LINFO1140 : Électronique de L'Informatique

Synthèse partie Digitale

Année 2021-2022

Jean-Didier Legat

5. Introduction to Digital Design

3 types de systèmes :

1) Système Décimal

Le système décimal est un système de numération utilisant la base dix. Pour chaque coefficient devant l'exposant, ce coefficient se trouvera entre 0 et 9. x * 10^n, ton x sera entre 0 et 9 (base 10). Le plus haut exposant de notre décomposition sera égal à n-1, où n représente le nombre de chiffres. (Ici, 9742 ➔ 4 chiffres, donc 4-1 = 3 donc 10^3)

$$9742_{10} = 9 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

nine thousands; seven hundreds; four tens; two ones

2) Système Binaire

Le système binaire est un système de numération utilisant la base deux. Pour chaque coefficient devant l'exposant, ce coefficient sera égal soit à 1 soit à 0. x * 2^n, ton x sera soit 0 ou 1 (base 2). Le plus haut exposant de notre décomposition sera égal à n-1, où n représente le nombre de chiffres. (Ici, 10110➔ 5 chiffres, donc 5-1 = 4 donc 2^4). Donc ici, les bases des exposant sont des 2.

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22_{10}$$

one sixteen; no eight; one four; one two; no one

3) Système Hexadécimal

Le système hexadécimal est un système un peu spécial qui en plus d'avoir les nombres de 0 à 9, possède également les lettres de A à F, où A = 10, B = 11,… Ici nous avons une base 16, donc les coefficients des exposants vont de 0 à 15. (Ici, 2ED ➔ 3 caractères, donc 3-1 = 2 donc 16^2)

$$2ED_{16} = 2 \times 16^2 + E \times 16^1 + D \times 16^0 = 749_{10}$$

two two hundred fifty six's; fourteen sixteens; thirteen ones

Hex Digit	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

- Conversion Décimal – Binaire

Valeur	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0	0
167	1	0	1	0	0	1	1	1
255	1	1	1	1	1	1	1	1

Pour convertir du décimal au binaire, on prend ce tableau et notre chiffre, et on regarde si notre chiffre est plus grand que la valeur du tableau auquel nous sommes en train de regarder.

Exemple : 167 :
167 > 128 ? oui, donc 167 – 128 = 39 et on note 1 dans le tableau
39 > 64 ? Non, donc on passe et on note 0
39 > 32 ? Oui, donc 39-32 = 7 et on note 1 dans le tableau
7 > 16 ? Non, donc on passe et on note 0
7 > 8 ? Non, donc on passe et on note 0
7 > 4 ? Oui, donc 7-4 = 3 et on note 1 dans le tableau
3 > 2 ? Oui, donc 3-2 = 1 et on note 1 dans le tableau
1 >= 1 ? Oui donc 1-1 = 0, on s'arrête et on note 1 dans le tableau
Résultat → 10100111

- Conversion Binaire – Décimal

Pour convertir dans ce sens-là, on note les différents chiffres de notre binaire dans le tableau, puis à chaque fois que nous avons un 1 dans le tableau, on additionne sa valeur.

Exemple : 10100111 :
128 + 32 + 4 + 2 + 1 = 167

- Conversion Hexadécimal – Binaire

On prend chaque chiffre de notre nombre et on le décompose. Un chiffre en hex équivaut à 4 bits et tu prends chiffre par chiffre et tu le transformes en binaire

Exemple : 80
Donc 8 = 1000 et 0 = 0000 ➔ 1000 0000

- Conversion Hexadécimal – Décimal

On décompose d'abord en binaire puis en décimal

Exemple : $F0_{16}$:
1111 0000 ➔ 11110000 ➔ **si complément à 2 et le bit situé le plus à gauche vaut 1, le premier bit est négatif !** $-2^7 + 2^6 + 2^5 + 2^4 = -16$ (aussi si résultat > *max_range**)

L'Addition Binaire : l'addition binaire se fait comme une addition classique

```
  11
 1011
+0011
----
 1110
```

- 0 + 0 = 0
- 1 + 0 = 1
- 0 + 1 = 1
- 1 + 1 = 0 plus 1 de retenue, soit 10

max_range : chiffre maximum positif qui s'obtient en mettant n fois des 1 (n est le nombre de bits) et en remplaçant le premier bit par un 0 (sinon premier bit négatif si CÀ2)

Inversion Binaire : L'inversion se fait simplement en inversant les bits d'un nombre (0 devient 1 et 1 devient 0) + 1

6 = 0110 ➔ On inverse tous les bits ➔ 1001 ➔ On ajoute 1 ➔ 1010

- C'est d'ailleurs comme cela que l'on fait une soustraction en binaire, on fait une addition avec l'opposé du second terme.

Exercise 3

Convert the following decimal numbers to 6-bit two's complement binary number and subtract them. Express your answer as a 6-bit two's complement binary number. $-28_{10} - 3_{10} = ?$

Valeur	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
28	0	0	0	1	1	1	0	0

→ -28 : (100011) +1 = 100100

3 0 0 0 0 0 0 1 1

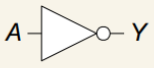
→ -3 : (111100) +1 = 111101

```
  11
 100100
+111101
-------
1000001
```

Attention ! pour passer du décimal au binaire, on note 0 si le nombre est égal à la valeur sauf pour la dernière valeur, 1. Exemple : 28 – 16 = 14 – 8 = 6 – 4 = 2. La valeur au-dessus est 2. Vu que les 2 nombres sont égaux, on note 0.

Les Différentes Portes Logiques :

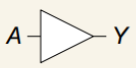
NOT



$Y = \overline{A}$

A	Y
0	1
1	0

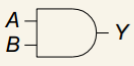
BUF



$Y = A$

A	Y
0	0
1	1

AND



$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

$Y = A + B$

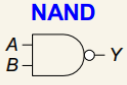
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

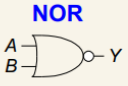
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

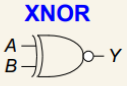
NOR



$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

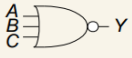
XNOR



$Y = \overline{A \oplus B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

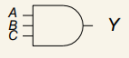
NOR3



$Y = \overline{A+B+C}$

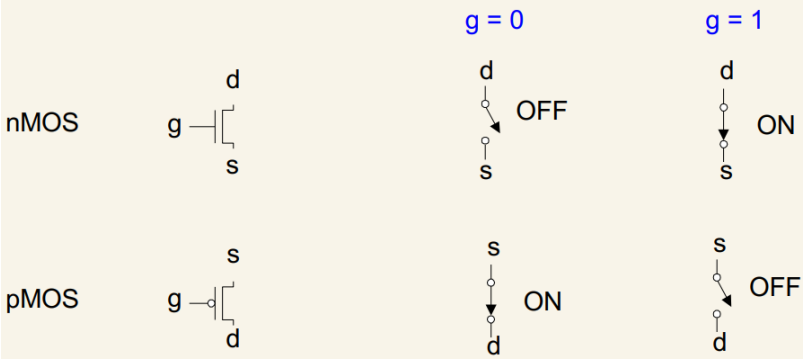
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

AND3



$Y = ABC$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



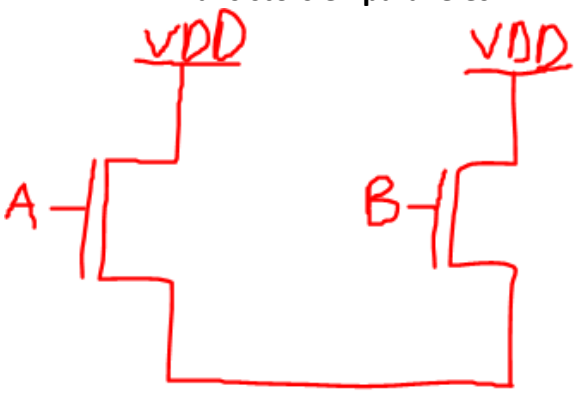
CMOS en série ➔ AND ➔ représente un fois *

CMOS en // ➔ OR ➔ représente un plus +

Transistors NMOS : laisse passer le courant quand l'entrée vaut 1, et l'empêche de passer quand elle vaut 0

Transistors PMOS : laisse passer le courant quand l'entrée vaut 0, et l'empêche de passer quand elle vaut 1

Transistors en parallèles



A+B

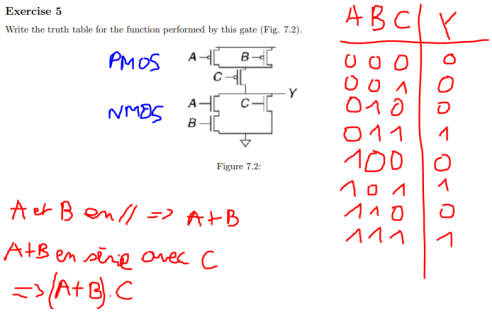
Transistors en séries

A*B

Pour trouver la valeur du paramètre W (en nm) :

W(NMOS) = 10 * n * L où n représente le nombre de NMOS max en série et L la longueur

W(PMOS) = 20 * n * L où n représente le nombre de PMOS max en série et L la longueur



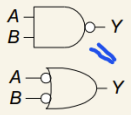
La partie PMOS est l'inverse de la partie NMOS, donc on ne prend compte que de cette partie

6 : Combinational Logic Design

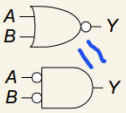
- De Morgan's Theorem

#	Theorem	Dual	Name
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots} = \overline{B_0} + \overline{B_1} + \overline{B_2} \ldots$	$\overline{B_0 + B_1 + B_2 \ldots} = \overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2} \ldots$	DeMorgan's Theorem

- $Y = \overline{AB} = \overline{A} + \overline{B}$



- $Y = \overline{A + B} = \overline{A} \bullet \overline{B}$

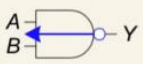


Théorème très important car il permet de simplifier nos fonctions logiques

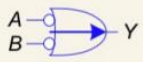
Ce changement de porte nous permet d'applique le « **Bubble Pushing** »

Bubble Pushing : Pratique consistant à déplacer les petites boules le long d'un circuit en partant de l'output à l'input

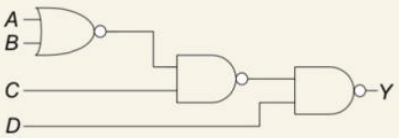
- Backward:
  - » Body changes
  - » Adds bubbles to inputs

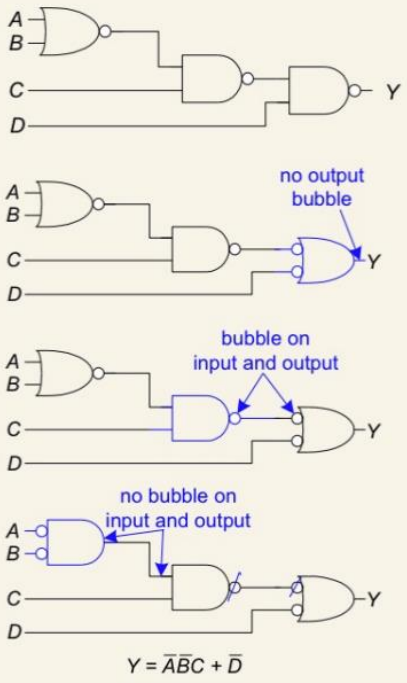


- Forward:
  - » Body changes
  - » Adds bubble to output



- Rules
  - » Begin at output, then work toward inputs
  - » Push bubbles on final output back
  - » Draw gates in a form so bubbles cancel





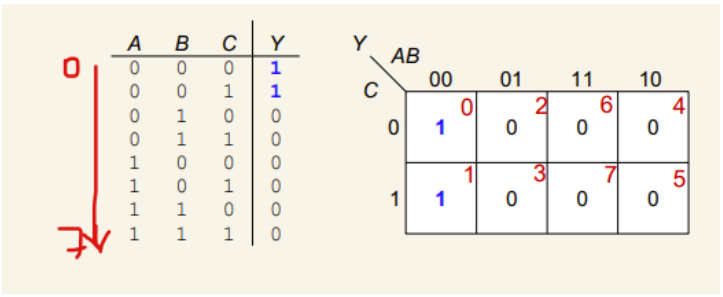
On part donc de l'output vers l'input et à chaque fois qu'on retrouve une petite boule aux 2 extrémités d'un fil, on peut les simplifier en les supprimant tout simplement

Diagramme de Karnaugh

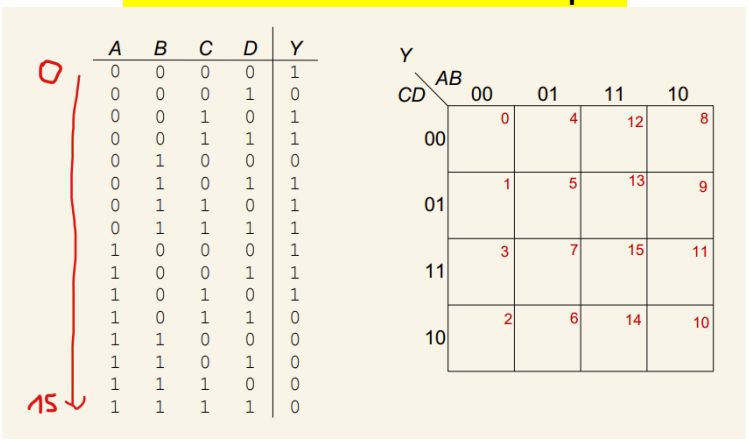
Le diagramme de Karnaugh permet de minimiser les expressions booléennes en combinant les termes, ainsi que de simplifier une fonction logique grâce à une représentation sous la forme de tableau où les paramètres sont les bits d'entrée et la valeur des cellules le bit de sortie

Le but est ensuite de rassembler les bits en paquets, pour ce faire on trace des rectangles les plus grands possibles afin de regrouper toutes les sorties qui sont à 1, ce qui peut ressembler à ceci en fonction du tableau

Positionnement des bits avec 3 inputs



Positionnement des bits avec 4 inputs

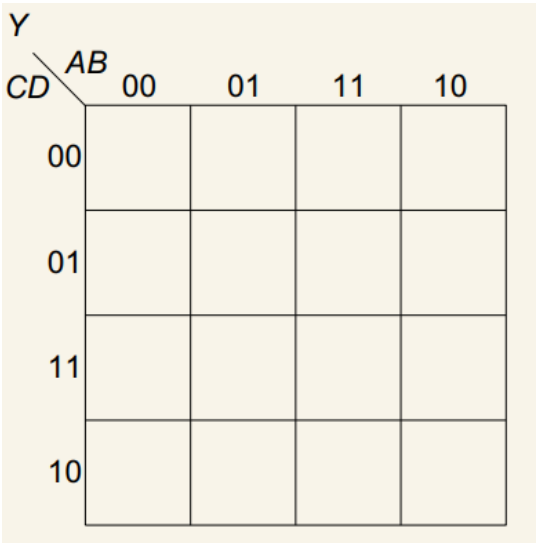


(J'ajoute ici une synthèse que j'avais déjà fait sur les tableaux de Karnaugh)

LINFO1140 : Résolution de Tableau de Karnaugh

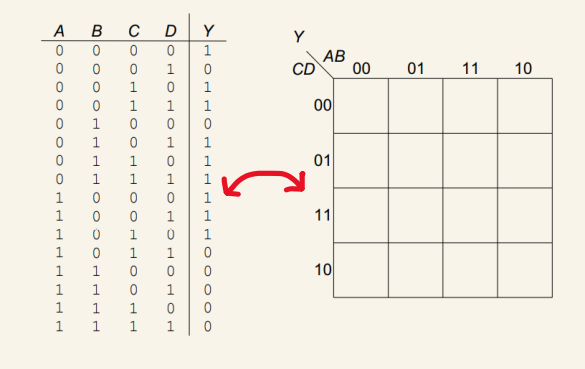
La *méthode du tableau de Karnaugh* permet d'effectuer des simplifications beaucoup plus rapidement sans avoir à écrire de longues équations.

Le tableau de Karnaugh est un tableau de 2^n cases, "n" étant le nombre de variables. A l'extérieur du tableau, on place les variables d'entrées. Exemple pour un tableau de Karnaugh composé de 4 variables :



Vu que l'on a 4 variables, nous avons $2^4 = 16$ cases.

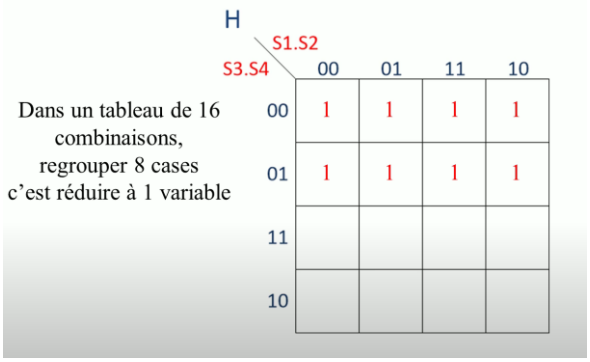
Le tableau de Karnaugh est en fait une construction graphique qui permet de visualiser différemment une table de vérité



A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Nous allons écrire dans les cases les différentes combinaisons de la table de vérité. Si on prend par exemple la première combinaison, la seule case où les 4 variables valent 0 est la toute première, en haut à gauche. Donc nous inscrivons dans cette même case 1.

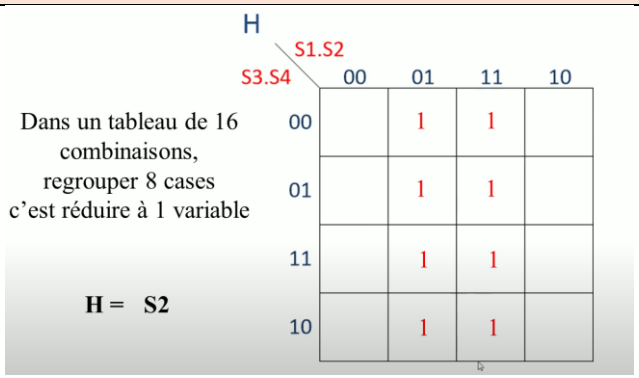
Règles de Simplification



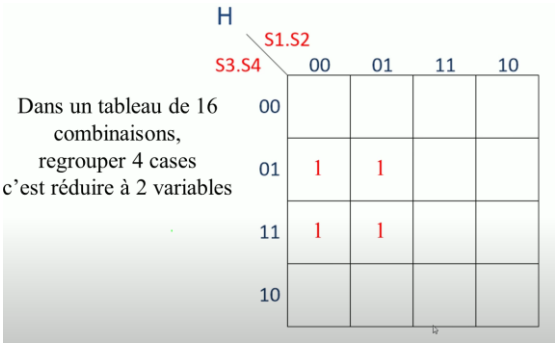
Nous devons trouver quel est le seul et unique point commun entre tous ces 1 dans ce cas-ci. On aperçoit très vite que c'est S3, car dans tous les points ici présents, tous ont 0 pour S3.

Mais la bonne réponse est (S3)' et non S3. *Pourquoi ?*

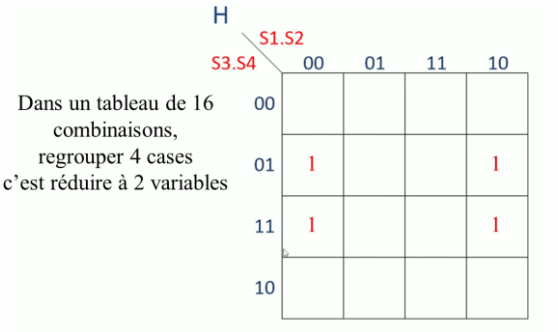
(Hypothèse) : si la variable / les variables qui ne varient pas valent :
- 0 ➔ not
- 1 ➔ la variable



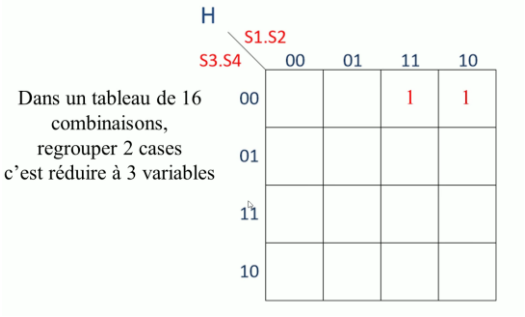
Ici, on voit bien que la variable qui ne varie pas est bel et bien S2, qui vaut 1, donc S2 tout court.



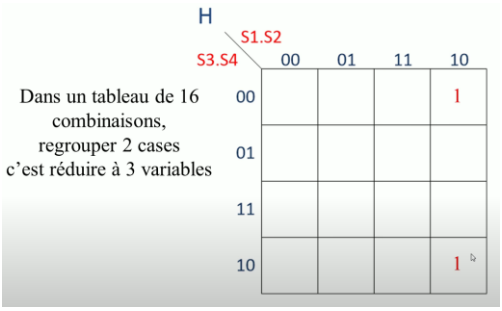
Dans ce cas, on doit trouver pas 1 mais 2 variables qui restent inchangées. Ici, ce sont S4, qui vaut 1, et S1, qui vaut 0, qui restent les mêmes. Donc la réponse est (S1)'S4



Nouveau cas de figure, ici, on doit partir du fait que le tableau peut se tourner sur lui-même. Le point commun est (S2)', puisque dans les 2 cas, que ce soit sur la partie gauche ou droite, S2 vaut 0. Le résultat final est (S2)'S4



Maintenant, nous devons trouver 3 variables. Nous trouvons facilement d'abord que S1 ne varie pas, vaut 1 donc S1. Et puisque nous n'avons qu'une case du côté de S3/S4, les 2 ne varient pas et valent 0 donc H = S1(S3)'(S4')



Ici, toutes les variables ne varient pas sauf S3, qui vaut 0 sur le 1 du dessus et 0 sur celui du dessous. Donc H = S1(S2)'(S4)'

ANNEXE

Nombre de cases	Nombre de variables à trouver
8	1
6	**Impossible**
4	2
2	3

Pour trouver des cases adjacentes d'un tableau de Karnaugh de 4 variables d'entrée :

4	2	2	4
3	1	1	3
3	1	1	3
4	2	2	4

Les cases portant le même chiffre sont des exemples de cases adjacentes
→ regroupement possible.

Remarques :

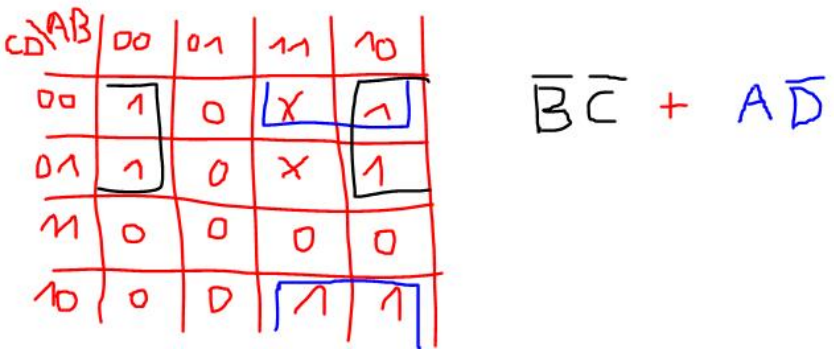
- Une ou plusieurs cases peuvent être communes à plusieurs groupements,
- Pour extraire l'équation de la fonction logique, on ne retient que les variables dont l'état ne change pas à l'intérieur d'un groupement et on effectue la somme logique (OU logique) de toutes les expressions trouvées,
- Le regroupement de 6 cases est impossible.

Exercise 2

Find a minimal Boolean equation for the function on Fig. 8.2 (x means "don't care") :

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	x
1	1	0	1	x
1	1	1	0	1
1	1	1	1	0

Figure 8.2:



Exercise 6

Find a minimized boolean equation for the function performed by the circuit of Fig. 8.6 :

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

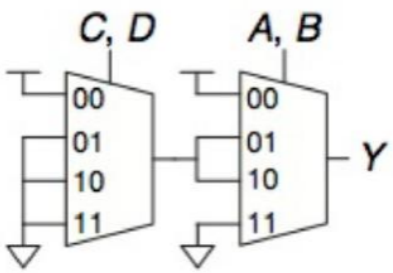


Figure 8.6:

CD/AB	00	01	11	10
00	1	1	0	1
01	1	0	0	0
11	1	0	0	0
10	1	0	0	0

$\bar{A}\bar{B} + \bar{C}\bar{D}B + \bar{B}\bar{C}\bar{D}$

Pour cet exo, on doit tout d'abord trouver la valeur de Y. Pour cela, on peut savoir que

Ligne Horizontale = source de courant donc vaut 1

Triangle = terre donc vaut 0

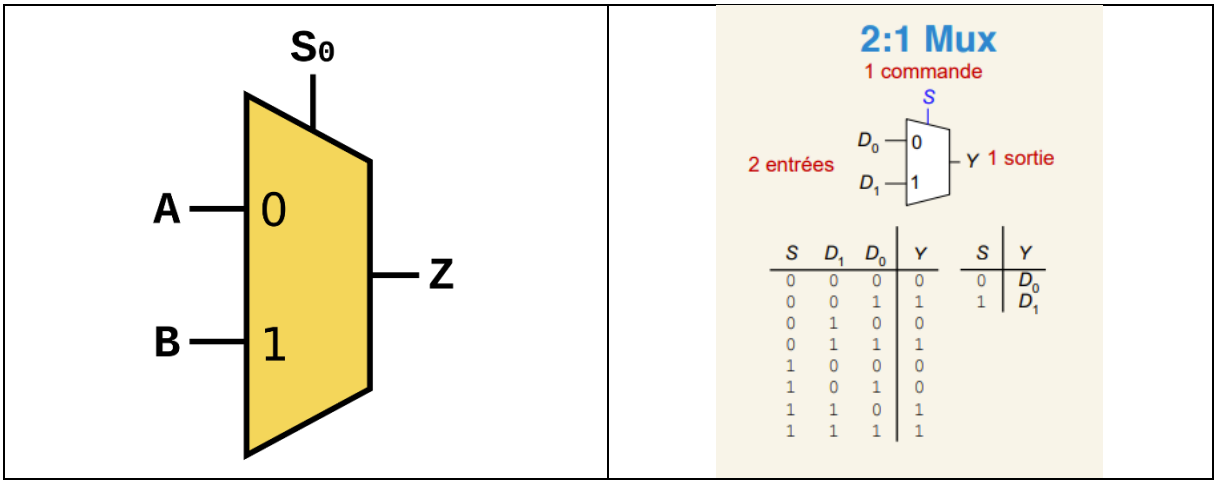
Pour trouver Y pour chaque combinaison, on doit faire une lecture du schéma. Pour 0000, on peut observer à droite pour AB que pour chaque AB valant 00, on a une source de courant donc 1. Donc pour toutes les combinaisons 00XX = 1. On peut observer en dessous que pour 11XX = 0 puisqu'on a une terre. Là où ça se complique, c'est pour AB valant 01 et 10. Puisque la valeur du courant pour ces combinaisons-là vaut en fait la valeur du courant appartenant à CD, donc la partie de gauche. Donc pour toutes 01XX et 10XX, on doit regarder au cas par cas pour trouver Y. Exemple : pour 0111, on a donc 01 pour AB donc on doit regarder dans la partie de gauche. On a CD = 11, et 11 est relié à un ground, donc la valeur du courant traversant 0111 vaut 0

Temps de contamination : temps le + long passant par le plus de portes logiques possibles

Temps de propagation : temps le + court passant par le moins de portes logiques possibles

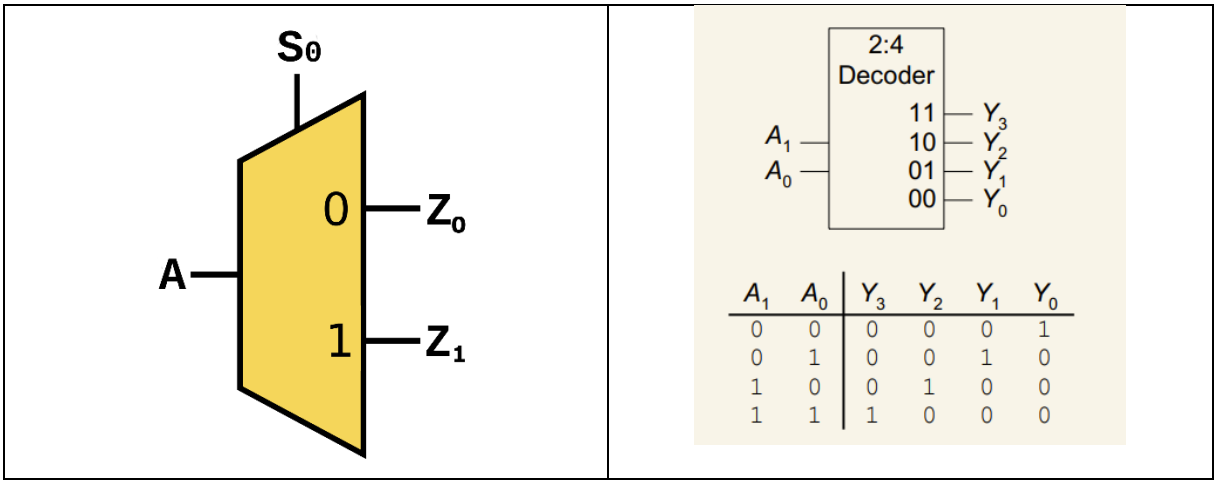
Multiplexeur :

Le multiplexeur est un dispositif permettant de choisir parmi plusieurs pistes pour n'en ressortir qu'une. Il permet donc de concentrer sur une même voie de transmission (l'output) différents types de liaison (les inputs). Formule : **N entrées ➔ 2^N sorties**



Décodeur (ou Démultiplexeur) :

Le décodeur est l'inverse du multiplexeur, il rassemble 1 ou plusieurs entrées et possède un nombre plus grand de sorties que d'entrée(s). Formule : **2^N entrées ➔ N sorties**



Timing = fait de calculer le temps nécessaire pour un circuit logique de changer d'état. Les changements d'états ne sont pas instantanés, on a un petit temps de latence :

1) *Propagation delay* = Le chemin le plus long entre l'entrée et la sortie, temps maximum pour avoir la bonne sortie (*Tpd*)
2) *Contamination delay* = Le chemin le plus court entre l'entrée et la sortie, temps minimum dans lequel la sortie ne change pas (*Tcd*)

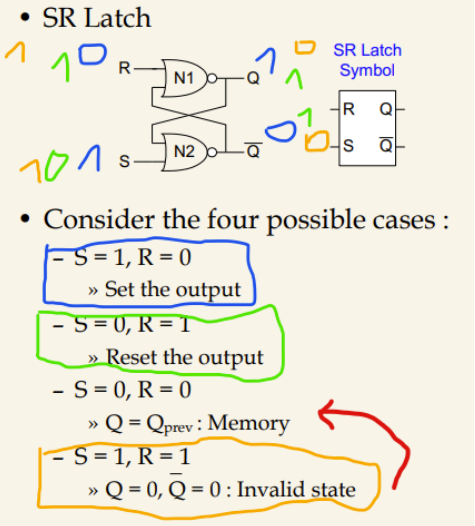
Lors de la transition *Propagation delay – Contamination delay* on observe ce que l'on appelle des **glitchs** : valeur de sortie instable

7 : Sequential Logic Design

SR-Latch

= 2 portes NOR croisées l'une sur l'autre. 4 combinaisons d'entrées possibles. Utilisation assez moindre

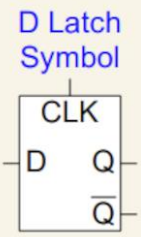
S = 1 & R = 0 ➔ Q = 1
S = 0 & R = 1 ➔ Q = 0
S = 0 & R = 0 ➔ Q = Q_{prev}
S = 1 & R = 1 ➔ Q = *Invalid* donc Q = 0

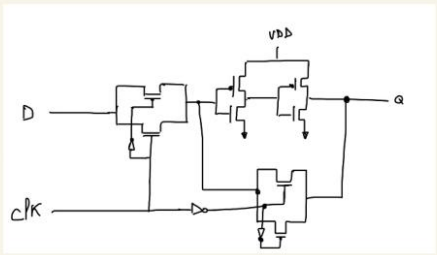


D-Latch

= première mémoire à introduire la notion de Clock. Permet d'utiliser un SR Latch de manière plus sûre. Peu utilisée pendant l'année puisqu'on a les D-FFs.

CLK = 1 ➔ Q = D
CLK = 0 ➔ Q = Q_{prev}

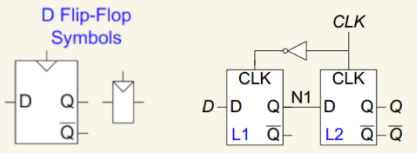


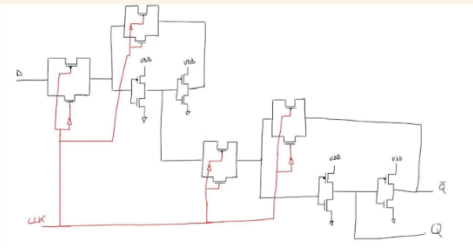


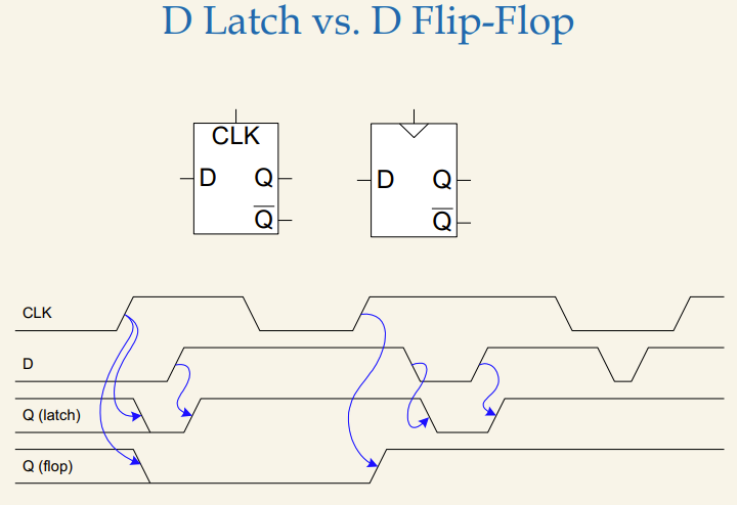
D-FF

= Le D Flip-Flop, ou D-FF, est un registre similaire au D-Latch à l'exception près que la valeur d'entrée n'est sauvée que lors du flanc montant de la clock.

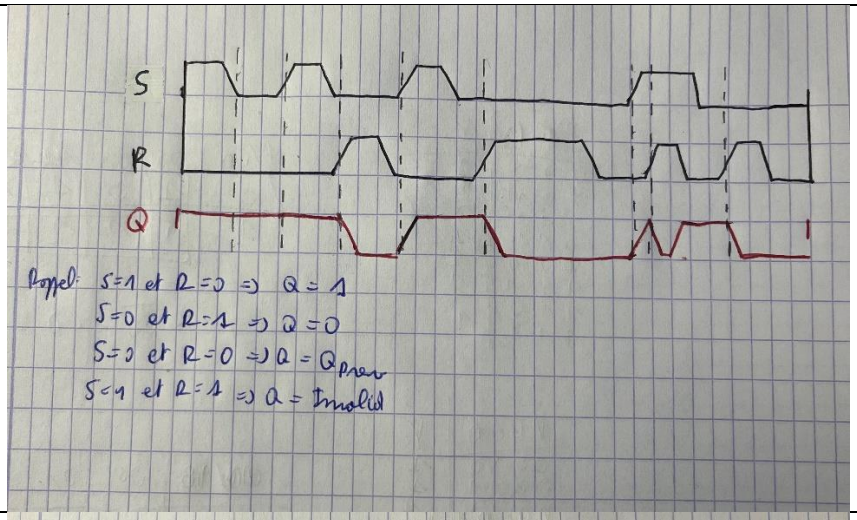
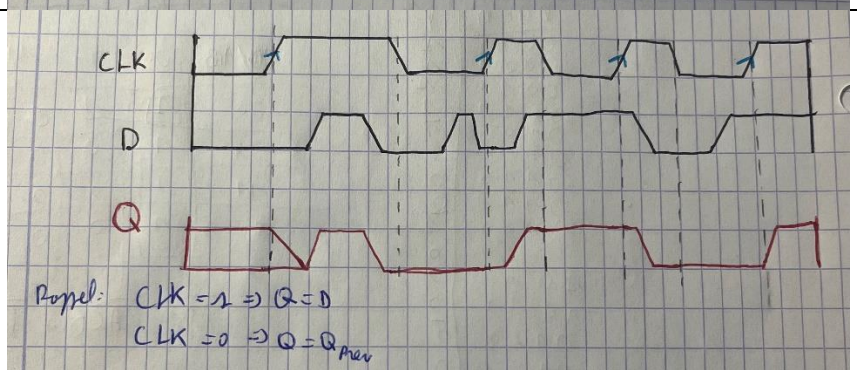
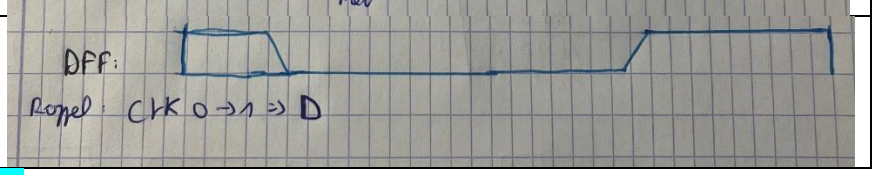
Quand CLK passe de 0 à 1 ➔ Q = valeur actuelle du D







Exemples Exos :

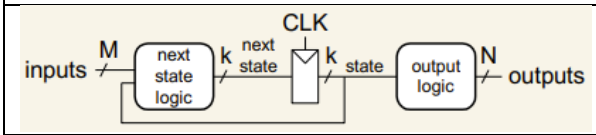
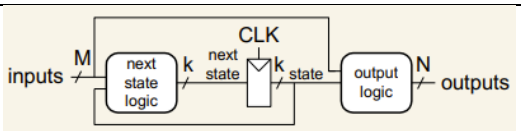
SR-Latch	
D-Latch	
D-FF	

FSM – Finite State Machine

Une FSM est un circuit qui a pour but de calculer son état futur sur base de son état actuel et des entrées. Pour ce faire il existe deux grands types de FSM : FSM de Moore et de Mealy.

Une **FSM** ou plutôt une ***Finite State Machine*** est une machine qui a des *états* qui sont sauvegardés dans un registre, et en fonction de ces états, nous allons calculer des sorties possibles. Il y a *2 types* de FSM :

1. FSM de Moore : sa sortie ne va dépendre uniquement de l'état (S0,…Sn), et en fonction de cet état, nous aurons des équations logiques et une/plusieurs sortie(s) qui va être calculée. On va calculer la sortie avec uniquement les états
2. FSM de Mealy : sa sortie peut dépendre non seulement des états mais aussi des entrées. On va calculer la sortie avec les états et les entrées

Moore	Mealy
Entrées	***Entrées + états***
	

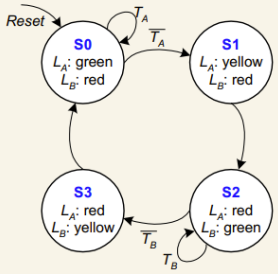
S0 = Reset

S1 sera atteint si l'entrée est égale au bon chiffre, si l'entrée n'est pas égale au bon chiffre, alors on retourne dans l'état S0

S2 (le troisième état) sera atteint si l'entrée est égale au bon chiffre, si l'entrée n'est pas égale au bon chiffre, alors on retourne dans l'état de *reset* et on revient en S0

Exemple de FSM :

- FSM State Transition Diagram
  - » Moore FSM: outputs labeled in each state
  - » States : Circles
  - » Transitions : Arcs



- FSM State Transition Diagram

Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

- FSM Encoded State Transition Table

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\overline{S_0}\overline{T_A} + S_1\overline{S_0}\overline{T_B}$$

- FSM Output Table

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Output	Encoding
green	00
yellow	01
red	10

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1}S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1S_0$$

On obtient les formules de S'1 & S'0 en essayant de trouver des combinaisons logiques qui donne le bon résultat pour S'1 et S'0

Autres formules importantes à connaitre :

$$T_c >= T_{pcq} + T_{pd} + T_{setup}$$

$$T_{cd} >= T_{hold} - T_{ccq}$$

$$\text{Fréquence} = 1 / T_c$$