Systemes Digitaux

Introduction au VHDL

Introduction

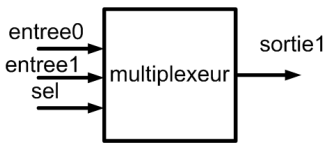
- Le VHDL est un langage developpe pour la simulation de circuits
    - Les gens pouvaient savoir si leurs circuits fonctionnaient bien ou pas
- Eventuellement, c'est devenu un langage pour concevoir des circuits
    - Les outils sont capables de prendre CERTAINES commandes et les mettre en circuit
    - Pour ca, il faut respecter les conventions

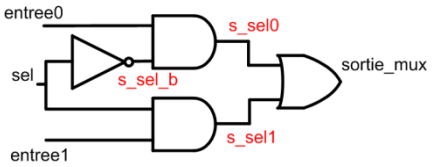
Introduction

- Le processus normal est le suivant:
  - On ecrit le VHDL
  - On synthetise (parfois on dit “compile”) pour traduire en portes logiques
  - On simule et on change le design au besoin
  - On assigne les PINs
  - On fait le “placement et routage” pour placer les portes logiques sur le FPGA et faire les connexions
  - On programme le FPGA

Procedure generale

- Pour etre efficace, on peut suivre une procedure generale:
    1) Faites un diagramme vu de l'exterieur: ca nous dit quelles entrees et sorties on a besoin pour l'entite
    2) Faites un diagramme du circuit interne: ca nous dit comment le tout est connecte
    3) Enumerez les signaux internes et donnez un nom a chacun (nom intelligent, de preference)
    4) Compilez souvent avant meme d'avoir fini... ca vous dira quoi corriger au fur et a mesure





Structure

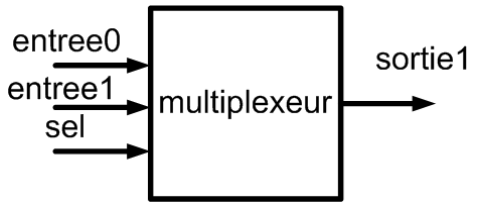
- Le VHDL a 3 grandes parties:
  - Les libraries
  - L'entite
  - L'architecture
- La specification des librairies est un copier-coller (pour les buts du cours):

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

Entite

- Par la suite on specifie le module VU DE L'EXTERIEUR avec l'entite:

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



Architecture

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

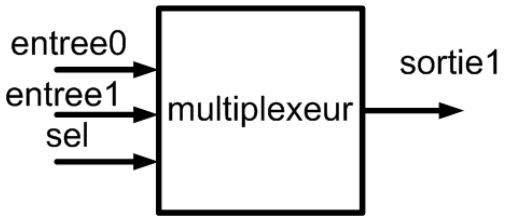
```
ARCHITECTURE rtl OF multiplexeur IS
   Signaux internes (SIGNAL) et definition des
   modules a inclure (COMPONENT)
BEGIN
   Description du systeme. Soit avec des commandes ou
   avec des instanciations.
END;
```

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

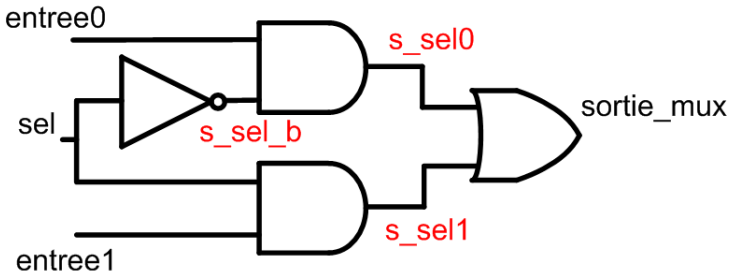
Le mot "multiplexeur" doit correspondre au nom de l'entite

Procedure generale

- On avait dessine sa vue exterieure:
  - Il a 2 entrees: entree0 et entree1
  - Il a 1 entree qui selectionne
  - Il a 1 sortie



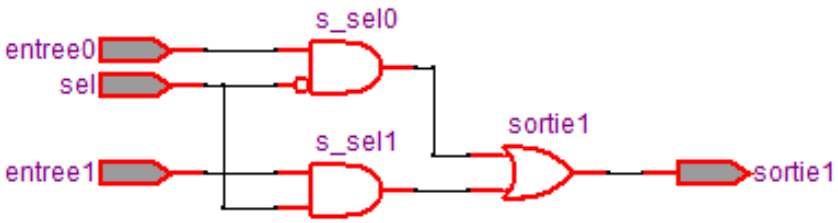
- Par la suite, on dessine le schema general pour identifier les signaux a l'interne
  - Pour en arriver a ca, il a fallu faire la table de verite, la table de Karnaugh et peut-etre la logique de Boole...



Retour a 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;
```

Dans Quartus II, on peut entrer dans
Tools→Netlist Viewers→RTL Viewer pour voir ceci
(apres compilation)



Types de description

- Ce qu'on vient de faire s'appelle le VHDL structurel:
  - On decrit la structure en montrant les connexions
  - C'est intuitif et on est habitue a ca...
- Il existe d'autres facons de decrire un systeme
  - VHDL structurel
  - VHDL rtl (certains appellent ca "comportemental")
  - Une combinaison des deux

VHDL RTL

- VHDL RTL nous permet de se deconnecter des details d'implementation
  - On va decrire le systeme avec des phrases (presque)
  - Rappel: Il faut respecter les conventions!
- Il existe plusieurs facons de decrire un multiplexeur
  - On va en montrer 2: a l'exterieur d'un PROCESS et a l'interieur d'un PROCESS
  - Les 2 sont bons...

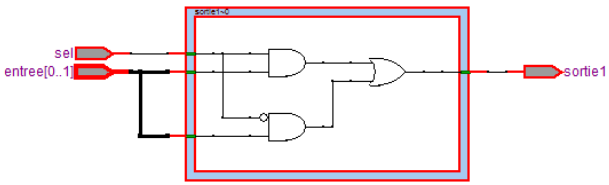
Commencons avec la methode a l'exterieur d'un PROCESS

La commande WHEN

- La commande WHEN s'utilise a l'exterieur d'un process seulement
  - C'est quoi un process? On va le voir bientot
- On peut decrire 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;
```

Ca donne un multiplexeur



Un PROCESS

- Un concept plus puissant est celui du PROCESS
  - Un process est un regroupement de commandes qui font quelque chose
  - Tous les process s'executent en parallele
  - A l'interieur d'un process, les commandes ont "comme" une sequence
- Il existe des process combinatoires et des process sequentiels

Dans les process, on a acces a des commandes differentes...

Un PROCESS

- Un process est utilise 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 element de sa liste de sensibilite change...

Process combinatoire

- Aujourd'hui, on se concentre sur les process combinatoires
- Il faut respecter les regles (combinatoires):
  - TOUS les signaux qui sont utilises (lus) DOIVENT etre dans la liste de sensibilite
  - Les signaux qui sont modifies ne peuvent PAS etre dans la liste de sensibilite
  - TOUTES les possibilites doivent etre consideres
  - NOTE: Le logiciel ne donnera pas d'erreur. Ca va juste MAL fonctionner

Process combinatoire

- On essaie de faire un multiplexeur...
- On va commencer par remplir la liste de sensibilite
    - Tous les signaux qui sont utilises/lus doivent y etre

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

Process combinatoire

- La commande la plus utilisee est le IF:
  - IF est disponible seulement dans les PROCESS
  - “SI (une telle condition) ALORS (on fait ca)”
  - “SINON (on fait cette autre chose)”

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

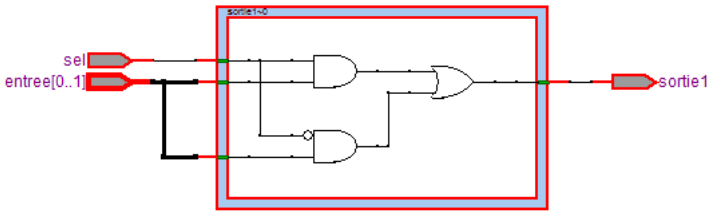
Exemple fictif... Ce n’est pas notre multiplexeur

Process combinatoire

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

Toutes les conditions sont considerees

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



Ca donne encore le meme schema

Pourquoi les regles?

- Que se passerait-il si on n'obeissait pas aux regles?
  - Il n'y aurait pas d'erreur detecte par le logiciel (peut-etre des avertissements)
  - On le mettrait sur le FPGA et ca ne fonctionnerait pas comme prevu
- Les regles sont la pour qu'on puisse savoir a 100% comment le circuit se comportera
  - Sinon, ca ne fonctionnera peut-etre pas...

Pourquoi les regles?

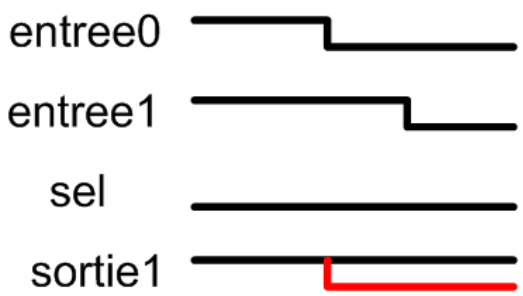
- Que se passe-t-il si la liste de sensibilite n'est pas complete?

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

entree0 et entree1 ne sont pas la quand ils devraient l'etre

Pourquoi les regles?

- Durant la simulation, le process est execute quand un signal change dans la liste
  - Sinon, il ne fait rien
- Imaginons le cas ou sel=0 et entree0 passe de 1 a 0...
  - Puisque entree0 n'est pas dans la liste de sensibilite, le process ne sera pas execute et la sortie ne changera pas



On s'attendrait a avoir la sortie en rouge

Pourquoi les regles?

- Quand on transforme en portes logiques (“synthese”), le logiciel ne regarde pas la liste
  - La synthese donnera un multiplexeur mais la simulation donnera quelque chose de bizarre
  - On VEUT que la simulation et la synthese concordent parce que la simulation nous aide a concevoir
  - Il faut donc que la liste de sensibilite soit COMPLETE.

Pourquoi les regles?

- Qu'arrive-t-il quand les possibilites ne sont pas toutes enumerees?

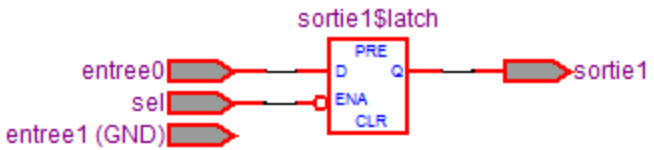
```
PROCESS (entree0, entree1, sel)
BEGIN
   IF sel = '0' THEN
       sortie1 <= entree0;
   ELSE
       sortie1 <= entree1;
   END IF;
END PROCESS;
```

Enlevons cette partie...

Pourquoi les regles?

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

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



Il va creer un element memoire... Au fond, c'est une bonne description pour une bascule D
Mais, ce n'est pas ce qu'on veut!

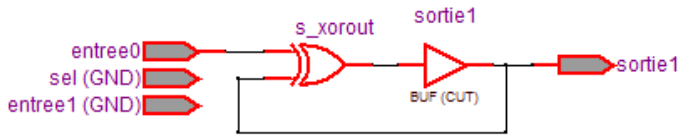
Pourquoi les regles?

- Pourquoi est-ce que les valeurs modifies ne doivent pas etre dans la liste de sensibilite?
  - Ca fait des “boucles combinatoires” et c’est normalement pas bon
- Si la sortie n’est pas utilisee, elle ne devrait pas etre dans la liste...
- Si elle est utilisee, on risque d’avoir des problemes...

Pourquoi les regles?

- Considerons 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 etre dans la liste de sensibilite
  - La sortie change et on est sensible a la sortie
  - Donc, le process recommence... Ca change la sortie
  - Le process recommence ENCORE.. Ca change la sortie...

C’est une boucle infinie...

Conception

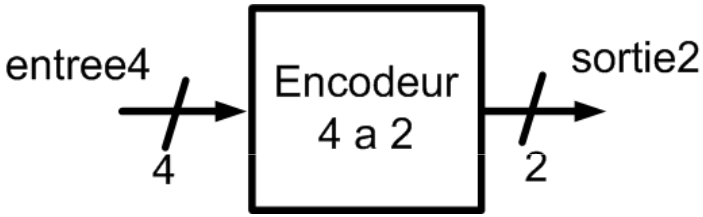
- Passons maintenant a la conception d'un module encodeur 4 a 2
- On va passer par toutes les etapes typiques
    - Definir l'entite
    - Definir l'architecture
- Essayez de comprendre le raisonnement

Exemple

- Concevez un encodeur de priorite 4 a 2 en VHDL...
  - On donne la plus grande priorite aux bit le plus significatif
  - Pas besoin d’ecrire les libraries pour cet exemple...

Exemple

- Je commence par dessiner l'exterieur
  - Ca m'aide a definir l'entite:




```
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

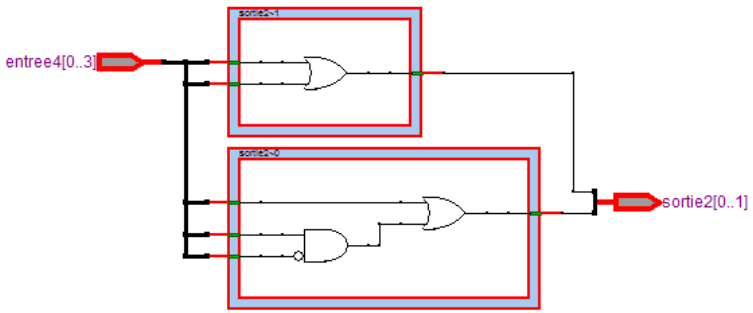
- Pour l'architecture, on devrait reflechir un peu
  - Si le bit le moins significatif, entree4(0), est 1, la sortie sera 00
  - Si le deuxieme bit, entree4(1), est 1, la sortie sera 01
  - Si le troisieme bit, entree4(2), est 1, la sortie sera 10
  - Si le bit le plus significatif, entree4(3), est 1,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

On nous avait dit qu'il y avait une priorite

Exemple

```
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 interessantes

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

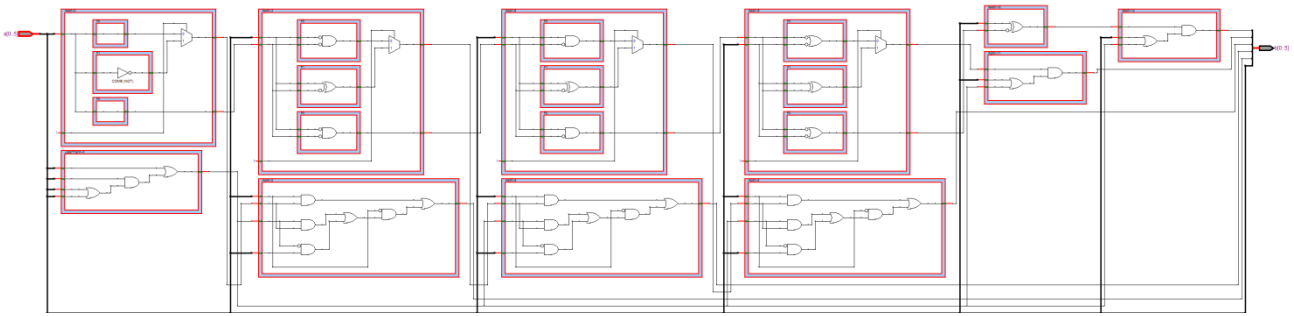
```
produit <= a * b;
```

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

Quelques notes interessantes

- Il est aussi possible de faire des comparaisons:
  - On peut utiliser =, > et < sans avoir a 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 interessantes

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

Quelques notes interessantes

- **LA** tache qui demarque un bon ingenieur c'est le deverminage:
  - Est-ce que vous etes capable de resoudre les problemes?
- Tout le monde est un bon ingenieur quand ca fonctionne bien
  - Un bon ingenieur sera capable faire des tests et trouver les erreurs quand ca va mal
  - Prenez le temps de developper vos propres techniques