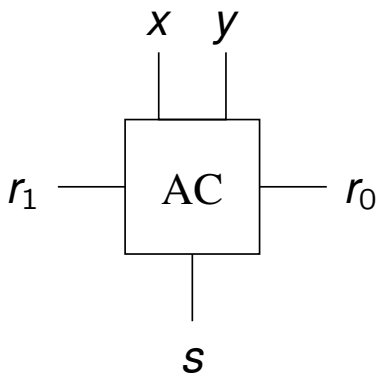
Additionneur complet

Un Additionneur complet calcule l'addition de trois bits x, y et r_0, ou r_0 représente la retenue d'une autre addition.

Résultat souhaité : $(r_1.s)_2 = x + y + r_0$

Réalisation à l'aide des DA :



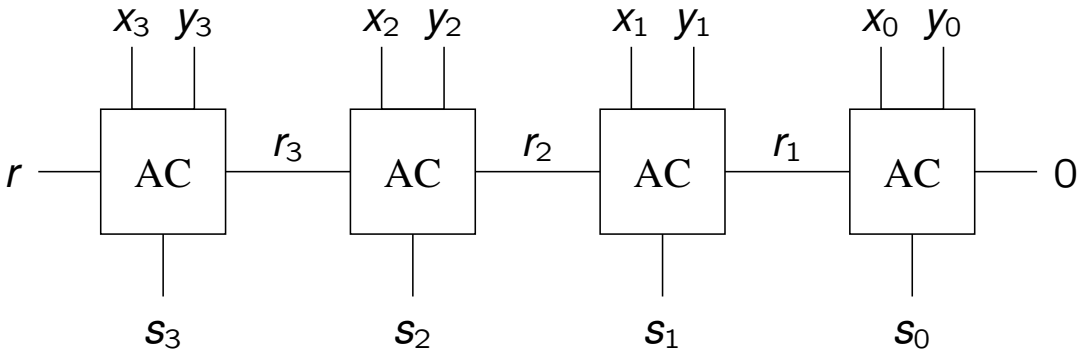


Addition de deux entiers

Supposons que nous avons deux entiers (non-négatifs) à deux bits :
$x = (x_3.x_2.x_1.x_0)_2$ and $y = (y_3.y_2.y_1.y_0)_2$.

On souhaite calculer la somme $s = x + y$ en forme $s = (r.s_3.s_2.s_1.s_0)_2$.

Réalisation avec enchaînement de quatre AC :



La géneralisation de principe à des vecteurs de n bits donnerait un circuit avec taille et profondeur $\mathcal{O}(n)$.

La profondeur est mauvaise car un ordinateur deviendrait deux fois plus lent si en augmente la taille des registres, p.ex. en passant de 32 à 64.

On va étudier une solution avec profondeur *logarithmique*.

Additioneur basé sur propagation/géneration

Soit $0 \leq i \leq j < n$. On considére les bits aux positions i à j dans les vecteurs x et y en entrée :

> le bloc $i..j$ génèrate une retenue si $r_{j+1} = 1$ *indépendemment* des bits à d'autres positions ;
>
> le bloc $i..j$ propage la retenue si $r_i = 1$ implique $r_{j+1} = 1$.

Calculer ces deux effets à partir de $(x_i, y_i) \cdots (x_j, y_j)$:

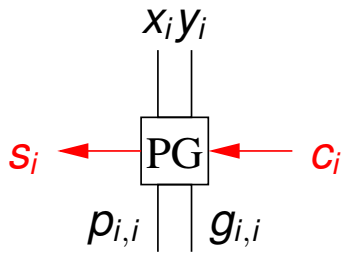
Si $i = j$, alors $g_{i,j} = 1$ ssi $x_i + y_i = 2$ et $p_{i,j} = 1$ ssi $x_i + y_i \geq 1$.

Sinon, soit $i \leq k < j$, alors:

$g_{i,j} = g_{k+1,j} \vee (g_{i,k} \wedge p_{k+1,j})$ et $p_{i,j} = p_{i,k} \wedge p_{k+1,j}$

Du coup, $r_{j+1} = g_{i,j} \vee (r_i \wedge p_{i,j})$.

Symbole pour un AC avec calcul des bits p/g en plus:



Symbole pour calculer la combinaison des bits p/g de deux blocs adjacents :

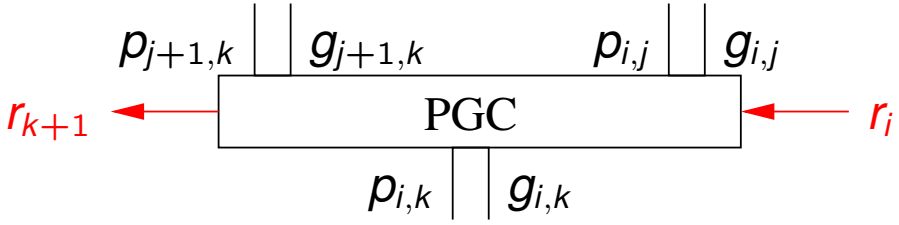
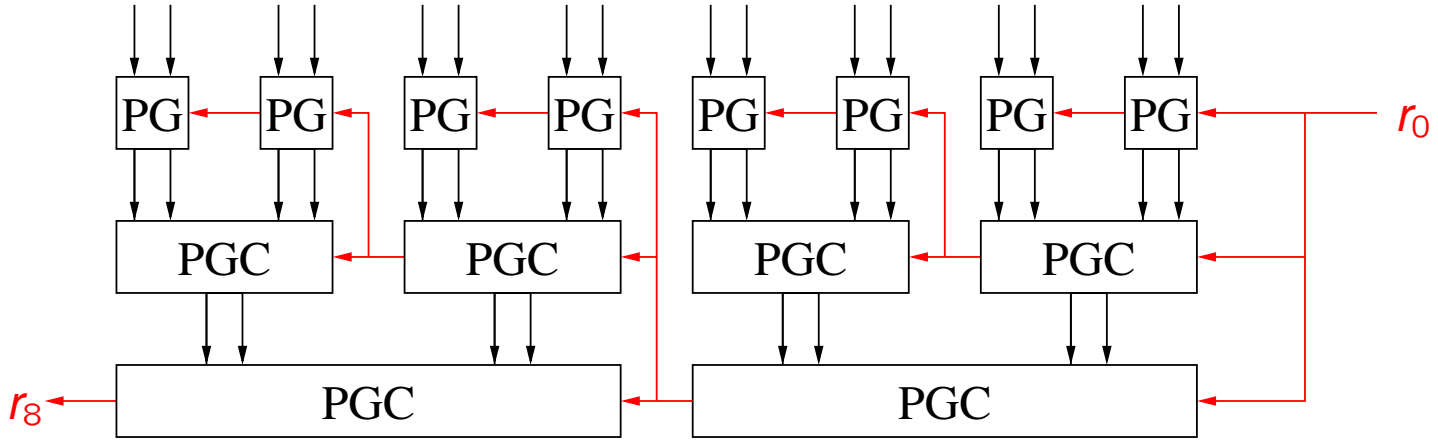


Diagramme simplifié pour $n = 8$, seule la propagation des retenue est détaillée :



Pour comprendre la profondeur du circuit, notons que

- le calcul des p/g est indépendent des valeurs de retenues ;
- la retenue d'un bloc PGC devient stable dès qu'on connaît les p/g et la retenue en entrée.

Du coup, le plus long chemin pour calculer tout les retenues est d'environ $2 \log n$.

D'autres fonctions logiques

Subtraction, multiplication, division, ...

Comparaison, tester pour zero, ...

AND, OR, XOR sur des mots ...

Décalage d'un mot

Multiplexeur: 2^n bits en entrée, en sélectionner un comme sortie

Décodeur: entree i (representée par n bits), sortie $y_i = 1$ et $y_j = 0$ pour $0 \leq j < 2^n, j \neq i$