Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Architecture : Circuits numériques et éléments d'architecture

1ère année

Année scolaire 2016–2017

Consignes

Les exercices de ce recueil sont classés en 4 catégories :

- Les exercices avec la mention "*Préparation*" sont proposés pour vous aider à préparer les séances de TDs ; il est recommandé de résoudre ces exercices avant les séances de TD.
- Les exercices avec la mention "*Pour aller plus loin...*" sont proposés pour vous aider à consolider les notions vues pendant les séances de TD ; il est recommandé de résoudre ces exercices après la séance de TD.
- Les exercices avec la mention "*Méthodologie*" sont étudiés en séance. Leur correction est fournie dans l'enoncé.
- Les exercices sans mention sont ceux qui seront étudiés pendant les séances ; il est recommandé de préparer ces exercices avant la séance de TD.

Toutes les questions traitées en séance disposent d'une correction à la fin du fascicule. L'enseignant vous donnera le numéro correspondant à chaque question.

Grenoble INP ensimag

Table d'association questions/réponses

1 td1

Question td1-1-1 -> Réponse 36
Question td1-1-2 -> Réponse 60
Question td1-1-3 -> Réponse 21
Question td1-1-4 -> Réponse 38
Question td1-2-1 -> Réponse 66
Question td1-2-2 -> Réponse 34
Question td1-2-3 -> Réponse 30
Question td1-2-4 -> Réponse 51
Question td1-4-1 -> Réponse 4
Question td1-4-2 -> Réponse 26
Question td1-4-3 -> Réponse 25
Question td1-4-4 -> Réponse 3
Question td1-5-1 -> Réponse 28
Question td1-5-2 -> Réponse 35

2 td2

Question td2-1-1 -> Réponse 43
Question td2-1-2 -> Réponse 45
Question td2-1-3 -> Réponse 64
Question td2-2-1 -> Réponse 17
Question td2-2-2 -> Réponse 20
Question td2-2-3 -> Réponse 27
Question td2-3-1 -> Réponse 23
Question td2-3-2 -> Réponse 61
Question td2-3-3 -> Réponse 8

3 td3

Question td3-1-1 -> Réponse 15
Question td3-1-2 -> Réponse 7
Question td3-1-3 -> Réponse 47
Question td3-1-4 -> Réponse 10
Question td3-1-5 -> Réponse 62
Question td3-2-1 -> Réponse 16
Question td3-2-2 -> Réponse 56
Question td3-2-3 -> Réponse 1
Question td3-3-1 -> Réponse 44
Question td3-3-2 -> Réponse 68
Question td3-3-3 -> Réponse 13

4 td4

Question td4-1-1 -> Réponse 49
Question td4-1-2 -> Réponse 55
Question td4-1-3 -> Réponse 67
Question td4-1-4 -> Réponse 12
Question td4-2-1 -> Réponse 22
Question td4-2-2 -> Réponse 14
Question td4-3-1 -> Réponse 32
Question td4-3-2 -> Réponse 54

5 td5

Question td5-2-1 -> Réponse 46
Question td5-2-2 -> Réponse 41
Question td5-2-3 -> Réponse 19
Question td5-3-1 -> Réponse 59
Question td5-3-2 -> Réponse 31
Question td5-3-3 -> Réponse 6

6 td6

Question td6-1-1 -> Réponse 53
Question td6-1-2 -> Réponse 9
Question td6-1-3 -> Réponse 58
Question td6-1-4 -> Réponse 24
Question td6-1-5 -> Réponse 11

7 td7

Question td7-1-1 -> Réponse 52
Question td7-1-2 -> Réponse 57
Question td7-1-3 -> Réponse 29
Question td7-1-4 -> Réponse 5
Question td7-1-5 -> Réponse 63

8 td8

Question td8-0-1 -> Réponse 39
Question td8-0-2 -> Réponse 65
Question td8-0-3 -> Réponse 71
Question td8-0-4 -> Réponse 33
Question td8-0-5 -> Réponse 72
Question td8-0-6 -> Réponse 18

9 td9

Question td9-0-1 -> Réponse 50
Question td9-0-2 -> Réponse 70
Question td9-0-3 -> Réponse 42

10 td10

Question td10-0-1 -> Réponse 48
Question td10-0-2 -> Réponse 37

11 td11

Question td11-2-1 -> Réponse 40
Question td11-2-2 -> Réponse 69
Question td11-2-3 -> Réponse 2

TD 1
Portes de base et minimisations de fonctions booléennes

Préparation
Ex. 1 : Codage des entiers naturels

Question 1 Remplir un tableau contenant les valeurs en base 2, en base 10 et en base 16 des entiers naturels entre 0 et 15 (inclus).

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Décimal	Binaire	Hexadécimal
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Notation : lorsque la base est ambiguë, on la note en indice du nombre. Par exemple, $9_{10} = 1001_2 = 9_{16}$. On utilise aussi souvent la notation du langage C pour les nombres en hexadécimal : $0x9 = 9_{16}$.

Question 2 Comment peut-on calculer facilement une valeur en hexadécimal à partir de sa valeur en binaire ? Convertir en hexadécimal la valeur binaire 101101011100_2.

Pour convertir du binaire à l'hexadécimal, il suffit de regrouper la valeur binaire 4 bits par 4 bits, car il faut exactement 4 bits pour coder un chiffre hexadécimal : $\underbrace{1011}_{B}\underbrace{0101}_{5}\underbrace{1100}_{C}$.

On appelle *nibble* ou « demi-octet » une agrégation de 4 bits.

Question 3 Faire les additions suivantes en binaire : $44 + 21$ et $200 + 100$ sur 8 bits [1]. Que constatez-vous pour le résultat de la 2e addition ? Comment peut-on détecter ce phénomène ?

$44_{10} + 21_{10} = 65_{10}$:

		0	0	1	0	1	1	0	0
+		0	0	0	1	0	1	0	1
			1	1	1	1				*retenues*
		0	1	0	0	0	0	0	1	

$200_{10} + 100_{10} = 300_{10}\ mod\ 256_{10} = 44_{10}$:

		1	1	0	0	1	0	0	0
+		0	1	1	0	0	1	0	0
	1	1								*retenues*
		0	0	1	0	1	1	0	0

Le résultat de l'addition $200 + 100$ ne tient pas sur 8 bits, il y a donc un dépassement de capacité. Pour les entiers naturels, on détecte ce dépassement grâce à la retenue sortante. On note que si on fait la même addition sur 9 bits, on trouve le résultat correct.

1. C'est à dire que les opérandes et le résultat de l'opération sont codés sur 8 bits.

Question 4 Quel intervalle d'entiers naturels peut-on coder sur n bits ? Combien de bits faut-il pour coder m valeurs différentes (avec $m \geq 2$) ?

On va utiliser la notation suivante :
$\lceil x \rceil$ représente le plus petit entier supérieur ou égal au réel x
$\lfloor x \rfloor$ représente le plus grand entier inférieur ou égal au réel x

Autrement dit :

— $\lceil x \rceil = min\{n \in \mathbb{N} \mid x \leq n\}$

— $\lfloor x \rfloor = max\{n \in \mathbb{N} \mid n \leq x\}$

Sur n bits, on peut coder tous les entiers dans $[0 \,..\, 2^n - 1]$. Il faut donc $\lceil \log_2(m) \rceil$ bits pour coder m valeurs différentes, mais il faut $\lfloor log_2(m) \rfloor + 1$ bits pour coder la valeur m.

Attention à ne pas confondre « combien de bits pour coder m valeurs différentes » et « combien de bits pour coder la valeur m en binaire ».

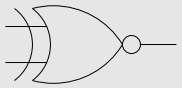
Préparation

Ex. 2 : Circuit comparateur d'entiers

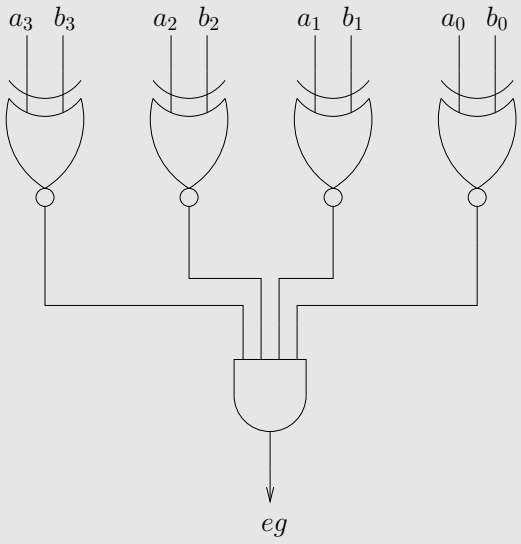
Dans cet exercice, on travaille sur des entiers A et B codés sur 4 bits, ce qu'on notera $A = a_3a_2a_1a_0$ et $B = b_3b_2b_1b_0$. Pour construire les circuits demandés, on suppose qu'on dispose de portes avec au maximum 4 entrées.

Question 1 Quelle porte élémentaire produit 1 en sortie ssi ses deux entrées sont égales ?

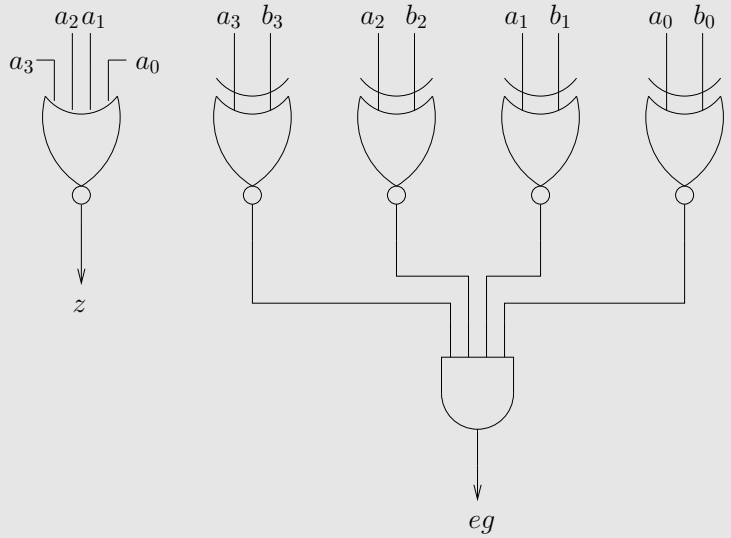
Il s'agit du XNOR, aussi appelé opérateur d'équivalence.



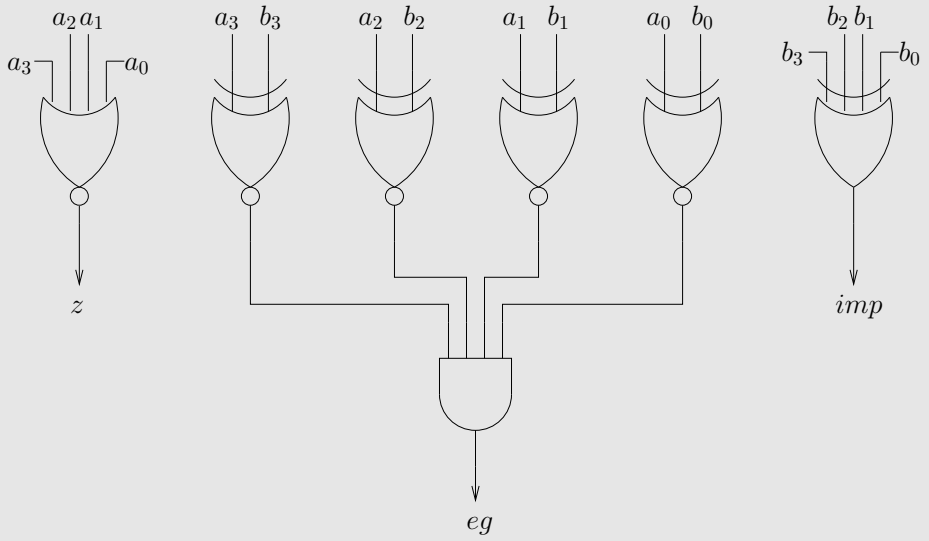
Question 2 Construire un circuit prenant en entrée 2 entiers A et B et produisant une sortie eg valant 1 ssi ces 2 entiers sont égaux.



Question 3 Etendre ce circuit pour ajouter une sortie z valant 1 ssi A vaut 0.



Question 4 Ajouter une sortie imp valant 1 ssi le nombre de bits composant B et valant 1 est impair. Par exemple, si $B = 5_{10} = 0101_2$ alors $imp = 0$ et si $B = 14_{10} = 1110_2$ alors $imp = 1$.



Méthodologie

Ex. 3 : Conception et minimisation de circuits combinatoires sur un exemple élémentaire

On travaille sur la fonction majorité de trois variables valant "vrai" ssi la majorité des paramètres de la fonction valent 1.

Table de vérité de la fonction Majorité

a	b	c	$Maj(a,b,c)$	termes canoniques
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{a}.b.c$
1	0	0	0	
1	0	1	1	$a.\overline{b}.c$
1	1	0	1	$a.b.\overline{c}$
1	1	1	1	a.b.c

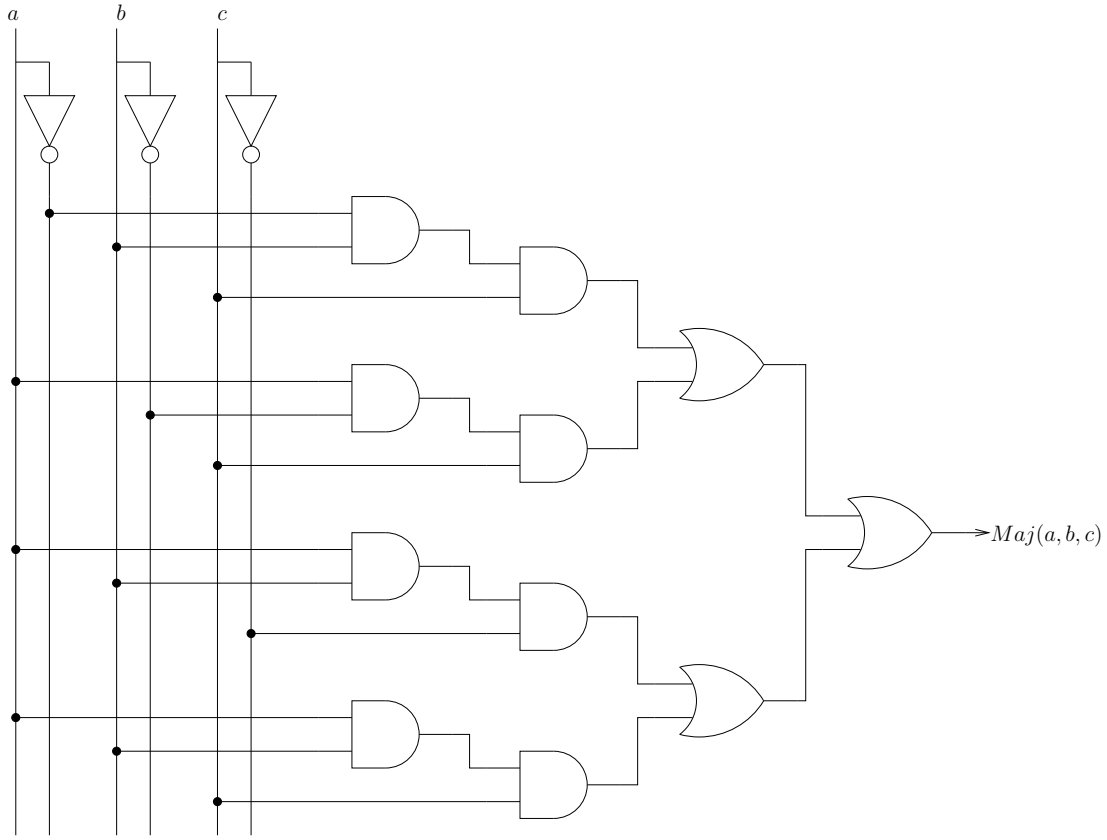
N.B : un "terme canonique" est le produit (AND) de toutes les variables de la fonction sous forme normale (a) ou complémentée ($\overline{a}$). On recherche une expression somme de produits donc on ne s'intéresse qu'aux termes correspondants aux 1 de la fonction.

Expression booléenne canonique de la fonction $Maj(a,b,c)$

C'est l'expression de la fonction $Maj(a,b,c)$ en fonction des paramètres a, b et c, sans chercher à minimiser cette expression. C'est donc la somme des termes canoniques déduits de la table de vérité : $Maj(a,b,c) = \overline{a}.b.c + a.\overline{b}.c + a.b.\overline{c} + a.b.c.$

Circuit implantant cette fonction

Pour dessiner le circuit implantant cette fonction, on n'utilisera ici que des portes de bases AND, OR à 2 entrées ainsi que des portes INV.



Minimisation de l'expression de la fonction

On montre ici comment on peut minimiser l'expression d'une fonction en utilisant le calcul booléen. On part de :

$$Maj(a,b,c) = \overline{a}.b.c + a.\overline{b}.c + a.b.\overline{c} + a.b.c$$

On remarque que les simplifications suivantes sont possibles, grâce à la règle $\overline{x}.y + x.y = y$:

$$\overline{a}.b.c + a.b.c = b.c$$

$$a.\overline{b}.c + a.b.c = a.c$$

$$a.b.\overline{c} + a.b.c = a.b$$

Faut-il choisir une (et une seule) de ces simplifications ? non, car on a aussi la règle $x + x = x$. On peut donc réécrire l'expression de la fonction Majorité comme ci-dessous :

$$\begin{aligned} Maj(a,b,c) &= \overline{a}.b.c + a.\overline{b}.c + a.b.\overline{c} + a.b.c + a.b.c + a.b.c \\ &= \overline{a}.b.c + a.b.c + a.\overline{b}.c + a.b.c + a.b.\overline{c} + a.b.c \\ &= b.c + a.c + a.b \end{aligned}$$

Tableaux de Karnaugh :
Une technique graphique pour la minimisation d'expressions booléennes

Un tableau de Karnaugh est une table de vérité, présentée de façon à ce que les adjacences du type $\overline{x}.y + x.y = y$ soient mises en évidence.

Le tableau de Karnaugh de la fonction $Maj(a,b,c)$ est donné ci-dessous. Remarquez que l'ordre de l'énumération des variables b et c n'est pas quelconque : une seule des variables change de valeur entre deux colonnes (y compris quand on rapproche la dernière colonne de la première : adjacence circulaire).

Remplir un tableau de Karnaugh revient à remplir une table de vérité. Par exemple, la troisième case de la première ligne correspond à $a = 0$ et $b = 1$ et $c = 1$: $Maj(0,1,1) = 1$ donc la case est remplie avec un 1.

a \ bc	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	1	1

Les groupements de points à 1 mettent en évidence les simplifications possibles. Il suffit alors de choisir un nombre minimal de regroupements qui couvrent tous les points à 1 de la fonction.

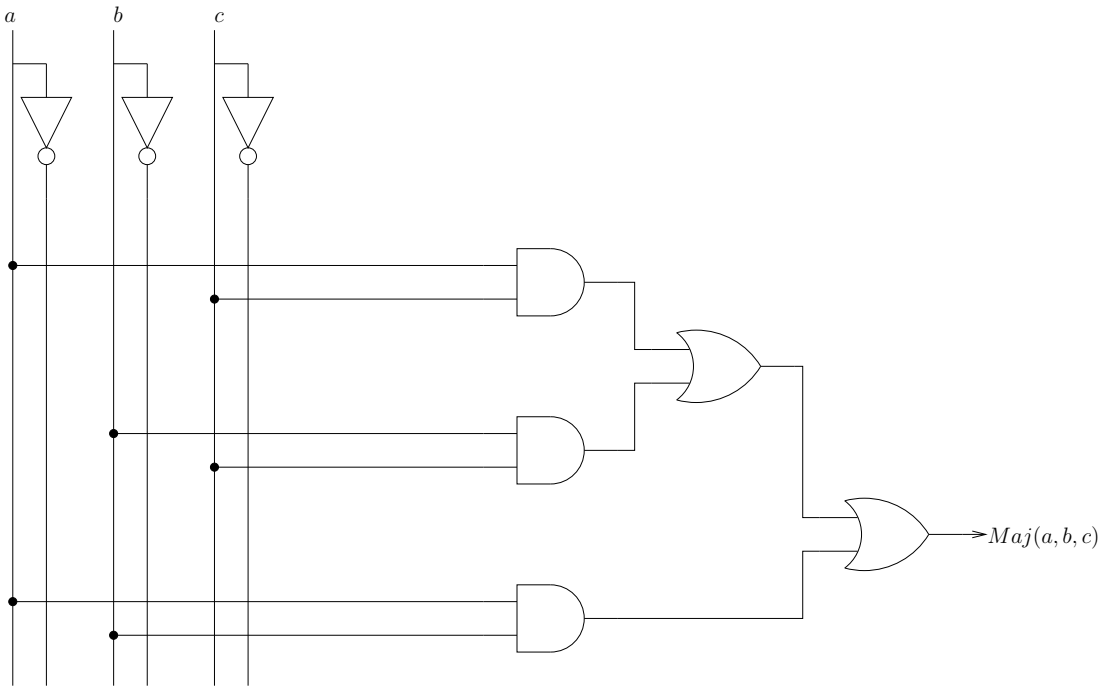
Pour chaque groupe de cases à 1, on extrait un terme construit à partir des variables qui ne changent pas de valeur pour l'ensemble des points regroupés. Par exemple, pour le groupement vertical de la troisième colonne : on remarque que la valeur de a change entre les deux cases regroupées, a n'apparaitra donc pas dans le terme correspondant à ce groupe. Ce qui est commun dans ce groupe ce sont les valeurs de b et c : on extrait donc le terme $b.c$

Sur cet exemple, il y a donc 3 termes qui correspondent aux trois groupements, on retrouve l'expression : $Maj(a,b,c) = b.c + a.c + a.b$.

N.B : Les groupements ne peuvent être que de 2, 4 ou 8 cases (puissances de 2). Par la suite, on utilisera souvent des tableaux avec 4 variables en entrées donc 16 cases. Le nombre de variables dans un terme correspond à la taille du groupement. Par exemple, dans un tableau de 16 cases (4 variables), un groupe de 8 cases correspond à un terme d'une variable, un groupe de 4 à un terme de 2 variables, un groupe de 2 à un terme de 3 variables et si il reste un 1 isolé le terme correspondant utilise les 4 variables.

Circuit minimisé

A partir de l'expression minimisée, on peut dessiner le circuit suivant :



Pourquoi a t on besoin de minimiser ?

On remarque que le circuit obtenu à partir de la forme minimisée a deux avantages : moins de portes au total (5 au lieu de 12) et moins de portes traversées (entre l'entrée et la sortie, au plus 4 portes au lieu de 5). Ce circuit minimisé est donc moins couteux en porte et plus rapide.

N.B : Chaque porte a un temps de propagation (différent suivant la technologie). Le temps de propagation à travers un circuit est déterminé par le nombre maximal de couches de portes à traverser entre une entrée et une sortie.

Remarque : dans la suite du semestre, nous n'utiliserons que les minimisations de fonctions booléennes à base de tableaux de Karnaugh.

Ex. 4 : Reste de la division entière

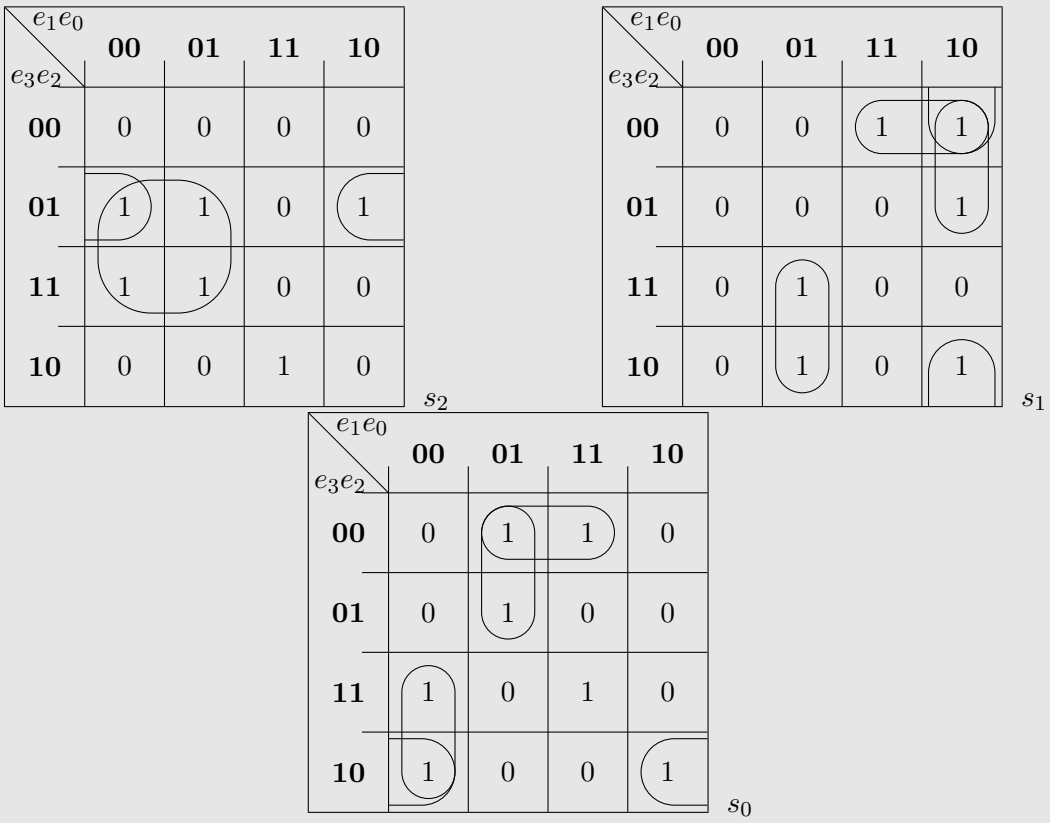
Objectif de l'exercice : premier contact avec les tableaux de Karnaugh, puis utilisation des φ .

Soit un entier naturel E dans l'intervalle [0 .. 15], donc codé sur 4 bits e_3, e_2, e_1, e_0 ; on veut réaliser un circuit qui calcule le reste S de la division entière de E par 7.

Question 1 Sur combien de bits doit être codé S ?

Le reste d'une division par 7 est dans l'intervalle [0..6], donc S est codé sur 3 bits.

Question 2 Représenter les fonctions de sortie du circuit à l'aide de tableaux de Karnaugh.



Question 3 Proposer des expressions minimisées des fonctions de sortie du circuit.

$$
\begin{aligned}
s_2 &= e_2.\overline{e_1} + \overline{e_3}.e_2.\overline{e_0} + e_3.\overline{e_2}.e_1.e_0 \\
s_1 &= e_3.\overline{e_1}.e_0 + \overline{e_3}.\overline{e_2}.e_1 + \overline{e_3}.e_1.\overline{e_0} + \overline{e_2}.e_1.\overline{e_0} \\
s_0 &= e_3.\overline{e_1}.\overline{e_0} + e_3.\overline{e_2}.\overline{e_0} + \overline{e_3}.\overline{e_2}.e_0 + \overline{e_3}.\overline{e_1}.e_0 + e_3.e_2.e_1.e_0
\end{aligned}
$$

On considère maintenant que l'entier E en entrée est dans l'intervalle [0 .. 9]. Les fonctions en sortie sont donc des fonctions Φ - booléennes.
Méthodologie : Pour toutes les cases qui correspondent à une valeur d'entrées non utilisée (ici, les cases correspondant aux valeurs de 10 à 15), la valeur de la sortie est indifférente. On note Φ dans les cases correspondantes : Φ veut dire 0 ou 1, au choix du concepteur. On pourra donc utiliser les Φ pour simplifier davantage l'expression de la fonction (s'ils permettent d'obtenir de plus gros groupements de cases).

Question 4 Modifier les tableaux de Karnaugh obtenus en question 2 et proposer des expressions minimisées des fonctions de sortie du circuit.

e_3e_2 \ e_1e_0	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	Φ	Φ	Φ	Φ
10	0	0	Φ	Φ

s_2

e_3e_2 \ e_1e_0	00	01	11	10
00	0	0	1	1
01	0	0	0	1
11	Φ	Φ	Φ	Φ
10	0	1	Φ	Φ

s_1

e_3e_2 \ e_1e_0	00	01	11	10
00	0	1	1	0
01	0	1	0	0
11	Φ	Φ	Φ	Φ
10	1	0	Φ	Φ

s_0

$$
\begin{aligned}
s_2 &= e_2.\overline{e_1} + e_2.\overline{e_0} \\
s_1 &= e_3.e_0 + \overline{e_2}.e_1 + e_1.\overline{e_0} \\
s_0 &= e_3.\overline{e_0} + \overline{e_3}.\overline{e_2}.e_0 + \overline{e_3}.\overline{e_1}.e_0
\end{aligned}
$$

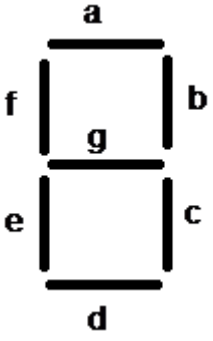
Pour aller plus loin...

Question 5 Dessiner le circuit à base de portes AND, OR et INV à partir des équations de la question 3. Peut-on diminuer la complexité de ce circuit en utilisant un même groupement pour plusieurs sorties ?

à faire en TD : au minimum mentionner la réutilisation de monôme/groupement

Ex. 5 : Afficheur 7-segments

Un afficheur 7-segments est un dispositif d'affichage d'un chiffre décimal composé, comme son nom l'indique, de sept segments pouvant être allumés ou éteints indépendamment les uns des autres. Les segments sont disposés comme illustré ci-dessous :



On considère que l'affichage est réalisé comme suit :

Le but de cet exercice est de concevoir un circuit prenant en entrée un chiffre décimal et produisant en sortie les sept sorties booléennes correspondant aux segments de l'afficheur.

Question 1 Sur combien de bits doit-être codée l'entrée ? Quelle politique adopter concernant les valeurs superflues ?

L'entrée est dans l'intervalle [0..9] donc elle doit être codée sur 4 bits e_3, e_2, e_1 et e_0. On considère que les configurations d'entrées correspondant aux valeurs supérieures à 9 ne peuvent jamais arriver, ce qui permet d'utiliser des Φ-*booléens* et donc de mieux simplifier les sorties.

Question 2 En utilisant des tableaux de Karnaugh, donner les expressions minimisées des 7 segments.

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	1	0	1	1
01	0	1	1	1
11	Φ	Φ	Φ	Φ
10	1	1	Φ	Φ

$a = e_3 + e_1 + \overline{e_2}.\overline{e_0} + e_2.e_0 = e_3 + e_1 + \overline{e_2 \oplus e_0}$

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	0	1	0
11	Φ	Φ	Φ	Φ
10	1	1	Φ	Φ

$b = \overline{e_2} + \overline{e_1}.\overline{e_0} + e_1.e_0 = \overline{e_2} + \overline{e_1 \oplus e_0}$

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	1	1	1	0
01	1	1	1	1
11	Φ	Φ	Φ	Φ
10	1	1	Φ	Φ

$c = e_2 + \overline{e_1} + e_0$

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	1	0	1	1
01	0	1	0	1
11	Φ	Φ	Φ	Φ
10	1	1	Φ	Φ

$d = e_3 + \overline{e_2}.\overline{e_0} + \overline{e_2}.e_1 + e_2.\overline{e_1}.e_0 + e_1.\overline{e_0}$

e_3e_2 \ e_1e_0	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	Φ	Φ	Φ	Φ
10	1	0	Φ	Φ

$e = e_1.\overline{e_0} + \overline{e_2}.\overline{e_0}$

e_3e_2 \ e_1e_0	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	Φ	Φ	Φ	Φ
10	1	1	Φ	Φ

$f = e_3 + \overline{e_1}.\overline{e_0} + e_2.\overline{e_0} + e_2.\overline{e_1}$

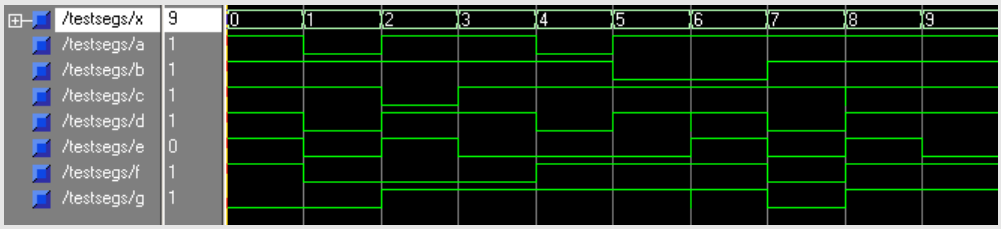
e_3e_2 \ e_1e_0	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	Φ	Φ	Φ	Φ
10	1	1	Φ	Φ

$g = e_3 + e_1.\overline{e_0} + e_2.\overline{e_1} + \overline{e_2}.e_1 = e_3 + e_1.\overline{e_0} + e_2 \oplus e_1$

Pour aller plus loin...

Question 3 Dessiner le circuit correspondant en utilisant des portes logiques de base.

On obtiendra le chronogramme suivant (non demandé) :



Pour aller plus loin...

Ex. 6 : Transcodeur en code de Gray

Objectif de l'exercice : repérer sur les tableaux de Karnaugh les damiers caractéristiques des fonctions XOR et NXOR.

On appelle code de Gray [2] (ou codage binaire réfléchi) un codage binaire ordonné dans lequel le codage ne change que d'un bit entre deux valeurs successives. Le code de Gray sur 2 ou 3 bits est utilisé dans les tableaux de Karnaugh. Soit donc le code de Gray sur 4 bits suivant :

2. Frank Gray, Bell Labs, 1953

Décimal	Gray
0	0000
1	0001
2	0011
3	0010

Décimal	Gray
4	0110
5	0111
6	0101
7	0100

Décimal	Gray
8	1100
9	1101
10	1111
11	1110

Décimal	Gray
12	1010
13	1011
14	1001
15	1000

On veut réaliser un circuit prenant en entrée un entier naturel codé sur 4 bits e_3, e_2, e_1, e_0 et produisant en sortie le code de Gray correspondant s_3, s_2, s_1, s_0.

Question 1 Déterminer les équations minimisées des sorties à l'aide de tableaux de Karnaugh.

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$s_3 = e_3$

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$s_2 = \overline{e_3}.e_2 + e_3.\overline{e_2} = e_3 \oplus e_2$

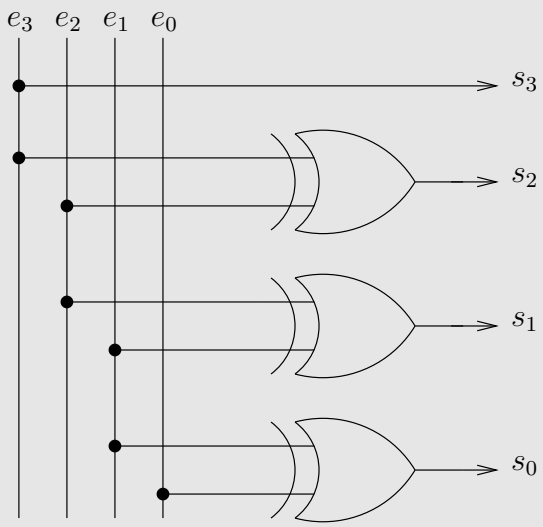
e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$s_1 = e_2.\overline{e_1} + \overline{e_2}.e_1 = e_2 \oplus e_1$

e_3e_2 \ e_1e_0	**00**	**01**	**11**	**10**
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$s_0 = \overline{e_1}.e_0 + e_1.\overline{e_0} = e_1 \oplus e_0$

Question 2 Dessiner le circuit correspondant en utilisant des portes logiques de base.

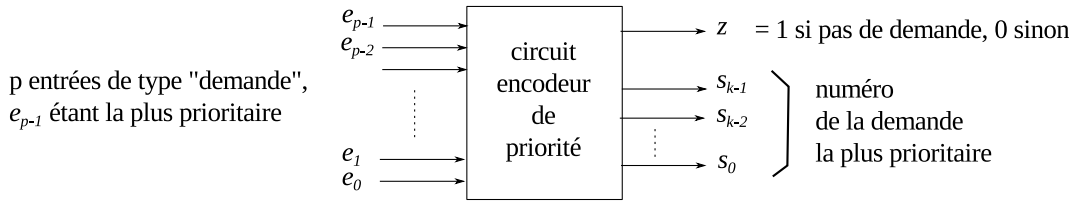


Question 3 Pour apprendre à repérer les expressions à base de XOR, faire le tableau de Karnaugh de $e_3 \oplus e_2 \oplus e_1 \oplus e_0$; de $e_3.(e_2 \oplus e_1 \oplus e_0)$.

Pour aller plus loin...

Ex. 7 : Encodeur de priorité

Objectif de l'exercice : déterminer des expressions à partir de tables de vérité partielles.



Un encodeur de priorité est un circuit qui donne en sortie le numéro (codé en binaire) de l'entrée active la plus prioritaire. Par exemple, si seules les entrées e_5 et e_8 sont à 1, la sortie (bits s_{k-1} à s_0) codera 8 en binaire.

Question 1 Si $p = 2^n$, déterminer le nombre de bits en sortie (la valeur de k). Donner une description formelle du circuit.

$k = n$. Définition formelle : circuit prenant en entrée 2^n bits notés $e_{2^n-1}...e_1e_0$ et produisant en sortie un entier naturel S codé sur n bits et un bit supplémentaire appelé z. On défini les sorties par :

— $z = 1$ et $S = 0$ ssi tous les e_i sont à 0 ;

— $z = 0$ et $S = i$ ssi $e_i = 1$ et $\forall j > i, e_j = 0$.

S est alors simplement l'écriture en binaire de i.

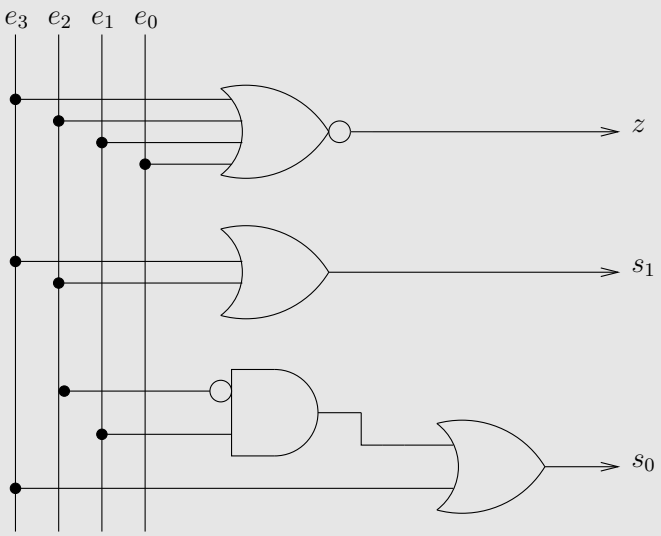
Question 2 Dans le cas $n = 2$, déterminer les équations minimisées de z et $S = s_1s_0$ en remplissant une table de vérité partielle (*i.e.* réfléchir sur les lignes pouvant être factorisées dans la table).

On peut faire une table de vérité partielle :

e_3	e_2	e_1	e_0	s_1	s_0	z
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	1	–	0	1	0
0	1	–	–	1	0	0
1	–	–	–	1	1	0

On a donc :
— $z = \overline{e_3}.\overline{e_2}.\overline{e_1}.\overline{e_0} = \overline{e_3 + e_2 + e_1 + e_0}$ (De Morgan) ;
— $s_1 = \overline{e_3}.e_2 + e_3 = e_3 + e_2$;
— $s_0 = \overline{e_3}.\overline{e_2}.e_1 + e_3 = e_3 + \overline{e_2}.e_1$.

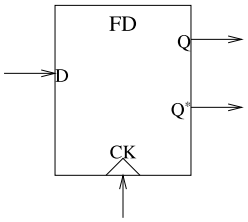
Question 3 Dessiner le circuit correspondant en utilisant des portes logiques de base.



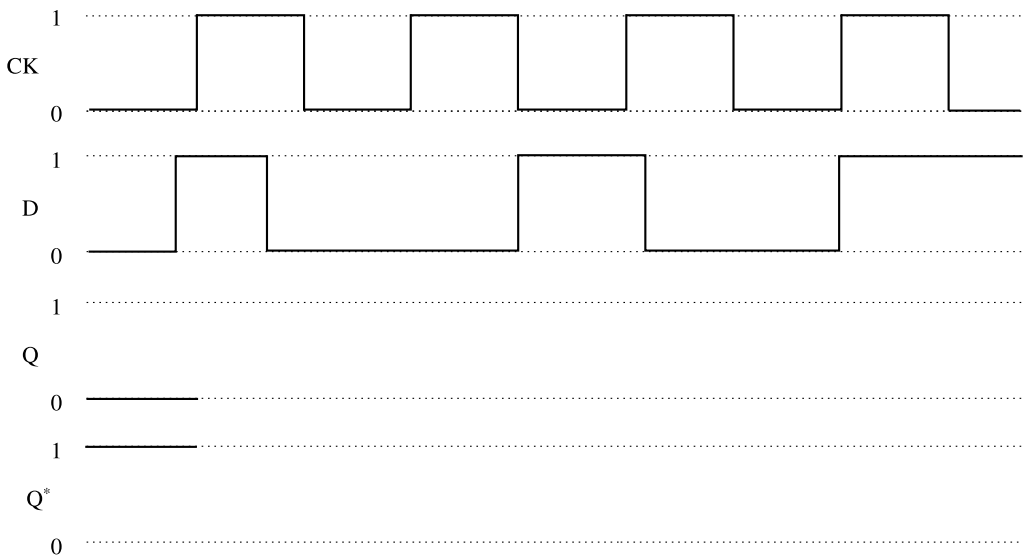
TD 2
Bascules et registres

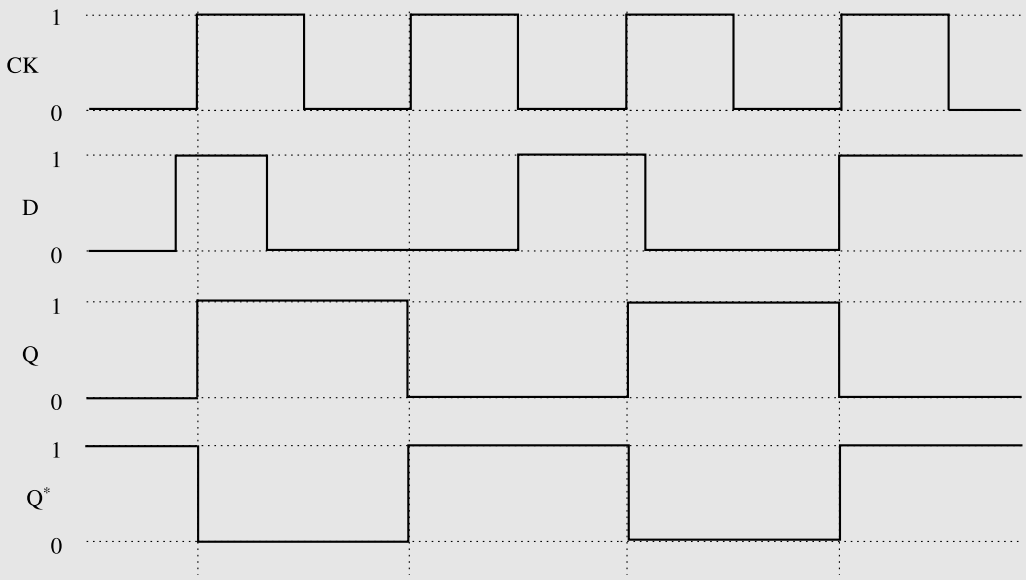
Ex. 1 : Bascule D

La bascule D (*D flip-flop*) illustrée ci-dessous est un dispositif qui permet d'échantillonner l'entrée D sur un front d'une entrée de commande usuellement notée CK, pour *clock* ou horloge. On suppose dans ce qui suit que l'échantillonnage a lieu sur le front montant (*rising edge*) du signal CK.



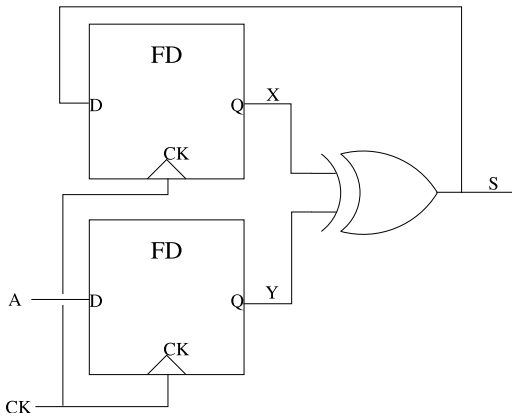
Question 1 Complétez le chronogramme ci-dessous en l'appliquant à une bascule D.





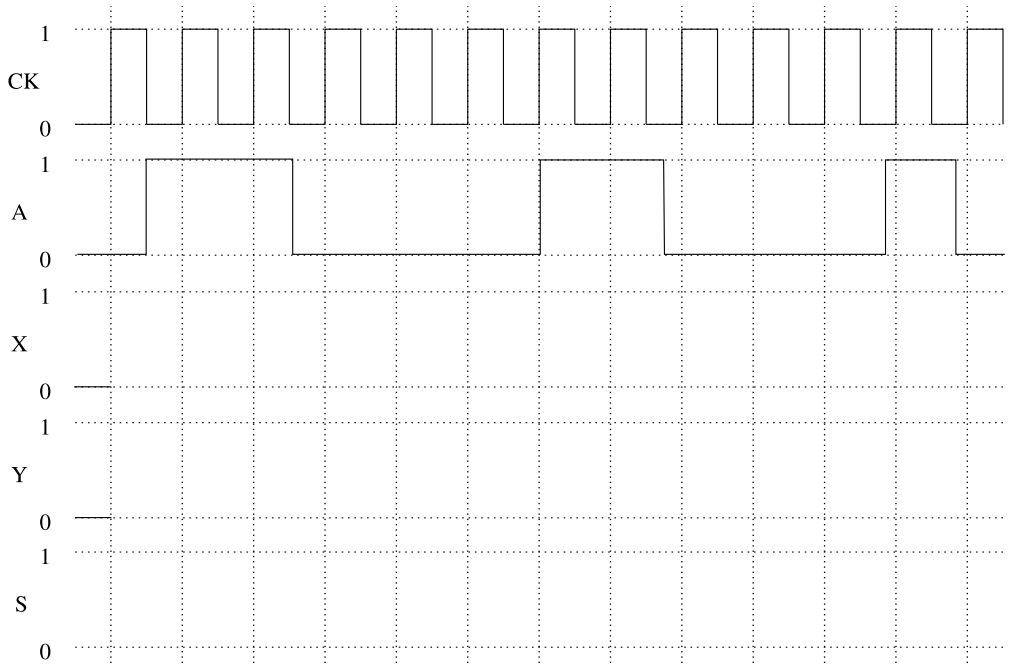
Les seuls instants à considérer sont les fronts montants, pour lesquels on recopie sur la sortie Q la valeur de l'entrée telle qu'elle était stabilisée juste avant le front montant de CK.

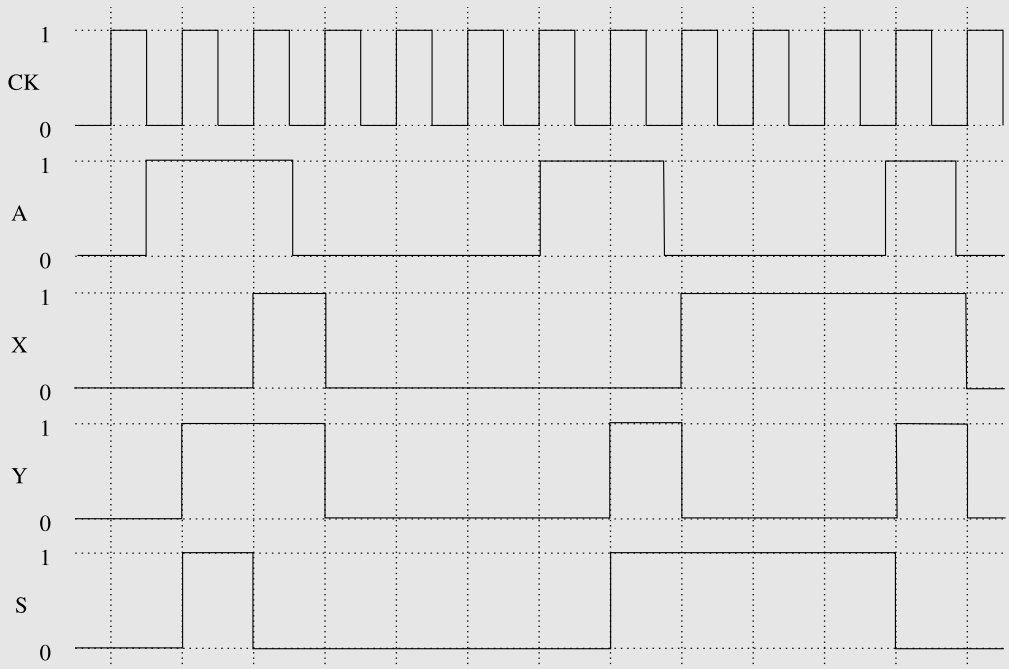
On se propose d'étudier le comportement d'un circuit suivant :



On suppose que l'on force les entrées du circuit comme indiqué sur le chronogramme suivant :

Question 2 Complétez le chronogramme suivant à partir des valeurs fournies sur les entrées du circuit et dans les bascules.





Les points importants à souligner : la bascule va re-synchroniser les signaux qui les attaquent et les faire durer exactement une période de l'horloge. La valeur échantillonnée est celle qui apparaît juste avant le front, ce qui a pour effet de filtrer les signaux les plus brefs (aussi appelés *glitchs*) s'ils n'existent pas au bon moment.

On appelle fréquence maximale de fonctionnement d'un circuit, la fréquence d'horloge au-delà de laquelle il existe un chemin entre deux bascules ayant un temps de propagation supérieur à la période de l'horloge. Ce chemin est généralement appelé "chemin critique".

Supposons que :

— le temps de traversée d'une porte XOR2 soit de 0.1ns
— le temps de traversée d'une bascule D soit de 0.1 ns
— le temps de maintien d'une bascule D soit de 0.01 ns
— le temps de prépositionnement d'une bascule D soit de 0.05 ns

Question 3 Donner le temps de traversé du chemin critique du circuit étudié. En déduire la fréquence maximum de fonctionnement.

$t_p = t_{BD}(CK- > Q) + t_{XOR} + t_{BD}(D- > CK) = 0.1 + 0.1 + 0.05 = 0.25ns$
f_{max}=4GHz

Notez bien que le temps de maintien d'une bascule n'intervient pas dans le calcul du chemin critique. Ce temps sert à vérifier que la donnée en entrée de la bascule est maintenue assez longtemps après le front d'horloge pour réaliser l'acquisition. Pour la bascule du haut, ce temps ne peut pas être violé car l'entrée est générée par 2 signaux synchrones. Comme le temps de traversée d'une bascule D est plus grand que son temps de maintien, les données qui serviront à créer la valeur à l'instant $t+1$ sont encore dans la bascule de départ lorsque la donnée en cours d'acquisition (instant t) franchit le temps de maintien. La bascule du bas prenant sa donnée du monde extérieur, pour s'assurer que le temps de maintien ne sera pas violée, il faut que le signal A ait la même fréquence que la fréquence locale.

Ex. 2 : Construction d'une bascule à écriture conditionnelle

On cherche à présent à réaliser, à partir d'une bascule D et de portes combinatoires simples, une bascule à écriture conditionnelle : une bascule dont l'écriture n'a pas lieu systématiquement à chaque front d'horloge, mais uniquement lorsqu'un autre signal (usuellement nommé WE pour *write enable*) est à 1 au moment du front d'horloge.

Question 1 Quels sont les signaux sur lesquels on peut agir pour implémenter cette écriture conditionnelle ?

— signal CK : le masquer permet d'empêcher une écriture dans la bascule
— signal D : recopier la sortie de la bascule sur le port d'entrée D lorsque WE est à 0.

Question 2 Proposez 2 schémas, l'un utilisant une porte AND et l'autre utilisant un multiplexeur, permettant de contrôler le changement de valeur dans la bascule.

Un schéma du mux est disponible en réponse td3.3.2, si besoin

Schéma 1

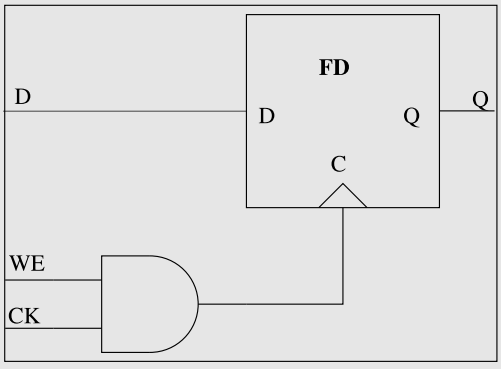
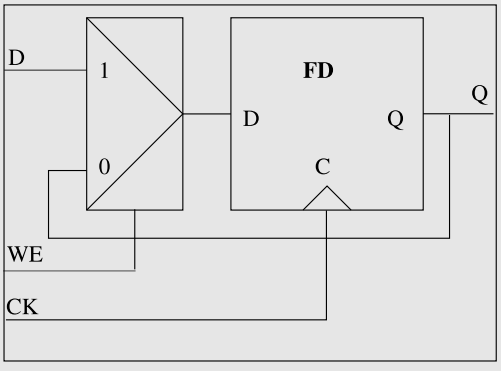
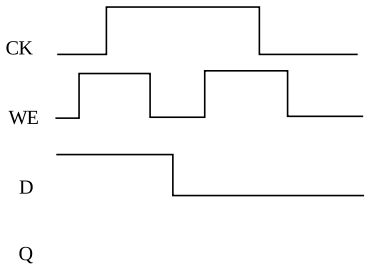


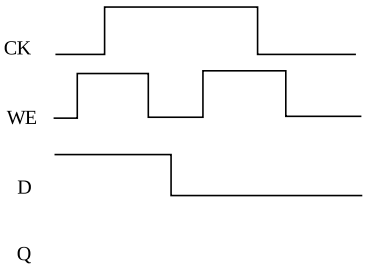
Schéma 2



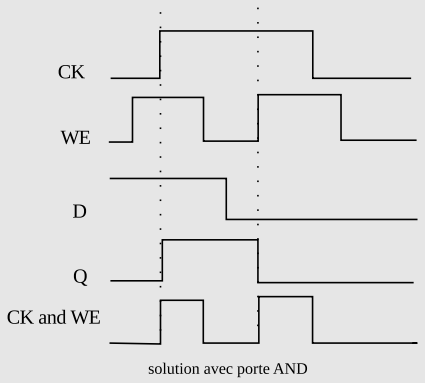
Question 3 Compléter les chronogrammes ci-dessous en indiquant l'évolution de la sortie Q pour chacune des solutions . En déduire les avantages et inconvénients de ces 2 approches vis-à-vis des contraintes temporelles sur le signal d'autorisation d'écriture.



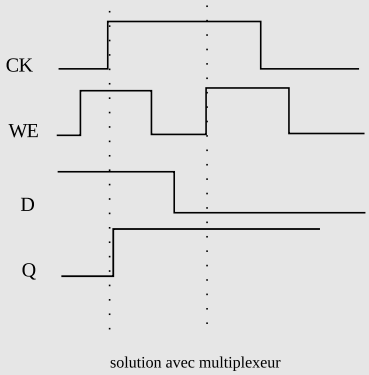
Solution avec porte and



Solution avec multiplexeur



solution avec porte AND



solution avec multiplexeur

L'avantage de la première solution est sa simplicité et sa compacité, mais le chronogramme montre qu'il peut se produire des fronts montants sur C intempestifs, qui déclenchent un échantillonnage de D. Il faut donc garantir que le signal WE ne passe jamais de 0 à 1 lorsque CK est à 1.

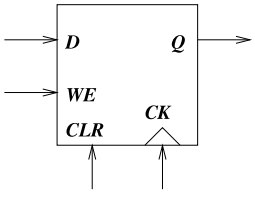
La deuxième solution est plus « sécurisée », car même si WE oscille pendant le niveau haut de l'horloge, la valeur sur l'entrée D ne sera échantillonnée que sur le « vrai » front montant de CK. On a donc la quasi-totalité du cycle pour déterminer si l'écriture doit avoir lieu c'est-à-dire calculer WE, alors que dans le premier cas on n'a que la partie du cycle durant laquelle CK est à 0.

Ex. 3 : Construction de registres

Par définition, un registre est un composant regroupant plusieurs bascules D. Le registre permet donc de mémoriser plusieurs bits dans un même composant.

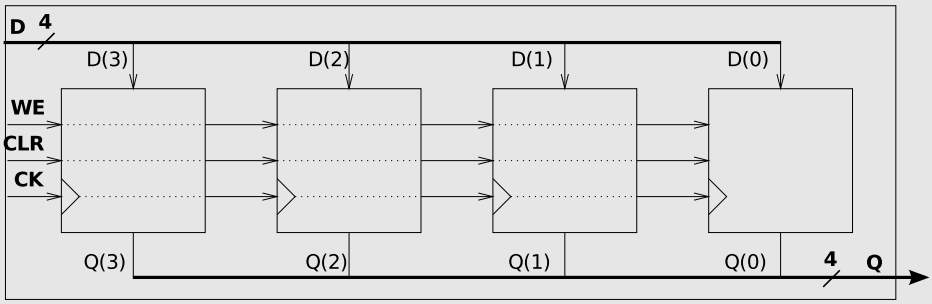
Pour construire des registres, on va utiliser des bascules D avec écriture conditionnelle dont la table de transition est donnée ci-dessous :

CLR	WE	CK	Q
1	–	–	0
0	0	–	Q_{prec}
0	1	↑	D



L'entrée CLR est une entrée de mise à 0 (*clear*) à effet asynchrone (cette entrée force la mise à 0 indépendamment du front d'horloge). L'entrée WE permet d'ignorer les fronts montants de l'horloge (lorsque $WE = 0$) et donc de mémoriser la même valeur sur plusieurs cycles sans avoir à la recharger.

Question 1 On appelle registre à chargement parallèle un registre pour lequel toutes les bascules sont activées en écriture sur le même front d'horloge. Construire un registre 4 bits à chargement parallèle à partir de ces bascules D, le signal WE autorisant le chargement du registre.

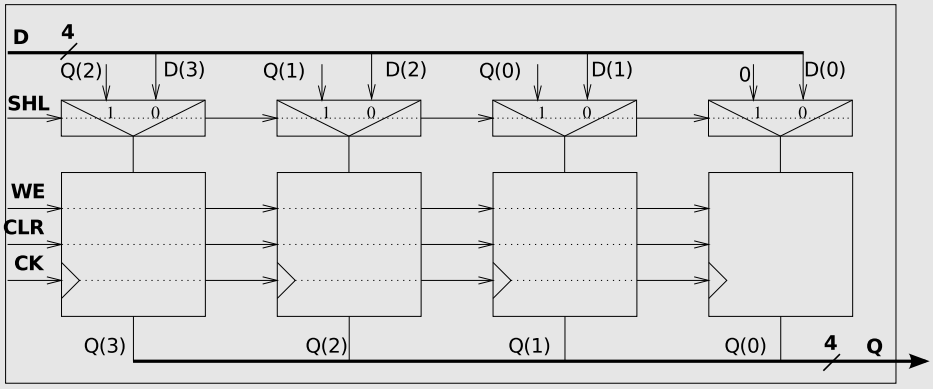


On note bien que les signaux WE, CLR et CK sont communs à toutes les bascules.

Question 2 Construire un registre 4 bits à chargement parallèle à décalage à gauche à partir de ces bascules D et de multiplexeur 2 vers 1. Le registre doit être conforme à la table de transition suivante :

CLR	WE	SHL	CK	Q
1	–	–	–	0
0	0	–	–	Q_{prec}
0	1	0	↑	D
0	1	1	↑	$Q_{prec} << 1$

Où SHL est une entrée booléenne indiquant si le registre doit réaliser un décalage à gauche synchrone, et $A << P$ représente le décalage du nombre A de P bits vers la gauche, avec insertion de 0 à la place des P bits de poids faible. On rappelle que décaler un entier de P bits vers la gauche revient à le multiplier par 2^P.



On note bien que pour effectuer un décalage, on doit activer à la fois SHL et WE : si on n'active que SHL, la nouvelle valeur ne sera pas chargée dans les bascules.

Question 3 Construire un registre 4 bits à chargement parallèle à décalage à droite à partir de ces bascules D et de $MUX_{1b}(2 \rightarrow 1)$. Le registre doit être conforme à la table de vérité suivante :

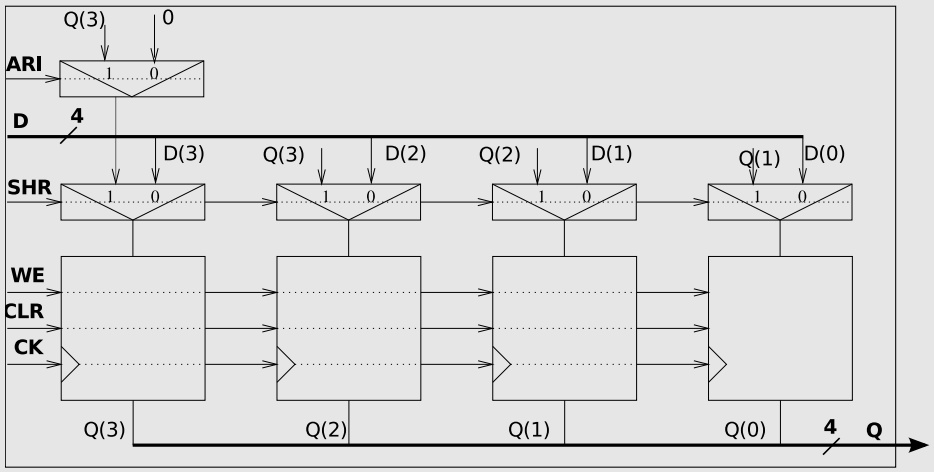
CLR	WE	SHR	ARI	CK	Q
1	–	–	–	–	0
0	0	–	–	–	Q_{prec}
0	1	0	–	↑	D
0	1	1	0	↑	$Q_{prec} >>> 1$
0	1	1	1	↑	$Q_{prec} >> 1$

Où

— $A >>> P$ représente le décalage du nombre A de P bits vers la droite, avec insertion de 0 à la place des P bits de poids forts : c'est le décalage « logique » à droite.

— $A >> P$ représente le décalage du nombre A de P bits vers la droite, avec insertion du bit de signe à la place des P bits de poids forts : c'est le décalage « arithmétique » à droite.

De façon symétrique au décalage à gauche, décaler un entier de P bits vers la droite revient à le diviser par 2^P : le décalage logique a donc un sens mathématique pour les entiers naturels, et le décalage arithmétique pour les entiers relatifs.



TD 3
Mémoires et macroblocs

Ex. 1 : Etude d'une mémoire vidéo

Soit la mémoire suivante utilisée dans une carte vidéo pour téléphone portable. La définition est de 320×240, 16 bits par pixels.

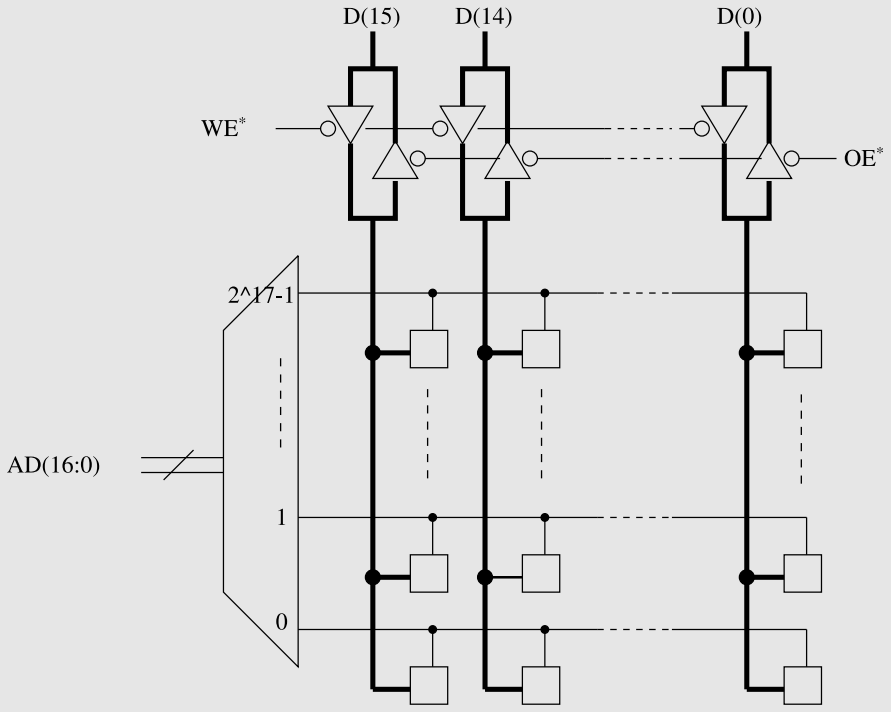
Il s'agit d'une mémoire asynchrone, on utilisera les caractéristiques temporelles du boîtier mémoire vu en cours :

— *Read Cycle Time* $t_{RC} = 10\ ns$;
— *Output Hold Time* $t_{OHA} = 2\ ns$;
— *Address Access Time* $t_{AA} = 3\ ns$.

Question 1 Combien de mots de 16 bits comporte cette mémoire ? Sur combien de bits doit-on coder les adresses ?

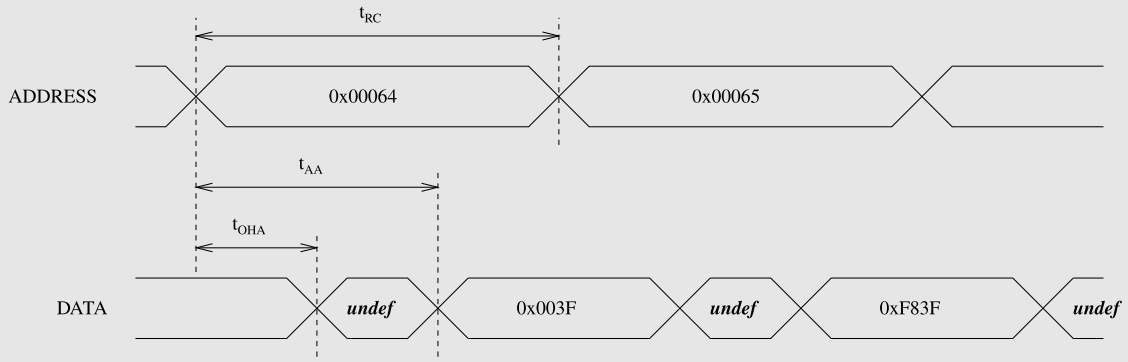
Il y a $320 \times 240 = 76800$ pixels, chacun codé sur 16 bits, donc 76800 mots de 16 bits en mémoire. Il faut 17 bits pour coder les adresses vu que $2^{16} = 65536 < 76800 \leq 2^{17} = 131072$.

Question 2 Proposer une structure simple de cette mémoire.



Les carrés au centre de la figure représentent un point mémoire de 1 bit (e.g. SRAM).

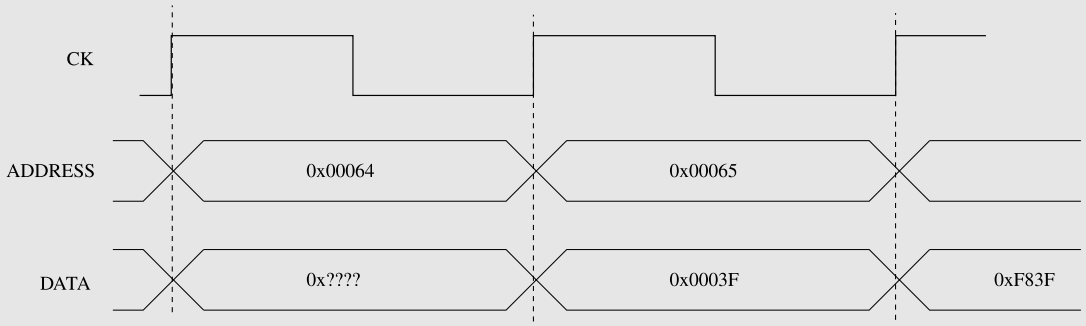
Question 3 Les pixels de coordonnées $(100, 0)$ et $(101, 0)$ affichent du bleu et du violet. On admettra que cela signifie que la mémoire contient les valeurs $0x003F$ et $0xF83F$ aux adresses $0x64$ et $0x65$. Dessiner le chronogramme représentant la lecture des mots mémoires correspondant.



Question 4 Quelle est la fréquence limite de fonctionnement de la carte graphique (*i.e.* le nombre de lectures en mémoire vidéo par seconde) ?

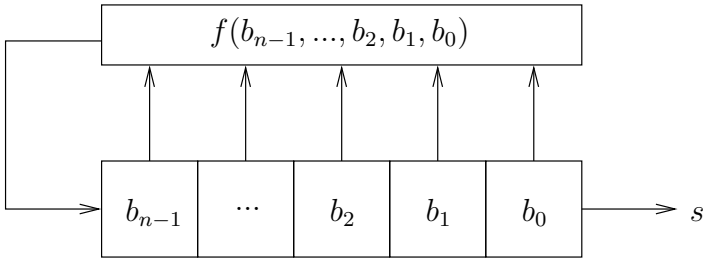
La période d'un cycle de lecture étant 10 ns, la fréquence maximale est 100 MHz.

Question 5 Supposons maintenant que la mémoire est synchrone et répond en un cycle. Redessiner le chronogramme de la question 3 en conséquence.



Ex. 2 : Générateur de nombres aléatoires

On travaille dans cet exercice sur un composant appelé registre à décalage à rétroaction linéaire (*Linear Feedback Shift Register*) servant à générer des séquences « pseudo-aléatoires » de bits. Le principe de ce composant est détaillé dans la figure ci-dessous.



Chaque case correspond à une bascule D : il y a donc n bascules numérotées de 0 à $n-1$, n étant le nombre de bits du LFSR. La sortie s est le bit pseudo-aléatoire généré par le LFSR : elle

affiche la valeur de la bascule numéro 0. A chaque cycle, on charge le contenu de la bascule i dans la bascule $i-1$: si on considère la séquence des valeurs contenues dans les bascules comme une valeur $B = b_{n-1}...b_1b_0$, alors B est décalée d'un bit vers la droite à chaque cycle. La valeur insérée dans la bascule $n-1$ est calculée en fonction des valeurs de chaque bascule via une fonction de rétroaction f définie par $f(b_{n-1}, ..., b_2, b_1, b_0) = c_{n-1}.b_{n-1} \oplus ... \oplus c_1.b_1 \oplus c_0.b_0$ où le c_i sont des constantes binaires données.

Dans la suite de l'exercice, on pose $n = 4$ et $f(b_3, b_2, b_1, b_0) = 0.b_3 \oplus 0.b_2 \oplus 1.b_1 \oplus 1.b_0 = b_1 \oplus b_0$.

Question 1 On suppose que les bascules sont initialisées avec les valeurs suivantes : $b_3 = 1$ et $b_2 = b_1 = b_0 = 0$. Remplir un tableau contenant les valeurs des bascules en fonction du temps jusqu'à ce qu'on retombe sur la valeur initiale.

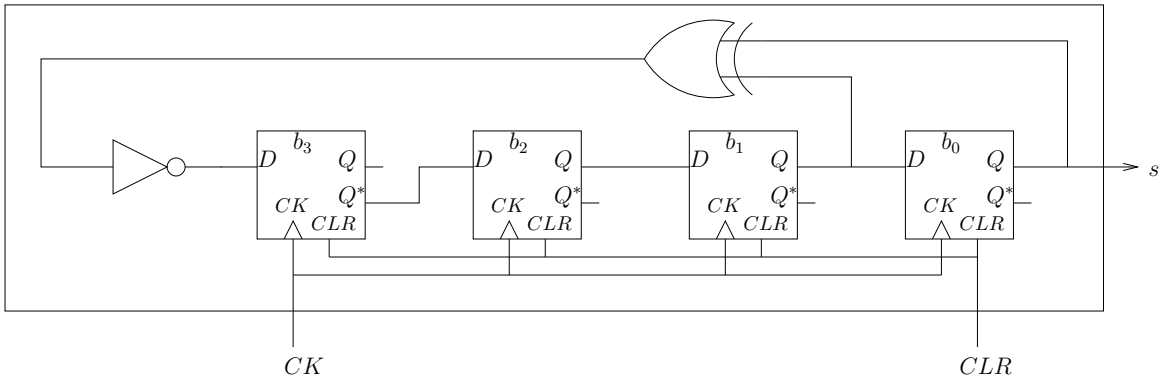
b_3	b_2	b_1	b_0	f
1	0	0	0	0
0	1	0	0	0
0	0	1	0	1
1	0	0	1	1
1	1	0	0	0
0	1	1	0	1
1	0	1	1	0
0	1	0	1	1

b_3	b_2	b_1	b_0	f
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0
0	1	1	1	0
0	0	1	1	0
0	0	0	1	1
1	0	0	0	0

Question 2 Exprimer le nombre de valeurs différentes possible en fonction de n. Que se passe-t'il si on se retrouve avec $b_3 = b_2 = b_1 = b_0 = 0$?

Il y a $2^n - 1$ valeurs différentes, puisqu'on peut coder 2^n valeurs différentes sur n bits et que le cas $B = 0$ n'est pas possible : si $b_3 = b_2 = b_1 = b_0 = 0$, f telle qu'on l'a choisie dans cet exercice est une fonction constante valant toujours 0.

Question 3 Une réalisation d'un LFSR 4 bits (avec la fonction de rétroaction : $f(b_3, b_2, b_1, b_0) = b_1 \oplus b_0$) à l'aide de 4 bascules D est proposée ci-dessous.

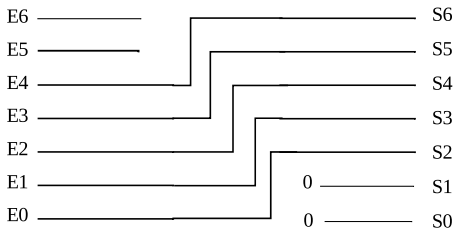


Justifiez l'utilisation de l'inverseur et de la connexion à $\overline{Q}$ sur la bascule $b3$.

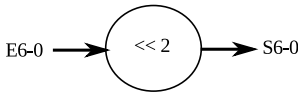
Pour la séquence initiale, il suffit de raisonner sur les valeurs en sortie des bascules : initialement, toutes les bascules sont à 0 après le *CLR*. Or, comme on prend la sortie $\overline{Q}$ de b_3, la séquence obtenue initialement en sortie des bascules est bien 1000. Pour les cycles suivants, on doit inverser l'entrée de b_3 pour annuler le fait qu'on prend $\overline{Q}$ en sortie de la bascule.

Ex. 3 : Décaleur à barillet

Un décalage d'un nombre constant de bits ne nécessite aucune porte logique : il s'agit de connecter les fils de façon adéquate (voir schéma ci-dessous).



Décalage à gauche de 2 positions



Symbole

Un décaleur à barillet (ou *barrel shifter*) est un opérateur combinatoire qui permet de décaler un nombre x, codé sur n bits, d'un certain nombre de bits $p < n$ vers la gauche ou la droite. On note typiquement :

- $x << p$ le décalage de x de p bits vers la gauche en remplissant les cases libérées à droite par des 0 ;
- $x >> p$ le décalage de x de p bits vers la droite en remplissant les cases libérées à gauche par le bit de signe du nombre x initial (on parle de décalage arithmétique) ;
- $x >>> p$ le décalage de x de p bits vers la droite en remplissant les cases libérées à gauche par des 0 (on parle de décalage logique).

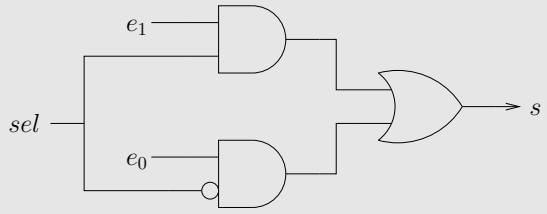
Question 1 Sur combien de bits doit être codé le décalage p si on suppose que x est codé sur 32 bits ? Déduisez-en la relation générale entre le nombre de bits de x (n) et de p (m). Mathématiquement parlant, à quelles opérations élémentaires correspondent les différents décalages ?

Si x est codé sur 32 bits, p est compris entre 0 et 31 (on suppose qu'on n'autorise pas le décalage de 32 bits qui a pour effet de mettre tous les bits de x à 0 ou à 1), on a donc besoin de 5 bits pour coder p. De façon générale, on a besoin de $m = \lceil log_2(n) \rceil$ (*i.e.* le plus petit entier supérieur ou égal à $log_2(n)$) bits pour coder p.

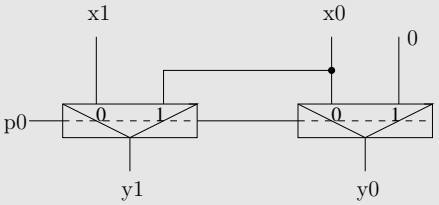
Décaler un entier relatif x de p bits vers la gauche revient à le multiplier par 2^p. Réciproquement, décaler arithmétiquement de p bits vers la droite revient à diviser par 2^p. Le décalage logique n'a de sens mathématique que pour les entiers naturels, mais est par ailleurs utile pour la manipulation de bits en programmation bas niveau (pilotes de périphériques, etc).

Question 2 Proposez l'implantation d'un décaleur à gauche d'une entrée x pour $n = 2$. Quelle porte élémentaire est idéale pour cette implantation ?

La porte de base pour cette opération est le mux 1 bit $2 \rightarrow 1$.



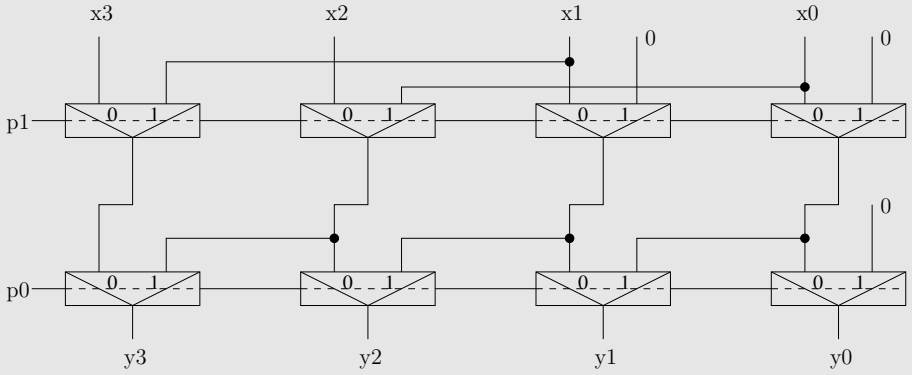
Il suffit de connecter les mux élémentaires comme précisé ci-dessous pour réaliser la fonction.



Question 3 En vous basant sur le résultat précédent, proposer une implantation pour $n = 4$ et $n = 8$. Evaluer le circuit : nombre de couches logiques à traverser et complexité en surface, l'unité étant le mux 1 bit $2 \rightarrow 1$.

L'astuce ici consiste à remarquer que les étages successifs peuvent chacun décaler d'une puissance entière de 2, et ainsi former un décalage de x avec n'importe quel entier naturel. Ainsi, p_0 décale de 0 ou 1 position, p_1 décale de 0 ou 2 positions, p_2 décale de 0 ou 4 positions, etc.

Le résultat pour un décaleur sur 4 bits est présenté ci-dessous.



Faire la version 8 bits permet de bien ancrer l'idée, mais coûte du temps, ... On fait le calcul en $\lceil log_2(n) \rceil$ couches logiques avec n mux élémentaires par couche logique, ce qui fait tout de même une complexité en surface en $O(n\lceil log_2(n) \rceil)$.

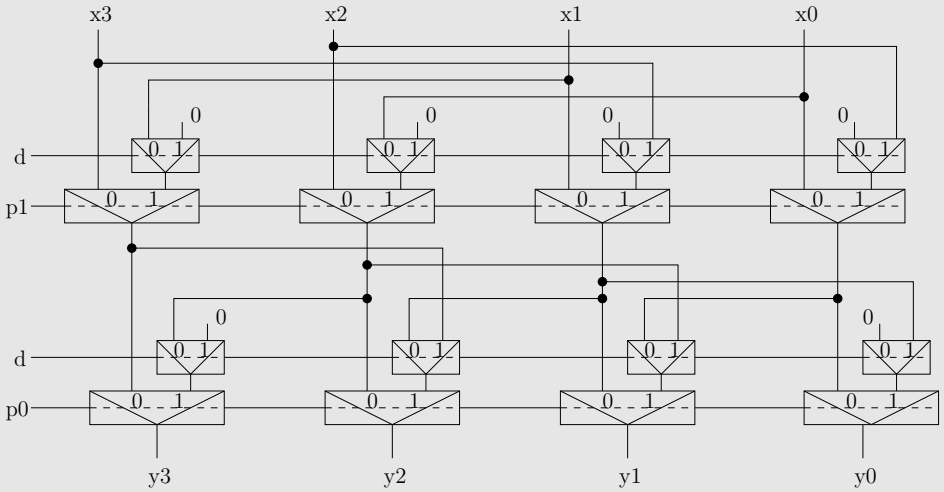
Pour aller plus loin...

Question 4 On veut maintenant gérer le décalage logique vers la droite. On ajoute pour cela une entrée d telle que $d = 1$ si on effectue un décalage à droite. Modifiez le décaleur 4 bits pour gérer le décalage logique à droite en plus du décalage à gauche. Attention, il y a deux solutions :

- la solution triviale est de choisir à chaque étage un bit de l'étage précédent en fonction du sens du décalage à effectuer,
- la solution avec "miroirs" : on utilise le décaleur à gauche vu à la question 3. Si on doit décaler à droite, on permute les bits de l'entrée ("miroir" : on met en entrée du décaleur les bits 0 à 3 au lieu des bits 3 à 0), puis on effectue sur cette nouvelle valeur un décalage à gauche ; les bits du résultat obtenu sont permutés à nouveau en sortie.

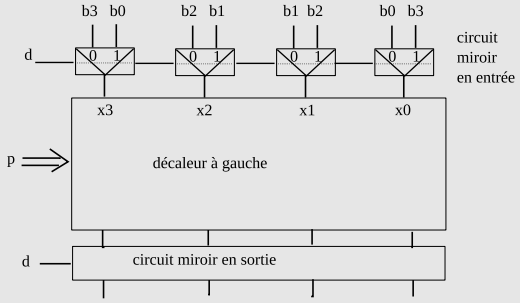
Evaluer les deux solutions.

Solution 1 :



On peut aussi le faire avec des mux $3 \rightarrow 1$, ce qui simplifie un peu le schéma.

Solution 2 :



Solution 1 : couches logiques : $\lceil log_2(n) \rceil$ avec $2 * n$ mux par couche logique, complexité en surface en $O(2 * n \lceil log_2(n) \rceil)$.

Solution 2 : couches logiques : $\lceil log_2(n) \rceil + 2$ avec n mux par couche logique, complexité en surface en $O(n(\lceil log_2(n) \rceil + 2))$.

Pour aller plus loin...

Question 5 On veut enfin ajouter la gestion du décalage arithmétique à droite. On dispose d'une nouvelle entrée a valant 1 si on veut effectuer un décalage arithmétique lors d'un décalage à droite. (Notez que le décalage à gauche est toujours arithmétiquement valable, indépendamment de la valeur de a). Modifier le schéma du décaleur gauche/droite 4 bits pour y ajouter la gestion du bit de signe dans les décalages à droite.

TD 4
Opérateurs Arithmétiques et Logiques

Préparation
Ex. 1 : Codage des nombres en complément à deux

Le codage en complément à deux est un codage permettant de représenter des entiers relatifs en binaire. Il existe d'autres codages moins utilisés (*e.g.* signe et valeur absolue, complément à un) que l'on ne détaillera pas ici.

Question 1 Remplir un tableau contenant les valeurs en base 2 et en base 16 des entiers relatifs entre -8 et 7 (inclus), codés en complément à 2^4 . Comment peut-on calculer $-X$ à partir de X en base 2 ?

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Décimal	Binaire	Hexadécimal
-8	1000	8
-7	1001	9
-6	1010	A
-5	1011	B
-4	1100	C
-3	1101	D
-2	1110	E
-1	1111	F

On a la relation $-X = \overline{X} plus 1$ sauf pour -2^{n-1} qui n'a pas d'opposé sur 4 bits (vu que l'intervalle des valeurs possibles n'est pas symétrique par rapport à 0) : $\overline{-8_{10}}+1 = \overline{1000_2}+1 = 0111_2+1 = 1000_2$ qui est le codage de -8_{10}.

Question 2 Quel intervalle d'entiers relatifs peut-on coder sur n bits ?

Sur n bits, on peut coder tous les entiers dans $[-2^{n-1} .. 2^{n-1}-1]$.

Question 3 Effectuer les opérations $2+3$, $-3+3$,$-2+-5$, $7+1$, $-7+-2$ sur 4 bits.

$2+3=5$:

	0	0	1	0	
+	0	0	1	1	
		1			retenues
	0	1	0	1	

$-3+3=0$:

	1	1	0	1	
+	0	0	1	1	
	1	1	1	1	retenues
	0	0	0	0	

$7+1=$ ... (non significatif) :

	0	1	1	1	
+	0	0	0	1	
	1	1	1		retenues
	1	0	0	0	

$-7+-2=$ ... (non significatif) :

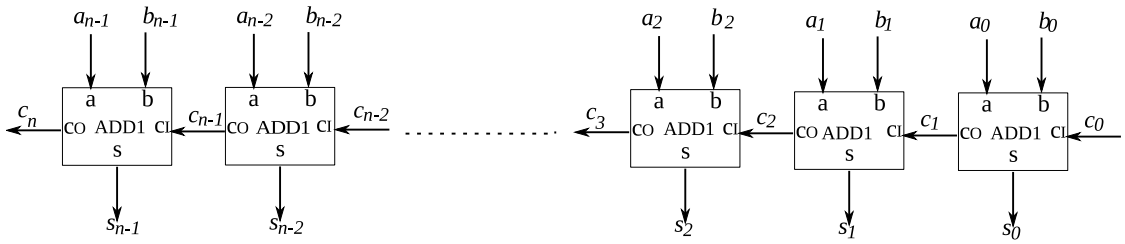
	1	0	0	1	
+	1	1	1	0	
1					retenues
	0	1	1	1	

Question 4 Comment peut-on détecter un dépassement de capacité lors d'une opération sur des nombres codés en complément à deux ?

On a un overflow ssi on a $a_{n-1} = b_{n-1}$ et $a_{n-1} \neq s_{n-1}$.

Ex. 2 : Additionneur binaire

On utilise l'architecture dite de « l'additionneur à propagation de retenue ». Le principe est de construire une cellule élémentaire d'addition 1 bit ADD1 capable d'effectuer une addition entre deux bits a et b, en prenant en compte une retenue entrante cI (carry in), et qui produit en sortie la somme s et une retenue sortante cO (carry out). Une fois cette cellule disponible, il est facile de construire un additionneur n bits en reliant la retenue sortante de la cellule de rang i à la retenue entrante de la cellule de rang $i+1$.

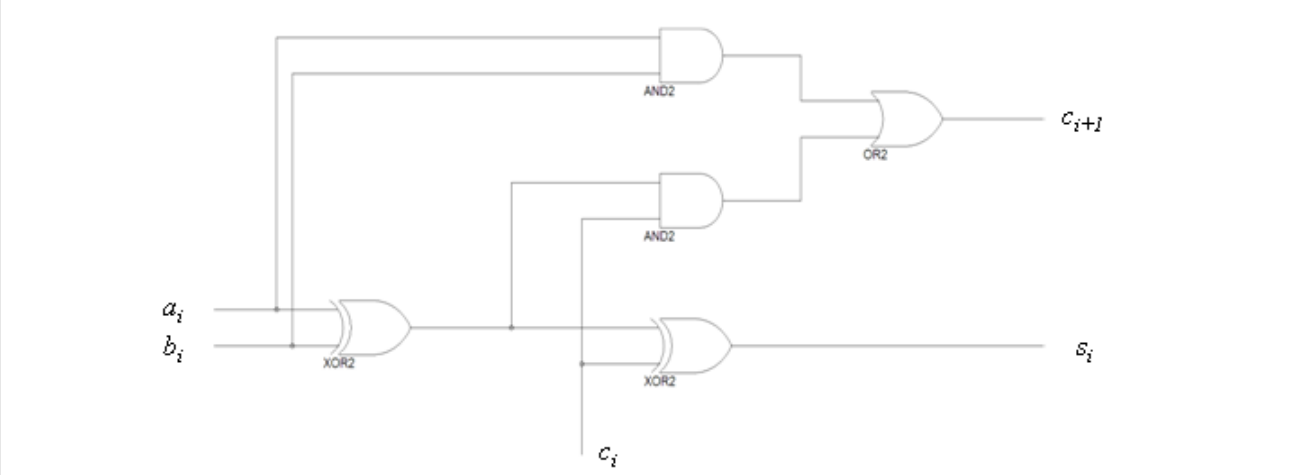


Question 1 Donner les expressions simplifiées de la somme s et de la retenue sortante cO d'une cellule élémentaire ADD1 en fonction des opérandes a et b et de la retenue entrante cI. Dessiner le schéma correspondant. Que doit valoir la retenue entrante c_0 de l'additionneur n bits ? Evaluer le temps de calcul pour une addition n bits.

a	b	cI	cO	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0

a	b	cI	cO	s
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$
\begin{aligned}
s &= \overline{a}.\overline{b}.cI + \overline{a}.b.\overline{cI} + a.\overline{b}.\overline{cI} + a.b.cI \\
&= a \oplus b \oplus cI \\
cO &= \overline{a}.b.cI + a.\overline{b}.cI + a.b.\overline{cI} + a.b.cI = \overline{a}.b.cI + a.\overline{b}.cI + a.b \\
&= (a \oplus b).cI + a.b
\end{aligned}
$$

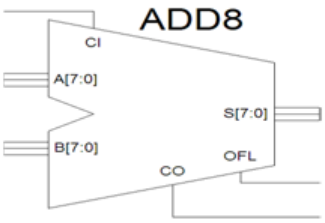


Pour la cellule de rang 0, la retenue entrante doit être à 0. Le temps de calcul est en O(n).

On remarque qu'on aurait pu utiliser la formule cO = Maj (a,b,cI). La réalisation à partir de cette formule demande plus de portes, mais est un peu plus rapide en temps de propagation que celle proposée.

Question 2 Un additionneur dispose également de deux sorties particulières, sur 1 bit chacune (appelées « indicateurs » (*flags*)) et définies par :

- CO (*Carry Out*) est la retenue sortante c_n générée par l'addition, qui vaut 1 ssi une addition portant sur des entiers naturels a généré un dépassement de capacité ;
- OFL (*Overflow*) vaut 1 ssi une addition portant sur des entiers codés en complément à deux a généré un dépassement de capacité.



Exprimer le bit OFL en fonction des c_i.

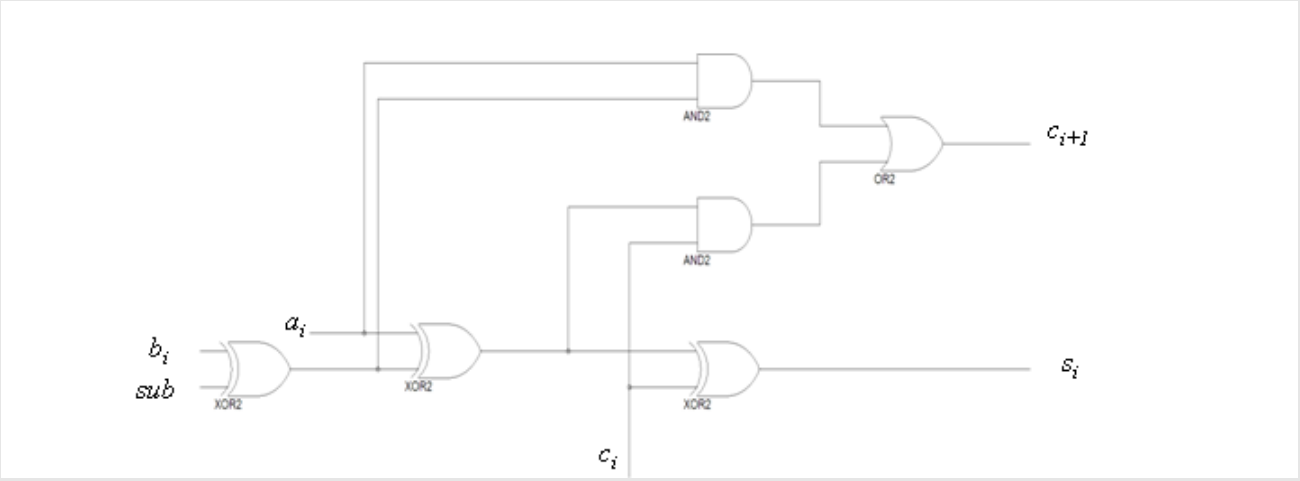
En injectant les conditions de dépassement de la question 1.4 dans les équations de la question 2.1, on montre l'implication : $(a_{n-1} = b_{n-1} \text{et} a_{n-1} =\neq s_{n-1}) \Rightarrow c_n \neq c_{n-1}$
L'implication inverse ($\Leftarrow$) se montre facilement en supposant $c_n = 0$ ou 1. Donc, on a dépassement de capacité ssi $c_n \neq c_{n-1}$. Comme XOR est l'opérateur « différence », on a $OFL = c_n \oplus c_{n-1}$.
On rappelle que pour les entiers naturels, il suffirait de regarder c_n.

Ex. 3 : Additionneur /soustracteur

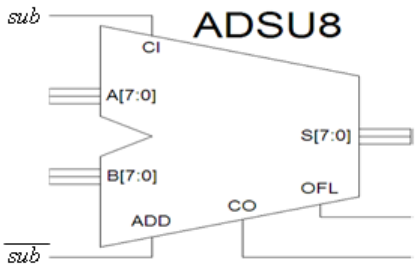
On veut maintenant implanter l'opération de soustraction en utilisant l'additionneur de l'exercice 2.

Question 1 En se basant sur la relation $-X = \overline{X} plus 1$, expliquer comment on peut simplement modifier le circuit d'une cellule élémentaire d'additionneur pour gérer aussi la soustraction de deux entiers relatifs codés en complément à deux, suivant la valeur d'une entrée *sub* qui vaut 0 pour une addition, 1 pour une soustraction. Dessiner le schéma correspondant à une cellule de rang i. Que faut-il faire pour la cellule de rang 0 ?

On ne considère ici que l'addition et la soustraction simple, sans prise en compte de la retenue entrante. On sait que A *moins* $B = A$ *plus* $(-B) = A$ *plus* $\overline{B}$ *plus* 1. On effectue A *plus* $\overline{B}$ en rajoutant sur l'ancien b_i le résultat de $b_i \oplus sub$ pour inverser b_i dans le cas d'une soustraction (méthode valable pour toutes les cellules). Il reste à ajouter 1 ; on utilise pour cela l'entrée $CI = c_0$ qui doit être à 0 pour l'addition, à 1 pour la soustraction : il suffit de connecter sub à CI.



Question 2 La sortie CO de l'additionneur/soustracteur est égale à la retenue c_n. Dans le cas d'une soustraction d'entiers naturels, que veut dire la valeur de CO ? Montrez que la variable $C = CO \oplus sub$ vaut 1 ssi une addition ou une soustraction portant sur des entiers naturels génère un résultat qui n'est pas un entier naturel codable sur n bits. On peut aussi dire que CO est actif à 1 pour l'addition, à 0 pour la soustraction des entiers naturels.



Note : l'indicateur OFL est toujours valide pour les entiers codés en complément en 2, que l'opération soit une addition ou une soustraction.

Pour une soustraction portant sur des entiers naturels, CO à 0 indique un résultat négatif, donc une "retenue sortante".
La soustraction sur des entiers naturels revient à les coder virtuellement sur $n+1$ bits, puis à ajouter A et le complément à 2 de B : le bit a_n est à 0, le bit b_n est à 1. Si le bit s_n est à 1, le résultat est négatif, donc non codable : il faut donc une retenue c_n à 1 pour mettre le bit (virtuel) s_n à 0.

On a donc $C = CO \oplus sub$ qui indique dans les 2 cas (addition et soustraction) un résultat non codable.

Pour aller plus loin...
Question 3 On cherche à effectuer une opération sur 2n bits par composition d'opérations sur n bits, en utilisant 2 additionneurs/soustracteurs n bits. Comment connecter les deux circuits ?

La sortie CO du circuit qui traite les n bits poids faible est connectée à l'entrée CI du circuit qui traite les n bits poids fort ;

Pour aller plus loin...

Ex. 4 : Unité Arithmétique et Logique

Dans cet exercice, on va étudier l'unité arithmétique et logique, et construire la partie centrale de ce circuit (l'étude complète d'un tel circuit dépasse le cadre du TD).

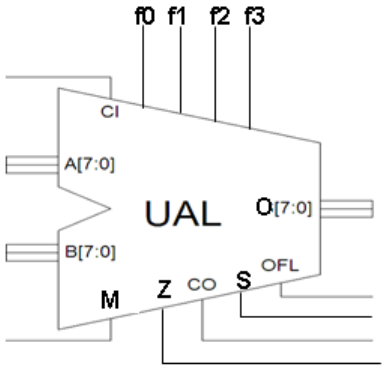
Une Unité Arithmétique et Logique (UAL ou ALU pour Arithmetic and Logic Unit) est un circuit combinatoire regroupant différentes opérations arithmétiques (addition, soustraction, etc.) et logiques (AND, OR, XOR, etc.) applicables à des valeurs codées sur un nombre n de bits (on parle alors d'« UAL n bits »).

Sur le schéma ci-dessous, le bit M permet de choisir le mode (M=0 : mode logique, M = 1 : mode arithmétique), les bits f_i représentent les entrées de fonction, qui permet de choisir l'opération que doit effectuer l'UAL. Les entrées A et B servent à coder les opérandes et O le résultat de l'opération (sur 8 bits dans l'exemple du schéma).

Outre CO et OFL, l'UAL dispose également de deux indicateurs, définis par :

— Z (*Zero*) vaut 1 ssi le résultat d'une opération est nul ;

— S (*Sign*) vaut 1 ssi le résultat d'une opération, interprété comme un entier relatif codé en complément à deux, est négatif.



Les opérations que doit réaliser l'UAL que nous concevons sont indiquées dans le tableau ci-dessous.

Opération	Mode M	f0 f1 f2 f3	Résultat
const0	0		0000
const-1	0		1111
add	1		F = A plus B plus CI
sub	1		F = A moins B moins 1 plus CI
notA	0		F = NOT(A)
notB	0		F = NOT(B)
xor	0		F =A XOR B
or	0		F = A OR B
and	0		F = A AND B
nopA	0		F = A
nopB	0		F = B

Question 1 Comment peut-on très simplement créer la génération des indicateurs Z et S par l'UAL ?

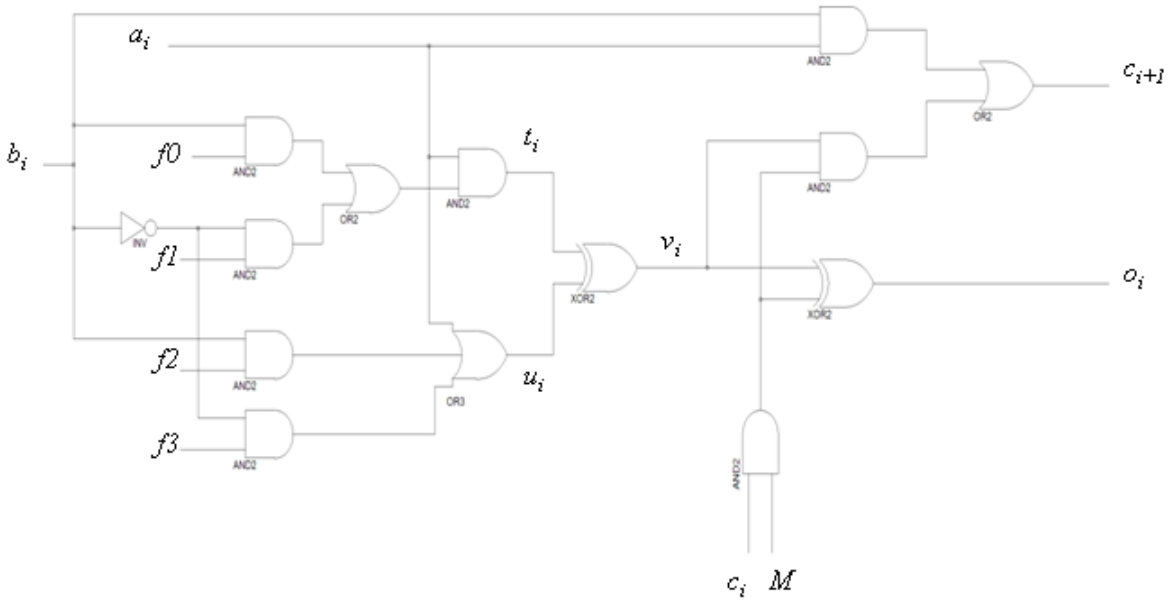
Pour Z, il suffit d'ajouter une porte NOR dont les entrées sont les n bits de la sortie O : si tous les o_i valent 0, la sortie du NOR vaudra 1.

Pour S, il suffit de prendre le bit de poids fort de la sortie O : $S = o_{n-1}$.

Question 2 On essaie de créer l'UAL avec un minimum de modifications de la cellule additionneur/soustracteur déjà conçue. Comment doit agir la commande de mode M sur les retenues c_i ?

Pour les opérations logiques, il faut inhiber la propagation de retenue , donc insérer une porte ET , pour faire c_i ET M à l'entrée de chaque étage.

On propose la cellule suivante (schéma pour la cellule de rang i) :



La sortie s_i de la cellule d'additionneur/soustracteur est égale à $a_i \oplus (b_i \oplus sub) \oplus c_i$. Si la propagation de la retenue est inhibée, $s_i = a_i \oplus (b_i \oplus sub)$. Pour concevoir l'UAL, on pose $o_i = v_i \oplus (c_i.M)$, avec $v_i = a_i \oplus (b_i \oplus sub)$ pour l'addition/soustraction.

On remarque que : $x \oplus y = x.y \oplus (x + y)$. On a donc :

$v_i = a_i(b_i \oplus sub) \oplus (a_i + b_i \oplus sub) = a_i(b_i.\overline{sub} + \overline{b_i}.sub) \oplus (a_i + b_i.\overline{sub} + \overline{b_i}.sub)$

Pour avoir plus de choix sur les termes pris en compte, on remplace les occurrences de sub et $\overline{sub}$ par $f0, f1, f2, f3$. On obtient donc : $v_i = a_i(b_i.f0 + \overline{b_i}.f1) \oplus (a_i + b_i.f2 + \overline{b_i}.f3)$

Pour simplifier le travail, on notera t_i le premier terme de l'expression de v_i et u_i le deuxième terme. On a donc : $t_i = a_i(b_i.f0 + \overline{b_i}.f1)$ et $u_i = a_i + b_i.f2 + \overline{b_i}.f3$

Question 3 Quelles valeurs doivent prendre M et les variables de sélection f_j pour l'addition ? la soustraction ?

Addition : M=1 et f0 f1 f2 f3 = 1 0 1 0 avec $c_0 = 0 : t_i = a_i.b_i)$, $u_i = a_i + b_i$, $v_i = a_i.b_i \oplus (a_i + b_i) = a_i \oplus b_i$, $o_i = a_i \oplus b_i \oplus c_i$ soustraction : M = 1 et f0 f1 f2 f3 = 0 1 0 1 avec $c_0 = 1 : t_i = a_i.\overline{b_i}$, $u_i = a_i + \overline{b_i}$ ...

Question 4 Déterminer les valeurs de t_i et u_i suivant les valeurs des f_j, puis ce que vaut v_i. En déduire les valeurs des f_j nécessaires pour les opérations logiques indiquées dans le tableau de fonctionnement de l'UAL.

Opération	Mode M	f0 f1 f2 f3	Résultat
const0	0	1100	0000
const-1	0	0011	1111
add	1	1010	F=A plus B plus CI
sub	1	0101	F=A minus B minus (notCI)
notA	0	1111	F=not(A)
notB	0	1001	F=not(B)
xor	0	1010	F=A XOR B
or	0	0010	F = A OR B
and	0	0100	F = A AND B
nopA	0	0000	F = A
nopB	0	0110	F = B

Question 5 Rechercher l'ensemble des opérations arithmétiques que peut faire cette UAL.

TD 5
Synthèse d'automates

Méthodologie
Ex. 1 : Méthodologie de synthèse d'automates : reconnaisseur de séquences

Soit un circuit prenant en entrée une lettre dans l'ensemble $\{a, b, c\}$ et générant en sortie une lettre dans l'ensemble $\{o, n\}$ qui vaut o ssi on vient de reconnaître la séquence $a + bcc^*$ et n sinon. Noter que la séquence de caractères en entrée est infinie : l'automate ne s'arrête jamais (*i.e.* pas d'état puits).

Graphe d'états de l'automate

On spécifiera en général l'automate comme un automate de Moore (sortie associée aux états).

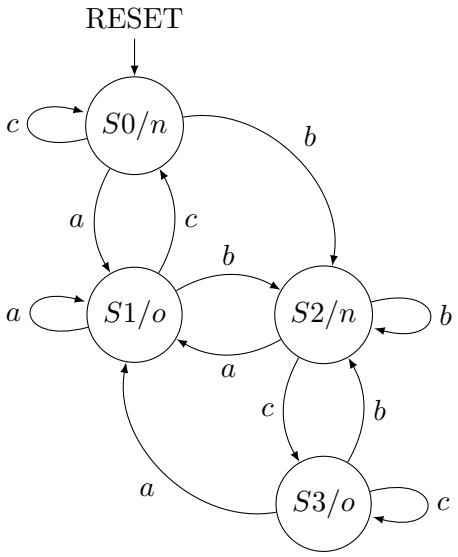


Table de transitions

Cette table est une représentation textuelle du graphe.

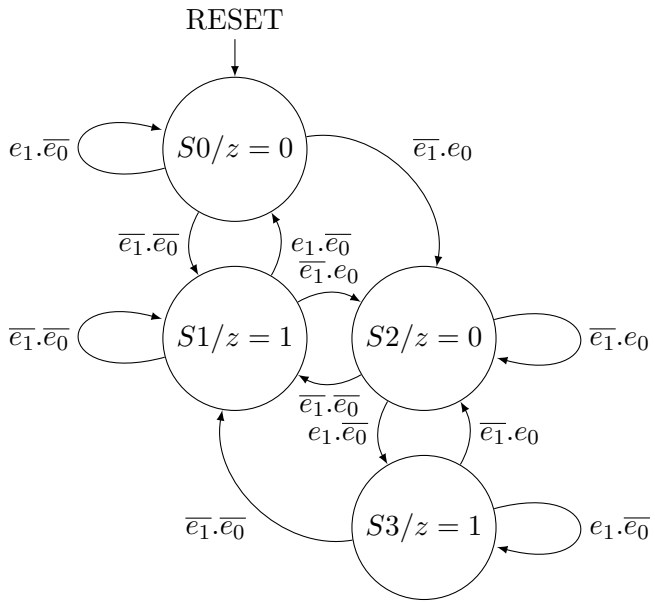
Etat courant	Sortie	Entrée	Etat futur
S0	n	a	S1
		b	S2
		c	S0
S1	o	a	S1
		b	S2
		c	S0
S2	n	a	S1
		b	S2
		c	S3
S3	o	a	S1
		b	S2
		c	S3

Codage des entrées et des sorties

On a besoin de 2 bits e_1 et e_0 pour coder les trois lettres en entrée : $a \equiv 00$, $b \equiv 01$ et $c \equiv 10$. Le code 11 est impossible. On a besoin de 1 bit z pour coder les deux lettres en sortie : $n \equiv 0$ et $o \equiv 1$.

Remarque : en général, le codage des entrées et des sorties d'un automate sur des variables booléennes est imposé par l'environnement, et cette étape ne sera pas nécessaire.

On peut réécrire le graphe d'états et la table de transitions de l'automate en utilisant ce codage des entrées et de la sortie :



Etat courant	Sortie	Entrée	Etat futur
S0	0	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		$e_1.\overline{e_0}$	S0
S1	1	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		$e_1.\overline{e_0}$	S0
S2	0	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		$e_1.\overline{e_0}$	S3
S3	1	$\overline{e_1}.\overline{e_0}$	S1
		$\overline{e_1}.e_0$	S2
		$e_1.\overline{e_0}$	S3

Codage logarithmique des états

On choisit ici un codage logarithmique, c'est-à-dire avec le nombre minimum de variables booléennes (le codage 1 parmi n, qui utilise une variable par état, est présenté à la fin de l'exercice).

En codage logarithmique, pour coder les 4 états possibles, il faut au minimum 2 bits q_1 et q_0. Pour cet exercice, on peut choisir le codage : $S0 \equiv 00$, $S1 \equiv 01$, $S2 \equiv 10$ et $S3 \equiv 11$.

Expressions simplifiées des variables d'état et de la sortie de l'automate

Il s'agit de trouver les expressions des variables q_1 et q_0 à l'instant $t+1$, en fonction de q_1, q_0 et des entrées à l'instant t. Nous utilisons des bascules D : pour obtenir la valeur souhaitée à l'instant $t+1$ en sortie des bascules, il faut mettre à l'instant t cette valeur sur les entrées D des bascules. On cherchera donc les expressons simplifiées de D_1 et de D_0 en fonction de q_1, q_0 et des entrées e_1 et e_0.

e_1e_0 \ q_1q_0	**00**	**01**	**11**	**10**
00	0	0	0	0
01	1	1	1	1
11	Φ	Φ	Φ	Φ
10	0	0	1	1

Expression de D_1

e_1e_0 \ q_1q_0	**00**	**01**	**11**	**10**
00	1	1	1	1
01	0	0	0	0
11	Φ	Φ	Φ	Φ
10	0	0	1	1

Expression de D_0

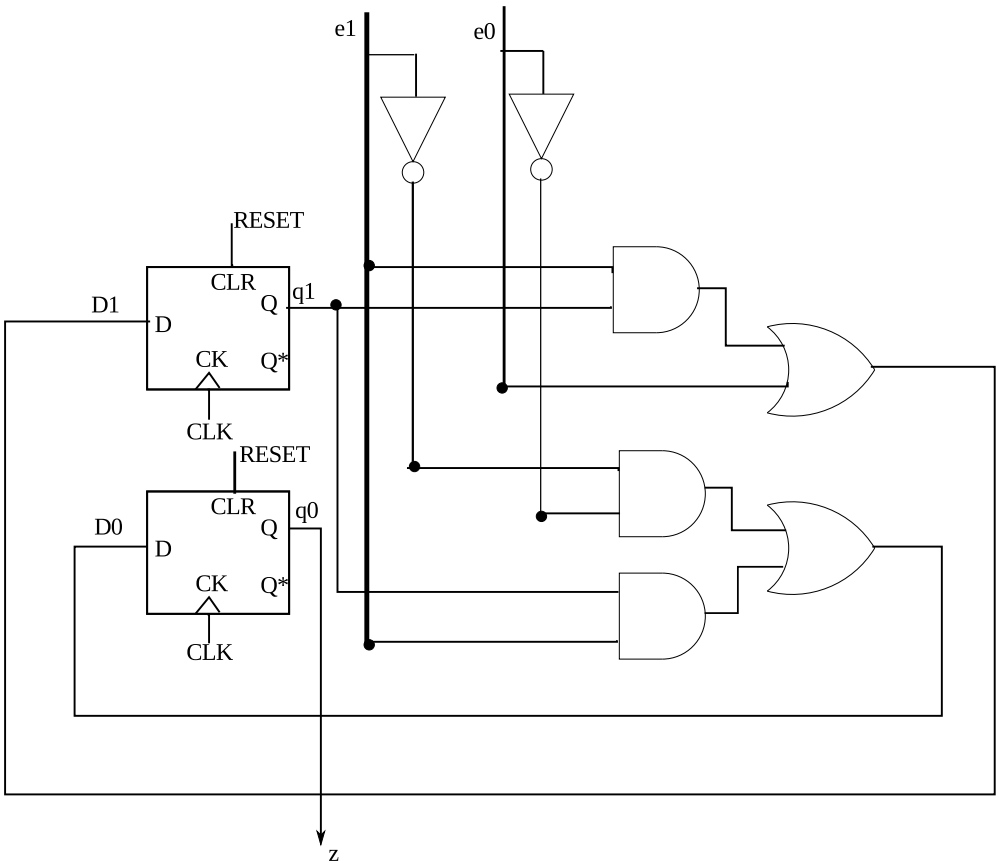
q_1 \ q_0	**0**	**1**
0	0	1
1	0	1

Expression de z

On a donc :

$$\begin{aligned} D_1 &= e_0 + e_1.q_1 \\ D_0 &= \overline{e_1}.\overline{e_0} + q_1.e_1 \\ z &= q_0 \end{aligned}$$

Schéma du circuit



Codage 1-parmi-n ou "one-hot"

Le principe de ce codage est d'associer une variable d'état à chaque état (donc une bascule par état). Une variable d'état (ou une bascule) à 1 signifie que l'état correspondant est actif ; quand la variable d'état est à 0, l'état est inactif.

On peut faire la synthèse du circuit à partir du graphe d'états ou de la table de transition.

Pour notre exemple : il y a 4 états, donc 4 bascules (entrées D_0, D_1, D_2 et D_3, sorties Q_0, Q_1, Q_2 et Q_3 avec la bascule i correspondant à l'état S_i). A partir de la table de transition on obtient la table de vérité des entrées de bascules :

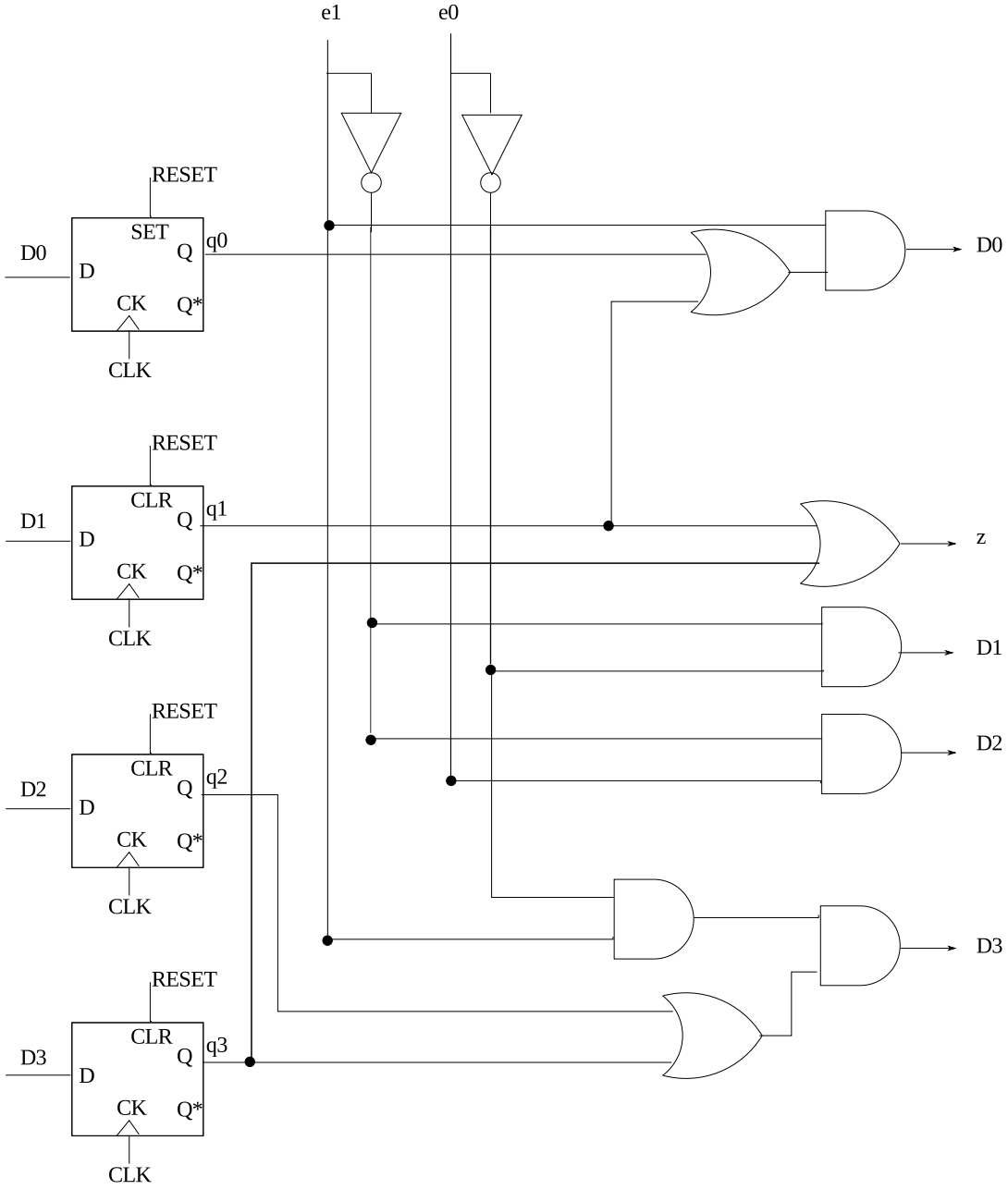
Etat courant	Sortie	Entrée	D0	D1	D2	D3
Q0	0	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		$e_1.\overline{e_0}$	1	0	0	0
Q1	1	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		$e_1.\overline{e_0}$	1	0	0	0
Q2	0	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		$e_1.\overline{e_0}$	0	0	0	1
Q3	1	$\overline{e_1}.\overline{e_0}$	0	1	0	0
		$\overline{e_1}.e_0$	0	0	1	0
		$e_1.\overline{e_0}$	0	0	0	1

A partir de cette table, on obtient :

$$\begin{aligned} D_0 &= e_1.\overline{e_0}.(Q_0+Q_1) \\ D_1 &= \overline{e_1}.\overline{e_0}.(Q_0+Q_1+Q_2+Q_3) = \overline{e_1}.\overline{e_0} \\ D_2 &= \overline{e_1}.e_0.(Q_0+Q_1+Q_2+Q_3) = \overline{e_1}.e_0 \end{aligned}$$

$$
\begin{aligned}
D_3 &= e_1.\overline{e_0}.(Q_2 + Q_3) \\
z &= Q_1 + Q_3
\end{aligned}
$$

Le circuit correspondant est donné ci-dessous. Noter que la bascule associée à l'état initial Q_0 est initialisée à 1 (pour que cet état devienne actif à l'initialisation), toutes les autres sont initialisées à 0.



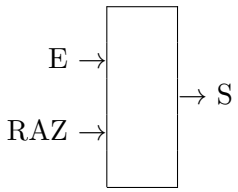
Ex. 2 : Décodeur « Non-Retour à Zéro Inversé »

On désire faire un module matériel permettant de décoder un signal NRZI avec un automate de Moore synchrone possédant 4 états, notés $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ et $\mathcal{D}$. Cet automate possède 2 entrées E et RAZ, et une unique sortie S. Les signaux sont les suivants :

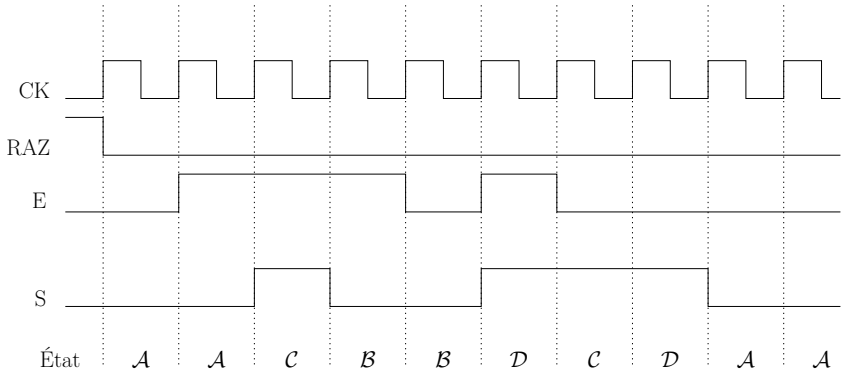
E fournit la donnée série (1 bit), c'est à dire la suite de bits à décoder un par un ;

RAZ force l'automate dans son état initial, l'état $\mathcal{A}$, lorsqu'elle est à '1' ;

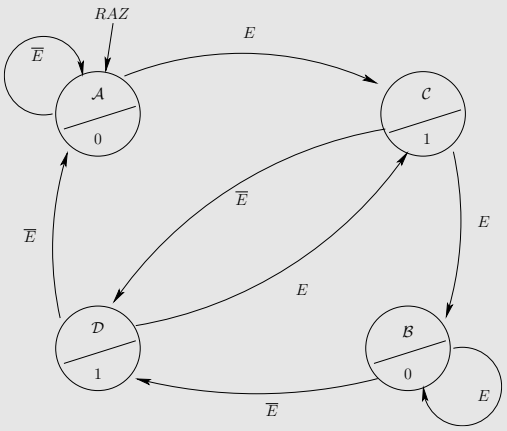
S est la sortie série.



Le chronogramme suivant montre un exemple de décodage qui illustre également le fonctionnement de l'automate.



Question 1 En utilisant ce chronogramme, déterminez le graphe d'états, en précisant les conditions de transitions et les valeurs de sortie pour chaque état.



Question 2 Les états de l'automate sont codés ainsi, sur deux bits notés Q1 et Q0.

	Q1	Q0
$\mathcal{A}$	0	0
$\mathcal{B}$	0	1
$\mathcal{C}$	1	0
$\mathcal{D}$	1	1

Construire la table de transition de l'automate, en notant D1 et D0, les valeurs futures à écrire dans les bascules D mémorisant Q1 et Q0.

État courant	Code Q1 Q0	S	E	État futur	Code D1 D0
$\mathcal{A}$	0 0	0	0	$\mathcal{A}$	0 0
			1	$\mathcal{C}$	1 0
$\mathcal{B}$	0 1	0	0	$\mathcal{D}$	1 1
			1	$\mathcal{B}$	0 1
$\mathcal{C}$	1 0	1	0	$\mathcal{D}$	1 1
			1	$\mathcal{B}$	0 1
$\mathcal{D}$	1 1	1	0	$\mathcal{A}$	0 0
			1	$\mathcal{C}$	1 0

Question 3 Donnez les expressions booléennes simplifiées de D1, D0 et S.

— $S = Q1$
— $D0 = Q1 \oplus Q0$
— $D1 = Q1 \oplus Q0 \oplus E$

Pour aller plus loin...
Question 4 Proposez une réalisation de cet automate en utilisant un codage 1 parmi n.

Pour aller plus loin...
Question 5 Dessiner le schéma du circuit permettant de décoder le code NRZI.

Utilisation de cet automate : Le codage NRZI (Non-Retour à Zéro Inversé) consiste à changer le niveau du signal codé chaque fois que l'on code un 1 (si le signal de sortie vaut 0, alors il devra valoir 1 au cycle suivant, et inversement) et à conserver le niveau du signal codé chaque fois que l'on code un 0 (si le signal de sortie vaut 0 au cycle courant, alors il y reste au cycle suivant, *idem* pour 1). Le signal de sortie est initialement à 0. Dans cet exercice, on a proposé un module matériel permettant de décoder un signal NRZI avec un automate de Moore synchrone possédant 4 états.

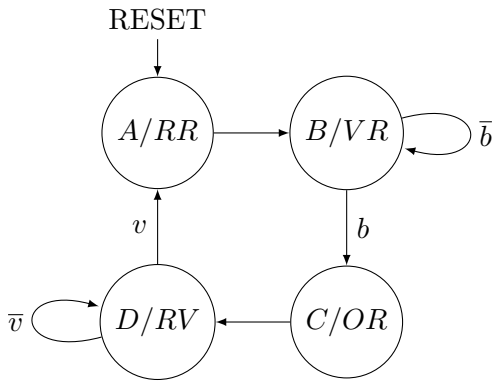
Ex. 3 : Gestion d'un feu tricolore

On cherche à modéliser le fonctionnement d'un feu tricolore composé d'un affichage à trois couleurs (Rouge, Orange, Vert) pour les véhicules et d'un affichage bicolore (Rouge, Vert) pour les piétons. Le feu fonctionne selon le principe très simplifié détaillé ci-dessous :

— le feu véhicules reste au vert tant qu'un piéton n'a pas appuyé sur le bouton d'appel ;
— le feu véhicules passe à l'orange dès qu'un piéton appuie sur le bouton d'appel ($b = 1$), puis après un certain temps il passe au rouge et le feu piétons passe au vert simultanément ;
— le feu piétons reste au vert tant qu'aucun véhicule n'est arrêté au feu ;
— lorsqu'un véhicule est détecté en attente devant le feu ($v = 1$), le feu piétons passe au rouge, puis après un certain temps, le feu véhicules passe au vert.

Pour simplifier, on considère que la notion de temps d'attente est manifestée par le passage d'un cycle (*i.e.* on ne prend pas en compte des durées d'attente différentes). Par convention, on décide que l'état initial est celui où les deux feux sont au rouge.

On donne un automate de Moore qui modélise le comportement de ce feu tricolore. Comme un énoncé en langue naturelle est toujours beaucoup plus imprécis qu'un modèle rigoureux, des choix d'interprétation ont été fait lors de la conception de cet automate : on considérera l'automate comme l'énoncé de référence.



Dans chaque état, la sortie est représentée par un couple de couleurs correspondant à l'affichage du feu véhicules suivi de celui du feu piétons.

Question 1 a) Combien y a t-il de combinaisons possibles et de combinaisons réelles pour les valeurs des feux piétons et véhicules ?
b) Peut-on établir un lien avec le nombre d'états de l'automate ?

a)
— Il y a 4 états car il a 4 temps (actions) à réaliser dans la séquence proposée.
— 6 combinaisons possibles
— 4 combinaisons réelles (2 interdites lorsque le feu piétons est au vert)
b) Pas de liens possibles. On peut très bien avoir plusieurs états avec la même sortie. Voir exo décodeur NRZI ou décodeur de séquence.

Question 2 Choisir un codage des états de l'automate et donner les expressions simplifiées des variables d'état de l'automate.

On a besoin de 2 bits q_1 et q_0 pour coder les 4 états possibles : $A \equiv 00$, $B \equiv 01$, $C \equiv 10$ et $D \equiv 11$. Les entrées sont naturellement codées sur les 2 bits b (bouton piétons) et v (véhicule détecté).

q_1q_0 \ bv	**00**	**01**	**11**	**10**
00	0	0	0	0
01	0	0	1	1
11	1	0	0	1
10	1	1	1	1

$D_1 = q_1.\overline{q_0} + q_1.\overline{v} + \overline{q_1}.q_0.b$

q_1q_0 \ bv	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	1	0	0
11	1	0	0	1
10	1	1	1	1

$D_0 = \overline{q_0} + \overline{q_1}.\overline{b} + q_1.\overline{v}$

Si on choisir le codage C=11 et D=10, les équations deviennent plus simples. On obtient $D_1 = q_0$ et $D_0 = q_1.q_0 + q_1.\overline{v} + q_0.b$

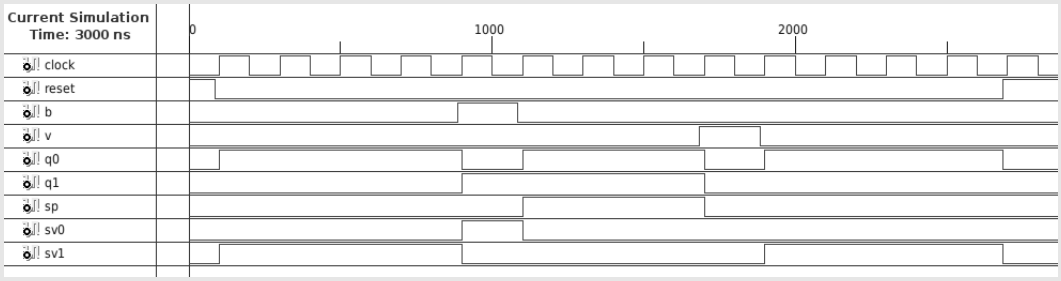
Question 3 Donner les expressions simplifiées des sorties de l'automate.

On note $S_V \in \{R, O, V\}$ la sortie correspondant au feu pour les véhicules et $S_P \in \{R, V\}$ celle correspondant au feu pour les piétons. On a alors besoin de deux bits s_{v1} et s_{v0} pour coder la sortie du feu véhicules (avec par exemple comme codage $R \equiv 00$, $O \equiv 01$ et $V \equiv 10$) et un bit s_p pour coder la sortie du feu piétons (avec simplement $R \equiv 0$ et $V \equiv 1$). On a alors trivialement à partir de l'automate :

$$\begin{aligned} s_{v1} &= \overline{q_1}.q_0 \\ s_{v0} &= q_1.\overline{q_0} \\ s_p &= q_1.q_0 \end{aligned}$$

Pour aller plus loin...
Question 4 Dessiner le schéma du circuit pilotant le feu tricolore.

Voir le schéma en annexe. On peut obtenir le chronogramme suivant en simulant sur une séquence d'entrée donnée :



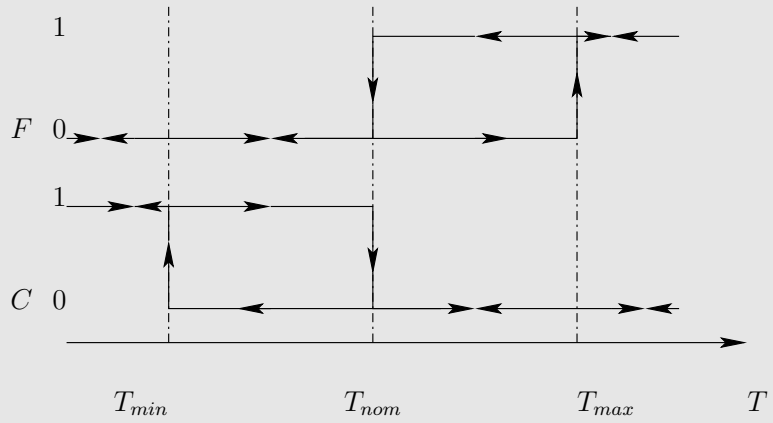
Pour aller plus loin...
Ex. 4 : Gestion de la température par hysteresis

On cherche à réaliser un système permettant d'assurer le maintien à une température quasi-constante d'un poulailler industriel, afin d'assurer aux volatiles une durée de vie compatible avec les normes Européennes. Ce système repose sur la disponibilité d'un climatiseur pouvant produire du chaud, du froid, ou ne rien faire, en fonction de la température courante de l'entrepôt (qui dépend de la température externe et du degré d'agitation des gallinacées).

Le système est basé sur le principe de l'hysteresis, c'est-à-dire que le comportement lors de l'accroissement de la température est différent du comportement lorsque celle-ci baisse. Trois températures de référence sont nécessaires au fonctionnement du système : $T_{min} < T_{nom} < T_{max}$. La production du chaud ou du froid, indiquée respectivement par un signal C à 1 ou F à 1, se passe comme suit :

- lorsque la température T devient inférieure à T_{min}, il y a production de chaud : $C \leftarrow 1$;
- la production de chaud s'arrête, *i.e.* $C \leftarrow 0$, lorsque la température devient supérieure ou égale à la valeur nominale T_{nom} ;
- lorsque la température devient supérieure ou égale à T_{max}, il y a production de froid $F \leftarrow 1$;
- la production de froid cesse lorsque la température devient inférieure à la valeur nominale T_{nom}.

Question 1 Peut-on représenter les valeurs de C et F en fonction de la température sous la forme de tables de vérité ? Justifiez.



Le système exploite un capteur de température qui fournit une information encodée sur 2 bits comme précisé ici :

b_1	b_0	cas
0	0	$T < T_{min}$
0	1	$T_{min} \leq T < T_{nom}$
1	1	$T_{nom} \leq T < T_{max}$
1	0	$T_{max} \leq T$

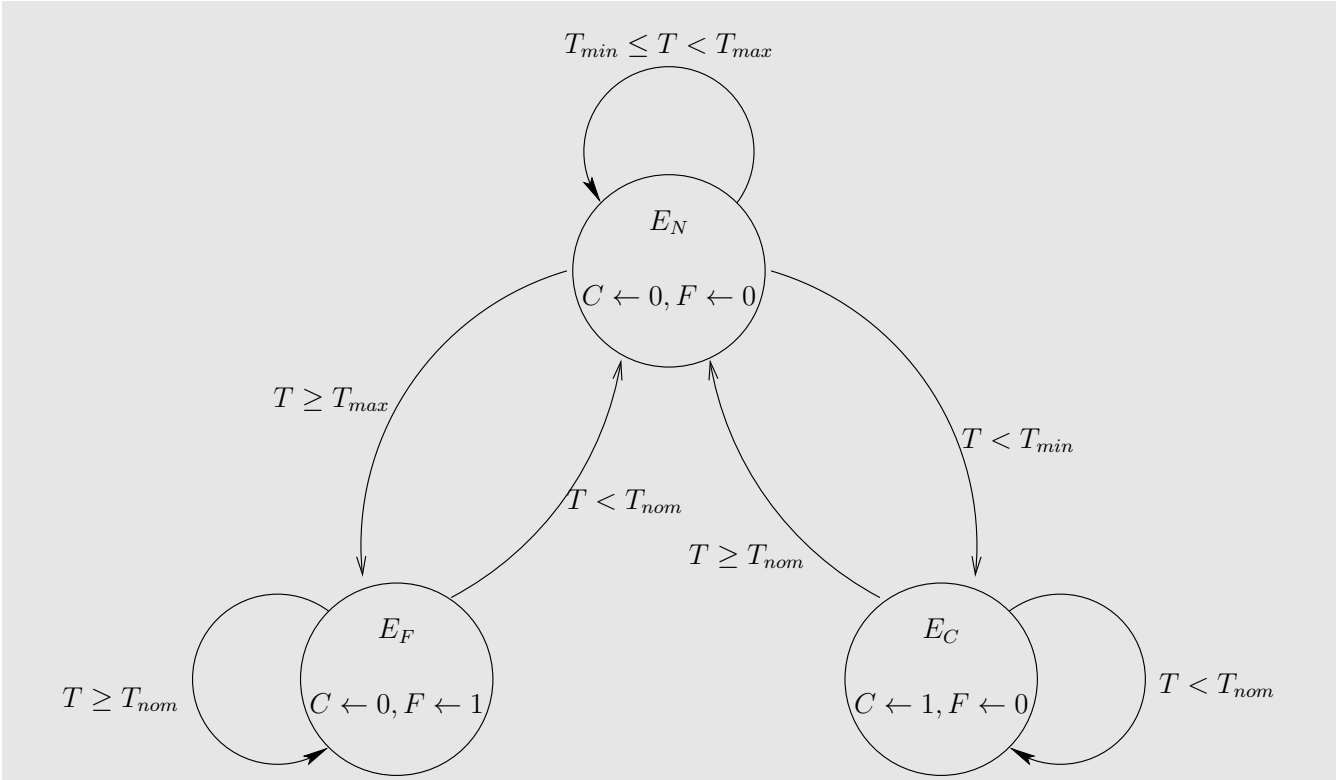
On cherche à formaliser le comportement du système comme un automate de Moore, en faisant l'hypothèse que la période de l'horloge de l'automate est très inférieure au temps qu'il faut pour que la température T du poulailler passe de T_{min} à T_{max} ou inversement, à cause de l'inertie thermique.

Question 2 Précisez quelles sont les entrées et les sorties de l'automate.

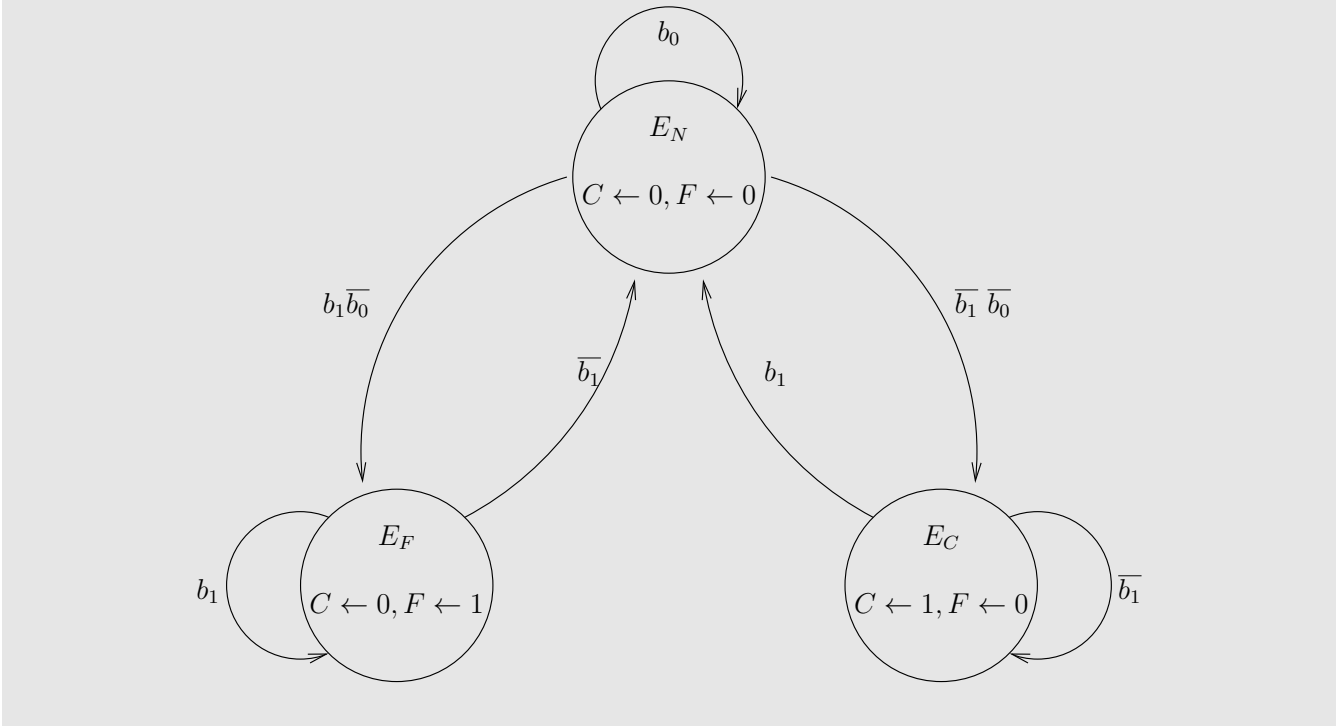
Entrées : b_1 et b_0 ;
Sorties : C et F.

Question 3 Donnez le nombre d'états de l'automate et représentez le graphe de l'automate en y incluant les conditions de transitions sous forme *d'intervalle de températures* sur les arcs et les valeurs des sorties dans les états. Transformez ensuite les intervalles de températures en conditions booléennes.

L'automate doit produire 3 paires de valeurs différentes : $C \leftarrow 0, F \leftarrow 0$ lorsque l'on se trouve à une température intermédiaire, $C \leftarrow 1, F \leftarrow 0$ lorsque l'on « vient » du froid, et $C \leftarrow 0, F \leftarrow 1$ lorsque l'on « vient » du chaud. Il y a donc 3 états. Cela donne, en température :



Cela donne, avec les conditions booléennes, et en s'assurant de respecter les conditions de complétude et d'orthogonalité :



Question 4 Proposez un codage logarithmique des états de façon à ce que les expressions des sorties C et F soient simples et donnez la table de transition qui précise les sorties et l'état futur en fonction des entrées et de l'état courant. On note Q_i les sorties du registre représentant l'état courant et D_i les entrées du registre, représentant l'état futur.

Codage des états : il suffit de choisir le code proposé par les sorties.

état	Q_1	Q_0
E_N	0	0
E_F	0	1
E_C	1	0

Table de transitions :

état courant	entrées	état futur	sorties
Q_1Q_0	b_1b_0	D_1D_0	CF
00	00	10	00
	01	00	
	10	01	
	11	00	
01	0x	00	01
	1x	01	
10	0x	10	10
	1x	00	

Avec ce choix, la fonction de génération devient 2 fils.

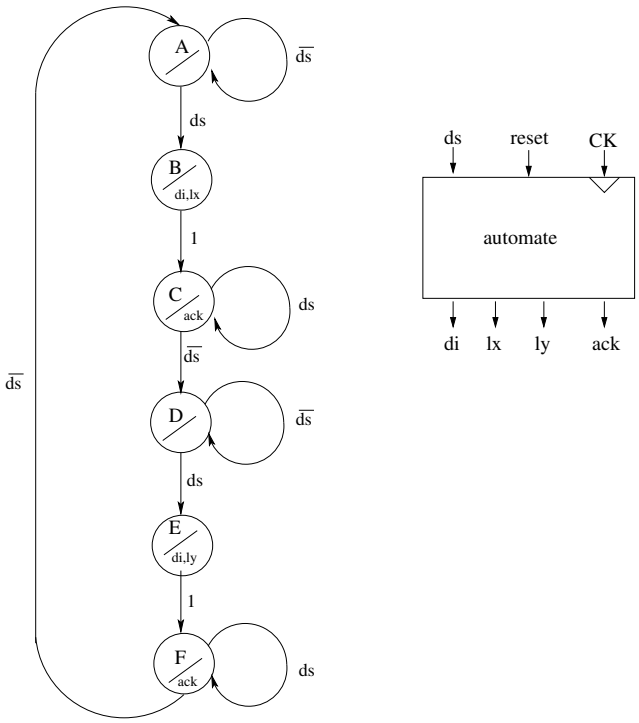
Question 5 Donnez les expressions simplifiées de C, F et des D_i, et faites le schéma en portes correspondant.

$D_0 = b_1.(Q_0 + \bar{b_0}.\bar{Q_1})$
$D_1 = \bar{b_1}.(Q_1 + \bar{b_0}.\bar{Q_0})$
$C = Q_1 et F = Q_0$

Pour aller plus loin...

Ex. 5 : automate de synchronisation

Soit un automate spécifié par le graphe ci-dessous. Outre l'entrée d'initialisation $RESET$, cet automate a une entrée : ds et 4 sorties di, ack, lx, ly. Il évolue au front montant de l'horloge CK.



Question 1 Construire la table de transition de cet automate.

Remarque : cet automate pourrait être un automate de synchronisation d'un "récepteur" de données communiquant avec un émetteur par ds (data sent, mis à 1 par l'émetteur) et ack (acknowledge par le récepteur). L'émetteur envoie 2 valeurs différentes, X puis Y, en positionnant ds à 1 durant l'envoi. Le récepteur échantillonne la donnée avec di (data in), puis la charge dans le registre X avec lx ou Y avec ly. Après le chargement, le récepteur signale qu'il a reçu la donnée avec ack à 1, puis l'émetteur remet à 0 ds.

état courant	entrées	état futur	sorties
	ds		di lx ly ack
A	0	A	0000
	1	B	
B	0	C	1100
	1	C	
C	0	D	0001
	1	C	
D	0	D	0000
	1	E	
E	0	F	1010
	1	F	
F	0	A	0001
	1	F	

Question 2 Proposez une réalisation de cet automate utilisant un codage 1 parmi n (one hot encoding) à partir de bascules D.

Besoin de 6 bascules D, reliées selon les équations suivantes.

$D_A = \bar{ds}.(Q_A + Q_F)$
$D_B = ds.Q_A$
$D_C = Q_B + ds.Q_C$
$D_D = \bar{ds}.(Q_C + Q_D)$
$D_E = ds.Q_D$
$D_F = Q_E + ds.Q_F$
$di = Q_B + Q_E$
$ack = Q_C + Q_F$
$lx = Q_B$
$ly = Q_E$

Question 3 On cherche à réaliser cet automate avec un codage logarithmique (avec 3 variables d'état, puisque'il y a 6 états). Rechercher un codage permettant de minimiser le nombre total de monômes dans les expressions de ces variables d'état et des sorties, puis donner les équations.

A priori, il faut 12 monômes pour synthétiser cet automate. On voit qu'il serait intéressant que :
- les codes de A et F soient adjacents
- les codes de C et D soient adjacents
- les codes de C et F soient adjacents.

ce qui ne donne plus que 9 monômes.
De plus, on peut coder C ou F par 000, ce qui supprime un monôme et un OU.
On propose : C codé 000, D codé 100, F codé 001, A codé 101, B codé 011, E codé 111.

Une correction pour aller plus loin donnée l'an dernier aux groupes d'Olivier.

Pour trouver un bon codage, il faut repérer et exploiter des situations favorisant des minimisations par absorption.

Situation 1 : Les sorties communes

C'est généralement la seule situation dont la connaissance est exigée.

Repérage et exploitation : pour un bit de sortie donné, tous les états l'activant doivent être placés en adjacence.

Raison : Simplification de la fonction de sortie par regroupement des valeurs à 1

exemple : B et E doivent être adjacent pour simplifier la sortie di, idem pour les états C et F avec la sortie ack.

Illustration 1 :

Supposons le codage suivant

q_2 \ q_1q_0	**00**	**01**	**11**	**10**
0	A	C	Φ	E
1	D	B	Φ	F

Ce dernier ne respecte aucune des situations décrites dans ce corrigé et mènera l'étudiant après 3 tableaux de Karnaugh à 4 variables et 4 tableaux à 3 variables, à un circuit utilisant une bonne vingtaine (à condition de factoriser un peu) de porte OR à 2 entrées et AND à 2 entrées dont au moins 5 pour la fonction de sortie. (On n'utilise pas de porte XOR dans cette exemple car les situations décrites ne permettent pas d'exploiter les portes XOR et leur utilisation fausserait les résultats).

Le codage suivant respecte la règle de la situation 1

q_2 \ q_1q_0	**00**	**01**	**11**	**10**
0	A	C	Φ	E
1	D	F	Φ	B

En réalisant les tableaux de Karnaugh, on arrivera à une quinzaine de monômes et surtout plus que 2 monômes sur la sortie.

Situation 2 : join

Connaissance à la limite du programme.

Repérage : Plusieurs transitions convergent vers un même état avec la même garde.

Exploitation : mettre tous les états de départs de ces transitions en adjacence.

Raison : Grouper les états de départs permet de simplifier la fonction de transition en lui présentant un état d'entrée simplifié.

Exemple : A et F doivent être adjacent (transitions vers A), idem pour C et D (vers D)

Illustration 2 :

Le codage suivant respecte les contraintes d'adjacences amenées par les situations 1 et 2.

q_2 \ q_1q_0	**00**	**01**	**11**	**10**
0	A	F	Φ	E
1	D	C	Φ	B

Après un peu d'entrainement supplémentaire au tableau de Karnaugh, vous devriez trouver une dizaine de monôme.

Situation 3 : Fork

A partir de là, il s'agit de connaissance facultative, mais ce qui ne vous tue pas étant censé vous rendre plus fort (au moins en archi).

Repérage : 2^k transitions quittent un même état avec une garde différente.

Exploitation : mettre tous les états d'arrivés de ces transitions en adjacence (sur une arête pour k=1, une face pour k=2... plus généralement on parle de k-cube) et transposer les adjacences sur les gardes au sein du k-cube

Raison : Simplification de la fonction de transition, car $\Theta(e, \acute{e}tat\_d\acute{e}part)$ est une combinaison de l'entrée e et de l'invariant du k-cube contenant les états de sorties.

Exemple : A et B doivent être adjacent (transitions partant de A), idem pour C et D (partant de D), D et E, A et F

q_2 \ q_1q_0	**00**	**01**	**11**	**10**
0	A	F	Φ	B
1	D	C	Φ	E

Illustration 3 :

Avec ce codage, on arrive à descendre à 8 portes OR2 ou AND2.

Situation 4 : Séparation par arc étiqueté

Repérage : Plusieurs transitions étiquetés avec la même garde

Exploitation : Placer les états de départ en adjacence sur un k-cube et les états d'arrivée sur un autre k-cube. Pour toutes les transitions concernées l'adjacence au sein du k-cube de départ doivent être conservés dans le k-cube d'arrivée.

Raison : Pour les gardes concernées, la fonction de transition ne dépend plus que de l'invariant du k-cube d'entrée pour fournir l'invariant du k-cube de sortie.

Exemple : avec la garde ds on passe de AFCD à BFCE, et avec $\bar{ds}$ de ACDF à AD

Illustration 4 :

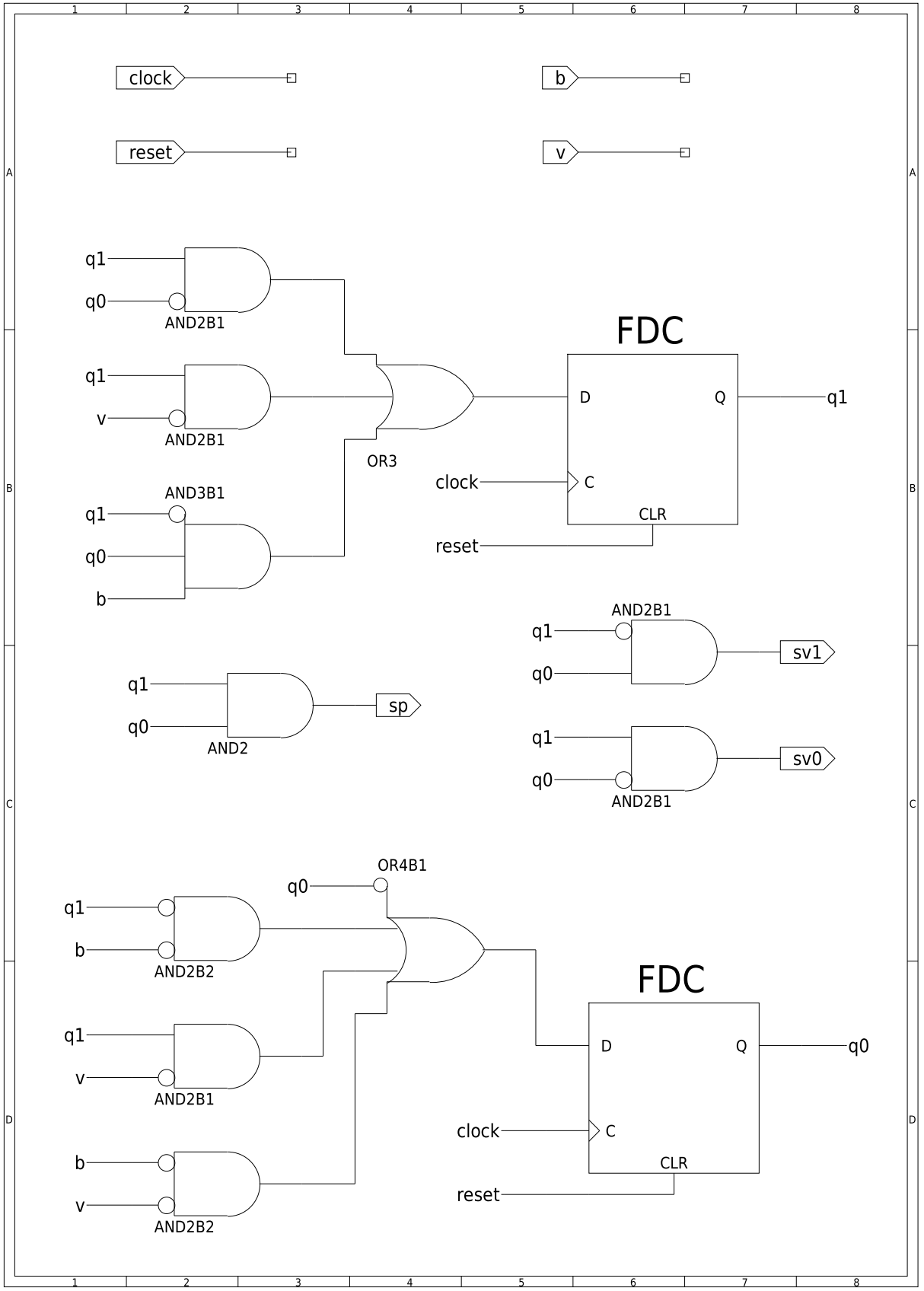
Vous remarquerez qu'il n'est plus possible de satisfaire toutes les situations exposées. Il faut faire des choix, ce qui pourrait nous emmener encore très loin. Le codage suivant ne respecte que les situations 1, 2 et 4

q_2 \ q_1q_0	**00**	**01**	**11**	**10**
0	A	F	B	Φ
1	D	C	E	Φ

Avec ce codage, on arrive à descendre à 7 portes OR2 ou AND2.

Pour conclure, cette correction présente de nombreuses manières d'optimiser la synthèse d'un automate. Seule la première est à connaitre. Le reste est là pour aiguiser votre curiosité. Il existe de nombreuses manières pour aller encore plus loin. On n'a pas parlé par exemple d'utiliser les portes XOR (on parle alors d'optimisations multi-niveau).

Pour les personnes intéressés pour aller encore plus loin sur le sujet, consulter le livre "Logic Synthesis for FSM-based Control Units" dispo à la BU de science.


clock
reset
b
v
q1
q0
AND2B1
q1
v
AND2B1
OR3
AND3B1
q1
q0
b
FDC
D
Q
q1
clock
C
CLR
reset
q1
q0
AND2
sp
AND2B1
q1
q0
sv1
q1
q0
AND2B1
sv0
OR4B1
q0
q1
b
AND2B2
q1
v
AND2B1
b
v
AND2B2
FDC
D
Q
q0
clock
C
CLR
reset

TD 6
Architecture PC/PO

Ex. 1 : PGCD

On travaille sur un circuit calculant le PGCD de deux entiers naturels strictement positifs, selon l'algorithme ci-dessous :

```
procedure PGCD is
  A, B: Positive; -- on suppose que les entiers sont codés sur 8 bits
begin
  Get(A); -- A :=A0
  Get(B); -- B :=B0
  while A /= B loop
    if A < B then
      B := B - A;
    else
      A := A - B;
    end if;
  end loop;
  Put(B);
end;
```

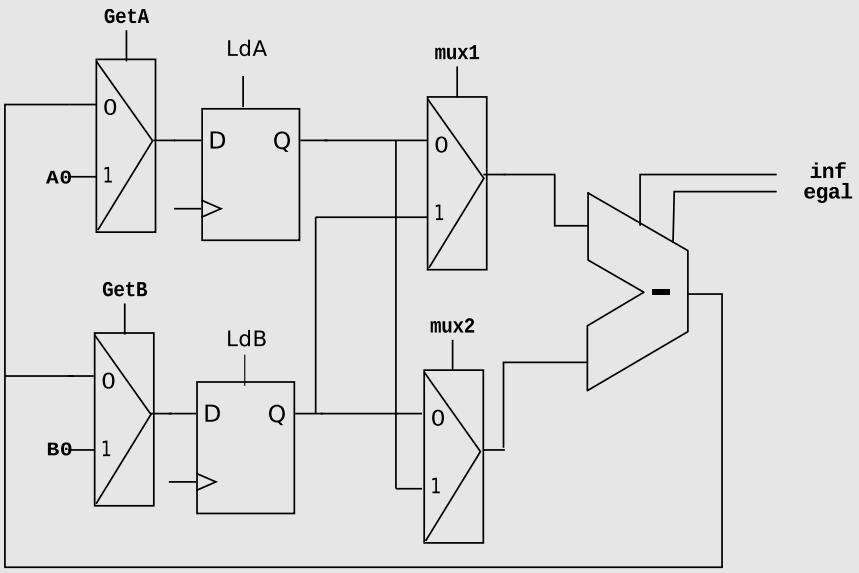
On décide qu'ici les paramètres ainsi que le résultat du calcul sont codés sur 8 bits.

Question 1 On commence par travailler sur la partie opérative du circuit PGCD. Quels composants de base peuvent être utilisés pour matérialiser les éléments de l'algorithme précédent ?

— A et B : registres sur 8 bits ;
— opérations `A /= B` et `A < B` : comparateur, soustracteur avec sorties de statut (égalité,signe), add/sub, UAL ;
— opérations `A - B` et `B - A` : soustracteur, additionneur / soustracteur, UAL.

En se rappelant ce qui a été vu au TD4, à partir d'un soustracteur 8 bits, on peut obtenir le signe du résultat (A-B) à partir de la retenue sortante (son inverse) et l'égalité (A == B) avec un NOR8 sur les bits 0-7. On suppose donc avoir un soustracteur disposant de ces deux sorties.

Question 2 En utilisant ce soustracteur et tous les composants de base nécessaires, construire la partie opérative du circuit PGCD. Identifier les signaux de compte-rendu envoyés à la partie contrôle. Penser à nommer systématiquement tous les signaux de contrôle de la PO.

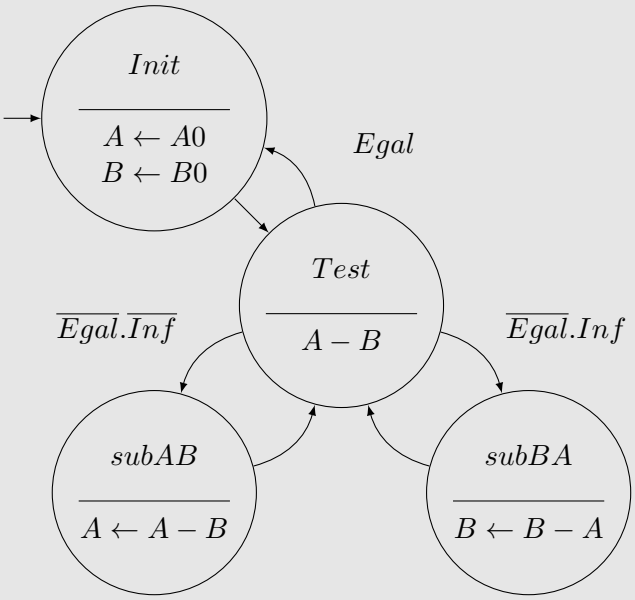


Question 3 Avec la PO construite, relever les actions pouvant être réalisées en parallèle dans l'algorithme de calcul du PGCD.

On peut charger en même temps les valeurs initiales de A et B vu qu'elles proviennent de sources différentes. On effectue en même temps la comparaison du `while` et celle du `if`.

Question 4 Dessiner l'automate de contrôle du circuit PGCD en précisant bien tous les signaux de contrôle envoyés à la PO, et en prenant en compte les signaux de compte-rendu envoyés par la PO.

On commence par dessiner l'automate sans les sorties associées aux états, mais en précisant les gardes sur les transitions (c'est à dire les compte-rendus en provenance de la PO).



Pour simplifier l'écriture, on ajoute dans chaque état les opérations à réaliser. Ces opérations sont souvent appelées transfert de registres à registres ou RTL (accronyme anglais de Register Transfert Level).

Chaque opération RTL peut être décomposée en un ensemble de signaux de contrôle envoyé de la PC à la PO. Elle représente donc une partie de la fonction de sortie de la machine d'état. Leur utilisation dans les états est donc une représentation condensée de la fonction de sortie de la PC. Avec cette notation, on suppose que les signaux de sorties non concernées par les opérations RTL de l'état de la PC sont inactif pour la logique séquentielle (registres, mémoires) ou indifférentes pour la logique combinatoire.

Sous la forme d'un tableau, la fonction de sortie s'exprime donc :

	Sorties					
Etat	$GetA$	LdA	$GetB$	LdB	$mux1$	$mux2$
$Init$	1	1	1	1	Φ	Φ
$Test$	Φ	0	Φ	0	0	0
$subAB$	0	1	Φ	0	0	0
$subBA$	Φ	0	0	1	1	1

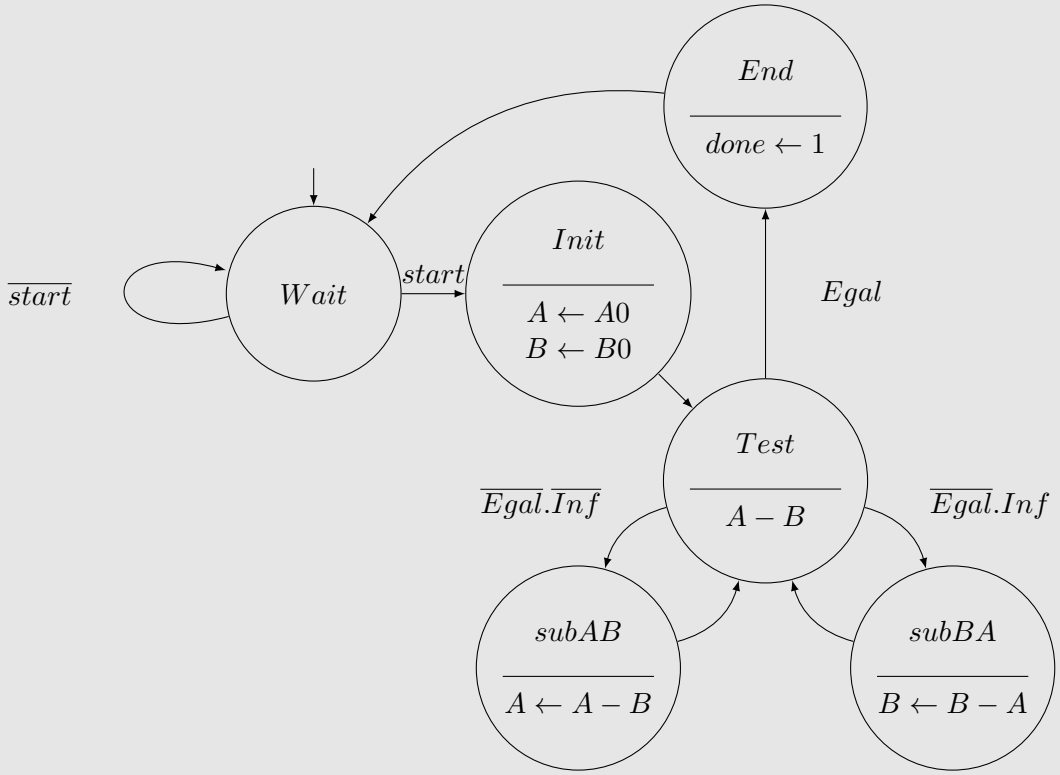
Note : on n'a pas précisé la valeur du signal *reset* envoyé aux registres de la PO, mais comme ici on est sûr que tous les registres seront affectés au moins une fois avant d'être lus, on peut décider que le *reset* de la PO est le même que celui de la PC et donc ne le faire qu'une fois au démarrage du circuit.

Le circuit PGCD sera en pratique utilisé par un autre circuit qui lui fournira ses opérandes A et B et récupérera le PGCD à la fin du calcul. Pour permettre la synchronisation entre le circuit utilisateur et le circuit PGCD, on introduit deux signaux de commandes :

— *start* est un signal envoyé par le circuit utilisateur au circuit PGCD pour lui demander de démarrer le calcul : il faut donc que les bonnes valeurs de A et B soient stabilisées en entrée du circuit PGCD quand *start* passe à 1 ;
— *done* est un signal envoyé par le circuit PGCD au circuit utilisateur pour lui signifier que le calcul est terminé et que le PGCD de A et B est disponible en sortie.

Question 5 Ajouter la gestion de ces deux signaux de commande. Quelle partie du circuit vous semble la plus adaptée pour gérer les communications avec l'extérieur du circuit PGCD ?

On doit les ajouter à l'automate de la PC :
— Le signal *start* est une entrée de la PC : on ajoute un état d'attente $Wait$ dans lequel on reste tant que $\overline{start}$;
— Le signal *done* est une sortie de la PC : on ajoute un état End par lequel on passe lorsque le calcul du PGCD est terminé, et *done* vaudra 1 dans cet état.



On met à jour le tableau des sorties de la PC pour ajouter les valeurs de *done* :

	Sorties						
Etat	*GetA*	*LdA*	*GetB*	*LdB*	*mux*1	*mux*2	*done*
Wait	Φ	0	Φ	0	Φ	Φ	0
Init	1	1	1	1	Φ	Φ	0
Test	Φ	0	Φ	0	0	0	0
subAB	0	1	Φ	0	0	0	0
subBA	Φ	0	0	1	1	1	0
End	Φ	0	Φ	0	Φ	Φ	1

Méthodologie : L'ordre des questions de ce TD représente les différentes étapes permettant de traduire un algorithme en circuit. Ces étapes sont :

1. Identification des variables à mettre en registre
2. Identification des opérations et choix des opérateurs
3. Réalisation de la PO : chaque ligne de l'algorithme définit un chemin de données entre les composants de base retenus précédemment. Lorsque le chemin passe par un composant de base déjà utilisé, un multiplexeur est ajouté sur le chemin de donnée. Le multiplexeur est commandé par un signal de contrôle.
4. Réalisation de la PC sous la forme d'un automate : chaque ligne de l'algorithme (éventuellement réécrit sous une forme optimisée) est affectée à un état. Les états sont reliés entre eux pour exprimer la séquentialité de l'algorithme. Les entrées de contrôle et les signaux de compte-rendus sont utilisés pour exprimer la fonction de transition.

TD 7
Architecture PC/PO

Ex. 1 : Bresenham

Le tracé de lignes, de cercles, d'ellipse, etc, est une opération réalisée très communément sur les cartes graphiques. Des algorithmes spécifiques ont été développés pour faire ces tracés sur des grilles de pixels (par ex. un écran) en utilisant uniquement des nombres entiers et sans recourir à la division (pour la ligne) ou la racine carrée ou les fonctions trigonométriques pour les cercles et ellipses. On se propose d'utiliser un algorithme dû à Jack Bresenham [1] qui permet de tracer un quart de cercle pour en faire la conception PC/PO.

L'algorithme est le suivant :

```
   // r est le rayon, disponible au début de l'algorithme
 1 x := 0;
 2 y := r;
 3 m := 5 − 4 × r;
 4 while x ≤ y do
       // out indique que les valeurs (x, y) sont disponibles, il n'y a pas
          d'opérateur associé
 5     out(x, y);
 6     if m > 0 then
 7         y := y − 1;
 8         m := m − 8 × y;
 9     end if
10     m := m + 8 × x + 12;
11     x := x + 1;
12 end while
```

On va utiliser la méthode présentée en cours et mise en œuvre en TD pour proposer une implantation de cet algorithme sous la forme PC/PO.

Question 1 Définissez l'ensemble des opérations à réaliser ; déduisez-en le type des unités fonctionnelles (registres ou opérateurs). On cherchera à utiliser les opérateurs *les plus simples* (c'est-à-dire qui implantent une opération arithmétique ou logique et une seule) pour chaque opération. On rappelle que les soustracteurs fournissent (possiblement entre autres) les informations suivantes : z qui vaut 1 si le résultat est nul, 0 sinon et s qui vaut 1 si le résultat est strictement négatif, 0 sinon.

Opérations à réaliser : $5 - 4 * r$, $x <= y$, $m > 0$, $y - 1$, $m - 8 * y$, $x + 1$, $m + 8 * x + 4$. D'où :
- multiplications par des constantes qui sont des puissances entières de 2, d'où simple décalage des fils ;
- soustracteurs pour les $-$ et les comparaisons ;
- additionneurs pour les $+$.

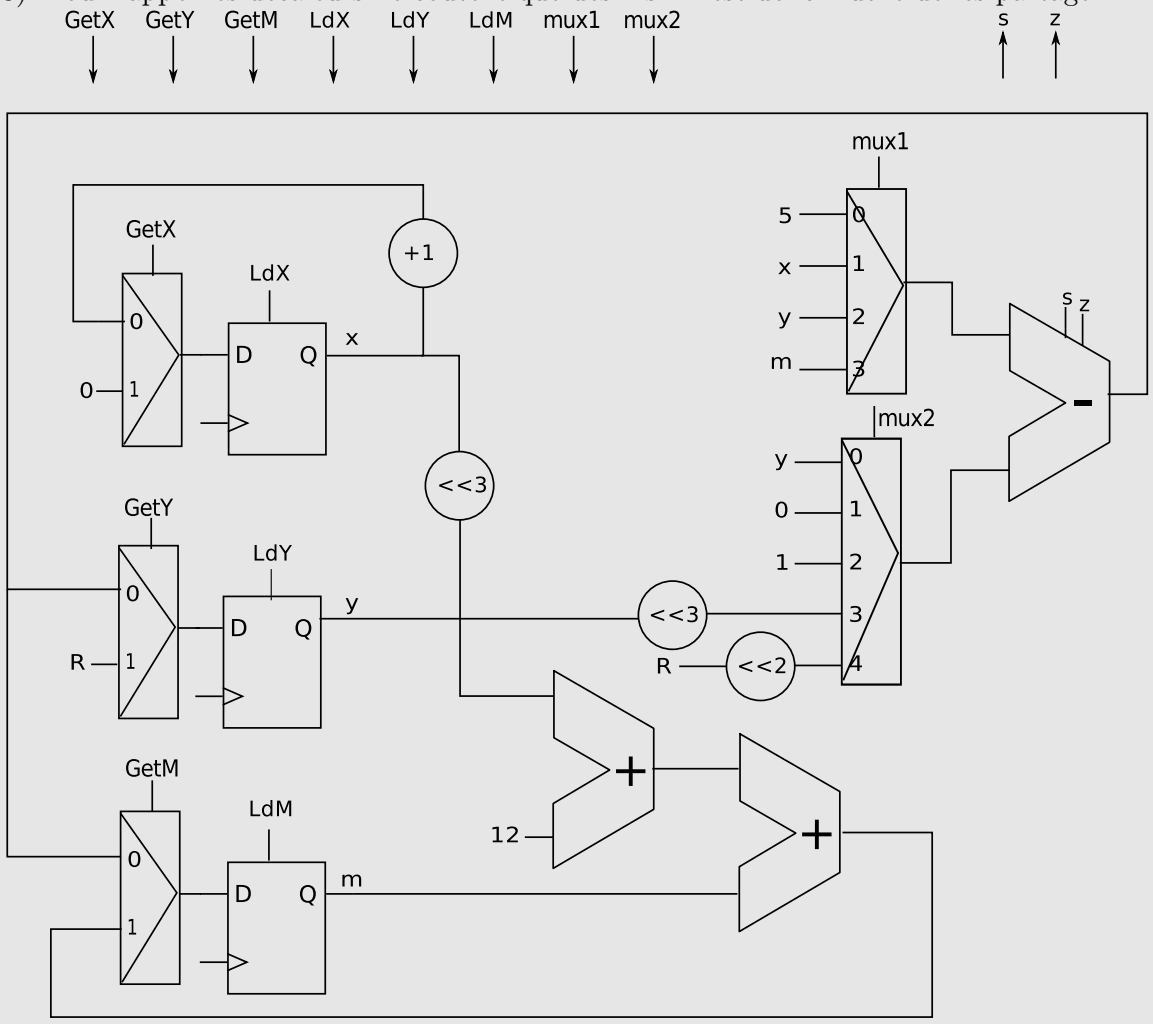
Question 2 Quelles opérations peut-on réaliser en parallèle ?
Réécrivez le programme en utilisant 1) la notion d'affectation concurrente présentée en cours et 2) en remplaçant les opérations par les opérations des opérateurs « simples » déterminés précédemment. Rappel. : $(\alpha, \beta) \leftarrow (3, 1)$ affecte 3 dans α et 1 dans β.

1. *"A linear algorithm for incremental digital display of circular arcs", Jack Bresenham, Communication of the ACM, Feb 1977, vol. 20, n. 2.*

```
   // r est le rayon, disponible au début de l'algorithme
1  (x, y, m) := (0, r, 5 − (r << 2));
   // l'équation booléenne s + z indique ≤ 0
2  while x − y ≤ 0 do
3  |  out(x, y);
   |  // l'équation booléenne s + z indique > 0
4  |  if m − 0 > 0 then
5  |  |  y := y − 1;
6  |  |  m := m − (y << 3);
7  |  end if
8  |  (m, x) := (m + (x << 3) + 12, x + 1);
9  end while
```

Question 3 Donnez le nombre d'opérateurs de chaque type nécessaire pour exécuter une ligne de l'algorithme en 1 cycle et proposez une partie opérative interconnectant les unités fonctionnelles. Indiquer clairement les signaux de commandes (sélecteurs de multiplexeurs, signaux de chargement de registres, etc) et les comptes-rendus.

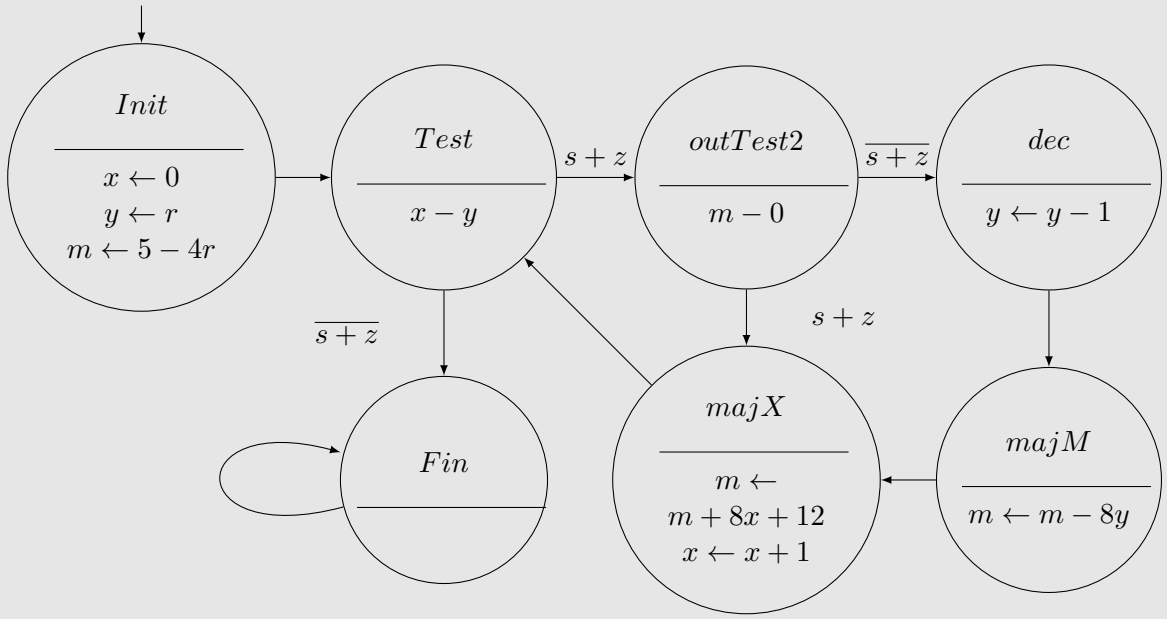
1 soustracteur, 3 additionneurs (2 additionneurs et 1 incrémenteur), 3 décaleurs constants(1 «2 et 2 «3). Pour rappel les décaleurs ne coûtent que des fils : il est donc inutile de les partager.



Pour aller plus loin, on peut remarquer que cette PO peut être optimisée au niveau des multiplexeurs du soustracteur. y apparait sur les 2 ports de cet opérateurs. En testant la condition du while

en faisant $y - x$, on peut économiser une entrée sur le port 1 du soustracteur.

Question 4 Proposez une partie contrôle qui pilote cette partie opérative. On fait l'hypothèse que l'entrée r est magiquement disponible lorsqu'on en a besoin, et que *out* est un état dans lequel on passe un cycle et un seul. Spécifiez les opérations de transfert registre à registre (RTL) dans chaque état. Donnez la fonction de sortie sous la forme d'un tableau.



Sous la forme d'un tableau, la fonction de sortie s'exprime donc :

	Sorties							
Etat	$GetX$	LdX	$GetY$	LdY	$GetM$	LdM	$mux1$	$mux2$
$Init$	1	1	1	1	0	1	0	4
$Test$	Φ	0	Φ	0	Φ	0	1	0
$outTest2$	Φ	0	Φ	0	Φ	0	3	1
dec	Φ	0	0	1	Φ	0	2	2
$majM$	Φ	0	Φ	0	0	1	3	3
$majX$	0	1	Φ	0	1	1	Φ	Φ
Fin	Φ	Φ	Φ	Φ	Φ	Φ	Φ	Φ

Question 5 Modifiez l'algorithme en utilisant la notion d'affectation conditionnelle pour faire disparaître le **if**. Pour mémoire : $\nu \leftarrow \gamma ? \alpha : \beta$ affecte α dans ν si $\gamma = 1$, β sinon.

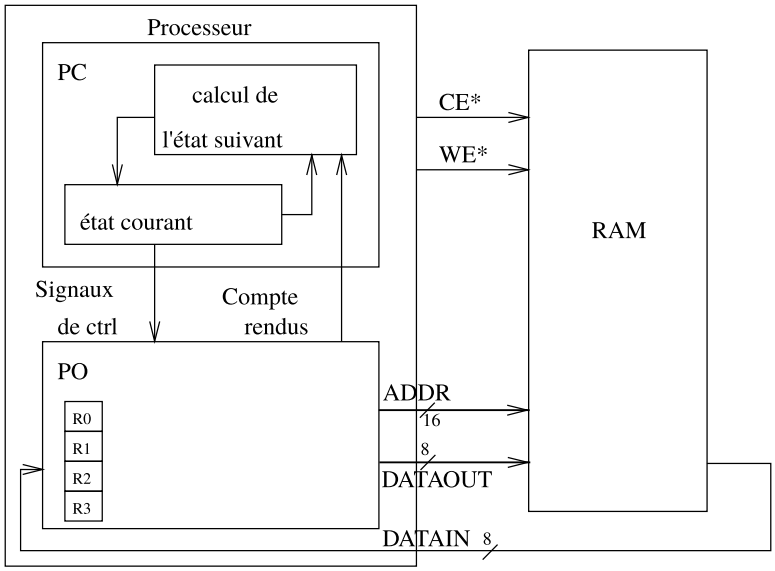
```
  // r est le rayon, disponible au début de l'algorithme
1 (x, y, m) := (0, r, 5 − (r << 2));
  // l'équation booléenne s + z indique x ≤ y
2 while x − y ⇒ s + z do
3 |   out(x, y);
  |   // l'équation booléenne s̄ + z̄ indique m > 0
4 |   (x, y, m) := (m > 0)?(x + 1, m + ((x − y) << 3) + 20, y − 1) : (x + 1, m + (x << 3) + 12, y);
5 end while
```

Pour aller plus loin...
Question 6 Proposez une PO, puis une PC correspondant à la question précédente.

Pour aller plus loin...
Question 7 En repartant de la forme initiale de l'algorithme, proposez une PO n'utilisant qu'un additioneur/soustracteur. Donnez ensuite la PC correspondante.

TD 8
Conception d'un processeur : partie opérative

L'objectif de ce TD et du suivant est de construire un processeur capable d'effectuer des instructions très simples. Le processeur construit sera utilisé dans les deux séances de TD suivantes. On suppose que ce processeur est équipé de 4 registres internes de 8 bits chacun (nommés `r0`, `r1`, `r2` et `r3`), d'un bus adresse sur 16 bits et de 2 bus donnée (lecture et écriture) sur 8 bits. On rappelle l'architecture globale du processeur et sa liaison avec la mémoire asynchrone :



Le jeu d'instructions de ce processeur est le suivant :

- `st ri, @mem` stocke le contenu du registre `ri` dans l'octet situé à l'adresse `@mem` ;
- `ld @mem, ri` stocke le contenu de l'octet situé à l'adresse `@mem` dans le registre `ri` ;
- `op rs, rd` stocke dans le registre `rd` le résultat de l'opération `rd op rs`.

Les opérations à deux opérandes supportées par ce processeur sont l'addition (instruction `add`), la soustraction (instruction `sub`), la conjonction (instruction `and`), la disjonction (instruction `or`) et la disjonction exclusive (instruction `xor`).

Les opérations unaires supportées (`rd := op rd`) sont la négation (instruction `not`), le décalage d'un bit vers la gauche (instruction `shl`) et le décalage arithmétique d'un bit vers la droite (instruction `shr`).

On donne également l'algorithme exécuté par le processeur pour récupérer, décoder et exécuter les instructions d'un programme situé à l'adresse $0x4000$:

```
1  PC ← 0x4000;
2  while True do
3  |  IR ← MEM(PC) ;
4  |  PC ← PC + 1 ;
5  |  switch IR do
6  |  |  case add :  rd ← rd + rs  ;                    // rd, rs ∈ {r0, r1, r2, r3}
7  |  |  ;
8  |  |  case sub :  rd ← rd − rs ;
9  |  |  ;
10 |  |  case and :  rd ← rd AND rs ;
11 |  |  ;
12 |  |  case or :  rd ← rd OR rs ;
13 |  |  ;
14 |  |  case xor :  rd ← rd XOR rs ;
15 |  |  ;
16 |  |  case not :  rd ← NOT rd ;
17 |  |  ;
18 |  |  case shl :  rd ← rd << 1 ;
19 |  |  ;
20 |  |  case shr :  rd ← rd >> 1 ;
21 |  |  ;
22 |  |  case st :
23 |  |  |  AD(15 : 8) ← MEM(PC) ;
24 |  |  |  PC ← PC + 1;
25 |  |  |  AD(7 : 0) ← MEM(PC);
26 |  |  |  PC ← PC + 1;
27 |  |  |  MEM(AD) ← ri  ;                           // ri ∈ {r0, r1, r2, r3}
28 |  |  case ld :
29 |  |  |  AD(15 : 8) ← MEM(PC) ;
30 |  |  |  PC ← PC + 1;
31 |  |  |  AD(7 : 0) ← MEM(PC);
32 |  |  |  PC ← PC + 1;
33 |  |  |  ri ← MEM(AD) ;
34 |  |  end case
35 |  endsw
36 end while
```

On remarquera que l'algorithme suppose que les instructions d'accès à la mémoire sont sur 3 mots et les autres instructions sur un seul mot.

Il s'agit donc de concevoir le processeur sur le modèle PC-PO.

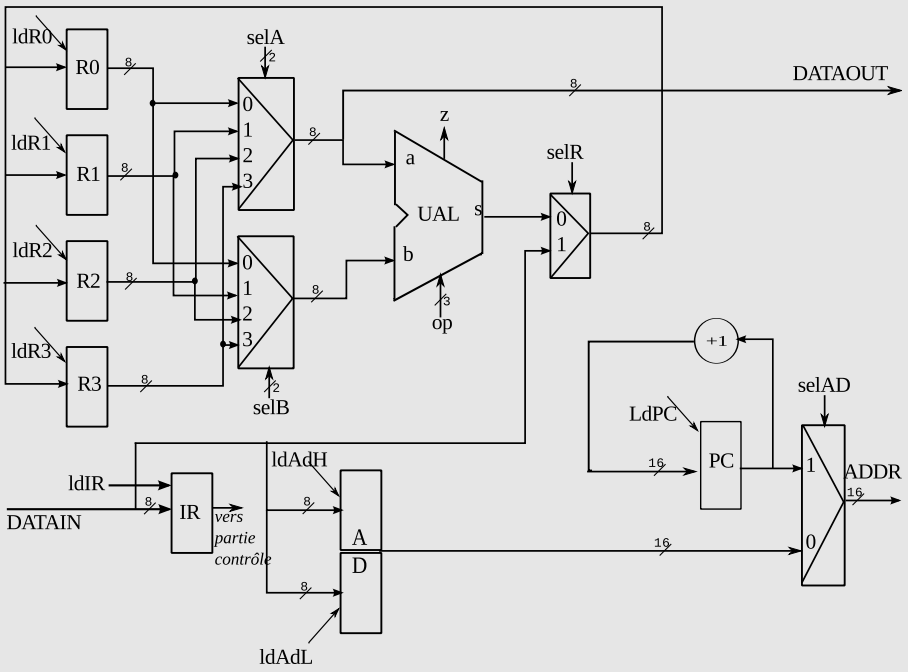
Question 1 Identifier et lister les registres dans cet algorithme. Préciser leur taille, puis expliquer leur rôle et leur fonctionnement.

- `R0`, `R1`, `R2` et `R3`, registres programmables sur 8 bits chacun ; ces registres sont directement utilisable par le programmeur. Un registre peut être chargé soit par le résultat d'un calcul (valeur en sortie de l'`UAL`), soit par une donnée lue en mémoire (arrivant sur `dataIN`). La valeur de chaque registre peut être envoyée vers la mémoire (sur `dataOUT`).
- `AD` est un registre 16 bits pouvant contenir une adresse (registre de travail) ; il est chargé avec une adresse en 2 étapes : chargement de l'octet d'adresse poids fort puis de l'octet d'adresse poids faible.
- `PC` est le registre 16 bits contenant l'adresse de l'instruction courante ; il peut être incrémenté.
- `IR` est un registre 8 bits contenant le *codop* de l'instruction en cours d'exécution ;

Question 2 Définir l'ensemble des opérations à réaliser. Proposer des opérateurs pour réaliser ces actions, en cherchant à utiliser le moins d'opérateurs possible.

Besoin d'une UAL 8 bits pour les opérations sur les registres `R0`, `R1`, `R2` et `R3` et d'un incrémenteur sur 16 bits pour pouvoir faire l'opération `PC + 1` en une étape.

Question 3 Proposez une partie opérative interconnectant les unités fonctionnelles identifiées (utiliser le composant décrit en annexe). Indiquer clairement les signaux de commandes (sélecteurs de mux, signaux de chargement de registres, etc) et les comptes-rendus.



En plus des signaux indiqués sur la figure, chacun des registres reçoit un signal d'horloge *clk* et le signal de reset, *rst*, qui les initialise à 0, sauf PC qui est initialisé à 0x4000.

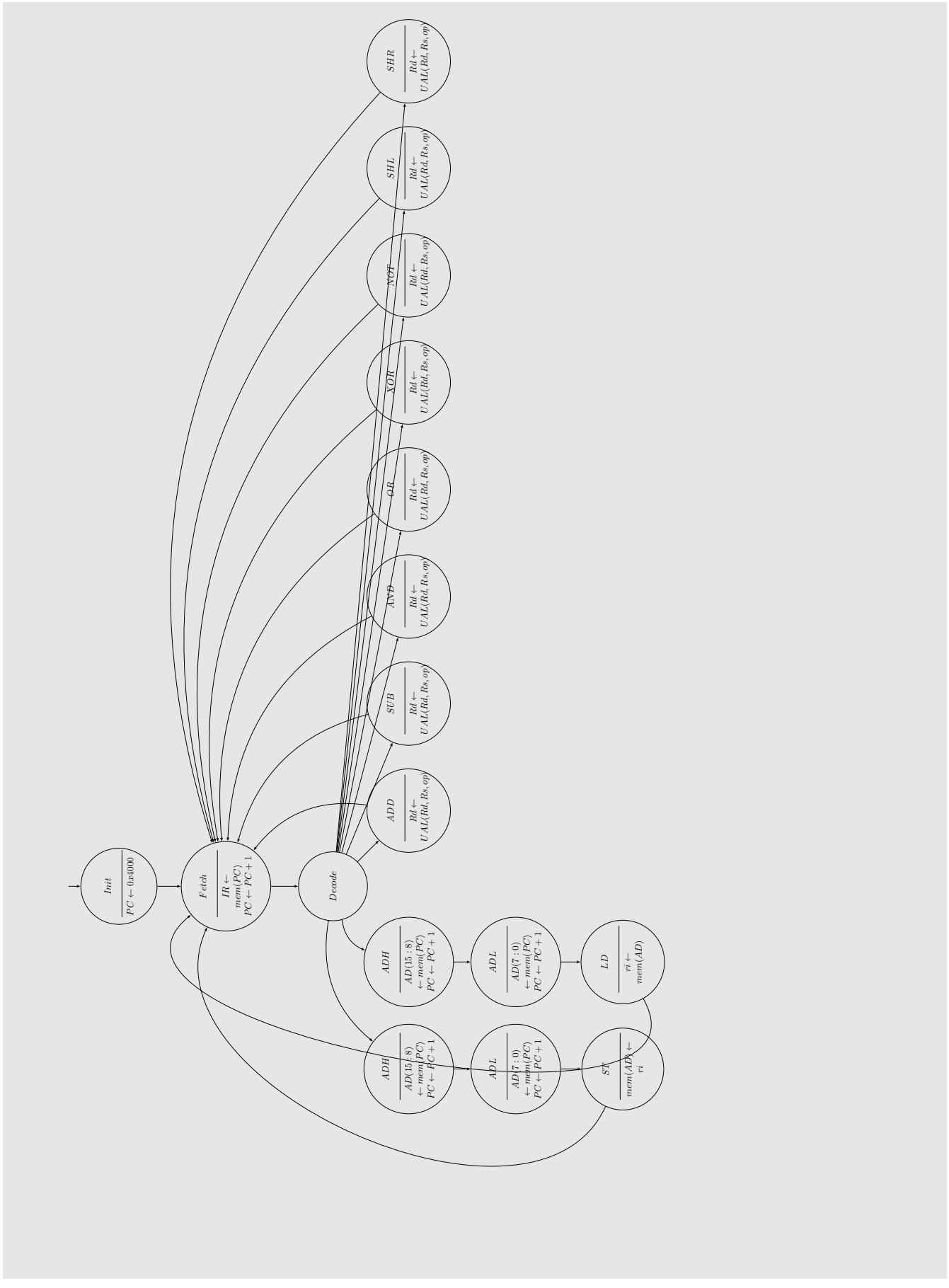
Le registre 16 bits *AD* est réalisé à base de 2 registres 8 bits chargés indépendamment par *ldAdH* et *ldAdL*.

Question 4 Repérer les actions de l'algorithme pouvant être réalisées en parallèle sur votre PO. Reécrire l'algorithme en utilisant la notion d'écriture concurrente.

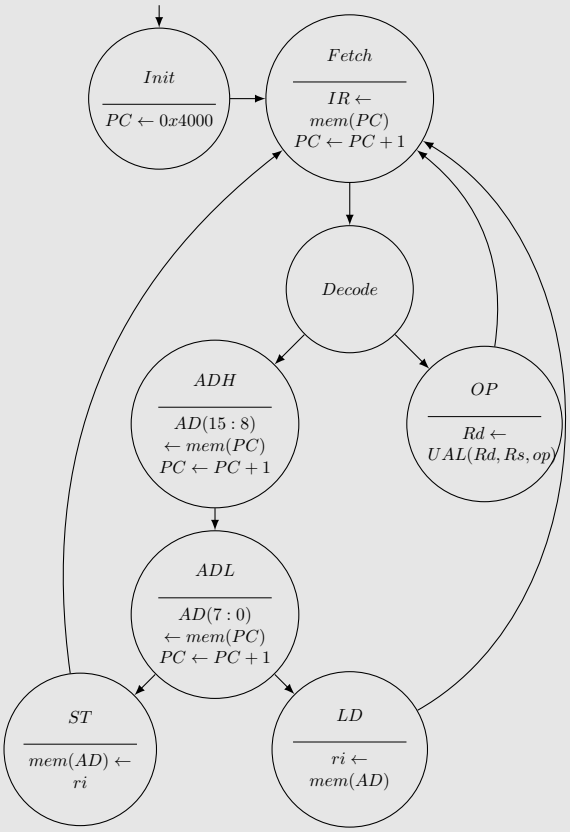
Pour piloter la PO, proposer une machine d'état correspondant au nouvel algorithme en précisant uniquement les opérations de transfert de registre à registre [1]. Ne préciser ni les signaux de contrôle ni les conditions sur transitions pour l'instant.

1. C'est à dire les opérations réalisées dans l'algorithme décrivant le processeur.

```
1  PC ← 0x4000;
2  while True do
3  |  (IR, PC) ← (MEM(PC), PC + 1) ;
4  |  switch IR do
5  |  |  case add : rd ← rd + rs ;                    // rd, rs ∈ {r0, r1, r2, r3}
6  |  |  ;
7  |  |  case sub : rd ← rd − rs ;
8  |  |  ;
9  |  |  case and : rd ← rd AND rs ;
10 |  |  ;
11 |  |  case or : rd ← rd OR rs ;
12 |  |  ;
13 |  |  case xor : rd ← rd XOR rs ;
14 |  |  ;
15 |  |  case not : rd ← NOT rd ;
16 |  |  ;
17 |  |  case shl : rd ← rd << 1 ;
18 |  |  ;
19 |  |  case shr : rd ← rd >> 1 ;
20 |  |  ;
21 |  |  case st :
22 |  |  |  (AD(15 : 8), PC) ← (MEM(PC), PC + 1) ;
23 |  |  |  (AD(7 : 0), PC) ← (MEM(PC), PC + 1);
24 |  |  |  MEM(AD) ← ri ;                           // ri ∈ {r0, r1, r2, r3}
25 |  |  case ld :
26 |  |  |  (AD(15 : 8), PC) ← (MEM(PC), PC + 1) ;
27 |  |  |  (AD(7 : 0), PC) ← (MEM(PC), PC + 1);
28 |  |  |  ri ← MEM(AD) ;
29 |  |  end case
30 |  endsw
31 end while
```



Question 5 Simplifier la PC de manière à réduire le nombre d'états successeurs de l'état "decode" (correspondant au test du switch).



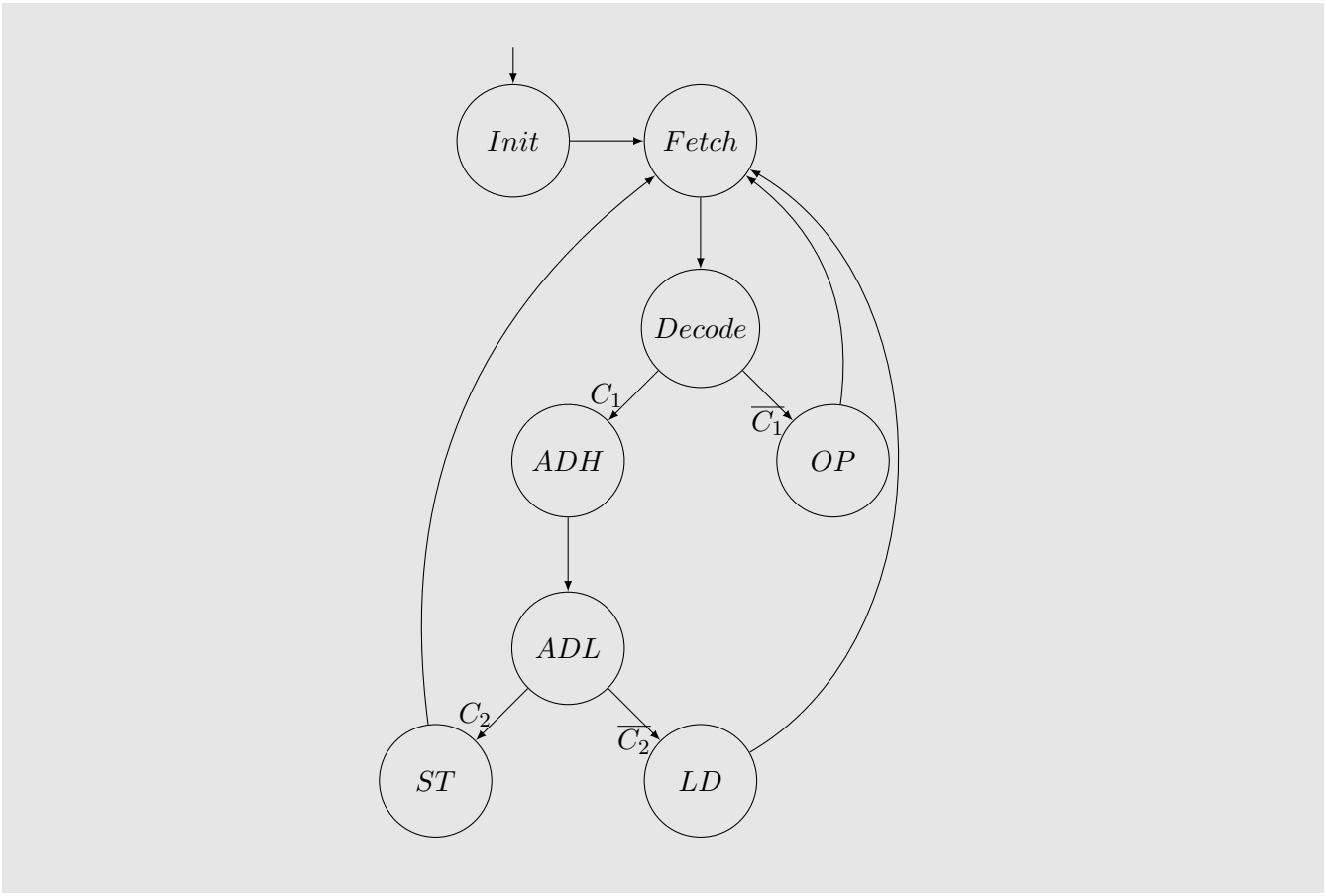
On va chercher maintenant à écrire les valeurs des conditions sur les transitions de la partie contrôle. Ces conditions de transitions dépendent du codage retenu pour chacune des instructions. Pour l'instant, on va utiliser des fonctions booléennes (f(IR)) :

— *AccesMem(IR)* : fonction exprimant si le codop de l'instruction stockée dans IR est un accès à la mémoire ;
— *AccesEcr(IR)* : fonction exprimant si le codop de l'instruction stockée dans IR est un accès à la mémoire en écriture.

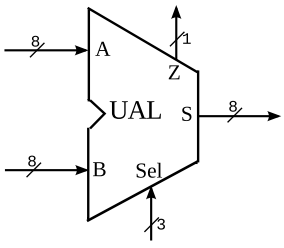
Question 6

Complétez les conditions manquantes sur les transitions de votre automate.

$C_1 = AccesMem(IR)$ et $C_2 = AccesEcr(IR)$



Annexe : UAL à utiliser dans ce TD



Sel	S
000	A or B
001	A xor B
010	A and B
011	not A
100	A + B
101	A - B
110	A « 1
111	A » 1

Z=1 ssi S=0

Pour aller plus loin...
Question 7 Construire l'UAL décrite ci-dessus en utilisant l'UAL vu au TD5, dont le fonctionnement est rappelé ci-dessous, et deux multiplexeurs 8 bits 2 vers 1. Donner les fonctions numériques dépendant des bits de **sel** permettant de contrôler l'UAL (via CI, M, f0, f1, f2 et f3) et les multiplexeurs.

Opération	Mode M	f0 f1 f2 f3	Résultat
add	1	1010	F= A + B + CI
sub	1	0101	F= A - B + 1 - CI
notA	0	1111	F=not(A)
xor	0	1010	F=A XOR B
or	0	0010	F = A OR B
and	0	0100	F = A AND B

$sel_1 sel_0$ / sel_2	**00**	**01**	**11**	**10**
0	ϕ	ϕ	ϕ	ϕ
1	0	1	ϕ	ϕ

CI

$sel_1 sel_0$ / sel_2	**00**	**01**	**11**	**10**
0	0	0	0	0
1	1	1	ϕ	ϕ

M

$sel_1 sel_0$ / sel_2	**00**	**01**	**11**	**10**
0	0	1	1	0
1	1	0	ϕ	ϕ

f_0

$sel_1 sel_0$ / sel_2	**00**	**01**	**11**	**10**
0	0	0	1	1
1	0	1	ϕ	ϕ

f_1

$sel_1 sel_0$ / sel_2	**00**	**01**	**11**	**10**
0	1	1	1	0
1	1	0	ϕ	ϕ

f_2

$sel_1 sel_0$ / sel_2	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	ϕ	ϕ

f_3

Pour mux1, on aura $sel_0.sel_1$ sur le mux1 et sel_2 sur le mux2.

TD 9
Conception d'un processeur : partie contrôle et codage des instructions

L'objectif de ce TD est de compléter la partie contrôle du processeur par la génération des signaux de contrôle vers la partie opérative, en choisissant un codage astucieux des instructions.

Question 1 On donne les fonctions booléennes suivantes :

- *selRs(IR)*, *selRd(IR)*, *selRi(IR)* : Fonction prenant les 8 bits de IR en entrée et retournant les 2 bits déterminant le numéro de registre Rs,Rd ou Ri de l'instruction décodée.
- *selUAL(IR)* : Fonction retournant l'opération réalisée par l'instruction en la codant sur 3 bits selon le codage de l'UAL fournit en annexe.

Exprimer, sous forme de tableau, pour chaque état de la PC la valeur des signaux de contrôle en fonction de *selRs(IR)*, *selRd(IR)*, *selRi(IR)*, *selUAL(IR)*.

Signaux États	Fetch	Decode	OP	ADH	ADL	ST	LD	Init
CE^*	0	1	1	0	0	0	0	1
WE^*	1	x	x	1	1	0	1	x
$ldPC$	1	0	0	1	1	0	0	0
$selAD$	1	x	x	1	1	0	0	x
$ldAdH$	0	0	0	1	0	0	0	0
$ldAdL$	0	0	0	0	1	0	0	0
$ldIR$	1	0	0	0	0	0	0	0
$selR$	x	x	0	x	x	x	1	x
$ldR0$	0	0	$selRd(IR)(0:1) = "00"$	0	0	0	$selRi(IR)(0:1) = "00"$	0
$ldR1$	0	0	$selRd(IR)(0:1) = "01"$	0	0	0	$selRi(IR)(0:1) = "01"$	0
$ldR2$	0	0	$selRd(IR)(0:1) = "10"$	0	0	0	$selRi(IR)(0:1) = "10"$	0
$ldR3$	0	0	$selRd(IR)(0:1) = "11"$	0	0	0	$selRi(IR)(0:1) = "11"$	0
$selA$	xx	xx	$selRd(IR)$	xx	xx	$selRi(IR)$	xx	xx
$selB$	xx	xx	$selRs(IR)$	xx	xx	xx	xx	xx
op	xxx	xxx	$selUAL(IR)$	xxx	xxx	xxx	xxx	xxx

On peut remarquer que les équations pour les ldRi sont celles d'un décodeur d'adresse ; en principe ce décodeur d'adresse est dans la PO et non dans la PC. Si les signaux $ldR0$, $ldR1$, $ldR2$, $ldR3$ sont générés à l'aide d'un décodeur : il faut générer un signal intermédiaire $écritureR$ qui est à 1 dans l'état OP et dans l'état LD. La valeur décodée $selRd(1:0)$ est le numéro du registre à charger, une sortie du décodeur ne peut être active que si le signal $écritureR$ est à 1.

Question 2 Proposer un codage astucieux de chaque instruction en essayant de minimiser les fonctions utilisées précédemment.

On cherche à faire des fonctions identité et il se trouve qu'on a l'espace nécessaire pour le faire :
— le numéro d'un registre se code sur 2 bits ;
— 1 bit pour AccesMem permet de répartir les instructions en deux catégories (opérations et accès mémoire) ;
— 3 bits pour codop pour coder les 8 opérations ;
— les adresses mémoires sont codées sur 16 bits.

Pour les opérations, on aura donc :

7	6 5 4	3 2	1 0
AccesMem (= 0)	selUAL	selRs	selRd

selUAL vaut selon l'annexe du td précédent :
`or` = 000, `xor` = 001, `and` = 010, `not` = 011, `add` = 100, `sub` = 101, `shl` = 110, `shr` = 111.

et on codera les bits (2 :3) à "00" pour les opérations unaires.

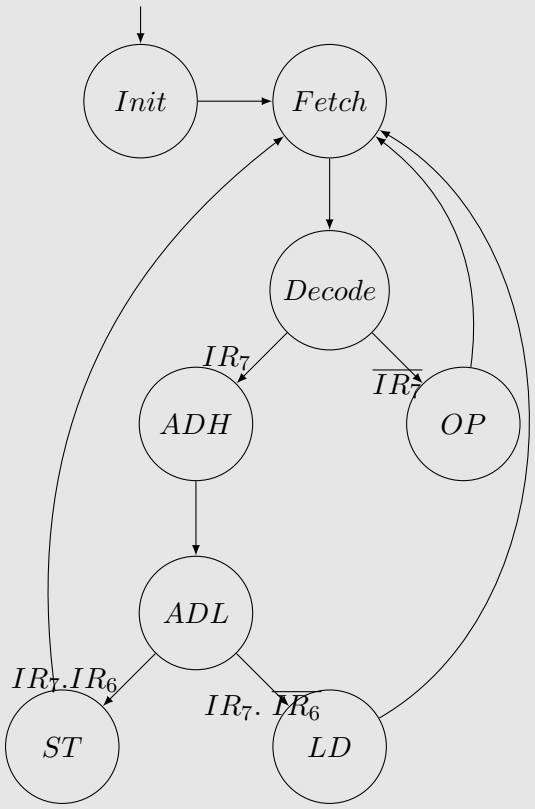
Pour les accès mémoire, on choisit de « gâcher » de la place pour avoir des instructions codées sur un nombre entier d'octets (ce qui permettra aussi d'étendre facilement le jeu d'instruction du processeur), donc :

7	6	5 4 3 2	1 0	7 0	7 0
AccesMem (= 1)	AccesEcr (= 1*pourst*)	inutilisé	selRi	adresse (1eroctet)	adresse (2^{e}octet)

Remarque : on pourrait placer AccesEcr où on veut. Par contre, il faut préciser qu'il faut placer selRi comme selRd pour avoir les mêmes fonctions dans l'état OP et l'état LD pour les signaux ld, et dans l'état OP et l'état ST pour selA.

On peut préciser qu'on aurait pu faire un choix différent pour le codage des adresses en mémoire. Ici en *big-endian* (l'octet de poids fort en premier), mais *little-endian* (l'octet de poids faible en premier) existe aussi voir cours CEP.

Question 3 Substituez dans le graphe d'états et dans le tableau des signaux de commandes les fonctions booléennes que nous avions définies de façon abstraite par des fonctions des bits du registre `IR`.



A noter : comme dans ce graphe, IR_6 n'est testé que quand on sait que IR_7 est à 1, on peut simplifier les conditions de transitions de l'état ADH aux états ST ou LD, en ne testant que IR_6.

Signaux **États**	Fetch	Decode	OP	ADH	ADL	ST	LD	Init
CE^*	0	1	1	0	0	0	0	1
WE^*	1	x	x	1	1	0	1	x
$ldPC$	1	0	0	1	1	0	0	0
$selAD$	1	x	x	1	1	0	0	x
$ldAdH$	0	0	0	1	0	0	0	0
$ldAdL$	0	0	0	0	1	0	0	0
$ldIR$	1	0	0	0	0	0	0	0
$selR$	x	x	0	x	x	x	1	x
$ldR0$	0	0	$\overline{IR_0}.\overline{IR_1}$	0	0	0	$\overline{IR_0}.\overline{IR_1}$	0
$ldR1$	0	0	$IR_0.\overline{IR_1}$	0	0	0	$IR_0.\overline{IR_1}$	0
$ldR2$	0	0	$\overline{(IR_0)}.IR_1$	0	0	0	$\overline{(IR_0)}.IR_1$	0
$ldR3$	0	0	$IR_0.IR_1$	0	0	0	$IR_0.IR_1$	0
$selA$	xx	xx	IR_1IR_0	xx	xx	IR_1IR_0	xx	xx
$selB$	xx	xx	IR_3IR_2	xx	xx	xx	xx	xx
op	xxx	xxx	$IR_6IR_5IR_4$	xxx	xxx	xxx	xxx	xxx

Pour aller plus loin...

Question 4 Proposez une réalisation de la partie contrôle à partir de bascules D et de portes, en codage 1 parmi n.

TD 10
Du programme en mémoire aux actions dans le processeur

L'objectif de ce TD est de détailler le fonctionnement du processeur construit aux séances précédentes. Le codage du jeu d'instructions pour cette séance est le suivant :

Instruction	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Opérations de la forme : $rd := rd\ op\ rs$								
`or rs, rd`	0	0	0	0	rs_1	rs_0	rd_1	rd_0
`xor rs, rd`	0	0	0	1	rs_1	rs_0	rd_1	rd_0
`and rs, rd`	0	0	1	0	rs_1	rs_0	rd_1	rd_0
`add rs, rd`	0	1	0	0	rs_1	rs_0	rd_1	rd_0
`sub rs, rd`	0	1	0	1	rs_1	rs_0	rd_1	rd_0
Opérations de la forme : $rd := op\ rd$								
`not rd`	0	0	1	1	0	0	rd_1	rd_0
`shl rd`	0	1	1	0	0	0	rd_1	rd_0
`shr rd`	0	1	1	1	0	0	rd_1	rd_0
Chargement : $rd := MEM(AD)$								
`ld AD, rd`	1	0	0	0	0	0	rd_1	rd_0
	ADH							
	ADL							
Stockage : $MEM(AD) := rs$								
`st rs, AD`	1	1	0	0	0	0	rs_1	rs_0
	ADH							
	ADL							

Question 1 Soit le contenu de la mémoire donné ci-dessous. Ré-écrire ce programme en langage d'assemblage. Que fait ce programme ?

Contenu de la mémoire :

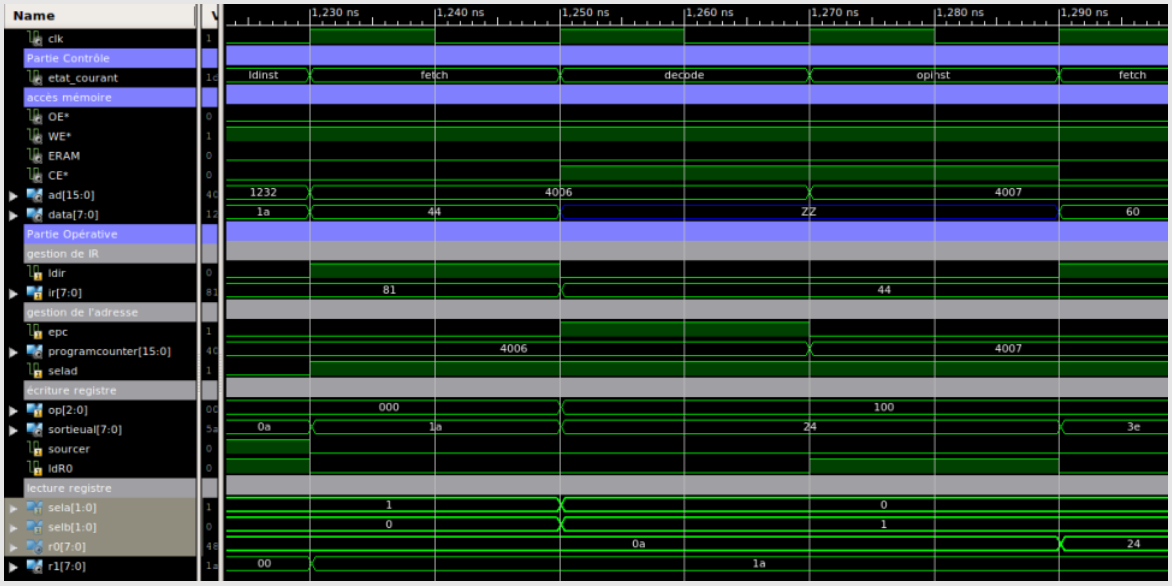
Adresse	Valeur
$0x1230$	$0x00$
$0x1231$	$0x0A$
$0x1232$	$0x1A$
$0x1233$	$0x12$
...	...
$0x4000$	$0x80$
$0x4001$	$0x12$
$0x4002$	$0x31$
$0x4003$	$0x81$
$0x4004$	$0x12$
$0x4005$	$0x32$
$0x4006$	$0x44$
$0x4007$	$0x60$
$0x4008$	$0x81$
$0x4009$	$0x12$
$0x400A$	$0x33$
$0x400B$	$0x54$
$0x400C$	$0xC0$
$0x400D$	$0x12$
$0x400E$	$0x30$
...	...

```
ld 0x1231, r0   -- r0 := 10
ld 0x1232, r1   -- r1 := 26
add r1, r0      -- r0 := 36
shl r0          -- r0 := 72
ld 0x1233, r1   -- r1 := 18
sub r1, r0      -- r0 := 54
st r0, 0x1230   -- 54 à 0x1230
```

Question 2 On s'intéresse à l'instruction présente à l'adresse 0X4006. Quels sont les chemins de données utilisés dans la PO durant l'exécution ? Quels sont les états empruntés par la PC pour traiter cette instruction ? En combien de cycles la traite t-on ? Pour résumer cette analyse, dessiner un chronogramme faisant apparaître l'horloge et la valeur des signaux importants de la PC et de la PO durant le traitement de cette instruction.

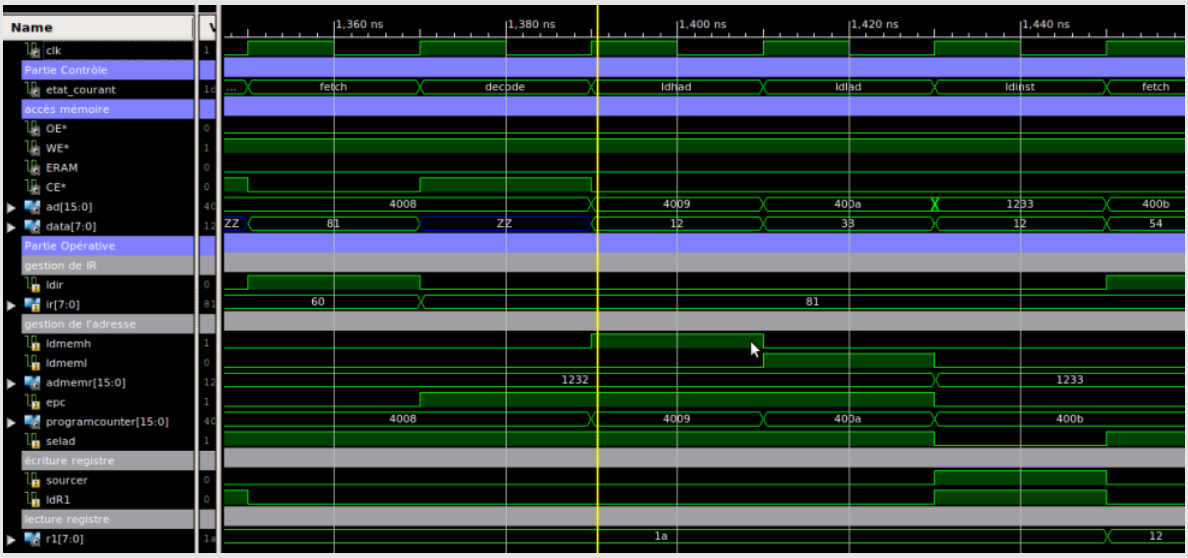
add r1 r0

On ouvre les chemins de données sortant des registres `R1` et `R0` via `selA` avec $selA(1:0) = 1$ et `selB` avec $selB(1:0) = 0$. Les valeurs passent par l'UAL, avec sur l'entrée $op(2:0)$ le code de l'addition. En sortie de l'UAL, on récupère le résultat de l'addition, qui revient en entrée des 4 registres de données via `selR`. On écrit dans le registre `R0` en mettant le $ldR0$ à 1.



Pour aller plus loin...
Question 3 Même exercice avec l'instruction à l'adresse 0x4008

idem avec le ld



TD 11
Modification d'un processeur

L'objectif de ce TD est de montrer les conséquences de l'ajout d'instructions sur la micro-architecture d'un processeur et sa machine d'état. Le processeur utilisé est le même que dans les séances précédentes.

Méthodologie
Ex. 1 : Chargement immédiat : intérêt et modification sur le processeur

Pour simplifier l'accès aux constantes dans un programme, on propose d'ajouter une instruction qui permet de charger un registre (8 bits) avec une constante (8 bits également). Cette nouvelle instruction sera notée `li IMM, rd`. Généralement nommée *load immediat* en anglais, elle effectue l'affectation suivante : $rd \leftarrow IMM$.

Question 1 Indiquez où doit se trouver la constante à charger dans le registre et proposez un encodage de l'instruction `li` qui s'insère simplement dans l'encodage existant.

La constante doit se trouver en mémoire dans l'octet suivant l'octet du code opération. Un encodage possible du premier octet de l'instruction est le suivant :

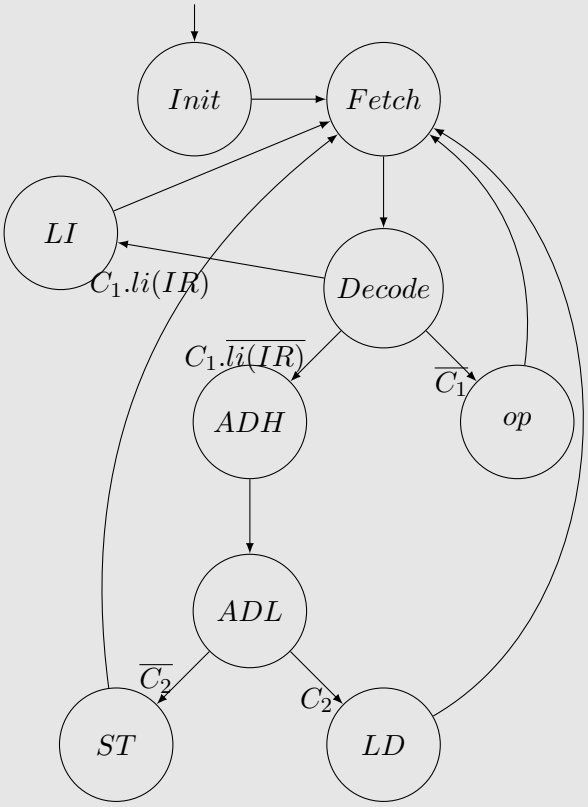
Instruction	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
`li IMM, rd`	1	0	0	0	1	0	rd_1	rd_0

Question 2 Est-il nécessaire de modifier la partie opérative ? Si oui, effectuez les modifications. Si non, justifier votre réponse.

Inutile de modifier la PO car il existe déjà le mux selR pour amener la valeur qui est sur DATAIN vers les registres généraux.

Question 3 Etendez la machine d'état de la PC obtenue au TD 9 de manière à supporter cette nouvelle instruction.

li(IR) est la fonction booléenne qui indique si un chargement est immédiat. Dans notre choix de codage, $li((IR) = IR_3$.



où $C_1 = AccesMem(IR)$, $C_2 = AccesEcr(IR)$.

Dans l'état *li*, on effectue l'opération RTL suivante : $(rd, PC) \leftarrow (MEM(PC), PC + 1)$ On doit donc avoir : $CE* = 0, WE* = 1, ldPC = 1, selAD = 1, selR = 1, ldRx = f(IR_1, IR_0), selA, selB, op$ en ϕ-booléen.

Ex. 2 : Branchement : intérêt et modification sur le processeur

Le programme ci-dessous calcule le produit de la constante `X` (située à l'adresse 0x2000) et la constante `Y` (0x2001) et stocke le résultat à l'adresse 0x1230.

Pour pouvoir exécuter ce programme sur notre processeur, on a besoin de rajouter des instructions de branchement :

- `jmp @mem` fait « sauter » l'exécution à l'adresse `@mem` (*i.e.* `PC := @mem`) ;
- `jz @mem` provoque un branchement à l'adresse `@mem` ssi $Z = 1$ (où Z est l'indicateur valant 1 ssi le résultat de la dernière opération était nul).

```
Cpt := Y;
Res := 0;
while Cpt /= 0 loop
  Res := Res + X;
  Cpt := Cpt - 1;
end loop;
```

```
0x4000: ld 0x2000, r0
0x4003: ld 0x2001, r1
0x4006: li 1, r2
0x4008: xor r3, r3
0x4009: or r1, r1
0x400A: jz 0x____
0x400D: add r0, r3
0x400E: sub r2, r1
0x400F: jmp 0x____
0x4012: st r3, 0x1230
```

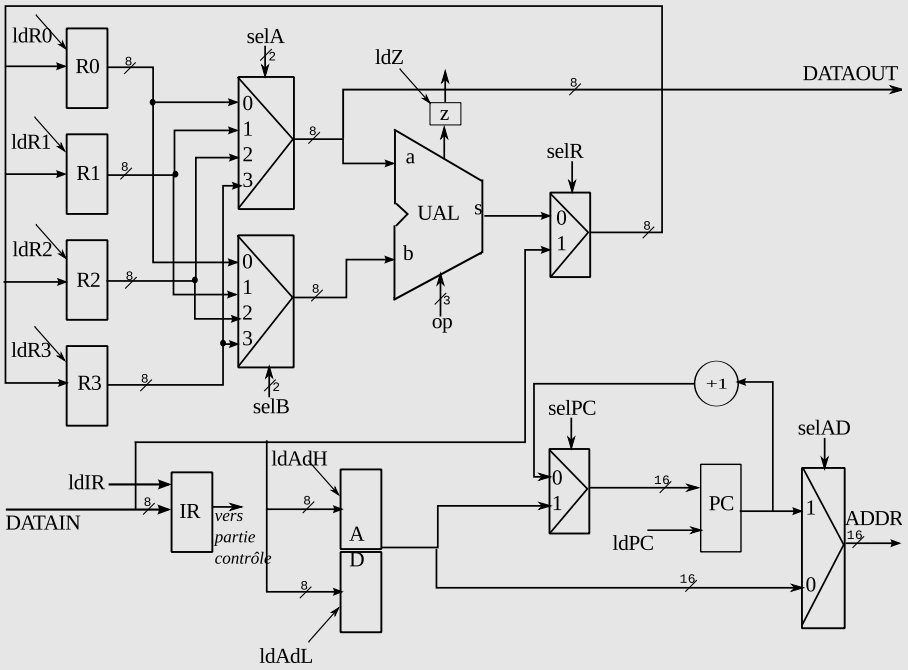
Question 1 Associer les lignes de l'algorithme écrit en pseudo-code (ci-dessus à gauche) à celles du programme écrit en langage machine (ci-dessus à droite), justifier les adresses des instructions et

compléter les champs manquants après les instructions de branchement.

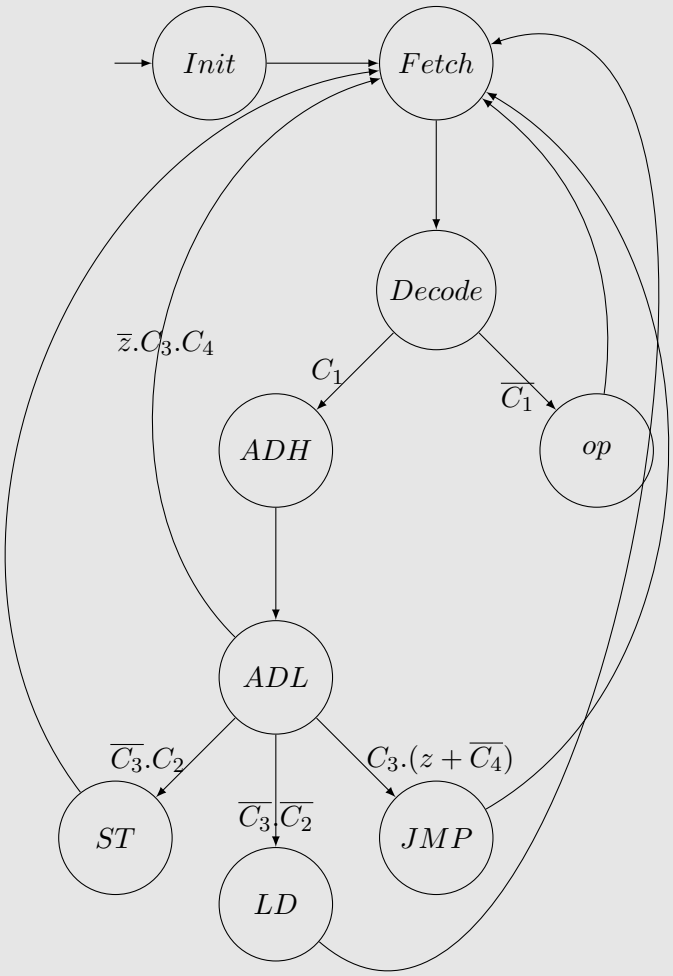
```
0x4000: ld 0x2000, r0 --r0=X
0x4003: ld 0x2001, r1 --r1=Cpt=Y
0x4006: li 1, r2 --r2=1
0x4008: xor r3, r3    --r3=res=0
0x4009: or r1, r1     --test sur r1 pour mise à jour du flag
0x400A: jz 0x4012     -- saut à la fin
0x400D: add r0, r3    -- res+=X
0x400E: sub r2, r1    -- Cpt--
0x400F: jmp 0x4009    -- saut vers le test
0x4012: st r3, 0x1230 -- res=r3
```

Question 2 Que faut-il ajouter à la partie opérative de notre processeur pour supporter ces nouvelles instructions ?

On ajoute 1 multiplexeur devant PC, controlé par selPC, et un registre 1-bit pour le flag Z controlé par ldZ.



Question 3 Modifier la PC en conséquence et proposer un codage pour ces instructions.



où $C_1 = AccesMem(IR)$, $C_2 = AccesEcr(IR)$, $C_3 = saut(IR)$ et $C_4 = sautCond(IR)$. Les deux dernières étant des fonctions exprimant si le branchement est conditionné ou non.

On peut placer saut(IR) sur un des bits inutilisés de IR (entre 2 et 5). Dès lors que l'on sait qu'on a affaire à un saut, les champs 0 et 1 n'ont plus de signification. On peut donc placer sautCond entre 0 et 5 en évitant l'emplacement de saut. Par exemple, saut(IR)=IR(2) et sautCond(IR)=IR(0) :

Branchement inconditionnel : $PC := AD$								
`jmp AD`	1	0	0	0	0	1	0	0
	ADH							
	ADL							
Branchements conditionnels : si Z=1 alors $PC = AD$								
`jz AD`	1	0	0	0	0	1	0	1
	ADH							
	ADL							

Dans la machine d'état, on rajoute les opérations registre à registre suivantes :

— $PC \leftarrow AD$ dans l'état JMP

— $Z \leftarrow z_{UAL}$ dans l'état op

Pour traduire ces opérations RTL, il faut ajouter les signaux de contrôle ldZ et selPC et l'état JMP à notre table de vérité. Attention aux affectations de PC dans les états fetch, ADH et ADL

Signaux	Fetch	Decode	op	ADH	ADL	ST	LD	Init	**JMP**
selPC	**0**	**x**	**x**	**0**	**0**	**x**	**x**	**x**	**1**
ldZ	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
CE^*									**1**
WE^*									**x**
$ldPC$									**1**
$selAD$									**x**
$ldiAdH$									**0**
$ldAdL$									**0**
$ldIR$									**0**
$selR$									**x**
$ldR0$									**0**
$ldR1$									**0**
$ldR2$									**0**
$ldR3$									**0**
$selA$									**xx**
$selB$									**xx**
op									**xxx**