Université Tunis El-Manar	**Année Universitaire** : 2016-2017
Faculté des Sciences de Tunis	**Module :** Synthèse en VHDL de systèmes embarqués
Section : IF4 Option ISEM	

TD N° 3

Exercice 1.

Dans cet exercice, nous proposons de réaliser un module matériel permettant de vendre des films pour 5DT. L'usager a le droit de mettre des pièces de 1DT ou de 2DT et dès que le montant arrive a 5DT ou plus, il y a un film qui sort. Il est aussi possible que l'usager mette 6DT et dans ce cas, la machine donne un film et remet aussi la monnaie. Si la personne mettait de l'argent pendant que le film sort, l'argent sera perdu.

On peut soit insérer 1DT, 2DT ou rien, donc on va avoir une entrée pour chaque type de monnaie. À la sortie, on peut soit donner un film ou soit donner un film et de la monnaie.

Les entrées et sorties de ce module matériel sont décrites comme suit :

```
ENTITY machine IS
PORT (
clk,reset : IN STD_LOGIC;
un_dinar : IN STD_LOGIC;
deux_dinars : IN STD_LOGIC;
film : OUT STD_LOGIC;
monnaie : OUT STD_LOGIC
);
END ENTITY ;
```

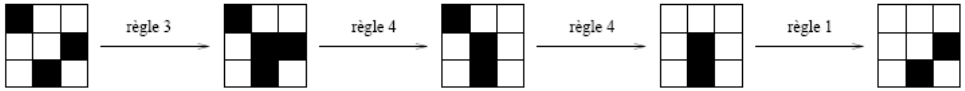
Donner la machine à états, puis le code VHDL du module matériel décrit.

Exercice 2.

Le célèbre « jeu de la vie » de Conway est une simulation impliquant un ensemble de « cellules » sur une grille cartésienne. À partir d'une configuration initiale, l'état de chaque cellule évolue en fonction de l'état des cellules voisines, ce qui en fait un automate cellulaire. L'état de toutes les cellules change en même temps, ce qui correspond à une génération. Une cellule est soit « vivante » (état 1) ou « morte » (état 0). L'automate est entièrement défini par les règles simples suivantes qui sont évaluées pour passer d'une génération à l'autre:

1. Si une cellule vivante a moins de 2 voisins, elle meurt d'isolement.
2. Si une cellule vivante a plus de 3 voisins, elle meurt d'étouffement.
3. Si une cellule morte a exactement 3 voisins, elle « nait », i.e. elle devient vivante.
4. Si une cellule vivante a 2 ou 3 voisins, elle conserve son état.

On considère qu'une cellule possède 8 voisins sur grille cartésienne. L'illustration suivante montre l'évolution d'une cellule (cellule au centre) en fonction de ses 8 voisins au fil de 4 générations. On voit que l'état des voisins change également, en fonction de leurs 8 voisins respectifs.



On s'intéresse dans ce qui suit à la conception d'une machine à états qui réalise une cellule dans le jeu de la vie. Assumez qu'il y a une génération par coup d'horloge et utilisez l'entité suivante:

```
entity cellule is port(
      nb_voisins: in std_logic_vector(3 downto 0);
      etatcellule: out std_logic;
      clk, reset: in std_logic));
end entity;
```

On assume également que l'entrée `nb_voisins` donne le nombre de voisins vivants.

1) Donner la machine à états décrivant ce circuit, puis traduire la machine à état en une description VHDL.
2) Donner le chronogramme des variations de tous les signaux y compris le signal indiquant l'état courant pour l'exemple donné.

Exercice 3.

On se propose de réaliser un module permettant de convertir une valeur binaire au code BCD (Binary Coded Decimal; Décimal codé binaire).
Nous rappelons que le codage BCD permet de coder chaque chiffre d'un nombre décimal sur 4 bits. À titre d'exemple, la valeur décimale 174 est représentée en binaire par la valeur 10101110, alors qu'elle est représentée en codage BCD par le nombre binaire 0001 0111 0100.
Le circuit à réaliser considère une valeur binaire codée sur 8 bits pour produire sur ses sorties le code BCD de la valeur considérée. Ce circuit est illustré par la figure 1. Il possède l'entrée d'initialisation asynchrone *reset*. L'entrée synchrone *start* qui permet de lancer la conversion de la valeur spécifiée à travers l'entrée *val*. Les sorties *U*, *D* et *C* devraient renvoyer respectivement le chiffre des unités, des dizaines et des centaines.

Figure 1 : Module de conversion binaire vers BCD

L'algorithme de conversion d'une valeur binaire au code BCD est donné comme suit :
Etape 1 : Si la valeur de la colonne U est supérieure ou égale à 5, ajouter 3 à cette colonne.
Etape 2 : Si la valeur de la colonne D est supérieure ou égale à 5, ajouter 3 à cette colonne.
Etape 3 : Décaler tous les bits d'une position à gauche.
Etape 4 : Si 8 opérations de décalage ont été effectuées, alors Terminer. Sinon Aller à l'étape 1.

Les étapes de l'exécution de cet algorithme pour la valeur 174 sont illustrées comme suit :

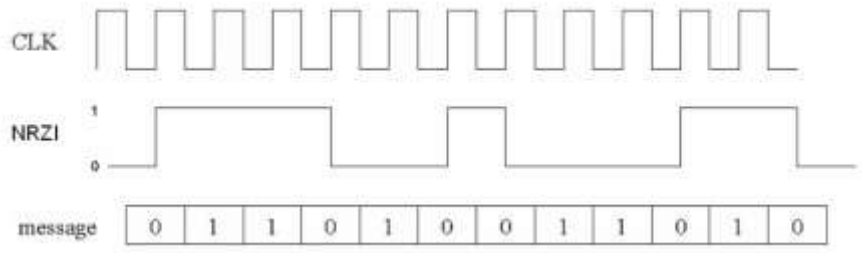
C	D	U	val	Opération
			10101110	
		1	0101110	Décalage
		10	101110	Décalage
		101	01110	Décalage
		1000	01110	Ajouter 3 à la colonne U
	1	0000	1110	Décalage
	10	0001	110	Décalage
	100	0011	10	Décalage
	1000	0111	0	Décalage
	1011	1010	0	Ajouter 3 à la colonne D et U
1	0111	0100		Décalage

1) Donner la machine à états décrivant le comportement du séquenceur pour cette réalisation. Indiquer le nombre de bistables D à utiliser par la solution proposée
2) Traduire la machine état donné en VHDL.

Exercice 4.

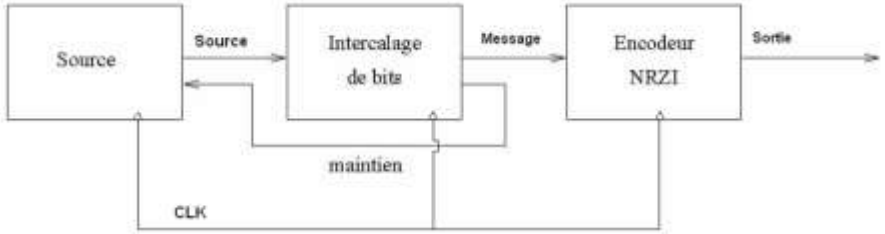
On utilise en communications numériques sur ligne câblée (paire torsadée, câble coaxial) divers "codes de ligne" pour communiquer en bande de base, c'est à dire que le signal est transmis tel quel sur la ligne. Ces codes constituent tout simplement une façon de représenter les bits à transmettre.

Le code en ligne que nous allons étudier est la code "NRZI" (Non Return to Zero Inverted) qui est utilisé dans le standard USB (Universal Serial Bus). Dans ce code, seuls les '0' génèrent une transition. Ainsi, pour transmettre le message : « 0 1 1 0 1 0 0 1 1 0 1 0 », on a la représentation suivante :



Cet encodeur possède un désavantage majeur : si une trop longue séquence de bits '1' successifs est transmise, il n'y a aucune transition sur la ligne. Ceci rend la synchronisation entre le transmetteur et le récepteur impossible. Pour remédier à cela, le standard USB prévoit une opération d'intercalage de bits ("bit stuffing"). En effet, dès qu'une chaîne de six '1' est transmise, on intercale un '0' pour forcer une transition. Il est facile d'enlever ces '0' au récepteur pour recomposer le message original.

Une portion de la chaîne de communication correspondante est illustrée sur la figure suivante :



La source : Elle génère les bits du message à transmettre sur le câble USB.
Intercalage de bit : Il doit être capable de détecter une séquence de 6 '1' successifs et de transmettre un message avec les éventuels « bits de stuffing ». Si un « bit de stuffing » intervient, la ligne "maintien" est placée à '1' pour un coup d'horloge afin de signaler à la source qu'il faut attendre. Ce temps correspond au temps de mettre le bit supplémentaire dans le message.
L'encodeur NRZI : Bloc réalisant la fonction NRZI.

1) Donner le graphe détaillé d'une machine d'état réalisant seulement le bloc de l'encodeur NRZI.
Note : On prend arbitrairement un état initial associé à la valeur de Sortie=0 pour commencer la machine à états.

2) Donner le code VHDL de l'encodeur NRZI avec une architecture codée en description comportementale.

3) Réaliser la description VHDL du module « intercalage de bits ».