# Le VHDL

VHSIC (Very High Speed Integrated Circuit) Hardware Description Language,

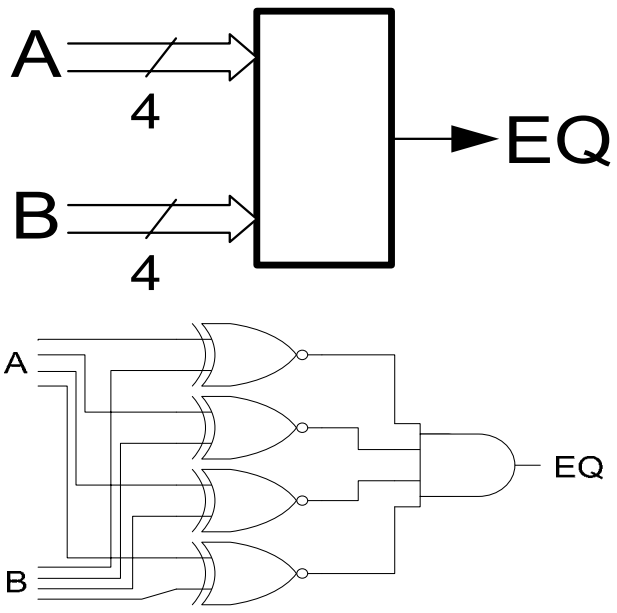
*(Ou encore V.H.S.I.C.H.D.L. :)*

# Plan de la présentation

- Objets et opérateurs de base
- Logique concurrente VS synchrone
- Les « Entity »
- Les « Component »
- Les « Package »
- Les « Block »
- Les « Test benches »
- Conclusion

# Analogie avec circuits logiques réels

- Exemple: Comparateur 4 bits




```
entity comparateur is
    port( A, B: in bit_vector(0 to 3);
          EQ: out bit);
end comparateur;

architecture arch of comparateur is
begin

    EQ <= '1' when (A = B) else '0';

end arch;
```

# Objets en VHDL

- Les différentes sortes d'objets
  - Signal : Fil conducteur.
  - Variable : Valeur intermédiaire d'une opération complexe.
  - Constant : Valeur fixe.
- Possèdent tous un type
  - bit
  - std_logic
  - std_logic_vector
  - …

Exemples de déclaration:

```
constant SRAM: bit_vector(15 downto 0) := X"F0F0";
signal Q: std_logic;
```

# Opérateurs

- Vaste choix d'opérateurs
  - Logiques: and, or, nand, nor, xor, etc.
  - Tests : =, /=, <, >, <=, etc.
  - Arithmétiques: +, -, /, *
  - Attributs : clock'event, clock'last_value, etc.

  Exemple, pour détecter un front montant :

  **if** Clock = '1' **and** Clock**'event then** …

# Logique concurrente et synchrone

- En VHDL, on peut définir des opérations concurrentes ou synchrones.
- Hors d'un bloc *Process*, la logique est toujours concurrente.
- À l'intérieur d'un bloc *Process*, la logique est synchrone.
- On peut également « pipeliner » les signaux dans un *Process*.

# Exemple de logique concurrente

```
signal A, B, C: std_logic_vector(7 downto 0);
signal Select : std_logic;
constant Init: std_logic_vector(7 downto 0) :=
   "01010101";
…
A <= B;
B <= Init when Select = '1' else C;
C <= A and B; <- Il faut additionner les délais
```

# Exemple de logique synchrone

```
Signal OutOnes, OutOnesLater : std_logic_vector(0 to 9);

process(Rst, Clk)        <- Liste de sensibilité
begin
    if Rst = '1' then      <- Aussitôt que Rst égale 1 (non synchrone)
            OutOnes <= (others => '0');
             OutOnesLater <= (others => '0');
    elsif rising_edge(Clk) then  <- Quand l'horloge est en montée
            OutOnes <= "111111111"; <- Après 1 clock
            OutOnesLater <= OutOnes ; <- Après 2 clock
    end if;
end process;
```

# Les entités (entity)

• Qu'est-ce que c'est ? Un tout, c'est-à-dire un bloc de code indépendant avec ses entrées et sorties

→Analogie avec C/ C++ : Une fonction

• Une entité est divisée en trois (3) parties :

• Déclaration de l'entité

```
Entity <Nom de l'entité> is
Port (
    <Signaux in et out>
  );
End <Nom de l'entité>
```

• Déclaration des constantes, signaux internes et components

```
Architecture Structural
<Nom de l'entité> is

<Constantes>
<Signaux internes>
<Components>
```

• Corps de l'entité : code utile

```
Begin
   <Code utile>
End Structural
```

**Règle générale : Une entité par fichier VHDL**

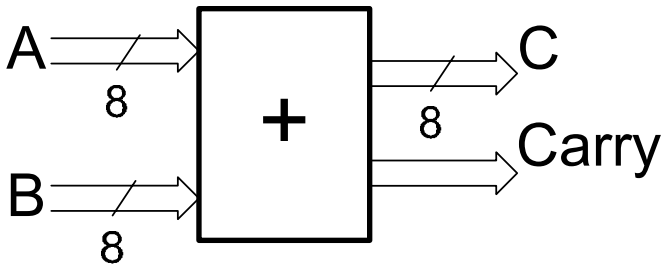
# Exemple d'entité : additionneur 8 bits

```
Entity Adder8bits is
  Port (
    clk:   in  std_logic;
    rst:   in  std_logic;
    A:     in  std_logic_vector(7 downto 0);
    B:     in  std_logic_vector(7 downto 0);
    C:     out std_logic_vector(7 downto 0);
    Carry: out std_logic
  );
End Adder8bits

Architecture behavioral of Adder8bits is

Signal AplusB: std_logic_vector(8 downto 0);

Begin
  Additiop : Process (rst, clk)
  Begin
    If (rst = '1') then
      Carry <= '0';
      C <= "00000000";
      AplusB <= (Others => '0');
    Elsif (clk'event and clk = '1') then
      AplusB <= (A & '0') + (B & '0');
      C <= AplusB(7 downto 0);
      Carry <= AplusB(8)
    End
  End process
End structural
```



# Les components

- But : Modularisation du code de façon hiérarchique

  →Analogie avec C/ C++ : Appel de fonction

- Méthode pour inclure d'autres entités dans une entité principale
  - Déclaration du component
  - Instanciation du component : « Port map »
- Niveaux d'instanciations infinis !

**Déclaration :**

```
Component <Nom du
   component>
Port (
    <Signaux in et
   out>
  );
End Component;
```

**Instanciation :**

```
<Nom de l'instance> : <Nom du
   component>
Port map (
    Signal1 => SignalInterne1,
    Signal2 => SignalInterne2
  );
```

# Exemple avec component : additionneur 16 bits

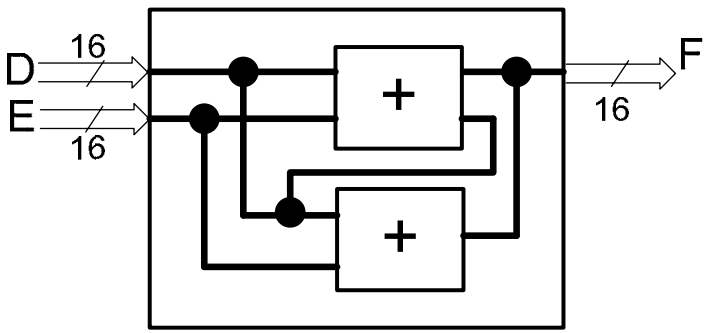
```
Entity Adder16bits is
  Port (
    D: in  std_logic_vector(15 downto 0);
    E: in  std_logic_vector(15 downto 0);
    F: out std_logic_vector(15 downto 0)
);
End Adder16bits;

Architecture Structural of Adder16bits is

Signal Carry:   std_logic;
Signal Dplus:   std_logic_vector(7 downto 0);

Component Adder8bits
 Port (
    A:     in  std_logic_vector(7 downto 0);
    B:     in  std_logic_vector(7 downto 0);
    C:     out std_logic_vector(7 downto 0);
    Carry: out std_logic
  );
End component;

Begin
  Adder1 : Adder8bits
  Port map (
    A     => D(7 downto 0),
    B     => E(7 downto 0),
    C     => F(7 downto 0),
    Carry => Carry
  );
```




```
Dplus <= D(15 downto 8) +
    ("0000000" & Carry);

Adder2 : Adder8bits
  Port map (
    A     => Dplus,
    B     => E(15 downto 8),
    C     => F(15 downto 8),
    Carry => Open
  );
End structural
```

# Les packages

- Les packages sont des librairies de code VHDL précompilées ou non

  →Analogie avec C/ C++ : #include “package.h”

- Les packages sont généralement déclarés au tous début des fichiers de code source VHDL (avant les entités)
- Employés à plusieurs sauces, par exemple :
  - Modularisation du code
  - Simulation de composantes numériques externes
  - Librairies nécessaires par les simulateurs employés

```
Package <Nom du package> is
  <Constantes>
  <Signaux>
  <Components>
  <Code utile, etc.>
End <Nom du package>
```

```
use work.<Nom du package>.all;
```

# Les blocks

• Ce que c'est : Une sous unité d'une entitée

→ C'est un mélange de process et de component : un process avec des ports d'entrée et de sortie

•Rarement utilisés en pratique : alourdi le code. On préconise plutôt l'utilisation des components

```
Entity <Nom de l'entité> is
  ...
Begin
  <Nom du bloc> : block
    port (
      <Signaux in et out>
    );
    port map (
      Signal1 => SignalInterne1,
      Signal2 => SignalInterne2
    );
  declarations for <Nom du block>
  begin
    <Code utile>
  End block <Nom du bloc>;
End Structural;
```

# Test bench et exemple

- Test bench : Librairies et entité « vide »
- Déclaration des components à simuler et signaux à monitorer
  - Attention : Init. les signaux
- Opérateurs Wait et After

```
library ieee;
use ieee.std_logic_1164.all;

Entity TestBench is
End TestBench;

Architecture Behavioral of TestBench is

Signal A: std_logic_vector(7 downto 0);
Signal Clk: std_logic := '0';

<Déclarer component à simuler>

Begin

Clk <= not(clk) after 10 ns;

Process(clk)
  Wait for 10 ns;
  A <= "10100101";
  Wait;
End process;

<Port map du component à simuler>

End Behavioral
```

# Conclusion

- Ceci n'était qu'un survol du VHDL. Plusieurs points n'ont pas été couverts :
  - Generic
  - Simulation : ModelSim, Matlab-Simulink
  - …
- Tutorial intéressant pour se familiariser avec le VHDL :

  http://csold.cs.ucr.edu/content/esd/labs/tutorial/
- Référence :
  http://www.peakfpga.com/vhdlref/index.html

# Questions