ELEC-H-305

Circuits logiques et numériques

2011-2012

Cours 5

Dragomir Milojevic

Dragomir.Milojevic@ulb.ac.be

Problèmes à résoudre

Cahier de charges verbal

On pose un problème sous forme d'un énoncé verbal.

Il s'agit de traduire cet énoncé selon le formalisme de l'Algèbre de Boole.

Cette traduction permet d'avoir une spécification formelle du problème.

Spécification formelle

Une description complète, précise et non-ambiguë de toutes les propriétés d'un système.

Exemple: une expression Booléenne, une TdV, une K-Map.

Problèmes à résoudre

Cahier de charges verbal

La première question à se poser:
Est-ce un problème combinatoire?

(Combinatoire: les sorties du système ne dépendent que des entrées)

Si OUI, alors on peut appliquer tout ce que vous avez appris jusqu'à présent.

Conseil:

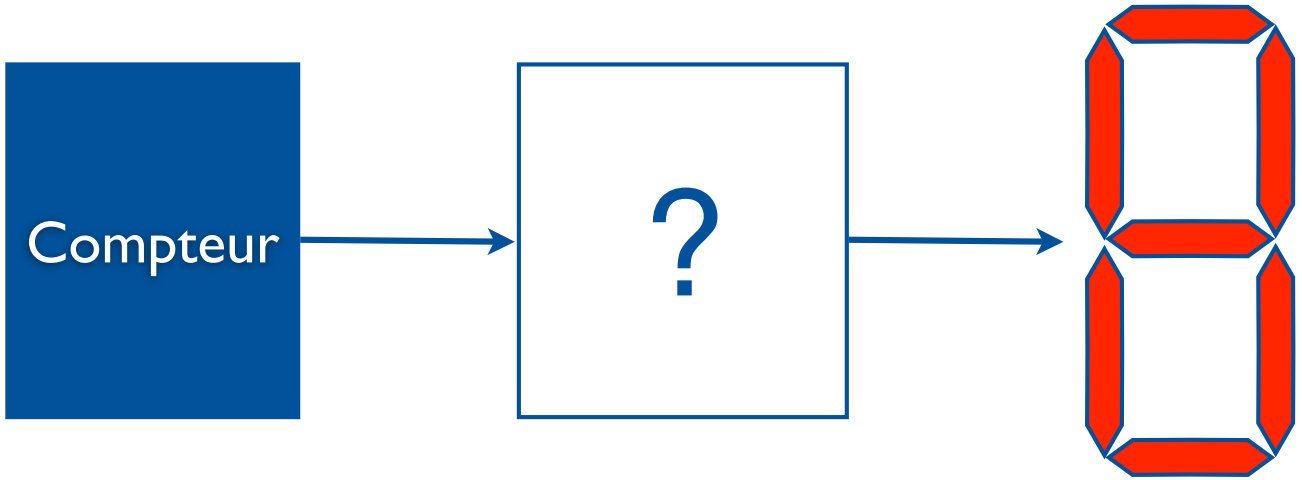
Commencer par déterminer le nombre d'entrées et de sorties du système (fixe le nombre de fonctions logiques — **les sorties** et le nombre de variables pour chaque fonction — **les entrées**).

Exemple de synthèse complet

Exemple

Afficheur 7-segments à LED.

Un compteur de 4 bits fourni un chiffre (codé en BCD). On souhaite afficher ce chiffre en décimal, à l'aide d'un afficheur à 7-segments (7 diodes LED).



Exemple de synthèse complet

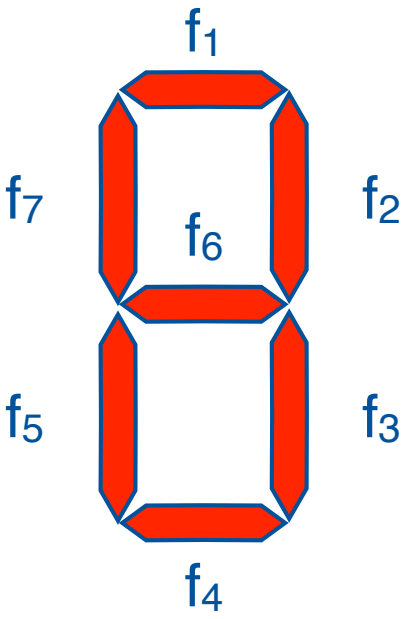
Analyse du problème

C'est un problème combinatoire (à une combinaison des variables à l'entrée — à un chiffre donc, correspondra toujours la même sortie):

- **Les entrées**
  ...compteur à 4 bits fourni un chiffre (codé en BCD)...
  Donc nous avons bien 4 entrées.
  La table de vérité comporte alors 2^4=16 lignes.
- **Les sorties**
  ... *un afficheur à 7-segments ...*
  Il y a donc 7 sorties.
- Il y a donc bien 7 fonctions logiques (i.e. 7 tables de vérités, 7 K-Maps) de 4 variables chacune.

Exemple de synthèse complet

Principe



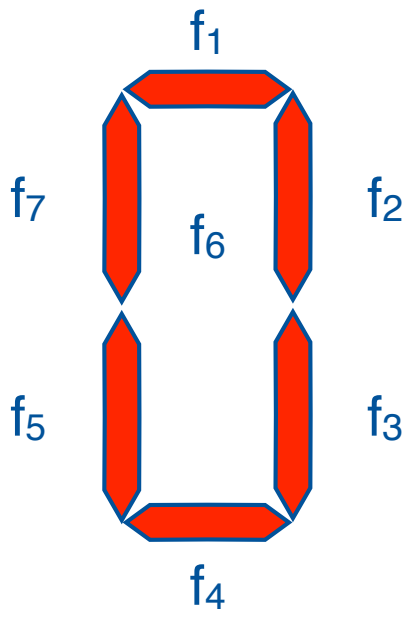
A chaque LED de l'afficheur on attribue une fonction logique:

0 - LED est éteinte

1 - LED est allumée

Exemple de synthèse complet

Principe

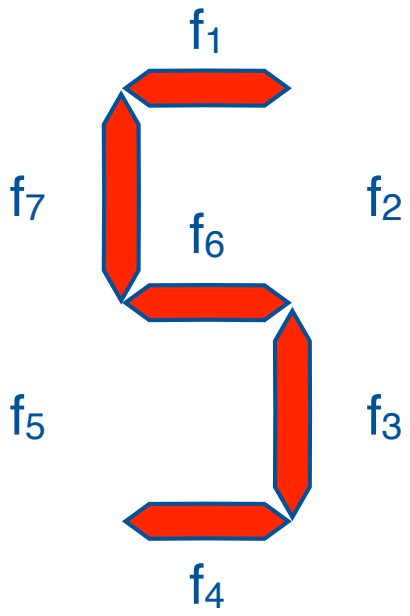


$f_1=f_2=f_3=f_4=f_5=f_7=1$

$f_6=0$

Exemple de synthèse complet

Principe

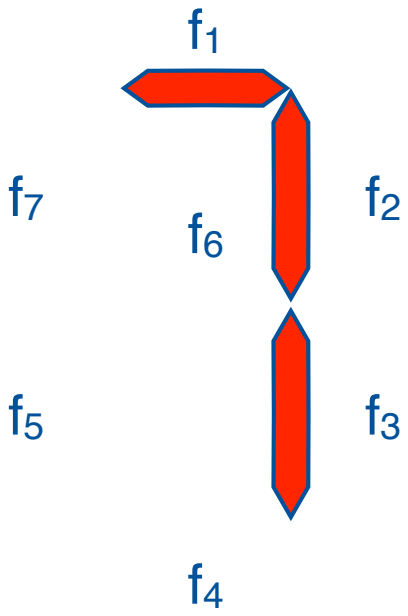


$f_1=f_3=f_4=f_6=f_7=1$

$f_2=f_5=0$

Exemple de synthèse complet

Principe



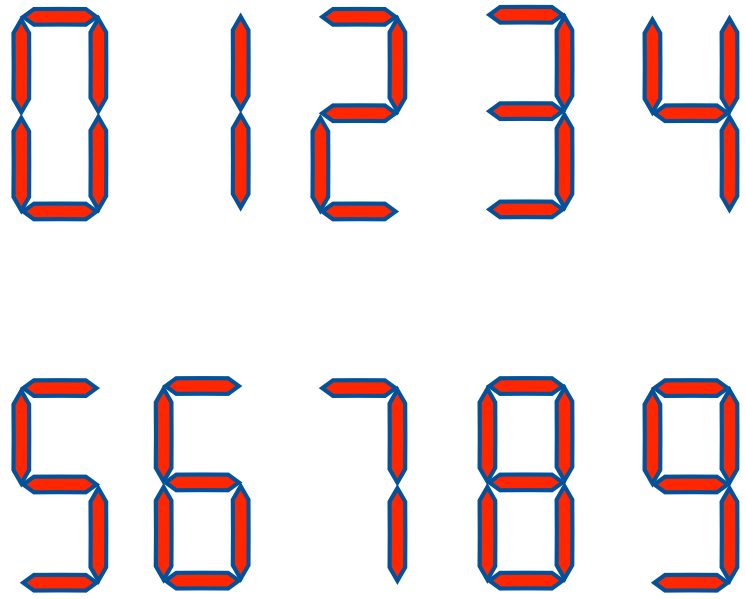
$f_1=f_2=f_3=1$

$f_4=f_5=f_6=f_7=0$

Exemple de synthèse complet

Tables de vérité

Décimal	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	1	1	1	1	1	0	1
1	0	1	1	0	0	0	0
2	1	1	0	1	1	1	0
3	1	1	1	1	0	1	0
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
10							
11							
12							
13							
14							
15							

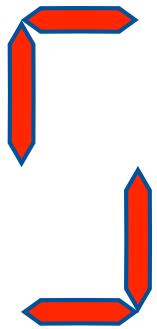


Exemple de synthèse complet

Que faire des nombres de 10 à 15?

1. On peut les considérer comme les ***don't cares***:

   - Lors de la simplification, certains de ces *don't cares* vont être mis à 1, pour une meilleur simplification (problème à plusieurs fonctions)
   - Pour cette combinaison, ce qui sera affiché peut ne pas avoir aucune signification...
   - Par exemple:



Exemple de synthèse complet

Que faire des nombres 10 à 15?

2. Si ce sont les *don't happen* alors pas de problème d'affichage erroné car les chiffres de 10 à 15 n'arriveront jamais à l'entrée de notre codeur.

Comment arriver à faire les *don't happen*

Par exemple : la valeur du compteur est automatiquement mis à zéro à chaque fois qu'il arrive à 10.

Les capteurs mutuellement exclusifs: détecteur de présence d'un ascenseur haut et bas ne peuvent pas être vraie en même temps.

Exemple de synthèse complet

Que faire des nombres 10 à 15?

3. On peut aussi prévoir l'affichage d'un code erreur

Décimal	f_1	f_2	f_3	f_4	f_5	f_6	f_7
...	...	...	...	...	...	...	...
10	1	0	0	1	1	1	1
11	1	0	0	1	1	1	1
12	1	0	0	1	1	1	1
13	1	0	0	1	1	1	1
14	1	0	0	1	1	1	1
15	1	0	0	1	1	1	1

Exemple de synthèse complet

Problème de l'afficheur

Les tables de vérité et les fonctions logiques ...

A faire lors des TPs, un bon exercice pour les K-Maps.

Un autre exemple

Convertisseur BCD/Excédent3

Pour le problème décrit par le cahier de charges suivant, établir le (s) table(s) de vérité, donner la liste des mintermes et des maxtermes, dresser la(es) K-map(s), donner la liste de tous les implicants premiers et simplifier le(s) expression(s).

On souhaite réaliser un circuit convertisseur du code binaire à quatre bits (sortie d'un compteur p.e) vers le code excédent 3.

Un autre exemple

Analyse du problème

- **Les entrées**
  ...*convertisseur du code binaire à quatre bits...*
  Donc nous avons bien 4 entrées. La table de vérité comporte 2^4=16 lignes.

- **Les sorties**
  ...*le code excédent 3...*
  Les chiffres de 0 à 9 sont codés sur 4 bits, nous avons bien 4 bits de sortie

- Il y a donc bien 4 fonctions logiques (i.e. 4 tables de vérités, 4 K-Maps) de 4 variables...

Un autre exemple

Table de conversion: décimal vers binaire des codes

Décimal	8421 (BCD)	2421
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	1011
6	0110	1100
7	0111	1101
8	1000	1110
9	1001	1111

Le système a 4 entrées binaires: donc la table devrait avoir 16 lignes.

Que faire des autres codes entrants?

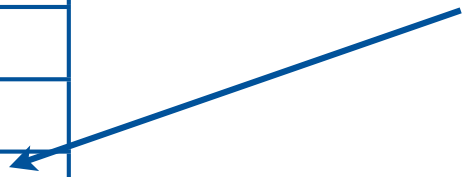
les *don't cares*!

Un autre exemple

Table de vérité

Décimal	BCD	2421
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	1011
6	0110	1100
7	0111	1101
8	1000	1110
9	1001	1111
10	1010	----
11	1011	----
12	1100	----
13	1101	----
14	1110	----
15	1111	----

f_4, f_3, f_2, f_1



Un autre exemple

K-maps

f_1	00	01	11	10	x_2x_1
00	0	0	1	0	
01	0	1	1	0	
11	–	–	–	–	
10	0	1	–	–	

x_4x_3

f_2	00	01	11	10	x_2x_1
00	0	0	1	1	
01	0	1	0	0	
11	–	–	–	–	
10	1	1	–	–	

x_4x_3

$f_1 = x_2x_1 + x_4x_1 + x_3x_1$

$f_2 = x_4 + x_3x_2'x_1 + x_3'x_2$

Un autre exemple

K-maps

f_3	00	01	11	10	x_2x_1
00	0	0	0	0	
01	1	0	1	1	
11	–	–	–	–	
10	1	1	–	–	

x_4x_3

f_4	00	01	11	10	x_2x_1
00	0	0	0	0	
01	0	1	1	1	
11	–	–	–	–	
10	1	1	–	–	

x_4x_3

$f_3 = x_4 + x_3x_2 + x_3x_1'$

$f_4 = x_4 + x_3x_1 + x_3x_2$

Circuits typiques

A. Additionneur

On souhaite réaliser un circuit additionneur de deux mots A et B codés sur 4 bits.

Le problème est évidement plus général (mots de *n* bits). On considère les mots de 4 bits pour la simplicité des solutions.

Deux solutions possibles:

A. Mimer le principe d'addition bit à bit
(séquence de calcul dans le temps).

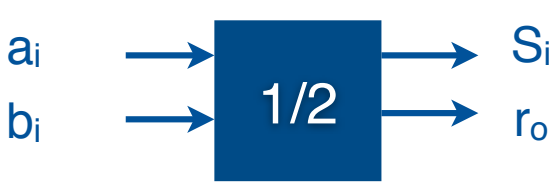
B. Anticiper le report pour un certain nombre de bits
(considérer le problème comme combinatoire).

Circuits typiques

A. Additionneur

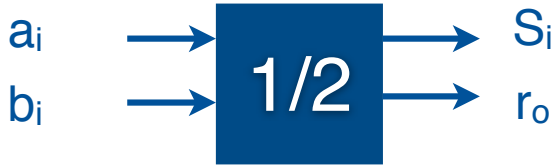
L'addition faite à l'aide d'un 1/2 additionneur (deux opérandes d'un bit — deux bits à l'entrée) plus *n-1* (ici 3) additionneurs complets (deux opérandes d'un bit, plus un bit de report d'un étage précédant).

Chacun de ces circuits élémentaires génère 2 bits de sortie (décrits formellement avec deux expressions Booléennes).



Circuits typiques

A. Synthèse de demi additionneur (K-Maps)



S_i	0	1	b_i
0	0	1	
1	1	0	

a_i

r_o	0	1	b_i
0	0	0	
1	0	1	

a_i

$$S_i = a_i'b_i + a_ib_i'$$
$$= a_i \oplus b_i$$

$$r_o = a_i\ b_i$$

Circuits typiques

A. Synthèse de additionneur complet (K-Maps)

S	00	01	11	10	a_ib_i
0	0	1	0	1	
1	1	0	1	0	

r_i

r	00	01	11	10	a_ib_i
0	0	0	1	0	
1	0	1	1	1	

r_i

$$S = a_i \oplus b_i \oplus r_i$$

$$r = a_i b_i + r_i b_i + r_i a_i$$

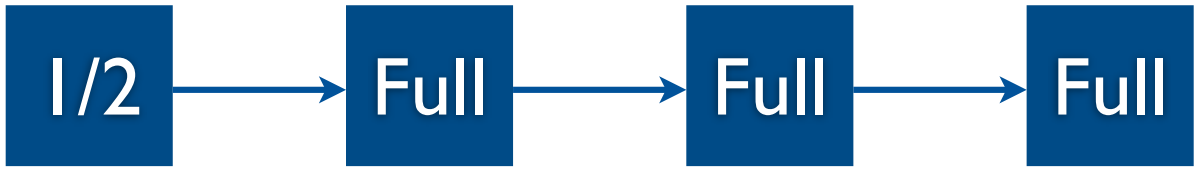
Circuits typiques

A. Additionneur complet sur 4 bits

Mis en série des éléments nécessaires pour obtenir un additionneur complet (*Ripple carry adder*). Un 1/2 additionneur plus *n-1* additionneurs complets.

Problème de propagation de report entre les étages: le résultat final sera obtenu après la propagation du report à travers tous les étages de l'additionneur!

n-délais à additionner (un délais: le délais d'un étage — d'une boite noire)



Circuits typiques

B. Additionneur Carry Look Ahead — CLA

Un additionneur complet comme un **SEUL** circuit combinatoire. Pas de problème de délais de calcul lié à la prorogation du report, car l'anticipation du report — *Carry Look Ahead*.

La réponse est obtenue après un seul délais.

Si un tel additionneur est réalisé à l'aide des mémoires (*Look Up Tables* — LUT), le résultat sera obtenu après un temps d'accès à la mémoire:

Circuits typiques

B. Analyse du problème

C'est un problème **combinatoire**, donc

- Les entrées
  ... *deux mots A et B codés sur 2 bits...*
  Donc nous avons bien 4 entrées.
  La table de vérité comporte 2^4=16 lignes.

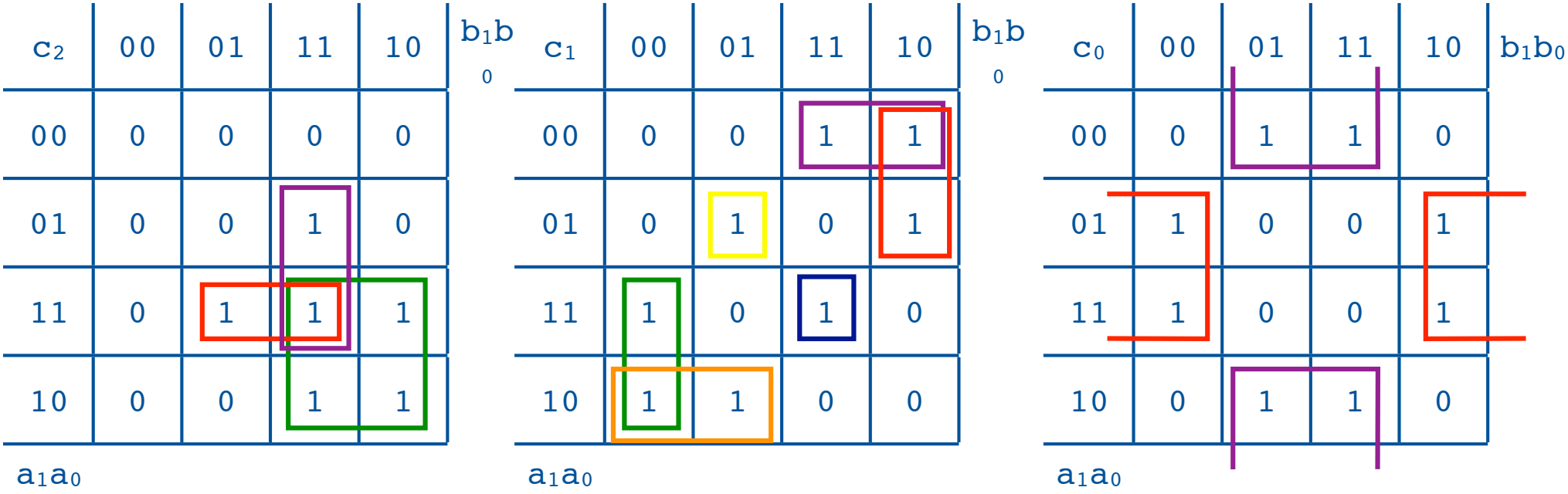
- Les sorties
  Résultat de l'addition : A=3, B=3; A+B=6 $(110)_2$

- Il y a donc bien 3 fonctions logiques
  (i.e. 3 tables de vérités, 3 K-Maps de 4 variables chacune...)

Circuits typiques

B. Additionneur — CLA (K-Maps)

K-maps



$c_2 = a_1b_1 + a_1a_0b_0 + a_0b_1b_0$

$c_1 = a_1b_1'b_0' + a_1a_0'b_1' + a_1'a_0'b_1 + a_1'b_1b_0' + a_1'a_0b_1'b_0 + a_1a_0b_1b_0$

$c_0 = a_0b_0' + a_0'b_0$

Circuits typiques

B. Additionneur complet sur n-bits

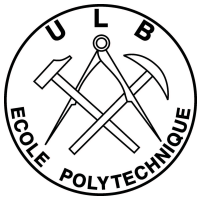
Pour un plus grand nombre de bits (32, 64, 128...) on fait la mise en série des additionneurs CLA.

Question: Pour quoi?

Nombre d'entrées/sorties = impact sur la taille de circuit (la mémoire) - Explosion combinatoire.

Il faut attendre aussi la propagation du report, mais le report doit traverser moins d'étages (donc c'est plus rapide, que l'approche avec des additionneurs complets).

Circuits typiques



Encodeur de priorité

Un mot `D` *est codé sur* n *bits. On souhaite connaître l'index du poids plus fort ayant un 1 (mot* `A` *- indique la position du MSB ayant un* `1`*).*

On souhaite également signaler la présence d'un 1 dans le mot `D`*, à l'aide d'un signal ANY qui sera mis à 1 lorsque le mot* `D` *présente au moins un 1.*

Circuits typiques

Encodeur de priorité

On considère un mot D de 4 bits ($d_3 d_2 d_1 d_0$).

L'index maximal est 4, nous avons donc besoin de 2 bits pour le mot A ($a_1 a_0$).

Un bit (ANY) est utilisé pour signaler que le mot n'est pas un 0.

La table de vérité simplifiée (on n'écrit pas toutes les combinaisons des variables à l'entrée - certaines combinaisons sont représentées comme les *don't care*).

d_3	d_2	d_1	d_0	a_1	a_0	ANY
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	–	0	1	1
0	1	–	–	1	0	1
1	–	–	–	1	1	1

Circuits typiques

Encodeur de priorité

K-maps

a_1	00	01	11	10	d_1d_0
00	0	0	0	0	
01	1	1	1	1	
11	1	1	1	1	
10	1	1	1	1	
d_3d_2					

a_0	00	01	11	10	d_1d_0
0	0	0	1	1	
1	0	0	0	0	
11	1	1	1	1	
10	1	1	1	1	
d_3d_2					

$$a_1 = d_3 + d_2$$

$$a_0 = d_3 + d_2'd_1$$

Circuits typiques

Encodeur de priorité

K-map - Synthèse des 0

ANY	00	01	11	10	d_1d_0
00	0	1	1	1	
01	1	1	1	1	
11	1	1	1	1	
10	1	1	1	1	
d_3d_2					

$$a_1 = d_3 + d_2 + d_1 + d_0$$

Simplification des fonctions

Méthodes de simplification

1. Axiomes et théorèmes

2. K-Maps

3. Méthode Quine-Mc.Cluskey

- Méthode systématique : c'est une analyse de **TOUTES** les possibilités de regroupement — recherche de **tous** les implicants premiers (méthode exhaustive).
- Utilisée pour des problèmes à grand nombre de variables.
- Elle garanti la meilleure solution, facilement programmable.

Simplification des fonctions

Méthode Quine-Mc.Cluskey

Algorithme en deux étapes:

A. Recherche des implicants premiers (phase d'**ANALYSE**)
   Recherche systématique des implicants par la méthode des tri successifs.

B. Couverture de la fonction (phase de **SYNTHESE**)
   Le choix des implicants premiers permettant la couverture de la fonction (on obtient le résultat de la simplification: expression optimale)

Simplification des fonctions

Méthode Quine-Mc.Cluskey

Rappel par rapport au K-Maps

Recherche des implicants premiers principe dans les K-Maps:
La recherche des TOUS les *n*-cubes de façon visuelle (y compris les *n*-cubes composés uniquement des *don't cares*).

f_1	00	01	11	10	x_2x_1
00	0	0	1	0	
01	0	1	1	0	
11	–	–	–	–	
10	0	1	–	–	

x_4x_3

$IP_1=x_2x_1$

$IP_2=x_3x_1$

$IP_3=x_4x_3$

$IP_4=x_4x_1$

$IP_5=x_4x_2$

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : méthode (11 étapes)

1. Pour une fonction `f` de *n* variables on commence par construire *m* groupements de **tous** les **mintermes** de la fonction.

2. Dans chaque groupement des mintermes (noté G_i) il y a `i` variables qui valent `1` i.e. `(n-i)` variables qui valent `0`.

Rm. Il est bien sur possible que il y ait *m=n* groupements.

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : remarque étapes 1. et 2.

   i. Lors de la construction des groupements faire attention au bit de poids plus fort et plus faible (interprétation décimale).

   ii. D'abord dresser la table de vérité et puis faire les groupements (aussi sous forme des tables).

   iii. Marquer **clairement** la séparation entre les différents groupements G_i (dans ces transparents ce sera une barre de tableau plus épaisse).

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

Exemple de construction des groupements G_i:

La fonction logique est donnée comme une liste des mintermes (équivalents décimaux).

$f(a,b,c)=\sum(0,1,3,4,5,7)$

	a	b	c	f	
0	0	0	0	1	x
1	0	0	1	1	x
2	0	1	0	0	
3	0	1	1	1	x
4	1	0	0	1	x
5	1	0	1	1	x
6	1	1	0	0	
7	1	1	1	1	x

→

Groupements G_i

	a	b	c
G_0	0	0	0
G_1	0	0	1
	1	0	0
G_2	0	1	1
	1	0	1
G_3	1	1	1

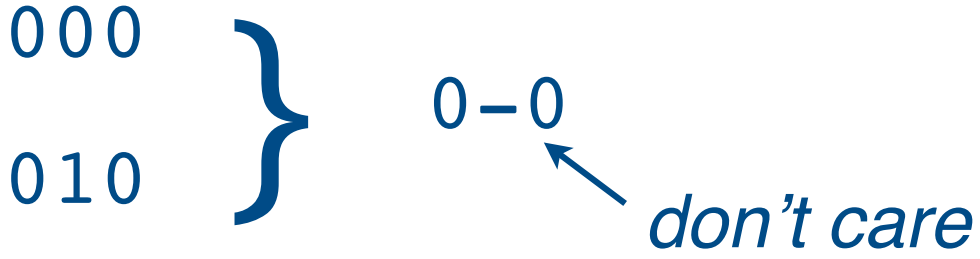
Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : méthode

3. On **compare** ensuite chaque minterme de G_i avec **TOUS** les mintermes appartenant à G_{i+1} (pour tous les i)

4. Lorsque la distance de Hamming entre deux mintermes est de 1, on note l'appariement entre les deux mintermes comme suit:



C'est une application de x+x'=1

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : méthode

Analogie avec les K-Maps : appariement implique la possibilité de réalisation d'un *n*-cube d'ordre 2 !

f_1	00	01	11	10	x_2x_1
00	0	0	1	0	
01	0	1	1	0	
11	0	0	0	0	
10	0	0	0	0	

x_4x_3

$$f_1 = x_4'x_3'x_2x_1 + x_4'x_3x_2x_1$$

$$= x_4'x_2x_1(x_3'+x_3)$$

$$= x_4'x_2x_1$$

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers: méthode

5. Chaque appariement entre G_i et G_{i+1} est donc un *n*-cube d'ordre 2.
   Tous les appariements de G_i et G_{i+1} sont écrits dans un nouveau groupement, noté G_i'.

6. Après une première passe il y donc au moins *m-1* groupements G_i' (il est possible qu'il n'y ait aucune paire de mintermes dont la distance de Hamming vaut 1).

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

Exemple de construction des groupements Gi'

Groupements G_i

	a	b	c
G_0	0	0	0
G_1	0	0	1
	1	0	0
G_2	0	1	1
	1	0	1
G_3	1	1	1

Notation intermédiaire

ici toutes les possibilités

G_0,G_1

000
001

000
100

G_1,G_2

001
011

001
101

~~100~~
~~011~~

100
101

G_2,G_3

011
111

101
111

Distance de Hamming =2

Groupements G_i'

G_0'

00– (0,1)
–00 (0,4)

G_1'

0–1 (1,3)
–01 (1,5)
10– (4,5)

G_2'

–11 (3,7)
1–1 (5,7)

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : méthode

7. Lors d'appariement de G_i et de G_{i+1}, il est possible qu'un minterme **n'a pas été regroupé** avec d'autres.

   Il s'agit d'un **impliquant premier** (c'est un *n*-cube d'ordre 1). On le(s) note(ent) IP_k dans l'ordre d'apparition.

8. L'ensemble G_i' constitue donc l'ensemble de **tous les *n*-cubes d'ordre 2** de la fonction.

9. Tous les mintermes de G_i qui ont été utilisés pour les appariement sont marqués avec un `x`.

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : méthode

10. Le processus est répété (avec les mêmes règles):

- pour tous les éléments des groupements G_i' (obtention de tous les *n*-cubes d'ordre 2)
- pour obtenir l'ensemble noté G_i'' (les *n*-cubes d'ordre 3) et ainsi de suite (obtention des *n*-cubes d'ordre 4, ...).

Attention!!! Lors de calcul de la distance de Hamming pour le passage de G_i' à G_i'', les *don't cares* sont également pris en compte.

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers : méthode

Exemples d'appariement dans le G_i' *pour obtenir* G_i'' *obtenir* G_i'' *avec les don't cares.*

-000	Distance de Hamming de 1:	-000
-100	on peut apparier.	--00
~~-000~~	Distance de Hamming de 2:	
~~01-0~~	on ne peut pas apparier.	

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers: méthode

11. On s'arrête lorsqu'il n'est plus possible de faire des appariements (on aurait trouvé le(s) plus grand *n*-cube(s) possible).

    On dresse la liste de tous les implicants premiers (IPs) trouvés.

    Les IPs sont écrit sous forme de '0' et de '1' et sous forme des monômes.

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

Exemple de construction des groupements G_i''

G_0'	00-	(0,1)	X
	-00	(0,4)	X
G_1'	0-1	(1,3)	X
	-01	(1,5)	X
	10-	(4,5)	X
G_2'	-11	(3,7)	X
	1-1	(5,7)	X

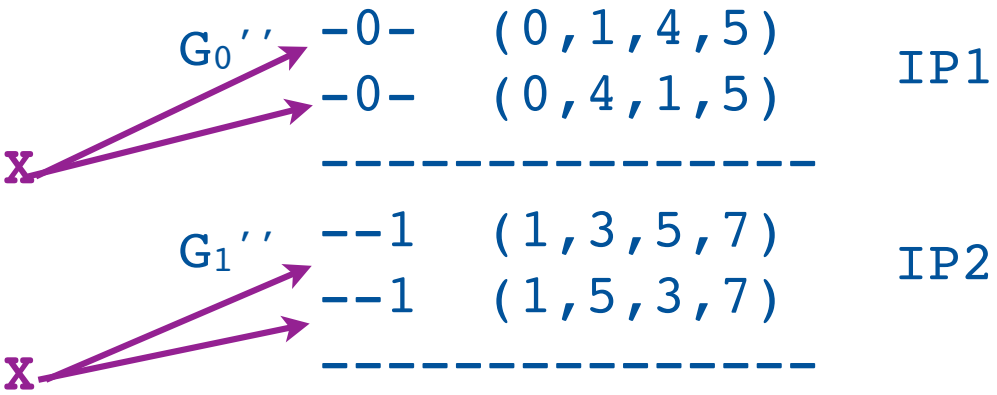
→

G_0''	-0-	(0,1,4,5)	IP1
	-0-	(0,4,1,5)	
G_1''	--1	(1,3,5,7)	IP2
	--1	(1,5,3,7)	

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

Liste des IPs



IP1 = -0- = b'

IP2 = --1 = c

X Ce sont les mêmes, on peut les omettre par la suite...

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

Vérification de résultat obtenu par K-Maps (pas toujours nécessaire... ni possible) mais dans un premier temps peut servir pour contrôle.

Fonction de départ (pour cette exemple K-Map c'est plus rapide):

$f(a,b,c)=\sum(0,1,3,4,5,7)$

IP1=b'
IP2=c

f	00	01	11	10	bc
0	1	1	1	0	
1	1	1	1	0	
a					

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

A. Recherche des implicants premiers
   Méthode lorsque la fonction logique n'est pas complètement spécifiée (**présence des indifférents - *don't cares***)

   1. On fait l'hypothèse que tous les indifférents valent 1
      L'idée est de permettre **TOUTES** les simplifications possibles due aux indifférents.

   2. Les implicants premiers **non-utiles** (ceux, constitués des indifférents uniquement) seront éliminés lors de la 2ème phase de l'algorithme (couverture - synthèse de la fonction logique)

Simplification des fonctions

Méthode Quine-Mc.Cluskey: Analyse

Exemple des implicants premiers non-utiles

f_1	00	01	11	10	x_2x_1
00	0	0	1	0	
01	0	0	1	0	
11	1	0	1	0	
10	1	0	1	0	
x_4x_3					

Pour la fonction f_1 on trouve

$IP_1=x_1x_2$

$IP_2=x_4x_1'x_2'$

Mais pour la fonction simplifiée on ne prends pas IP_2 (c'est un IP inutile car il ne couvre que les indifférents)