

Architecture des ordinateurs

première partie des annales

Arnaud Giersch, Benoît Meister et Frédéric Vivien

1 TD 1 : Arithmétique des ordinateurs et codage

1. Donner la valeur décimale des entiers suivants, la base dans laquelle ces entiers sont codés étant précisée.

(a) 1011011 et 101010 en binaire (base 2) ;

Correction : $1011011_2 = 91_{10}$, $101010_2 = 42_{10}$.

(b) A1BE et C4F3 en hexadécimal (base 16) ;

Correction : $A1BE_{16} = 41\,406_{10}$, $C4F3_{16} = 50\,419_{10}$.

(c) 77210 et 31337 en octal (base 8).

Correction : $77210_8 = 32\,392_{10}$, $31337_8 = 13\,023_{10}$.

2. Coder l'entier 2 397 successivement en base 2, 8 et 16.

Correction : $2\,397_{10} = 100101011101_2 = 4535_8 = 95D_{16}$.

3. Donner la valeur décimale du nombre 10101, dans le cas où il est codé en base 2, 8 ou 16.

Correction : $10101_2 = 21_{10}$, $10101_8 = 4\,161_{10}$, $10101_{16} = 65\,793_{10}$.

Même question avec le nombre 6535 codé en base 8 ou 16.

Correction : $6535_8 = 3\,421_{10}$, $6535_{16} = 25\,909_{10}$.

4. Combien d'entiers positifs peut-on coder en binaire sur un octet ?

Correction : Un octet contient 8 bits, on peut donc coder $2^8 = 256$ entiers.

Combien de bits faut-il pour représenter 65 563 entiers différents en binaire ?

Correction : Avec b bits, on peut coder 2^b entiers différents. Pour coder n entiers, il nous faut donc m bits tels que $2^{m-1} < n \leq 2^m$, c.-à-d. $m-1 < \log_2 n \leq m$. On a donc $m = \lceil \log_2 n \rceil$.

Pour $n = 65\,563$, on a $m = \lceil \log_2 65\,563 \rceil = 17$.

5. Soit un ordinateur dont les mots mémoire sont composés de 32 bits. Cet ordinateur dispose de 4 Mo de mémoire. Un entier étant codé sur un mot, combien de mots cet ordinateur peut-il mémoriser simultanément ?

Correction : $4\text{ Mo} = 4 \times 2^{20}$ octets, un mot est composé de 4 octets. Cet ordinateur peut donc mémoriser $\frac{4 \times 2^{20}}{4} = 2^{20} = 1\,048\,576$ mots

Quelle est la plus grande valeur entière (décimale) que cet ordinateur peut mémoriser, cette valeur étant représentée par son codage binaire pur ? Donner un ordre de grandeur du nombre de chiffres en codage décimal.

Correction : La mémoire contient 4×2^{20} octets, c.-à-d. $4 \times 2^{20} \times 8 = 33\,554\,432$ bits. La plus grande valeur entière que cet ordinateur peut mémoriser est donc $2^{33\,554\,432} - 1$.

Le nombre de chiffres en décimal est de $\left\lceil \log_{10} \left(2^{32 \times 2^{20}} - 1 \right) \right\rceil \simeq 2^{20} \log_{10} 2^{32} \simeq 10^6 \times 3,2 \times \log_{10} 2^{10} \simeq 10^7$. Le nombre exact de chiffres en décimal est 10 100 891.

6. Coder en binaire sur un octet les entiers 105 et 21 puis effectuer l'addition binaire des entiers ainsi codés. Vérifier que le résultat sur un octet est correct. Même question avec les entiers 184 et 72.

Correction :

$$\begin{array}{r} 1101001 \quad (105) \\ + \quad 10101 \quad (21) \\ \hline 1111110 \quad (126) \end{array}$$

Ce résultat est correct.

$$\begin{array}{r} 10111000 \quad (184) \\ + \quad 1001000 \quad (72) \\ \hline (1)00000000 \quad (0) \end{array}$$

Ce résultat n'est pas correct (sur 8 bits).

7. Coder en binaire sur un octet les entiers 79 et 52 puis effectuer la multiplication binaire des entiers ainsi codés. Même question avec les entiers 135 et 46.

Correction :

$$\begin{array}{r} \times \quad 1001111 \quad (79) \\ \quad 110100 \quad (52) \\ \hline 1001111 \\ 1001111 \\ + \quad 1001111 \\ \hline 1000000001100 \quad (4108) \end{array}$$

$$\begin{array}{r} \times \quad 10000111 \quad (135) \\ \quad 101110 \quad (46) \\ \hline 10000111 \\ 10000111 \\ 10000111 \\ + \quad 10000111 \\ \hline 1100001000010 \quad (6\,210) \end{array}$$

8. Indiquer la valeur codée par le mot de 16 bits 1101100101110101 suivant qu'il représente un entier non signé, ou un entier signé.

Correction : *En non signé, la valeur est $1101100101110101_2 = 55\,669_{10}$. En signé, le premier bit (bit de signe) vaut 1, c'est donc un nombre négatif dont la valeur est $-101100101110101_2 = -22\,901_{10}$.*

Même question avec le mot 1001000011101101.

Correction : *En non signé, la valeur est $1001000011101101_2 = 37\,101_{10}$. En signé, c'est un nombre négatif dont la valeur est $-1000011101101_2 = -4\,333_{10}$.*

2 TD 2 : Arithmétique des ordinateurs (suite)

1. Indiquer la valeur codée par la suite 1101100101110101 qui représente un entier signé en complément à 2 sur 16 bits.

Correction : *C'est un nombre négatif. Complément à 2 : 0010011010001011 donc -9867 .*

Même question avec la suite 1001000011101101.

Correction : *C'est un nombre négatif. Complément à 2 : 0110111100010011 donc -28435 .*

2. Représentation binaire des entiers négatifs

- (a) Coder sur 4 bits les entiers 7, 2, 0, -2 , -7 et -8 avec les représentations suivantes :

– signe et valeur absolue ;

Correction : *0111, 0010, 0000 ou 1000, 1010, 1111, n/a*

– complément à 1 ;

Correction : *0111, 0010, 0000 ou 1111, 1101, 1000, n/a*

– complément à 2.

Correction : *0111, 0010, 0000, 1110, 1001, 1000*

- (b) Coder les entiers 61 et -61 sur un octet en utilisant la représentation par le signe et la valeur absolue. Montrer que l'addition binaire de ces entiers ainsi codés produit un résultat incorrect. Montrer qu'en revanche le résultat est correct si ces entiers sont codés en utilisant la représentation par le complément à 2.

Correction :*Signe et valeur absolue :*

$$\begin{array}{r}
 00111101 \quad (61) \\
 + \quad 10111101 \quad (-61) \\
 \hline
 11111010 \quad (-122)
 \end{array}$$

Complément à deux :

$$\begin{array}{r}
 00111101 \quad (61) \\
 + \quad 11000011 \quad (-61) \\
 \hline
 00000000 \quad (0)
 \end{array}$$

3. Effectuer en binaire (8 bits) les opérations $1 - 2$, $51 + 127$, $-3 - 127$, $-127 + 127$, $-63 - 63$. Préciser, pour chaque opération, la retenue et le débordement.

Correction : *On code les nombres négatifs en complément à 2.**Débordement :*

- L'addition de deux nombres de signes différents ne produit jamais de débordement (la valeur absolue du résultat est toujours inférieure au maximum des valeurs absolues des deux opérandes).
- L'addition de deux nombres de même signe produit un débordement si le signe du résultat est différent du signe des deux opérandes.

$$\begin{array}{r}
 00000001 \quad (1) \\
 + \quad 11111110 \quad (-2) \\
 \hline
 11111111 \quad (-1)
 \end{array}$$

retenue : 0, débordement : 0

$$\begin{array}{r}
 00110011 \quad (51) \\
 + \quad 01111111 \quad (127) \\
 \hline
 10110010 \quad (-78)
 \end{array}$$

retenue : 0, débordement : 1

$$\begin{array}{r}
 11111101 \quad (-3) \\
 + \quad 10000001 \quad (-127) \\
 \hline
 01111110 \quad (126)
 \end{array}$$

retenue : 1, débordement : 1

$$\begin{array}{r}
 10000001 \quad (-127) \\
 + \quad 01111111 \quad (127) \\
 \hline
 00000000 \quad (0)
 \end{array}$$

retenue : 1, débordement : 0

$$\begin{array}{r}
 11000001 \quad (-63) \\
 + \quad 11000001 \quad (-63) \\
 \hline
 10000010 \quad (-126)
 \end{array}$$

retenue : 1, débordement : 0

4. Représentation des réels

- (a) En virgule fixe, décoder le nombre binaire 11.011 puis coder en binaire le réel 11.625.

Correction :

$$11.011_2 = [1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}]_{10} = [2 + 1 + 0.25 + 0.125]_{10} = 3.375_{10}$$

$$11.625_{10} = [8 + 2 + 1 + 0.5 + 0.125]_{10} = [2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3}]_{10} = 1011.101_2$$

- (b) En virgule flottante normalisée, coder en binaire au format simple précision le réel 12.575

Correction :

$$\begin{aligned}
 12.575_{10} &= 1100.1001001 \dots_2 = [0.11001001001 \dots \times 10^{100}]_2 \\
 &\quad 0|10000011|11001001001100110011001|
 \end{aligned}$$

puis effectuer le codage inverse.

Correction : *bit de signe = 0 \rightarrow nombre positif.**exposant biaisé = $10000011_2 = 131_{10} \rightarrow$ exposant : $10000011 - 01111111 = 100_2 = 4_{10}$* *la mantisse est normalisée : 0.11001001001100110011001*

$$\begin{aligned}
 [0.11001001001100110011001 \times 10^{100}]_2 &= 1100.1001001100110011001_2 \\
 &= [2^3 + 2^2 + 2^{-1} + 2^{-4} + 2^{-7} + 2^{-8} + 2^{-11} \\
 &\quad + 2^{-12} + 2^{-15} + 2^{-16} + 2^{-19}]_1 \quad 0 \\
 &= 12.5749988555908203125_{10}
 \end{aligned}$$

5. Opérations en virgule flottante.

Soit $a = [0.10010 \times 10^{101}]_2$ et $b = [0.11010 \times 10^1]_2$. Calculer $a + b$ et $a \times b$.

Correction : Avant de faire l'addition, il faut que les deux exposants soient égaux ($a = 1001 \times 10^1$, $b = 0.1101 \times 10^1$). Pour faire la multiplication, on multiplie les mantisses puis on additionne les exposants. Dans les deux cas, le résultat doit ensuite être normalisé.

$$\begin{array}{r} 1001.0000 \times 10^1 \\ + \quad 0.1101 \times 10^1 \\ \hline 1001.1101 \times 10^1 \\ = 0.10011101 \times 10^{101} \end{array}$$

$$\begin{array}{r} 0.1001 \times 10^{101} \\ \times \quad 0.1101 \times 10^1 \\ \hline 1001 \\ + \quad 1001 \\ \hline 0.01110101 \times 10^{110} \\ = 0.1110101 \times 10^{101} \end{array}$$

3 TD 3 : Algèbre de Boole

1. Montrer comment l'opérateur **et** peut être obtenu à partir des opérateurs **ou** et **non**. De même pour l'opérateur **ou** avec les opérateurs **et** et **non**.

Correction : $\text{non}(a \text{ ou } b) = (\text{non } a) \text{ et } (\text{non } b) \Rightarrow \text{non}((\text{non } a) \text{ ou } (\text{non } b)) = a \text{ et } b$
 $\text{non}(a \text{ et } b) = (\text{non } a) \text{ ou } (\text{non } b) \Rightarrow \text{non}((\text{non } a) \text{ et } (\text{non } b)) = a \text{ ou } b$

2. On note respectivement les opérateurs **ou**, **et**, **xor** et **non** par $+$, \cdot , \oplus et $\bar{}$. Montrer à l'aide de tables de vérité que $A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$ et que $A \oplus B = (A + B) \cdot (\bar{A} + \bar{B})$

Correction : Tables de vérités :

A	B	\bar{A}	\bar{B}	$A \oplus B$	$\bar{A} \cdot B$	$A \cdot \bar{B}$	$\bar{A} \cdot B + A \cdot \bar{B}$
1	1	0	0	0	0	0	0
1	0	0	1	1	0	1	1
0	1	1	0	1	1	0	1
0	0	1	1	0	0	0	0

A	B	\bar{A}	\bar{B}	$A \oplus B$	$A + B$	$\bar{A} + \bar{B}$	$(A + B) \cdot (\bar{A} + \bar{B})$
1	1	0	0	0	1	0	0
1	0	0	1	1	1	1	1
0	1	1	0	1	1	1	1
0	0	1	1	0	0	1	0

3. Montrer que $A + (\bar{A} \cdot B) = A + B$ et que $A \cdot (\bar{A} + B) = A \cdot B$

Correction : On utilise la distributivité de l'opérateur **ou** sur l'opérateur **et**, et inversement :

$$\begin{aligned} A + (\bar{A} \cdot B) &= (A + \bar{A}) \cdot (A + B) = 1 \cdot (A + B) = A + B \\ A \cdot (\bar{A} + B) &= (A \cdot \bar{A}) + (A \cdot B) = 0 + (A \cdot B) = A \cdot B \end{aligned}$$

4. Déterminer le complément de l'expression $A + \bar{B} \cdot C$

Correction : On utilise les lois de de Morgan ; l'opérateur **et** est prioritaire :

$$\overline{A + \bar{B} \cdot C} = \bar{A} \cdot \overline{\bar{B} \cdot C} = \bar{A} \cdot (B + \bar{C}) = \bar{A} \cdot B + \bar{A} \cdot \bar{C}$$

5. Montrer que les deux règles d'associativité sont duales, i.e. montrer qu'à partir de la règle d'associativité de l'opérateur **ou**, on peut déduire, en utilisant les lois de de Morgan, l'associativité de l'opérateur **et** (et inversement).

Correction :

$$A + (B + C) = (A + B) + C \Leftrightarrow \overline{A + (B + C)} = \overline{(A + B) + C} \Leftrightarrow \overline{A} \cdot (\overline{B + C}) = (\overline{A + B}) \cdot \overline{C}$$

A, B , et C sont des variables muettes. Par changement de variable $\{(\overline{A} \rightarrow A'), (\overline{B} \rightarrow B'), (\overline{C} \rightarrow C')\}$ on obtient la propriété d'associativité du **ou** : $A' \cdot (B' \cdot C') = (A' \cdot B') \cdot C'$

6. Écrire l'expression $\overline{A \oplus B}$ uniquement avec les opérateurs **ou**, **et** et **non**

Correction : D'après 2. :

$$A \oplus B = \overline{A} \cdot B + A \cdot \overline{B} \Leftrightarrow \overline{A \oplus B} = \overline{\overline{A} \cdot B + A \cdot \overline{B}} \Leftrightarrow \overline{A \oplus B} = (A + \overline{B}) \cdot (\overline{A} + B)$$

7. Montrer que la fonction **nor** forme un groupe logique complet.

Correction : Pour cela, on montre que la fonction **nor** permet d'exprimer tous les opérateurs logiques :

- **non** : $\text{nor}(A, A) = \overline{A}$
- **et** : $\text{nor}(\text{nor}(A, A), \text{nor}(B, B)) = \text{nor}(\overline{A}, \overline{B}) = \overline{\overline{A} + \overline{B}} = A \cdot B$
- **ou** : $\text{nor}(\text{nor}(A, B), \text{nor}(A, B)) = \overline{\text{nor}(A, B)} = \overline{\overline{A + B}} = (A + B)$.

8. Simplifier au maximum les expressions logiques suivantes.

(a) $\overline{A} \cdot B + A \cdot B$

Correction :

$$\overline{A} \cdot B + A \cdot B = (\overline{A} + A) \cdot B = 1 \cdot B = B$$

(b) $(A + B) \cdot (A + \overline{B})$

Correction :

$$(A + B) \cdot (A + \overline{B}) = A + B \cdot \overline{B} = A + 0 = A$$

(c) $A + A \cdot B$

Correction :

$$A + A \cdot B = A \cdot 1 + A \cdot B = A \cdot (1 + B) = A \cdot 1 = A$$

(d) $A \cdot (A + B)$

Correction :

$$A \cdot (A + B) = (A + 0) \cdot (A + B) = A + 0 \cdot B = A + 0 = A$$

(e) $\overline{A} \cdot \overline{B} + \overline{A + B + C + D}$

Correction :

$$\begin{aligned} \overline{A} \cdot \overline{B} + \overline{A + B + C + D} &= \overline{(A + B) \cdot (A + B + C + D)} \\ &= \overline{(A + B) \cdot ((A + B) + (C + D))} \end{aligned}$$

donc, d'après l'exercice 8d,

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

(f) $A + B \cdot \overline{C} + \overline{A} \cdot (\overline{B \cdot \overline{C}}) \cdot (A \cdot D + B)$

Correction :

$$A + B \cdot \overline{C} + \overline{A} \cdot (\overline{B \cdot \overline{C}}) \cdot (A \cdot D + B) = (A + B \cdot \overline{C}) + (\overline{A + B \cdot \overline{C}}) \cdot (A \cdot D + B)$$

d'après l'exercice 3,

$$A + B \cdot \overline{C} + \overline{A} \cdot (\overline{B \cdot \overline{C}}) \cdot (A \cdot D + B) = (A + B \cdot \overline{C}) + (A \cdot D + B) = (A + A \cdot D) + (B + B \cdot \overline{C})$$

d'après l'exercice 8c,

$$A + B \cdot \overline{C} + \overline{A} \cdot (\overline{B \cdot \overline{C}}) \cdot (A \cdot D + B) = A + B$$

(g) $(A \oplus B) \cdot B + A \cdot B$

Correction :

d'après l'exercice 2,

$$\begin{aligned}(A \oplus B) \cdot B + A \cdot B &= (\bar{A} \cdot B + A \cdot \bar{B}) \cdot B + A \cdot B \\ &= \bar{A} \cdot B + A \cdot \bar{B} \cdot B + A \cdot B \\ &= \bar{A} \cdot B + A \cdot B\end{aligned}$$

d'après l'exercice 8a,

$$= B$$

(h) $A + \bar{A} \cdot B + \bar{A} \cdot \bar{B}$

Correction :

$$A + \bar{A} \cdot B + \bar{A} \cdot \bar{B} = (A + \bar{A} \cdot B) + \bar{A} \cdot \bar{B}$$

d'après l'exercice 3,

$$A + \bar{A} \cdot B + \bar{A} \cdot \bar{B} = (A + B) + (\bar{A} + \bar{B}) = 1$$

9. Démontrer que toute fonction à trois variables $F(A, B, C)$ est égale à

$$F(A, B, C) = A \cdot F(1, B, C) + \bar{A} \cdot F(0, B, C)$$

Correction : A est une variable booléenne : les deux valeurs qu'elle peut prendre sont 0 et 1 :

– si $A = 0$, $0 \cdot F(1, B, C) + 1 \cdot F(0, B, C) = F(0, B, C) = F(A, B, C)$;

– si $A = 1$, $1 \cdot F(1, B, C) + 0 \cdot F(0, B, C) = F(1, B, C) = F(A, B, C)$.

10. Montrer que les lois de de Morgan s'étendent à un nombre quelconque de variables.

Correction :

(a) $\overline{A_1 \cdot A_2 \cdot \dots \cdot A_n} = \bar{A}_1 + \bar{A}_2 + \dots + \bar{A}_n$ avec $n \geq 2$. La démonstration se fait par récurrence sur n (le nombre de variables).

$n = 2$ c'est la loi de de Morgan « basique » ;

$n > 2$ on utilise l'associativité de $+$ et \cdot :

$$\begin{aligned}\overline{A_1 \cdot A_2 \cdot \dots \cdot A_n} &= \overline{(A_1 \cdot A_2 \cdot \dots \cdot A_{n-1}) \cdot A_n} \\ &= \overline{(A_1 \cdot A_2 \cdot \dots \cdot A_{n-1})} + \bar{A}_n \\ &= (\bar{A}_1 + \bar{A}_2 + \dots + \bar{A}_{n-1}) + \bar{A}_n \\ &= \bar{A}_1 + \bar{A}_2 + \dots + \bar{A}_{n-1} + \bar{A}_n\end{aligned}$$

(b) $\overline{\bar{A}_1 + \bar{A}_2 + \dots + \bar{A}_n} = \bar{A}_1 \cdot \bar{A}_2 \cdot \dots \cdot \bar{A}_n$ avec $n \geq 2$. Le raisonnement est similaire.

11. Génération et simplification d'expressions logiques

Considérer la fonction définie par la table de vérité ci-dessous :

A	B	C	$F(A, B, C)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(a) Générer une expression logique correspondante :

i. sous forme de sommes de produits ;

Correction :

$$\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

ii. sous forme de produits de sommes.

Correction :

$$\overline{\overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C}} = (A + B + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C})$$

(b) Simplifier les deux expressions en utilisant les règles de l'algèbre de Boole.

Correction :

i.

$$\begin{aligned} & \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} \\ &= \overline{A} \cdot \overline{B} \cdot C + (A + \overline{A}) \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot (C + \overline{C}) \\ &= \overline{A} \cdot \overline{B} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \\ &= (A + \overline{A} \cdot C) \cdot \overline{B} + B \cdot \overline{C} \\ &= (A + C) \cdot \overline{B} + B \cdot \overline{C} \\ &= A \cdot \overline{B} + B \cdot \overline{C} + \overline{B} \cdot C \\ &= A \cdot \overline{B} + (B \oplus C) \end{aligned}$$

ii.

$$\begin{aligned} & (A + B + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \\ &= (A \cdot A + A \cdot \overline{B} + A \cdot \overline{C} + B \cdot A + B \cdot \overline{B} + B \cdot \overline{C} + C \cdot A + C \cdot \overline{B} + C \cdot \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \\ &= (A + A \cdot B + A \cdot C + A \cdot \overline{B} + A \cdot \overline{C} + B \cdot \overline{C} + \overline{B} \cdot C) \cdot (\overline{A} + \overline{B} + \overline{C}) \\ &= A \cdot \overline{A} + A \cdot B \cdot \overline{A} + A \cdot C \cdot \overline{A} + A \cdot \overline{B} \cdot \overline{A} + A \cdot \overline{C} \cdot \overline{A} + B \cdot \overline{C} \cdot \overline{A} + \overline{B} \cdot C \cdot \overline{A} + \\ & \quad A \cdot \overline{B} + A \cdot B \cdot \overline{B} + A \cdot C \cdot \overline{B} + A \cdot \overline{B} \cdot \overline{B} + A \cdot \overline{C} \cdot \overline{B} + B \cdot \overline{C} \cdot \overline{B} + \overline{B} \cdot C \cdot \overline{B} + \\ & \quad A \cdot \overline{C} + A \cdot B \cdot \overline{C} + A \cdot C \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{C} \cdot \overline{C} + B \cdot \overline{C} \cdot \overline{C} + \overline{B} \cdot C \cdot \overline{C} \\ &= A \cdot B \cdot \overline{C} + A \cdot \overline{B} + A \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{C} + \overline{B} \cdot C + B \cdot \overline{C} \\ &= (A \cdot \overline{B}) \cdot (1 + C + \overline{C}) + \overline{B} \cdot C + (A + 1) \cdot (B \cdot \overline{C}) \\ &= A \cdot \overline{B} + B \cdot \overline{C} + \overline{B} \cdot C \\ &= A \cdot \overline{B} + (B \oplus C) \end{aligned}$$

(c) Construire le diagramme de Karnaugh et déterminer une expression logique associée.

Correction : Une table de Karnaugh se construit à partir de l'expression logique sous forme de somme de produits. Dans la somme de produits utilisée, chaque produit doit contenir toutes les variables de l'expression. Par exemple, on mettra une expression dépendant de A et B sous la forme d'une somme de produits de A, \overline{A} , B, \overline{B} . Pour mettre l'expression sous la forme voulue, la formule $(A + \overline{A})B = B$ est très utile.

$$\overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

Chaque colonne de la table de Karnaugh doit différer de ses voisines d'un et un seul littéral. Nous avons 3 variables et les tables de Karnaugh sont à 2 dimensions : il faut regrouper deux variables. Ici nous choisissons de regrouper B et C. On regroupe les 1 en morceaux rectangulaires, selon les principes suivants :
– faire les plus grands morceaux possibles,

- faire le moins de morceaux possibles,
- le nombre de 1 dans un morceau doit être une puissance de 2,
- ne faire un nouveau morceau que s'il permet de regrouper des 1 qui n'ont pas encore été regroupés, en se rappelant que la ligne du bas et la ligne du haut sont considérées comme adjacentes, et qu'il en est de même pour la colonne la plus à droite et la colonne la plus à gauche.

$A \backslash BC$	BC	$\overline{B}C$	$\overline{B}\overline{C}$	$B\overline{C}$
A	0	1	1	1
\overline{A}	0	1	0	1

Chaque morceau donne naissance à un produit de variables. Lorsqu'une variable et son inverse sont dans le même morceau, cette variable s'élimine (parce que $(A + \overline{A}) = 1$).

$$\overline{B} \cdot C + A \cdot \overline{B} + B \cdot \overline{C}$$

12. Considérer les fonctions logiques suivantes. Pour chacune d'elles,
- construire le diagramme de Karnaugh ;
 - utiliser le diagramme pour simplifier les expressions.

(a) $F_1(A, B, C) = A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$

Correction : La table de Karnaugh est présentée figure 1.

$A \backslash BC$	BC	$\overline{B}C$	$\overline{B}\overline{C}$	$B\overline{C}$
A	1	1	0	1
\overline{A}	0	0	0	0

$A \backslash BC$	BC	$\overline{B}C$	$\overline{B}\overline{C}$	$B\overline{C}$
A	1	1	1	0
\overline{A}	0	0	1	0

FIG. 1 – Table de Karnaugh pour $F_1(A, B, C)$.

FIG. 2 – Table de Karnaugh pour $F_2(A, B, C)$.

Expression simplifiée : $F_1(A, B, C) = A \cdot B + A \cdot C$.

(b) $F_2(A, B, C) = \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} + A \cdot B \cdot C$

Correction : La table de Karnaugh est présentée figure 2.

Expression simplifiée : $F_2(A, B, C) = A \cdot C + \overline{B} \cdot \overline{C}$

(c) $F_3(A, B, C) = \overline{A} \cdot \overline{B} + \overline{A} \cdot B \cdot \overline{C} + \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C$

Correction :

$$\begin{aligned} F_3(A, B, C) &= \overline{A} \cdot \overline{B} + \overline{A} \cdot B \cdot \overline{C} + \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C \\ &= \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C \end{aligned}$$

La table de Karnaugh est présentée figure 3.

Expression simplifiée : $F_3(A, B, C) = \overline{B} + \overline{A} \cdot \overline{C}$

(d) $F_4(A, B, C, D) = B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{D}$

$A \backslash BC$	BC	$\overline{B}C$	$B\overline{C}$	$B\overline{C}$
A	0	1	1	0
\overline{A}	0	1	1	1

FIG. 3 – Table de Karnaugh pour $F_3(A,B,C)$.

$AB \backslash CD$	CD	$\overline{C}D$	$\overline{C}\overline{D}$	$C\overline{D}$
AB	0	0	1	1
$\overline{A}B$	0	0	1	1
$\overline{A}\overline{B}$	0	0	0	0
$A\overline{B}$	0	0	0	0

FIG. 4 – Table de Karnaugh pour $F_4(A,B,C,D)$.

Correction :

$$\begin{aligned}
 F_4(A,B,C,D) &= B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{D} \\
 &= A \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{D}
 \end{aligned}$$

La table de Karnaugh est présentée figure 4.

Expression simplifiée : $F_4(A,B,C,D) = B \cdot \overline{D}$

(e) $F_5(A,B,C,D) = \overline{A} + A \cdot B + A \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C \cdot D$

Correction :

$$\begin{aligned}
 F_5(A,B,C,D) &= \overline{A} \cdot B \cdot C \cdot D + \overline{A} \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot D \\
 &\quad + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot \overline{D} \\
 &\quad + A \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot D + A \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot D
 \end{aligned}$$

La table de Karnaugh est présentée figure 5.

Expression simplifiée : $F_5(A,B,C,D) = B + \overline{A} + C$

$AB \backslash CD$	CD	$\overline{C}D$	$\overline{C}\overline{D}$	$C\overline{D}$
AB	1	1	1	1
$\overline{A}B$	1	1	1	1
$\overline{A}\overline{B}$	1	1	1	1
$A\overline{B}$	1	0	0	1

FIG. 5 – Table de Karnaugh pour $F_5(A,B,C,D)$.

$AB \backslash CD$	CD	$\overline{C}D$	$\overline{C}\overline{D}$	$C\overline{D}$
AB	1	1	0	0
$\overline{A}B$	0	0	1	1
$\overline{A}\overline{B}$	0	0	1	1
$A\overline{B}$	0	0	1	1

FIG. 6 – Table de Karnaugh pour $F_6(A,B,C,D)$.

(f) $F_6(A,B,C,D) = \overline{A} \cdot \overline{B} \cdot \overline{D} + \overline{A} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot D + \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D}$

Correction :

$$\begin{aligned}
 F_6(A,B,C,D) &= \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \\
 &\quad + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \\
 &\quad + A \cdot \overline{B} \cdot C \cdot \overline{D}
 \end{aligned}$$

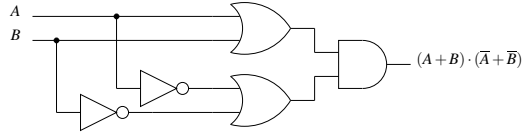
La table de Karnaugh est présentée figure 6.

Expression simplifiée : $F_6(A,B,C,D) = \bar{A} \cdot \bar{D} + A \cdot B \cdot D + \bar{B} \cdot \bar{D}$

4 TD 4 : Circuits combinatoires

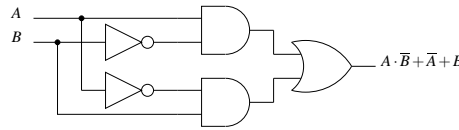
1. Exprimer la fonction **xor** comme un produit de sommes et réaliser le circuit logique correspondant.

Correction : $A \oplus B = (A + B) \cdot (\bar{A} + \bar{B})$



Même question en exprimant **xor** comme une somme de produits.

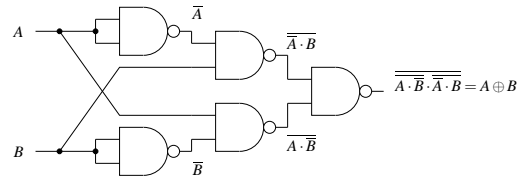
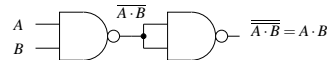
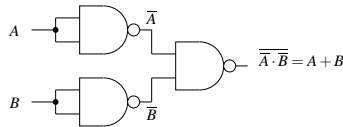
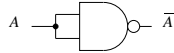
Correction : $A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B$



2. La fonction **nand** formant un groupe logique complet, réaliser, uniquement avec des portes **nand**, les circuits logiques **not**, **and**, **or** et **xor** (les formules sont rappelées ci-dessous).

- $\text{not}(A) = \text{nand}(A, A)$
- $\text{and}(A, B) = \text{nand}(\text{nand}(A, B), \text{nand}(A, B))$
- $\text{or}(A, B) = \text{nand}(\text{nand}(A, A), \text{nand}(B, B))$
- $\text{xor}(A, B) = \text{nand}(\text{nand}(\text{nand}(A, A), B), \text{nand}(A, \text{nand}(B, B)))$

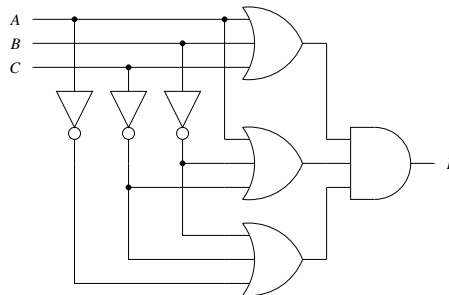
Correction :



3. Réaliser un circuit logique qui implémente la fonction F .

$$F = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C})$$

Correction :



4. Un *générateur de parité impaire* est une fonction qui retourne 1 si le nombre de bits à 1 est impair et 0 sinon. Définir cette fonction pour un mot de 4 bits. Donner un circuit logique implémentant cette fonction.

Correction : La formule pour le générateur de parité impaire sur 4 bits (P) obtenue directement à partir de la table de vérité est :

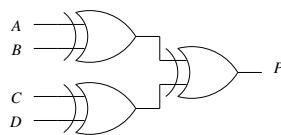
$$P = A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot C \cdot \bar{D} + \bar{A} \cdot \bar{B} \cdot C \cdot D + \dots + \bar{A} \cdot B \cdot C \cdot D + A \cdot \bar{B} \cdot C \cdot D + A \cdot B \cdot \bar{C} \cdot D + A \cdot B \cdot C \cdot \bar{D}$$

ce qui donnerait un circuit beaucoup trop compliqué !

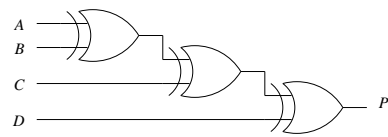
On remarque que pour deux bits, $P = A \oplus B$:

A	B	P
0	0	0
0	1	1
1	0	1
1	1	0

On en déduit les circuits suivants :

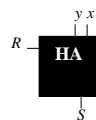


ou



5. Rappeler les principes d'un demi-additionneur puis d'un additionneur complet. Dédurre de ces principes un circuit logique qui implémente le complément à 2 sur n bits.

Correction : Le demi-additionneur possède deux entrées (x et y) et deux sorties (R et S). S correspond au bit de rang zéro du résultat de l'addition binaire de x et y , R au bit de rang 1 (retenue).



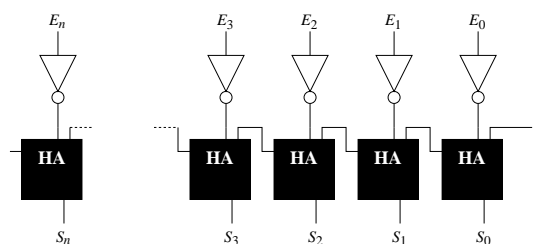
x	y	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x \oplus y$$

$$R = x \cdot y$$

Un additionneur complet s'obtient en enchaînant des demi-additionneurs de manière à propager correctement la retenue.

On obtient selon le même principe le circuit effectuant un complément à deux :



6. Réaliser un circuit pour un décrémenteur à n bits.

Correction : On pourrait imaginer utiliser un soustracteur avec le deuxième opérande égal à 1 pour réaliser un décrémenteur.

Or la décrémentation (comme l'incrémement) est une opération fréquente sur certains registres (compteur ordinal, registre d'index, registre d'adresse, ...). Donc pour gagner du temps on souhaite construire un circuit spécialisé.

Considérons $A_{n-1} \dots A_1 A_0$ le nombre binaire à décrémenter. Appelons $S_{n-1} \dots S_1 S_0$ le résultat de la décrémentation.

On sait que S est obtenu par soustraction de 1 et propagation de la retenue (R_i). On a donc :

$$S_0 = A_0 - 1 :$$

A_0	1	S_0	R_0
0	1	1	1
1	1	0	0

$$S_0 = \overline{A_0}$$

$$R_0 = \overline{A_0}$$

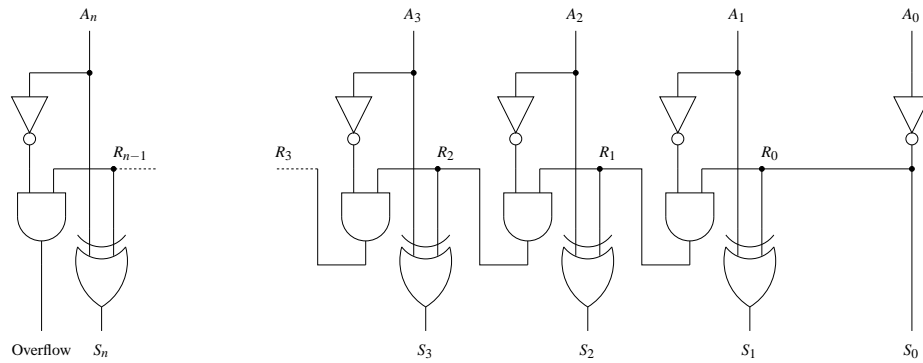
$$\text{et } S_i = A_i - R_{i-1} :$$

A_i	R_{i-1}	S_i	R_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$S_i = A_i \oplus R_{i-1}$$

$$R_i = \overline{A_i} \cdot R_{i-1}$$

D'où le circuit :



7. Le soustracteur

(a) Réaliser un demi-soustracteur (table de vérité et circuit).

Correction : Le demi-soustracteur est défini par la table de vérité suivante (le bit B_i est retranché au bit A_i) :

A_i	B_i	D_i	R_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

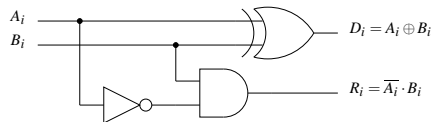
D_i contient la différence $A_i - B_i$
 R_i contient la retenue éventuelle

On a donc :

$$D_i = A_i \oplus B_i$$

$$R_i = \overline{A_i} \cdot B_i$$

D'où le circuit logique :



(b) Réaliser un soustracteur binaire complet (ou étage de soustracteur) selon deux modes :

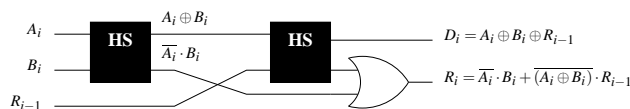
i. avec deux demi-soustracteurs ;

Correction : Pour obtenir un soustracteur binaire complet il faut prendre en compte l'éventuelle retenue précédente R_{i-1} . La table de vérité est :

R_{i-1}	A_i	B_i	D_i	R_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

$$\begin{aligned}
 D_i &= \overline{R_{i-1}} \cdot (A_i \oplus B_i) + R_{i-1} \cdot \overline{(A_i \oplus B_i)} \\
 &= (A_i \oplus B_i) \oplus R_{i-1} \\
 R_i &= \overline{R_{i-1}} \cdot \overline{A_i} \cdot B_i + R_{i-1} \cdot \overline{A_i} \cdot \overline{B_i} \\
 &\quad + R_{i-1} \cdot \overline{A_i} \cdot B_i + R_{i-1} \cdot A_i \cdot B_i \\
 &= \overline{A_i} \cdot B_i \cdot (\overline{R_{i-1}} + R_{i-1}) \\
 &\quad + (\overline{A_i} \cdot \overline{B_i} + A_i \cdot B_i) \cdot R_{i-1} \\
 &= \overline{A_i} \cdot B_i + (\overline{A_i} \oplus B_i) \cdot R_{i-1}
 \end{aligned}$$

d'où le circuit :



Ce schéma correspond au fait que le soustracteur est réalisé en

- (1) retranchant B_i de A_i (1^{er} demi-soustracteur) ;
- (2) puis en retranchant R_{i-1} de la différence obtenue.

ii. avec un demi-additionneur et un demi-soustracteur.

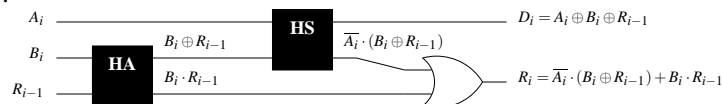
Correction : Une autre manière de procéder consiste à :

- (1) additionner B_i et R_{i-1} avec un demi-additionneur (cette opération peut évidemment engendrer une retenue) ;
- (2) puis en retrancher le résultat obtenu de A_i .

Cela est obtenu par transformation des fonctions logiques :

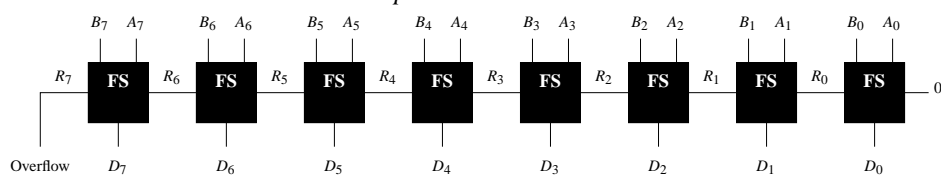
$$\begin{aligned}
 D_i &= A_i \oplus (B_i \oplus R_{i-1}) \\
 R_i &= \overline{A_i} \cdot (B_i \cdot \overline{R_{i-1}} + \overline{B_i} \cdot R_{i-1}) + (\overline{A_i} + A_i) \cdot B_i \cdot R_{i-1} \\
 &= \overline{A_i} \cdot (B_i \oplus R_{i-1}) + B_i \cdot R_{i-1}
 \end{aligned}$$

d'où le circuit :



(c) Réaliser un soustracteur parallèle pour mots de 8 bits.

Correction : On utilise 8 soustracteurs complets :



8. Le (dé)multiplexeur

Un *multiplexeur* est un circuit logique qui dispose de 2^n entrées, d'une unique sortie et de n lignes de sélection. Son principe de fonctionnement consiste à connecter, selon la configuration binaire présente sur les n lignes de sélection, l'une des entrées à la sortie. Les n lignes de sélection différencient 2^n configurations binaires, chacune de ces configurations correspondant à l'entrée du multiplexeur qui doit être connectée à la sortie.

Un *démultiplexeur*, pour sa part, est un circuit logique qui dispose d'une unique entrée, de 2^n sorties et de n lignes de sélection. Son principe de fonctionnement, à l'inverse de celui du multiplexeur, consiste à connecter, selon la configuration binaire présente sur les lignes de sélection, l'entrée à l'une des sorties.

(a) Réaliser un multiplexeur à quatre voies (c'est-à-dire un multiplexeur à quatre entrées).

Correction : Soit S_0, S_1 les lignes de sélection, I_0, \dots, I_4 les entrées et O la sortie (cf. fig. 7).

(b) Réaliser un démultiplexeur à quatre voies (c'est-à-dire un démultiplexeur à quatre sorties).

Correction : Soit S_0, S_1 les lignes de sélection, I l'entrée et O_0, \dots, O_3 les sorties (cf. fig. 8).

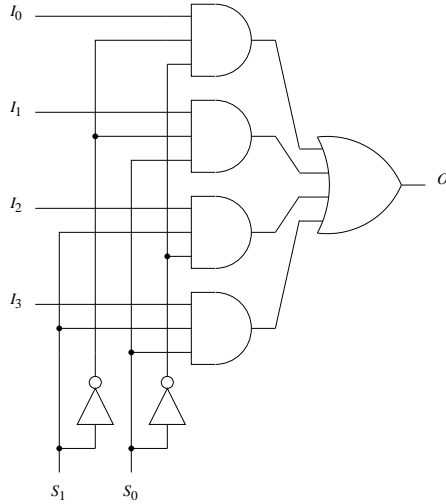


FIG. 7 – Multiplexeur à 4 voies.

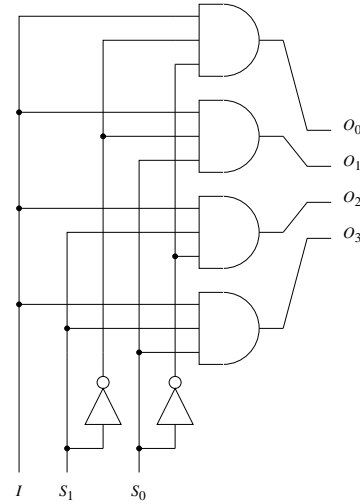


FIG. 8 – Démultiplexeur à 4 voies.

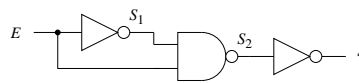
9. Les hasards logiques

Le *temps de passage d'une porte logique* est la durée entre l'instant où les signaux sont appliqués à l'entrée et celui où leur effet se répercute en sortie. Jusqu'à présent, ce temps de passage a été ignoré dans un souci de simplification. Toutefois, le temps de passage d'une porte logique n'est jamais nul (de l'ordre de 5 à 25 ns).

Si un étage logique est construit à l'aide de portes logiques (c'est-à-dire si la sortie d'une porte logique attaque l'une des entrées de la porte logique suivante) alors le temps de passage de l'étage est au moins égal à la somme des temps de passage des portes logiques qui le composent : dans ce cas, les temps de passage s'ajoutent. Il en résulte qu'un changement des données en entrée d'un montage, non seulement mettra un certain temps à se répercuter en sortie, mais pourra en plus provoquer des changements d'état (impulsions) non souhaités à la sortie. De telles impulsions parasites sont appelées *hasards logiques*.

(a) Mise en évidence d'un hasard logique.

i. Exprimer la valeur de la sortie S du circuit ci-dessous en fonction de son entrée E .



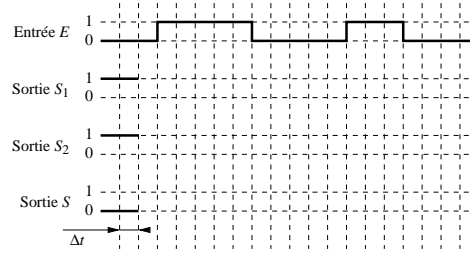
Correction :

$$S_1 = \overline{E}$$

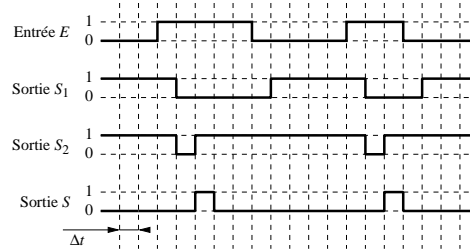
$$S_2 = \overline{\overline{E}} \cdot E = 1$$

$$S = \overline{S_2} = \overline{1} = 0$$

ii. Compléter le chronogramme suivant de ce circuit (on considère que toutes les portes logiques mises en jeu ont un même temps de passage Δt).



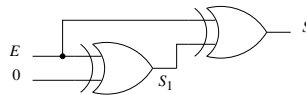
Correction :



Ce circuit se comporte comme un détecteur de transitions. Il peut être simplifié en remplaçant le nand et le deuxième not par un and, car $\text{not}(\text{nand}(a,b)) = \text{and}(a,b)$.

(b) Exemples de mise à profit des hasards logiques : *détecteur de transitions*.

i. Exprimer la valeur de la sortie S du circuit ci-dessous en fonction de son entrée E .



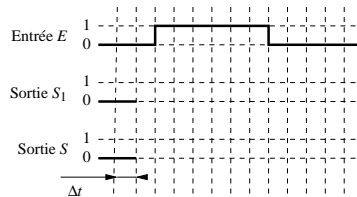
Correction :

$$S_1 = E \oplus 0 = E \quad (\text{c'est la fonction identité, mais avec un délai})$$

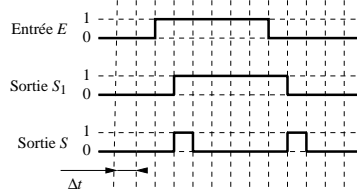
$$S = E \oplus E = 0$$

La sortie de ce circuit devrait donc toujours valoir 0. Mais le retard introduit par le premier xor implique que tout changement d'un état de E entraîne une impulsion positive de S comme le montre le chronogramme ci-dessous.

ii. Compléter le chronogramme suivant de ce circuit (on considère que toutes les portes logiques mises en jeu ont un même temps de passage Δt).

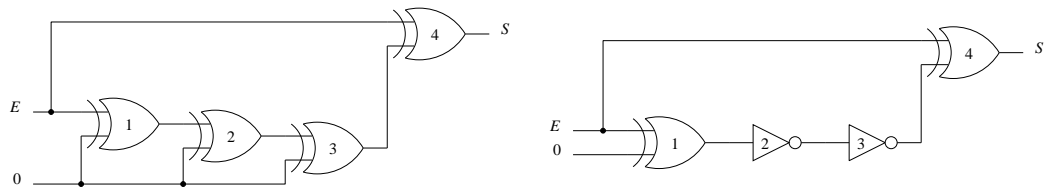


Correction :



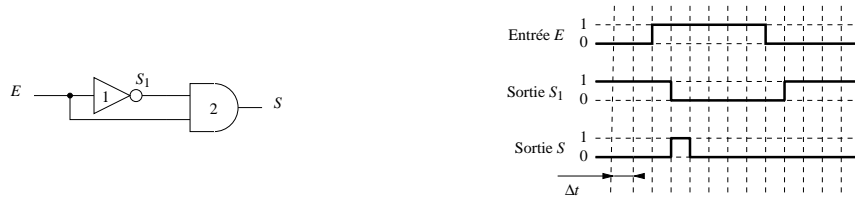
iii. Réaliser un détecteur de transitions pour lequel la durée des impulsions en S est de $3\Delta t$.

Correction : Il suffit d'avoir un délai de $3\Delta t$ entre les deux entrées du xor. Ce délai peut être obtenu en ajoutant au circuit précédent deux portes xor, ou à l'aide d'un autre circuit réalisant l'identité avec un délai de $3\Delta t$, comme par exemple un xor et deux not.



- (c) Réaliser un *détecteur de front montant*, c'est-à-dire un détecteur de transitions qui ne répond que lorsque le signal d'entrée passe d'un niveau bas à un niveau haut.

Correction : On se base toujours sur un délai entre deux entrées d'une porte logique. Les portes logiques choisies filtrent ensuite les cas souhaités. Ici, on utilise une porte not et and.



- (d) Réaliser un *détecteur de front descendant*, c'est-à-dire un détecteur de transitions qui ne répond que lorsque le signal d'entrée passe d'un niveau haut à un niveau bas.

Correction : Ici on utilise une porte logique not et nor.

