Systèmes digitaux

Cours 9

VHDL

- Au dernier cours, on a vu le VHDL combinatoire (sans élément mémoire)
- Les commandes peuvent être à l'intérieur d'un PROCESS ou sans PROCESS
- Aujourd'hui, on parle du VHDL séquentiel (avec mémoire)
    - Les mémoires utilisées seront des flip flops

VHDL séquentiel

- Contrairement aux circuits combinatoires, il **faut** un PROCESS en circuit séquentiel
- Le PROCESS a une structure très bien définie
  - Il ne faut surtout pas déroger de la structure suivante:

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
            VOUS FAITES CE QUE VOUS VOULEZ ICI…
    END IF;
END PROCESS;
```

Rien ici

Pas d'autres conditions!

Rien ici

VHDL séquentiel

- Dans la liste de sensibilité, il n'y a seulement que l'horloge (clk)

```
PROCESS (clk)
```

- Il y a un IF qui identifie s'il y a un changement sur clk et s'il est maintenant égal à 1:
  - Donc, il veut savoir si c'est un front montant d'horloge
  - Il ne faut PAS changer cette ligne, ni ajouter/enlever des conditions

```
IF clk'EVENT AND clk = '1' THEN
```

VHDL séquentiel

- Il ne doit RIEN y avoir entre BEGIN du PROCESS et le premier IF
- Les deux dernières lignes du PROCESS doivent être END IF; et END PROCESS;

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
            VOUS FAITES CE QUE VOUS VOULEZ ICI…
    END IF;
END PROCESS;
```

VHDL séquentiel

- Chaque assignation dans un PROCESS séquentiel va créer une flip flop
    - L'ordre n'a aucune importance...

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
      q1 <= entree;
      q2 <= q1;
    END IF;
END PROCESS;
```

q1 et q2 deviennent des flip flops…

VHDL séquentiel

- Il est possible d'utiliser des IF à l'intérieur du IF principal:

```
PROCESS (clk)
BEGIN
      IF clk'EVENT AND clk = '1' THEN
            IF entree = '1' THEN
                  sortie<= a+b;
            END IF;
      END IF;
END PROCESS;
```

VHDL séquentiel

- En VHDL, quand on ne spécifie pas toutes les options, il garde sa valeur précédente
  - Même en PROCESS combinatoire (sans clk)
  - Alors, si on ne veut pas d'éléments mémoires, il faut spécifier toutes les options

```
PROCESS (entree)
BEGIN
       IF entree = "0000" THEN
              sortie <= '1';
       ELSIF entree = "0010" THEN
              sortie <= '0';
       END IF;
END PROCESS;
```

- Process combinatoire
- Énumération incomplète
- Élement mémoire!

VHDL séquentiel

- Il y a 2 solutions, selon notre intention:
  - Si on ne veut pas d'éléments mémoire, il faut spécifier toutes les autres options
  - Si on veut un élément mémoire, il faut utiliser un process séquentiel, donc avec PROCESS (clk)

```
PROCESS (entree)
BEGIN
        IF entree = "0000" THEN
                sortie <= '1';
        ELSE
                sortie <= '0';
        END IF;
END PROCESS;
```

```
PROCESS (clk)
BEGIN
        IF clk'EVENT AND clk='1' THEN
                IF entree = "0000" THEN
                        sortie <= '1';
                ELSIF entree ="0010" THEN
                        sortie <= '0';
                END IF;
        END IF;
END PROCESS;
```

Exemples de code VHDL

- Un compteur doit contenir des éléments mémoires:
  - À chaque coup d'horloge, son compte devient plus élevé que sa valeur précédente
  - Si s_compte est de 4 bits, ça irait de 0 à 15 et ça recommencerait
  - Si s_compte est de 8 bits, ça irait de 0 à 255 et ça recommencerait

```
PROCESS (clk)
BEGIN
        IF clk'EVENT and clk = '1' THEN
                s_compte <= s_compte + 1;
        END IF;
END PROCESS;
```

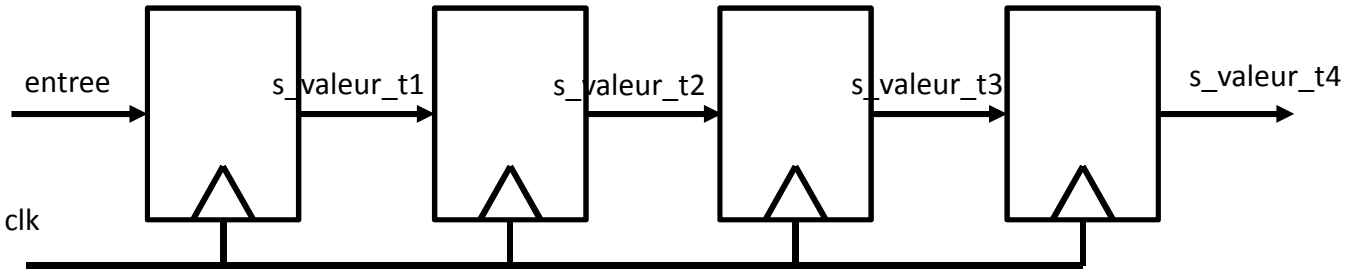
Exemples de code VHDL

- Comment faire pour que le compte arrête avant de se rendre au maximum?
    - Par exemple, avec 4 bits, comment faire un compteur qui va de 0 à 11 avant de recommencer?
- Comment faire pour que le compte aille de 0 à 11 et s'arrête à 11?
- Comment faire pour que le compte aille de 11 à 0 avant de recommencer à 11?

Exemples de code VHDL

- Pour faire un registre à décalage, il est possible de faire ceci:

```
PROCESS (clk)
BEGIN
        IF clk'EVENT and clk = '1' THEN
                s_valeur_t1 <= entree;
                s_valeur_t2 <= s_valeur_t1;
                s_valeur_t3 <= s_valeur_t2;
                s_valeur_t4 <= s_valeur_t3;
        END IF;
END PROCESS;
```



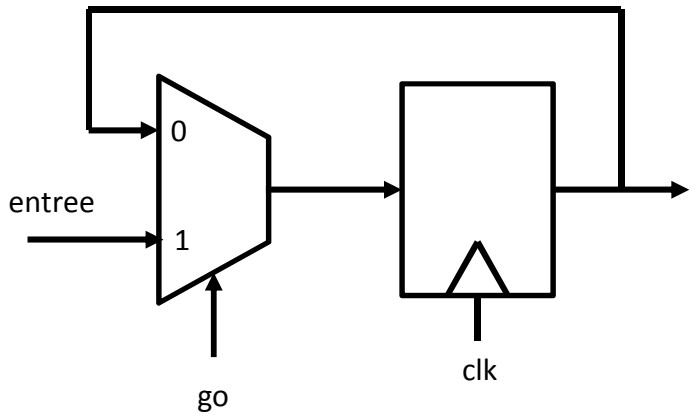
Exemples de code VHDL

- À partir de ce code, écrivez le code VHDL pour une moyenne glissante

```
PROCESS (clk)
BEGIN
        IF clk'EVENT and clk = '1' THEN
                s_valeur_t1 <= entree;
                s_valeur_t2 <= s_valeur_t1;
                s_valeur_t3 <= s_valeur_t2;
                s_valeur_t4 <= s_valeur_t3;
        END IF;
END PROCESS;
```

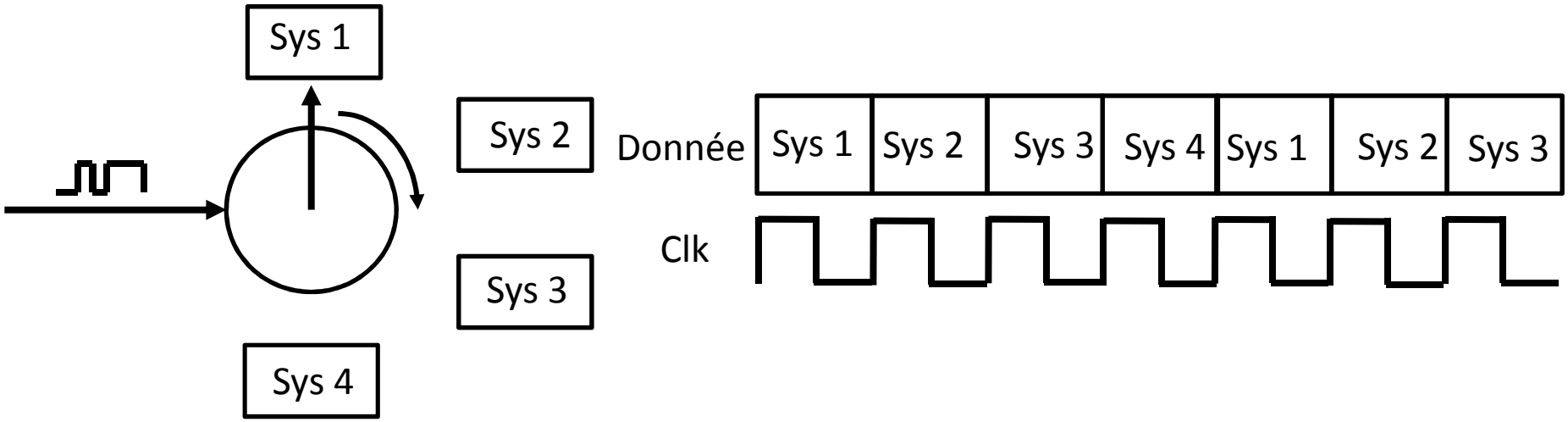
Exemples de code VHDL

- Comment faire un registre qui enregistre la valeur *entree* seulement quand *go*='1'?
  - Sinon, il garderait sa valeur précédente
  - Le circuit ci-dessous décrit le comportement du code souhaité



Exemples de code VHDL

- Imaginons un système qui reçoit des données et qui les distribue dans 4 systèmes différents
- Chaque sous-système reçoit une nouvelle donnée à chaque 4 bits que le système reçoit.



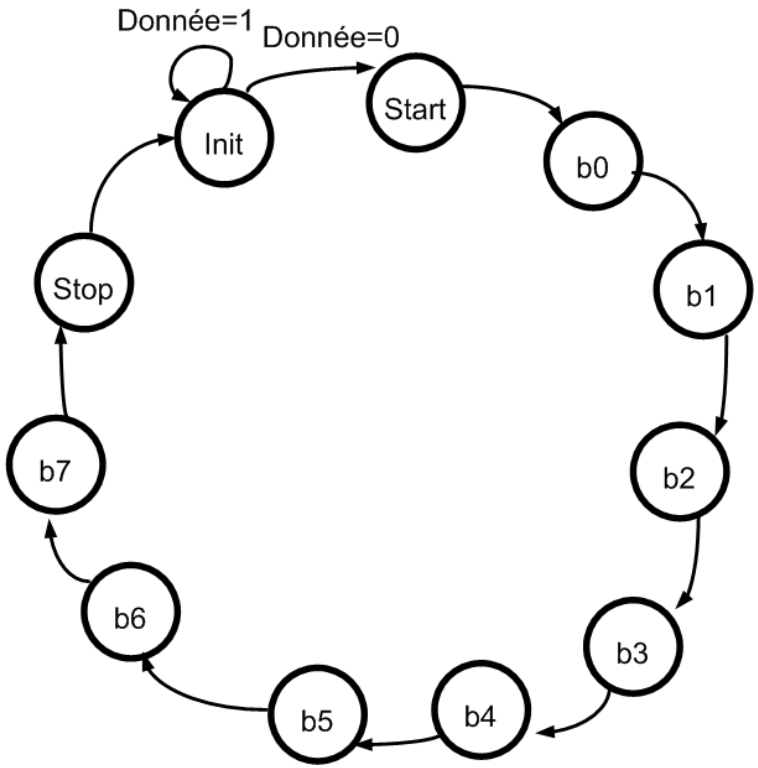
Concevez un système qui effectue ce travail...

Retour sur le projet

- Objectif: Contrôler la position d'un servomoteur à l'aide d'un ordinateur
    - Communication UART (série)
- FPGA doit recevoir la donnée et réagir différemment selon ce qui est envoyé
- Il y a 2 façons de procéder:
    - Enregistrer les données et comparer par la suite
    - Effectuer prendre une décision à mesure que chaque bit est reçu

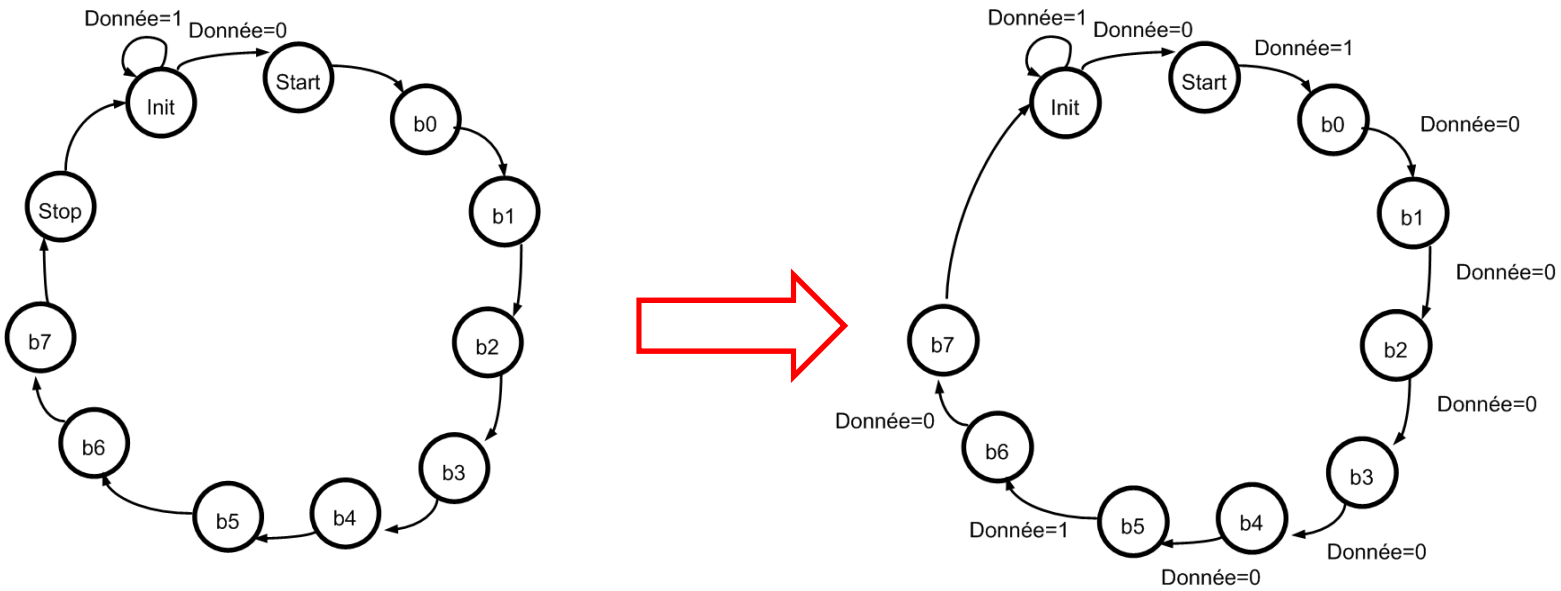
Réception UART

- Peu importe l'approche, le diagramme d'états peut ressembler à ceci...



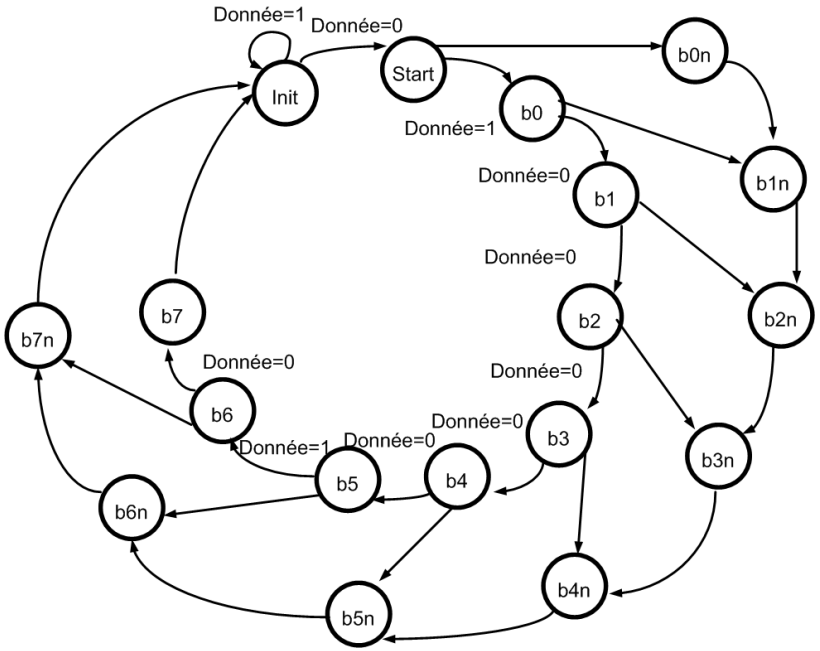
Réception UART

- Si je voulais détecter la lettre A, par exemple...
  - On se rend compte que STOP n'est pas nécessaire...
  - Tous les états donnent une sortie de 0, sauf b7
  - Mais qu'arrive-t-il quand la donnée n'est pas bonne?



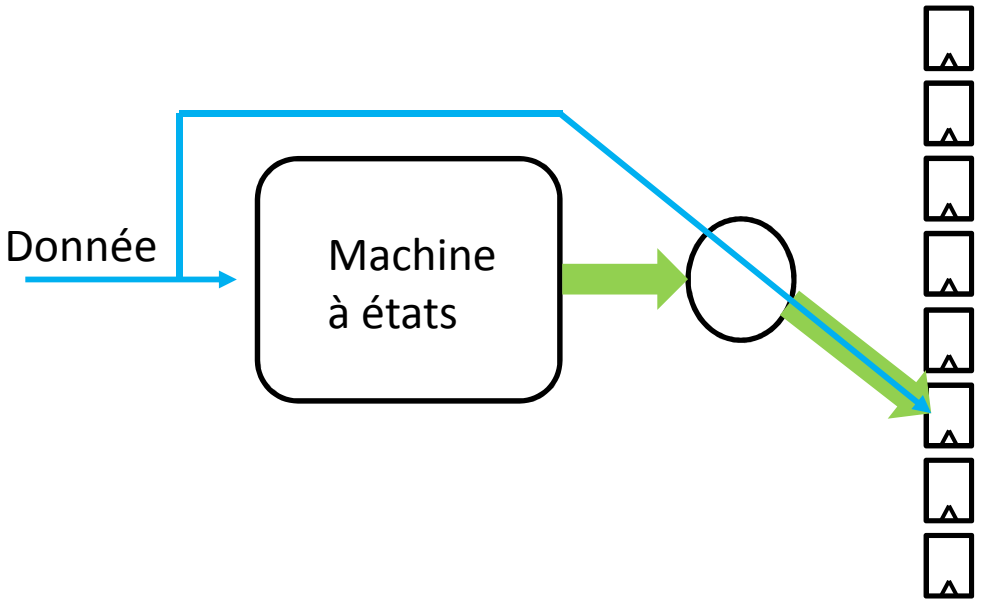
Réception UART

- On devrait avoir une boucle de mauvaise réponse:
  - Une fois dans cette boucle, on n'en sort plus
  - Sert à s'assurer que 8 bits aient été envoyés



Réception UART

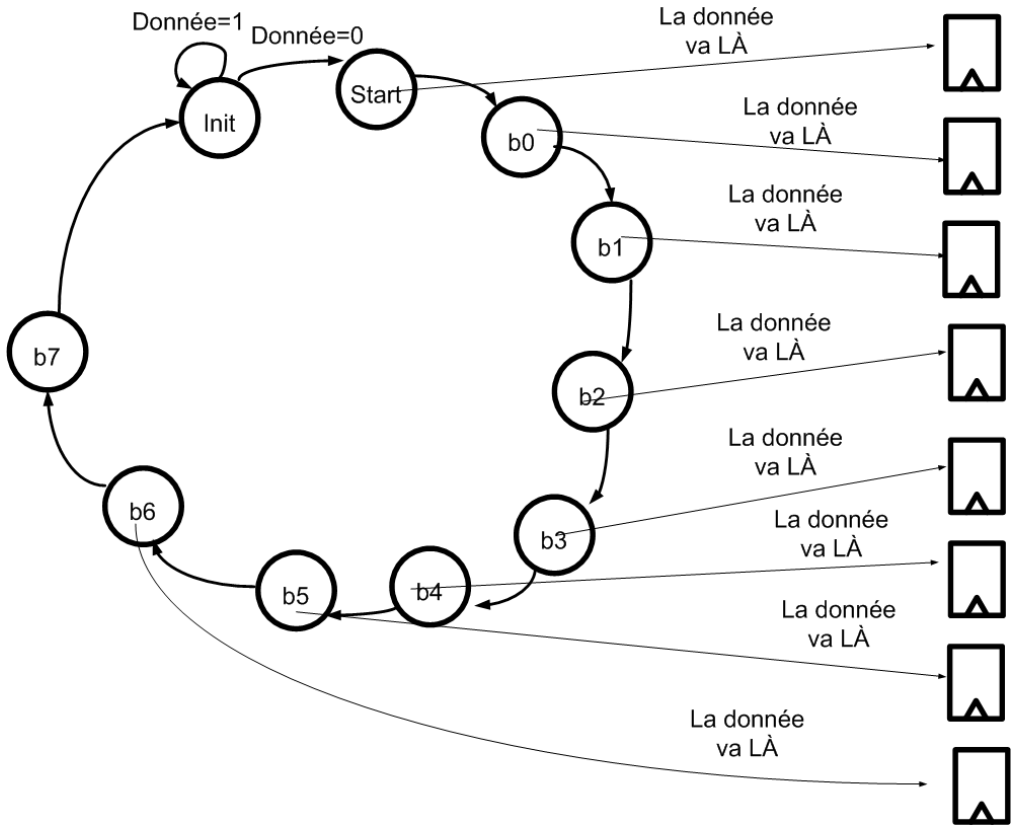
- Une réception plus générale demanderait plus de travail:
  - Une machine à états qui dit où enregistrer les bits
  - Une mémoire qui enregistre les bits



NOTE:
-La sortie du diagramme d'états dit OÙ enregistrer la donnée
-La donnée qui est enregistrée provient du UART

Réception UART

- Si on ouvre la machine à états, on retrouverait ceci:

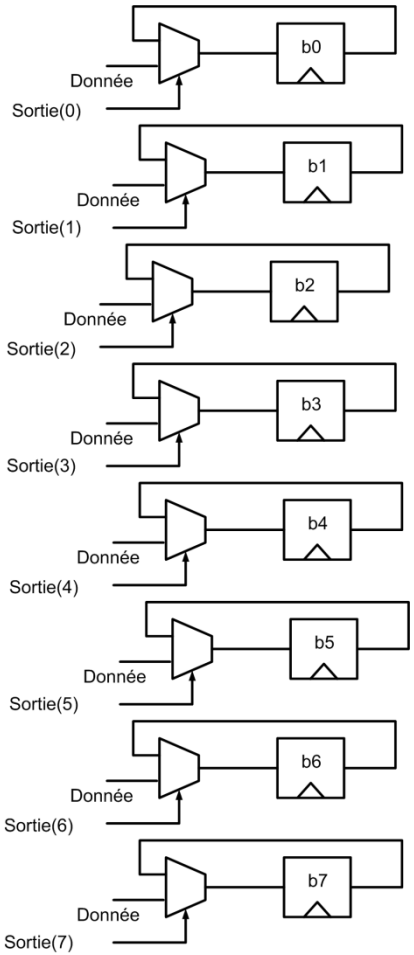


Réception UART

- Pour indiquer où la donnée va aller, on peut utiliser l'encodage one-hot:
  - Quand on est à l'état Start, la donnée qui arrive va aller dans b0
  - On veut alors générer la sortie suivante: "00000001"
  - Quand on est à l'état b0, la donnée qui arrive va aller dans b1
  - On veut alors générer la sortie suivante: "00000010"
  - Et ainsi de suite

Réception UART

- Cette sortie sera utilisée pour écrire dans les flip flops...
- Rappel:
  - Chaque état a une sortie de 8 bits
  - Sortie en one-hot
  - Indique OÙ la donnée doit être enregistrée
  - L'entrée DONNÉE est le même signal UART



Machines à états

- On a vu les machines à états au dernier cours
  - Convertir idées en diagrammes d'états
  - Convertir diagrammes d'états en tableaux d'états
  - Convertir tableaux d'états en circuits
- Au laboratoire, on a vu comment faire les machines à états avec le logiciel:
  - On part directement du diagramme d'états et il génère le code VHDL pour nous
  - Sommes-nous capables de le faire nous-mêmes en VHDL?

Machines à états en VHDL

- Il y a plusieurs façons de concevoir des machines à états en VHDL
- Pour faire une machine à états, on va créer 2 PROCESS:

Process 1

- Un process combinatoire qui va dire “Si on est dans l’état X et que l’entrée était 1, passe à l’état Y”

Process 2

- Les états sont des mémoires (flip flops).. Il faut donc un process séquentiel

Process combinatoire

- Dans le process combinatoire, le but est de savoir QUAND passer d'un état à l'autre
- Un façon de la faire, c'est faire des IF:

```
IF s_etat_present = "000" THEN
    IF entree = '1' THEN
        s_etat_prochain <= "001";
    ELSE
         s_etat_prochain <= "000";
    END IF;
ELSIF s_etat_present = "001" THEN (...)
```

C'est un peu emcombrant... on peut le faire avec des CASE

Process combinatoire

- Pour ce faire il faut:
  - Spécifier la variable qu'on veut comparer
  - Spécifier l'action à prendre pour chaque cas

```
CASE s_etat_present IS
    WHEN “000” => (...)
    WHEN “001” => (...)
    (...)
    WHEN OTHERS => (...)
END CASE;
```

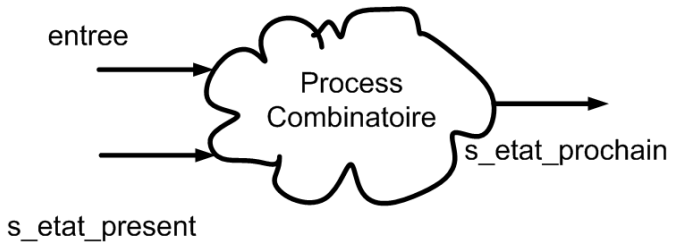
Process combinatoire

- Puisque c'est un process combinatoire, il faut énumerer TOUTES les possibilités
    - Sinon ça donne des éléments mémoires
    - Et dans un process combinatoire, un élément mémoire est une bascule (pas voulu)
- On devrait aussi spécifier les sorties dans le process combinatoire:
    - Pour une machine de Moore, la sortie dépend de l'état
    - Pour une machine de Mealy, ça depend aussi du input

Process combinatoire

- Un exemple plus complet ressemblerait à:

```
CASE s_etat_present IS
    WHEN “000” =>
        sortie_moore <= ‘0’;
        IF entree = ‘1’ THEN
            sortie_mealy <= ‘0’;
            s_etat_prochain <= “001”;
        ELSE
            sortie_mealy <= ‘1’;
            s_etat_prochain <= “000”;
        END IF;
    WHEN “001” => (...)
END CASE;
```

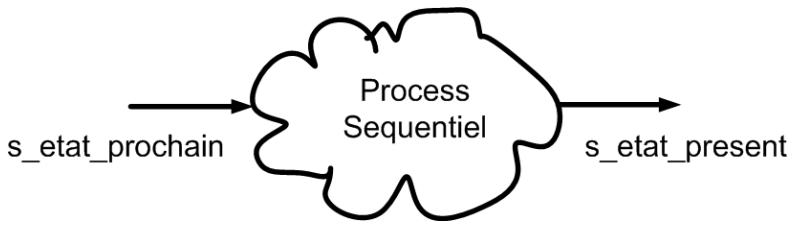


Tout ceci est à l’interieur d’un process

Process séquentiel

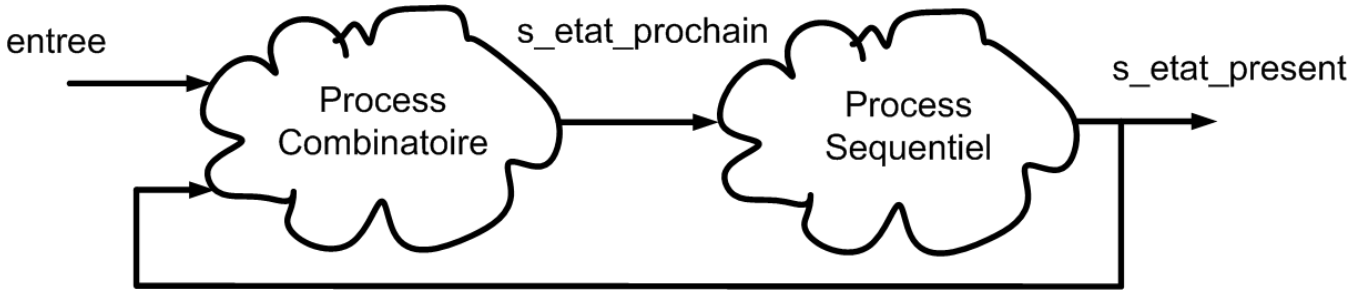
- Dans le process séquentiel, on a les flip flops:
  - Toutes les ASSIGNATIONS deviennent des flip flops
- Dans une machine à états, les flip flops contiennent les états (s_etat_present)
  - Ce serait donc la seule chose à avoir:

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        s_etat_present <= s_etat_prochain;
    END IF;
END PROCESS;
```



Machine à états en VHDL

- On aura le circuit suivant:
  - Si on écrivait bien le code, le circuit final devrait ressembler à ceci...



Définir les types

- On est habitué à fonctionner avec les STD_LOGIC et STD_LOGIC_VECTOR...
    - Il existe aussi plusieurs autres types qu'on découvrira éventuellement
- En plus, on peut aussi définir nos propres types
    - Utile pour les machines à états

Definir les types

- On definit souvent les états avec 000, 001,...
  - Souvent, ça ne veut rien dire...
  - Plus tard, il est très difficile de se rappeler de leur signification
- On pourrait définir un TYPE qui s'appelle ETAT
  - Aurait des noms plus intéressants:
  - Exemple: etat_debut, etat_3undesuite, etc.
- Pour ce faire, on déclare avec les SIGNAL:

```
ARCHITECTURE rtl OF quelquechose IS
TYPE etats IS (etat_debut, etat_3undesuite, etat_fin);
SIGNAL s_etat_present : etats;
BEGIN
```

Exemple complet en VHDL

- Nous aimerions faire une machine à états pour une machine distributrice de films:
  - Elle n'accepte que les 1$ et les 2$ (1 entrée pour chaque... les 2 ne peuvent pas entrer en même temps)
  - à partir de 5$, ça donne un film (sortie film = '1')
  - Si on obtient 6$, ça redonne la monnaie de 1$ (en mettant monnaie='1')
  - NOTE: Si l'usager mettait 1$ ou 2$ PENDANT que la machine donne le film, l'argent est perdu!
  - Faites une machine de Moore..

Exemple complet en VHDL

- On veut transformer ça en diagrammes d'états
  - Pour énumerer les états, on imagine un scénario
  - On imagine le cas où une personne met 5 1$ de suite...
- Au début, il n'y a pas d'argent...
  - Après ça, il y a 1$
  - Il y a 2$
  - Il y a 3$
  - Il y a 4$
  - Il y a 5$... ça donne un film et ca revient a 0$...

Diagramme d'états

- Notre premier diagramme d'états ressemble à ceci:
  - Les 2 sorties sont le film et la monnaie
  - On voit que la monnaie est toujours 0
  - Il manque aussi la possibilite de mettre 2$...

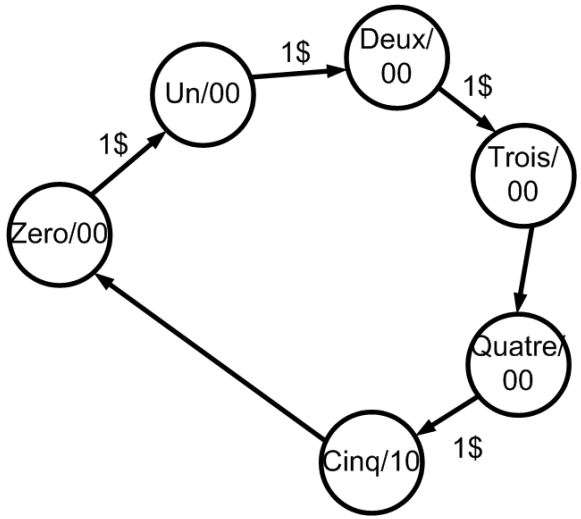


Diagramme d'états

- On ajoute toutes les transitions pour 2$:
  - À la place de passer à l'autre état, on en saute 1
  - Il faut aussi ajouter l'etat 6$... Si j'ai 4$ et je mets 2$
  - C'est seulement à 6$ que j'ai 1$ de monnaie...

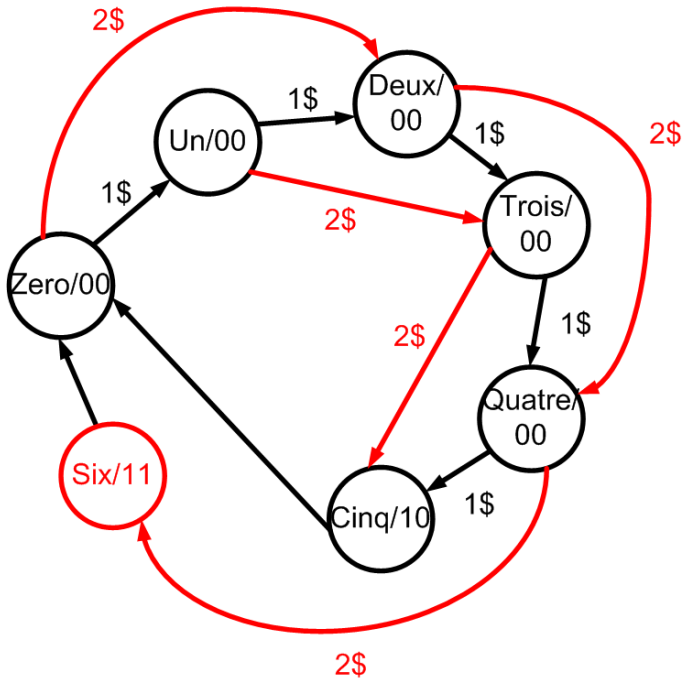


Diagramme d'etats

- À partir du diagramme d'états, je peux déjà écrire le VHDL
- On définit les états entre ARCHITECTURE et le BEGIN...

```
TYPE etats IS (zero, un , deux, trois, quatre, cinq, six);
SIGNAL etat_present  : etats;
SIGNAL etat_prochain : etats;
```

- On écrit maintenant les 2 process (un combinatoire et un séquentiel)

Process séquentiel

- On commence par le process séquentiel puisqu'il est toujours presque pareil

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        s_etat_present <= s_etat_prochain;
    END IF;
END PROCESS;
```

- C'est le même code qu'on avait vu il y a quelques diapos...

Process combinatoire

- On considère le cas où on a 0$...

```
CASE etat_present IS
        WHEN zero =>
            film <= '0';
            monnaie <= '0';
            IF un_dollar = '1' THEN
                etat_prochain <= un;
            ELSIF deux_dollars = '1' THEN
                etat_prochain <= deux;
            ELSE
                etat_prochain <= zero;
            END IF;
```

Machine de Moore alors les sorties sont ici..

Avec 1$ je passe à UN
Avec 2$ je passe à DEUX
Sinon, je reste là

Process combinatoire

- Les états jusqu'à 4$ (inclus) sont semblables

```
WHEN quatre =>
    film <= '0';
    monnaie <= '0';
    IF un_dollar = '1' THEN
        etat_prochain <= cinq;
    ELSIF deux_dollars = '1' THEN
        etat_prochain <= six;
    ELSE
        etat_prochain <= quatre;
    END IF;
```

Toujours pas assez d'argent

Avec 1$ je passe à CINQ
Avec 2$ je passe à SIX
Sinon, je reste là

Process combinatoire

- À 5$, on a le film et on passe à 0$

```
WHEN cinq =>
        film <= '1';
        monnaie <= '0';
        etat_prochain <= zero;
```

- À 6$, on a le film, on redonne 1$ et on passe automatiquement a 0$ au prochain état

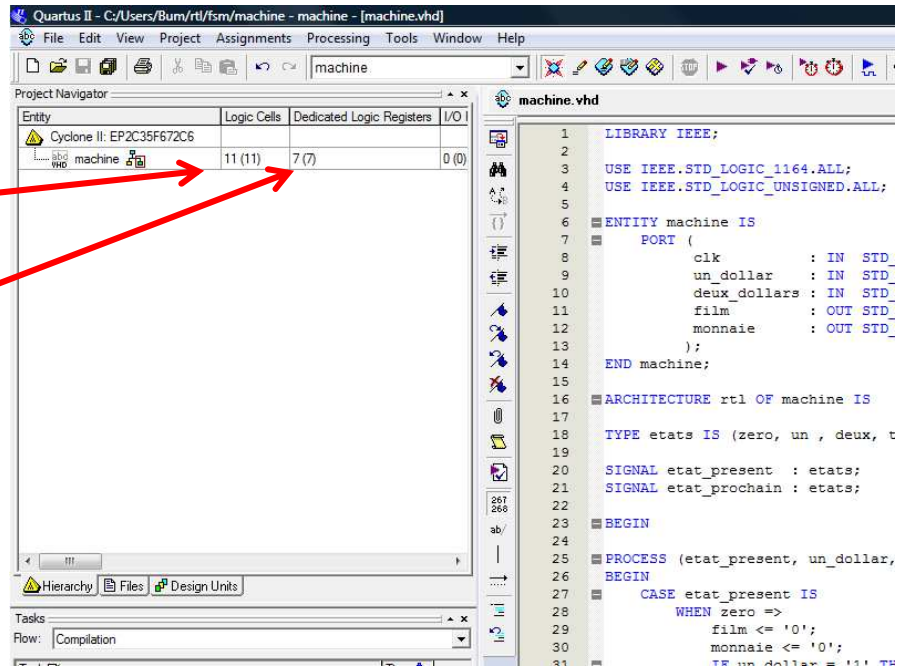
```
 WHEN six =>
        film <= '1';
        monnaie <= '1';
        etat_prochain <= zero;
```

Synthèse

- Le code VHDL complet devrait être sur la page web du cours...
- Après avoir fait le code, on pourrait regarder le résultat:

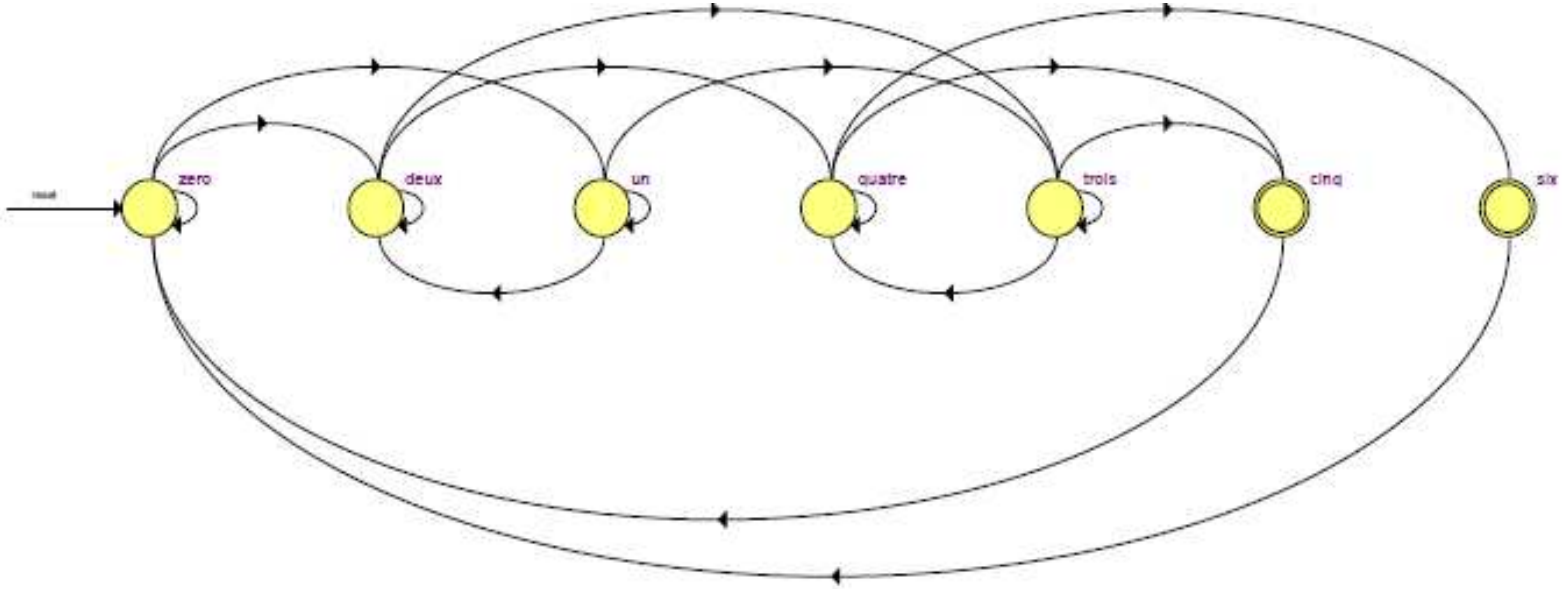
Ç utilise 11 cellules logiques

Il utilise 7 flip flops...



Synthèse

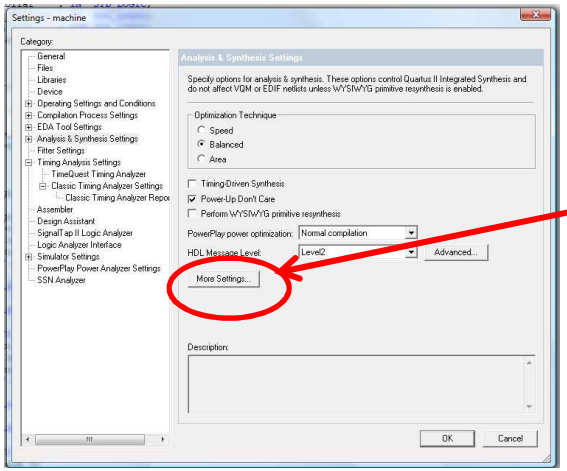
- Si on avait défini un TYPE , le logiciel peut parfois détecter qu'il y a machine d'états
- Dans ce cas, il peut générer l'image suivante



Ça se trouve sous Tools → Netlist Viewers → State machine viewer

Synthèse

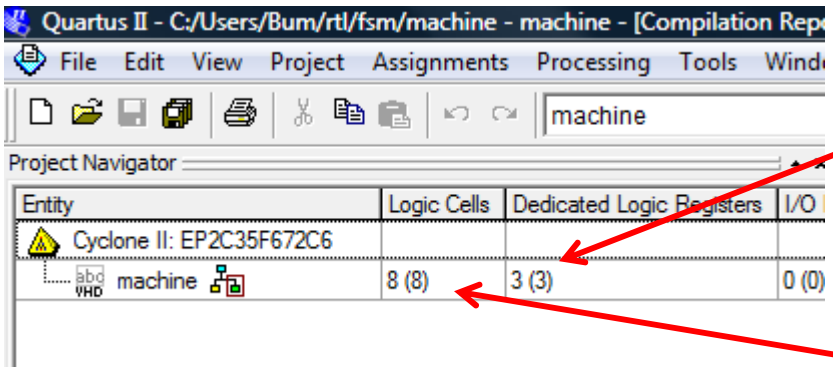
- Notre machine a 7 états et Quartus a utilisé 7 flip flops
  - Il a voulu l'implementer en utilisant l'encodage one-hot
  - C'est lui qui décide ce qu'il trouve le mieux
- Il est possible de lui imposer l'encodage voulu
  - Il faut aller dans Assignments puis Settings



Quand cette fenêtre apparait, cliquez sur More Settings

Synthèse

- Une autre fenêtre apparaît:
  - Sélectionnez l'option State Machine Processing
  - La valeur montrée devrait etre AUTO...
  - Changez-le à MINIMAL BITS
- Après la synthèse, ça donne ceci...



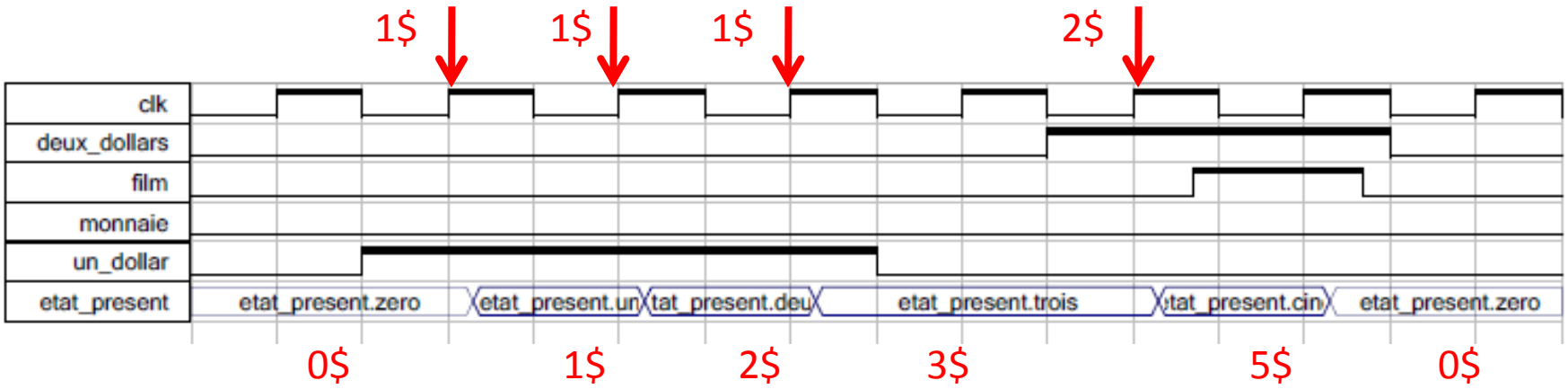
On a 3 bits...

On a 8 cellules logiques

Avec nos connaissances, on est capable de mieux guider le logiciel...

Simulation

- Finalement, on peut le simuler
  - On créée une horloge
  - Parfois on met 1$ parfois on met 2$
  - Les états sont en bas



Ça semble bien fonctionner... Testons le système de monnaie...

Simulation

- Durant cette partie de la simulation, on a mis l'état à 4$ et puis on a ajouté 2$
  - On obtient le film
  - Et en même temps, on reçoit la monnaie