Chapitre 1

Numériser: c'est pour se présenter les choses, images nombres, mots parlés, par des symboles.

ex: 10 chiffres - 10 doigts

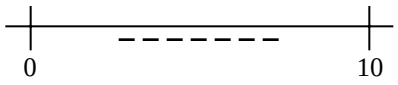
ex: image TV: 250,000 points image-pixel-30ms

ex: lettres de l'alphabet

→ en design numérique: représentation BINAIRE - base 2

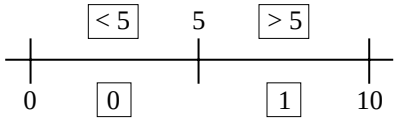
→ numérique: revient à catégoriser tout avec 2 états

1.1 Continu au discret

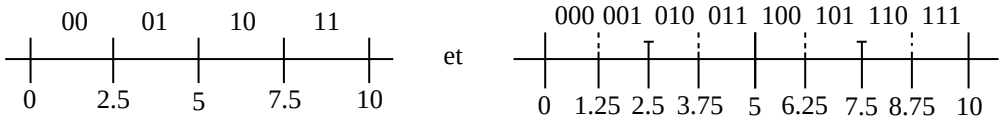


contient un nombre infini de points car domaine continu

(peut être numérisé en divisant par 2 et en assignant symbole 0 et 1



et on poursuit car cette division est approximative



en ajoutant des 0 et des 1. Ici on a 8 nombres disponibles dans cette représentation. C'est en fait une conversion A/N!

• • • • • • • •

000 001 010 011 100 101 110 111

Pourquoi numériser?

On vit dans un monde numérique: calculs scientifiques, transactions banquaires, enregistrements numériques, bientôt, la télévision, "l'âge de l'information" → l'âge du code binaire.

-beaucoup d'avantages (+que d'inconvénient)

1 Les données numériques sont moins sensibles au bruit (P/r à data analogue) pour l'acquisition, la transmission et la mémorisation de données).

ex signal: A[v] = signal + bruit
fantôme image TV, disque, gravé musique
signal radio contaminé
code anti-bruit redondant

}changement de status si bruit > largeur de zone

2 Traitement (mémorisation par circuits à semi conducteur à coût raisonnable:

-millions de transistors ou interruptions sur une puce silicium
-changent des millions des fois par seconde
-consomment peu d'énergie
coût μ$/switch

3 Combinaison facile avec théorème booléens

-1854 Geroge Boole -publie théorie mathématique à ce propos ensemble de théorèmes avec des relation vrai-faux
-algèbre de Boole permet de minimiser l'usage du hardware.
-2 états: vrai - faux et 1-0

1.2 Avantages de 2 états

-disponibilité d'interrupteurs dans les ic

$$\begin{cases} 1 - fermé \\ 0 - ouvert \end{cases}$$

-interrupteurs à la base des portes, à la base des circuits numériques arithmétiques.

Encodage: consiste à représenter quelque choses par un code

Question encoder l'ensemble {0,1,2,3} en binaire

ex: chiffre

0	1	2	3
↕	↕	↕	↕
00	01	10	11

$3 \leq 4$

⇒2bits suffisent

Terminologie
- nibble - 4 bits
- byte -8 bits(ou octet)
- 1 string: suite de bits (ou chaîne)

1.3 Représentation binaire

-important: chaque position à une signification particulière

$$01010 \neq 00110$$

-en binaire le codage se limite à 2 symboles

-d'autres bases existent: 10,8,16... (hexadéainal),...

-conversion base B dans base D

$$D = \sum_{i=o}^{i\,max} P_i B^i \leftarrow base$$

imax: nombre de positions -1 dans la base B.

P: Poids

ex: convertir le nombre 72_8 dans la base 10

$$P_o=2,\ P_1=7$$
$$D= 2 \times 8^0 + 7 \times 8^1 = 2 + 56 = 58_{10}$$

ex: convertir le nombre $(1234)_5$ dans la base 10

$$D= 4 \times 5^0 + 3 \times 5^1 + 2 \times 5^2 + 1 \times 5^3$$
$$=4 + 15 + 50 + 125 = 194_{10}$$

ex: convertir le nombre $(1234)_{16}$ dans la base 10

$$D = 1 \times 16^3 + 2 \times 16^2 + 3 \times 16^1 + 4 \times 16^0$$
$$=4096 + 2 \times 256 + 3 \times 16 + 4$$
$$=4660_{10}$$

Représentations

	Décimal	Binaire	Hexadécimal	Octal
	0	0000	0	0
	1	0001	1	1
	2	0010	2	2
	3	0011	3	3
	4	0100	4	4
	5	0101	5	5
	6	0110	6	6
	7	0111	7	7
	8	1000	8	10
BCP	9	1001	9	11
	10	1010	*A*	12
	11	1001	*B*	13
	12	1100	*C*	14
	13	1101	*D*	15
	14	1110	*E*	16
	15	1111	*F*	17
	16	10000	10	20

Exemple:

$10_H = 1 \text{ X } 16^1 + 0 \text{ X } 16^0 = 16_{10}$

$20_8 = 2 \text{ X } 8^1 + 0 \text{ X } 8^0 = 16_{10}$

4 bits permettent de coder $4^2 = 16$ états, de 0 à 15_{10}

symboles octal: {0,1,2,3,4,5,6,7} total 8 symboles
hexadécimal {0,1.....9, A,B,C,D,E,F} total 16 symboles

•BCD : Binary Coded Decimal: code les nombres décimaux sur 4 bits

On emploit 1 ensemble de 4 bits par chiffre

$\frac{0001 \quad 0000}{99_{10}}$, donc 5 bits en BCD

$99_{10} = 1001 \quad 1001$ *en* BCD (8 bits)

Utile dans les calculatrices notamment (évite les conversions fastidieuses, conversions directes).

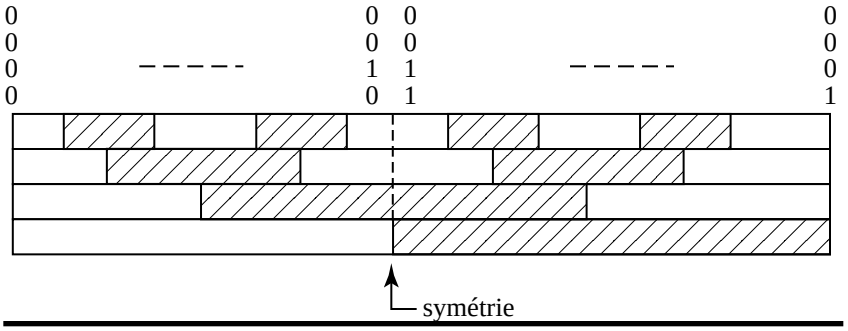
ex BCD 1024_{10} en BCD? $[0001\ 0000\ 0010\ 0100]_2$

•Gray code: d'un code à l'autre le code Gray ne change que d'un bit, il y a aussi une certaine symétrie .

1 – 0000
2 – 0001
3 – 0011
4 – 0010
5 – 0110
6 – 0111
7 – 0101
8 – 0100 { *axe* de symétrie
9 – 1100
10 – 1101
11 – 1111
12 – 1110
13 – 1010
14 – 1011
15 – 1001
16 – 1000

2 0001	15 1001
3 0011	14 1011
4 0010	13 1010
'	'
'	'
'	'
8 0100	9 1100

Utilité: encoder des entrées multiples qui changent 1 à la fois



Représentation de grands nombres en binaire

On rajoute simplement des bits

conversion octale ← binaire — aisée: simplement faire des groupements de 3 ou
hex← binaire — ou 4 bits.

exemple: $(10\ 10\ 11\ 01\ 01)_2$ en hex et octal

$$octal: \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{110}_{6}\ \underbrace{101}_{5} \rightarrow 1265_8$$

$$hex: \underbrace{0010}_{2}\ \underbrace{1011}_{B}\ \underbrace{0101}_{5} \rightarrow 2B5_H$$

vérifions: $1265_8 = 1 \times 8^3 + 2 \times 8^2 + 6 \times 8^1 + 5 \times 8^0$ $= 512 + 2 \times 64 + 48 + 5 = 693_{10}$

et $2B5H = 2 \times 16^2 + 11 \times 16^1 + 5 \times 16^0 = 2 \times 256 + 176 + 5$
$= 693_{10}$

exemple convertir $2B5_H$ en binaire et octal

groupes de 3 et de 4 $\Rightarrow \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{110}_{6}\ \underbrace{101}_{5} \Rightarrow (\underbrace{0010}_{2}\ \underbrace{1011}_{B}\ \underbrace{0101}_{5}$

$\Rightarrow 12658_8$

C'est ce qu'on appelle le codage positionnel puisque la position du code influence sa valeur. On verra plus tard le codage en point flottant.

1.4 Méthodes:conversion binaire à décimal:

1 On emploie la relation

$$D = \sum_{i=o}^{N-1} B_i 2^i$$

$B_i = 0$ ou 1, le i^{ième} digit et N = le nombre de bits

ex: 1010_2 ici N=4
$n = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$n = 8 + 0 + 2 + 10_{10}$

ex: $((1010\ 11\ 101)_2$ = ? en base 10

$1\ 2^8 + 0\ 2^7 + 1\ 2^6 + 0\ 2^5 + 1\ 2^4 + 1\ 2^3 + 1\ 1.2^2 + 0\ 2^1 + 1\ 2^0$

$256 + 0 + 64 + 0 + 16 + 8 + 4 + 0 + 1 = 349_{10}$

2 binaire→déc. on peut aussi d'abord convertir en hex puis en décimal

ex $(101011101)_2 = 15D_H$

$\underbrace{0001}_{1}\ \underbrace{0101}_{5}\ \underbrace{1101}_{D_H} = 1 \times 16^2 + 5 \times 16^1 + 13 \times 16^0$

$= 256 + 80 + 13$

$= 349_{10}$ m̂ résultat, moins de risque d' erreur

3 méthode 3: calculatrice ou Mathlab ou calculette sur windows

Calcul de fractions, c'est en fait le même procédé, mais avec des exposants négatifs.

ex: $(0.1)_2 = 0 \times 2^0 + 1 \times 2^{-1} = 0 + \frac{1}{2} = 0.5_{10}$

ex: $(0.1010)_2 = 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3} = 0.5 + 0.125 = (0.625)_{10}$

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$

Expression générale:

$$D_n = \sum_{k=-N}^{+m} B_k 2^k$$

pour le nombre

$Bm\ B_{m-1} \ldots B_2 B_1 B_0.\ B_{-1}\ B_{-2} \ldots B_{-n}$

1.5 Conversion **Base→10 au binaire**

1 Méthode par soustraction successive de la plus grande puissance de 2 on décompose le nombre en somme de puissance de 2.

ex: $33_{10} = (?)_2$
$33_{10} = 32 + 1 = 2^5 + 2^0 = (10001)_2$

ex: $1997_{10} = (?)_2$

1997 $= 1024 + 512 + 256 + 128 + 64 + 8 + 4 + 1$
$= 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^3 + 2^2 + 2^0$

1	1	1	1	1	0	0	1	1	0	1
10	9	8	7	6	5	4	3	2	1	0

$(11111001101)_2$

2 Méthode par division successive. On peut diviser* par 2 ou plus simplement par 16 pour convertir en hex directement

ex: $(33)_{10}$

$16^1 = 16$
$16^2 = 256$
$16^3 = 4096$

33/16 = 2

donc 2 x 16 + 1 = 33
(21)hex = 0010 0001 $(10\ 0001)_2$

*avec la plus grande puissance possible

ex $(1997)_{10}$ =1997/256 = 7 reste 205, 205/16 = 12 reste 13

$= 7CD_H = (\underbrace{0111}_{7}\ \underbrace{1100}_{12}\ \underbrace{1101}_{13})_2$

ex: $321_{10} = (?)_2$

321/256 = 1 reste 65, 65/16 = 4 reste 1
321 = (141)H $(0001\ 0100\ 0001)_2$
rép. $(101000001)_2$

En divisant par 16, ça va plus vite que par 2 et la conversion Hex $\rightarrow$ Binaire est directe.

Conversion de fractions

Le processus peut être sans fin, il faut donc spécifier une résolution ex: sur 8,16, 32 bits.

-Les mêmes méthodes s'appliquent.

ex $(0.4)_{10} = (?)_2$

$0.4 = 0.25 + 0.125 + 0.015625 + ...$

$=[0.390625]$

$=2^{-2} + 2^{-3} + 2^{-6} + ...$

donc

0.0	1	1	0	0	1
–1	–2	–3	–4	–5	6

$(0.011001)_2$

Table 1.4 Table des fractions

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$
$2^{-4} = 0.0625$
$2^{-5} = 0.03125$
$2^{-6} = 0.015625$

1.6 Troncature et arrondissement

Si on stoppe simplement le calcul de la fraction après un nombre arbitraire de chiffres, on parle de "troncature".

ex $3.14159 \leftrightarrow \pi$ tronqué à 5 chiffres après le point

On peut aussi compléter, arrondir la dernière position pour + de précision

Ex $\underbrace{3.1416}_{arrondir}$ plus précis que $\underbrace{3.1415}_{tronquer}$

Sujet traité plus en détail en "Analyse numérique pour ingénieur."

1.7 Addition de nombres binaires non-négatifs

Avec la somme de 2 bits:

$$4\ cas \begin{cases} 0 + 0 = 0 & \hookleftarrow \text{somme} \\ 0 + 1 = 1 & \\ 1 + 0 = 1 & \\ 1 + 1 = 0 & \text{retenue}_1 \quad \text{somme}_o \end{cases}$$

En additionnant 2 nombres binaire de plusieurs bits, cela revient au fond à additionner 3 bits (avec la retenue). Examinons les cas possibles:

0	0	0	0	1	1	1	1}
0	0	1	1	0	0	1	1}
0	+1	+0	+1	+0	+1	+0	+1}
00	01	01	10	01	10	10	11

Retenue
nombre A
Nombre B
résultat: retenue et somme

ex additionner 7_{10} + 22_{10} en binaire

00111
22 16+4+2= 10110
$(11101)_2 = 1D_H = 16 + 13 = 29_{10}$

1.7 Addition modulaire

On peut représenter l'addition de 2 bits par un bloc

A	B	somme	retenue
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A-
B-

C'est le demi additionneur ou Half-Adder

- somme
-retenue

On verra plus tard comment réaliser ce bloc en portes discrètes et maintenant l'additionneur complet sur 1 bit (1 bit Full adder) 1BFA

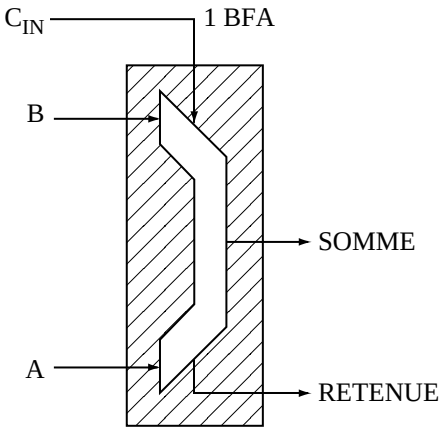
Cin	A	B	somme	retenue

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

Table du demi – additionneur

C_{in} HA (Half - Adder)

B



somme

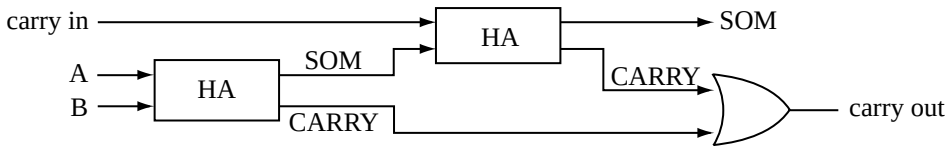
retenue

ou

carry

out

On peut tout de suite voir le développement du 1BFA en termes de HA



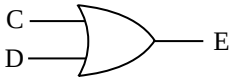
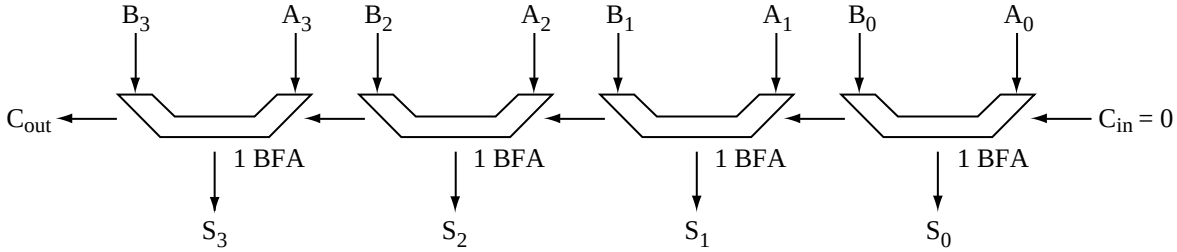


Table de vérité ou

CD	E
00	0
01	1
10	1
11	1

Ce qui se voit bien de la table du IBFA en considérant le cas avec C_{in}=0 (on revient à un et puis le cas avec le carry à 1=C_{in} ou l'on fait: somme partielle + C_{in}

Pour des additions généralisées, on juxtapose des 1BFA puisque sur (4 bits): {B3,B2,B1,B0} + {A3,A2,A1,A0} = {tout, S_3,S_2,S_1}



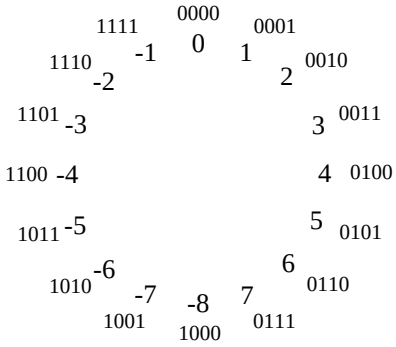
C'est un exemple de bon design, puisque modulaire (ie concevoir un système à partir de sous-systèmes génériques). Requiert peut-être plus de composants qu'une approche directe, mais + facile à traiter/réparer) moins coûteux à long terme.

1.8 Soustraction et modulaité

A-B = A+(-B) donc on peut employer une approche avec IBFA modulaire en exprimant en binaire un nombre négatif (-B) Par les compléments 2:

1^0- choisir un nombre max de bits
2^0- poser que le bit à gauche (le + significatif): bit de signe
3^0-inverser tous les bits du chiffre puis additionner 1

Sur 4 bits ça peut se représenter par une roue:



ex:-7=?

+7	0111
⇓	1000 + 1
–7←	1001

ex: avec zéro: 0000: +0

1111
1
0000-0

exemple: calculer 7-3?

7-3-= 7+(-3) = (0111)+(1101)

+3:	0011		0111	+7
	1100	+	1101	-3
+	1		0100	+4
-3	1101		notre notation est sur 4 bits,	
			-on rejette donc le 5ième bit.	

exemple calculer 7+3

0111: +7
\+ 0011: +3
1010: -6!!!! c'est qu'en notation complément 2 sur 4 bits,+ 10 n'existe pas.

Les valeurs possibles vont de [-8,+7] (7+3=10) C' est ce qu'on appelle un overflow ou dépassement. Pour l'éviter ici, on aurait dû travailler sur une notation com. 2 à 5 bits plutôt.

Détection d'overflow: comparer les signes des nombres additionnés ensemble:

si on additionne 2nb de $\hat{m}$
signe, et que le signe de la somme est différent, c'est qu'il y a eu overflow!.
"carrry out", "overflow" sont des bits de status, utiles pour le système.

note: •overflow: indique une erreur

•carry out: peut servir pour des calculs subséquents

important notation en complément 2 (C2) permet d'employer les modules vus précédemment pour réaliser des +, -, mais il faut bien déterminer un préalable la longueur max des mots.

1.9 Addition de fractions binaires

Le mêne procédé d'addition avec ou sns C2 demeure valide)

Ex1 $(.75 + 1.25)_{10} = 2_{10}$

0.1	2^{-1}	0.5
0.01	2^{-2}	0.25
0.001	2^{-3}	0.125

↓ ↓

0.5+0.25 1.01 et 1.01

0.11 +0.11

10.00=$+2_{10}$

Ex2 Calculez (1.25 -0.75) avec C2 en binaire

On choisit un système à 4 bits pour éviter les overflow.

$+1.25_{10} \leftrightarrow (1.01)_2$

$+0.75_{10} \leftrightarrow$ C2 : 11.01

1.25 01.01

+(-.75) 11.01

00.10 =0.5 qui est correct

1.8 Codes et Encodage

On a vu les codes suivants:
- binaire positionnel
- Gray
- BCD
- C2 pour nombre signés

d'autres sont disponibles:
- musique
- code morse (1840) inventeur 20 mots/minutes du télégraphe
- code à Barres (ex UPC: 10 digits)
- ASCII: 7 bits code représentant 128 caractères (a...z, A....Z,0,....9), symboles typographiques un 8 bit sert pour la parité (#total bit à 1 est pair ou impair)

(note en ASCII les chiffres sont encodés en BCD sur 8 bits.)

Pour récupérer le nombre BCD, il suffit donc simplement de "stripp off" (un pack) tous les bits sauf $B_3 \sim B_0$

exemple $0 \leftrightarrow$ 30 Hex 0110000 donc 011 et 0000

$9 \leftrightarrow$ 39 Hex 0111 001 donc 011 et 1001

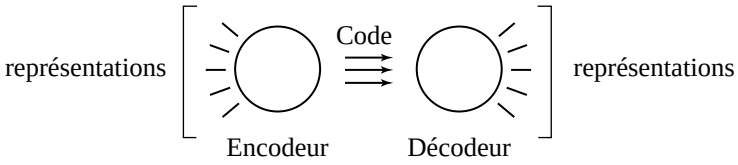
Caractéristiques d'un bon code

-s'il est : "standard", accepté par tous $\Rightarrow$ meilleur

-rapidité: (pour encoder) (décoder)
-fiabilité: limite les erreurs lors du codage si rapide ⇒ moins de risque d'erreur
-compression de l'entrée: i.e. moins de lignes de sortie que de lignes d'entrées, critique pour par ex: compression de signaux TV ex BCD pour nombre 0 à 99 ⇒ 8 bits
alors qu'en binaire 99_{10} = 64+32+2+1=100011 ⇒ 7 bits seulement.

-compatibilité "acquiscement"

-sécurité: encryptage avec une clé



1.10 Des interrupteurs aux portes

-interrupteurs/robinets/valves: connus depuis des lustres

ex: aqueduc, instruments de musique

ouvert/fermé: 2 états - différents niveaux d'ouverture

ex: manette de toilette ex: -potentiomètres, robinet de douche
-gradateur d'intensité

utile facilement réalisable en silicium
-commutable à grande vitesse $>10^6$ fois/s
-dissipe: peu d'énergie
-occupe: peu de place (microns)

coûte presque rien

Pour comprendre l'électronique numérique ça commence avec les interrupteurs.
1 0
ouvert- fermé
comme un robinet: fermé l'eau ne passe pas
ouvert l'eau passe - électrons, photons, etc.

Distinction entre interrupteurs normaux et interrupteurs numériques: capacité d'un SW de commander un autre SW (effet en cascade) + interconnexions.
donc pour un SW, schéma + général:

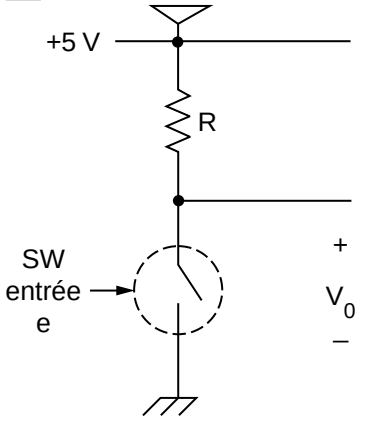
Si C >seuil ⇒ SW fermé
Si C < seuil ⇒ SW ouvert

seuil ~2.5 volt (pour famille TTL, 5 volt)

Porte=Circuit avec un ou plusieurs SW qui a une fonction particulière et versatile pour être un bloc d'un système, plus grand.

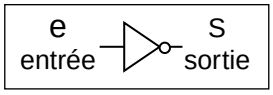
ne pas confonde avec la gate d'un transistor MOS

1 INVERSEUR: porte inverseuse



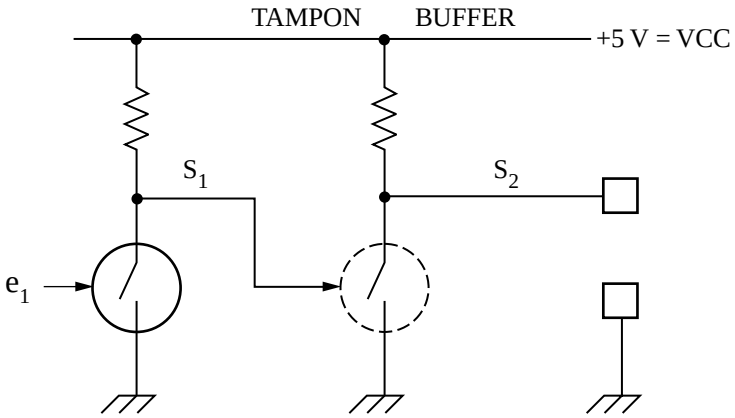
entrée	sortie	
0	1	c' est un inverseur
1	0	

symbole



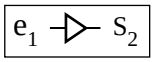
INVERSEUR

2 TAMPON (BUFFER)



$e_1 = S_2$

Buffer du tampon



3 PORTE ET et NON-ET (NAND)

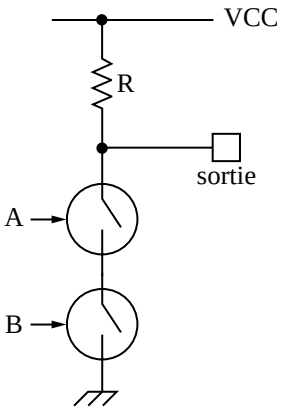
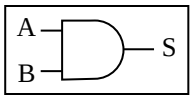


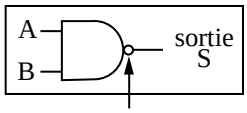
Table de vérité

A	B	S(ET)	S(NON-ET)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0



ET

NON - ET

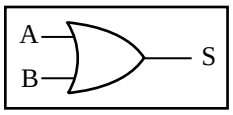


bulle

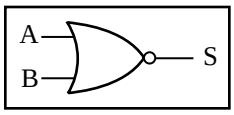
4 PORTE NON- OU (NOR)

TABLE DE VÉRITÉ			
A	B	S(OU)	S(NON-OU)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

symboles:

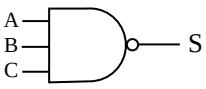


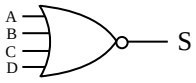
OU
(OR)



NON - OU
(NOR)

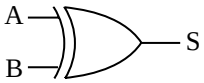
5 On peut aussi avoir des portes à entrées multiples





6 <u>Porte ou Exclusif</u> (XOR) Porte plus complexe

Symbole XOR



Symbole (NXOR)

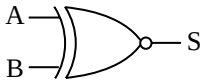
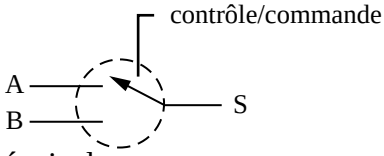


Table de vérité

A	B	S(XOR)	S(NXOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

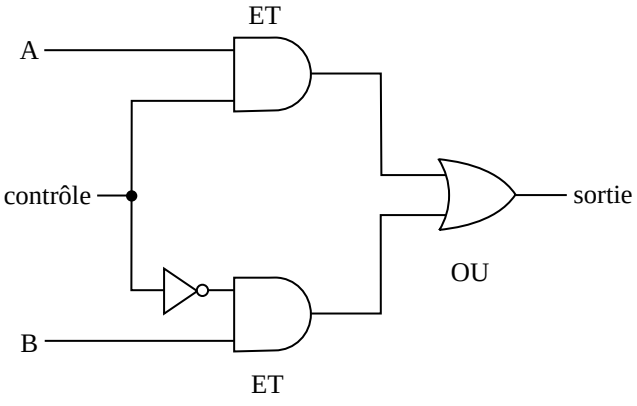
1.11 Multiplexeurs

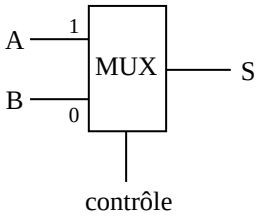
On veut ici créer un circuit qui permet de "passer/transférer" un signal numérique d'une ou de l'autre origine A au B.



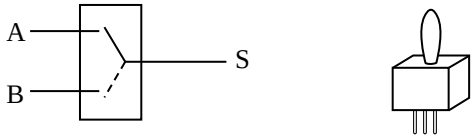
contrôle	sortie S
1	A
0	B

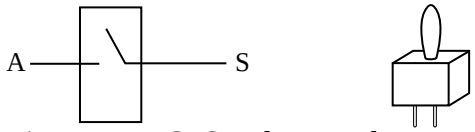
équivalent en portes:





c'est en fait un SPDT "single pole - double throw"

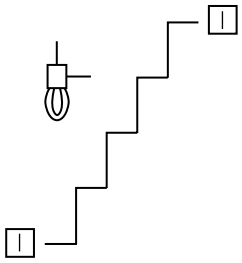




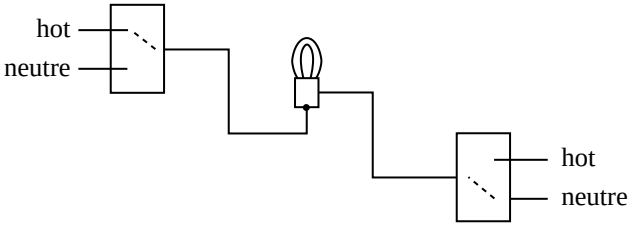
P/rapport au SPST plus simple

exemple: d'utilisation circuits d'escalieur "three-way"

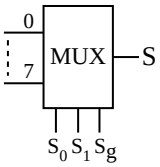
on veut réaliser un circuit permettant d'éclairer un luminaire à partir de 2 SW

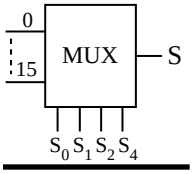


Le circuit emploie 2 SPDT SW



un SPDT est un MUX 2 à 1 (2 entrées 1 sortie)
Il existe aussi des MUX plus importants:

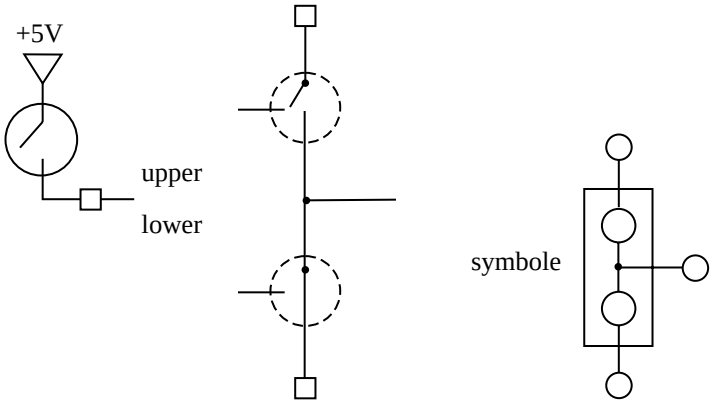




1.12 Sortie flottante

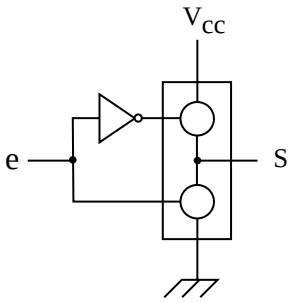
Problème: un simple SW a sa sortie flottante si ouvert. Cela n'est pas très pratique car cela mène à des incertitudes! Ce problème peut se régler au moyen de circuits à TOTEM POLE

C'est ce qu'on appelle un arrangement qui peut être compris de la façon suivante: 2 interrupteurs en cascades.



On peut le connecteur comme ceci:

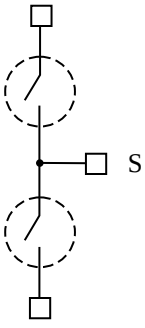
Si les 2 SW sont ouverts ou fermés jamais en même temps, la sortie est toujours définie Hi ou Lo.



Cet arrangement se construit bien en CMOS (complementary Metal Oxide Semi-conducteur).

1.13 Troisième état

Lors de la connexion de plusieurs portes ensemble, on a souvent besoin d'un 3ième état "haute impédance" ou "état flottant" pour éviter d'endommager un circuit.



ça revient à notre arrangement "Totem Pole" avec les 2 SW ouverts.

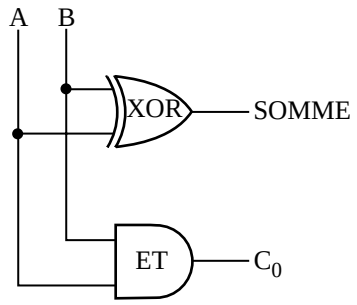
Exemple d'utilisation Mémoires, Buses

1.14 Circuits pour additionneurs

A	B	Somme	Retenue Co

On a vu que:	0	0	0	0	on remarque que C_o=AB Somme = A ⊕ B
	0	1	1	0	
	1	0	1	0	
	1	1	0	1	

avec donc le design suivant



On verra dans les chapitres suivants des méthodes systématiques de design.

pour par exemple -minimiser la taille des design
-minimiser les délais des propagation ("glitches")

1.15. CAD pour circuits design

2 approches -traditionnelles (utile petit circuit), à la main
-CAD (utile pour grand circuit)

Les méthodes traditionnelles

emploient -MSI sur PCB wirewrap
-ex: 7400 séries TTL
voir livre TTL databook de Texas Instruments.

-méthodes sur papier, gros design: on découpe en sous bloc
-changement → travail de re-design majeur

Encore employée pour la "glue logic" entre parties plus importantes réalisées en CAD

Méthodes CAD

-temps de mise au point rapide pour
- systèmes importants
e.g. contrôleur numérique d'autos
-emploie des chips programmables, coût réduit
-fiabilité
-utilisation de langage CAD e.g. ABEL, VHDL, Verilog

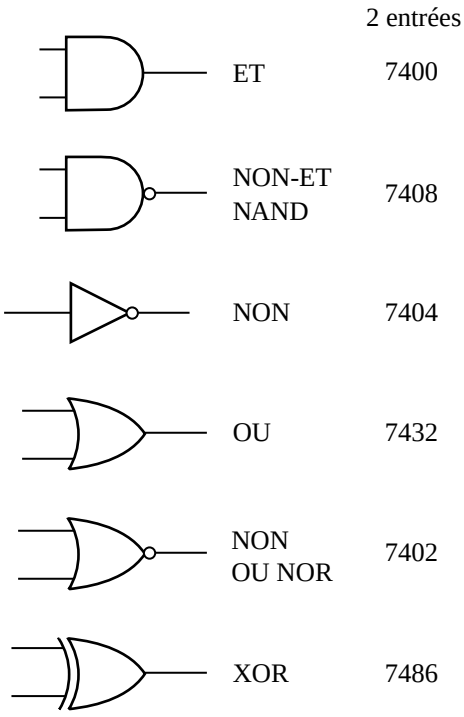
Étapes:

Highlevel design	outils de synthèse	simulateur	Hardware programmable
EX: TTL 0.50$ FPGA 10$ (FPGA= 50TTL)		trouve problème de timing de conception de charge	-FPGA -PROM -ROM -PAL

D'autres cours en gel-gif traitent de ces domaines

Familles logiques - Considérations pratiques.

Les portes simples sont disponibles dans la famille TTL sous formes des circuits SSI (small scale intégration)

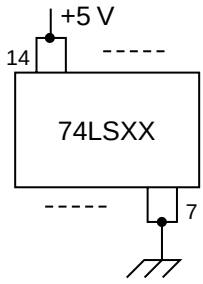


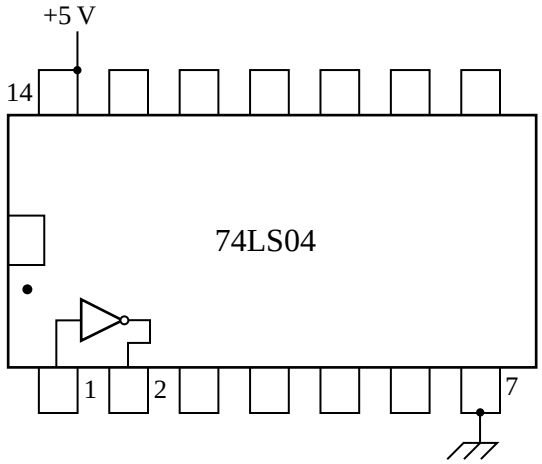
La Famille TTL nécessite une alimentation entre les pins 14 et 7

Les numéros de circuits ci sont 74LSXX
D'autres familles sont disponibles:

-CMOS, série 4000
-en version 3.3 volt DC aussi

Le labo portera sur une première familiarisation avec les portes éléemtaires SSI.





Dans le logiciel Xilinx Foundation, les circuits TTL se retrouvent sous l'appellation "x74xxx".