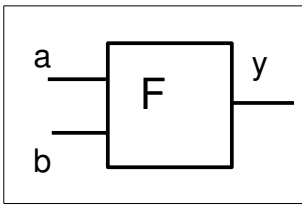
TD1 ENSL1 : Fonctions logiques élémentaires

WIKI : accessible depuis http://moutou.pagesperso-orange.fr/

I) Définitions

Un état logique est représenté par une valeur binaire (ou booléenne) : {0,1}, {vrai, faux}, {true, false}.
Une variable booléenne (ou variable logique [Logic variable, Binary variable, Boolean variable]) est une grandeur représentée par un symbole pouvant prendre des valeurs booléennes.
Une fonction logique [Logic function] est une fonction d'une ou plusieurs variables booléennes.
Cette fonction sera représentée par un dessin ou en langage de description matérielle comme ci-dessous :




```
-- Commentaire VHDL
ENTITY Fct IS
PORT(a,b : IN BIT;
       y : OUT BIT);
END Fct;
```

II) Représentation

[Truth table] : table de vérité

Comment savoir ce que fait une fonction booléenne ?
En énumérant toutes les possibilités sur les entrées, c'est ce que l'on appelle une table de vérité.
Avec cette table on connaît parfaitement la fonction F.
Une table de vérité comporte deux parties :
- partie gauche appelée "SI" concerne les entrées

-partie droite appelée "ALORS" concerne les sorties

Table de vérité

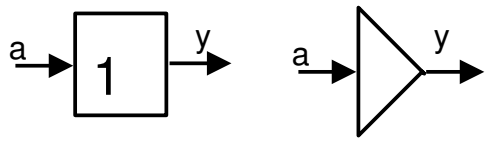
b	a	y
0	0	0
0	1	0
1	0	0
1	1	1

III) Fonctions élémentaires

Il existe des fonctions qu'il faut parfaitement connaître pour faire de la logique. Nous allons présenter maintenant les portes les plus courantes. [We will now describe commonly used gates]

1°) Fonctions d'une seule variable

fonction OUI

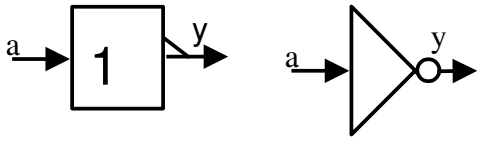


a	y
0	0
1	1

y=a

```
--VHDL
y <= a;
```

fonction NON [NOT]



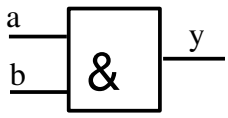
a	y
0	1
1	0

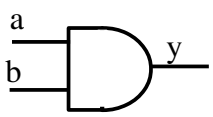
$y=\bar{a}$

```
--VHDL
y <= NOT a;
```

2°) Fonctions de deux variables

fonction ET [AND]



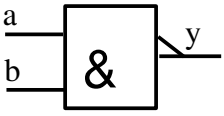


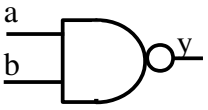
b	a	y
0	0	0
0	1	0
1	0	0
1	1	1

y=a.b

```
--VHDL
y <= a AND b;
```

fonction ET-NON [NAND]



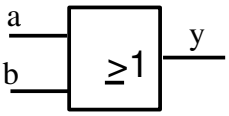


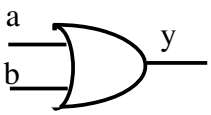
b	a	y
0	0	1
0	1	1
1	0	1
1	1	0

$y = \overline{a.b}$

```
--VHDL
y <= a NAND b;
```

Fonction OU [OR]



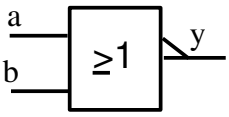


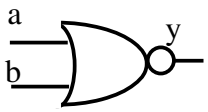
b	a	y
0	0	0
0	1	1
1	0	1
1	1	1

y=a+b

```
--VHDL
y <= a OR b;
```

Fonction OU-NON [NOR]



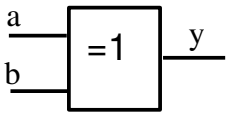


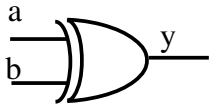
b	a	y
0	0	1
0	1	0
1	0	0
1	1	0

$y = \overline{a+b}$

```
--VHDL
y <= a NOR b;
```

Fonction OU Exclusif [XOR]





b	a	y
0	0	0
0	1	1
1	0	1
1	1	0

y = a⊕b

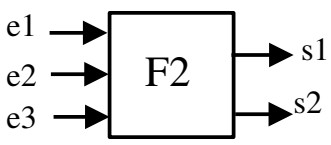
```
--VHDL
y <= a XOR b;
```

Le complément d'un ou exclusif est la fonction identité [XNOR].

IV) Exercices

Exercice 1

Obtain the XNOR function's truth table.
Write a Boolean expression for XOR and XNOR functions.
Write a VHDL entity for F2 function.



Exercice 2

Remplir le tableau (la dernière ligne en dernier). L'équivalence de F d'après /F est ce que l'on

appelle les lois de De Morgan.

Entrées			***ET Logiques***				***OU Logiques***			
	b	a								
	0	0	1	0	0	0	1	1	1	0
	0	1	0	1	0	0	1	1	0	1
	1	0	0	0	1	0	1	0	1	1
	1	1	0	0	0	1	0	1	1	1
Expression F										
Expression /F										
Symbole ANSI										
Symbole normalisé										
équivalent d'après /F										

Indication :
L'obtention de l'équation à partir de la table de vérité est immédiate pour le ET. On utilise la règle : recherche du 1 et écriture du terme correspondant. Cette même technique pour le OU donnerait trois termes ! Il faut donc en utiliser une autre : un OU est zéro si les deux entrées sont à 0. On cherche donc le zéro maintenant...

Conclusion :
From the truth table we can obtain a Boolean expression in two distinct ways :
- F is equal to 1 if any term corresponding to a one is satisfied
- F is equal to 0 if any term corresponding to a zero is satisfied

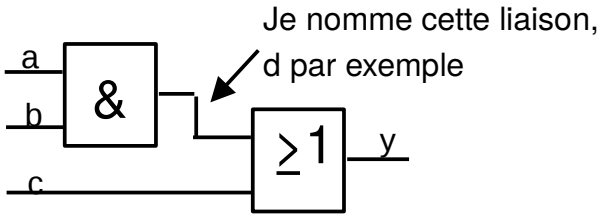
You can then use De Morgan's law (or not) to find out the corresponding Boolean expression.

TD2 ENSL1 : Du schéma aux équations

I) Assemblage des fonctions élémentaires

A partir des fonctions élémentaires présentées au TD 1, il est possible d'en construire des plus complexes, ayant par exemple 3 variables d'entrées... Une question vient alors à l'esprit, comment trouver la table de vérité correspondante ?

Exemple



c b a	d	y
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	1	1
...		
1 1 1	1	1

d = a.b
et
y = d + c
donnent
y = (a.b) + c

A partir d'un schéma il est donc très simple d'obtenir une table de vérité. On peut aussi obtenir une équation ce que l'on examine maintenant.

II) Obtention d'une équation algébrique à partir d'une table de vérité

Une table de vérité est constituée de deux parties : à gauche la partie SI, à droite la partie ALORS. Pour obtenir une équation à partir d'une table de vérité, il suffit :

- de chercher les 1 de la partie ALORS,
- pour chacun des 1 écrire un terme produit en s'aidant de la partie SI correspondante,
- séparer ces termes produits par des plus.

Exemple

SI		ALORS	[MINTERM]
b	a	y	
0	0	1	$\bar{b}\cdot\bar{a}$
0	1	0	
1	0	0	
1	1	1	$b\cdot a$

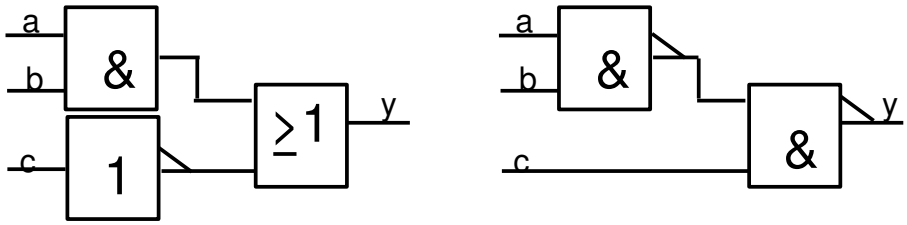
$$y=\bar{b}\cdot\bar{a}+b\cdot a$$

Il est facile de montrer que plusieurs schémas différents peuvent donner une même table de vérité.

Exercice 1

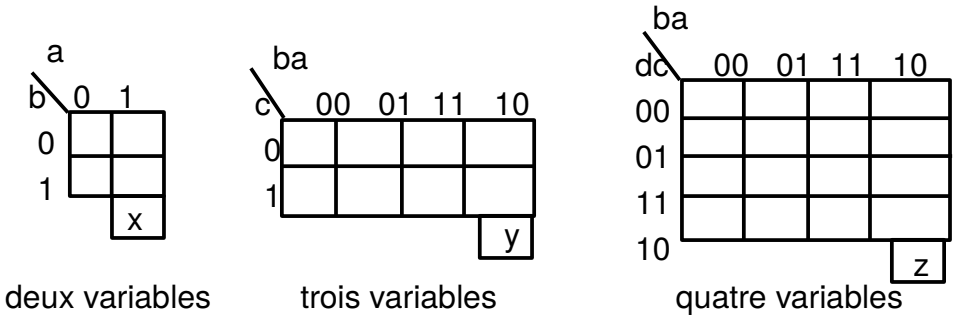
Verify if the following two combinational logic circuits have the same truth table ?

Then write a boolean expression for y in Figure below in terms of the inputs a, b, c.



III) Les tableaux de Karnaugh ([Karnaugh map])

Lorsque le nombre de variables d'entrées augmente l'écriture d'une table de vérité devient de plus en plus fastidieuse. Il existe une représentation plus synthétique que la table de vérité pour les fonctions à plusieurs variables c'est le tableau de Karnaugh :



deux variables trois variables quatre variables

IV) Primitives LUT4 en schématique Xilinx

Présentation des LUTs

Une table de vérité de trois entrées peut être représentée par un nombre 8 bits que l'on convertit en hexadécimal. Soit donc la table de vérité suivante (trois entrées notées e0, e1 et e2, une sortie notée s) :

e2	e1	e0	s	
0	0	0	0	b_0
0	0	1	1	b_1
0	1	0	1	b_2
0	1	1	0	b_3
1	0	0	1	b_4
1	0	1	0	b_5
1	1	0	1	b_6
1	1	1	0	b_7

Vous pouvez synthétiser la table de vérité à l'aide d'un seul nombre sur 8 bit (poids faible en haut) :

- en binaire ce nombre vaut : 0b01010110
- en hexadécimal ce nombre vaut : 0x56 (noté X"56" en VHDL)

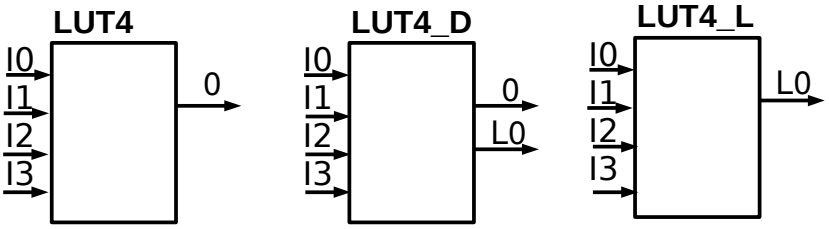
Aucune simplification à faire ici

La valeur hexadécimale 56 (notée X"56" en VHDL) est la valeur avec laquelle il faudra initialiser votre LUT avec un composant **lut3** ("01010110" en binaire est aussi admis).

Pour 4 entrées, on utilise une **LUT4** avec 4 chiffres hexadécimaux.

Pour initialiser une LUT, double cliquer dessus et remplir le champ INIT.

INFO	Les FPGA que l'on utilise sont essentiellement composés de LUT4. **La transformation table de vérité vers LUT4 ne demande aucun calcul booléen mais une transformation binaire vers hexadécimal.**

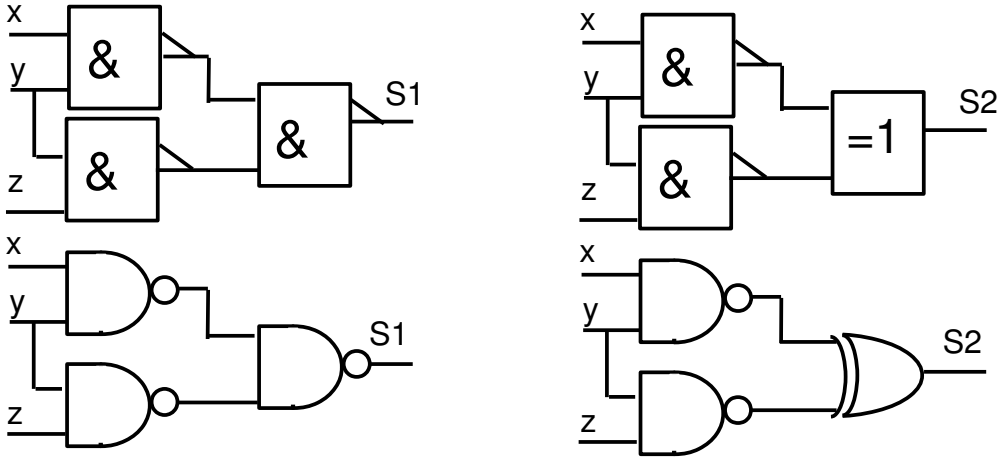


La sortie LO sert uniquement pour une connexion interne, d'où son nom : Local Output.

LUT1, LUT2, LUT3, LUT1_D, LUT2_D, LUT2_L, LUT3, LUT3_D et LUT3_L sont également disponibles.

Exercice 2

1) Construct the truth tables, plot the following functions on a 3-variable Karnaugh map (K-map) and write LUT3 description for S_1 and S_2.



2) Write the Boolean expressions for S1 and S2 in terms of the inputs x, y, z.
3) Obtain the two Boolean functions from the truth tables.
4) (**Optional**) Write VHDL programs for these Boolean functions.

Exercice 3

Verify the following identities by the truth table or the Karnaugh map method

a) $a+\bar{a}\cdot b=a+b$

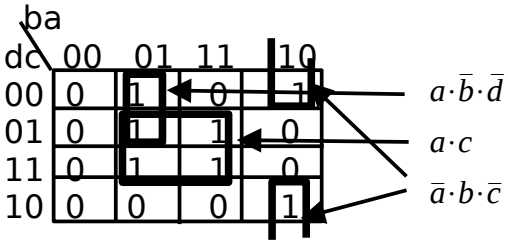
b) $(a+b)\cdot(a+\bar{b})=a$

TD3 ENSL1 : Simplification et implantation de formes disjonctives

I) Simplification par Karnaugh

Une équation obtenue à partir d'une table de vérité s'appelle une forme disjonctive ou somme de produits.
Les tableaux de Karnaugh permettent de simplifier ces formes disjonctives en regroupant des termes. Pour expliquer cela nous allons définir :
contiguïté booléenne : deux termes produits sont contigus s'ils sont constitués des mêmes variables booléennes et qu'une seule variable apparaît comme non complémentée dans un terme et complémentée dans l'autre,
contiguïté physique : deux cases voisines horizontalement ou verticalement d'un tableau de Karnaugh.

Exemple



On peut avoir des regroupements de 1, 2, 4, 8, 16 cases (puissance de 2)

Contiguïté physique = contiguïté booléenne sauf sur les bords (intérêt des tableaux de Karnaugh)

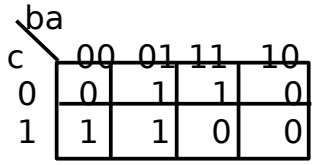
On trouve $y = a\cdot c + a\cdot\bar{b}\cdot\bar{d} + \bar{a}\cdot b\cdot\bar{c}$

Pour obtenir un terme à partir d'un regroupement, on se "ballade" dans le regroupement et on regarde toutes les variables qui changent : elles sont alors éliminées. L'objectif d'une simplification par tableaux de Karnaugh est de réaliser les regroupements les plus grands possibles et en nombre le plus petit possible.

La forme simplifiée obtenue à l'aide d'un tableau de Karnaugh est une forme disjonctive simplifiée. Celle obtenue à partir de la table de vérité est dite disjonctive canonique.
[en anglais : Sum-of-products (SOP), or disjunctive normal form (DNF)]

Exercice 1

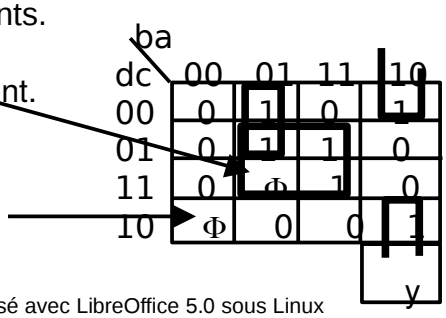
Find the minimum sum-of-product form.



Parfois il arrive que pour une raison quelconque, une ou plusieurs combinaisons des entrées ne peut en aucun cas arriver. Dans ce cas ce qui se passera en sortie n'a aucune importance : on dit que l'on a des cas indéterminé. Ils sont notés x ou X ou ϕ ou Φ. On les choisit alors leurs valeurs comme cela nous arrange pour faire les regroupements.

Ce Φ est pris à 1 car il est dans le regroupement.

Ce Φ est pris à 0 car il n'est pas dans un regroupement.



On trouve ainsi : $y = a \cdot c + a \cdot \bar{b} \cdot \bar{d} + \bar{a} \cdot b \cdot \bar{c}$

II) Implantation d'une forme disjonctive

Une forme disjonctive, qu'elle soit simplifiée ou pas, s'implante de manière naturelle en une structure ET-OU (les ET d'abord pour finir par les OU). Cette forme ET-OU conduit directement, en utilisant De Morgan, à un schéma en ET-NON (NAND). Prenons comme exemple la forme simplifiée du tableau de Karnaugh précédant.

$$y = a \cdot c + a \cdot \bar{b} \cdot \bar{d} + \bar{a} \cdot b \cdot \bar{c}$$

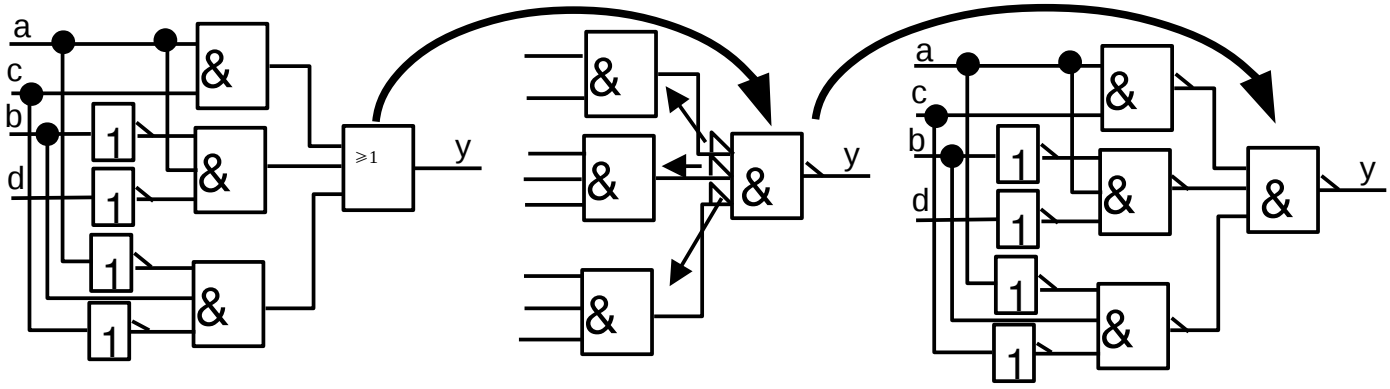


Schéma ET/OU (3 couches) — De Morgan sur le OU — Schéma en ET-NON (3 couches)

Pour obtenir un schéma en ET-NON (NAND) on part d'une forme disjonctive si possible simplifiée et on fait une schéma en trois couches ET/OU puis on transforme le OU final en ET-NON en faisant glisser les inverseurs de ses entrées. Le schéma obtenu est alors en trois couches mais utilise des portes à nombre d'entrées illimité.

Remarque : les seules synthèses qui nous intéressent cette année sont les synthèses ET/OU et les synthèses en ET-NON : ce sont les seules que l'on utilisera par la suite en TD et TP.

Exercice 2

Plot the following functions on a K-map and hence find the minimum sum-of-product representations.
Realize the resulting functions under the assumption that the only logic gates available are nands with 3 inputs (except y4).
Write the corresponding VHDL programs (structural description).

$y_1 = a \cdot (b + c)$
$y_2 = a \cdot b + \bar{c}$
$y_3 = \bar{a} \cdot b + a \cdot c$
$y_4 = \bar{a} \cdot b + a \cdot \bar{b}$

$y_5 = ((\bar{a}+b)\cdot(a+c)+b\cdot c)\cdot(a+b+c)$

$y_6 = \bar{a}\cdot b\cdot\bar{c}+a\cdot\bar{b}\cdot c$

Realize the expression

$F = \bar{x}_1\cdot[\bar{x}_2(\bar{x}_3\cdot x_4+x_3\cdot\bar{x}_4)+x_2\cdot(\bar{x}_3\cdot\bar{x}_4+x_3\cdot x_4)]+x_1\cdot[\bar{x}_2(\bar{x}_3\cdot\bar{x}_4+x_3\cdot x_4)+x_2\cdot(\bar{x}_3\cdot x_4+x_3\cdot\bar{x}_4)]$ as an AND/OR circuit . Hence obtain a 6 level NAND-gate realization.

III) Primitives schématiques Xilinx

A deux entrées : and2, or2, nand2, nor2 et xor2 (I0, I1, O)

A deux entrées : and2b1, or2b1, nand2b1 et nor2b1 (not I0, I1, O)

A trois entrées : and3, or3, nand3, nor3 et xor3 (I0, I1, I2, O)

A trois entrées : and3b1, or3b1, nand3b1 et nor3b1 (not I0, I1, I2, O)

A trois entrées : and3b2, or3b2, nand3b2 et nor3b2 (not I0,not I1, I2, O)

et on poursuit jusqu'à 5 variables... puis jusqu'à 9 mais sans les inversions sur les entrées.

Remarque : and3b3 existe, ainsi que and5b5.

Exercice 3

Reprendre l'exemple de la page précédente et en faire un schéma avec les primitives de Xilinx.

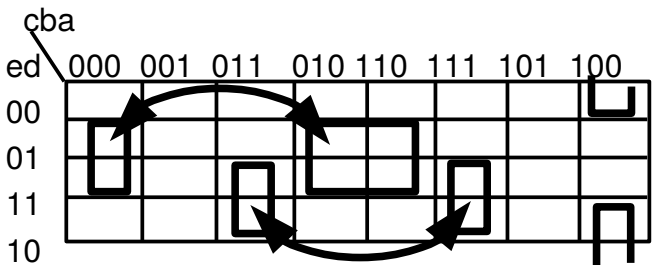
TD4 ENSL1 : Compléments sur la simplification

I) Simplification par Karnaugh

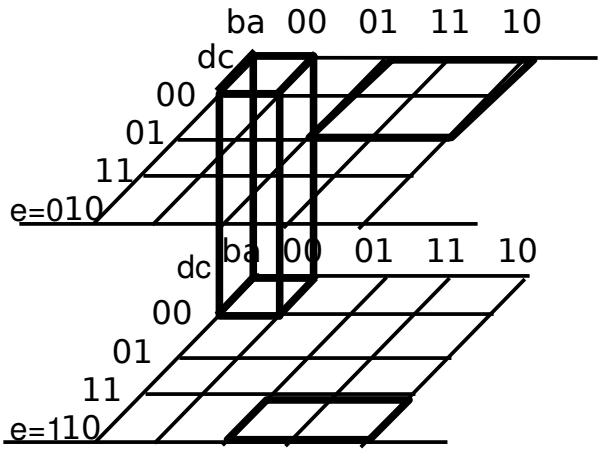
Les tableaux de Karnaugh permettent d'obtenir les formes disjonctives ou conjonctives simplifiées en regroupant des termes (représentés par des 0). Cela fonctionne très bien jusqu'à 4 variables. Que se passe-t-il après ?

De la difficulté de garder la contiguïté booléenne proche de la contiguïté physique

Regardons ce qui se passe pour le cas de cinq variables :



Si l'on réalise un tableau de Karnaugh à 5 variables on remarque qu'à côté des regroupements traditionnels il en existe qui ne sont plus contigus (flèche sur le dessin). Une technique plus intéressante consiste à réaliser un tableau de Karnaugh dans l'espace.



Il s'agit de réaliser deux tableaux de karnaugh, un pour la cinquième variable e = 0 et un pour e = 1. On regroupe ensuite dans l'espace, sur un ou deux tableaux.
Avec six variables on aura quatre tableaux de Karnaugh plats à superposer (et pas dans n'importe quel ordre).

Pour éviter cela on utilise la simplification algébrique.

II) Simplification algébrique

La technique pour obtenir une forme disjonctive simplifiée consiste à effectuer les parenthèses et ensuite à appliquer une des règles suivantes :

Élément neutre	$a \cdot 1 = a$	$a + 0 = a$
Élément absorbant	$a \cdot 0 = 0$	$a + 1 = 1$
Idempotence	$a \cdot a = a$	$a + a = a$
Complément	$a \cdot \bar{a} = 0$	$a + \bar{a} = 1$

Commutativité	$a\cdot b=b\cdot a$	$a+b=b+a$
Associativité	$a\cdot(b\cdot c)=(a\cdot b)\cdot c$	$a+(b+c)=(a+b)+c$
Distributivité	$a\cdot(b+c)=a\cdot b+a\cdot c$	$a+(b\cdot c)=(a+b)\cdot(a+c)$
Relations diverses	$a\cdot(a+b)=a$ $a\cdot(\bar{a}+b)=a\cdot b$ $a\cdot(\overline{a+b})=0$ $a\cdot(\overline{a\cdot b})=a\cdot\bar{b}$	$a+(a\cdot b)=a$ $a+(\bar{a}\cdot b)=a+b$ $a+(\overline{a\cdot b})=1$ $a+(\overline{a+b})=a+\bar{b}$
De Morgan	$\overline{a\cdot b}=\bar{a}+\bar{b}$ $a\cdot b=\overline{\bar{a}+\bar{b}}$	$\overline{a+b}=\bar{a}\cdot\bar{b}$ $a+b=\overline{\bar{a}\cdot\bar{b}}$
Fonction Biforme	$a\cdot b+\bar{a}\cdot c=(a+c)\cdot(\bar{a}+b)$	$(a+c)\cdot(\bar{a}+b)=a\cdot b+\bar{a}\cdot c$
Consensus	$a\cdot b+\bar{a}\cdot c+b\cdot c=a\cdot b+\bar{a}\cdot c$	$(a+b)\cdot(\bar{a}+c)\cdot(b+c)=(a+b)\cdot(\bar{a}+c)$
Consensus généralisé	$a\cdot b+\bar{a}\cdot c+b\cdot c\cdot d=a\cdot b+\bar{a}\cdot c$	$(a+b)\cdot(\bar{a}+c)\cdot(b+c+d)=(a+b)\cdot(\bar{a}+c)$

Dualité

Si l'on cherche une forme disjonctive simplifiée on peut procéder de la manière suivante :
- on effectue d'abord toutes les parenthèses pour trouver une forme disjonctive,
- on regroupe tous les termes qui ne diffèrent que d'une variable (combinaison par contiguïté booléenne) en ajoutant ces termes simplifiés,
- on retire ensuite tous les termes qui sont inclus dans d'autres termes,
- on cherche les simplifications par consensus.

Les combinaisons peuvent se faire plusieurs fois.

Les combinaisons peuvent se faire plusieurs fois.
Exemple de combinaison : $a\cdot\bar{b}\cdot c+\bar{a}\cdot\bar{b}\cdot c=\bar{b}\cdot c$
Exemple d'inclusion $a\cdot\bar{b}\cdot c\subset\bar{b}\cdot c$ qui donne : $a\cdot\bar{b}\cdot c+\bar{b}\cdot c=\bar{b}\cdot c$

Exercice 1

Use the algebraic method to simplify the following functions and hence obtain a realization of each function which uses AND, OR and INVERTER gates only :

$y_1=\bar{a}\cdot b\cdot c+a\cdot c+(a+b)\cdot\bar{c}$
$y_2=b\cdot c+a\cdot c+a\cdot b+b$
$y_3=(a\cdot\bar{b}+c)\cdot(a+\bar{b})\cdot c$
$y_4=(a\cdot c+b\cdot\bar{c})\cdot(a+\bar{c})\cdot b$
$y_5=(\bar{a}\cdot b+a\cdot\bar{b})\cdot(a\cdot b+\bar{a}\cdot\bar{b})$

Pour le fun, preuve algébrique du consensus

consensus : $f=a\cdot b+\bar{a}\cdot c+b\cdot c=a\cdot b+\bar{a}\cdot c+b\cdot c\cdot(a+\bar{a})=a\cdot b+\bar{a}\cdot c+b\cdot c\cdot a+b\cdot c\cdot\bar{a}$
et inclusion 3 dans 1 et 4 dans 2.

III) Factorisations pour retirer une variable

Dans les FPGA un peu anciens, on dispose de LUT4 mais dans les plus modernes de LUT6. Notre problème va donc être de trouver des techniques pour faire disparaître une variable pour pouvoir entrer dans les LUTs.

Une technique consiste à mettre en facteur une variable et son complément et à regarder si les deux facteurs sont identiques. Il faut pour cela transformer ces facteurs en forme canonique.

Exemple simple

Soit la forme algébrique à trois variables

$f = a \cdot b + \bar{a} \cdot b \cdot c + \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c} + a \cdot c$ à écrire sous la forme $f = a \cdot CF_1 + \bar{a} \cdot CF_2$ Si CF1=CF2 (CF pour Canonic Form) alors a disparait.

Si l'on factorise a on trouve

$f = a \cdot (b + c) + \bar{a} \cdot b \cdot c + \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c} = a \cdot b \cdot c + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c} + \bar{a} \cdot b \cdot c + \bar{a} \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c}$

où l'on voit que les deux facteurs sont identiques et donc que a disparaît.

Exercice 2

1°) Try to remove the "a" variable in the following Boolean expressions using theorems and identities :

$A = a \cdot b \cdot \bar{d} + a \cdot d + \bar{a} \cdot b + \bar{a} \cdot d$

$B = \bar{c} \cdot \bar{d} + \bar{a} \cdot \bar{b} + c \cdot \bar{d} + a \cdot \bar{b}$

2°) Xilinx states that the XC4005 Combinational Logic Bloc (CLB) F and G look up tables can perform any 4 input logic functions, and H box can perform any 3 input function. Prove that using F, G and H look up tables, you can calculate ANY function of 5 variables.

Hint : every function can be written as $f = a \cdot CF_1 + \bar{a} \cdot CF_2$

See also : Product-of-sums (POS), or conjunctive normal form (CNF)

TD5 ENSL1 : Sorties multiples, addition, codes

Les simplifications dans le cas des sorties multiples sont encore plus complexes. Nous n'allons pas aborder ce problème de front mais plutôt par des exemples choisis.

I) Représentation des nombres

1°) Nombres positifs

Ce système créé par Leibnitz (17e s) utilise la base 2 donc l'ensemble des symboles utilisés est Bb={0,1}.
Un nombre entier est représenté en format fixe par l chiffres dans sa base b=2. Il s'écrit donc :

$Nb = a_n a_{n-1} ..... a_1 a_0$ avec l = n+1. avec a_i = 0 ou 1

La quantité de nombres de l chiffres qu'il est possible de représenter s'appelle la capacité de représentation. Si l'on remarque que le plus grand nombre représenté peut prendre la valeur :

$N_{10max} = 2^l - 1$,

la capacité C devient :

$C = N_{10max}$ + représentation du zéro soit $C=N_{max}+1 = 2^l$

La formule suivante

$N_{10} = a_n.2^n + a_{n-1}.2^{n-1}+.....+a_0.2^0$ (poids 1, 2, 4, 8, 16, ...)

exprime la signification de l'écriture binaire et permet un calcul des valeurs binaires en décimal.
Nombres fractionnaires (n+1 chiffres avant la virgule, m chiffres après la virgule) :

$N_{10} = a_n.2^n + a_{n-1}.2^{n-1}+.....+a_0.2^0 + a_{-1}.2^{-1} + ... + a_{-m} .2^{-m}$

Par exemple si n = 2 et m = 2 :

$N_2=(100,11)_2=(1.2^2+0.2^1+0.2^0+1.2^{-1}+1.2^{-2})_2$

2°) Nombres entiers positifs et négatifs (nombres relatifs)

Nous présentons maintenant la représentation par le complément vrai de N (complément à deux).

La représentation des nombres positifs est inchangée, seule celle des nombres négatifs doit être présentée. Si l'on désire représenter un nombre négatif, on part de sa valeur absolue en binaire, on complémente bit à bit et on ajoute 1.
Le premier bit reste toujours le bit de signe.

Ainsi dans un système utilisant l = n+1 chiffres binaires, on représentera les nombres sur l'intervalle $[-2^n, 2^{n-1}]$. Sur 8 bits cela donne tout calculs faits : [-128, +127].

3°) Overflow

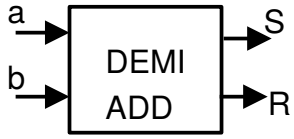
La somme de deux nombres positifs doit donner un nombre positif, la somme de deux nombres négatifs doit donner un nombre négatif. Si ce n'est pas le cas on a un overflow (dépassement de capacité). Avant de passer aux opérations arithmétiques nous donnons un comparatif avec d'autres représentations.

Exemples de représentation

Valeur algébrique décimale	Représentation des nombres par :		
	Bit de signe et valeur absolue	Complément restreint	Complément vrai
-127	11111111	10000000	10000001
-105	11101001	10010110	10010111
-15	10001111	11110000	11110001
-3	10000011	11111100	11111101
-2	10000010	11111101	11111110
-1	10000001	11111110	11111111
0	10000000 00000000	11111111 00000000	00000000
+1	00000001	00000001	00000001
+127	01111111	01111111	01111111

II) Du demi-additionneur à l'additionneur 1 bit

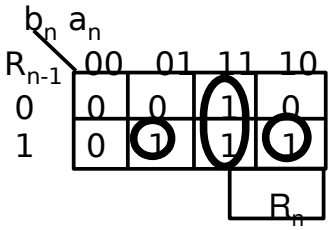
Les opérations d'addition sur 1 bits sont très simples et donnent un résultat sur 2 bits. Le bit de poids fort est appelé R (pour retenue= carry en anglais) et le poids faible est appelé S (pour somme).

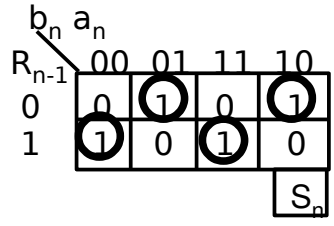


b	a	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = a \oplus b$

$R = a.b$





$S_n = a_n \oplus b_n \oplus R_{n-1}$

$R_n = a_n.b_n + R_{n-1}.(a_n \oplus b_n)$

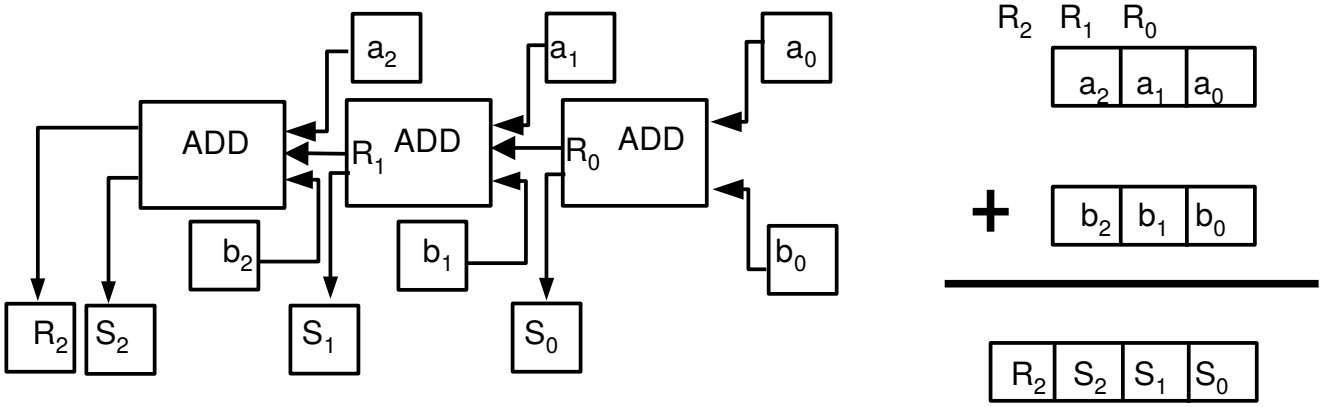
Remarquez que R_n n'est pas simplifié au mieux pour faire apparaître un OU exclusif.

Exercice 1

Realize the adder circuit with XOR and NAND gates.
Write the truth table for a one-bit full subtractor whitch has inputs A, B, BorrowIn, and outputs D and BorrowOut. Implement logic for D and BorrowOut using XOR and NAND gates.
Realize a circuit which can select between adder and soustractor

II) De l'addition à 1 bit à l'addition à n bits (cascader, mise en cascade)

Voici un schéma de principe qui peut facilement être généralisé sur 8 ou plus bits.



Exercice 2

1) Quel est le temps de propagation de l'additionneur 3 bits si un additionneur a lui-même un temps de propagation t_p=10 ns.

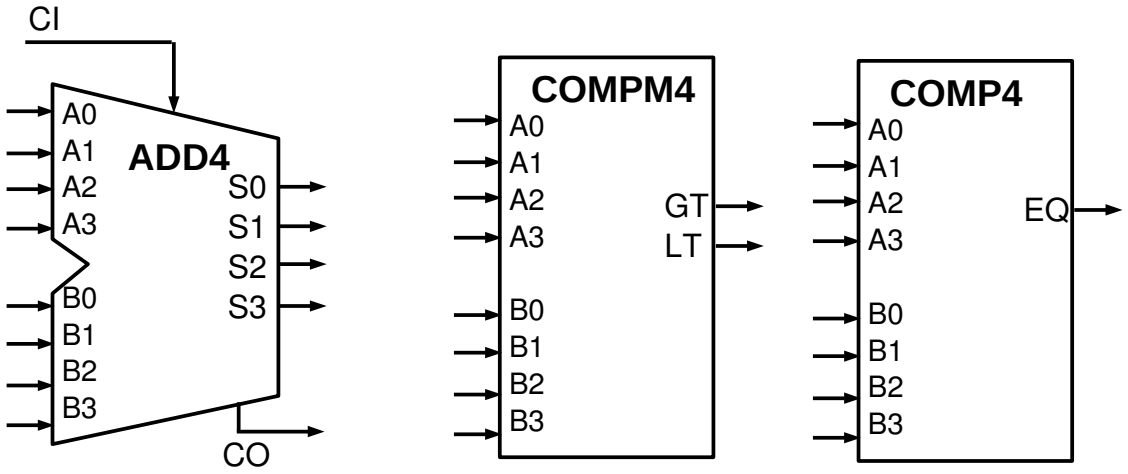
2) Design a two 4-bit number adder.

Terminons par un exemple qui montre comment une soustraction peut se transformer en une addition si l'on utilise la représentation en complément à deux.

100	01100100	01100100	
-75	-01001011	+10110101	
=25	00011001	1 00011001	lecture directe du résultat
50	00110010	00110010	
-75	-01001011	+10110101	
-25	11100111	11100111	le résultat est en complément à deux : 256-231=25

III) Arithmétique et comparaison en schématique Xilinx

Voici quelques exemples de composants Xilinx :



IV) Arithmétique et VHDL

L'additionneur ci-dessus peut être facilement décrit en VHDL :

```
-- addition sur 8 bits
ENTITY add3 IS
PORT (a,b :IN INTEGER RANGE 0 TO 7;
      s :OUT INTEGER RANGE 0 TO 15);
END add3;

ARCHITECTURE add OF add3 IS
BEGIN
  s <= a + b; -- ne pas confondre + avec OR
END add;
```

V) Codes et transcodage

1°) Code Gray

Le code Gray a déjà été utilisé sur deux bits dans les tableaux de Karnaugh. On vous montrera en cours comment obtenir un code Gray par symétrie pour un nombre de bits quelconque.
Le bits B_i désignent le binaire tandis que les bits G_i désignent le code GRAY.

n	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Remarque

Les entitées en VHDL pour entrées et sorties multiples sont plutôt déclarées sous la forme :

```
ENTITY binGray IS
PORT (b :IN BIT_VECTOR(3 DOWNTO 0);
      g :OUT BIT_VECTOR(3 DOWNTO 0));
END binGray;
```

Exercice 3

A combinational circuit is required to act as an encoder to convert binary code into Gray code. Determine minimum sum-of-products expressions for the output functions G_3, G_2, G_1, G_0. Realize the resulting functions under the assumption that the only logic gates available are XOR gates with 2 inputs.
If LUT4s are used, what are the hexadecimal value to initialize them.

2°) Les codes décimaux

Les codes décimaux sont des codes pour les dix premières valeurs binaires. Voici un code appelé Excess 3 par exemple :

n	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Exercice 4

Un décodeur Excess 3 est un circuit qui prend comme entrée le code 4 bits $E_3E_2E_1E_0$ et valide une sortie parmi les 10 (indice n de la table ci-contre). On évitera de confondre ce circuit avec un transcodeur BCD Excess 3.
Étudier un tel circuit, l'implanter.
Écrire le programme VHDL correspondant.

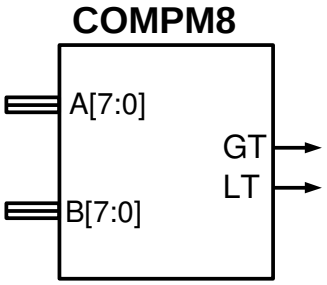
Exercice 5

Trouver la table de vérité d'un transcodeur BCD vers code GRAY (c'est à dire un code GRAY cyclique sur 10 valeurs seulement).

Exercice 6

Si on effectue l'addition de 2 nombres signés N1 et N2 de signes respectifs s1 et s2, le résultat possède un signe s3. A partir des états 0 ou 1 des 3 variables s1,s2 et s3, construire la valeur de l'overflow V = f(s1,s2,s3). Donner les équations de V. Proposer un schéma à NANDs.

Composants Xilinx



TD6 ENSL1 : DS Année 2014

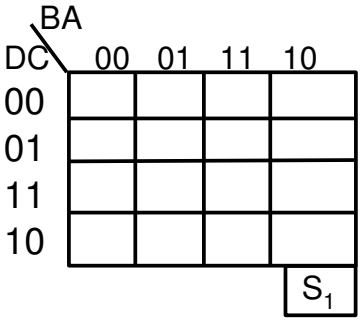
Exercice 1

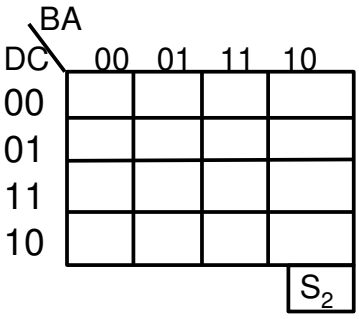
On donne les expressions

$S_1=(\bar{A}\cdot B\cdot C\cdot\bar{D})+(\bar{D}\cdot C\cdot\bar{A})+(\bar{D}\cdot A)$

$S_2=\overline{(A+\bar{B}+\bar{C})\cdot(D+\bar{B})\cdot(\bar{D}+\bar{C})}$

Remplir les tableaux de Karnaugh correspondants en indiquant les regroupements





Donner les formes simplifiées disjonctives correspondantes

Réponse :

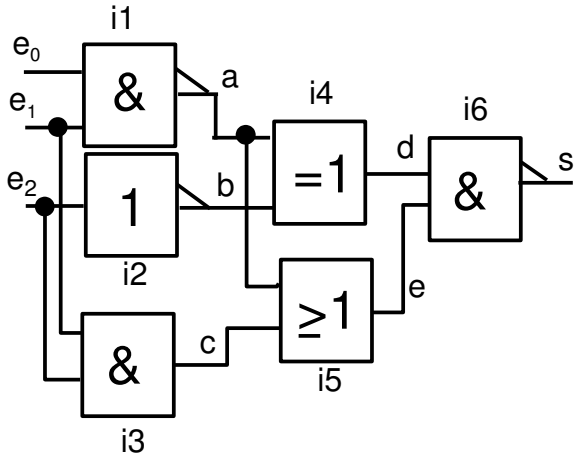
S_1 =

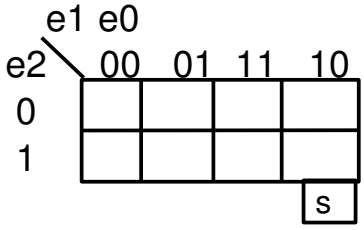
Réponse :

S_2 =

Exercice 2

On donne le schéma ci-dessous.





1°) Remplir le tableau de Karnaugh ci-dessus pour s=f(e0,e1,e2) en vous aidant de la table de vérité ci-dessous. On ajoutera les regroupements pour simplification.

e2	e1	e0	a	b	c	d	e	s
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

Feuille de réponse n°2

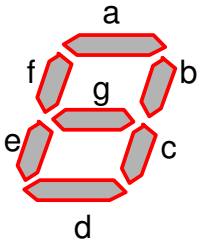
2°) En déduire la forme disjonctive simplifiée
Réponse :

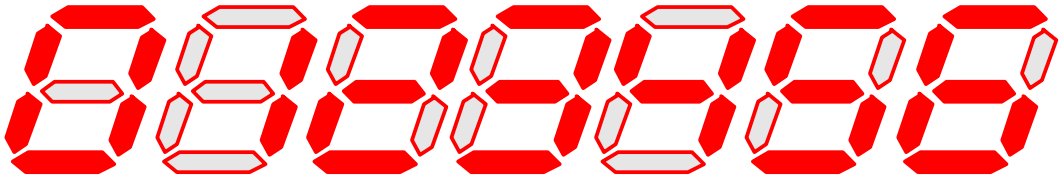
3°) En déduire un schéma en ET-OU avec des ET et des OU à nombre d'entrées illimité.
Réponse :

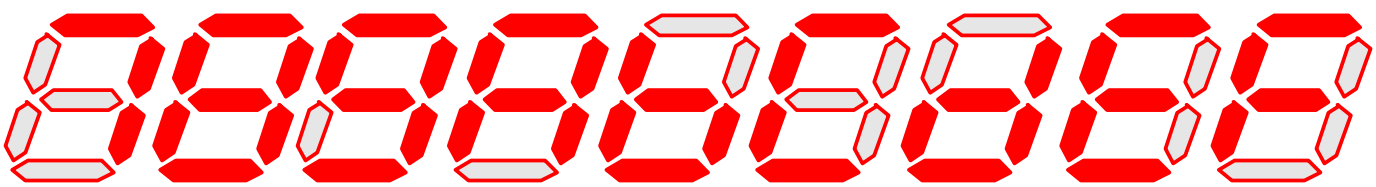
4°) Nous allons réaliser un montage équivalent à l'aide d'une LUT3 (LUT à trois entrées). En déduire la valeur hexadécimale d'initialisation de la LUT3 correspondante.
Réponse :

Exercice 3

On désire réaliser un transcodeur binaire sept segments comme en TP. On s'intéresse seulement au segment a. D'autre part, contrairement au TP, on doit réaliser un 1 pour allumer le segment correspondant.







NOM :
Prénom :
Groupe : Feuille de réponse n°3

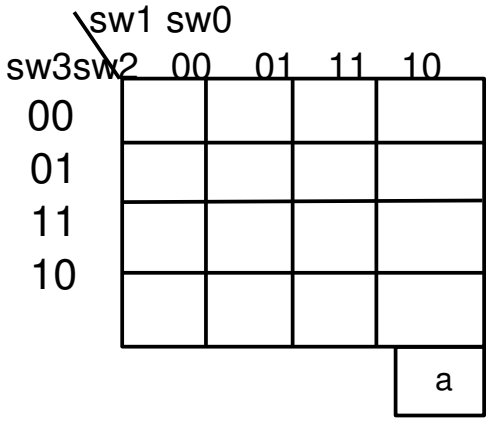
1°) Remplir la table de vérité ci-dessous.

Réponse :

sw3	sw2	sw1	sw0	a
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

2°) Combien de termes la forme disjonctive non simplifiée (canonique) contient-elle ?
Réponse :

3°) Remplir le tableau de Karnaugh ci-contre pour le segment "a".

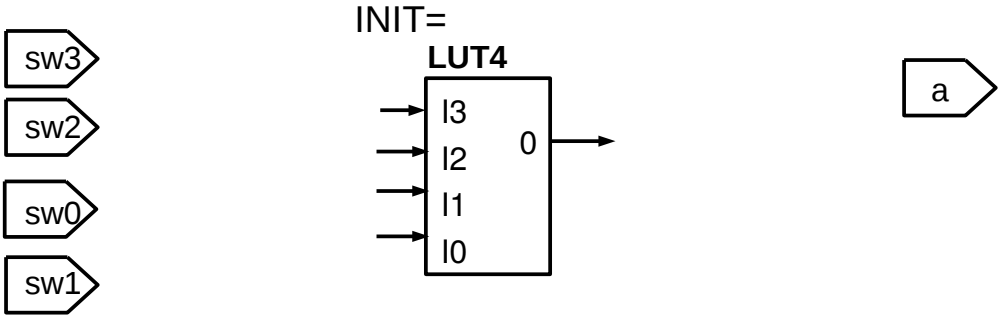


4°) La simplification complète étant délicate, on vous demande de trouver les 4 regroupements de 4 "1" de ce tableau de Karnaugh en les dessinant. Vous les exprimerez ensuite sous forme de terme produit ci-dessous

Réponse :

Feuille de réponse n°4

5°) Proposer un schéma de ce circuit en utilisant une LUT4 en précisant son initialisation et en essayant de respecter le poids faible de la table de vérité.
Réponse :



Exercice 4 (Simplification algébrique)

1°) L'expression $y_1=(\bar{A}\cdot B\cdot C\cdot\bar{D})+(\bar{A}\cdot\bar{B}\cdot C\cdot\bar{D})+(\bar{D}\cdot A)$ se simplifie-t-elle par combinaison ? Si oui, simplifiez-la.
Réponse :

2°) Est-il possible de pousser la simplification plus loin ? Si oui donnez la nouvelle forme simplifiée.
Réponse :

3°) L'expression $y_2=(\bar{A}\cdot B\cdot C\cdot\bar{D})+(\bar{A}\cdot B\cdot\bar{D})+(\bar{D}\cdot\bar{A})$ se simplifie-t-elle par inclusion ? Si oui, simplifiez-la.
Réponse :

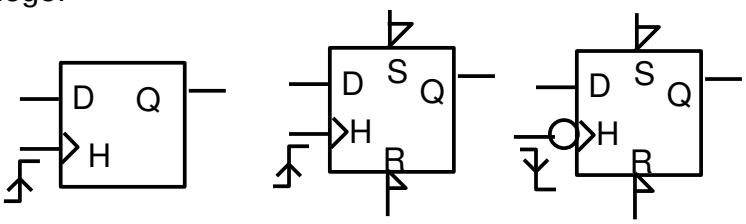
4°) L'expression $y_3=(\bar{A}\cdot B\cdot C\cdot\bar{D})+(\bar{D}\cdot C\cdot A)+(\bar{D}\cdot C\cdot B)$ contient-elle un consensus ? Si oui, simplifiez-la.
Réponse :

TD7 ENSL1 : Les bascules D et diagrammes d'évolutions

(WIKI : http://fr.wikiversity.org/wiki/Logique_s%C3%A9quentielle)

I) Bascules D

L'équation de récurrence de la bascule D n'a pas beaucoup d'intérêt, il faut mieux retenir son fonctionnement : recopie son entrée lors de l'arrivée d'un front d'horloge. On peut ajouter à cette entrée D synchrone (prise en compte sur front d'horloge) des entrées asynchrones (qui ne nécessitent aucune horloge.



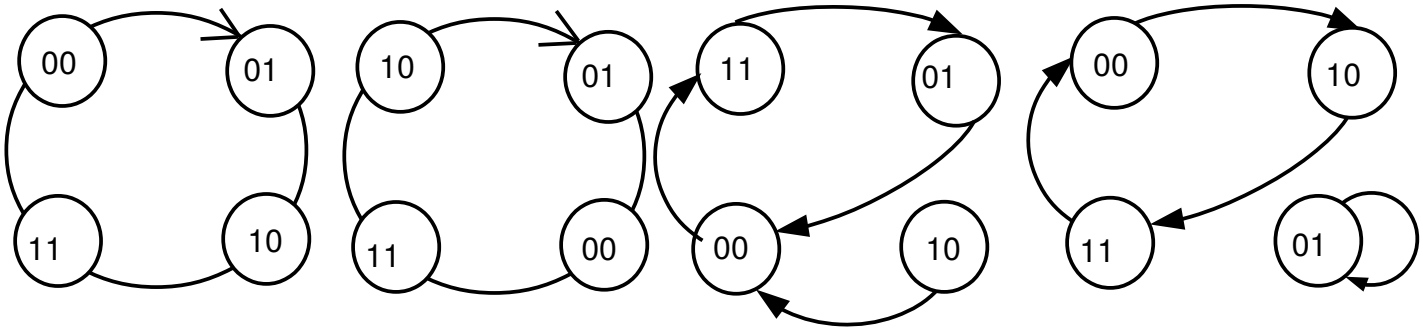
Le programme VHDL correspondant est présenté maintenant.

```
ENTITY bascule_D IS
PORT(d,horl,set,reset : IN BIT;
     s : OUT BIT);
END bascule_D;
```

```
ARCHITECTURE D_bascule OF bascule_D IS
BEGIN
  PROCESS(horl,set,reset)
  BEGIN
    IF (reset='1') THEN
      s<='0';
    ELSIF (set='1') THEN
      s<='1';
    ELSIF (horl'EVENT and horl='1') THEN
      s<=d;
    END IF;
  END PROCESS;
END D_bascule;
```

II) Diagrammes d'évolutions

Les montages séquentiels simples sont en général représenté par un diagramme d'évolution. Il s'agit d'un ensemble d'états (cercles) reliés entre eux par des flèches.



Remarque : le dernier diagramme d'évolution avec un état isolé est à éviter (Hang-Up State).

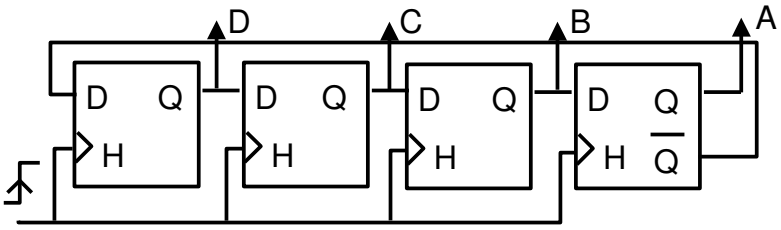
Les diagrammes d'évolutions peuvent être aussi variés que ceux présentés ci-dessus. Ils peuvent avoir un ou plusieurs cycles. La suite des états n'est pas forcément dans l'ordre naturel (du comptage).

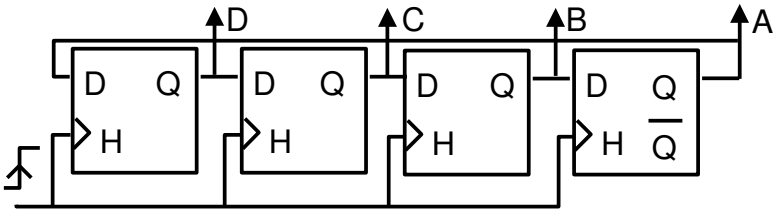
INFO

Si l'on veut trouver un diagramme d'évolution à partir d'un schéma utilisant des bascules D, il faut positionner les sorties des bascules D (qui constitueront l'état présent), puis chercher ce qui en résultera sur les entrées de ces bascules (qui constitueront l'état futur). En répétant ce travail pour chacune des possibilités en entrées on trouvera le diagramme d'évolution.

Exercice 1

Find the state diagrams of the circuits shown below.

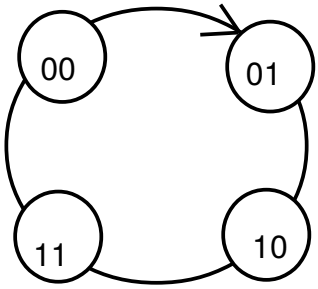




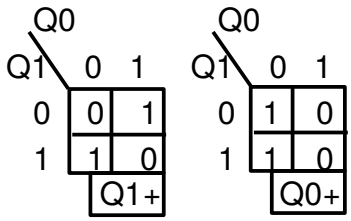
n flip-flops give 2^n states.

III) Des diagrammes d'évolutions aux équations de récurrences

Il est facile de construire une table des transitions (état présent ; état futur) à partir d'un diagramme d'évolution. Cela constitue tout simplement la table de vérité de l'équation de récurrence cherchée. Si on veut une forme simplifiée il faudra utiliser un ou plusieurs tableaux de Karnaugh. Par exemple pour le premier diagramme d'évolution donné en haut de cette page, on trouve :



État présent		État futur	
Q1	Q0	$Q1^+$	$Q0^+$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Équations de récurrence :

$Q_1^+ = Q_1 \oplus Q_0$

Q0+ = /Q0

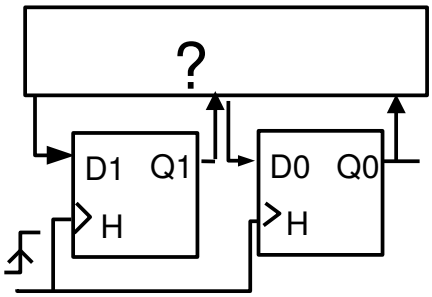
Exercice 2

Trouver les équations de récurrence de chacun des diagrammes d'évolution présentés au début de ce TD.

TD8 ENSL1 : Implantation en D

I) Implantations à l'aide de bascules D

Si l'on connaît les équations de récurrence, il est facile d'obtenir un schéma à l'aide de bascules D. Il suffit d'implanter les équations de récurrence dans le rectangle ci-dessous. Si on ne les connaît pas il suffit de les chercher (voir TD7).



Exercice 1

Design a modulo 8 counter using D flip-flops.
Write a VHDL program with CASE style.

Cours : division des fréquences et binaire

Exercice 2

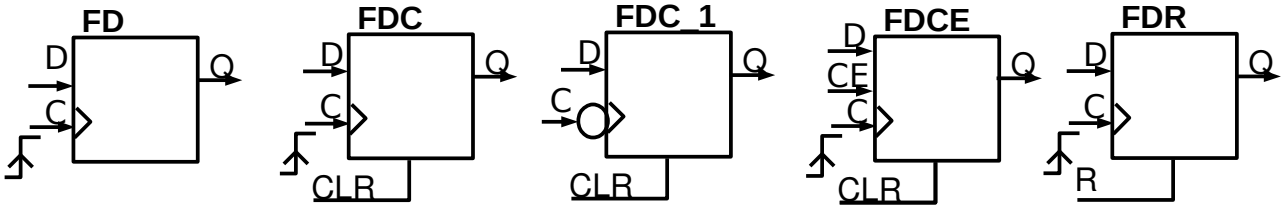
Réaliser la synthèse d'un diviseur de fréquence par trois avec des bascules D. Donner le programme VHDL correspondant.

Exercice 3

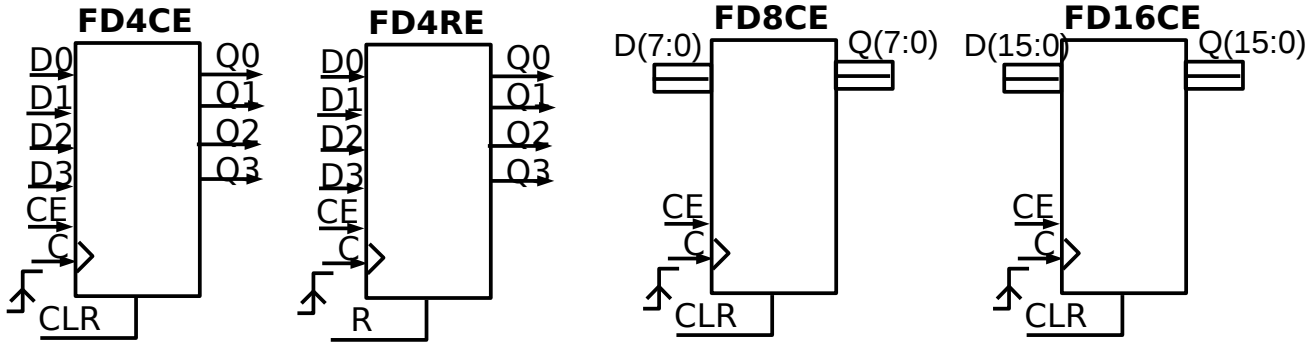
Réaliser un générateur de signaux carrés déphasés de 90°.

II) Primitives schématiques Xilinx

Sur un bit



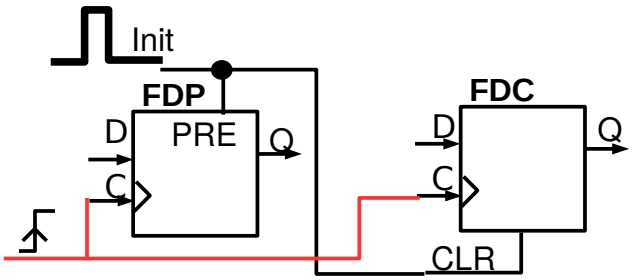
Sur plusieurs bits



Synchronous reset

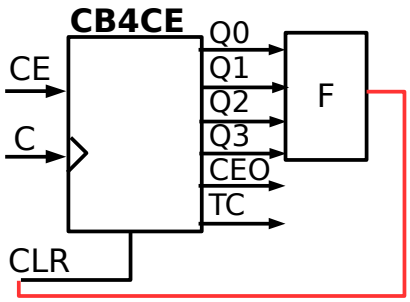
I) Initialisation

Lors de la mise sous tension les bascules en circuits intégrés sont initialisés à 0 (les circuits programmables à 1). Pour être sûr de l'initialisation il faut utiliser les entrées asynchrones, par exemple comme cela est montré à la figure ci-contre.



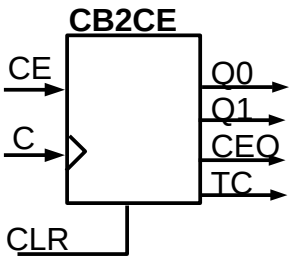
II) Forçage asynchrone

Il est parfois plus simple de transformer un montage synchrone en un autre montage par bouclage (asynchrone)
Ce type de montage se rencontre en pratique même s'il est déconseillé à utiliser.
Exemple : 0 ->11 pour des heures



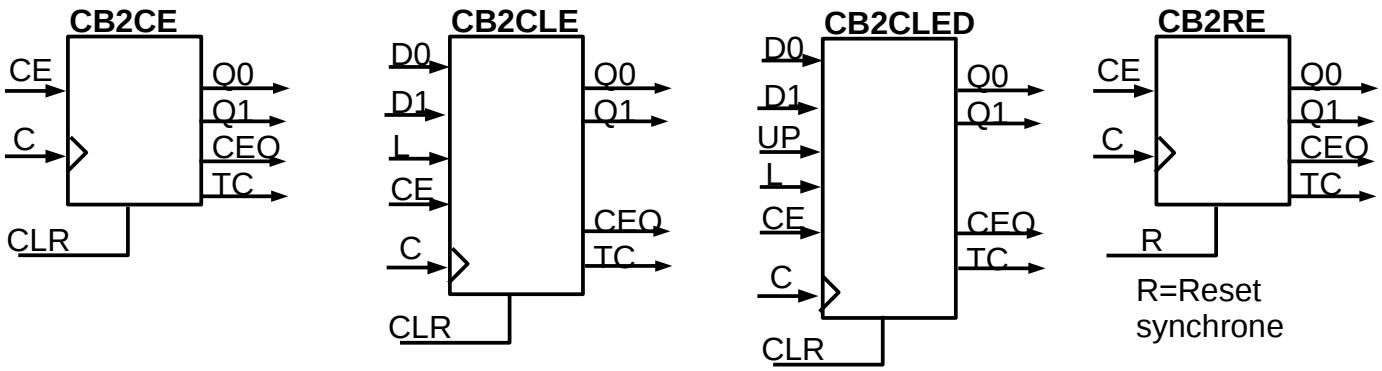
IV) Étude de compteurs

CB2CE : 2-Bit Cascadable Binary Counter with Clock Enable and Asynchronous Clear



This design element is an asynchronously clearable, cascadable binary counter. The asynchronous clear (CLR) input, when High, overrides all other inputs and forces the Q outputs, terminal count (TC), and clock enable out (CEO) to logic level zero, independent of clock transitions. The Q outputs increment when the clock enable input (CE) is High during the Low-to-High clock (C) transition. The counter ignores clock transitions when CE is Low. The TC output is High when all Q outputs are High.
Create larger counters by connecting the CEO output of each stage to the CE input of the next stage and connecting the C and CLR inputs in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than *n* (*t*CE-TC), where *n* is the number of stages and the time *t*CE-TC is the CE-to-TC propagation delay of each stage. When cascading counters, use the CEO output if the counter uses the CE input or use the TC output if it does not.
This counter is asynchronously cleared, outputs Low, when power is applied. For FPGA devices, power-on conditions are simulated when global set/reset (GSR) is active. GSR defaults to active-High but can be inverted by adding an inverter in front of the GSR input of the appropriate STARTUP_*architecture* symbol.

Il existe aussi des CB4CE, CB8CE et CB16CE. La série CB2CLE possède une entrée de chargement "L" et deux entrées "D0" et "D1". Il existe aussi CB4CLE, CB8CLE et CB16CLE. CB2CLED ajoute encore une entrée UP pour choisir comptage ou décomptage.



Exercice 1

1°) Réaliser un compteur décimal en utilisant la technique du forçage asynchrone à l'aide d'un CB4CE. Les sorties sont encore individuelles sur 4 bits et s'appellent Q0 à Q3. Comment le rendre cascadable ?
Comparer au schéma extrait du DS (2013) donné plus loin.
2°) Le compteur **CB4RE** est à reset synchrone. La priorité de R sur CE nécessite une modification du schéma de la question 1. Une **LUT4** et une porte ET entre la sortie de la LUT et CE suffisent-elles ?

Exercice 2

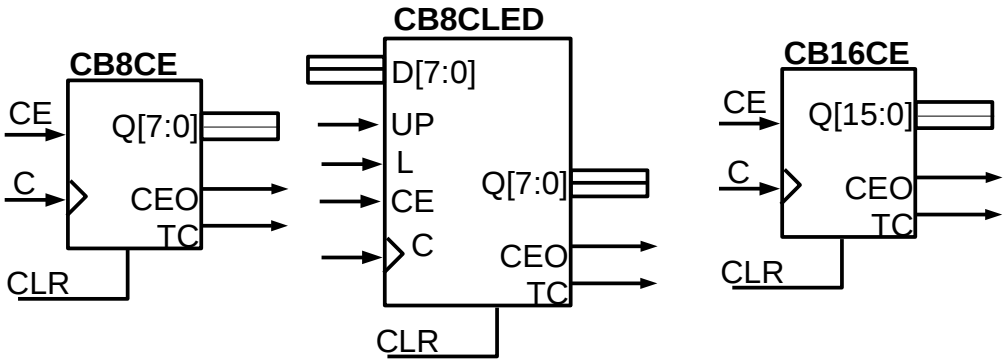
Using 3 type D flip-flops and extra logic, design a 3 bit counter which counts in the sequence $Q_2Q_1Q_0$ = 0,1,3,2,6,7,5,4,0, ... Show state diagram, state transition table and schematic diagram.

Exercice 3

Réaliser un générateur de signaux triphasés (120°). Le rapport cyclique sera 0,5. Montrer qu'il faut nécessairement 6 états pour le réaliser. Que faire des 2 états restants ? Écrire le programme VHDL correspondant.

Documentation schématique Xilinx : quelques compteurs

Les quelques compteurs présentés plus haut peuvent avoir des sorties sur 8 ou 16 bits.



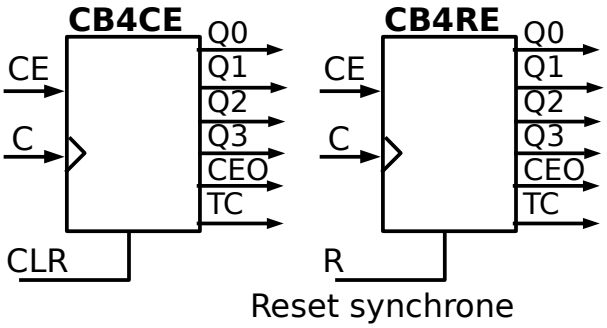
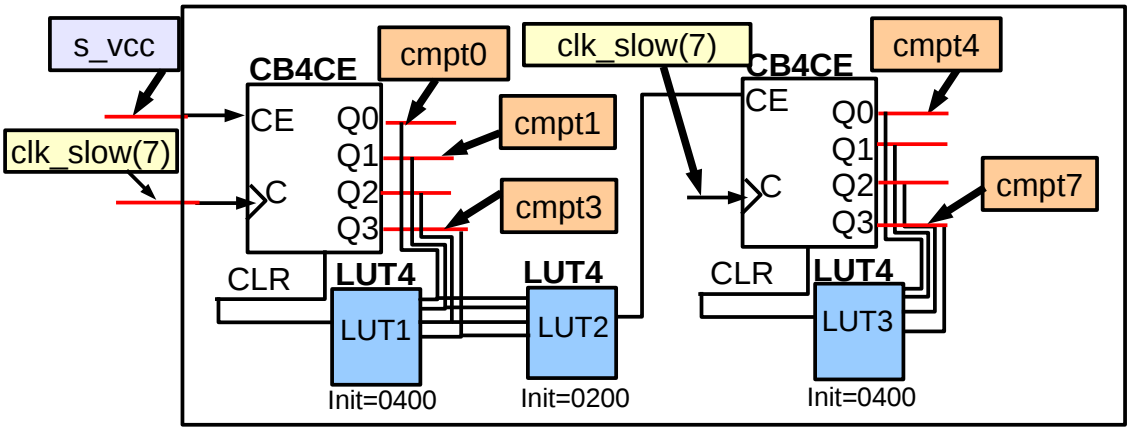


Schéma du DS 2013 :

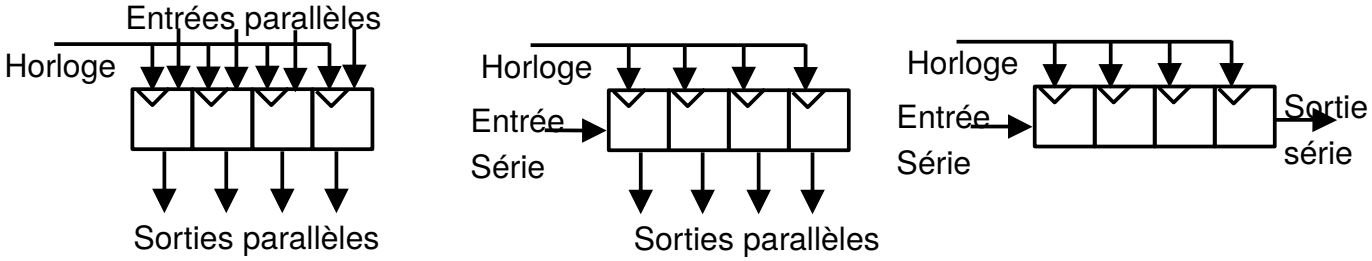
On suppose que les LUT4s sont câblées en respectant les poids sur le schéma suivant.



TD10 ENSL1 : Registres

I) Structure de base des registres

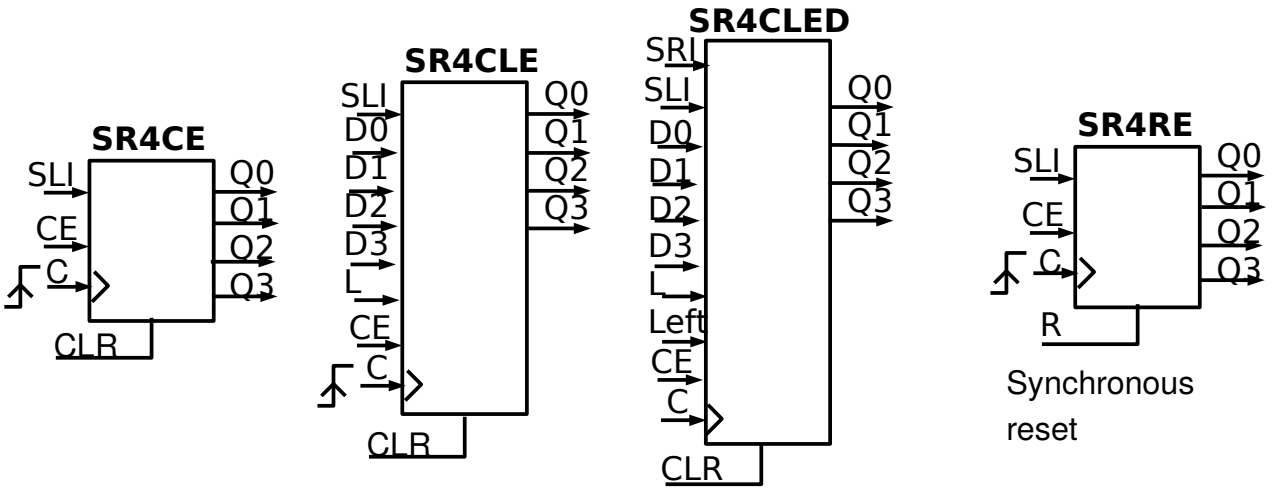
La structure d'un registre dépendra du mode, série ou parallèle, utilisé pour y écrire l'information et pour la lire ensuite.
- écriture et lecture parallèle (registre tampon, Buffer register)
- écriture et lecture en série (registre à décalage, Shift Register)
- écriture en parallèle et lecture en série (Parallel IN - Serial OUT)
- écriture série et lecture parallèle (Serial IN - parallel OUT)



Le chargement peut être asynchrone.
Remarque : un petit essai de spécification d'un registre à décalage par diagramme d'évolution nous montre les limites d'une telle spécification.

II) Primitives schématiques Xilinx pour les registres

Les registres à décalage sont uniquement à décalage gauche (vers les poids forts) sauf si une entrée Left est présente.

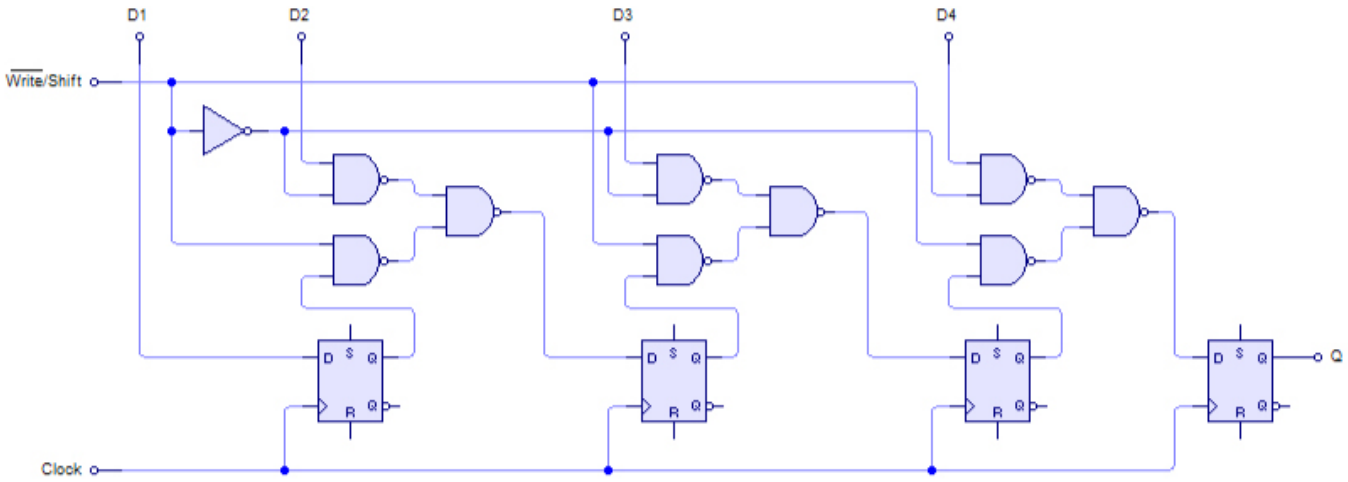


Naturellement les registres SR4RLE et SR4RLED existent aussi.

Exercice 1

Analyse the schematic below to find how to assert the Write/Shift control line for a load and show that when asserting with the complement we effectively have a shift register. What is the entry in this case ?

What is the name of this register ?



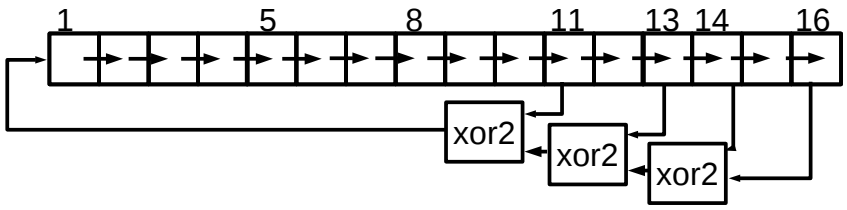
(From wikipedia http://en.wikipedia.org/wiki/Shift_register)

Exercice 2

Donner le schéma d'un registre 3 bits programmable, à écriture et lecture en série par décalage à droite ou à gauche, circulaire ou non. Prévoir deux entrées de programmation P1 et P2, et donner le code de programmation choisi. Utiliser des bascules D synchrones à front montant. Indications : on a encore ici une bonne illustration de la méthode du SI-ALORS. Écrire le programme VHDL correspondant.

Exercice 3 Linear Feedback Shift Register (LFSR)

The Linear Feedback Shift Register (LFSR) is a circuit commonly used in communications applications. Its primary purpose is to generate a pseudo-random sequence of bits that also cover the complete set of combinations of n-bits (except for zero).

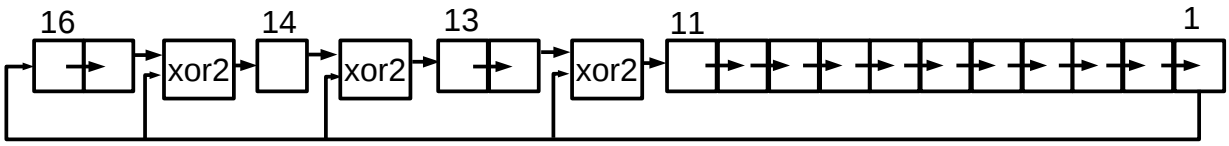


The image above (blatantly stolen from the LFSR Wikipedia article) shows the structural idea of the LFSR. Each bit position shifts one spot to the right within the register on each clock period. The left-most bit (denoted as “1” in the figure) is calculated by taking XORs of various selected bits within the register. The bits selected as tap points within the LFSR dictate what kind of pattern the register will go through. If you select the tap points carefully, the LFSR will hit every possible state of n bits, except for 0, and then start the pattern over again with the initial value.

1) With the Fibonacci's LFSR presented above, can you say what type of Xilinx register do you need for a realization ?
What are the recurrence equations of this register ?
Materialize this LFSR with a **SL16CE** Xilinx schematic component.

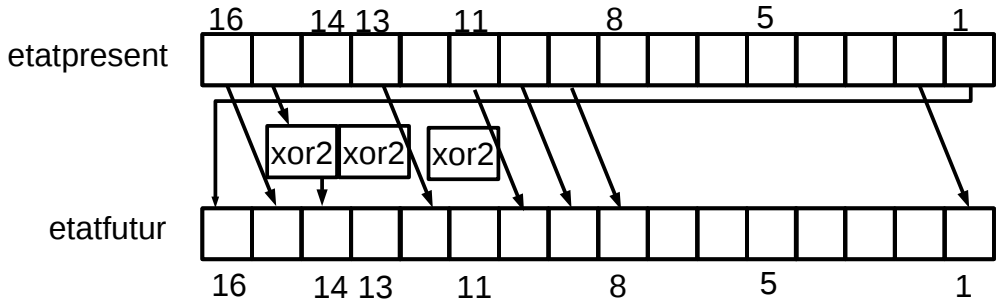
2) Nous allons gérer un générateur de Galois (1811-1832) de 16 bits dont voici le schéma tiré

de Wikipédia (http://en.wikipedia.org/wiki/Linear_feedback_shift_register):

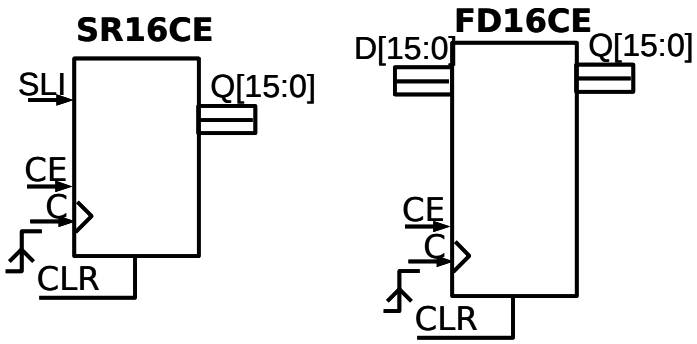


Le schéma ci-dessus représente essentiellement des décalages vers la droite. Si vous ne voulez pas les écrire bit à bit vous utiliserez l'opérateur '&' VHDL qui définit une concaténation (voir indications plus loin).

Compléter le schéma équivalent de ce LFSR qui sépare l'état présent de l'état futur. En déduire les 16 équations de récurrences correspondantes.



3) Réaliser ce LFSR en utilisant un composant **FD16CE**.



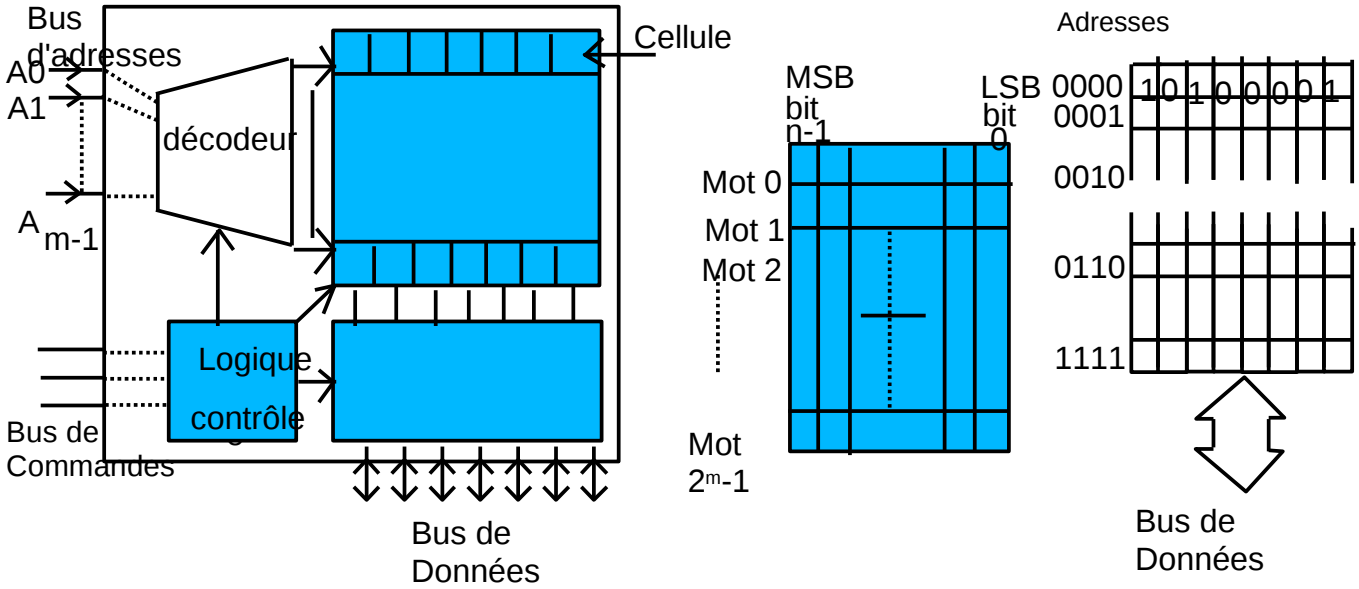
TD11 ENSL1 : Mémoires RAM, ROM ...

I) Les mémoires

Les mémoires ROM (réalisée en fonderie silicium)
Les mémoires RAM (statiques SRAM, dynamiques DRAM)
Les mémoires EPROM. Dénomination 27C256_20. 27=EPROM, C=CMOS 256=capacité en Kbits, 20= temps d'accès en unité 10 ns (ici 200 ns). Elles sont programmables électriquement, mais s'effacent avec des ultra-violet
Les EEPROM sont programmables et s'effacent électriquement, mais s'effacent avec des ultra-violet

II) Organisation des mémoires : les bus

Une mémoire comporte trois bus : bus d'adresses (sur m bits), le bus de données (sur n bits) et le bus de commande qui dépend beaucoup du type de mémoire. Fonctionnellement, une mémoire peut être représentée comme un tableau de 2^m mots de n bits.



Capacité des mémoires : $C = n .2^m$ bits.

Notion d'octet, de kilo octet ... $C = n/8 .2^m$ octets = $(n/8*1024) .2^m$ koctets ...

III) Bus de commande

Each memory device has at least one chip select (CS) or chip enable (CE) or select (S)pin that enables the memory device. This enables read and/or write operations.

Each memory device has at least one control pin. For ROMs, an output enable (OE) or gate (G) is present. The OE pin enables and disables a set of tristate buffers. For RAMs, a read-write (R/W) or write enable (WE) and read enable (OE) are present

CE	OE	WE	Fonctionnement
0	0	0	Mémoire sélectionnée en écriture et lecture. État à éviter !
0	0	1	Mémoire sélectionnée en lecture
0	1	↑	Mémoire sélectionnée en écriture. L'écriture est effective avec le front
0	1	1	Mémoire sélectionnée et sans opération

1	X	X	Mémoire non sélectionnée

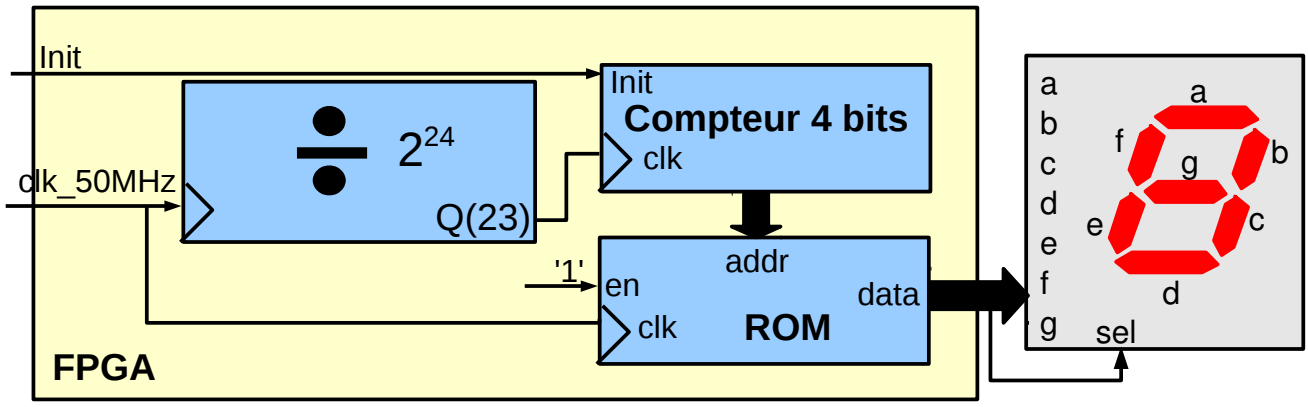
Écriture dans une mémoire : la mémoire dispose d'une entrée notée /WE (Write Enable) qui doit prendre la valeur 0. L'écriture devient possible sur un front de /WE si la mémoire est sélectionnée (/CE=0)

IV) Exercice

Exercice 1

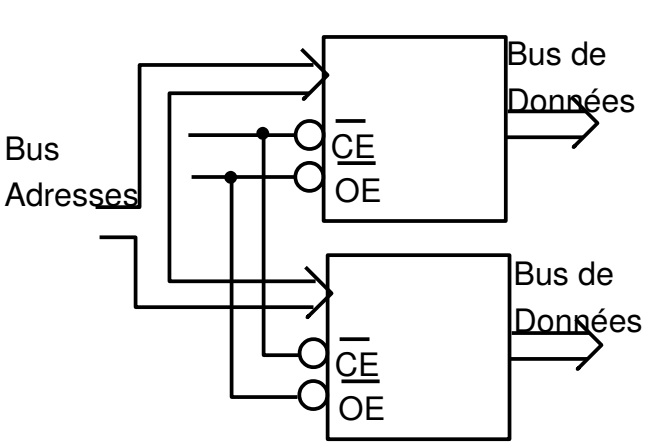
Nous allons utiliser un compteur normal, c'est à dire qui compte en binaire sur 4 bits suivi d'une mémoire pour le transcodage. L'horloge de ce compteur sera réalisée par le poids fort d'un compteur comme indiqué dans la figure qui suit.

Dimensionner la ROM nécessaire à votre problème. Calculer les valeurs hexadécimales contenues dans cette ROM.

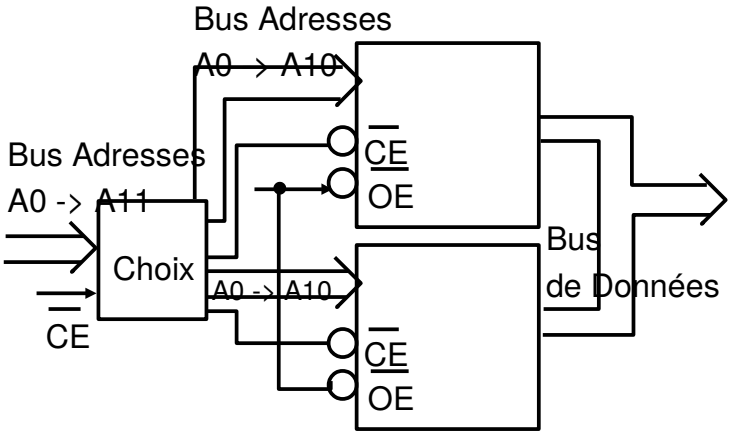


V) Associations de mémoires

Comment faire une mémoire avec deux composants mémoires pour augmenter la capacité ? Il y a deux façons de faire : doubler n ou doubler le nombre de mots de n bits. Voici comment on réalise cela en pratique.



Agrandissement du bus de données



Agrandissement du bus d'adresses
A0 -> A10 devient A0 ->A11

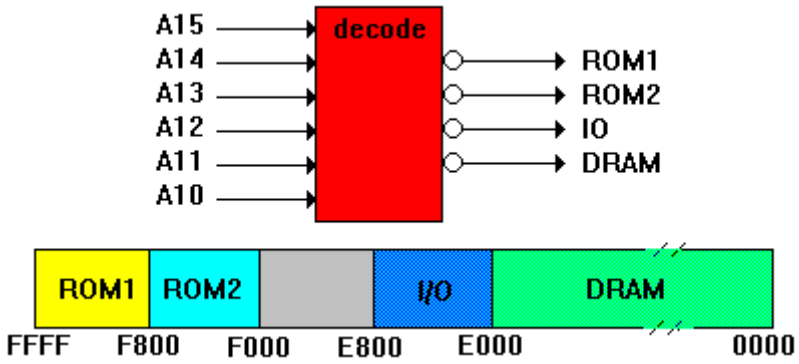
VI) Exercices

Exercice 2

Un projet nécessite l'utilisation de 4 RAMB16_S4 pour réaliser une mémoire de données sur 16 bits et de taille 8ko.
1) Ces données vous semblent-elles cohérentes ?
2) Faire un schéma d'ensemble en prenant soin de dimensionner correctement les adresses et données.

Exercice 3 Address decoding

The following figure shows the block diagram for this design and a continuous block of memory divided into four sections containing DRAM, I/O, ROM1 and ROM2. The purpose of this decoder is to monitor the six high-order bits (A15-A10) of a sixteen-bit address bus and select the correct section of memory based on the value of these address bits.



Memory Section	Address Range (Hex)
DRAM	0000-DFFF
I/O	E000-E7FF
ROM2	F000-F7FF
ROM1	F800-FFFF

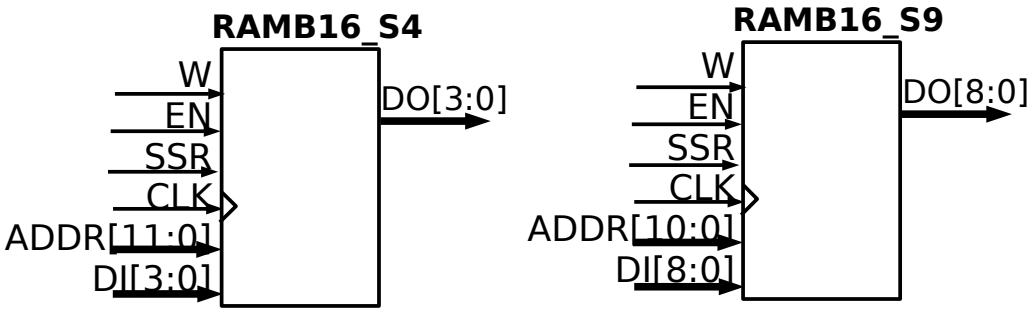
Find out every output equation.

Exercice 4 Address decoding

We want to realize a memory interface to an 8-bit data bus (D7-D0) using a 16-bit address bus (A15-A0) of a micrprocessor-based system. The micoprocessor has an /WR signal to write and an /RD signal to read. Give the selection equations for :
- a 16K ROM for a memory address range that begins at location 2000H
- a 8K ROM for a memory address range that begins at location 8000H
- a 4K ROM for a memory address range that begins after the 8K ROM location.

VI) Quelques primitives Xilinx de mémoires

Une RAMB16 correspond à 16 kbits et donc 2 ko. Ses bus données (et donc forcément adresse) peuvent être choisis de différentes manières. En voici deux exemples :


RAMB16_S4
W
EN
SSR
CLK
ADDR[11:0]
DI[3:0]
DO[3:0]
RAMB16_S9
W
EN
SSR
CLK
ADDR[10:0]
DI[8:0]
DO[8:0]

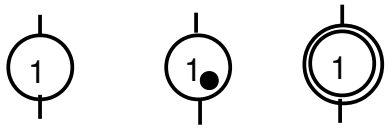
TD12 ENSL1 : Description par graphe d'états

I) Graphe d'états

Un graphe d'état est une suite d'états et de transitions réceptives.

1°) États

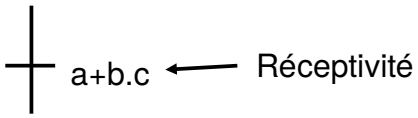
Comme on l'a fait jisqu'à présent, les états sont représentés par des cercles.



inactif actif initial

2°) Transitions

Les transitions par contre deviennent réceptives (barrées par un trait)



3°) Actions

Une ou plusieurs actions sont associées aux états pour « commander » les sorties.

II) Équations de récurrence

Code One-hot-One (une bascule par état)

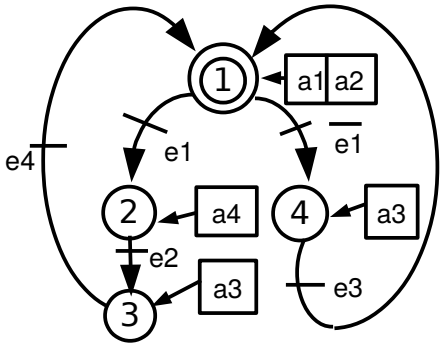
On cherche pour chacun des états i les conditions d'activations AC_i et les déactivations D_i puis on écrit :

$x_i^+ = AC_i + \overline{D_i} \cdot x_i + Init$ pour un état initial et

$x_i^+ = (AC_i + \overline{D_i} \cdot x_i) \cdot \overline{Init}$ pour un état normal.

III) Implantation

On implante ces équations de récurrence facilement avec des bascules D (voir TD 10).



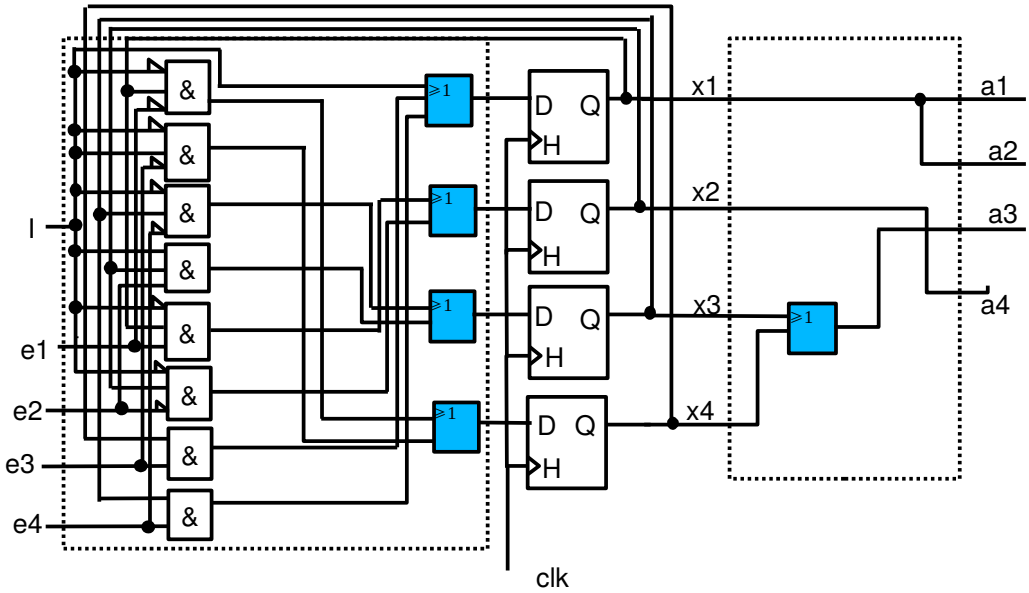
AC1 = x3.e4+x4.e3 D1 = e1+/e1=1
AC2 = x1.e1 D2 = e2
AC3 = x2.e2 D3 = e4
AC4 = x1./e1 D4 = e3

$x1^+$ = x3.e4+x4.e3 + Init
$x2^+$ = (x1.e1+x2./e2)./Init
$x3^+$ = (x2.e2+x3./e4)./Init
$x4^+$ = (x1./e1+x4./e3)./Init

Équations de sorties
a1 = x1 a2 = x1
a3 = x3 + x4
a4 = x2

Muni de toutes ces équations on écrit le programme VHDL :

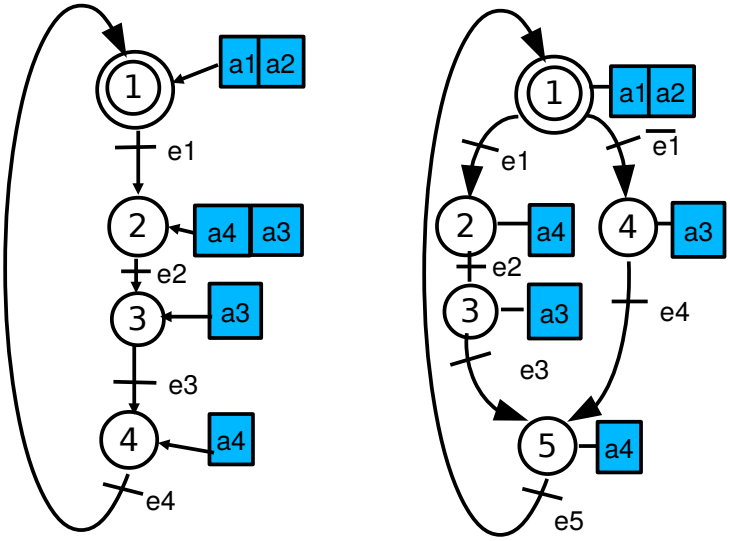
```
-- programme VHDL correspondant au graphe d'états précédent
ENTITY graf1 IS
PORT (I,e1,e2,e3,e4,clk : IN BIT;
      a1,a2,a3,a4 : OUT BIT);
END graf1;
ARCHITECTURE agraf1 OF graf1 IS
SIGNAL x1,x2,x3,x4 : BIT;
BEGIN
  PROCESS(clk) BEGIN
    IF (clk'event AND clk='1') THEN
      x1 <= (x3 AND e4) OR (x4 AND e3) OR I;
       x2 <= (x1 AND e1 AND NOT I) OR (x2 AND NOT e2 AND NOT I);
       x3 <= (x2 AND e2 AND NOT I) OR (x3 AND NOT e4 AND NOT I);
       x4 <= (x1 AND NOT e1 AND NOT I) OR (x4 AND NOT e3 AND NOT I);
    END IF;
  END PROCESS;
  a1 <= x1;
  a2 <= x1;
  a3 <= x3 OR x4;
  a4 <= x2;
END agraf1;
```



IV) Exercices

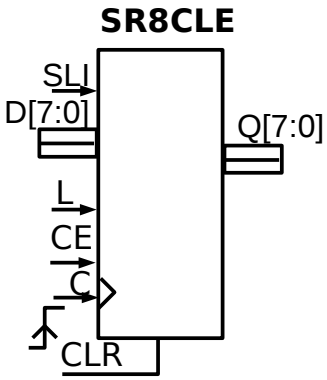
Exercice 1

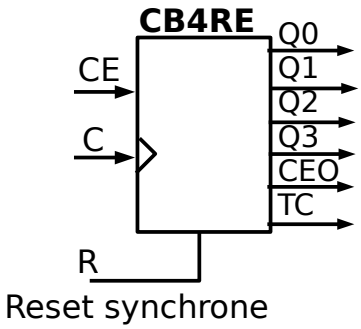
Given these two states flow diagram, build up the corresponding sequential functions. Write a VHDL program.

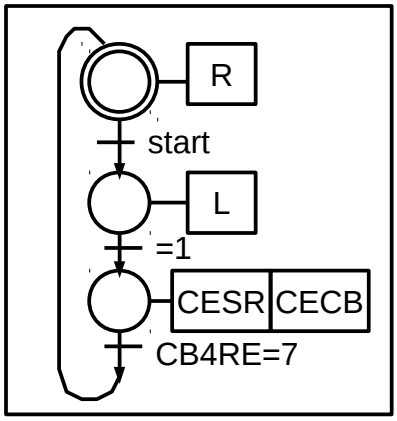


Exercice 2 Parallel to serial conversion

Design a system which can convert 8 parallel input bits (DataIn[7:0]) to a serial output SerialOut. The datapath of the system should use an 8 bit shift register (SR8CLE) and a 3 bit counter (only CB4RE available in Xilinx library) and any other combinational logic. The control section should use a D FF based More FSM to keep track of whether the system is waiting for START or shifting.
In addition to schematic, show a timing diagram of operation, and state diagram for the controller FSM.







La programmation se fait toujours à travers un projet. La première chose à apprendre est donc de savoir créer et gérer un projet. On rappelle qu'un projet permet de rassembler plusieurs types de fichiers ensembles.

Comment démarrer un projet

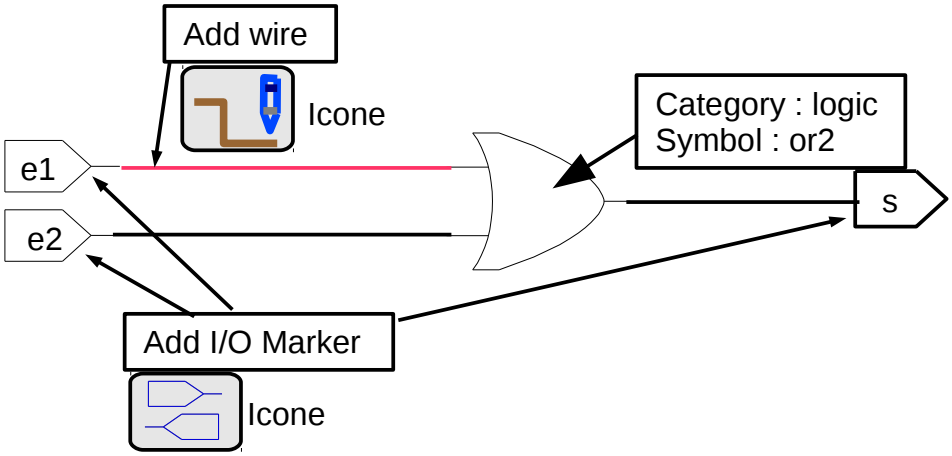
La première chose que l'on se demande est quel est le FPGA cible que l'on utilise et quelle sorte de boîtier on utilise ? En ce qui nous concerne ce sera un SPARTAN 3E 250 (**xc3s250e**) dans un boîtier **cp132** (frequency upgrade : **-5**).

On choisit ensuite le type de langage : pour nous ce sera du schématique... et on se laisse guider.

Notre premier projet sera composé de deux fichiers : un fichier de schéma (extension .sch) et un fichier de contrainte (extension .ucf)

Saisir un schéma

On va commencer par un exemple très simple mais qui nous permettra de comprendre comment l'environnement fonctionne.



On a deux entrées que l'on va relier à deux interrupteurs (sw0 et sw1 positionnés respectivement en L13 et L14) et une sortie qui va nous servir à allumer une led (led0 en F12).

Dans l'ordre, il est bon de commencer par les composants, puis par les connexions (wire) pour terminer par les I/O Marker.

Saisir le fichier ucf

C'est un fichier texte dont la syntaxe est relativement simple. Il existe un outil graphique pour faire cela mais nous utiliserons un éditeur de texte simple.

```
net "e1" loc="L13";
net "e2" loc="L14";
net "s" loc="F12";
```

Seul la documentation de la carte sur laquelle est votre FPGA vous permet de trouver comment s'appellent les broches de votre FPGA pour les mettre dans le fichier ucf correspondant. Cette

documentation vous sera toujours fournie.

Notez que pour ouvrir un fichier ucf, il ne faut pas double-cliquer dessus : cliquer simplement et aller chercher (sur la gauche) dans la fenêtre process :

User Constraints

|__ Edit constraints (text).

Compilation

La compilation a comme objectif de réaliser un fichier binaire (d'extension .bit) qui sera téléchargé dans le FPGA. On clique dans la fenêtre projet sur le fichier le plus haut dans la hiérarchie (pour le moment ce sera notre schéma) puis dans la fenêtre process on choisit "Generate Programming File" (double clic)

Téléchargement

Le téléchargement dans la carte se fait avec Impact (trouvé dans la fenêtre process) :

Configure Target Device

|__ Manage Configuration Project (Impact)

Convention de noms pour les portes

Pour s'y retrouver parmi les nombreuses portes logiques disponibles, nous donnons les indications suivantes. Les noms des portes sont de la forme pNbM

- p est un nom de porte comme and, or, nor, nand, xor et xnor représentant le ET, le OU, le OU-NON le ET-NON, le OU-Exclusif, et l'identité.
- N est le nombre d'entrées de la porte, typiquement de 2 à 9 et même parfois 16.
- b est une lettre suivie d'un chiffre M variant de 1 à typiquement 5 désignant le nombre d'entrées complémentées. Certains composants n'ont pas cette extension, par exemple le ou exclusif va de xor2 à xor9.

La suite est importante pour les projets un peu plus conséquents et peut être passée en première lecture.

Renommer un fil (net)

Renommer un fil est utile pour relier un fil à une entrée sans dessiner cette liaison.

Pour renommer un fil, **ne pas passer** par un click droit sur le fil en essayant de trouver "Object Property", mais plutôt :

- edit -> Rename -> rename Selected net puis donner le nom puis OK.

Création de plusieurs feuilles

Quand votre dessin devient complexe il faut le répartir sur plusieurs feuilles. Pour se faire :

- click droit dans la feuille -> object properties New

Pendant que vous y êtes, il est possible de changer la taille de la feuille ou des feuilles.

Création d'un symbole

Il pourra nous arriver de rassembler un schéma complexe dans un symbole. Ceci peut être fait

ainsi :

Dans la fenêtre Design (tout en haut) : Design Utilities -> Create Schematic Symbol

Ceci peut être réalisé aussi comme ceci:

- File -> New -> Symbol

Ou avec utilisation du sorcier (Wizard)

- Tools -> symbol wizard
- Next

Donner un nom au symbole et ajouter le nom des entrées et sorties puis next.

Remarque : Le "symbol wizard" permet de transformer directement votre schéma en cours en symbole si vous cochez "using schematic" et il vous trouve alors tout seul l'ensemble des entrées/sorties.

TP1 - Tables de vérité et LUTs (3 heures)

L'objectif de ce TP est de montrer qu'avec un FPGA la connaissance d'une table de vérité suffit pour réaliser un schéma (sans les simplifier). On utilisera encore un SPARTAN 3E 250 (**xc3s250e**) dans un boîtier CP132.

Utilisation ADEPT sous Linux

Détection	`djtgcfg enum` donne Basys2 come username
Téléchargement	`djtgcfg prog -d Basys2 --index 0 --file toto.bit <<<y`
Lancement ISE	`/opt/Xilinx/14.5/ISE_DS/ISE/bin/lin/ise &`

I) Un exemple simple pour comprendre l'IDE Xilinx

Vous allez réaliser un simple OU en suivant un exemple complet réalisé par votre enseignant. L'objectif est d'apprendre :

- à réaliser un fichier de schéma, le sauver et l'ajouter au projet
- à réaliser un fichier de contrainte pour choisir les entrées et sorties, et l'ajouter au projet
- à compiler
- à charger le fichier dans votre FPGA avec IMPACT ou ADEPT

Modifier cet exemple pour ajouter une sortie qui réalise un ET.

Fichier ucf

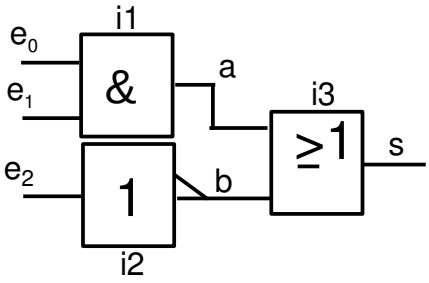
Pour éviter de chercher nous donnons quelques contenus du fichier ucf :

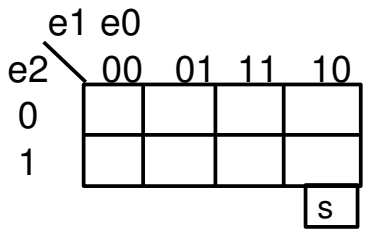
```
# Pin assignment for SWs
NET "sw7" LOC = "N3";  # Bank = 2, Signal name = SW7
NET "sw6" LOC = "E2";  # Bank = 3, Signal name = SW6
NET "sw5" LOC = "F3";  # Bank = 3, Signal name = SW5
NET "sw4" LOC = "G3";  # Bank = 3, Signal name = SW4
NET "sw3" LOC = "B4";  # Bank = 3, Signal name = SW3
NET "sw2" LOC = "K3";  # Bank = 3, Signal name = SW2
NET "sw1" LOC = "L3";  # Bank = 3, Signal name = SW1
NET "sw0" LOC = "P11";  # Bank = 2, Signal name = SW0
# Pin assignment for LEDs
NET "Led7" LOC = "G1" ; # Bank = 3, Signal name = LD7
NET "Led6" LOC = "P4" ; # Bank = 2, Signal name = LD6
NET "Led5" LOC = "N4" ;  # Bank = 2, Signal name = LD5
NET "Led4" LOC = "N5" ;  # Bank = 2, Signal name = LD4
NET "Led3" LOC = "P6" ; # Bank = 2, Signal name = LD3
NET "Led2" LOC = "P7" ; # Bank = 3, Signal name = LD2
NET "Led1" LOC = "M11" ; # Bank = 2, Signal name = LD1
```

```
NET "Led0" LOC = "M5" ;  # Bank = 2, Signal name = LD0
```

II) Analyse d'un schéma simple puis synthèse équivalente

On donne le schéma ci-dessous.





1°) After realizing the circuit above, plot the corresponding function in the truth table below and in the Karnaugh map above. Find the simplified sum of product expression of s=f(e0,e1,e2).

e2	e1	e0	a	b	s
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

2°) Realize the corresponding simplified expression as a nand circuit.

Exercice 2 (Vote au directoire)

Le comité directeur d'une entreprise est constitué de quatre membres :
- le directeur D,
- ses trois adjoints A, B, C.

Lors des réunions, les décisions sont prises à la majorité.
Chaque personne dispose d'un interrupteur pour voter sur lequel elle appuie en cas d'accord avec le projet soumis au vote.
En cas d égalité du nombre de voix, celle du directeur compte double.
On vous demande de réaliser un dispositif logique permettant l'affichage du résultat du vote sur lampe V (pour nous ce sera une LED).

1°) Écrire une table de vérité pour la sortie V puis un tableau de Karnaugh

2°) Réaliser le schéma logique ET/OU de la sortie V

Des tables de vérité à l'hexadécimal (cours)

Une table de vérité de trois entrées peut être représentée par un nombre 8 bits que l'on convertit en hexadécimal. Soit donc la table de vérité suivante (trois entrées notées e0, e1 et e2, une sortie notée s) :

e2	e1	e0	s	
0	0	0	0	b_0
0	0	1	1	b_1
0	1	0	1	b_2
0	1	1	0	b_3
1	0	0	1	b_4
1	0	1	0	b_5
1	1	0	1	b_6
1	1	1	0	b_7

Vous pouvez synthétiser la table de vérité à l'aide d'un seul nombre sur 8 bit (poids faible en haut) :

- en binaire ce nombre vaut : 0b01010110
- en hexadécimal ce nombre vaut : 0x56 (X"56" en VHDL)

Aucune simplification à faire ici

La valeur hexadécimale 56 est la valeur qu'il vous faudra utiliser dans la partie INIT d'une (catégorie) LUT avec un composant **lut3**.

Pour initialiser une LUT, double cliquer dessus et remplir le champ INIT.

Combien de bits nécessite l'initialisation d'une **lut4** à 4 entrées ? Combien de chiffres hexadécimaux ?

3°) Transformer la table de vérité de la question 1° en LUT4 et réalisez-en le en schéma et essayez-le.

Exercice 3 (Vote au directoire amélioré)

Une société est composée de 4 actionnaires ayant les nombres suivants d'actions A=60, B=100, C=160 et D=180.
Nous désirons construire une machine à voter automatiquement, tenant compte dans le résultat du poids en actions de chaque personne. La machine dispose de 4 boutons poussoirs A, B, C, D et le résultat sera un voyant V qui s'allumera si la majorité pondérée appuie sur les boutons.

1°) Écrire une table de vérité pour la sortie V.
Transformer cette table de vérité en LUT4.
Réaliser le schéma correspondant et vérifier que vous avez la bonne table de vérité.

2°) Réaliser le schéma logique ET-NON de la sortie V.

TP2 - Des tables de vérités aux simplifications (en VHDL)

L'objectif de ce TP est de montrer qu'avec un FPGA la connaissance d'une table de vérité permet de réaliser un tableau de Karnaugh pour une simplification. On utilisera encore un SPARTAN 3E 250 (**xc3s250e**) dans un boîtier CP132.

Utilisation ADEPT sous Linux

Détection	`djtgcfg enum` donne Basys2 come username
Téléchargement	`djtgcfg prog -d Basys2 --index 0 --file toto.bit <<<y`
Lancement ISE	`/opt/Xilinx/14.5/ISE_DS/ISE/bin/lin/ise &`

2-bit multiplier

Exercise 1 (English)

Design and build a combinational logic circuit with four inputs and four outputs that multiplies two 2-bits numbers, labeled X, and Y. The output is labeled Z. Use a truth table and find out the corresponding with select when.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity multiplier is
    port (  X,Y: in STD_LOGIC_VECTOR (1 downto 0);    -- binary inputs
        Z : out STD_LOGIC_VECTOR (3 downto 0));       -- binary output
end multiplier;
```

Indication sur with select when

Soit la table de vérité suivante :

a3	a2	a1	a0	s1	s0
0	1	0	1	1	1
0	1	1	0	0	1
1	1	0	1	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
ENTITY demo IS PORT(
  a : in STD_LOGIC_VECTOR(3 DOWNTO 0);-- 4 entrées
  s : out STD_LOGIC_VECTOR(1 DOWNTO 0));
  -- 2 sorties
END demo;
```

Il manque 13 lignes à cette table, qui sont retirées, mais donnent des 0 en sortie

Notez que la partie gauche de la table de vérité appelée partie SI concerne les entrées, et la partie droite (appelée ALORS) concerne les sorties. La donnée d'une table de vérité permet directement de trouver l'entité. Elle permet aussi d'écrire directement l'architecture avec un "with select when" :

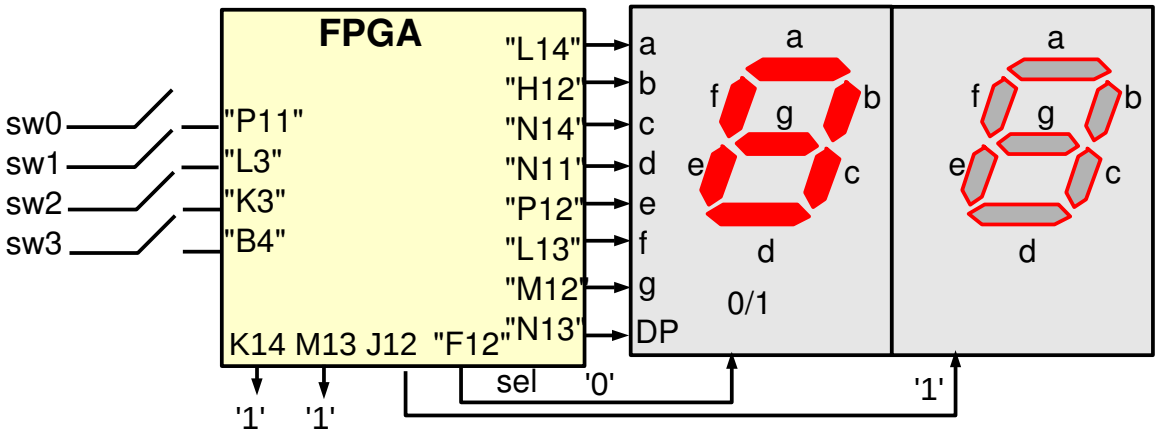
```
1             ARCHITECTURE mydemo OF demo IS
2             BEGIN
3               WITH a SELECT             --style with select when
4                 s <= "11" WHEN "0101", -- premiere ligne
5                      "01" WHEN "0110", -- deuxieme ligne
6                      "10" WHEN "1101", -- troisieme ligne
7                      "00" WHEN OTHERS; -- pour les 13 lignes retirées
8             END mydemo;
```

Transcodeur binaire 7 segments

On va réaliser un transcodeur binaire décimal vers un afficheur 7 segments. L'objectif de ce TP est d'utiliser toutes les techniques classiques de synthèse combinatoire.

Présentation du sujet

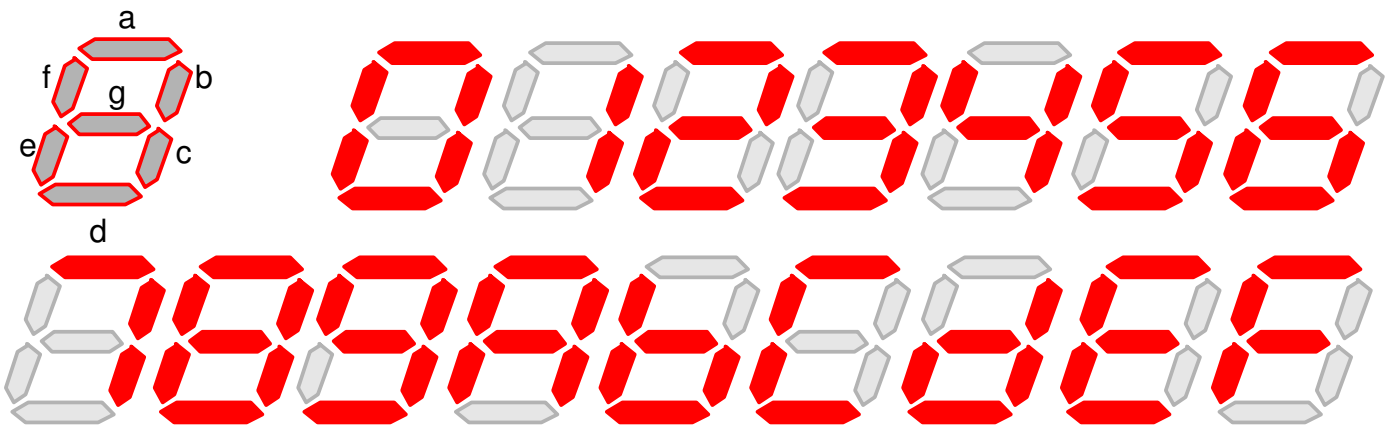
Un schéma présente de manière symbolique ce que l'on cherche à faire.



Sortie sel pour sélectionner afficheur

sel est choisi à 0 pour sélectionner l'afficheur de gauche.

La valeur binaire (de 0 à 9) choisie sur les interrupteurs est convertie pour être affichée :



Transcodeur hexadécimal vers sept segments et « with select when »

On va s'intéresser au transcodeur binaire décimal vers un afficheur 7 segments. L'objectif de ce TP est d'utiliser les techniques classiques des tables de vérité spécifiques aux FPGA pour réaliser un composant complet.

Exercise 2 (English)

First complete the truth table below. Read the first line carefully before starting.

Realize this truth table as circuit.

sw3	sw2	sw1	sw0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							
1	0	1	0							
1	0	1	1							
1	1	0	0							
1	1	0	1							
1	1	1	0							
1	1	1	1							

Exercice 3 : synthèse du segment "a"

En cherchant à simplifier, rechercher l'équation disjonctive correspondant au segment a. Écrire l'équation disjonctive, puis implanter .

Exercice 4 : synthèse du segment "b"

Réaliser de la même façon le segment b.

Exercice 5 : synthèse du segment "c"

Réaliser de la même façon le segment c.... et la suite.

TP3 Arithmétique et multiplexeurs (Schématique)

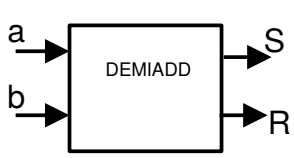
L'arithmétique consiste à implanter des fonctions de base de l'arithmétique, c'est à dire addition, soustraction et multiplication. L'utilisation de l'arithmétique dans un FPGA doit suivre ses propres règles. Nous allons commencer par examiner l'addition, d'abord sans se préoccuper du fait que l'on utilise un FPGA, puis on cherchera à utiliser les composants Xilinx. On utilisera encore un SPARTAN 3E 250 (**xc3s250e**) dans un boîtier CP132.

Utilisation ADEPT sous Linux

Détection	`djtgcfg enum` donne Basys2 come username
Téléchargement	`djtgcfg prog -d Basys2 --index 0 --file toto.bit <<<y`
Lancement ISE	`/opt/Xilinx/14.5/ISE_DS/ISE/bin/lin/ise &`

Du demi-additionneur à l'additionneur 1 bit

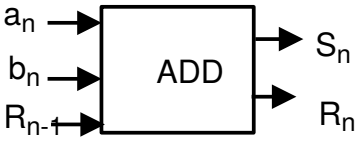
Les opérations d'addition sur 1 bits sont très simples et donnent un résultat sur 2 bits. Le bit de poids fort est appelé R (pour retenue= carry en anglais) et le poids faible est appelé S (pour somme).

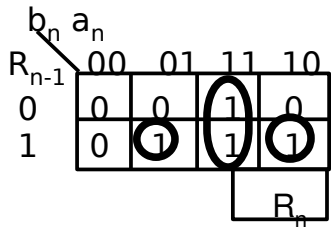


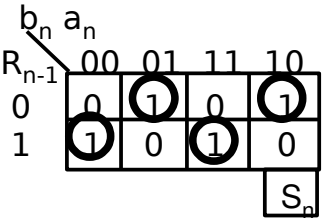
b	a	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = a \oplus b$

$R = a.b$







$S_n = a_n \oplus b_n \oplus R_{n-1}$

$R_n = a_n.b_n + R_{n-1}.(a_n \oplus b_n)$

Remarquez que R_n n'est pas simplifié au mieux pour faire apparaître un OU exclusif.

Exercice 1

1) Implanter un additionneur 1 bit avec deux ou-exclusifs à deux entrées et trois portes NANDs. Pour les tests, les entrées seront reliées à des interrupteurs et les sorties à des LEDs.

2) Copier 4 fois le schéma précédent et montrer que l'on peut réaliser ainsi un additionneur 4

bits. N'oubliez pas que la somme de deux nombres de 4 bits peut donner un nombre de 5 bits.

Faites attention lorsque vous dupliquez un schéma.

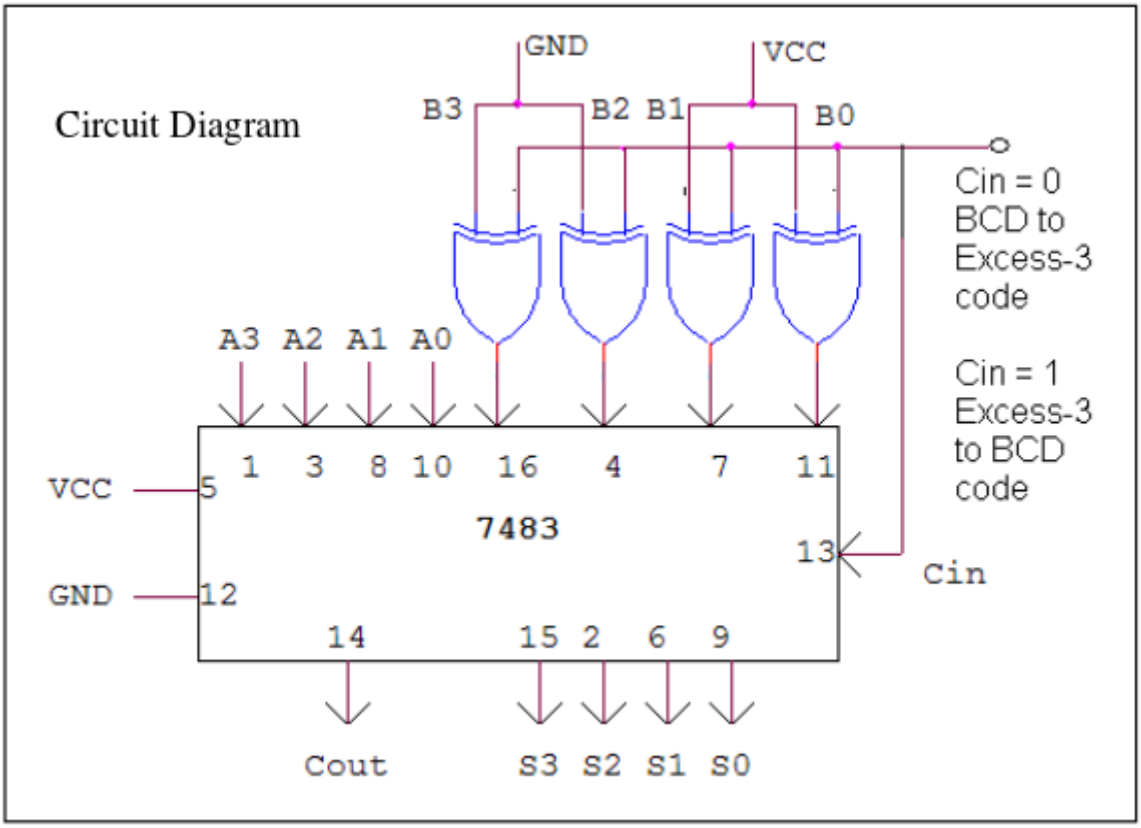
Tous les fils portant le même nom sont automatiquement reliés. Vous devez donc renommer les entrées et sorties entre le copier et le coller.

Additionneur 4 bits

Comme indiqué en introduction, si vous avez de l'arithmétique à faire il faut mieux utiliser des composants tout faits. En effet, tout FPGA moderne possède un certain nombre de parties pré-câblées réalisant ces fonctions. On utilisera ainsi le composant **ADD4** dans la catégorie "arithmetic". Les entrées de ce composant seront des interrupteurs extérieurs mais ses sorties seront destinées à être affichées sur un afficheur sept segments... et c'est là que les choses se compliquent un peu.

Exercise 2 (BCD TO EXCESS- 3 CODE CONVERTERS)

Code converter is a combinational circuit that translates the input code word into a new corresponding word. The excess-3 code digit is obtained by adding three to the corresponding BCD digit. To Construct a BCD-to-excess-3-code converter with a 4-bit adder feed BCD-code to the 4-bit adder as the first operand and then feed constant 3 as the second operand. The output is the corresponding excess-3 code.



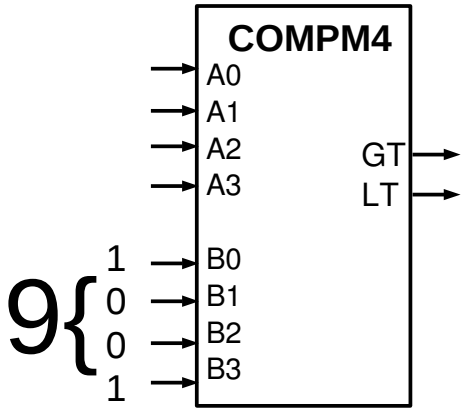
See exercise 4 of chapter 5 to obtain the corresponding truth table.

Realize this circuit diagram : replace 7483 TTL circuit with Xilinx's **ADD4.**

Exercice 3 (ds TP 2012)

Première partie

Réaliser la comparaison seule (figure ci-contre). Le fait de fixer les entrées Bi à 9 en fait une fonction de 4 entrées A3,A2,A1,A0.

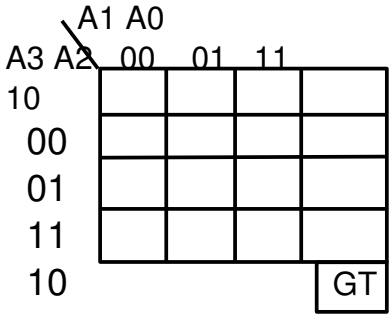


Réaliser le fichier ucf pour utiliser sw3, sw2, sw1 et sw0 en entrée et led0 en sortie (pour GT seulement). Le réaliser dans le FPGA et faire constater à l'enseignant. On rappelle que 1 est réalisé avec vcc et 0 avec gnd.

Deuxième partie

A l'aide du schéma de la première partie, remplir une table de vérité puis un tableau de Karnaugh.

En déduire un schéma ET/OU. Le réaliser dans le FPGA et faire constater à l'enseignant.



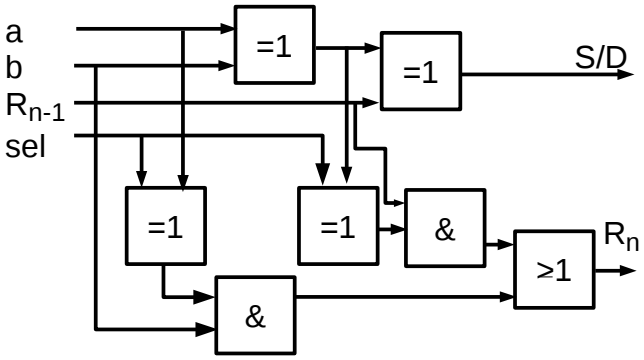
A3	A2	A1	A0	GT
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Troisième partie

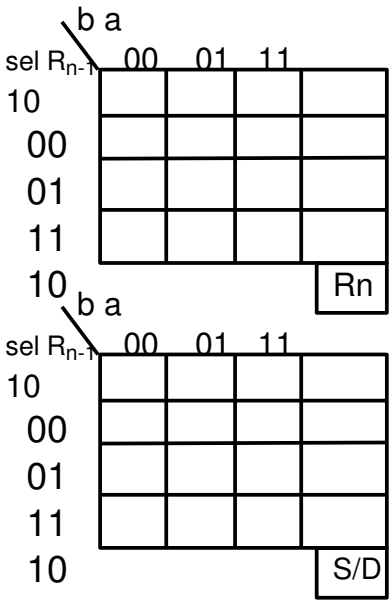
Déduire de la table de vérité une LUT4 réalisant l'ensemble. La câbler puis l'essayer dans le FPGA. Faire constater à l'enseignant.

Exercice 4 (ds TP 2012)

Première partie
Réaliser l'additionneur soustracteur (figure ci-contre). C'est une fonction de 4 entrées sel,R_{n-1},b,a et deux sorties.
Réaliser le fichier ucf pour utiliser sw3, sw2, sw1 et sw0 en entrée et led1,led0 en sortie (pour R_n,S/D).
Le réaliser dans le FPGA et faire constater à l'enseignant.



Deuxième partie
A l'aide du schéma de la première partie, remplir une table de vérité puis les tableaux de Karnaugh.
En déduire un schéma ET/OU de Rn. Le réaliser dans le FPGA et faire constater à l'enseignant.



sel	R_{n-1}	b	a	Rn	S/D
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Troisième partie
Déduire de la table de vérité une LUT4 réalisant S/D. La câbler en complétant le schéma de la deuxième partie, puis les essayer dans le FPGA. Faire constater à l'enseignant.

TP4 Introduction au séquentiel

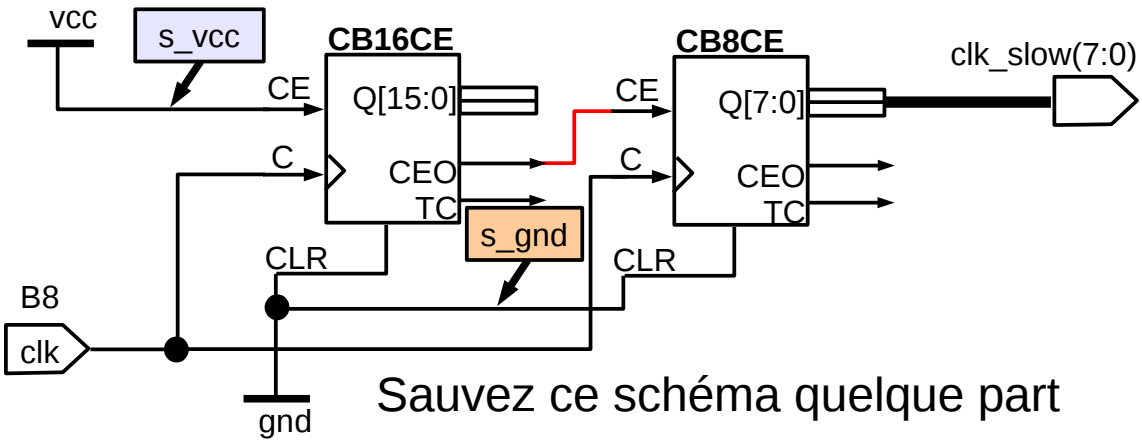
On utilisera encore et toujours un SPARTAN 3E 250 (**xc3s250e**) dans un boîtier CP132.

Téléchargement	`djtgcfg prog -d Basys2 --index 0 --file toto.bit <<<y`

Après le combinatoire naquit l'horloge....

Les compteurs permettent de diminuer la fréquence d'horloge.

Travail à réaliser (en schématique)



Exo1 : Vous disposez d'une horloge 50MHz en broche "B8" de votre composant FPGA SPARTAN 3E. Réaliser à l'aide d'un compteur 24 bits une horloge d'environ 3 Hz. Sortie sur une LED, le clignotement de celle-ci doit être visible à l'œil.

Indications

```
# clock pin for Basys2 Board
NET "clk" LOC = "B8"; # Bank = 0, Signal name = MCLK
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
```

L'horloge que l'on vient de réaliser sera utilisée presque systématiquement par la suite.

Par abus de langage on appellera compteur dans la suite un élément séquentiel qui comporte une horloge mais qui ne compte pas forcément en binaire. Dans ce dernier cas, son équation de récurrence ne peut donc pas s'écrire simplement à l'aide d'une addition.

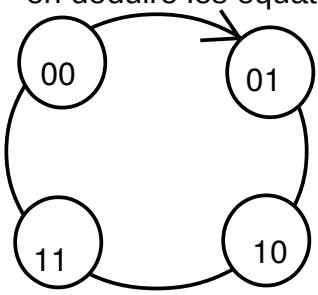
Réaliser un graphe d'évolution simple

Avant toute chose, il nous faut nous demander comment utiliser des équations de récurrences booléennes. On vous représente ci-après la façon de procéder :

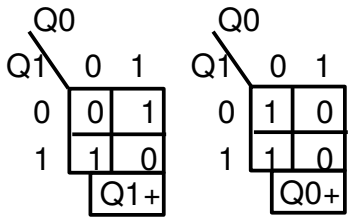
- trouver le diagramme d'évolution

- en déduire le tableau état présent/état futur

- en déduire les équations en utilisant éventuellement un tableau de Karnaugh



État présent		État futur	
Q1	Q0	Q1+	Q0+
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Nous avons à gauche un diagramme d'évolution, au centre un tableau état présent état futur et à droite les tableaux de Karnaugh permettant de trouver les équations de récurrences.
Équations de récurrence :

$Q_1^+ = Q_1 \oplus Q_0$
$Q_0^+ = /Q_0$

Exo2 : 1°) Realize the system specified above with a state diagram. The clock is, as usual, "clk_slow(7)" (see l'exercise 1). Add two LEDs as outputs to be able to check your schematic.

2°) Design a modulo-4 counter with an input D, for Down. When D is not asserted, the counter counts up, modulo-4. When D is asserted (1), the counter counts down, modulo-4. Draw a state diagram to implement it and derive the state table. Then design a complete sequential circuit using OR/AND gates, and FD flip-flop.

Indication : deux composants de type **FD** dans flip-flop seront choisis pour les bascules D. Vous choisirez les portes que vous voulez pour la partie combinatoire.

Compteur simple précédent en VHDL

Voici les extraits importants du programme VHDL correspondant :

```
1             PROCESS(clk) BEGIN
2               IF clk'event and clk='1' THEN -- ou IF rising_edge(clk) THEN
3                   -- équations de récurrences ici
4                   q1 <= q1 xor q0;
5                   q0 <= not q0;
6               END IF;
7             END PROCESS;
```

Exo3 : Transformer le programme ci-dessous en symbole schématique puis l'insérer dans le schéma ci-dessus en prenant "clk_slow(7)" comme horloge avec deux leds comme sorties.

```
1             library ieee;
2             use ieee.std_logic_1164.all;
3             ENTITY exo4 IS PORT ( clk : IN std_logic; --clk_slow(7) ici
4               q0,q1 : OUT std_logic);
5             END exo4;
6             ARCHITECTURE aexo4 OF exo4 IS
7             SIGNAL ioq0, ioq1 : std_logic;
8             BEGIN
9               PROCESS(clk) BEGIN
10                IF clk'event and clk='1' THEN -- ou IF rising_edge(clk) THEN
```

```
11                  -- équations de récurrences ici
12                  ioq1 <= ioq1 xor ioq0;
13                  ioq0 <= not ioq0;
14                END IF;
15              END PROCESS;
16            q0 <= ioq0;
17            q1 <= ioq1;
18            END aexo4;
```

Remarque

Il existe une méthode qui évite le passage par les équations de récurrences : on part du graphe d'évolution et on obtient directement le programme VHDL :

```
1             library IEEE;
2             use IEEE.STD_LOGIC_1164.ALL;
3             ENTITY cmpt IS PORT (
4               clk: IN STD_LOGIC;
5               q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
6             END cmpt;
7             ARCHITECTURE acmpt OF cmpt IS -- comment éviter les equations
8               SIGNAL ioq : STD_LOGIC_VECTOR(1 DOWNTO 0));
9                BEGIN
10                PROCESS(clk) BEGIN
11                 IF (clk'EVENT AND clk='1') THEN
12                   CASE ioq is  --style case when
13                     WHEN "00" => ioq <="01"; -- premiere transition
14                     WHEN "01" => ioq <="10"; -- deuxieme transition
15                     WHEN "10" => ioq <="11"; -- troisieme transition
16                     WHEN OTHERS => ioq <="00"; -- quatrieme transition
17                   END CASE;
18                END IF;
19              END PROCESS;
20              q <= ioq;
21            END acmpt;
```

Le style "case when" est au graphe d'évolution ce que le "with select when" est à la table de vérité : un moyen d'éviter les équations.

Compteur décimal à sortie directe sept segments (1 digit en VHDL)

Le compteur sept segments est réalisé en VHDL et sera assemblé avec le dispositif destiné à réaliser l'horloge lente : cascade d'un **CB16CE** avec un **CB8CE**.

Travail à réaliser

Exo4 :

1) Écrire les équations logiques a+, b+, c+, d+ ,e+, f+, g+ en fonction des entrées a, b, c, d, e, f, g.

2) Il nous reste un problème important à résoudre, c'est l'initialisation. En effet pour bien faire il faudrait étudier les 127-10=117 états restants pour voir comment ils se connectent sur notre cycle. C'est un travail important qu'il est impossible de réaliser à moins d'y passer 117 soirées (à raison d'une transition par soirée) soit presque 4 mois !!!

Pour éviter cela on va prévoir une entrée d'initialisation appelée Init au cas où à la mise sous tension on se trouve dans un état non prévu. Cette entrée fonctionnera de manière synchrone, lorsqu'elle sera à 1 un front d'horloge provoquera l'affichage du 0 en sortie du compteur.

Écrire maintenant les équations de récurrence trouvées en 2°) en ajoutant convenablement l'entrée Init.

3) Vous êtes prêt maintenant à écrire le programme VHDL complet. Écrire ces équations dans le langage VHDL puis transformer le VHDL en symbole. Ajouter dans le schéma en utilisant toujours "clk_slow(7)" comme horloge

```
1            library IEEE;
1            use IEEE.STD_LOGIC_1164.ALL;
2            ENTITY cmpt7seg IS
3              PORT(CLK_lent : IN STD_LOGIC;
4              a,b,c,d,e,f,g : OUT STD_LOGIC);
5            -- ou  alors : s7segs : out std_logic_vector(6 downto 0));
6            END cmpt7seg;
```

Exo5 : Refaire le même travail avec une entrée d'initialisation.

```
1            library IEEE;
1            use IEEE.STD_LOGIC_1164.ALL;
2            ENTITY cmpt7seg IS
3              PORT(CLK_lent, Init : IN STD_LOGIC;
4              a,b,c,d,e,f,g : OUT STD_LOGIC);
5            -- ou  alors : s7segs : out std_logic_vector(6 downto 0));
6            END cmpt7seg;
```

Exo6 : Refaire le même travail que l'exo5 en utilisant un case when et toujours "clk_slow(7)" comme horloge en transformant le VHDL en symbole.

```
1            library IEEE;
1            use IEEE.STD_LOGIC_1164.ALL;
2            ENTITY cmpt7seg IS
3              PORT(CLK_50MHz : IN STD_LOGIC;
4              s_7segs : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
5            END cmpt7seg;
```

Remarquez l'utilisation du STD_LOGIC_VECTOR au lieu des sorties individuelles. Le fait qu'il soit en OUT implique que l'on utilisera un signal interne pour les "CASE".

TP5 Schématique, VHDL et Compteurs

Association de deux composants VHDL en schématique

Nous vous donnons deux composants VHDL et on vous demande d'en réaliser un assemblage.

Travail à réaliser

Exo1 : On vous donne les deux composants VHDL suivants. On vous demande de les transformer en symbole (en complétant les transitions de « cmpt7seg ») et de les réunir à l'aide de la schématique.

```
1             library IEEE;
2             use IEEE.STD_LOGIC_1164.ALL;
3             use ieee.std_logic_arith.all;
4             use ieee.std_logic_unsigned.all;
5             ENTITY cmpt24bits IS
6             PORT(clk_50MHz : IN STD_LOGIC;
7               clk_slow : OUT STD_LOGIC);
8             END cmpt24bits;
9
10            ARCHITECTURE acmpt24bits OF cmpt24bits IS
11              signal cmpt : std_logic_vector(23 downto 0);
12            BEGIN
13              process(clk_50MHz) begin
14                if rising_edge(clk_50MHz) then
15                  cmpt <= cmpt + 1;
16                end if;
17              end process;
18              clk_slow <= cmpt(23);
19            END acmpt24bits;
20
21
22            library IEEE;
23            use IEEE.STD_LOGIC_1164.ALL;
24            ENTITY cmpt7seg IS
25            PORT(CLK : IN STD_LOGIC;
26              s_7segs : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
27            END cmpt7seg;
28            ARCHITECTURE arch OF cmpt7seg IS -- comment éviter les equations
29              SIGNAL s7segs : STD_LOGIC_VECTOR(6 DOWNTO 0);
30            BEGIN
31              PROCESS(clk) BEGIN
32                IF (clk'EVENT AND clk='1') THEN
33                 CASE s7segs is --style case when
34                   WHEN "0000001" => s7segs <="???????"; -- premiere transition
35                   WHEN "1001111" => s7segs <="???????"; -- deuxieme transition
36                   WHEN "0010010" => s7segs <="???????"; -- troisieme transition
37                   WHEN "0000110" => s7segs <="???????";
38                   WHEN "1011100" => s7segs <="???????";
39                   WHEN "0100100" => s7segs <="???????";
40                   WHEN "0100000" => s7segs <="???????";
41                   WHEN "0001111" => s7segs <="???????";
42                   WHEN "0000000" => s7segs <="???????";
43                   WHEN OTHERS => s7segs <="0000001"; -- dernière transition
44                 END CASE;
45               END IF;
46              END PROCESS;
47              s_7segs <=s7segs;
48            END arch;
```

Nous allons reprendre le problème du compteur sept segments mais de façon plus normale.

Compteur à sortie sept segments sur deux digits (schématique)

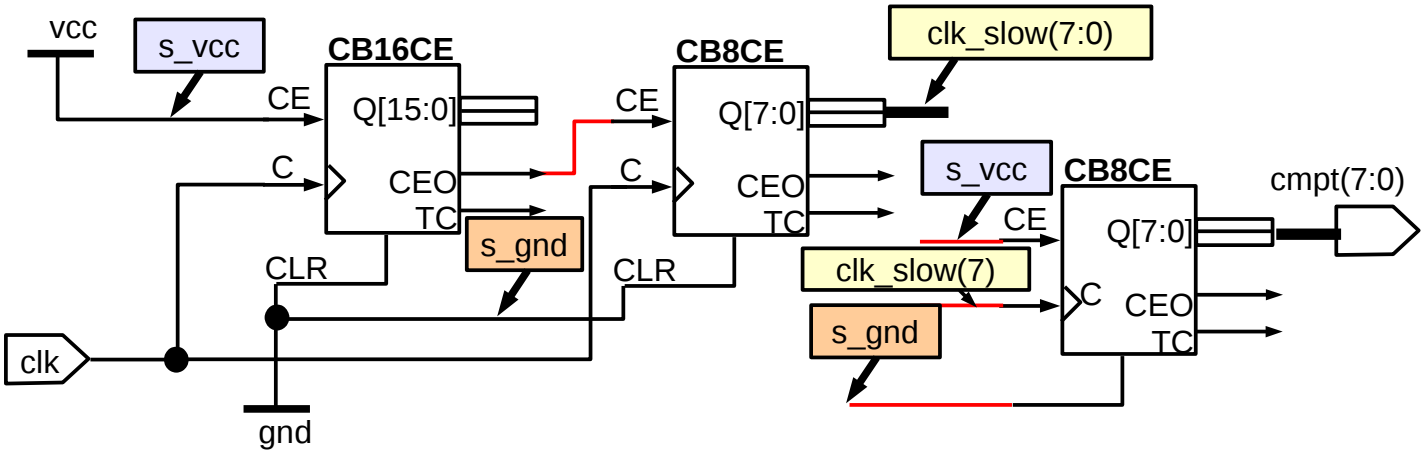
Cette partie doit être réalisée avec les afficheurs sept segments reliés à la carte.

Nous allons réaliser cet exercice en plusieurs étapes.

Exo2 : Nous désirons implanter un ensemble composé d'un compteur 8 bits et sortant sa valeur sur deux afficheurs sept segments (affichage de 00 à FF)... Attention, le poids faible doit être à droite !!!

1°) Réaliser d'abord un compteur simple 8 bit associé à une division par 2^{24} et sortant directement sur des LEDs.

Voici le schéma de principe qui est très simple à réaliser pour vous si vous avez sauvé l'exercice 1 du TP précédent (sinon tant pis pour vous) :

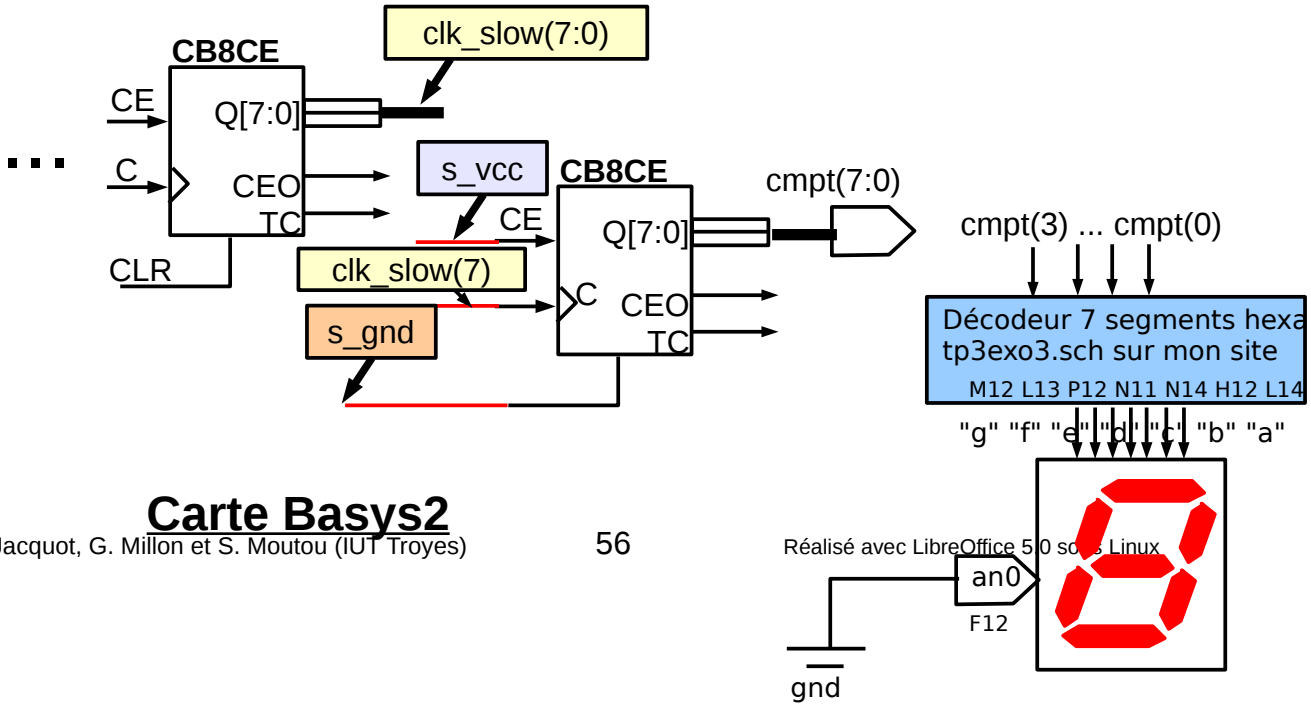


La seule entrée est clk qui se trouve en "B8" sur le FPGA. Les 8 sorties sont reliées aux 8 LEDs de la carte. Remarquez que le seul compteur qui nous intéresse est le CB8CE de droite, les deux autres étant là pour ralentir l'horloge.

2°) Réaliser maintenant l'affichage des 4 bits de poids faibles sur un seul afficheur. Le transcodeur "Décodeur 7 segments hexa" est toujours disponible sur mon site par le lien :

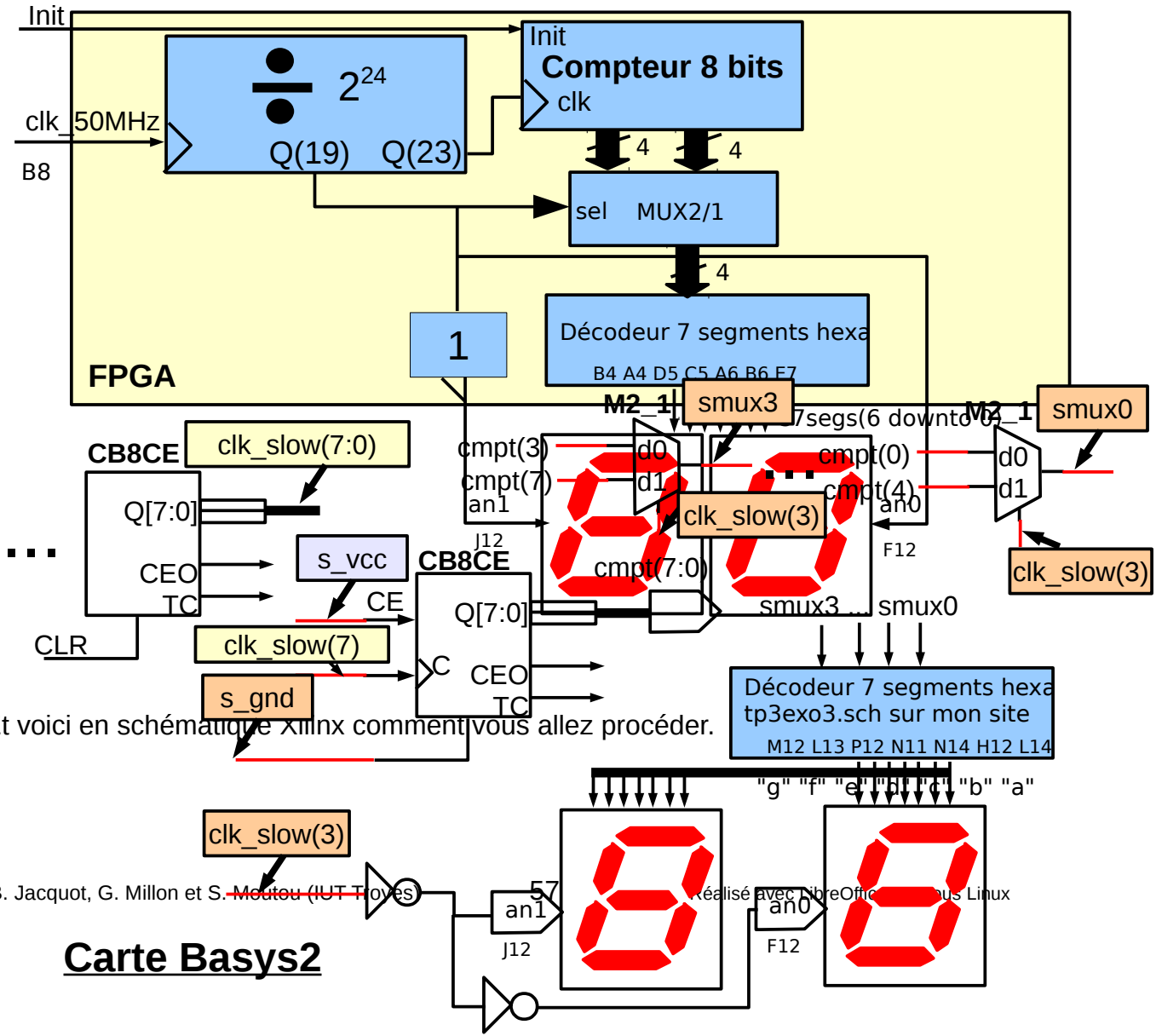
http://moutou.pagesperso-orange.fr/tp3exo3.sch

Il manque naturellement à gauche du dessin ci-dessous le compteur 16 bits qui doit rester dans votre schématique.



Carte Basys2

3°) Réaliser maintenant l'affichage des 8 bits du compteur sur deux afficheurs. On utilisera 4 multiplexeurs **M2_1** pour réaliser le multiplexeur et encore un composant **CB8CE** pour le compteur 8 bits. Voila en principe ce que l'on cherche à faire :



Et voici en schématique Xilinx comment vous allez procéder.

Carte Basys2

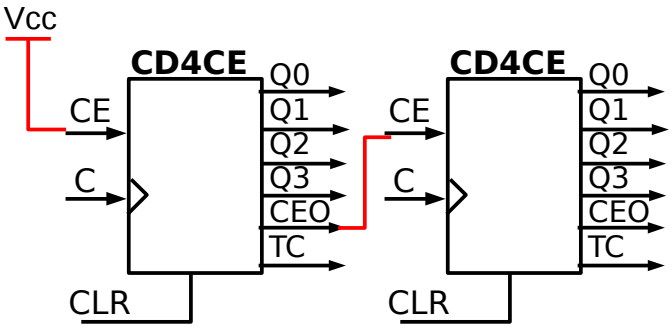
Renommer les entrées du transcodeurs de la question 2°) doit être fait avec délicatesse. Le mieux est peut être d'effacer le transcodeur et d'en remettre une version fraîchement téléchargée.

Compteur BCD sur deux digits (schématique)

Nous cherchons maintenant à modifier le compteur de l'exercice précédent (exo2) pour qu'il affiche de 00 à 99.

Travail à réaliser

Exo3 : Les compteurs BCD 4 bits **CD4CE** seront cascadés pour fournir un compteur BCD sur 8 bits (deux digits). Ces deux compteurs remplacent le compteur **CB8CE** qui fournit "cmpt(7:0)". Prévoir les tests correspondants à partir du schéma de l'exercice 2.



Compteur/décompteur BCD sur deux digits à partir du VHDL donné

Exo4 : Réaliser un compteur/décompteur BCD 4 bits en Prévoir les tests correspondants à partir du schéma de l'exercice 2.

Indication

Une aide pour réaliser un compteur décompteur BCD cascadable peut être trouvée ici :

http://fr.wikibooks.org/wiki/TD5_VHDL_et_CAO#El.C3.A9ments_d.27implantation_en_VHDL

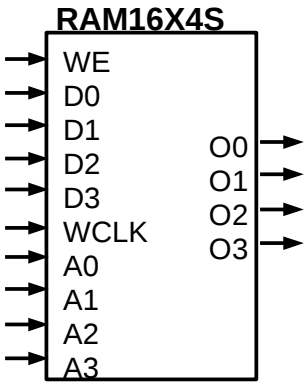
TP6 Mémoires et compteurs

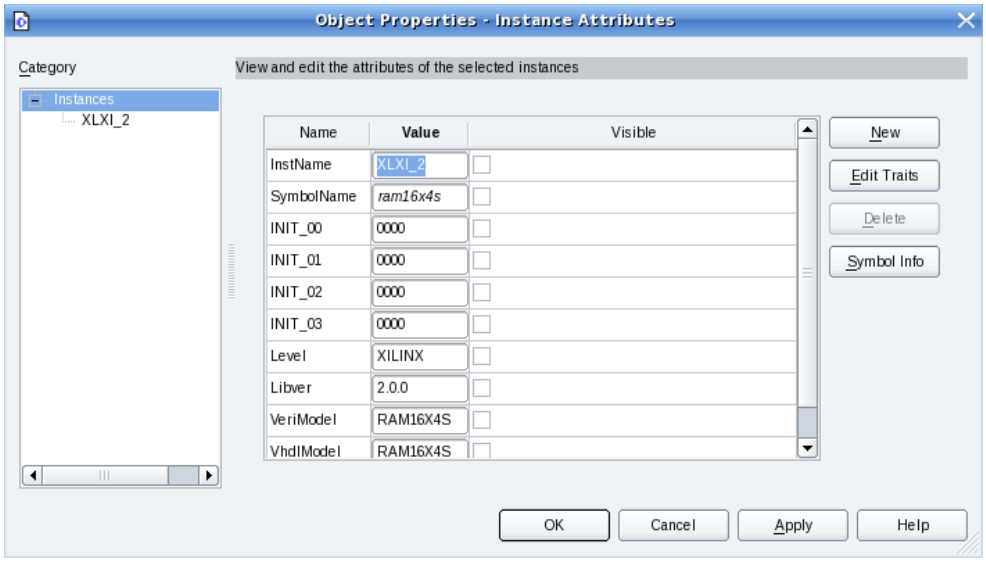
Compteur à sortie sept segments avec mémoire

Nous allons utiliser un compteur normal, c'est à dire qui compte en binaire sur 4 bits suivi d'une mémoire pour le transcodage.

Présentation des mémoires utilisées

Pour la mémoire RAM16X4S, regardez attentivement la fenêtre de dialogue qui s'ouvre quand on clique sur "Object properties" de la RAM correspondante. Vous en déduirez la façon de l'initialiser.





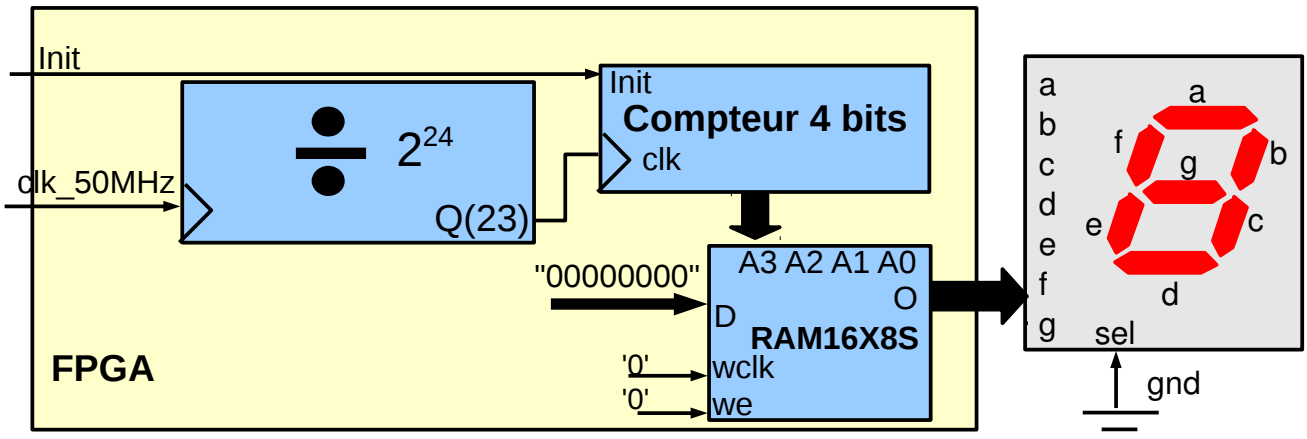
Travail à réaliser (schématique)

Exo1 : Calculer les valeurs hexadécimales contenues dans cette ROM **RAM16X8S** pour réaliser le transcodage qui convient.

Utiliser comme compteur 4 bits un **CB4CE**.

Reprendre le diviseur de fréquence par 2**24 (**CB16CE** cascadé avec **CB8CE**).

Assembler le tout.



Texte défilant

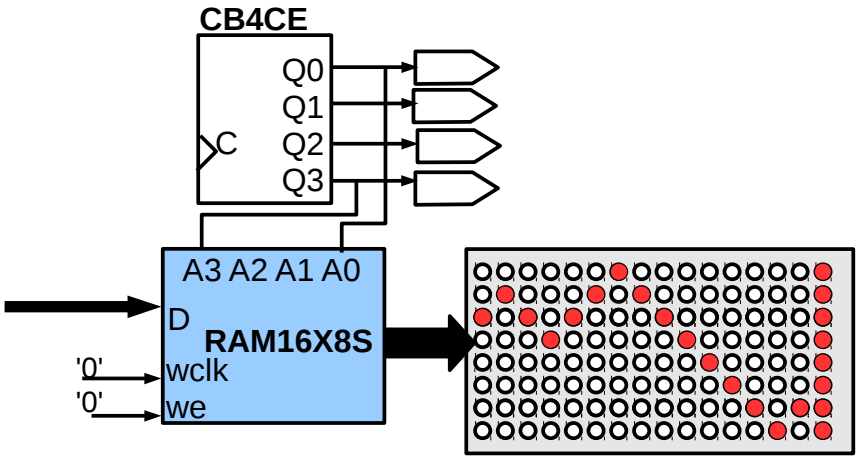
Exo2 :Donner le contenu de la RAM pour afficher le texte : "bonjour A touS" qui défile sur un afficheur. Réaliser l'ensemble et tester. Faire constater.

Chenillard sur LEDs (Schématique)

Exo3 : 1°) On reprend l'exercice précédent mais on sort maintenant sur 8 LEDs. On vous demande de réaliser un chenillard aller et retour. Il faut donc changer le fichier ucf et le contenu de la **RAM16X8S**.

2°) DS 2014

Les sorties précédentes sont maintenant utilisées comme adresses d'une **RAM16x8s** comme dans la figure ci-contre. Donner le contenu de la RAM pour afficher le chenillard ci-contre.
Réaliser l'ensemble et tester. Faire constater.



Chenillar sur LEDs (VHDL et schématique)

Le schéma de principe est identique au schéma de l'exercice précédent mais sera fait complètement en VHDL. Une façon de procéder consiste à remplacer petit à petit chacun des éléments de la schématique de l'exercice 1 par un programme VHDL en en faisant un symbole.

Exo4 : En partant de l'exercice 2, réaliser le diviseur par 2^{24} en VHDL, le transformer en symbole de schématique et insérer le tout dans le schéma.

Tester et faire valider.

Barregraphe sur LEDs (Schématique)

Exo5 : On désire afficher une valeur de 0 (tout éteint) à 8 (tout allumé) sur les 8 LEDs. Toute valeur supérieure à 8 affichera 8 (effet de saturation).

1°) Expliquer pourquoi on doit prendre une **RAM16X8S**.

2°) Calculer son contenu et tester avec le même schéma que exo2.

3°) Implanter et tester en prenant un compteur **CB4CE** en le faisant compter de 0 à 8 seulement. On rajoutera donc une LUT4 correctement initialisée sur le CLR du **CB4CE**.

4°) On complique maintenant le compteur. Prendre un compteur/décompteur **CB4CLED** qui remplacera le **CB4CE** et la LUT4. (D0, D1, D2, D3 et L ne seront pas utilisées et donc reliées à la masse).