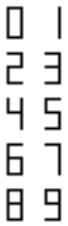
# ES102/PC4 : énoncé et corrigé

## 1) Portes CMOS complexes pour afficheur 7 segments et quelques leçons à en tirer



Malgré l'omniprésence d'écrans à tout faire, l'affichage de chiffres décimaux se contente encore souvent d'afficheurs plus simples, dits à 7 segments (tous allumés pour afficher le chiffre 8 : voir ci-contre). Pour les premières questions ci-dessous, on affiche en fait seulement les chiffres *k* entre 0 et 7 (version octale). Le code binaire de *k* est noté $(qdu)_2$. A chacun des 7 segments correspond donc une fonction booléenne de q, d et u.

1a) L'une d'elles est-elle croissante ? Ou décroissante ?

Aucune n'est constante car chaque segment est tantôt éteint, tantôt allumé, selon les chiffres. Pour une fonction croissante, il faut donc d'abord un segment éteint en 0 et allumé en 7 : il n'y en a aucun. Pour une fonction décroissante, c'est l'inverse qu'il faut : 3 candidats apparaissent. Malheureusement, pour chacun, il existe une arête du 3-cube où le segment réapparaît quand la variable passe de 0 à 1 : entre les chiffres 1 et 3 pour le segment bas, entre 4 et 6 pour le segment droit bas, entre 2 et 6 pour le segment droit haut.

Bref, aucune des 7 fonctions n'est croissante, ni décroissante. Rien d'étonnant à cela. Sans propriété particulière, ce sont des échantillons quelconques de $F_3$ (où les fonctions croissantes et décroissantes ne sont qu'une petite minorité). On parle de « logique aléatoire ». Malgré cela, plusieurs rencontres intéressantes (certes provoquées ;-) nous attendent ci-dessous. Mais on sait d'avance qu'elles nécessiteront toujours quelques inverseurs pour complémenter certaines entrées (et qu'il ne suffira jamais d'en mettre un en sortie).

1b) Concevoir en logique complémentaire la porte CMOS qui fournit la fonction booléenne *sdb* servant à commander le **s**egment situé sur le côté **d**roit, en **b**as.

Ce cas est traité sur la figure suivante, à gauche. Une table de Karnaugh montre les 8 chiffres à afficher. Le segment droit bas (sdb) est épaissi/noirci lorsqu'il est présent. C'est le cas dans toutes les cases sauf en (q, d, u) = (0, 1, 0). On reconnaît ainsi une NAND3(q', d, u').

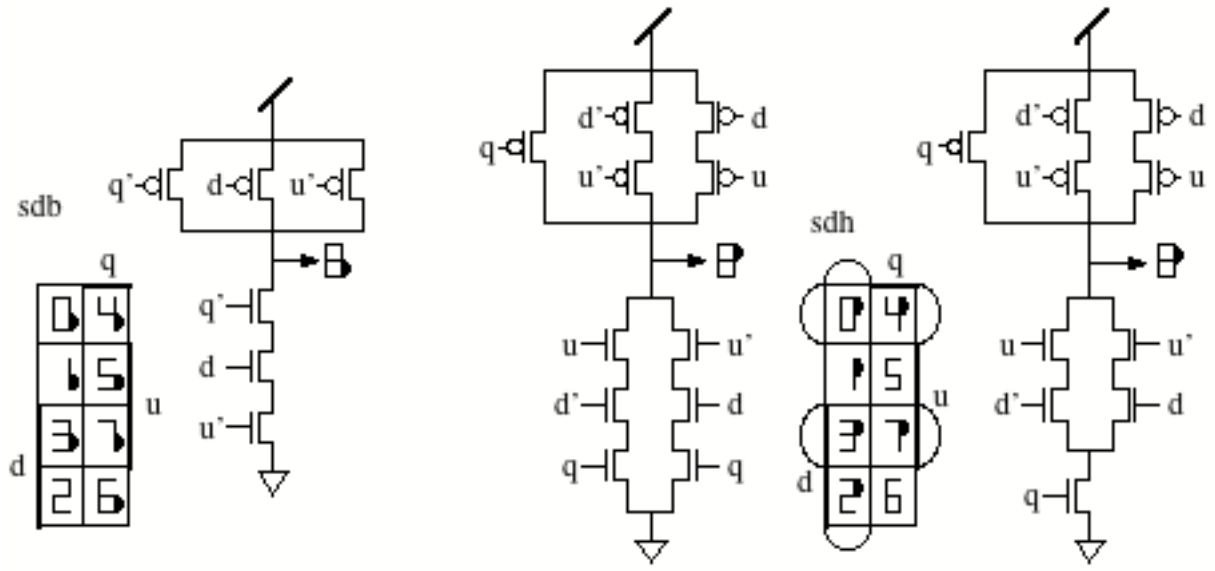
Mais on profite de ce cas élémentaire pour éprouver la méthodologie vue en cours. Selon celle-ci, FDM(sdb)/ fournit la structure du bloc *p*. En d'autres termes, il faut mettre *sdb* sous forme disjonctive minimale, puis complémenter chaque variable (ce que signifie le / ci-dessus). On obtient alors une formule non plus booléenne mais *structurelle* décrivant le bloc *p* en faisant correspondre ET et OU logiques respectivement à des mises en série et en parallèle de transistors *p* commandés par les variables. Ici, FDM(sdb)=q+d'+u, d'où le bloc *p* ci-dessous.

Le bloc *n*, qui produit les 0 de la fonction, s'obtient de manière similaire en mettant sous FDM le complémentaire de *sdb*. Or FDM(/sdb)=q'du', d'où le bloc *n* ci-dessous. Finalement, les formules de synthèse vues en cours conduisent bien à la porte NAND à 3 entrées attendue.

1c) Concevoir la porte CMOS fournissant la commande *sdh* du **s**egment situé côté **d**roit, en **h**aut.

Ce cas est traité sur la partie droite de la figure suivante. Sur la table de Karnaugh, les 1 de la fonction sont regroupés visuellement en 3 « gélules » qui correspondent chacune à un sous-cube maximal. Pour synthétiser le bloc *p*, on recourt à FDM(sdh)/ = q+d'u'+du et, pour le bloc *n*, à FDM(/sdh) = qd'u+qdu'. D'où l'implantation en logique complémentaire présentée ci-dessus au centre.

1d) Dans le bloc *n* de la porte CMOS que l'on vient de synthétiser pour *sdh*, il est naturel de mettre en commun les transistors commandés par q. Quel en est le sens algébrique ?




La mise en commun aboutit au schéma de droite sur la figure précédente. La formule structurelle du bloc *n* est q(d'u+du'). Ce n'est plus une forme disjonctive (∑∏), mais un produit de sommes de produits (∏∑∏), c'est-à-dire une forme à 3 niveaux et non plus 2. On passe ainsi, certes modestement, d'une minimisation *biniveau* à une minimisation *multiniveau* de la fonction booléenne /sdh, avec gain en compacité.

1e) Simplifier le montage précédent dans le cas où q=1. Qu'obtient on ? Comment modifier le montage pour obtenir une porte XOR ?

Non passant, le pMOS commandé par q=1 peut être supprimé. Passant, le nMOS commandé par q=1 revient à une connexion permanente. Le résultat est une implantation de la porte XNOR car, pour q=1, sdh=(d⊕u)'. Elle nécessite 8 transistors (contre 4 pour les portes NAND ou NOR). Il faut en outre 2 inverseurs pour les entrées, d'où un total de 12 transistors.
Pour obtenir une XOR, il suffit de permuter l'une des deux variables (d ou u) avec son complément, partout où elle apparaît.

1f) Comment se positionne le délai d'une porte XOR ainsi réalisée par rapport à celui de ses consoeurs ? Est-ce cohérent avec les performances de la *bibliothèque* de portes de la PC2 ?

Le montage obtenu ci-dessus souffre du même handicap que la porte NOR : 2 pMOS en série. Il faut aussi ajouter un délai d'inverseur, comme il le faudrait pour une OR. Finalement on doit avoir $\tau_{XOR}\approx\tau_{OR}$. Dans PC2/Exo3, on avait $\tau_{OR}=2{,}5\tau_{inv}$ et $\tau_{XOR}=3\tau_{inv}$, grandeurs de fait assez proches. En tout cas, on sait désormais pourquoi $\tau_{XOR}$ ne pouvait être égalé par un montage en portes : son implantation directe en transistors est fondamentalement plus rapide. Par ailleurs, connaissant mieux la porte NOR désormais, on s'étonnera d'avoir eu $\tau_{NOR}=\tau_{NAND}$…

1g) Retrouver la FDM de *sdh* par la méthode de Quine.

Afin de bien illustrer le principe de fusion multiple, chaque produit de variables est précédé ci-dessous du ou des chiffres qu'il représente. Les produits en caractère gras sont ceux qui s'avèrent ne participer à aucune fusion : ce sont les impliquants premiers de sdh. On montre ensuite facilement qu'ils sont essentiels. La disjonction des 3 constitue donc *la* FDM de sdh.

| | | |
|---|---|---|
| 0 : q'd'u' | 0∪1 : q'd' | (0∪1)∪ (2∪3) : **q'** |
| 1 : q'd'u | 0∪2 : q'u' | (0∪2)∪ (1∪3) |
| 2 : q'du' | 0∪4 : **d'u'** | |
| 4 : qd'u' | 1∪3 : q'u | |
| 3 : q'du | 2∪3 : q'd | |
| 7 : qdu | 3∪7 : **du** | |

Ainsi a-t-on un peu pris conscience de la lourdeur de cet algorithme glouton.

On souhaite désormais afficher les 10 chiffres décimaux, d'où *k* codé sur 4 bits : $(hqdu)_2$. Cependant, les possibilités 10 à 15 sont inexploitées. A chacun des 7 segments de l'afficheur correspond donc une fonction booléenne *incomplètement spécifiée*.

1h) Traitant d'abord les cas indifférents comme des 1, déterminer, sans écrire la moindre formule booléenne, le nombre de transistors nécessaires pour réaliser la porte CMOS fournissant la commande *sgh* du **s**egment situé sur le côté **g**auche, en **h**aut.

Il faut dessiner la table de Karnaugh, ce qui est fait sur la figure ci-dessous à gauche. Le recouvrement des 1 se fait au moyen de sous-cubes de dimensions respectives : 3, 2, 2 et 2. Le nombre de pMOS nécessaires est la somme de leurs codimensions : 1+2+2+2=7. Le recouvrement des 0 se fait par 3 sous-cubes de dimension 1, d'où 3x3=9 nMOS nécessaires.

1i) Les cas indifférents peuvent-ils être mieux exploités pour minimiser le coût en transistors ?

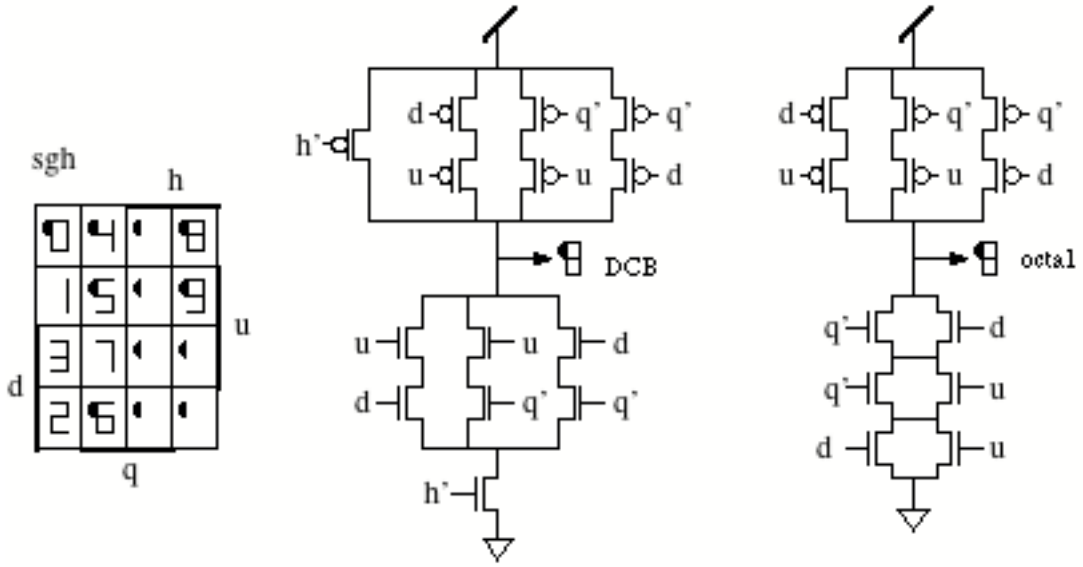
Oui, en mettant plutôt des 0 là où hd(q'+u)=1, on augmente de 1 le nombre de pMOS mais on diminue de 2 celui des nMOS. D'où une petite économie d'un transistor.

1j) Au vu des deux questions précédentes, il serait tentant d'exploiter différemment les cas indifférents selon que l'on synthétise le bloc *p* ou le bloc *n*. Pourquoi ne faut-il pas le faire ?

La tentation est de mettre plus de 1 dans les cas indifférents pour le bloc *p*, et plus de 0 pour le bloc *n*, pour bénéficier de regroupements plus avantageux. Mais ceci revient à provoquer sciemment un court-circuit alimentation-masse sur les cas indifférents concernés. Or, ce n'est pas parce qu'on ne compte pas utiliser ces cas qu'ils ne vont pas se produire, ne serait-ce qu'au hasard des transitions sur h, q, d et u. On s'interdit donc de telles pratiques !

1k) En fait, on reste sur des 1 pour les cas indifférents car cela va permettre une factorisation avantageuse de FDM. Achever de concevoir la porte CMOS fournissant *sgh*.

L'exploitation de la table de Karnaugh ci-dessous fournit FDM(sgh) = h+d'u'+qu'+qd'. La structure du bloc *p* est donc donnée par FDM(sgh)/ = h'+du+q'u+q'd. Le bloc *n* découle de FDM(/sgh) = h'du+h'q'u+h'q'd, que l'on factorise par h', d'où l'implantation ci-dessous au centre.




1l) Pour h=0, quel genre de fonction est *sgh* ? Dans l'implantation correspondante, constater que les formules structurelles des blocs *n* et *p* ne sont pas duales, contrairement aux observations faites jusque-là sur des cas plus simples. Si on forçait $\gamma_n$ à être duale de $\gamma_p$, que se passerait-il ? Quelle conclusion en tirer ?

Sur la table, côté h=0, il y a seulement quatre 1, regroupés autour de k=4. Cette configuration est typique d'une fonction majorité. Plus précisément, pour h=0, sgh=Maj(q,d',u').

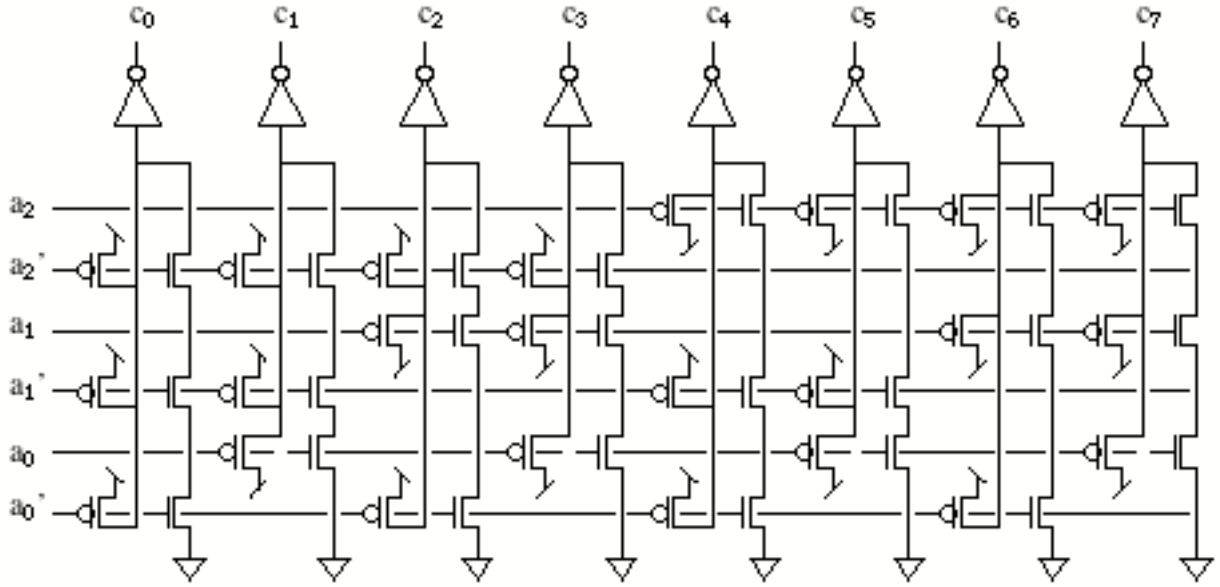
Les formules $\gamma_n$ et $\gamma_p$ (avec h=0) ne sont pas duales : on ne passe pas de l'une à l'autre en échangeant + et ·. La propriété de dualité, observée sur les portes les plus simples, trouve donc ici un (premier) contre-exemple.

La formule duale de du+q'u+q'd est (d+u)(q'+u)(q'+d). Elle correspond au montage de droite sur la figure ci-dessus : à chaque mise en parallèle dans le bloc *p* correspond une mise en série dans le bloc *n*, et vice-versa. Ce montage est correct puisqu'il respecte par construction les équations f=$\gamma_p$/ et f=/$\gamma_n$. Mais le bloc *n* présente alors des séries de 3 transistors au lieu de 2, via lesquelles il sera moins rapide d'imposer un 0 en sortie.

On comprend ici l'intérêt d'implanter séparément les blocs *n* et *p*, chacun bénéficiant de son propre calcul de FDM, plutôt que passer de l'un à l'autre par échange série/parallèle (alias transformation à la De Morgan).

## 2) Question d'adresse

Quelle est la fonction réalisée par la structure CMOS ci-dessous ?




Le montage ci-dessus s'avère constitué de 8 portes AND3 juxtaposées de gauche à droite. Chacune est faite d'une NAND3 surmontée d'un inverseur. Deux fils horizontaux amènent chaque entrée et son complémentaire à chaque porte. Il en résulte une organisation matricielle très régulière qui impose en outre le repliement des blocs *n* et *p* en vis-à-vis au sein de chaque NAND3. Mais chacune est différente. Pour celle de gauche, le bloc *n* est passant lorsque $a_0'a_1'a_2'$=1. Donc $c_0$=1 ssi $a_0$=$a_1$=$a_2$=0. Il y a une seule différence avec la suivante : $a_0'$ voit son rôle pris par $a_0$. Donc $c_1$=1 ssi $a_0$=1 et $a_1$=$a_2$=0. On réalise finalement que $c_i$=1 ssi $(a_2a_1a_0)_2$=i. Ceci tient à l'organisation très particulière des transistors : alternance de présence ou d'absence de transistors qui correspondent exactement aux positions des bits 0 et 1 lorsqu'on énumère les codes des entiers de 0 à 7. Nommons A le nombre $(a_2a_1a_0)_2$. La fonction assurée par le montage ci-dessus est donc de rendre toutes les sorties $c_i$ nulles, sauf $c_A$=1.

Le montage ci-dessus est ainsi qualifié de *décodeur* au sens où la i-ème sortie est activée (mise à 1) par présentation de son *code* binaire sur les entrées. Le code en question est souvent une *adresse*. Un décodeur (d'adresse) est effectivement requis pour aller chercher une donnée rangée à une certaine adresse en mémoire, elle aussi organisée de façon matricielle et où chaque **c**olonne (ou ligne) de cellules mémoire est pilotée par un signal $c_i$ qui la traverse de bout en bout.