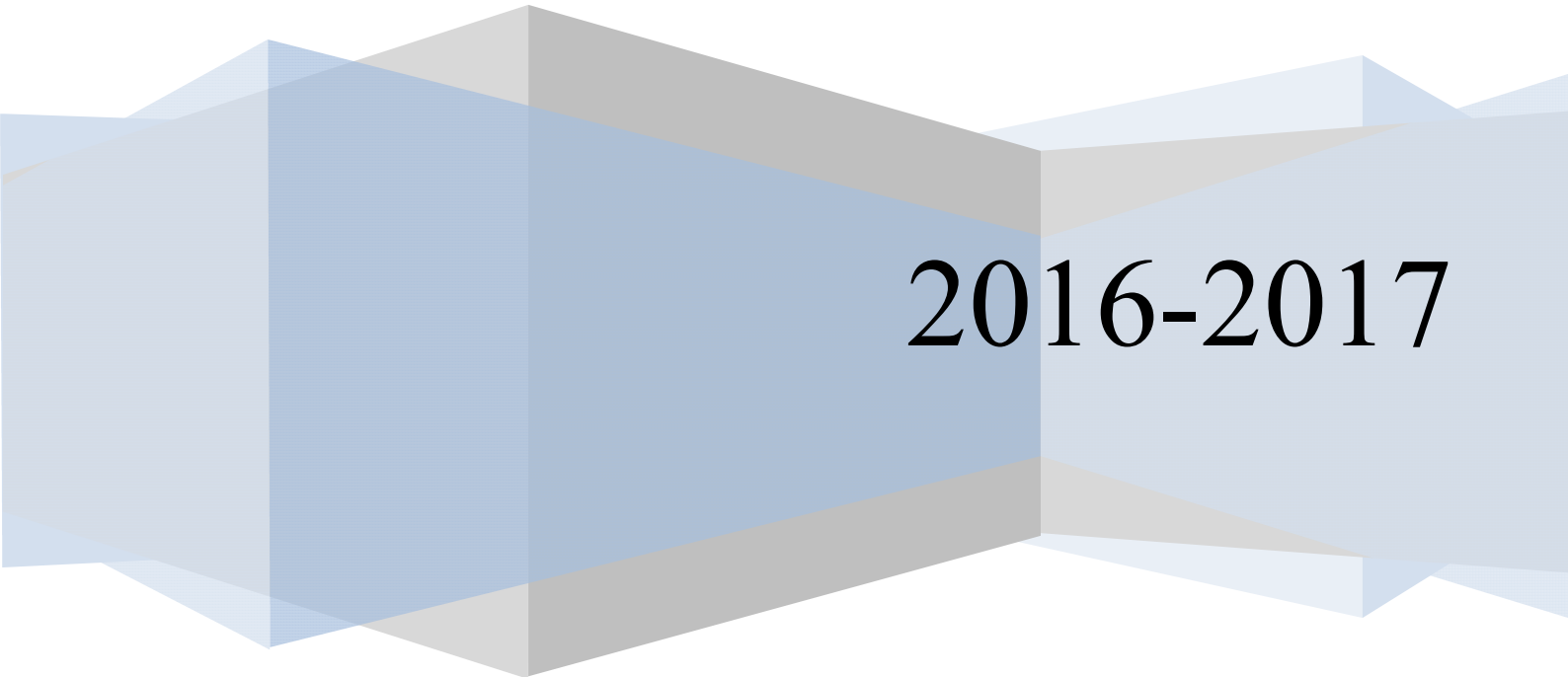


TP VHDL

Initiation au langage et à la mise en œuvre de
fonctions logiques sur FPGA

F. Bony – J. Perchoux – E. Peuch – A. Quotb

The background of the bottom half of the page features a complex, abstract geometric design. It consists of several overlapping, semi-transparent polygons in shades of light blue and grey, creating a sense of depth and modernity.

2016-2017

Préambule

Evaluation

Vous avez 5 séances de 3h30 pour mener à bien votre TP VHDL.

Pendant les séances TP vous **devez faire valider**, au fur et à mesure, votre travail: pour cela nous vous avons indiqué dans ce document les étapes à valider auprès d'un enseignant (partie théorique, code et mise en œuvre sur la carte). Cette partie constituera l'évaluation de vos compétences en VHDL.

Un test **individuel de 15 min** sera programmé lors de la dernière séance de TP pour évaluer vos connaissances.

Bonnes pratiques

L'outil de développement utilisé est le logiciel ISE:



Utilisez le document **"Prise en main du logiciel ISE"** et commencez par créer votre projet.

Projet = ensemble des composants développés pour réaliser votre horloge.

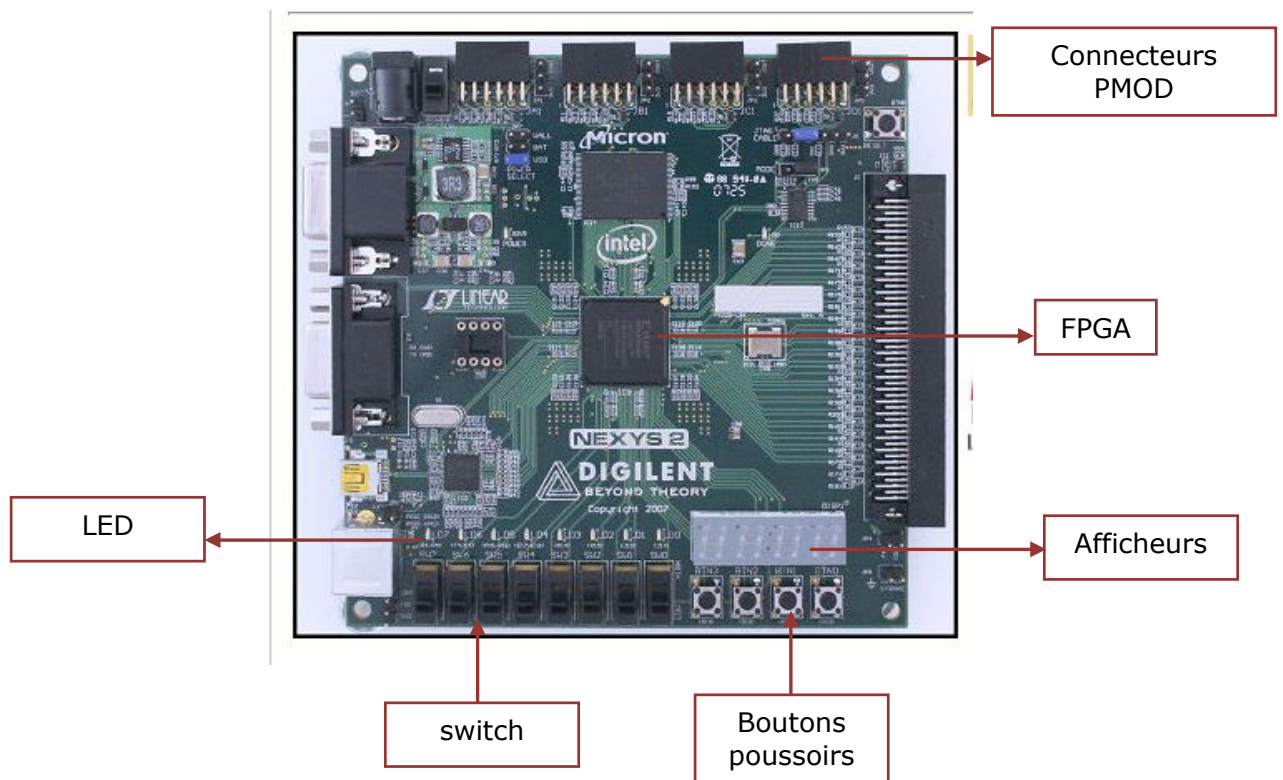
Il faut créer ce projet sur **l'unité de disque local soit D:** (ne pas travailler sur le serveur).



N'oubliez donc pas de sauvegarder votre projet sur une clé et sur le serveur à la fin de chaque séance de TP!!!!

Matériel utilisé

Pour réaliser votre TP vous disposez d'une carte de développement Nexys2:





Avant de continuer, lire le document "qu'est-ce qu'un FPGA?" disponible sur Moodle UE numérique.



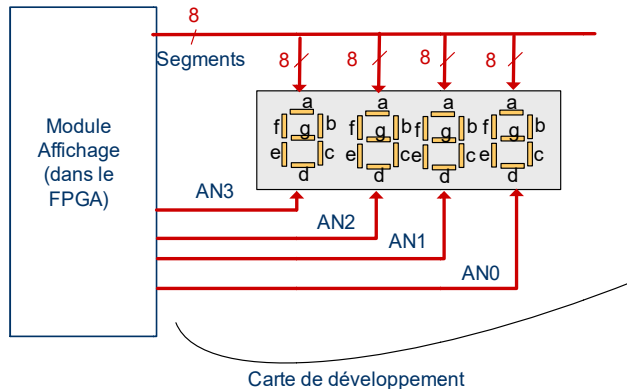
Documents à votre disposition sur Moodle:

- Prise en main du logiciel ISE
- Documentation technique de la carte de développement Nexys 2
- Utilisation de la carte Nexys2: configuration et chargement d'un fichier .bit dans le FPGA

Conception du module affichage

⇒ L'affichage multiplexé

Sur la carte Nexys 2, les afficheurs sont multiplexés: ils sont câblés en parallèle.



Notez que, sur la carte de développement, l'afficheur N°0 correspondant au signal de sélection AN0 est **l'afficheur de poids faible** (donc à droite sur la carte).

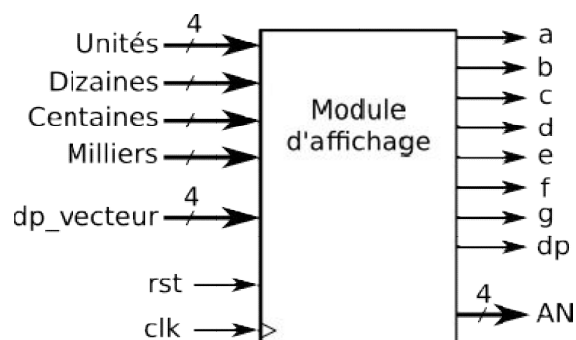
C'est-à-dire que les 4 afficheurs partagent le même décodeur BCD -> 7 segments. Il est donc nécessaire d'effectuer un multiplexage temporel afin d'en activer un seul à la fois. Le multiplexage doit se faire à une fréquence supérieure à 100 Hz pour que l'affichage soit permanent pour notre œil (Persistance rétinienne) mais inférieur à 5 kHz à cause des temps de montées dans les LED d'affichage. On utilisera ici une fréquence de rafraîchissement de 1 kHz (**H_1kHz**).

Les signaux **AN3**, **AN2**, **AN1**, **AN0** permettent de sélectionner ou d'inhiber respectivement l'afficheur 3, 2, 1 ou 0.

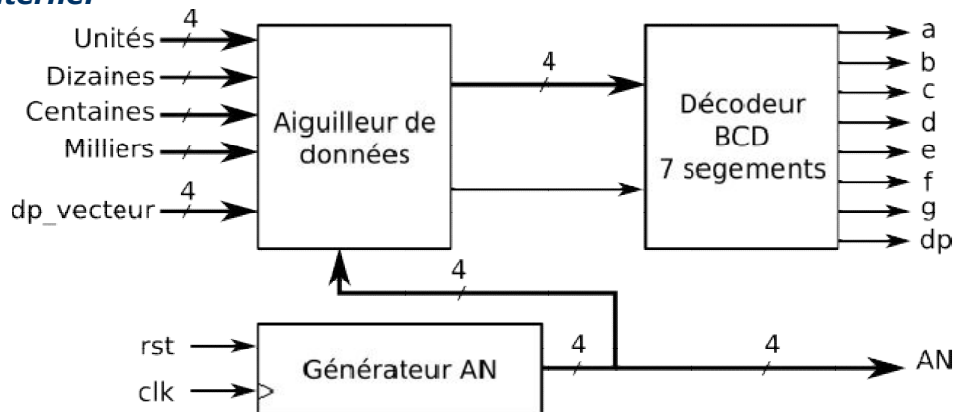
ANx	
0	Afficheur N°x sélectionné
1	Afficheur N°x inhibé

On peut maintenant en déduire la vue interne du module d'affichage.

⇒ Vue externe (rappel):

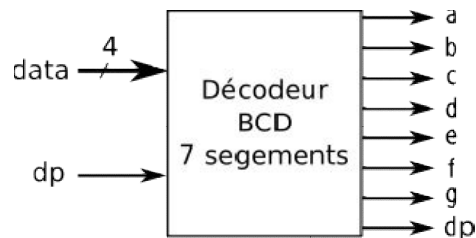


⇒ **Vue interne:**

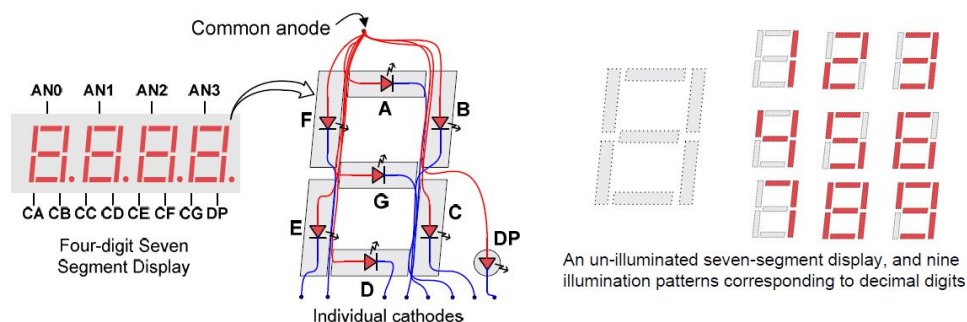


Nous allons maintenant réaliser chacun de ces composants.

I. Le décodeur 7 segments



Les afficheurs 7 segments sont à anode commune qui est reliée à la tension d'alimentation Vcc.



"Extrait de la documentation technique Nexys 2"

Les cathodes de chaque segment (a, b, c, ...Dp) sont reliées à une patte du FPGA. **Pour allumer un segment** il faudra appliquer **un 0 logique** à la patte correspondante du FPGA.

1. Travail préparatoire : Ecrivez la table de vérité du décodeur pour les sorties a à g.

Data	a	b	c	d	e	f	g
0 0 0 0							
0 0 0 1							
.							
.							
.							
1 1 1 1							

2. Réalisation du décodeur sans la gestion du point

- Créer un fichier module VHDL décrivant le comportement du décodeur.
- Créer un fichier de test et simuler votre module avec ISIM

3. Implémentation dans la carte Nexys2: phase de test

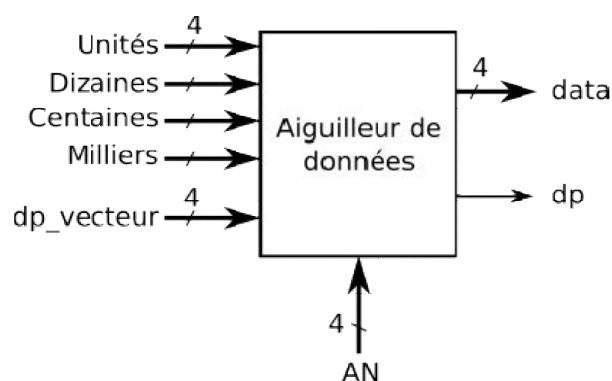
Utilisez le document "prise en main du logiciel ISE" pour assigner les pattes de votre composant aux broches d'entrée/sortie du FPGA (page 5).
Les entrées AN du composant seront reliées aux interrupteurs.

Implémentez votre composant dans le FPGA et testez son fonctionnement.



Validation par un enseignant

II. L'aiguilleur de données



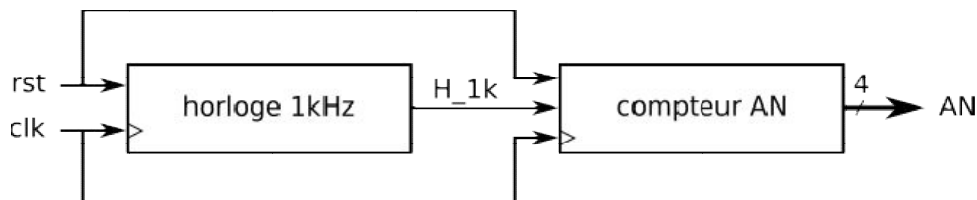
L'aiguilleur de données doit envoyer les données sur le bon afficheur.
Les données sont donc aiguillées en fonction des différentes combinaisons des **AN** = 1110 ou 1101 ou 1011 ou 0111.
Pour véhiculer les données, on utilise un multiplexeur 4 vers 1.

- Développer le module VHDL effectuant cette fonctionnalité. Simuler avec ISIM.

III. Le générateur des signaux AN



- Le générateur des signaux AN est un générateur de séquence. C'est donc un système séquentiel synchrone qui, en fonctionnement normal, change la valeur de ses sorties sur front d'un signal d'horloge. Le rafraichissement des afficheurs se fait à la fréquence de 1 kHz. Il faut donc réaliser une division de fréquence



L'entrée **rst** est asynchrone. La séquence à réaliser est la suivante:

- Réaliser le diviseur de fréquence à 1Khz. La durée de l'état haut de ce signal d'un seule période d'horloge.
- Réaliser le générateur des **AN**



Avant l'implémentation dans la carte, simulez le bon fonctionnement de votre générateur des AN

- Réalisez votre générateur de signaux AN et implémentez-le dans la carte pour tester son fonctionnement en visualisant les signaux **AN** et **H_1kHz** sur l'oscilloscope. Pour cela, reliez ces 5 signaux aux broches des connecteurs PMOD (Cf. doc technique de la carte Nexys 2 page 15). Puis utilisez la nappe de l'analyseur logique de l'oscilloscope.
Alternantive : vous pouvez également diminuer la fréquence du signal H_1kHz afin d'observer directement sur les LED de la carte Nexys2



Validation par un enseignant

IV. Mise en oeuvre du module affichage

Assemblez vos composants dans un module VHDL par une description structurelle afin de réaliser votre module d'affichage.

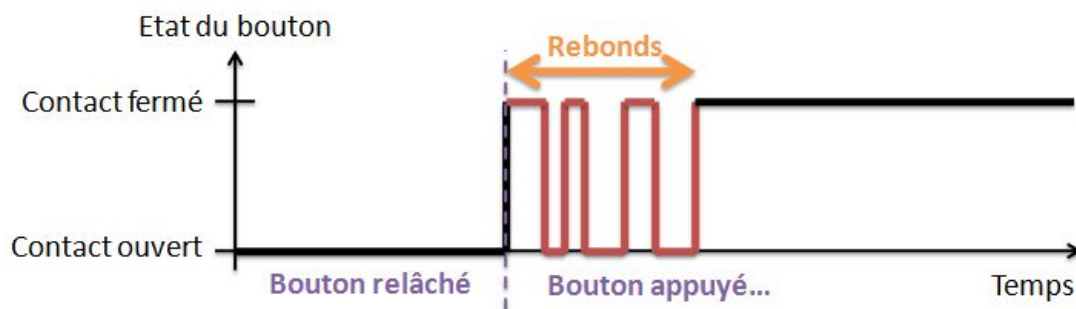
Implémentez votre composant dans la carte et testez son fonctionnement.



Validation par un enseignant

Conception d'un système anti-rebonds

Un interrupteur électromécanique subit un rebond mécanique qui altère le contact électrique avant sa stabilisation soit au niveau haut (interrupteur fermé), soit niveau bas (interrupteur ouvert). Heureusement, la constante de temps du système électronique (fréquence horloge) est bien plus rapide que celle du système mécanique.

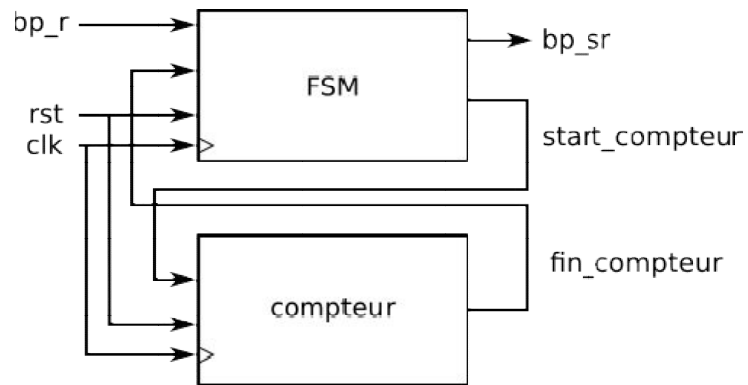


Il est nécessaire de filtrer numériquement les rebonds. Pour cela nous allons concevoir une machine à états qui va éliminer les fronts parasites. Le principe est le suivant : lorsque le système détecte un front montant (ou descendant) sur le signal d'entrée issu de l'interrupteur **Bp_r**, il force sa sortie **Bp_s_r** au niveau haut (bas) pendant une période de 2 ms quel que soit l'état du signal d'entrée. Après cette temporisation, le signal de sortie **Bp_s_r** reste dans le même état jusqu'à ce qu'un nouveau front soit détecté sur le signal de interrupteur **Bp_r**.

⇒ **Vue externe :**



- Modéliser le fonctionnement du système par une machine à état de type Moore. La temporisation sera réalisée par l'utilisation d'un compteur. La machine commandera ce compteur (cf. vue interne).

⇒ Vue interne:

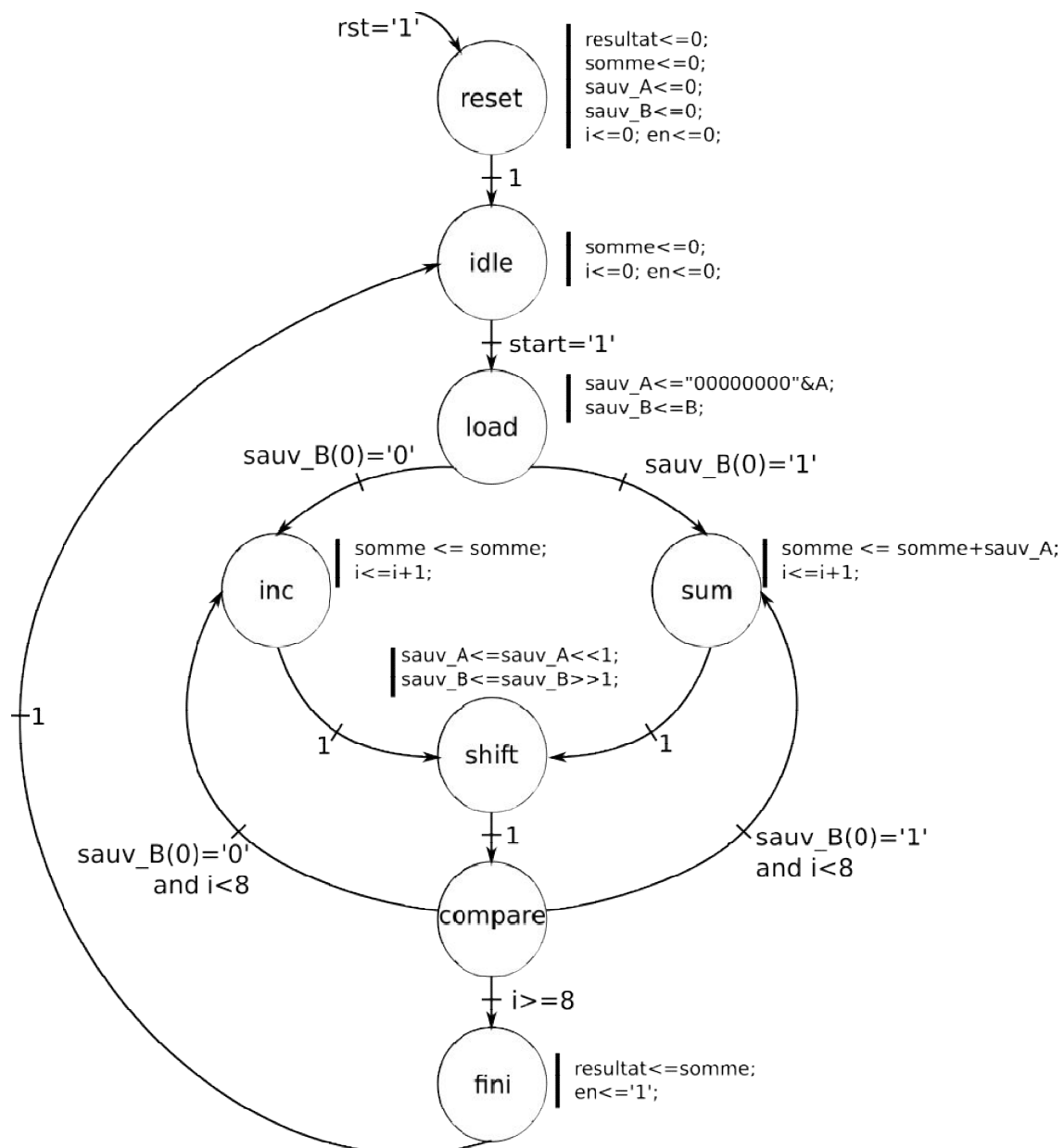
- Réaliser cette machine avec 3 *process*. Le compteur pourra être un 4^{ième} *process* ou un autre module VHDL.
- Simuler votre système anti-rebond.
- Pour le tester, développer un environnement VHDL de test utilisant deux compteurs et les afficheurs 7 segments de la carte affichant le nombre de rebonds observés sur les signaux **Bp_r** et **Bp_s_r**.

➤  **Validation par un enseignant**

Conception d'un multiplieur câblé

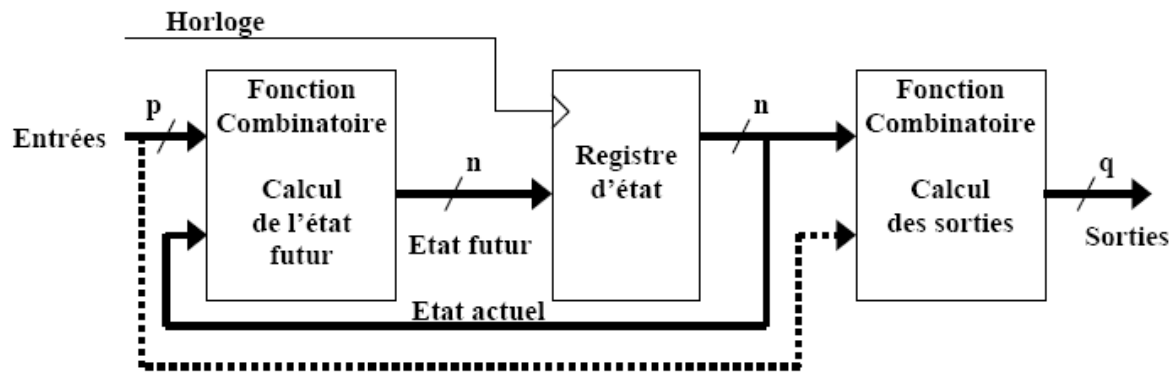
On se propose de réaliser un opérateur arithmétique réalisant la multiplication entre deux vecteurs **A** et **B** de 8 bits.

Une machine à états modélise ce multiplieur.



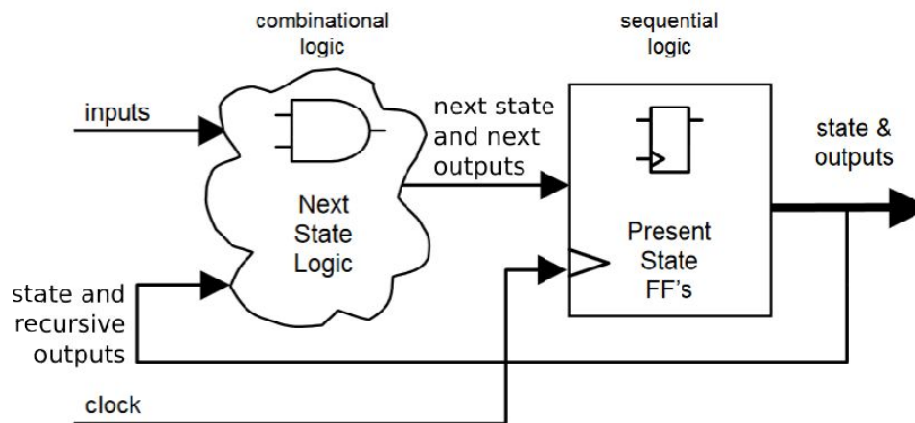
Dans cette machine, des signaux internes subissent des actions récurrentes donc séquentielle (ex : **i**<=**i**+1; **Somme**<=**Somme**+**Sauv_A**;...).

Sans modification de la structure générale d'une machine à états (ci-dessous pour rappel), il n'est pas possible de réaliser ses actions récurrentes.



En effet, le bloc combinatoire calculant les sorties (ou les signaux internes) n'est pas séquentiel. Une action de type $i \leq i+1$ entraînerait une boucle combinatoire et donc l'instabilité du système.

La structure modifiée de la machine d'état est donnée par le schéma ci-dessous :



Deux solutions dans le codage en VHDL sont alors offertes :

- soit un module avec un seul *process* séquentiel dans lequel une structure combinatoire à base d'un *CASE* calcule le nouvel état et les sorties "modifiées" sur le nouvel état calculé.
- Soit une structure à 3 *process* (F,M,G) dont le *process* des sorties est synchrone (bloc G synchrone).

La première des solutions, si elle semble la plus simple pose plusieurs problèmes liés en particulier à la complexité du *process* unique et à la difficulté de retrouver clairement le diagramme d'état à partir du code qui entraîne des problèmes de pérennité du code. Nous optons donc pour la deuxième solution.

- Une fois votre module VHDL réalisé, écrire un fichier de test et réaliser la simulation.
- Implémentez le multiplieur sur la carte Nexys2. Le résultat pourra dans un premier temps être affiché sur les LEDs, mais un affichage sur les afficheurs 7 segments serait plus élégant.



Validation par un enseignant

Réalisation d'une mémoire RAM simple port

Réaliser des modules génériques, plutôt qu'instancier des ressources numériques internes propres aux composants numériques cibles (FPGA ou bibliothèques de composants d'un ASIC) permet leur réutilisation dans différents projets.

En VHDL, le paramétrage se fait par l'utilisation de générique défini dans l'entité. (mot clé **GENERIC**). Ce sont généralement des entiers ou des naturels servant à dimensionner la taille des objets (les vecteurs par exemple).

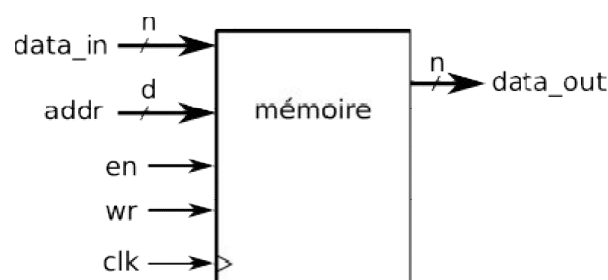
```
entity registre is
  generic (NbBits : INTEGER := 8);
  port (D : in std_logic_vector (NbBits-1 downto 0);
        Load : in std_logic ;
        Enable : in std_logic ;
        Clock : in std_logic ;
        Q : out std_logic_vector (NbBits-1 downto 0)
        );
end registre ;
```

Pour illustrer ce concept, nous souhaitons réaliser une mémoire dont la taille est fonction de 2 paramètres :

- n : naturel définissant la taille du bus de donnée.
- d : naturel définissant la taille du bus d'adresse.

Une mémoire est un tableau de vecteur (ARRAY) qui devra donc être déclaré dans l'architecture du module VHDL comme suit (X et Y étant les dimensions de la mémoire):

```
type memoire is array(X downto 0) of std_logic_vector(Y downto 0);
signal memo: memoire;
```



La liste des signaux mis en œuvre est décrite ci-dessous

- **En** : signal qui autorise l'écriture de la donnée **Data_in** à l'adresse **Addr**, et la lecture sur **Data_out** de la donnée située à l'adresse **Addr**.
- **Wr** : Signal d'écriture
- **Clk** : Horloge, toute opération est synchrone.

On notera que bien qu'étant un système séquentiel, une mémoire n'a pas de signal de reset !

Remarque : une conversion de type *std_logic_vector* à *integer* est nécessaire. Pour réaliser cette conversion :

https://www.doulos.com/knowhow/vhdl_designers_guide/numeric_std/

- Ecrire un module VHDL de cette mémoire générique.
- Vérifier après synthèse que vous avez effectivement réalisé une RAM
- Simuler votre composant.



Validation par un enseignant