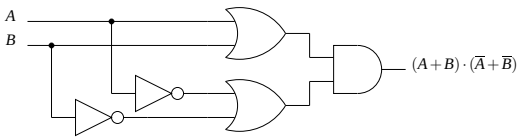
Architecture des ordinateurs
Corrigé du TD 4 : Circuits combinatoires

Arnaud Giersch, Benoît Meister et Frédéric Vivien

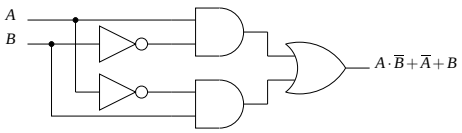
1. Exprimer la fonction **xor** comme un produit de sommes et réaliser le circuit logique correspondant.

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

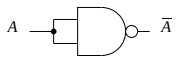
   Même question en exprimant **xor** comme une somme de produits.

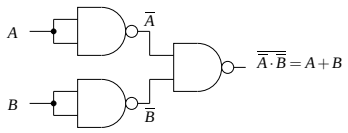
   Correction : $A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$

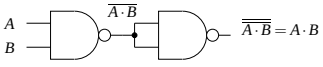
   

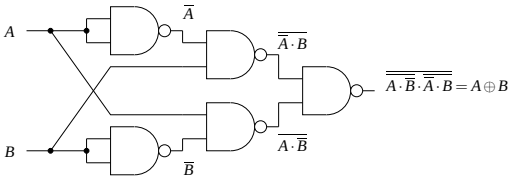
2. La fonction **nand** formant un groupe logique complet, réaliser, uniquement avec des portes **nand**, les circuits logiques **not, and, or** et **xor** (les formules sont rappelées ci-dessous).
   - $\text{not}(A) = \text{nand}(A,A)$
   - $\text{and}(A,B) = \text{nand}(\text{nand}(A,B),\text{nand}(A,B))$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

   Correction :

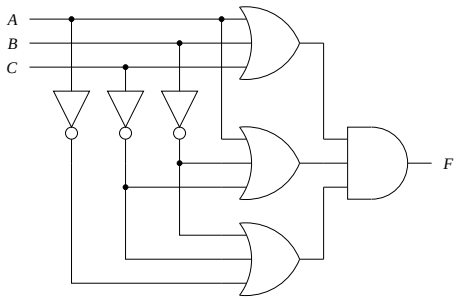
   

3. Réaliser un circuit logique qui implémente la fonction F.

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

   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.

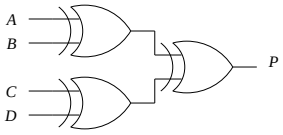
   Correction : *La formule pour le générateur de parité impaire sur 4 bits (P) obtenue directement à partir de la table de vérité est :*

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

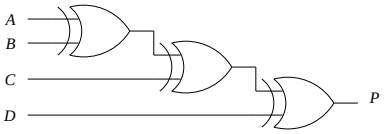
   ce qui donnerait un circuit beaucoup trop compliqué !
   On remarque que pour deux bits, $P = A \oplus B$:

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

   On en déduit les circuits suivants :

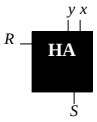


ou



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

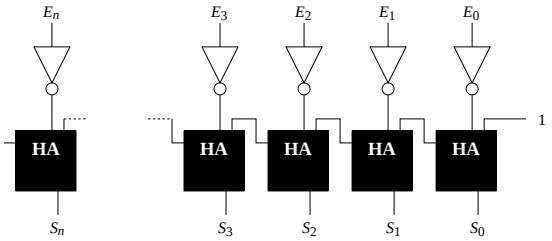
   Correction : *Le demi-additionneur possède deux entrées (x et y) et deux sorties (R et S). S correspond au bit de rang zéro du résultat de l'addition binaire de x et y, R au bit de rang* 1 *(retenue).*



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'incrémentation) 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 $A_{n-1}\dots A_1A_0$ le nombre binaire à décrémenter. Appelons $S_{n-1}\dots S_1S_0$ 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

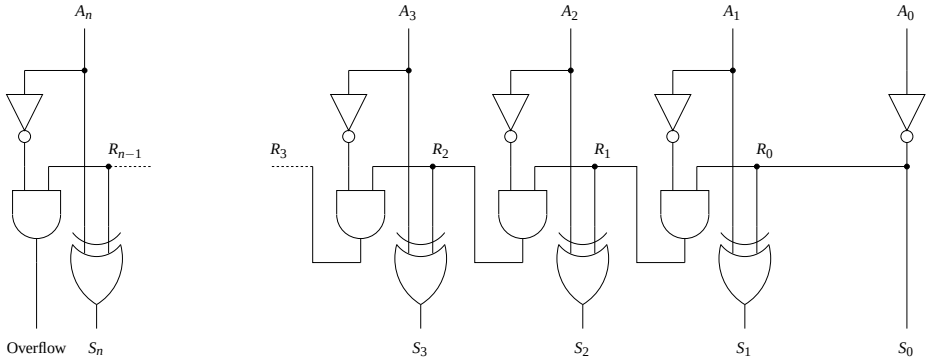
$$S_0 = \overline{A_0}$$
$$R_0 = \overline{A_0}$$

   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) :*

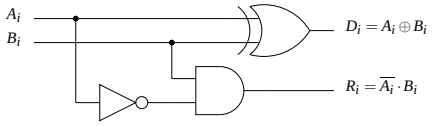
A_i	B_i	D_i	R_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

   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 :

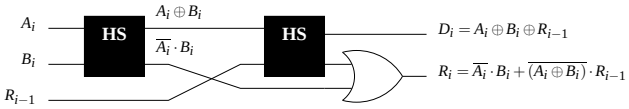
i. avec deux demi-soustracteurs ;

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 :



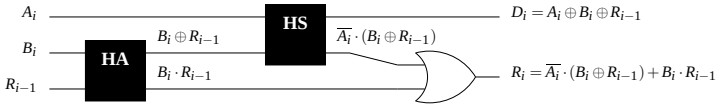
Ce schéma correspond au fait que le soustracteur est réalisé en
(1) retranchant B_i de A_i (1^{er} demi-soustracteur) ;
(2) puis en retranchant R_{i-i} de la différence obtenue.

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 :

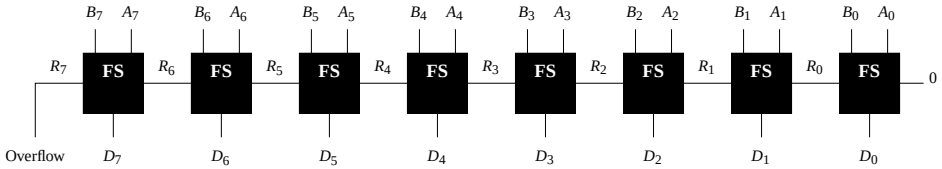
$$
\begin{aligned}
D_i &= A_i \oplus (B_i \oplus R_{i-1}) \\
R_i &= \overline{A_i} \cdot (B_i \cdot \overline{R_{i-1}} + \overline{B_i} \cdot R_{i-1}) + (\overline{A_i} + A_i) \cdot B_i \cdot R_{i-1} \\
&= \overline{A_i} \cdot (B_i \oplus R_{i-1}) + B_i \cdot R_{i-1}
\end{aligned}
$$

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, $I_0, \ldots, I_4$ les entrées et O la sortie (cf. fig. 1).*

(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 $O_0, \ldots, O_3$ les sorties (cf. fig. 2).*

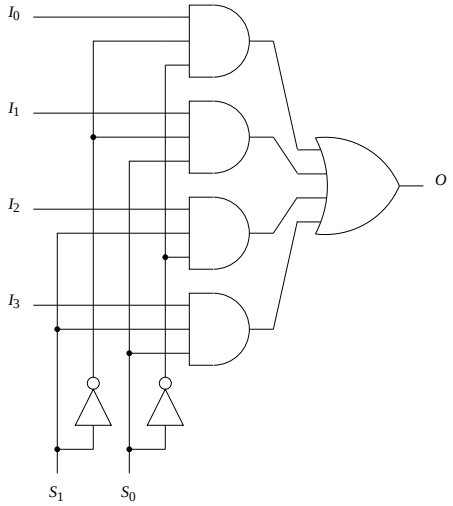


FIG. 1 – Multiplexeur à 4 voies.

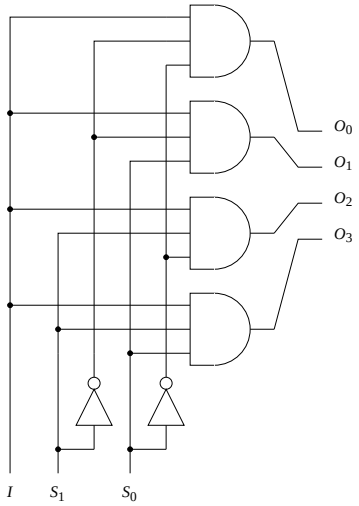


FIG. 2 – Démultiplexeur à 4 voies.

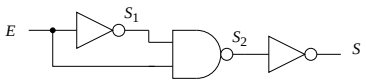
9. **Les hasards logiques**

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.

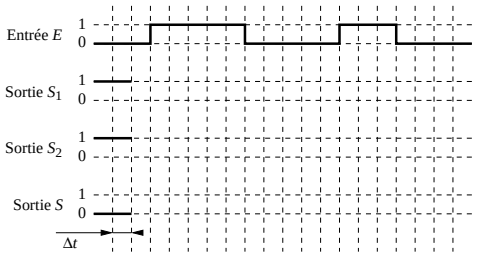
i. Exprimer la valeur de la sortie S du circuit ci-dessous en fonction de son entrée E.



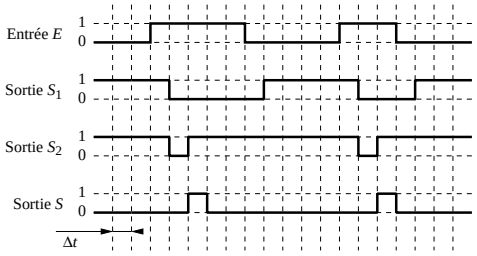
Correction :

$$S_1 = \overline{E}$$
$$S_2 = \overline{\overline{E} \cdot E} = 1$$
$$S = \overline{S_2} = \overline{1} = 0$$

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).



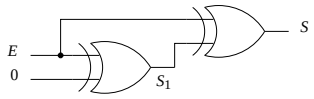
Correction :



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* not(nand(a,b)) = 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.

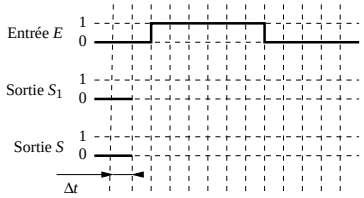


Correction :

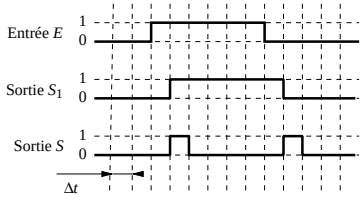
$$S_1 = E \oplus 0 = E \quad \textit{(c'est la fonction identité, mais avec un délai)}$$
$$S = E \oplus E = 0$$

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).

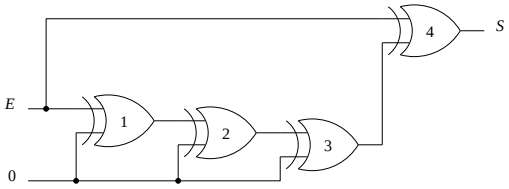


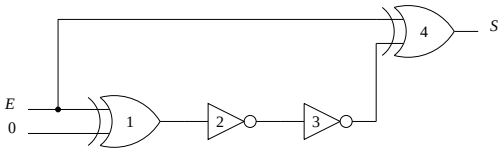
Correction :



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

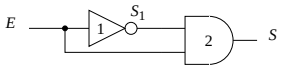
Correction : *Il suffit d'avoir un délai de* $3\Delta t$ *entre les deux entrées du* xor. *Ce délai peut être obtenu en ajoutant au circuit précédent deux portes* xor, *ou à l'aide d'un autre circuit réalisant l'identité avec un délai de* $3\Delta t$, *comme par exemple un* xor *et deux* not.

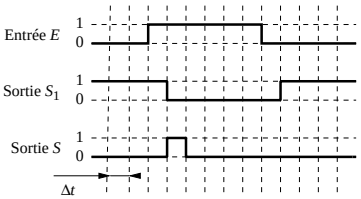




(c) Réaliser un *détecteur de front montant*, c'est-à-dire un détecteur de transitions qui ne répond que lorsque le signal d'entrée passe d'un niveau bas à un niveau haut.

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.