Exercices corrigés
(architecture ordinateurs et circuits logiques)

A- Questions de culture générale (non corrigées ici)

1) Comment fonctionne le « tactile » d'une tablette tactile ?

2) Qu'est-ce qu'un *ripper* de DVD ?

3) De multiples phénomènes sont cycliques, et le nombre de cycles par seconde (ou fréquence) est exprimé avec une unité appelée Hertz (*Hz*).[1]

a) La Terre fait un tour complet toutes les 24 heures. Quelle est la fréquence associée ?

b) Quelle est la fréquence du courant alternatif ?

c) Quelle est la fréquence d'horloge d'un ordinateur ?

d) Dans quelle zone se situent les fréquences acoustiques perceptibles par l'être humain ?

e) Donner les ordres de grandeur des fréquences électromagnétiques pour :

* les ondes radio
* les ondes des fours à micro-ondes
* l'infrarouge
* la lumière visible
* l'ultra-violet
* les rayons X
* les rayons cosmiques

f) Entre les micro-ondes et l'infrarouge se trouve la zone du *térahertz* (*THz*). Ce domaine est actuellement très étudié. Quelle est une application (critiquable !) de ce genre de fréquences ?

4) Qu'est-ce qu'une clé USB et comment fonctionne-t-elle ?

5) Les disques électroniques SSD (*solid state drive*) vont-ils supplanter les disques durs HDD (*hard disk drive*) ?

B- Algèbre binaire

1) Addition de nombres entiers binaires « signés »

1-A) On considère ces opérations écrites en base 10 :

a) – 61 – 44

b) – 61 – 72

c) 99 – 35

d) 99 + 35

On dispose d'une machine travaillant sur des nombres binaires de longueur 8 (8 bits). Faire manuellement ce que l'additionneur de la machine ferait automatiquement, et donner les résultats obtenus en binaire. Eventuellement, en cas d'erreur, indiquer pourquoi.

---

[1] Rappel sur les unités :
$M : 10^6$, $G : 10^9$, $T : 10^{12}$, $P : 10^{15}$

On commence par écrire les nombres positifs en base 2 sur 8 bits, en procédant avec des divisions par 2 successives. A partir de là si l'on veut le nombre avec un signe moins, on prend le complément et l'on ajoute 1. Puis on additionne les nombres concernés.

a) 61 = 00111101 – 61 = 11000011
44 = 00101100 – 44 = 11010100
Par addition :
11000011
+11010100
1**10010111 ,** on supprime le bit de trop. On a bien 10010111 = – 128 + 16 + 4 + 2 + 1= – 105 et – 61 – 44 = – 105

b) 72 = 01001000 – 72 = 10110111
– 61 11000011
– 72 10111000
– 133 1**01111011 ,** on supprime le bit de trop, le résultat est faux (il est positif). Il y a débordement (*overflow*) : on est en dehors de la zone entre –127 et +127 correspondant aux nombres signés de 8 bits. Cela peut se tester en constatant que les deux dernières retenues à gauche sont 10.

c) 99 = 01100011 35 = 00100011 – 35 = 11011101
99 01100011
– 35 11011101
64 1**01000000** , on supprime le bit de trop. Le résultat est juste (c'est toujours le cas pour une vraie soustraction puisqu'il ne peut pas y avoir *overflow*).

d)
99 01100011
35 00100011
134 10000110 , aucun bit de trop, mais le résultat est faux (il est négatif). Il y a *overflow* (le résultat 134 n'est pas dans la zone de -127 à 127), ce qui se teste en constatant que les deux dernières retenues (en position 8 et 7) sont 01.

1-B) Addition de nombres entiers signés. On travaille ici sur des nombres de 8 bits.
a) Ecrire les nombres 109 et 88 (base 10) en binaire signé sur une longueur de 8 bits.
b) Ecrire les nombres **–** 109 et **–** 88 en binaire signé sur 8 bits, en passant par l'intermédiaire du complément à 2.
c) Faire les additions suivantes, en utilisant les résultats précédents :

- 109 – 88
- – 109 – 88
- – 109 + 88

En cas d'erreur pour cause de débordement, indiquez-le.

a) 109 = 01101101
88 = 01011000
b) – 109 = 10010011
– 88 = 10101000

c) 109 – 88 : 01101101
10101000
~~1~~**00010101** (on supprime comme toujours le bit de trop, qui n'a rien à voir avec un problème de débordement)**.** On a bien 109 – 88 = 22, soit 10101 en binaire

– 109 – 88 : 10010011
10101000
~~1~~**00111011** résultat faux, puisqu'il est positif (une fois supprimé le bit de trop). C'est normal puisque le résultat n'est pas compris entre – 128 et 127, où se trouvent les nombres signés sur 8 bits. Il y a débordement.

– 109 + 88 : 10010011
01011000
11101011 et l'on a bien – 109 + 88 = – 21, soit 11101011 en binaire.

2) Changement de base

a) On se donne le nombre 32745 en base 8. Comment s'écrit-il en base 16 ?
Indication : passer par l'intermédiaire de la base 2.

On prend chaque chiffre de 32745 en le convertissant en binaire sur 3 bits, ce qui donne :
011 010 111 100 101. C'est le nombre converti en binaire. Puis on fait une lecture de ce nombre par blocs de 4 bits à partir de la droite, ce qui donne ici :
0011 0101 1110 0101. Puis on remplace chacun de ces blocs par un chiffre entre 0 et 15, en utilisant les lettres *A*, *B*, …, *F* à partir du « chiffre » 10, soit :
35E5, ce qui est l'écriture du nombre en base 16.

Autre méthode (plus longue) : on commence par écrire 32745 en base 10, en utilisant la définition du nombre en base 8 :
$32745 = 3 . 8^4 + 2 . 8^3 + 7 . 8^2 + 4 . 8 + 5 = 13\,797$
Puis on fait des divisions successives par 16 à partir de 13797 jusqu'à avoir un quotient nul, et on lit les restes en remontant, soit 3, 5, 14, 5, ce qui donne bien 35E5.

b) Un nombre s'écrit 753 en base 8. Comment s'écrit-il en base 16 ?

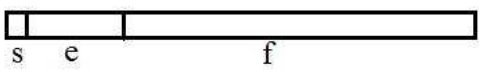
753 = 111 101 011 en binaire
= 0001 1110 1011 réécrit par blocs de 4 à partir de la droite
= **1 E B** en hexadécimal

Autre exemple pour s'entraîner : 765 en base 8 devient 1F5 en base 16.

3) Nombre flottant en binaire

3-a) On se donne le nombre 35,6 en base 10. Le convertir en flottant sur 32 bits (simple précision).

Rappelons qu'il convient de l'écrire sous la forme :
$(-1)^s . 2^{e-127} . (1 + f_1 . 2^{-1} + f_2 . 2^{-2} + f_3 . 2^{-3} + \ldots + f_{23} . 2^{-23})$ avec *s*, *e* et *f* qui occupent chacun une partie des 32 bits, respectivement 1, 8 et 23 bits.



On a déjà $s = 0$.

Ensuite, on écrit le nombre en virgule fixe : la partie entière est 35, soit 100011 en binaire, et pour ce qui est derrière la virgule, soit 0,6 , on fait des multiplications par 2 modulo 1 pour le convertir en binaire :

0,6 . 2 = 1,2 1
0,2 . 2 = 0,4 0
0,4 . 2 = 0,8 0
0,8 . 2 = 1,6 1
0,6 . 2 = 1,2 1
etc.

Le nombre en virgule fixe est 100011,1001100110011…

En virgule flottante, il s'écrit comme une puissance de 2 multipliée par un nombre qui a comme partie entière 1, soit ici 2^5 . 1, …, car on doit déplacer la virgule de 5 crans à gauche, ce qui donne : 1,00011100110011<u>0011</u>… et $e - 127 = 5$, soit $e = 132 = 10000101$

Finalement le nombre s'écrit 0 10000101 00011100110011001100110.

3-b) On considère le nombre à virgule : 73,55 (base 10)

a) Ecrire ce nombre en binaire en virgule fixe.

b) Ecrire ce nombre en binaire en virgule flottante, en simple précision (sur 32 bits). Le résultat obtenu est-il parfaitement exact ?

a) 73 s'écrit 1001001. Et pour 0,55 on fait des multiplications par deux modulo 1

0,55 . 2 = 1,1 1
0,1 . 2 = 0,2 0
0,2 . 2 = 0,4 0
0,4 . 2 = 0,8 0
0,8 . 2 = 1,6 1
0,6 . 2 = 1,2 1
0,2 . 2 = 0,4 0
0,4 . 2 = 0,8 0

0,55 s'écrit 0,1000110011…

73,55 s'écrit en virgule fixe 1001001,10001100110011…

b) Décalons la virgule de 6 crans, le nombre devient 1,0010010111001100… et l'on doit le multiplier par 2^6. Le nombre est positif, d'où $s = 0$, la puissance de 2 est telle que $e - 127 = 6$, soit $e = 133$, ce qui donne 10000101 en binaire. Finalement, le nombre s'écrit :

0 10000101 00100110 0011 0011 0011001

Puisque ce nombre est tronqué, ce qui est au-delà du 23è chiffre après la virgule étant négligé, il n'est pas exactement égal au nombre 73,55.

4) Multiplication : Ecrire les deux nombres 27 et 22 (base 10) en binaire (descendant). Puis faire la multiplication de ces deux nombres en binaire. Vérifier le résultat obtenu en binaire en faisant la multiplication en base 10.

27 = 11011

22 = 10110
Faisons la multiplication :

```
      11011
    × 10110
    -------
      00000
     11011
    11011
   00000
  11011
  ----------
  1001010010
```

Vérifions : 27 . 22 = 594 qui s'écrit bien 1001010010

C- Circuits logiques combinatoires

1) Simplification d'équation

On se donne l'équation $t = x\bar{y} + z(\bar{x} + y)$. Commencer par réécrire cette équation sans parenthèses, avec trois termes.

a) Première méthode de simplification : construire la table de vérité, puis le tableau rectangulaire de Karnaugh avec *xy* d'une part et *z* d'autre part. En déduire la forme simplifiée de *t*.

b) Deuxième méthode :

* Commencer par démontrer le théorème du consensus, $XY + \bar{X}Z + YZ = XY + \bar{X}Z$, en utilisant une table de vérité avec l'équation qui en découle, ou bien deux tables de vérité.

* Puis, en utilisant le théorème du consensus, réécrire $x\bar{y} + \bar{x}z$, et en déduire la forme simplifiée de *t*.

a) La table de vérité de $t = x\bar{y} + z\bar{x} + yz$ conduit à l'équation :

$t = \bar{x}\bar{y}z + \bar{x}y z + x\bar{y}\bar{z} + x\bar{y}z + xyz$. Simplifions-la avec un tableau de Karnaugh.

z \ xy	00	01	11	10
0	0	0	0	1
1	1	1	1	1

$t = x\bar{y} + z$

b) La table de vérité de $XY + \bar{X}Z + YZ$ donne :

$XY + \bar{X}Z + YZ = \bar{X}\bar{Y}Z + \bar{X}YZ + XY\bar{Z} + XYZ$ qui se simplifie :

$= \bar{X}Z(Y + \bar{Y}) + XY(Z + \bar{Z}) = \bar{X}Z + XY$. C'est le théorème du consensus.

Grâce à ce théorème, on peut écrire : $x\bar{y} + \bar{x}z = x\bar{y} + \bar{x}z + \bar{y}z$, et *t* devient ;

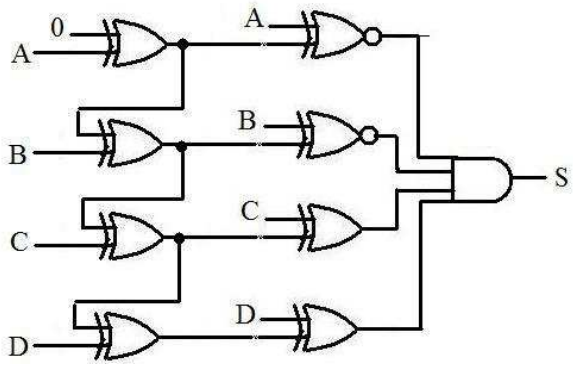
$t = x\bar{y} + \bar{x}z + yz = x\bar{y} + \bar{x}z + \bar{y}z + yz = x\bar{y} + \bar{x}z + z(y + \bar{y}) = x\bar{y} + \bar{x}z + z$

$= x\bar{y} + (\bar{x} + 1)z = x\bar{y} + z$

2) Portes XOR

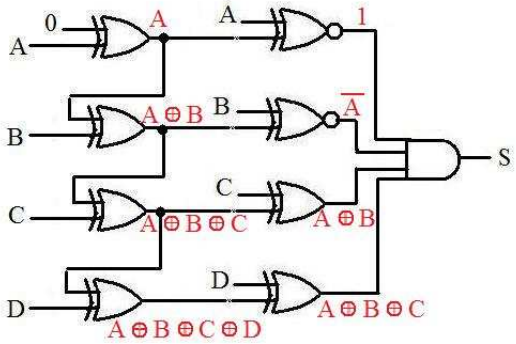
a) Que valent $0 \oplus a$, et $a \oplus a$?

b) On se donne ce circuit logique avec quatre bits d'entrées *A*, *B*, *C*, *D* et une sortie *S*. Montrer qu'il existe deux cas exactement pour les entrées aboutissant à *S* = 1 en sortie, et donner ces deux cas. Pour ce faire, ajouter sur le dessin les résultats obtenus à la sortie de chacune des portes XOR du schéma.



a) $0 \oplus a = a$, $a \oplus a = 0$ comme on le constate en faisant *a* = 0 puis *a* = 1. Notons que l'on a aussi $1 \oplus a = \overline{a}$.

b)

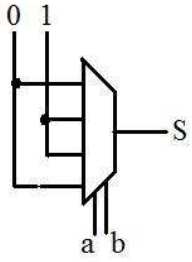


Pour avoir S = 1, il convient que toutes les entrées sur la porte ET valent 1, ce qui impose :

A = 0, $A \oplus B = 1$ d'où *B* = 1, puis $A \oplus B \oplus C = 1$, soit *C* = 0. Comme *D* est quelconque, on trouve bien deux solutions pour (*A*, *B*, *C*, *D*) : (0, 1, 0, 0) ou (0, 1, 0, 1).

3) Multiplexeurs (MUX pour les intimes)

3-a) On a ce schéma de multiplexeur 1 parmi 4. Qu'obtient-on en sortie, cette sortie *S* étant seulement fonction des deux bits *a* et *b* de sélection ? On supposera que les quatre entrées du multiplexeur sont numérotées de 0 à 3 de haut en bas (et le nombre *ab* est lu en binaire descendant).

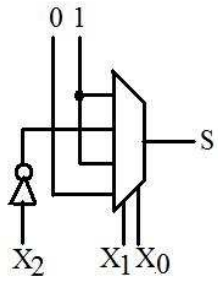


On a la table de vérité :

a	*b*	*S*
0	0	0
0	1	1
1	0	1
1	1	0

On reconnaît le XOR : $S = a \oplus b$.

3-b) Avec cet autre multiplexeur 1 parmi 4, combien vaut la sortie S en fonction des variables X_2, X_1, X_0, les deux dernières correspondant aux fils de sélection ? Puis simplifier cette équation grâce à un tableau de Karnaugh.



La table de vérité s'écrit :

*X*2	*X*1	*X*0	*S*
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

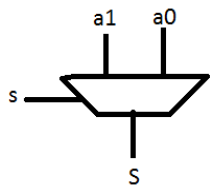
Pour simplifier construisons le tableau de Karnaugh :

X2 \ X1 X0	00	01	11	10
0	1	1	0	1
1	1	0	0	1

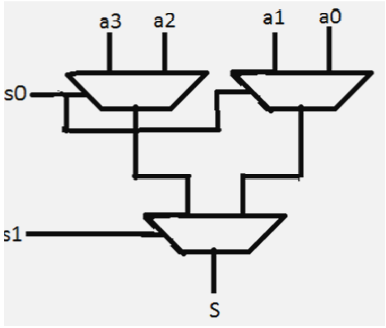
D'où l'équation $S = \overline{X_0} + \overline{X_2}\ \overline{X_1}$

3-c)

1)



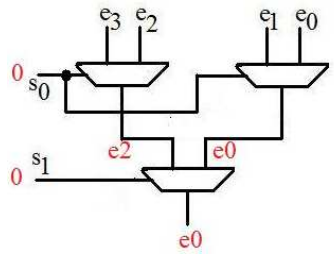
On rappelle qu'un multiplexeur 1 parmi 2 ayant en entrées *a*1, *a*0 met dans la sortie *S* ce qui est dans *a*0 (entrée de droite) lorsque *s* = 0 ou ce qui est dans *a*1 (entrée de gauche) lorsque *s* = 1. On utilise maintenant un circuit comportant trois multiplexeurs 1 parmi 2 comme indiqué sur le dessin ci-dessous. Sans faire de table de vérité, indiquer ce que l'on obtient en sortie quand le nombre *s*1*s*0 (en binaire descendant) prend les valeurs 00, 01, 10, 11.



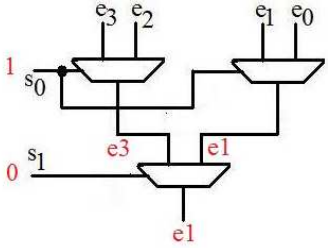
Quel type de multiplexeur correspond finalement à ce circuit ?

Les quatre cas possibles :

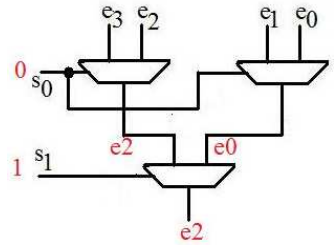
$s_1s_0 = 00$



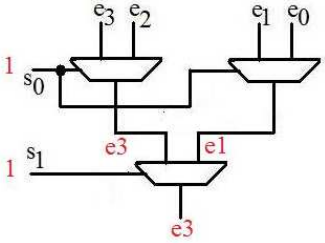
$s_1s_0 = 01$



$s_1s_0 = 10$



$s_1s_0 = 11$



On obtient ainsi un multiplexeur 1 parmi 4.

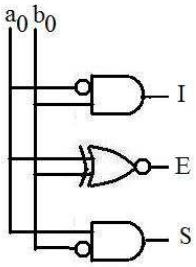
4) Circuit comparateur

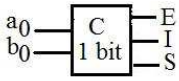
Ce circuit possède deux entrées qui sont des nombres en binaire, et trois sorties, *E*, *I*, *S*, indiquant respectivement si les deux nombres sont égaux (*E* = 1), et sinon *E* = 0, ou si le premier est inférieur au second (*I* = 1 et sinon 0), ou si le premier est supérieur (*S* = 1 et sinon 0).

a) Cas où les deux nombres binaires a_0 et b_0 sont de longueur 1. Donner les équations de *E*, *I* et *S*, et dessiner le circuit.

On a entrevu ce cas dans le cours. On trouve facilement les sorties en fonction des deux bits d'entrée, soit :

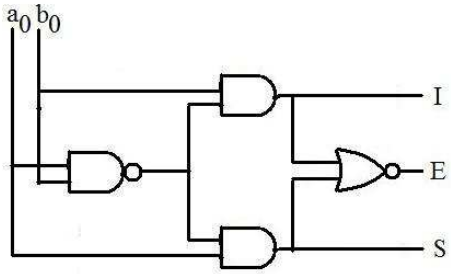
$E = \overline{a_0 \oplus b_0}$, $I = \overline{a_0}\, b_0$, $S = a_0\, \overline{b_0}$.





b) Vérifier que $a \oplus b = \bar{a}b + a\bar{b}$ et $\overline{ab}b = \bar{a}b$. En déduire le schéma du circuit sans utiliser de porte XOR.

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



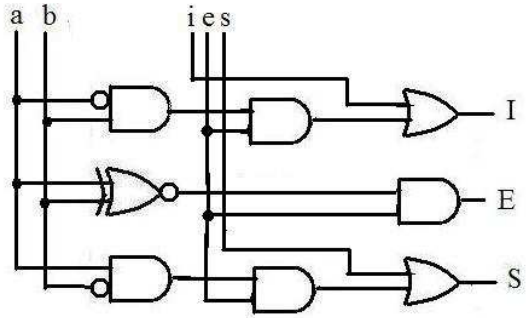
c) On prend deux nombres de deux bits chacun en entrée, soit a_1a_0 et b_1b_0 (binaire descendant). Donner les équations des trois sorties de ce comparateur 2 bits. Pour éviter de construire des tables de vérité complexes, exprimer, par exemple pour *E*, ce que signifie $a_1a_0 = b_1b_0$ en utilisant des relations entre bits, ce qui fera apparaître deux portes XOR en utilisant les résultats du comparateur 1 bit précédent, et faire de même pour *I* et *S*.

$a_1a_0 = b_1b_0$ signifie : $a_1 = b_1$ et : $a_0 = b_0$, d'où $E = (a_1 \oplus b_1)(a_0 \oplus b_0)$

$a_1a_0 < b_1b_0$ signifie : $a_1 < b_1$ ou $(a_1 = b_1$ et $a_0 < b_0)$, d'où $I = \overline{a_1}b_1 + (a_1 \oplus b_1)\overline{a_0}b_0$.

De même $S = a_1\overline{b_1} + (a_1 \oplus b_1)a_0\overline{b_0}$

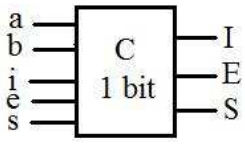
d) L'objectif est d'obtenir le circuit du comparateur deux bits du *c)* en utilisant deux comparateurs 1 bit en succession. Pour cela il convient d'aménager le comparateur 1 bit en lui ajoutant trois entrées *e, i, s* qui proviennent du comparateur en amont. On utilise le schéma suivant :



Qu'obtient-on en sortie ?

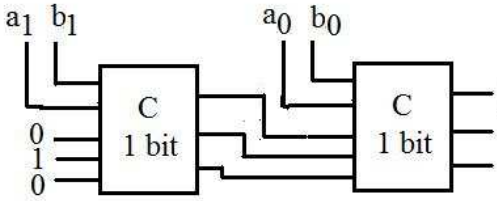
On trouve :

$$I = e\bar{a}b + i$$
$$E = \overline{a \oplus b}\, e$$
$$S = ea\bar{b} + s$$



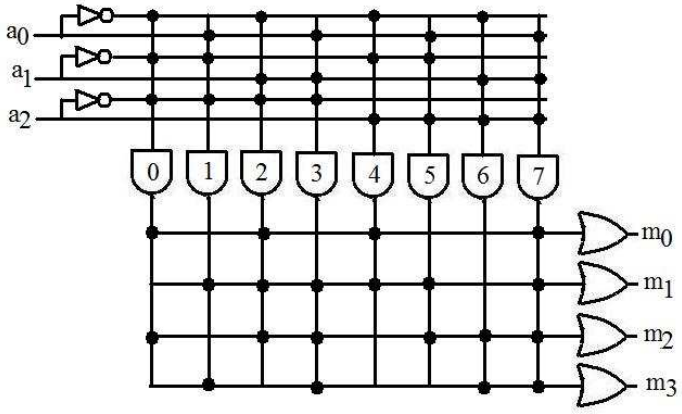
e) Construire un comparateur 2 bits grâce à deux comparateurs 1 bit. Indiquer les jonctions, et préciser les valeurs à donner à *i*, *e*, *s* dans le comparateur initial.

Pour retrouver les équations du *c)*, il suffit de joindre les deux comparateurs 1 bit comme indiqué, en mettant au départ $i = 0$, $e = 1$ et $s = 0$. En généralisant on passerait de la même façon à un comparateur *n* bits.



5) PROM

On considère le schéma suivant, avec trois entrées et quatre sorties. Il est dessiné sous forme simplifiée : sur chaque porte ET arrive en fait 3 fils, et sur chaque porte OU il en arrive 4 pour la sortie m_0, 6 pour la sortie m_1, etc.



a) Lorsque l'on met en entrée un nombre $a_2a_1a_0$ correspondant à un nombre *i* en base 10, quelles sont les sorties des portes ET numérotées de 0 à 7 ?

b) Ce circuit représente une ROM (*Read Only Memory*) déjà programmée. En effet le nombre binaire $a_2a_1a_0$ correspond à une adresse (de 0 à 7), et la sortie associée est un nombre $m_3m_2m_1m_0$ (compris entre 0 et 15), que l'on considère comme le contenu –indélébile- de la case mémoire d'adresse $a_2a_1a_0$. A chaque adresse, que l'on peut écrire en base 10, se trouve un nombre que l'on peut aussi écrire en base 10. Donner le tableau des résultats obtenus.

Remarque : Rappelons qu'une mémoire ROM, dite *morte*, conserve ses données que l'ordinateur soit éteint ou allumé, on dit qu'elle est *non volatile*. Une mémoire ROM au sens strict est conçue et fabriquée en usine, pour s'intégrer ensuite par milliers ou millions

d'exemplaires dans les ordinateurs. Il existe aussi des PROM qui, elles, sont programmables par l'utilisateur en fonction de ses besoins, mais une fois et une seule de façon définitive. Une PROM vierge est formée d'un quadrillage de fils entrecroisés à angle droit, reliés en chaque croisement par un fusible qui permet au courant de passer d'un fil dans une direction vers l'autre. Pour mettre en place les données associées à son programme, le possesseur de la PROM utilise ou fait utiliser un appareil programmateur chargé de griller les fusibles en certains croisements, les deux fils devenant alors indépendants. Ce phénomène est irréversible. C'est ainsi que l'on programme une PROM, pour arriver par exemple au schéma du circuit précédent.

6) Circuit *hidden bit*

On considère un circuit logique à k entrées numérotées : a_1, a_2, …, a_k, chacune correspondant à un bit 0 ou 1, et une sortie B, ainsi définie :

Pour chaque combinaison des entrées, on calcule la somme s des entrées (en base 10), c'est-à-dire le nombre d'entrées égales à 1, puis on fait

- $B = 0$ si $s = 0$
- $B = a_s$ si $1 \leq s \leq k$, autrement dit pour toute valeur non nulle de la somme s, on met en sortie B la valeur de l'entrée qui porte ce numéro s.

a) Traiter le cas où $k = 1$ (une seule entrée a_1), en construisant la table de vérité pour en déduire l'équation donnant B.

b) Traiter le cas où $k = 2$. Simplifier l'équation obtenue à partir de la table de vérité. Constater que l'on obtient pour B le même résultat particulièrement simple que pour $k = 1$.

c) Cas où $k = 3$. Construire la table de vérité, avec en colonnes a_1, a_2, a_3, s et B. En déduire l'équation donnant B par rapport aux entrées. Puis utiliser un tableau de Karnaugh pour simplifier cette équation. Dessiner le circuit correspondant.

d) Traiter de la même façon le cas où $k = 4$, pour aboutir au dessin du circuit après avoir simplifié l'équation.

a)

$a1$	s	B
0	0	0
1	1	1

d'où $B = a_1$

b)

$a1$	$a2$	s	B
0	0	0	0
0	1	1	0
1	0	1	1
1	1	2	1

On constate ici aussi que $B = a_1$.

c)

$a1$	$a2$	$a3$	s	B
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	1
1	0	1	2	0

1	1	0	2	1
1	1	1	3	1

$$B = \overline{a_1}\, a_2\, a_3 + a_1\, \overline{a_2}\, \overline{a_3} + a_1\, a_2\, \overline{a_3} + a_1\, a_2\, a_3$$

Simplifions grâce au tableau de Karnaugh :

a_1a_2 / a_3	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$B = a_1\, \overline{a_3} + a_2\, a_3$$

d)

$a1$	$a2$	$a3$	$a4$	s	B
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	2	0
0	1	0	0	1	0
0	1	0	1	2	1
0	1	1	0	2	1
0	1	1	1	3	1
1	0	0	0	1	1
1	0	0	1	2	0
1	0	1	0	2	0
1	0	1	1	3	1
1	1	0	0	2	1
1	1	0	1	3	0
1	1	1	0	3	1
1	1	1	1	4	1

Et maintenant le tableau de Karnaugh correspondant :

a_1a_2 / a_3a_4	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	0	1	1	1
10	0	1	1	0

Finalement $B = a_2\, a_3 + a_1\, \overline{a_3}\, \overline{a_4} + a_1\, a_3\, a_4 + \overline{a_1}\, a_2\, a_4$

D- Circuits séquentiels

1) Compteur modulo 8

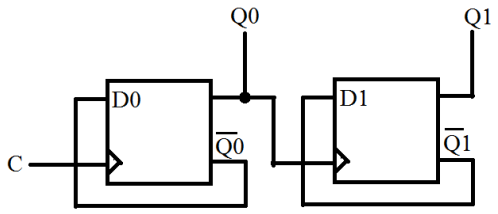
Construire le circuit du compteur modulo 8, lâchant les nombres binaires de 000 à 111 (ou encore de 0 à 7 en base 10) dans l'ordre naturel, et de façon cyclique.

On a vu en cours le compteur modulo 4. Pour passer modulo 8 il suffit de rajouter une troisième bascule dans ce circuit asynchrone.

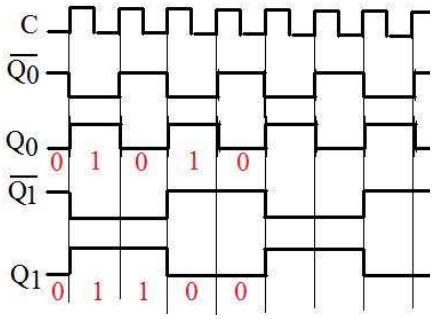
2) Décompteur modulo 4

Construire un décompteur modulo 4 en utilisant des bascules *D* dont le front actif est le front montant de l'horloge. Cela signifie que le nombre binaire Q_1Q_0 associé aux deux sorties Q_0 et Q_1 donne successivement les nombres 0 3 2 1 0 3 2 1 0 … indéfiniment répétés. Faire le chronogramme correspondant.

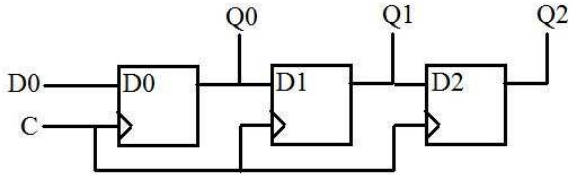
Rappelons que tous les nombres ramenés modulo 4 sont entre 0 et 3. A la différence du compteur vu en cours, il suffit d'envoyer dans l'entrée d'horloge de la deuxième bascule *D* la sortie Q_0 de la première bascule (au lieu de son complément).

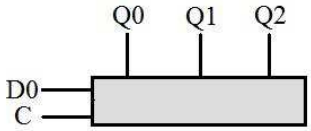


Le chronogramme permet de vérifier la validité du circuit :
On suppose au départ Q_0 et Q_1 égaux à 0.

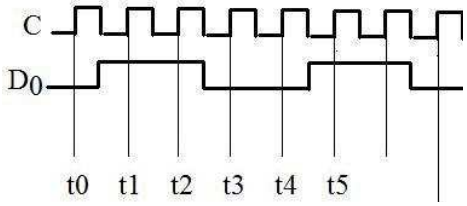


3) Guirlande lumineuse



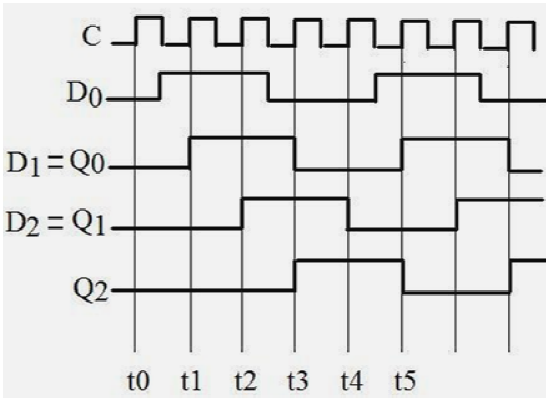


On considère ce circuit, utilisant trois bascules *D* (front montant) en série et en synchrone, avec deux entrées dont l'horloge *C* et avec trois sorties. Dans l'entrée D_0 de la première bascule, on envoie un signal périodique dont la fréquence est la moitié de celle du signal d'horloge et déphasée (décalée) par rapport à celui-ci, comme indiqué dans le chronogrammme suivant :



Terminer le chronogramme en ajoutant ce qui se passe pour les trois sorties Q_0 Q_1 Q_2 (en les supposant toutes à 0 initialement). On suppose maintenant que ces trois sorties sont reliées à trois lampes L_0 L_1 L_2, une lampe étant allumée lorsque la sortie correspondante est à 1, et éteinte pour 0. Indiquer l'évolution de ces lampes dans le temps, en indiquant ce qui se passe juste autour des instants t_0, t_1, t_2, etc. correspondant aux fronts actifs de l'horloge *C*.

Rappelons qu'à chaque front montant de l'horloge, la sortie d'une bascule prend la valeur de l'entrée (celle juste avant le front montant de l'horloge). Le reste du temps, la sortie conserve sa valeur. Cela donne dans le cas présent :



Au départ toutes les lampes sont éteintes. A l'instant t_1, la lampe L_0 s'allume. A l'instant t_2, la lampe L_1 s'allume à son tour. A l'instant t_3 la lampe L_3 s'allume mais L_1 s'éteint. A l'instant t_4, L_2 s'éteint à son tour, et seule L_3 reste allumée. Puis à t_5, L_1 se rallume et L_3 s'éteint, etc.

E- Programmation

1) Nombres en binaire

a) Faire le programme qui à partir d'un nombre entier positif en base 10 donne son écriture en binaire (descendant).

b) Ajouter à ce qui précède (ou faire une fonction) le programme qui permet d'afficher, à partir du bit en position *k*, les *j* bits qui lui succèdent (en comptant le bit numéro *k*), *k* et *j* étant donnés.[2] Rappelons que la position d'un bit commence à 0 en allant de droite à gauche. Par exemple :

nombre en binaire descendant : 1 1 0 1 0 0 1 1 1
positions des bits : 8 7 6 5 4 3 2 1 0

Avec $k = 3$ et $j = 4$, on doit obtenir 0 1 0 0 toujours en binaire descendant.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int nombre,q,r[20],longueur,i,p, k,j;
   nombre = 5389;    printf("Nombre en base 10 : %d ",nombre);
 /** Nombre en binaire descendant */
  q = nombre; i=0;
```

[2] On trouvera dans le cours un programme plus élaboré que celui présenté ici.

```
  while (q>0)   { r[i]= q%2; q = q/2;   i++; }
  longueur=i;  /** longueur du nombre en binaire */
  printf("\nNombre en binaire descendant : ");
  for(p=longueur-1;p>=0; p--) printf("%d ",r[p]);      printf("\n\n");
   /** Bits à partir de la position k sur une longueur j, en binaire descendant */
   /** Cela ne marche que si  j + k <= longueur */
   k = 3; j=4;    /** un exemple */
   printf("\nPartie de longueur j a partir du bit numero k, en binaire descendant : \n\n");
   for(i = j+k-1; i>=k; i--)    printf("%d ",r[i]);
   getchar();return 0;
}
```

Un résultat de ce programme :

```
Nombre en base 10 : 5689
Nombre en binaire descendant : 1 0 1 1 0 0 0 1 1 1 0 0 1
Partie de longueur 5 a partir du bit numero 3, en binaire descendant :
     0 0 1 1 1
```

2) Les erreurs de l'ordinateur

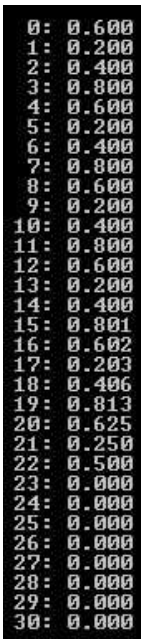
Il y a des années, une publicité disait, à propos d'un ordinateur d'une certaine marque : « *Il est monstrueusement inhumain, car il ne fait jamais d'erreurs* ». Cette vision est pour le moins idéaliste. Comment un être humain, qui fait toujours des erreurs, pourrait-il construire une machine qui ne fait pas d'erreurs ? Mais il est vrai que si l'on travaille avec des nombres entiers, et dans une zone finie, bien délimitée, l'ordinateur ne fait aucune erreur. Et quand on travaille avec des nombres flottants, les légères erreurs (de flottement) qui se produisent loin derrière la virgule sont en général imperceptibles, heureusement. Mais dans certains cas, il peut se produire une accumulation d'erreurs, comme dans l'exercice suivant :

Partir du nombre $u_0 = 0{,}6$. Puis pratiquer la relation de récurrence $u_{n+1} = 2\ u_n$ ramené modulo 1. Par « modulo 1 » on veut dire que si u_n dépasse 1, on enlève 1 de façon que u_n soit toujours compris entre 0 et 1, plus précisément u_n appartient toujours à l'intervalle [0 1[. On obtient l'évolution suivante : 0,6 → 0,2 (c'est 1,2 – 1) → 0,4→ 0,8→ 0,6→ 0,2→ 0,4→ 0,8→ 0,6→ … avec la répétition de la même séquence indéfiniment. Faire maintenant le programme sur machine, et constater les dégâts. Pourquoi en est-il ainsi ?

Programme :

```
#include <stdio.h>
#include <stdlib.h>
int main()
{  int i; float u;
   u=0.6; printf("\n  0: %3.3f",u);
   for(i=1; i<=30; i++)
     {  u = 2.*u;  if (u>=1.) u = u - 1. ;
        printf("\n%3.d: %3.3f",i,u);
     }
   getchar();return 0;
}
```

Résultats :



Ainsi tout devient faux très rapidement. Pour le comprendre, prenons le nombre 0,6. Il s'écrit en binaire en virgule fixe : 0,1001 1001 1001… Passons en flottant. Pour cela on décale la virgule d'un cran vers la droite ce qui donne 1,001 1001 …et l'on multiplie par 2^{-1}. Le nombre s'écrit alors en flottant (simple précision sur 32 bits) :

0 01111110 00110011001100110011001, la partie derrière la virgule se réduisant à 23 bits. Le nombre se trouve ainsi tronqué. Le fait de multiplier par deux décale la virgule d'un cran. Il est normal qu'après 23 itérations, on arrive à la fin du nombre tronqué. Que se passe-t-il au-delà ? Il y a uniquement des 0, comme on peut le constater dans les résultats du programme. Et cela n'a plus rien à voir avec le problème posé.[3]

Remarque : Par contre, si l'on fait le même programme sur une calculette, il ne se produit pas d'erreurs. Pourquoi ? Parce qu'une calculette n'est pas programmée de la même façon qu' un ordinateur. La conversion en binaire se fait chiffre par chiffre sur quatre bits. Ainsi avec 6 qui s'écrit 0110 sur 4 bits, le nombre 0,6 devient 0, 0110 et les calculs sont ensuite faits à partir de ce nombre qui lui est exact.

[3] Je donne un autre exemple, bien plus intéressant, sur mon site, dans *travaux exploratoires, le paradoxe du pentagone des milieux.*