

Architecture et Système

Stefan Schwoon

Cours L3, 2022/2023, ENS Cachan

Circuits logiques

Composants pour construire un ordinateur, physiquement réalisés par des transistors, se comportant comme des portes logiques.

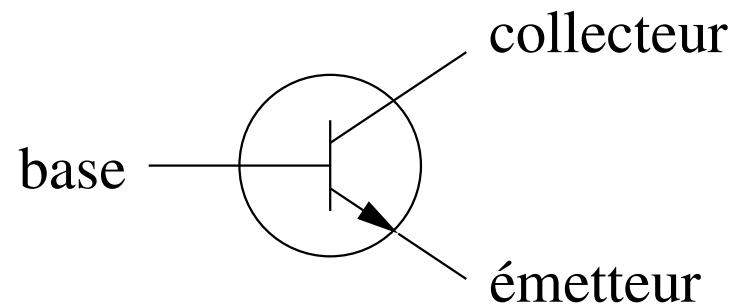
Questions étudiées dans la suite:

- Circuits statiques (entrée/sortie): fonctions logiques et arithmétiques
- Aspects dynamiques : stockage, mémoire, transfert de données, microprogrammation

Transistor

Découverte théorique 1925, utilisation dans les ordinateurs à partir des mi-1950s.

Il existe une grande variété de transistors. Exemple : transistor bipolaire NPN



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

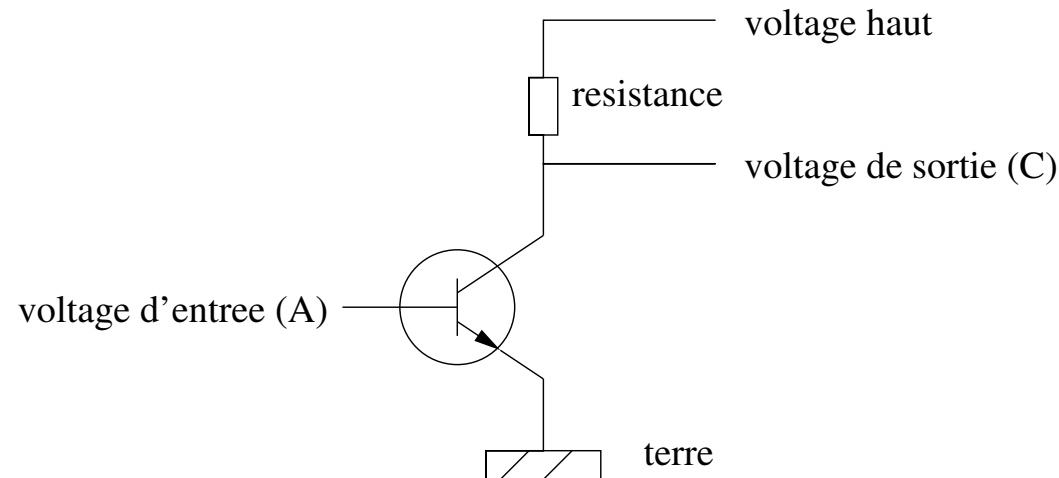
Transistor réalisant une manipulation logique

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;

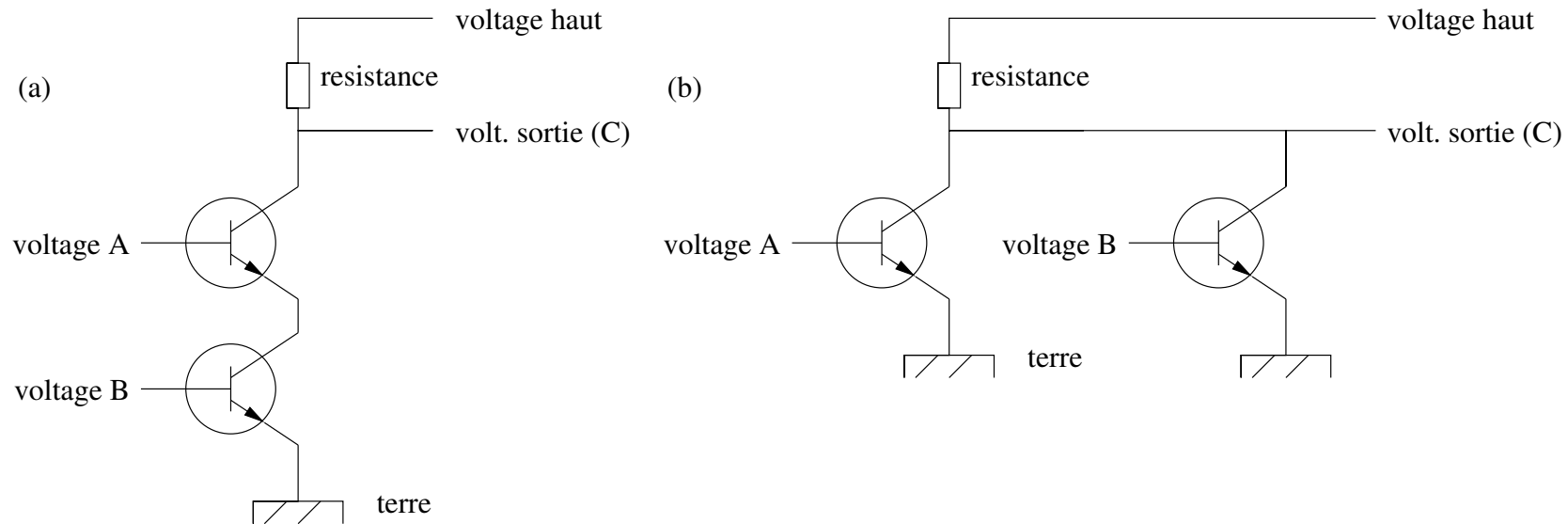
soit le voltage est en-dessus du seuil, alors il y a un flux.

Utilisation d'un transistor pour la négation logique :



Opérations binaires

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



Interprétation : voltage haut $\hat{=}$ 1, voltage bas $\hat{=}$ 0

Importance de NON-ET et NON-OU

Les opérateurs NON-ET et NON-OU (NAND/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 :

$$\neg A \equiv A \bar{A};$$

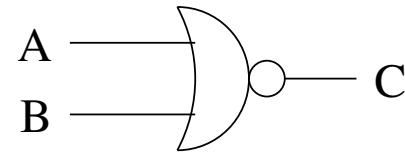
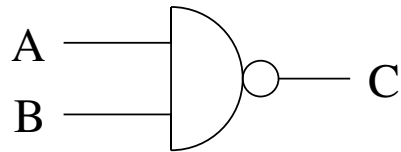
$$A \wedge B \equiv (A \bar{B}) \bar{(A \bar{B})};$$

$$A \vee B \equiv (A \bar{A}) \bar{(B \bar{B})}.$$

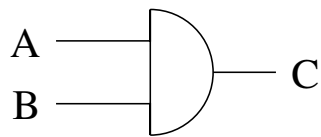
Désormais, pour faire abstraction des détails physiques, on représentera les circuits en forme de diagrammes avec des **portes logiques** qui traitent des bits avec valeurs 0 et 1.

Portes logiques

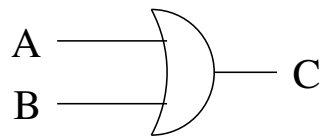
Diagrammes pour NON-ET (à gauche) et NON-OU (à droite):



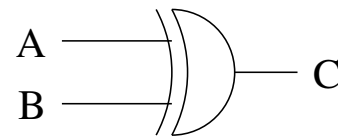
Diagrammes dérivés, réalise p.ex. par la combinaison de plusieurs portes NON-ET/OU.



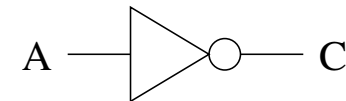
ET



OU



XOR



NON

Complexité des circuits

Dans un ordinateur, on traite des circuits logiques qui réalisent des fonctions assez complexes et avec beaucoup de bits en entrée (p.ex., l'addition sur des entiers de 32 ou 64 bits).

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

leur **taille** – minimiser le nombre de transistors utilisés, c'est à dire 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 réagir aux changements du voltage en entrée.

Minimiser les deux au même temps - parfois contradictoire !

Mésures de complexité

Dans le suivant on s'intéresse à des fonctions avec n bits en entrée où n est variable (mais typiquement une puissance de 2).

On s'intéresse particulièrement au comportement des circuits quand n grandit, c'est à dire la *complexité asymptotique*.

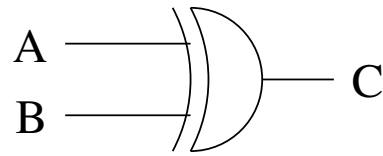
Objectif typique: taille $\mathcal{O}(n)$, profondeur $\mathcal{O}(\log n)$.

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

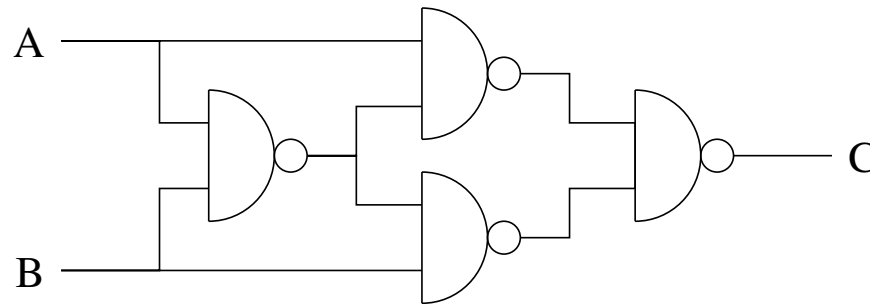
On se permettra des portes OU, ET, NON etc ; la complexité mesurée en nombre de transistors n'augmente que d'un facteur constant.

On peut même se permettre des portes ET/OU avec plus que deux valeurs en entrée, dans la mesure où le nombre d'entrées reste indépendant 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 arithmétiques: demi-additionneur

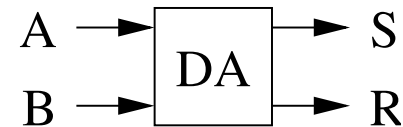
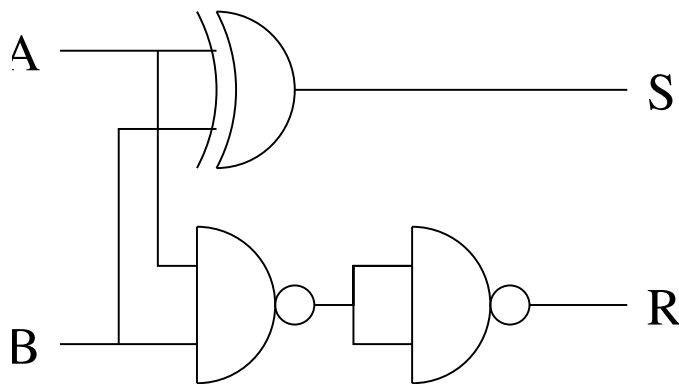
Fonction réalisée par un **demi-additionneur**:

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 :

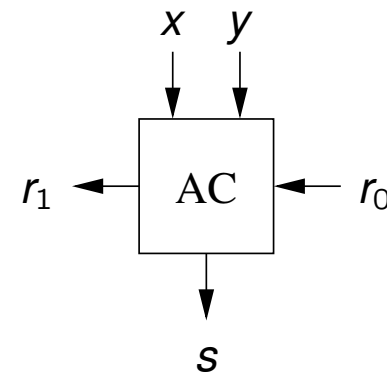
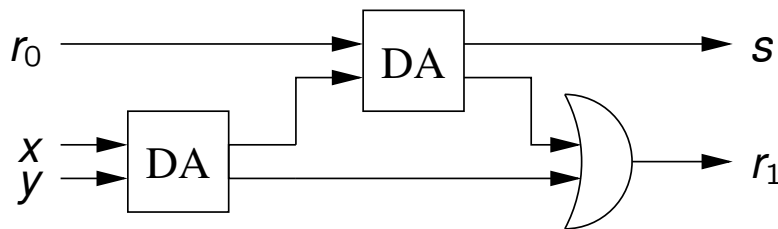


Additionneur complet

Un **additionneur complet** réalise l'addition de trois bits x , y et r_0 , où 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 :



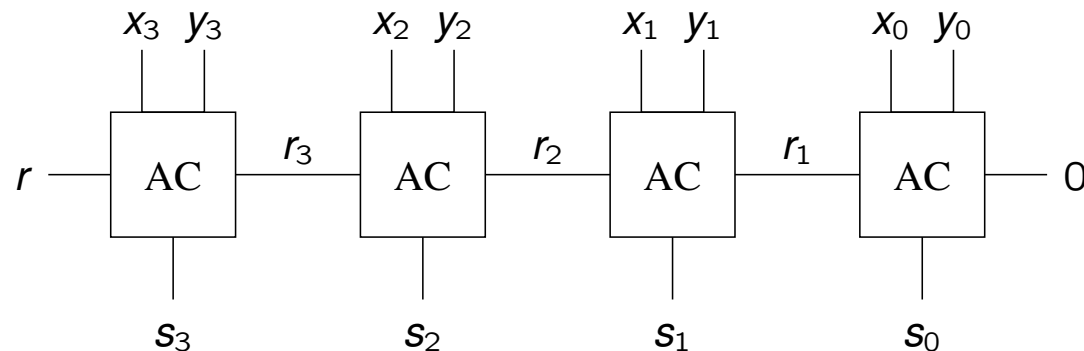
Addition de deux entiers

Supposons que nous avons deux entiers (non-négatifs) à deux bits :

$$x = (x_3 \cdot x_2 \cdot x_1 \cdot x_0)_2 \text{ and } y = (y_3 \cdot y_2 \cdot y_1 \cdot y_0)_2.$$

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

Réalisation avec enchaînement de quatre AC :



La généralisation du principe d'enchaînement 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 on augmente la taille des registres, p.ex. en passant de 32 à 64.

On va étudier une solution avec profondeur *logarithmique*.