Circuits logiques combinatoires.

1. Généralités
   - Fonctions booléennes
   - Circuits logiques combinatoires
2. Quelques circuits logiques de base
   - Décodeur
   - Multiplexeur
   - Additionneur
3. Optimisation de circuits
   - Délai d'un circuit logique
   - Additionneur à propagation de retenue anticipée

Pour manipuler des informations en binaire, on assimile les deux bits 0 et 1 aux deux valeurs de vérité de l'algèbre de Boole : on met au point des expressions booléennes, puis on implante ces expressions par des circuits logiques.

On distingue deux types de circuits logiques :

- Les *circuits combinatoires*, dans lesquels les signaux de sortie ne dépendent que des signaux d'entrées du circuit à l'instant considéré.
  Ex : la plupart des circuits pour l'addition de nombres binaires.
- Les *circuits séquentiels*, disposant de propriétés de mémorisation, et qui présentent donc un certain nombre d'états : les signaux de sortie dépendent des signaux d'entrées à l'instant considéré, mais aussi de l'état du circuit.
  Ex : mémoires, registres, unité de commande. . .

On s'intéressera dans un premier temps aux circuits combinatoires, permettant par exemple de concevoir l'UAL d'un processeur.

Plan

1. Généralités
   - Fonctions booléennes
   - Circuits logiques combinatoires
2. Quelques circuits logiques de base
   - Décodeur
   - Multiplexeur
   - Additionneur
3. Optimisation de circuits
   - Délai d'un circuit logique
   - Additionneur à propagation de retenue anticipée

Fonctions booléennes

L'*algèbre de Boole* $\mathbb{B} = \{0, 1\}$ est munie de trois opérations :

• une opération unaire involution, la *négation* (NOT) ;

• de deux opérations binaires commutatives, associatives, distributives l'une par rapport à l'autre et possédant chacune un élément neutre :

- la *conjonction* (AND), avec pour élément neutre 1,
- la *disjonction* (OR), avec pour élément neutre 0.

Si a et b sont deux variables booléennes, on notera par commodité :

$$\mathrm{NOT}(a) = \overline{a}, \quad \mathrm{AND}(a, b) = ab, \quad \mathrm{OR}(a, b) = a + b.$$

a	b	$\overline{a}$	ab	$a + b$
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Formulaire

- éléments neutres : $a + 0 = a, \quad a{\cdot}1 = a$
- éléments absorbants : $a + 1 = 1, \quad a{\cdot}0 = 0$
- idempotance : $a + a = a, \quad a{\cdot}a = a$
- tautologie/antilogie $a + \overline{a} = 1, \quad a{\cdot}\overline{a} = 0$
- commutativité : $a + b = b + a, \quad ab = ba$
- distributivité : $a + (bc) = (a + b)(a + c), \quad a(b + c) = ab + ac$
- associativité : $a + (b + c) = (a + b) + c = a + b + c,$

  $a(bc) = (ab)c = abc$
- lois de Morgan : $\overline{ab} = \overline{a} + \overline{b},$

  $\overline{a + b} = \overline{a}{\cdot}\overline{b}$
- autres relations : $a + (ab) = a, \quad a + (\overline{a}b) = a + b,$

  $a(a + b) = a, \quad (a + b)(a + \overline{b}) = a$

On peut former en utilisant les opérations de base des *expressions booléennes*.
On appelle *littéral* une variable booléenne ou sa négation.

Ex : si a, b, c sont 3 variables booléennes, on peut écrire l'expression booléenne $\overline{a}b + c\overline{b}$. Cette expression comporte 4 littéraux.

Toute fonction de $\mathbb{B}^n$ dans $\mathbb{B}$ s'exprime par une expression booléenne.
Toute expression booléenne à n variables représente une fonction de $\mathbb{B}^n$ dans $\mathbb{B}$.

Ex : On considère la fonction $f : \mathbb{B}^2 \to \mathbb{B}$ définie par la table de vérité suivante :

a	b	$f(a, b)$
0	0	1
0	1	0
1	0	0
1	1	0

$f(a, b) = \overline{a}\overline{b}$ est une expression booléenne représentant la fonction f.

Il n'y a pas unicité de la représentation d'une fonction par une expression...
Ex : si $f(a,b) = \overline{a}\overline{b}$, alors on a aussi $f(a,b) = \overline{a+b}$.

On distingue donc deux *formes normales* :

- La forme normale disjonctive : disjonction de conjonctions de littéraux.
  Ex : $\overline{a}b\overline{c} + \overline{a}\overline{b} + \overline{a}\overline{b}$.

- La forme normale conjonctive : conjonction de disjonctions de littéraux.
  Ex : $(\overline{a} + b + \overline{c})\cdot(\overline{a} + \overline{b} + c)\cdot(a + \overline{b} + \overline{c})$.

Si la fonction et non toujours vraie ou toujours fausse, on peut assurer l'unicité de ces deux formes : chaque variable doit apparaître au plus une fois dans chaque terme/facteur, et les termes/facteurs ne se répétent pas.

Pour mettre f sous forme normale disjonctive :

• pour chaque ligne où f prend pour valeur 1, on forme une conjonction prenant pour valeur 1 *uniquement* sur l'affectation des variables de cette ligne.

a	b	c	$f(a,b,c)$	
0	0	0	1	$\rightarrow \bar{a}\bar{b}\bar{c}$
0	0	1	0	
0	1	0	1	$\rightarrow \bar{a}b\bar{c}$
0	1	1	0	
1	0	0	1	$\rightarrow a\bar{b}\bar{c}$
1	0	1	1	$\rightarrow a\bar{b}c$
1	1	0	0	
1	1	1	0	

• On forme la disjonction des termes ainsi obtenus :

$$f(a,b,c) = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}\bar{c} + a\bar{b}c.$$

Pour mettre f sous forme normale conjonctive :

- on écrit $\overline{f(a,b,c)}$ sous forme normale disjonctive.

a	b	c	$f(a,b,c)$	$\overline{f(a,b,c)}$	
0	0	0	1	0	
0	0	1	0	1	$\rightarrow \overline{a}\overline{b}c$
0	1	0	1	0	
0	1	1	0	1	$\rightarrow \overline{a}bc$
1	0	0	1	0	
1	0	1	1	0	
1	1	0	0	1	$\rightarrow ab\overline{c}$
1	1	1	0	1	$\rightarrow abc$

$$\overline{f(a,b,c)} = \overline{a}\overline{b}c + \overline{a}bc + ab\overline{c} + abc.$$

- En utilisant les lois de Morgan, on obtient la forme normale conjonctive :

$$f(a,b,c) = (a+b+\overline{c})\cdot(a+\overline{b}+\overline{c})\cdot(\overline{a}+\overline{b}+c)\cdot(\overline{a}+\overline{b}+\overline{c}).$$

D'autres fonctions utiles :

- **Le XOR.** On note aussi $XOR(a, b) = a \oplus b$:

$a \oplus b = 1$ ssi une seule des deux variables a et b a la valeur 1.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Le XOR peut s'exprimer à l'aide de AND, OR et NOT :

$$a \oplus b = (a + b)\overline{ab} = (a + b)(\bar{a} + \bar{b}) = \bar{a}b + a\bar{b}.$$

L'opérateur XOR est commutatif et associatif.

- **Le NAND et le NOR :**

a	b	$\overline{ab}$	$\overline{a+b}$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

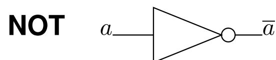
- $\text{NAND}(a, b) = \overline{ab} = \bar{a} + \bar{b}$.
- $\text{NOR}(a, b) = \overline{a + b} = \bar{a}\bar{b}$.

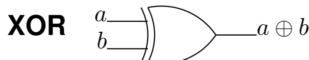
Circuits logiques combinatoires

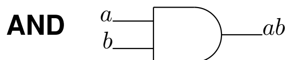
Un *signal logique* est un dispositif physique pouvant transmettre une valeur de vérité d'un endroit à un autre (donc aussi un bit), et sera représenté par un trait.

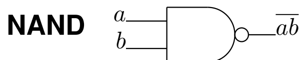
Vu de l'extérieur, un *circuit logique* présente des signaux d'entrée et de sortie : chaque signal de sortie est une fonction logique des signaux d'entrée.

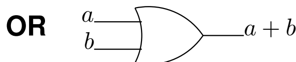
Les *portes logiques* sont les briques de base pour la réalisation de circuits logiques plus complexes :

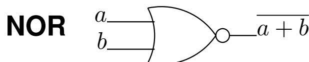










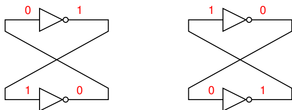


Définit de façon récursive, un circuit logique combinatoire (CLC) peut être :

- une porte de base,
- un fil,
- la juxtaposition de deux CLCs l'un à côté de l'autre,
- obtenu en connectant les sorties d'un CLC aux entrées d'un autre CLC,
- obtenu en connectant entre elles deux entrées d'un CLC.

Cette définition interdit :

- de faire des cycles, car cela permettrait des situations mal définies, *e.g.*,



- de connecter des sorties entre elles (si une sortie est 1 et l'autre 0 ?)

Plan

1 Généralités
- Fonctions booléennes
- Circuits logiques combinatoires

2 Quelques circuits logiques de base
- Décodeur
- Multiplexeur
- Additionneur

3 Optimisation de circuits
- Délai d'un circuit logique
- Additionneur à propagation de retenue anticipée

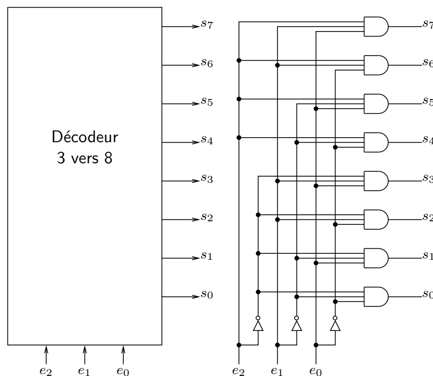
Décodeur

Un *décodeur n vers 2^n* est un circuit qui présente :

- n entrées e_i, qui forment l'entier $(e_{n-1} \dots e_0)_2$;
- 2^n sorties s_i, indicées de 0 à $2^n - 1$.

La seule ligne de sortie active est la ligne $s_{(e_{n-1} \dots e_0)_2}$.

Ex : Décodeur 3 vers 8.



Multiplexeur

Un *multiplexeur 2^n vers 1* est un circuit qui présente :

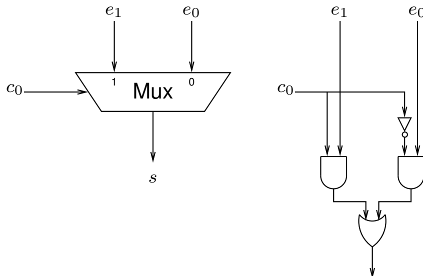
- 2^n entrées e_i indicées de 0 à $2^n - 1$;
- n lignes de sélection, qui forment l'entier $(c_{n-1} \dots c_0)_2$;
- 1 sortie s.

Lorsque les lignes de sélection forment un entier $(c_{n-1} \dots c_0)_2$,

$$s = e_{(c_{n-1} \dots c_0)_2}.$$

Une des entrées est sélectionnée en fonction des lignes de sélection.

Ex : Multiplexeur 2 vers 1.

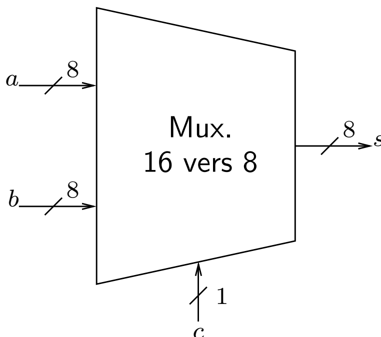


Un *multiplexeur $k{\cdot}2^n$ vers k* est un circuit qui présente :

- $k{\cdot}2^n$ entrées et n lignes de sélection ;
- k signaux de sortie.

Un tel multiplexeur sélectionne k signaux parmi les $k{\cdot}2^n$ signaux d'entrée.

Ex : Multiplexeur 16 vers 8 :



Exercice : comment réaliser un mux. 8 vers 4 avec des mux. 2 vers 1 ?

Additionneur

Pour la conception d'additionneurs, une brique qui peut être utilisée est le *demi-additionneur* ou *half adder* :

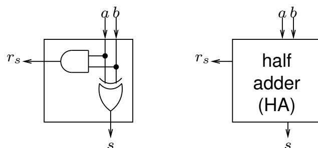
- entrées : deux bits à sommer a et b ;
- sorties : un bit de somme s et un bit de retenue sortante r_s.

a	b	s	r_s
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On constate que

$$s = a \oplus b, \quad \text{et} \quad r_s = a \cdot b.$$

On en déduit le circuit suivant :



Un demi-additionneur ne peut pas prendre en compte de retenue entrante. . .

Intéressons nous donc à un additionneur 1 bit complet ou *full adder* :

- entrées : deux bits à sommer a et b, et un bit de retenue entrante r_e ;
- sorties : un bit de somme s et un bit de retenue sortante r_s.

a	b	r_e	s	r_s
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

On constate que

$$s = \overline{a}(b \oplus r_e) + a(\overline{b \oplus r_e}).$$

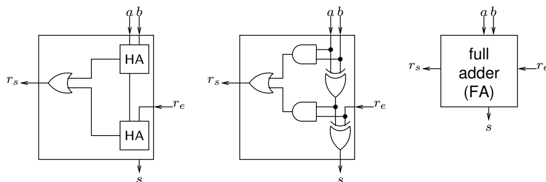
Comme $x \oplus y = \overline{x}y + x\overline{y}$, on obtient

$$s = a \oplus (b \oplus r_e) = a \oplus b \oplus r_e.$$

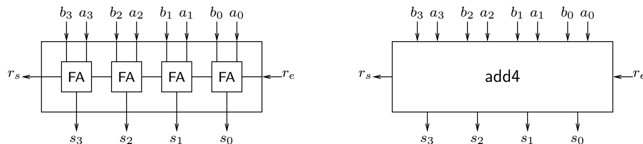
D'autre part,

$$\begin{aligned} r_s &= \overline{a}br_e + a\overline{b}r_e + ab\overline{r_e} + abr_e \\ &= (\overline{a}b + a\overline{b})r_e + ab(\overline{r_e} + r_e) \\ &= (a \oplus b)r_e + ab. \end{aligned}$$

On en déduit le circuit d'un additionneur 1 bit complet ou *full adder* :



On peut enchaîner *k full adders* de manière à obtenir un additionneur de deux entiers naturels de *k* bits. Voici par exemple un additionneur 4 bits :



Ici, on propage la retenue comme on le fait en posant l'opération « à la main » : on verra par la suite qu'il est possible de concevoir des circuits plus « rapides ».

Plan

1. Généralités
   - Fonctions booléennes
   - Circuits logiques combinatoires
2. Quelques circuits logiques de base
   - Décodeur
   - Multiplexeur
   - Additionneur
3. Optimisation de circuits
   - Délai d'un circuit logique
   - Additionneur à propagation de retenue anticipée

Il existe beaucoup de circuits différents pour une fonction booléenne f : comment en choisir un d'après certains critères ?

On s'intéresse souvent au nombre de portes logiques, représentatif de la surface occupée par le circuit. Mais il existe d'autres critères, comme le délai ou la consommation d'énergétique.

Les algorithmes d'optimisation connus sont exponentiels en le nombre de variables : on ne trouve pas toujours un circuit optimal en un temps praticable.

Pour le délai, on va voir sur un exemple qu'il est possible de le réduire en introduisant plus de parallélisme dans le circuit initial.

Délai d'un circuit logique

Toute réalisation physique d'un circuit a un certain *délai* de stabilisation τ, défini comme la durée qui s'écoule entre un changement des entrées (instant t) et la stabilisation des sorties dans l'état correspondant (instant $t + \tau$). Les sorties peuvent passer par des états transitoires pendant l'intervalle $[t, t + \tau]$.

Chaque porte logique élémentaire présente un délai.

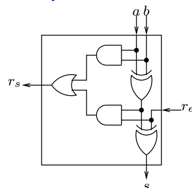
Pour déterminer le délai d'un circuit, il faut :

- calculer le délai associé à chaque chemin d'une entrée vers une sortie,
- identifier un chemin dont le délai est maximal : c'est un *chemin critique*.

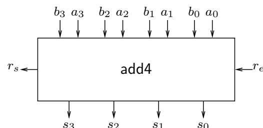
Le délai d'un circuit est donc égal au délai d'un chemin critique.

Rem : Il n'y a pas forcément unicité du chemin critique.

Additionneur à propagation de retenue anticipée

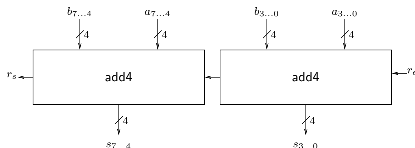


Si on attribue le même délai de 1 ut à toutes les portes logiques, le délai d'un *full adder* est de 3 ut. De plus le délai de r_e à r_s est de 2 ut.



Dans un additionneur 4 bits, le chemin critique est celui allant de r_e à r_s : délai de 8 ut.

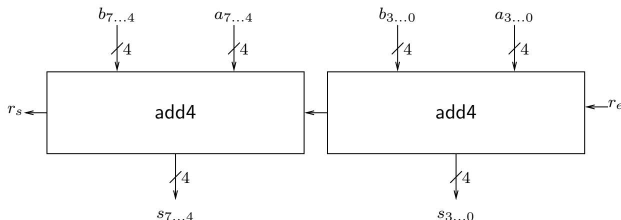
Dans un additionneur 16 bits, le délai est de 16 ut.



Dans ces exemples, la propagation de la retenue est séquentielle : le délai est proportionnel à la longueur du chemin de propagation de la retenue.

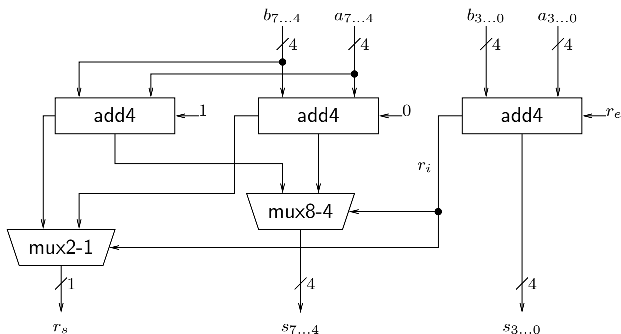
Pour implanter les additioneurs de la partie précédente, on s'est basé sur l'algorithme « à la main » : on somme les bits de même poids, en propageant la retenue éventuelle des poids faibles vers les poids forts.

On a obtenu un additionneur 8 bits à partir de deux additionneurs 4 bits :



Le chemin critique est le chemin allant de r_e à r_s : le délai du circuit est de 16 ut.

Comment obtenir un additionnneur 8 bits avec un délai de 10 ut entre r_e et r_s ?



On suppose un délai de 2 ut pour les multiplexeurs. Au bout de 8 ut,

- la retenue intermédiaire r_i pour les 4 bits de poids faibles est connue,
- les 4 bits de poids forts sont connus, quelle que soit r_i,
- reste à sélectionner les 4 bits de poids forts d'après r_i.

En tout, le délai de r_e à r_s tombe à 10 ut.

On a introduit du *parallélisme*, ce qui a permis de diminuer le délai mais en augmentant la surface : comme souvent, il y a un compromis.