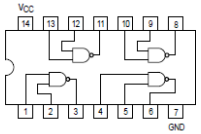
Programmation des systèmes numériques en Langage VHDL
1ère année 2011-2012

PRÉSENTATION

Problématique

- Un composant standard :
  - 6 à 14 broches environ (dont 2 alimentations)
  - Occupe plusieurs mm^2 voire de cm^2
  - Nécessite de nombreuses connexions



- Une réalisation en composants standards
  - Ne permet donc pas la réalisation de fonction très compliquée
  - Nécessiterait trop de connexions, de surface, …

Les composants configurables

- Utilisation de composants (re)configurables comportant des milliers de portes logiques
  - Fabriquants: Xilinx, Altera, Lattice
- Grand nombres de connexions internes possibles :
  - Par exemple par fusibles / ou par anti-fusibles
- Nécessité de générer un fichier configurant ces connexions par des moyens informatiques :
  - Un environnement de développement (*IDE*)
  - Un Langage standardisé : le VHDL

VHDL

- **V : *VHSIC* :** Very High Scale Integrated Circuit
- ***HDL* :** Hardware Description Language
- Langage de description de systèmes
- Adapté à la description, la simulation et la synthèse des systèmes numériques
- Extension du langage : *VHDL-AMS*.

  → Description de systèmes physiques (dont analogique, capteurs) et/ou numériques

VHDL

- Attention : **Description ≠ Réalisation**

  → Toute description VHDL n'est pas synthétisable
- Seul un sous-ensemble du Langage va permettre la synthèse
- VHDL permet de programmer des tests

  → Simulation

VHDL

- Langage normalisé
  - Normes en 1987, 1993, 2002, 2008
- Langage portable :
  - D'un IDE à un autre
  - D'un composant à un autre
- Un unique langage pour
  - Concevoir, simuler, synthétiser
- Un langage textuel
  - Écrire des fichiers de code

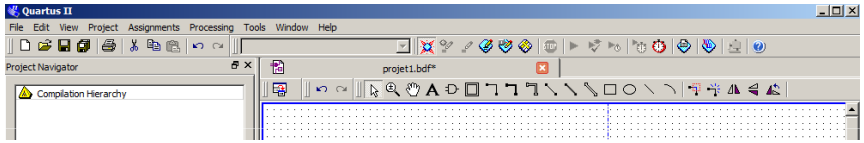
VHDL : objectifs à l'IOGS

- Commande et contrôle de systèmes
- Synchronisation de systèmes
- Mesure de durée, de fréquence
- Acquisition de grandeurs physiques et de signaux
- Traitement de signaux

- Les TP de 1A-2S
- Les projets de 1A-2S
- La 2A

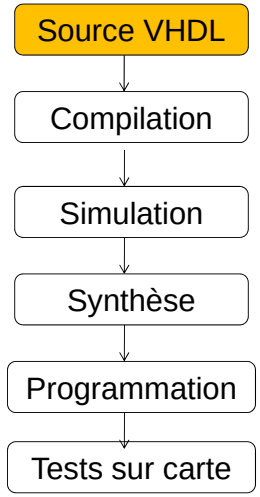
VHDL : contexte à l'IOGS

- IDE : Quartus d'Altera



- Carte : DE0 d'Altera

Cycle de conception en VHDL

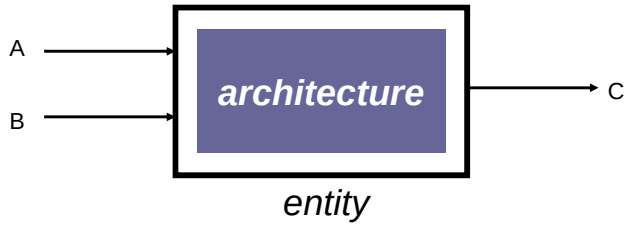


INSTITUT d'OPTIQUE GRADUATE SCHOOL

DESCRIPTION DU LANGAGE VHDL

Dualité Entité - Architecture

- Une fonction numérique est décrite :
  - ☐ Par ses Entrées/Sorties → **entity**
  - ☐ Puis par son comportement interne → **architecture**



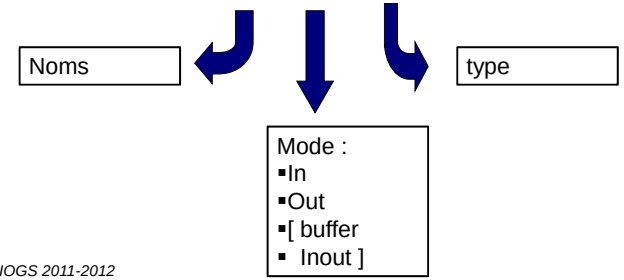
Dualité Entité - Architecture

- Une fonction numérique a un jeu fixé d'E/S :
  → Une seule entité pour une fonction donnée
  → Mais une ou plusieurs architectures

- De nombreuses façons de décrire la même architecture
  → De nombreux codes possibles
  (Comme pour la réalisation de fonctions avec des portes logiques)

Port d'une entité

- Port = accès électrique de la fonction : **in** ou **out**
  - Accessible parfois sur les broches du composant programmable
  - Ou uniquement à des fils internes au composant
  - *Exemple :* A, B : in std_logic ;

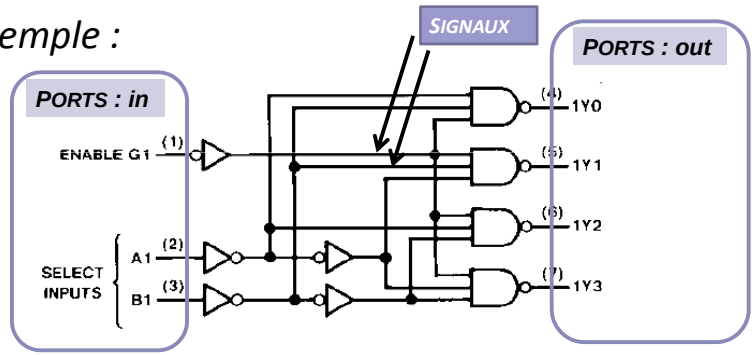


Types de base en VHDL

- Type pour un bit : ***std_logic***
  A,B : std_logic ;
- Type pour un mot de N bits : ***std_logic_vector***
  VAL : std_logic_vector(3 downto 0) ;
  ADC : std_logic_vector(1 to 8) ;
- *Nb : il a existé les types* bit *et* bit_vector *: obsolètes*
- ***Integer*** : type entier sur 32 bits (par défaut)
  N : integer ;
- Type ***Real*** : non synthétisable
- Type ***Enum :*** type énuméré (vu ultérieurement)

Objets de base à manipuler en VHDL

- *Exemple :*



- *Le codage de la fonction va remplacer le composant physique*

Objets de base à manipuler en VHDL

- ***Port*** : entrées – sorties
- ***Signal*** : connexion interne à une fonction : il s'agit de signaux physiques.

```
signal Somme : std_logic ;
Signal sortie : std_logic_vector(3 downto 0) ;
```

- ***Variable*** : variables de type informatique permettant la description des algorithmes

```
Variable temp : std_logic := '1' ;
```

- ***Constantes*** : Pour définir des valeurs permanentes

```
constant TAILLE_BUS : integer := 8 ;
constant ZERO: bit:= '0';
```


Objets de base à manipuler en VHDL

- Tous ces objets vont
  - ☐ Devoir être nommés et typés
  - ☐ Avec un des types définis en VHDL
- Noms : tous les caractères a, … , z, A , … , Z,0, …, 9, _
  Un nom doit commencer par une lettre;
  Pas de différence entre MAJUSCULE et minuscule.

Un exemple de programme Vhdl

```
Library ieee;       -- en-tête
USE ieee.std_logic_1164;
```

en-tête

```
Entity EXEMPLE is   --entité
port(
       A,B,C : in std_logic;
       S1, S2: out std_logic);
End EXEMPLE ;
```

entité

```
Architecture ARCHI of EXEMPLE is
Signal X : std_logic  --declaration
Begin                 -- début code
   X <= A and B;
   S2 <= (not A) or(A and B and C);
S1 <= X ;
End ARCHI ;     -- fin code
                -- ARCHI facultatif
```

architecture

Règles d 'écriture

- En-tête : nécessaire au bon fonctionnement
- Commentaire : `-- commentaire`
- Chaque ligne se termine par `;`
- Les objets manipulés sont <u>typés</u> et <u>nommés</u>
- L'Architecture contient :
  - ☐ Une partie déclarative
    - Déclaration de signaux, de variables nécessaires à l'algorithme
    - Mais pas de déclaration de ports : ils sont déjà déclarés dans l'entité
  - ☐ Puis le codage de la description (**begin … end;**)

Règles d'écriture

- Commentaires :
  - N'importe où
- Espaces
  - On en rajoute autant qu'on veut
- Saut de lignes
  - Pour améliorer la lisibilité (pas au milieu d'un nom)
- Majuscules, minuscules :
  - Pas de différence en VHDL
- Plusieurs entités-architectures dans un même fichier, plusieurs fichiers dans un projet vhdl

Instructions concurrentes

- Notion **Très** importante en VHDL
- Pour les lignes :

```
X <= A and B;
S2 <= (not A) or(A and B and C);
```

  - Pour ces 2 lignes : **Peu importe l'ordre**
    car chaque ligne va générer un bloc électronique
    ➔ 2 blocs séparés qui fonctionnent en parallèle
  - Les 2 instructions sont dites **concurrentes**

Affectations en VHDL

- ***A,B : in std_logic ;***

```
A <= '0' ;   -- A de type std_logic
VAL <= " 01011010"  ; -- VAL de type std_logic_vector
B <= C ; ;   --  B et C  de type std_logic
B <= VAL(2)  ;   -- B de type std_logic
B1 <= VAL(1)&VAL(2) ;  -- & : concaténation
        -- B1  de type std_logic_vector sur 2 bits
```

- ***Pour les variables*** : :=

  Mêmes types d'exemple

Affectations en VHDL : un piège

```
…
Port(   A,B,C : in std_logic;
        S1, S2: out std_logic);

Architecture ARCHI of EXEMPLE is
Begin                   -- début code
   S1 <= A and B;
   S2 <= (not S1) or(A and B and C);
End ARCHI ;     -- fin code
```

- Ce code va **poser problème** car S1 est une **sortie** et ne peut pas être utilisée comme entrée !

→ Solution : soit déclarer S1 comme *inout* ou comme *buffer*

→ Solution plus simple : déclarer un signal :

```
signal X : std_logic ;
X <= A and B;
S2 <= (not X) or(A and B and C);
S1 <= X ;
```

Opérateurs en VHDL

De la priorité <u>la plus basse à la plus haute</u> :

- ***Opérateurs logiques (5):*** and – nand – or – nor – xor
- ***Opérateurs relationnels (6) : = /= > >= <***
- ***Opérateurs d'addition(3) :*** + - & (concaténation)
  Ex : A(0)& A(1)
- ***Opérateurs de signe (2) :*** + -
- ***Opérateurs de multiplication (4) :*** * / rem(reste division entière) mod (modulo)
- ***Opérateurs de plus haut niveau (3) :*** not abs (valeur absolue) et ** (exponentiation)

Affectations conditionnelles en VHDL

- ***Structure de choix : When*** `expression` ***else***

```
Architecture ARCHI_CHOIX of CHOIX is
begin
  S <= A     when(SEL = "00") else
        B    when(SEL = "01") else
       "0100"  when(SEL = "10") else
       "1000";
End ;
```

- **La condition *(…)* peut être une expression logique**
- **Les expressions sont évaluées dans l'ordre**
- **On recommande le *else* final**

Affectations conditionnelles en VHDL

- ***Structure de choix : With*** `selection` ***select***

```
Architecture ARCHIT of VALEUR is
begin
   whith SEL select
        S <=    "0001" when "00",
                "0010" when "01",
                "0100" when "10",
                "1000" when others;  --indispensable
End ARCHIT;
```

- **A gauche du *when* on peut avoir une expression.**
- **A droite du when on a forcément une constante**
- **On recommande le *when others* final**
- **Adapté au codage diret d'une table de vérité**

Notion de Process

- Le Process est une instruction concurrente
- Il a une liste de sensibilité :
  - ☐ Liste des signaux dont les variations vont activer le process

```
process (A,B,C,D,SEL)
-- partie déclarative (si besoin)
begin
   if SEL = …
-- suite du code
end;
```

Le process ne sera exécuté que si un des signaux A,B,C, D ou SEL évolue.

Affectations conditionnelles dans les Process

- Dans les process les instructions de choix ont une syntaxe différente.
- *Exemple : le **if then else*** ou ***if then elsif***

```
process (A,B,C,D,SEL)
 begin
     if    SEL = "00" then   S <= A ;
     elsif SEL = "01" then   S <= B ;
     elsif SEL = "10" then   S <= C ;
      else                   S <= D ;
   end if;
 end process;
```


Affectations conditionnelles dans les Process

- *Exemple : le **case … is … when***

```
process (A,B,C,D,SEL)
      begin
          case SEL is
             when "00" =>   S <= A ;
             when "01" =>   S <= B ;
             when "10" =>   S <= C ;
             when others => S <= D ;
          end case;
      end process;
```

- *Alternatives exclusives. Toutes doivent être traitées dans le même **case***

Exemples: divers codages d'un Mux 2>1

```
In : E0,E1,SEL : std_logic
Out : S : std_logic
---------------------------------
S <= E0 and not(SEL) or E1 and SEL
---------------------------------
S <= E1 when (SEL = '1') else
     E0 ;
---------------------------------
With SEL select
 S <= E0 when '0'
 S <= E1 when others ;
---------------------------------
Process(E0,E1,SEL)
Begin
  if (SEL='0') then S <= E0 ;
  else S <= E1 ;
```

```
-----------------------
Process(E0,E1,SEL)
Begin
   case SEL is
       when '0'=> S<= E0
       when '1'=> S<= E1
   end case ;
End process ;
```


Exemples : codage d'une fonction logique

```
In : A,B,C : std_logic
Out : S : std_logic
…
S <=   1 when " 000 " else
       0 when "001" else
       1 when "010" else
       1 when "011" else
       0 when "100" else
       0 when "101" else
       0 when "110" else
       1 when "111" ;
```

Les types de Process

- 3 types de Process:
  - ☐ Process purement combinatoires
    - Process(A,B,C,D)
    - On pourrait aussi les écrire sans utiliser de process
  - ☐ Process clocké
    - Process(CLK)
  - ☐ Process clocké avec RAZ (ou RAU) asynchrone
    - Process(RAZ,RAU,CLK)

Les types de Process

- Process **à proscrire**:
  - ☐ Mélange de combinatoire et de clocké
    - Process(A,B,C,D,CLK)
- Bonne règle de codage de Process clocké avec RAZ (ou RAU) asynchrone

```
process (CLK,RAZ)
  begin
    -- ici traiter toutes les Initialisations
    -- puis traiter le front d'horloge
    -- les choix se font avec les instructions de choix
  adéquates
  end process;
```


Les mots réservés du langage

abs	component	generic	next	record	use
access	configuration	guarded	nor	register	
after	constant		not	rem	variable
alias		if	null	report	
all	disconnect	in		return	wait
and	downto	inout	of		when
architecture		is	on	select	while
array	else		open	severity	with
assert	elsif	label	or	signal	
attribute	end	library	others	subtype	xor
	entity	linkage	out		
begin	exit	loop		then	
block			package	to	
body	file	map	port	transport	
buffer	for	mod	procedure	type	
bus	function		process		
		nand		units	
case	generate	new	range	until	

Les types de programmation en VHDL

- VHDL est un langage de description
- 3 types de descriptions :
  - ☐ Par flots de données : — Déjà traité
    - Association de portes logiques
    - `S <= A and B and (not C);`
  - ☐ Comportementale : — En partie traité
    - Par description d'un algorithme
    - Utilisation de when …, de process()
  - ☐ Structurelle : — Non traité
    - Par association de composants (components) et instanciation