Systèmes Digitaux

Cours 8

Aujourd'hui

- On va prendre une pause des circuits séquentiels
- On va plutôt se concentrer sur l'apprentissage d'un langage de description matériel
    - Comme un langage de programmation à l'exception de quelques détails

Introduction

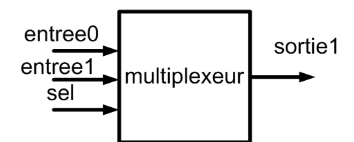
- Le VHDL est un langage developpé pour la simulation de circuits
  - Vérifier le fonctionnement des circuits sans les construire
- Éventuellement, c'est devenu un langage pour concevoir des circuits
  - Les outils sont capables de prendre certaines commandes et les transformer en circuit
  - Pour ça, il faut respecter les conventions

Introduction

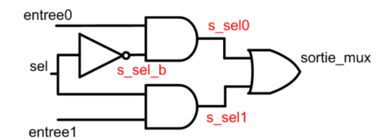
- Le processus de développement est le suivant:
  - On écrit le programme (VHDL)
  - On synthétise (“compile”) pour traduire en portes logiques
  - On simule et on change le design au besoin
  - On assigne les PINs
  - On RESYNTHÉTISE (“re-compile”) pour faire les connexions aux PINs
  - On programme le FPGA

Procédure générale

- Pour être efficace, on peut suivre une procédure générale:
  1) Faire un diagramme vu de l'extérieur:
     - Ça montre les entrées et les sorties



  2) Faire un diagramme du circuit interne
     - Ça montre comment le tout est connecté



  3) Énumérer les signaux internes et donner un nom à chacun
  4) Compiler souvent <u>avant même</u> d'avoir terminé
     - Ça évite des corrections majeures à la fin

Structure

- Le VHDL contient 3 grandes parties:
  - Les libraries
  - L’entité
  - L’architecture

Librairies

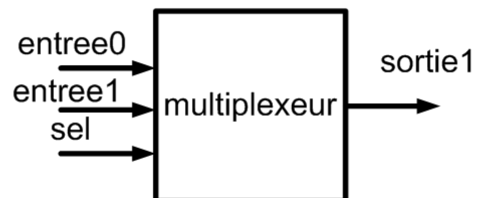
- La spécification des librairies est un copier-coller (pour les buts du cours):

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Entité

- Le module VU DE L'EXTÉRIEUR s'appelle l'entité:

```
ENTITY multiplexeur IS
 PORT (
       entree0 : IN  STD_LOGIC;
       entree1 : IN  STD_LOGIC;
       sel:      IN  STD_LOGIC;
       sortie1 : OUT STD_LOGIC
      );
END multiplexeur;
```



Entité

- STD_LOGIC est une autre façon de dire “bit”:
    - Les bits sont un peu restrictifs: 0 ou 1
- Avec les STD_LOGIC, on peut spécifier des comportements plus complexes:
    - Un noeud flottant (‘Z’)
    - Une valeur non spécifiée (‘U’)
    - Don’t care (‘-’)
    - Etc.

Entité

- STD_LOGIC c’est pour spécifier 1 bit
- Pour plusieurs bits, c’est STD_LOGIC_VECTOR
  - Il faut aussi spécifier la taille
  - STD_LOGIC_VECTOR(3 DOWNTO 0) est un vecteur de 4 bits
  - STD_LOGIC_VECTOR(7 DOWNTO 0) est un vecteur de 8 bits
- L’usage de DOWNTO veut dire que le bit à gauche est le plus significatif…

Architecture

- Finalement, on a l'architecture:
  - L'architecture contient l'information interne du design
- Sa structure ressemble à:

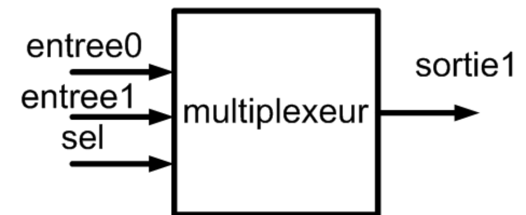
```
ARCHITECTURE rtl OF multiplexeur IS
  Signaux internes (SIGNAL) et définition des
  modules à inclure (COMPONENT)
BEGIN
  Description du système. Soit avec des commandes ou
  avec des instanciations.
END;
```

Le mot rtl est un nom qu'on donne... Ça aurait pu être n'importe quoi

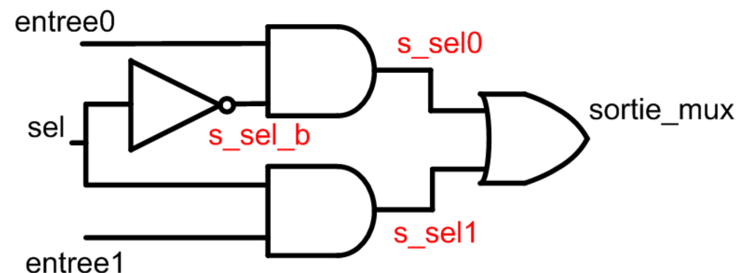
Le mot "multiplexeur" doit correspondre au nom de entité

Procédure générale

- La vue extérieure du module donne:
  - Il a 2 entrées: entree0 et entree1
  - Il a 1 entrée qui sélectionne
  - Il a 1 sortie

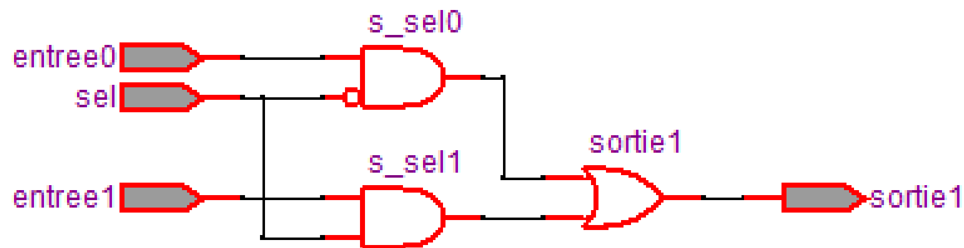


- Par la suite, on dessine le schéma général pour identifier les signaux à l'interne
  - Pour en arriver à ça, il a fallu faire la table de verité, la table de Karnaugh et peut-être la logique de Boole...



Retour à l’architecture

```
ARCHITECTURE rtl OF multiplexeur IS
   SIGNAL s_sel0 : STD_LOGIC;
   SIGNAL s_sel1 : STD_LOGIC;
   SIGNAL s_sel_b : STD_LOGIC;
BEGIN
   s_sel_b <= NOT sel;
   s_sel0 <= entree0 AND s_sel_b;
   s_sel1 <= entree1 AND sel;
   sortie1 <= s_sel0 OR s_sel1;
END;
```



Types de description

- Ce qu'on vient de faire s'appelle le VHDL structurel:
  - On décrit la structure en montrant les connexions
  - Ça ressemble beaucoup aux circuits en diagramme bloc
- Il existe d'autres façons de décrire un système
  - VHDL structurel
  - VHDL rtl (certains appellent ça "comportemental")
  - Une combinaison des deux

Les dessins en diagramme bloc sont souvent plus rapides à concevoir
que le VHDL structurel

VHDL RTL

- VHDL RTL nous permet de se déconnecter des détails d'implantation
  - On décrit le système avec des phrases (presque)
  - Rappel: Il faut respecter les conventions!
- Il existe plusieurs façons de décrire un multiplexeur
  - On va en montrer 2: à l'extérieur d'un PROCESS et à l'intérieur d'un PROCESS
  - Les 2 sont bons...

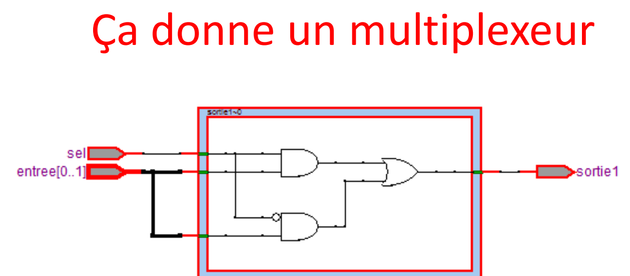
Commençons avec la méthode à l'extérieur d'un PROCESS

La commande WHEN

- La commande WHEN s’utilise à l’extérieur d’un PROCESS seulement
    - C’est quoi un PROCESS? On va le voir bientot
- On peut décrire un multiplexeur comme suit:
    - On met sortie1 <= entree0 quand sel=0
    - Sinon, on met sortie1 <= entree1

```
ARCHITECTURE rtl OF multiplexeur IS
BEGIN
      sortie1 <= entree0 WHEN sel=‘0’
                                ELSE
                      entree1;
END;
```

Ça donne un multiplexeur



Un PROCESS

- Un PROCESS est un regroupement de commandes:
  - Tous les process s'exécutent en parallèle
  - À l'interieur d'un process, il y a un peu de séquence
  - Très différent de la programmation séquentielle typique
- Il existe des process combinatoires et des process séquentiels
  - Aujourd'hui, on voit les commandes combinatoires

Dans les process, on a accès à des commandes différentes...

Un PROCESS

- Un process est utilisé comme ceci

```
ARCHITECTURE rtl OF multiplexeur IS
BEGIN
  PROCESS (LISTE DE SENSIBILITE)
  BEGIN
      COMMANDES DANS LE PROCESS
  END PROCESS;
END;
```

  - Il peut y avoir plusieurs process dans une architecture
  - Chaque process roulera quand un élément de sa liste de sensibilité change...

Process combinatoire

- Aujourd’hui, on se concentre sur les process combinatoires
- Il faut respecter les règles combinatoires:
    - TOUS les signaux qui sont utilisés (lus) DOIVENT être dans la liste de sensibilité
    - Les signaux qui sont modifiés (écrits) ne peuvent PAS être dans la liste de sensibilité
    - TOUTES les possibilités doivent etre considérés
    - <u>NOTE: Le logiciel ne donnera pas d’erreur. Ça va juste MAL fonctionner</u>

Process combinatoire

- Faisons un multiplexeur
  - On commence par remplir la liste de sensibilité
  - Tous les signaux qui sont utilisés/lus doivent y être

```
PROCESS (entree0, entree1, sel)
BEGIN
  COMMANDES A AJOUTER DANS LE PROCESS
END PROCESS;
```

N'utilisez pas d'accents dans le code VHDL!

Process combinatoire

- La commande la plus utilisée est le IF:
  - IF est disponible seulement dans les PROCESS
  - "SI (une telle condition) ALORS (on fait ça)"
  - "SINON (on fait cette autre chose)"

```
PROCESS (a,b)
BEGIN
       IF a='1' THEN
       ...
       ELSIF b='1' THEN
       ...
       ELSE
       ...
       END IF;
END PROCESS;
```

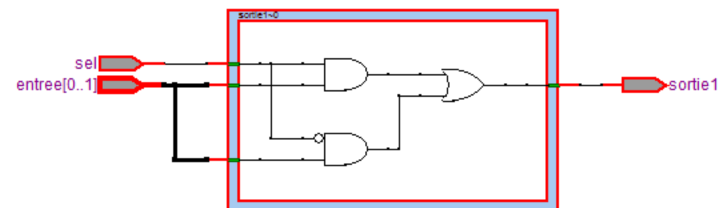
Exemple fictif... Ce n'est pas notre multiplexeur

Process combinatoire

- Une façon de faire le multiplexeur est ceci:
  - “Si sel=0, la sortie sera entree0”
  - “Sinon, la sortie sera entree1”

Toutes les conditions sont considérées

```
PROCESS (entree0, entree1, sel)
BEGIN
   IF sel = ‘0’ THEN
       sortie1 <= entree0;
   ELSE
       sortie1 <= entree1;
   END IF;
END PROCESS;
```



Ça donne le même schéma

Pourquoi les règles?

- Que se passerait-il si on n’obéissait pas aux règles?
  - Il n’y aurait pas d’erreur détecté par le logiciel (peut-être des avertissements)
  - On le mettrait sur le FPGA et ça ne fonctionnerait pas comme prévu
- Les règles sont là pour qu’on puisse savoir à 100% comment le circuit se comportera
  - Sinon, ça ne fonctionnera peut-être pas...

Pourquoi les regles?

- Que se passe-t-il si la liste de sensibilité n’est pas complète?

```
PROCESS (sel)
BEGIN
   IF sel = ‘0’ THEN
        sortie1 <= entree0;
   ELSE
        sortie1 <= entree1;
   END IF;
END PROCESS;
```

Pourquoi les règles?

- Durant la simulation, le PROCESS est executé quand un signal change dans la liste
  - Sinon, il ne fait rien
- Imaginons le cas suivant:
  - sel=0, donc sortie = entree0
  - Si entree0 passe de 1 a 0...
  - entree0 n'est pas dans la liste, le process ne sera pas executé et la sortie ne changera pas

On s'attendrait à avoir la sortie en rouge

Pourquoi les règles?

- Quand on fait la synthèse (compilation), le logiciel ne regarde pas la liste
  - La synthèse donnera un multiplexeur mais la simulation donnera quelque chose de bizarre
  - On VEUT que la simulation et la synthèse concordent
  - Il faut donc que la liste de sensibilité soit complète.

Pourquoi les règles?

- Qu'arrive-t-il quand les possibilités ne sont pas toutes énumerées?

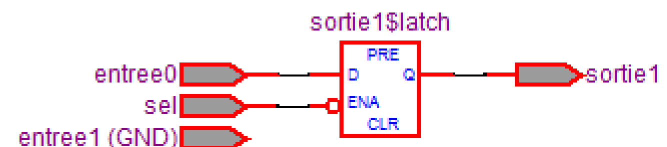
```
PROCESS (entree0, entree1, sel)
BEGIN
   IF sel = '0' THEN
        sortie1 <= entree0;
   ELSE
        sortie1 <= entree1;
   END IF;
END PROCESS;
```

Enlevons cette partie...

Pourquoi les règles?

- Si la sel=0, la sortie1 <= entree0
  - Sinon… qu'est-ce qui se passe?
  - Le VHDL va GARDER la valeur précédente

```
PROCESS (entree0, entree1, sel)
BEGIN
   IF sel = '0' THEN
      sortie1 <= entree0;
   END IF;
END PROCESS;
```



Il va créer un élément mémoire… Ce serait une bonne description pour une bascule D
Mais, ce n'est pas ce qu'on veut!

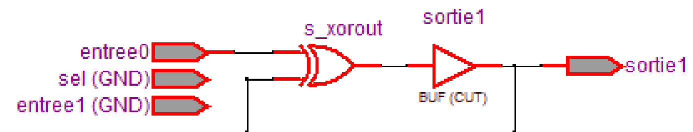
Pourquoi les règles?

- Pourquoi est-ce que les valeurs modifiées ne doivent pas être dans la liste de sensibilité?
  - Ça fait des “boucles combinatoires” et c’est normalement pas bon
- Si la sortie n’est pas utilisée, ça ne change rien
- Si elle est utilisée, on risque d’avoir des problèmes...

Pourquoi les règles?

- Considérons le code suivant:

```
PROCESS (entree0, s_xorout)
BEGIN
   s_xorout <= s_xorout XOR entree0;
END PROCESS;
```



- On lit entree0 et s_xorout, donc ils doivent être dans la liste de sensibilité
    - La sortie change et on est sensible à la sortie
    - Donc, le process recommence... Ça change la sortie
    - Le process recommence ENCORE.. Ça change la sortie...

C'est une boucle infinie...

Conception

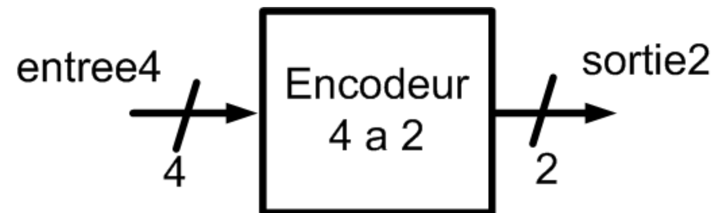
- Passons maintenant à la conception d’un module encodeur 4 à 2
- On va passer par toutes les étapes typiques
    - Définir l’entité
    - Définir l’architecture
- On va tenter de comprendre le raisonnement

Exemple (seul)

- Concevez un encodeur de priorite 4 à 2 en VHDL...
  - On donne la plus grande priorité aux bits les plus significatifs
  - Pas besoin d'écrire les libraries pour cet exemple...

Exemple (seul)

- Je commence par dessiner l'extérieur
  - Ça m'aide à définir l'entité:




```
ENTITY encodeur4a2 IS
     PORT (
      entree4: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
      sortie2: OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
            );
      END encodeur4a2;
```

La barre oblique avec le chiffre dit que le signal a plusieurs bits

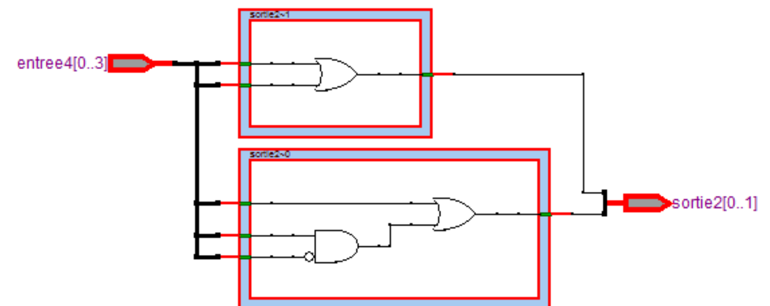
Exemple (seul)

- Pour l’architecture, on devrait réflechir un peu
    - Si le bit le moins significatif, entree4(0), est 1, la sortie sera 00
    - Si le deuxième bit, entree4(1), est 1, la sortie sera 01
    - Si le troisième bit, entree4(2), est 1, la sortie sera 10
    - Si le bit le plus significatif, entree4(3), est 1, la sortie sera 11
- On va commencer par voir si entree4(3) est 1
    - Si oui, ça va donner 11... Sinon, on regarde le prochain

Exemple (seul)

```
ARCHITECTURE rtl OF encodeur4a2 IS
BEGIN
   PROCESS (entree4)
   BEGIN
        IF entree4(3)='1' THEN
                sortie2 <= "11";
        ELSIF entree4(2)='1' THEN
                 sortie2 <= "10";
        ELSIF entree4(1)='1' THEN
                sortie2<= "01";
        ELSE
                sortie2<= "00";
        END IF;
   END PROCESS;
END;
```

On commence avec le plus prioritaire et on continue vers le moins prioritaire



Quelques notes intéressantes

- L'addition, la soustraction et la multiplication peuvent se faire rapidement:
  - Il n'est pas nécessaire de les créer a la main
- On peut utiliser les commandes +, - et *
  - On pourrait donc faire l'opération suivante:

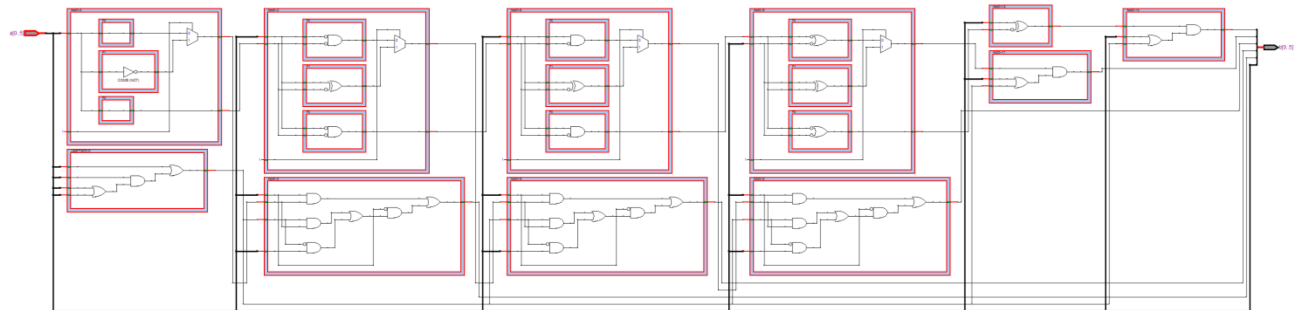
```
produit <= a * b;
```

- On ne peut pas faire de division sauf pour des puissances de 2
  - Pour la division, il faudrait le faire à la main

Quelques notes intéressantes

- Il est aussi possible de faire des comparaisons:
  - On peut utiliser =, > et < sans avoir à les concevoir
  - Le logiciel devrait comprendre ces termes et les traduire en portes logiques automatiquement
- Ce code devrait fonctionner (dans un process)

```
IF a > "001001" THEN
    b <= a + "000110";
ELSE
    b <= a;
END IF;
```



Quelques notes intéressantes

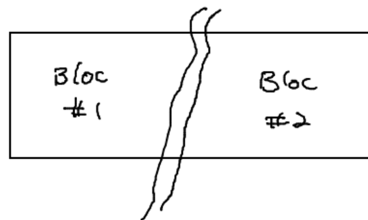
- La meilleure façon de verifier si un code est bon c'est avec le logiciel
    - Faites une compilation SOUVENT et le logiciel vous dira où vous avez fait des erreurs (s'il y en a)
    - Une tâche qui n'est pas evidente c'est de comprendre les messages du logiciel
    - Corriger une ou deux erreurs et recompilez. Parfois plusieurs erreurs vont disparaître en même temps.

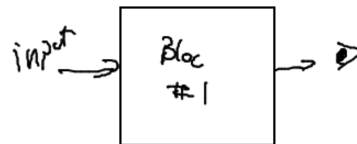
Quelques notes intéressantes

- **LA** tâche qui demarque un bon ingénieur c'est le déverminage (débogage):
    - Êtes-vous capables de résoudre les problemes?
- Tout le monde est un bon ingenieur quand ça fonctionne bien
    - Un bon ingénieur sera capable faire des tests et trouver les erreurs quand ça va mal
    - Prenez le temps de développer vos propres techniques

Exemples de techniques

- Propagez les signaux et examinez les signaux intermédiaires
  - “Est-ce qu’il fonctionne encore bien rendu ici?”
  - La simulation permet d’examiner les signaux intermédiaires
  - On peut aussi faire sortir les signaux intermédiaires aux PINs sur la carte FPGA
- Brisez le système en moreaux et stimulez-les de façon séparée...





Circuit séquentiel en VHDL

- Pour avoir des circuits séquentiels en VHDL, il faut faire des PROCESS
- Ces process auront des particularités qu'il FAUT respecter à tout prix:
  - La liste de sensibilité ne contient QUE l'horloge
  - La structure DOIT être

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        ...On fait ce qu'on veut ici SEULEMENT
    END IF;
END PROCESS;
```

Circuit séquentiel en VHDL

- Un process séquentiel a plusieurs caractéristiques:
  - Toutes les assignations forment des flip flops
  - Les assignations sont faites en parallèle
  - Dans un process séquentiel, on a le droit de lire et d’écrire sur le même signal

Flip flops

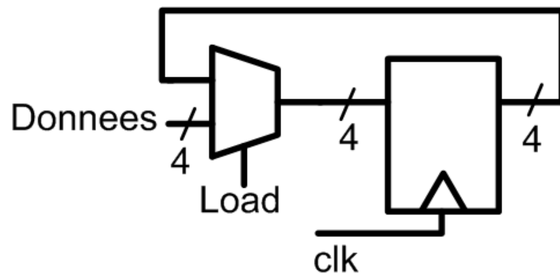
- Le circuit séquentiel le plus simple, c’est le flip flop…

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

Q et D peuvent être de n’importe quelle taille

Registre avec chargement parallèle

- Pour charger les données en présence d'autorisation
  - Il faut définir les signaux à 4 bits (si c'est ce qu'on veut)




```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        IF load = '1' THEN
            Q <= donnees;
        END IF;
    END IF;
END PROCESS;
```