Architecture des ordinateurs : TD2

Université de Tours

`Département informatique de Blois`

Logique booléenne et circuits combinatoires

*
* *

Problème 1

On veut concevoir un circuit combinatoire permettant de comparer deux nombres A et B de 4 bits, $A = \langle a_3 a_2 a_1 a_0 \rangle_2$ et $B = \langle b_3 b_2 b_1 b_0 \rangle_2$.
Le circuit possède deux sorties :

$$G(A,B) = \begin{cases} 1 & \text{Si } A > B \\ 0 & \text{Sinon} \end{cases} \quad \text{et} \quad E(A,B) = \begin{cases} 1 & \text{Si } A = B \\ 0 & \text{Sinon} \end{cases}$$

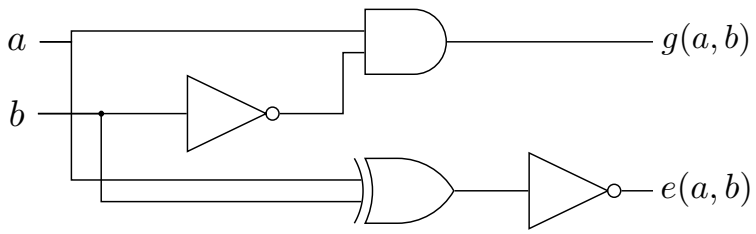
1. Soient a et b deux bits. Soit un circuit à deux sorties :

$$g(a,b) = \begin{cases} 1 & \text{Si } a > b \\ 0 & \text{Sinon} \end{cases} \quad \text{et} \quad e(a,b) = \begin{cases} 1 & \text{Si } a = b \\ 0 & \text{Sinon} \end{cases}.$$

   Donner l'expression logique de g et e puis dessiner le circuit correspondant à un comparateur 1 bit.

   On a : $g(a,b) = a \wedge \neg b$

   On a : $e(a,b) = a \Leftrightarrow b$
   $= \neg(a \oplus b)$



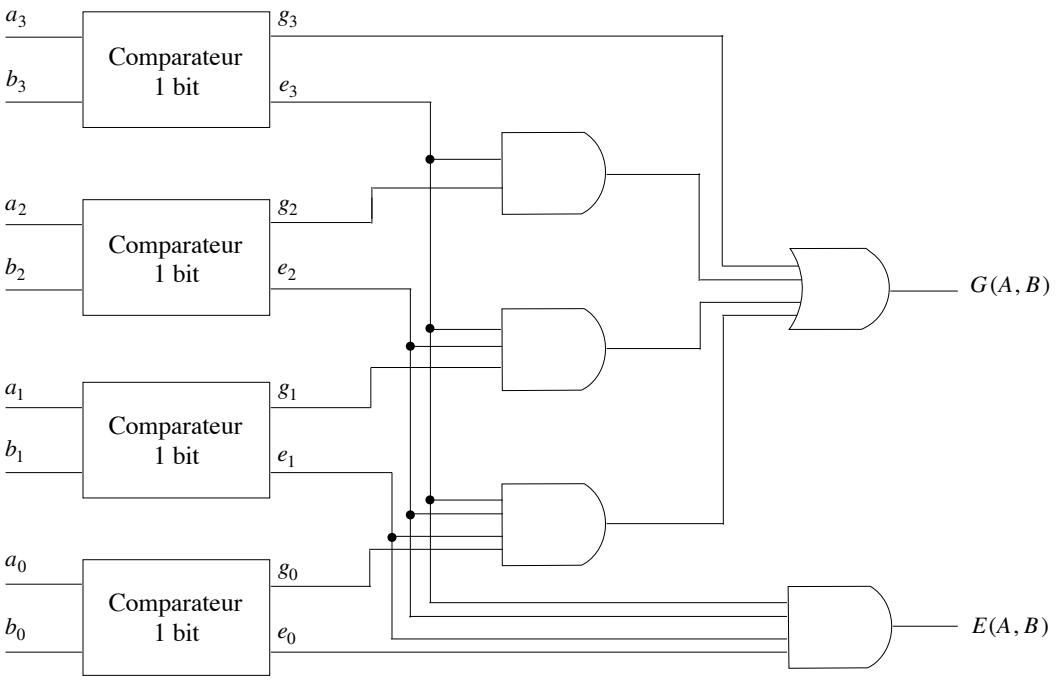
2. En utilisant le circuit de la question précédente ainsi que des portes logiques $\vee, \wedge$ (éventuellement à entrées ≥ 3) et $\neg$, proposer un circuit permettant de comparer deux nombres de 4 bits.

   On a $A > B = G(A,B)$ dans un des cas suivants est vrai :

   - $a_3 > b_3 \equiv g_3$
   - $a_3 = b_3 \wedge a_2 > b_2 \equiv e_3 \wedge g_2$
   - $a_3 = b_3 \wedge a_2 = b_2 \wedge a_1 > b_1 \equiv e_3 \wedge e_2 \wedge g_1$
   - $a_3 = b_3 \wedge a_2 = b_2 \wedge a_1 = b_1 \wedge a_0 > b_0 \equiv e_3 \wedge e_2 \wedge e_1 \wedge g_0$

On réalise chacun de ces cas au sein du circuit puis on les fusionne par une porte OU (car au moins un des cas doit être à vrai).

Pour la fonction $E(A, B)$, celle-ci vaut vrai si et seulement si $\forall i \in \{0, 1, 2, 3\}, a_i = b_i$, soit $E(A, B) = e_3 \wedge e_2 \wedge e_1 \wedge e_0$.



Problème 2

On cherche à représenter la fonction implémentant un *hidden bit*.
Cette fonction prend en entrée k valeurs booléennes et retourne une valeur booléenne. Soient k entrées binaires $a_1, a_2, ..., a_k$ et soit $s = \text{card}(\{i | a_i = 1\})$.
La sortie S est alors égale à 0 si $s = 0$ et elle est égale à a_s si $s \in [\![1, k]\!]$.

1. Dresser la table de vérité de S pour $k = 3$.
2. Simplifier S à l'aide des tableaux de Karnaugh.
3. Dessiner le circuit combinatoire correspondant. À quel autre circuit celui-ci correspond-il ?

Problème 3

L'opérateur $Nand$ noté $\uparrow$ est un opérateur très utilisé en électronique et dans la réalisation des microprocesseurs car il forme un système complet de connecteurs à lui seul.

1. Montrer que $x \oplus y = [(x \uparrow y) \uparrow x] \uparrow [y \uparrow (x \uparrow y)]$. On rappelle que l'opérateur $\oplus$ désigne le $OU\ exclusif$ (ou Xor).

On rappelle que : $x \uparrow y = \neg(x \wedge y)$

De, même, on rappelle que $x \oplus y = (x \vee y) \wedge (\neg x \vee \neg y)$, dès lors, on a :

$$\begin{aligned} x \oplus y &= (x \vee y) \wedge (\neg x \vee \neg y) \\ &= (x \vee y) \wedge \neg(x \wedge y) \\ &= (x \vee y) \wedge (x \uparrow y) \end{aligned}$$

$= [x \wedge (x \uparrow y)] \vee [y \wedge (x \uparrow y)]$ *Distributivité*

$= \neg(\neg[x \wedge (x \uparrow y)] \wedge \neg[y \wedge (x \uparrow y)])$ *Double négation* et *Loi de de Morgan*

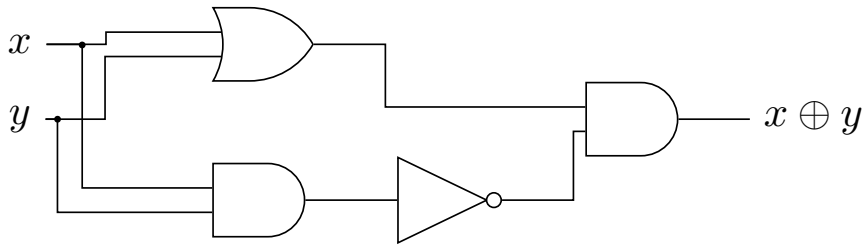
$= \neg([x \uparrow (x \uparrow y)] \wedge [y \uparrow (x \uparrow y)])$

$= [x \uparrow (x \uparrow y)] \uparrow [y \uparrow (x \uparrow y)]$

2. Sur la modélisation de $\oplus$:

   (a) Proposer un circuit bien modélisé de l'opérateur $\oplus$ à l'aide du système d'opérateurs $\{\vee, \wedge, \neg\}$.

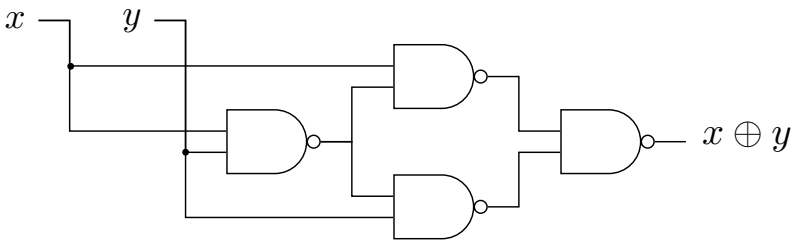
   On utilise le fait que $x \oplus y = (x \vee y) \wedge \neg(x \wedge y)$



   (b) Expliquer pourquoi la modélisation 2.(a) n'est pas satisfaisante. Proposer un circuit logique à l'aide de l'opérateur Nand. Pourquoi cette modélisation est meilleure ?

   Le circuit précédent n'est pas satisfaisant car il ne minimise pas le nombre de portes logiques différentes.

   À l'aide du connecteur Nand, on obtient le circuit suivant :



   Ce circuit est meilleur que le précédent car il contient le même nombre de portes logiques mais utilise uniquement le Nand qui est un système complet ; ceci permet des économies en terme de commande de composants ou en simplicité de gravure des wafers.

Problème 4

Soit la fonction booléenne P de n variables booléennes définie telle que :

$$P(x_1, ..., x_n) = \bigoplus_{i=1}^{n} x_i$$

Où $\oplus$ désigne l'opérateur Ou exclusif (XOR). On rappelle que l'opérateur $\oplus$ est associatif et commutatif, ainsi P s'écrit aussi comme $P(x_1, ..., x_n) = x_1 \oplus x_2 \oplus ... \oplus x_n$.
Cette fonction est appelée fonction de *clé de parité*.

1. Montrer que la valeur de la fonction P est 1 si et seulement s'il y'a, parmi $x_1, ..., x_n$, un nombre impair de variables valant 1.

Par commutativité et associativité de $\oplus$, il est possible de grouper les variables x_i selon leur valeur de vérité. On crée les ensembles $E^+ = \{x_i | x_i = 1\}$ et $E^- = \{x_j | x_j = 0\}$. On a évidemment $E^+ \cap E^- = \emptyset$ et $E^+ \cup E^- = \{x_1, ..., x_n\}$.

Dès lors : $P(x_1, ..., x_n) = \bigoplus_{x_i \in E^+} x_i \oplus \bigoplus_{x_j \in E^-} x_j$

$= \bigoplus_{x_i \in E^+} x_i$ (Car 0 est l'élément neutre de $\oplus$.)

$= \underbrace{\underbrace{1}_{1} \oplus 1 \oplus 1 \oplus 1 \oplus ... \oplus 1}_{0} = 0$

On observe la forme d'une suite de forme 1 0 1 0 1 0 ... .

On en déduit que :

$$P(x_1, ..., x_n) = \begin{cases} 1 & \text{si } |E^+| = 2n + 1 \\ 0 & \text{sinon} \end{cases}$$

2. On considère x, un vecteur binaire tel que $x = \langle x_1, ..., x_n \rangle$.

   (a) Écrire une fonction `cle_parite` en C ou en Java utilisant les opérateurs ^ (ou exclusif) et $\gg$ (décalage à droite) qui implémente la fonction $P(x)$.

```
int cle_parite(int x) {

    int cle = 0; // Initialisation à l'élément neutre
    for(; x > 0; x >>= 1) { // Suppression du dernier bit à chaque itération
        cle ^= (x & 1); // Récupération du dernier bit (opérateur & 1)
                        // puis XOR (opérateur ^)
    }
    return cle;

}
```

   (b) Écrire une fonction `cle_parite_log` en C ou en Java, plus efficace, permettant de calculer $P(x)$ s'appuyant sur l'exemple ci-dessous.

   Exemple : pour 8 bits stockés dans la donnée x, on calcule :

   $y = 4$ forts $\oplus$ 4 faibles de $x = \langle x_7 \oplus x_3, x_6 \oplus x_2, x_5 \oplus x_1, x_4 \oplus x_0 \rangle$

   $z = 2$ forts $\oplus$ 2 faibles de $y = \langle y_3 \oplus x_1, y_2 \oplus y_0 \rangle$

   $t = z_1 \oplus z_0$ puis on retourne t.

3. Combien d'étapes pour 16, 32, 64 bits ? pour 2^n ? Combien d'étapes avec la méthode initiale ? Pourquoi qualifie t-on cette méthode de logarithmique ?

4. Appliquer la méthode de la question 2.b pour les valeurs $x = 2^8 - 1$ et $x = -15$.

5. Traduire sur papier les méthodes 2.a et 2.b du calcul de la clé de parité $P(x)$ sous forme de circuits logiques à 8 entrée et 1 sortie, avec des portes XOR.

Problème 5

On dit qu'une fonction booléenne est sous *forme minimale* si elle se réduit au système de connecteurs $\{\Rightarrow, 0\}$. Où 0 représente la valeur FAUX d'arité 0.

On veut montrer que toute formule $f \in \mathcal{B}$ admet une forme minimale équivalente.

1. Montrer que les opérateurs $\neg$ et $\vee$ peuvent être exprimés à l'aide du système $\{\Rightarrow, 0\}$.

On sait que $p \Rightarrow 0 \equiv \neg p \vee 0$. Dès lors, $p \Rightarrow 0 \equiv \neg p$.

Par linéarité de $\neg$, on déduit que $p \vee q \equiv \neg p \Rightarrow q$. Dès lors, on a $p \vee q \equiv (p \Rightarrow 0) \Rightarrow q$.

2. Donner la table de vérité de la formule $(p \Rightarrow (q \Rightarrow 0)) \Rightarrow 0$ et la comparer à $p \wedge q$.

p	q	$(p \Rightarrow (q \Rightarrow 0))$	$((p \Rightarrow (q \Rightarrow 0)) \Rightarrow 0)$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

La table de vérité de la formule $((p \Rightarrow (q \Rightarrow 0)) \Rightarrow 0)$ est équivalente à celle de $p \wedge q$.

3. Que pouvez-vous déduire à l'aide des questions précédentes ?

On peut déduire que $\{\Rightarrow, 0\}$ forme un système complet de connecteurs.

4. Déduire des questions précédentes une fonction `min` qui transforme toute fonction f de la logique booléenne en une fonction équivalente sous forme minimale.

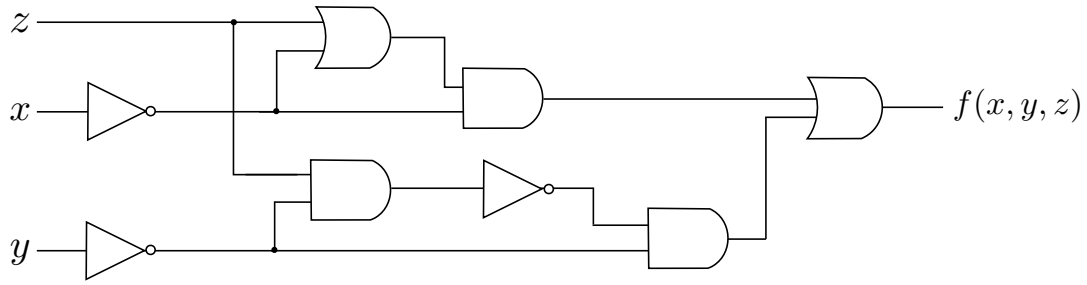
Soient $x \in B$ et $f, f' \in \mathcal{B}^2$. On considère l'ensemble de règles suivant :

- $\texttt{min}(0) = 0$
- $\texttt{min}(1) = 0 \Rightarrow 0$
- $\texttt{min}(x) = x$
- $\texttt{min}(f \Rightarrow f') = \texttt{min}(f) \Rightarrow \texttt{min}(f')$
- $\texttt{min}(f) = \texttt{min}(f) \Rightarrow 0$
- $\texttt{min}(f \vee f') = (\texttt{min}(f) \Rightarrow 0) \Rightarrow \texttt{min}(f')$
- $\texttt{min}(f \wedge f') = (\texttt{min}(f) \Rightarrow (\texttt{min}(f') \Rightarrow 0)) \Rightarrow 0$

Problème 6

Démontrer les assertions vraies. Donner un contre-exemple ou justifier soigneusement les assertions fausses.

1. L'unique connecteur unaire existant en logique booléenne est la négation $\neg$.
2. Le système de connecteurs $\{\oplus, \vee, 1\}$ est complet. On précise que 1 est la constante VRAI d'arité 0.
3. Il est vrai que $x \uparrow y = [(x \downarrow x) \downarrow (y \downarrow y)] \downarrow [(x \downarrow x) \downarrow (y \downarrow y)]$.
4. La fonction logique $f(x, y, z) = \neg x \wedge y$ est identique au circuit logique ci-dessous.



5. Soient x_3, x_2, x_1 et x_0 quatre variables booléennes. On note $x = \langle x_3 x_2 x_1 x_0 \rangle_2$. On pose la fonction booléenne $\varphi(x) = \begin{cases} 1 & \text{si } x < \langle 1001 \rangle_2 \\ 0 & \text{sinon} \end{cases}$.

   Il est vrai que $\varphi(x) = \neg x_3 \vee (\neg x_2 \wedge \neg x_1)$.