Electronique numérique 3 : Introduction & VHDL

Patrick.Garda@upmc.fr

I. Architectures comportementales en VHDL

Patrick.Garda@upmc.fr

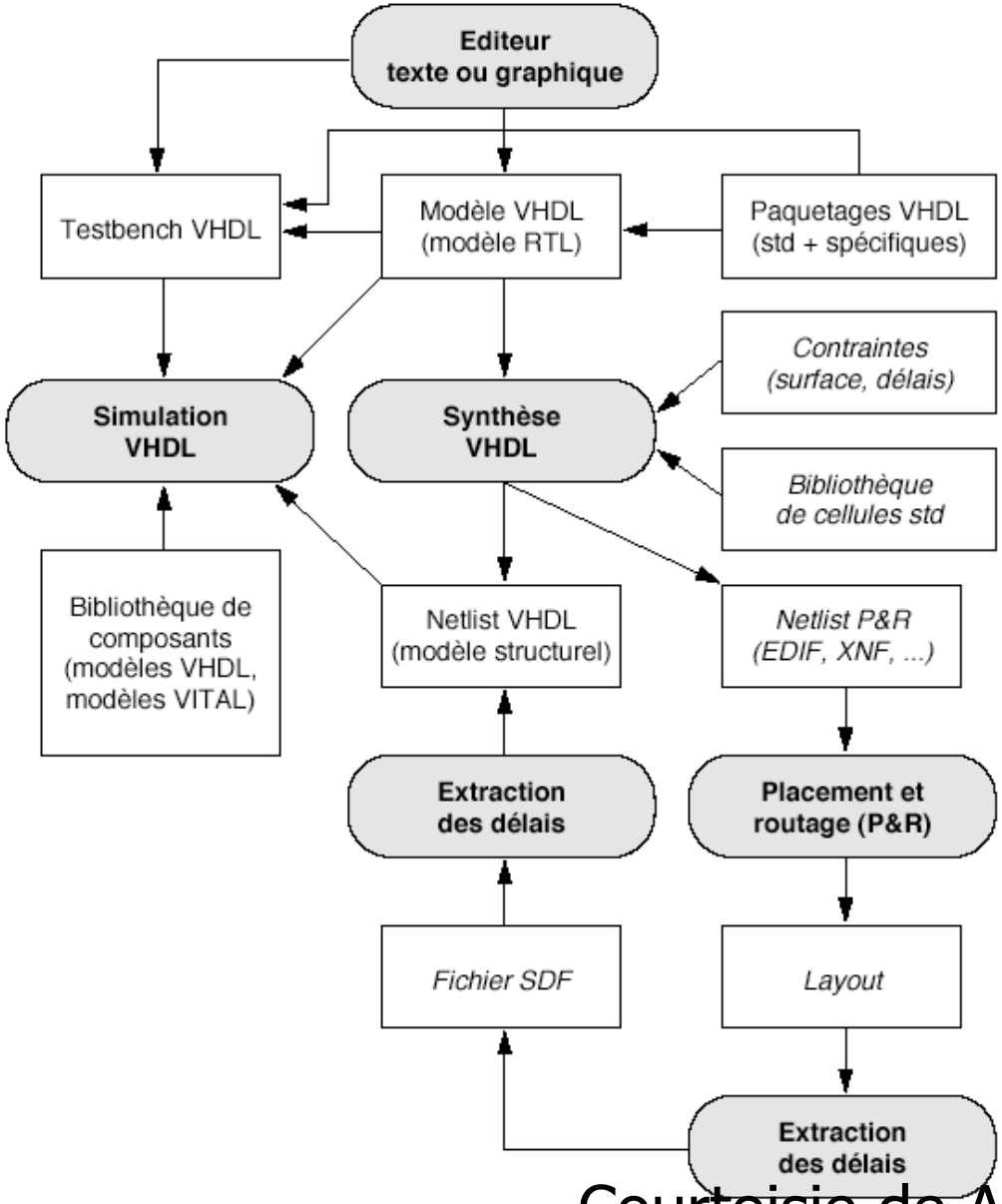
1. VHDL : Documentation

- The designer's guide to VHDL
  - Peter Ashenden, Morgan Kaufman
  - http://www.ashenden.com.au/designers-guide/DG.html
- Conception de circuits VLSI
  - F.Anceau, Y.Bonnassieux, Dunod, 2007
- Alain Vachoux : Instant VHDL
  - http://lsm.epfl.ch/webdav/site/lsm/users/104020/public/VHDL_instant_v2.1.pdf

2. Logiciels & cartes ALTERA

- Free Quartus II Web Edition : http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html
- Free Modelsim Altera Starter Software: https://www.altera.com/download/software/modelsim-starter
- Carte DE-2 : http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html?GSA_pos=1&WT.oss_r=1&WT.oss=DE-2

3. Flot de conception VHDL



Courtoisie de Alain Vachoux, EPFL

4. Description d'un circuit électronique

- Description circuit électronique en 2 parties :
  - Interface unique : nom, entrées et sorties
  - Une (au moins) ou plusieurs architectures
- Architectures
  - Structurelle
  - Comportementale
  - RTL : Register Transfer Level

5. Pourquoi architecture comportementale ?

- Pour décrire l'architecture d'un circuit sans décrire l'électronique qui le compose.
- Gain :
  - Plus rapide à développer.
  - Plus rapide à simuler.
- Intéressant pour :
  - La spécification.
  - La modélisation.

Spécificités de l'électronique

- Quand un circuit est mis sous tension, toutes ses portes fonctionnent simultanément.
- Les circuits ont souvent plusieurs sorties, dont les valeurs sont calculées en même temps.

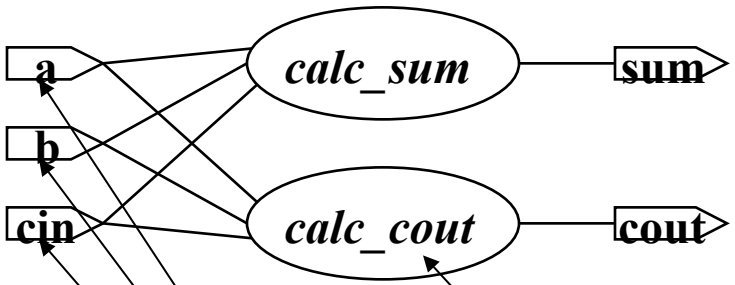
6. Architecture comportementale

- L'architecture est composée d'un ensemble de processus.
  - Elle ne décrit pas l'électronique du circuit.
- Chaque processus exécute un programme.
  - Parce que les programmes permettent d'exprimer tous les calculs des valeurs des sorties en fonction des entrées.
- L'architecture est composée d'un ensemble de processus qui s'exécutent simultanément.
  - Pour prendre en compte les spécificités de l'électronique :
    - Calculs de plusieurs sorties en même temps.
    - Tous les éléments du circuit sont actifs simultanément.

7. Processus

- Chaque processus possède une liste de sensibilité :
  - Il s'agit d'un ensemble de signaux qui sont les entrées du processus.
- Par défaut le processus est en veille : il attend que l'un des signaux de sa liste de sensibilité change de valeur.
- Lorsque cela se produit, le corps du processus est exécuté séquentiellement.
- Le processus ne possède pas de signaux internes.
- Pour nommer le processus, on peut lui attribuer une étiquette.
- La réalisation électronique du processus n'est pas décrite.

Exemple : Architecture comportementale addc




```
architecture behaviour of addc is
    begin
        calc_sum : process (a, b, cin) is
            begin
            …
            end process calc_sum ;

        calc_cout :  process (a, b, cin) is
            begin
            …
            end process calc_cout ;
end architecture behaviour ;
```

Etiquette

Liste de sensibilité

8. Instructions séquentielles

- Elles sont utilisées pour :
  - Ecrire le corps des processus
- Problème :
  - VHDL initialement défini pour la simulation !
  - Sous-ensemble de VHDL synthétisable : IEEE 1076.6
  - Paquetage pour la synthèse : IEEE 1076.3
- Instructions séquentielles qui peuvent être synthétisées :
  - Conditionnelle if
  - Branchement case
  - Boucle for
- Instructions dont la synthèse n'est pas assurée :
  - Boucle while
  - Boucle loop

9. Instruction if

```
if condition1 then
    --- instructions séquentielles 1
    [elsif condition2 then
    --- instructions séquentielles 2]
    [elsif condition3 then
    --- instructions séquentielles 3]
    [else
    --- instructions séquentielles n]
    end if ;
```

- Les expressions entre [] sont optionnelles.
- Les conditions :
  - Condition1
  - Condition2
  - Condition3
  - Sont booléennes.

Exemple de if : multiplexeur 2 vers 1

```
entity mux21 is
    port (
        y          : out std_logic ;
        a, b       : in std_logic ;
        S          : in std_logic ;
        ) ;
    end entity mux21 ;
```

```
architecture behaviour of mux21
    is begin
      process (s, a, b) is
      begin
      if s = '0' then
            y <= a after 1 ns ;
            else  y <= b after 1 ns  ;
            end if ;
            end process ;
            end architecture behaviour ;
```

10. Instruction case

```
case expression is
    when VALUE-1 =>
        -- instructions séquentielles 1
    when VALUE-2 | VALUE-3  =>
        -- instructions séquentielles 2
    when VALUE-M to VALUE-N  =>
        -- instructions séquentielles 3
    when VALUE-S downto VALUE-R  =>
        -- instructions séquentielles 3
    when others =>
        -- instructions séquentielles n
    end case ;
```

- Restrictions :
  - Chaque valeur du sélecteur doit correspondre à une clause et une seule.
  - Les clauses Intervalle ne sont possibles que pour des expressions scalaires.

Exemple de case : multiplexeur 4 vers 1

```
entity mux41 is
    port (
      y              : out std_logic ;
      a, b, c, d     : in std_logic ;
      sel            : in std_logic _vector (0 to 1) ;
      ) ;
    end entity mux41 ;
```

Architecture multiplexeur 4 vers 1

```
architecture behaviour of mux41 is begin
    process (sel, a, b, c, d) is begin
    case sel is
        when "00"       =>  y <= a after 5 ns ;
        when "01"       =>  y <= b after 5 ns  ;
        when "10"       =>  y <= c after 5 ns ;
        when others     =>  y <= d after 5 ns ;
        end case ;
        end process ;
        end architecture behaviour ;
```

Multiplexeur 4 vers 1 avec 2 bits de sélection

```
entity mux41 is
    port (
        y                : out std_logic ;
        a, b, c, d       : in std_logic ;
        sel1, sel0       : in std_logic ;
        ) ;
    end entity mux41 ;
```

```
architecture behaviour of mux41 is begin
    process (sel1,sel0, a, b, c, d) is begin
    case sel1&sel0 is
        when "00"    =>  y <= a after 5 ns ;
        when "01"    =>  y <= b after 5 ns  ;
        when "10"    =>  y <= c after 5 ns ;
        when others =>  y <= d after 5 ns ;
        end case ;
        end process ;
        end architecture behaviour ;
```

Architecture avec if imbriqués

```
entity mux41 is
    port (
        y                : out std_logic ;
        a, b, c, d       : in std_logic ;
        sel1, sel0       : in std_logic ;
        ) ;
    end entity mux41 ;
```

```
architecture behaviour of mux41 is begin
    process (sel1,sel0, a, b, c, d) is begin
        if sel1 = '0' then
            if sel0 = '0' then
                y <= a after 1 ns ;
                else  y <= b after 1 ns  ;
                end if ;
        else
            if sel0 = '0' then
                y <= c after 1 ns ;
                else  y <= d after 1 ns  ;
                end if ;
        end if ;
    end process ;
    end architecture behaviour ;
```

11. Instruction de boucle for

- Syntaxe

[**loop_label**]
for **identifier** in **discrete_range** loop
    -- **instructions du corps de boucle**
    end loop **[loop_label]** ;

- Identificateur de boucle : **identifier**
  - pas déclaré
  - ne peut pas être modifié
  - accessible seulement dans le corps de la boucle

Exemple de boucle for

```
entity mux821 is
    port (
     y     : out std_logic_vector(0 to 7) ;
     a, b : in std_logic_vector(0 to 7) ;
     S     : in std_logic ;
     )
     end entity mux821 ;
```

```
architecture behaviour of mux821 is
    begin
      process (s, a, b) is
          begin
          for i in 0 to 7 loop
            case s is
              when '0' =>
                        y(i) <= a(i) after 1 ns ;
              when others =>
                        y(i) <= b(i) after 1 ns ;
              end case ;
            end loop ;
      end process ;
    end architecture behaviour ;
```

12. Variables

- Définition
  - Les variables sont utilisées dans les instructions séquentielles
  - Elles ne correspondent à aucune réalité électronique
  - Elles jouent le rôle des variables dans un langage de programmation
  - Elles ont une portée limitée au processus dans lequel elles sont déclarées
  - Elles sont typées
- Affectation
  - Symbole d'affectation de variables :=
  - L'affectation d'une variable est instantanée
- Utilisation
  - Affectation d'un signal à une variable
  - Exécution d'un algorithme séquentiel
  - Affectation d'une variable à un signal
- Autre utilisation : identificateur de boucle

Exemple de variable

```
library ieee ;
use ieee.std_logic_1164.all ;
entity comp_64 is
    port (
          y : out std_logic ;
          a : in std_logic_vector (63 downto 0);
          b : in std_logic_vector (63 downto 0) ;
          ) ;
    end entity  comp_64;
```

Exemple de variable : suite

```
architecture behaviour of comp_64 is
    begin process (a, b)
        variable z : std_logic ;
        begin
        z := '1' ;
        for i in 63 downto 0 loop
            z := z and (a(i) xnor b(i)) ;
            end loop ;
        y <= z after 50 ns ;
        end process;
end architecture behaviour ;
```

13. Commentaires sur les architectures

- La description comportementale est propre à VHDL.
- Elle permet en particulier de décrire la spécification du circuit
  - c'est-à-dire de décrire sa fonction sans décrire sa structure :
    - Ni à l'aide de composants de bibliothèque
    - Ni à l'aide de portes.
- Le passage de la description comportementale à la description structurelle avec des composants de bibliothèque est la synthèse.
  - Elle est actuellement semi-automatique

II. Bascules

Patrick.Garda@upmc.fr

1. Circuits combinatoires et séquentiels

- Circuit combinatoire
  - Il n'y a pas de boucle dans le circuit
  - L'état des sorties ne dépend que de l'état présent des entrées
- Circuit séquentiel
  - Il y a une boucle dans le circuit
  - Les sorties dépendent des entrées et des valeurs précédentes des sorties
  - Les sorties dépendent des entrées et des valeurs précédentes des entrées
  - Le circuit contient des éléments de mémorisation

2. Circuits asynchrones ou synchrones

- Circuits asynchrones
  - La propagation des informations dans le circuit est asynchrone
  - Les éléments de mémorisation peuvent changer d'état à n'importe quel instant
  - + : basse consommation, protection des informations
  - - : conception difficile

- Circuits synchrones
  - Une horloge est
    - un signal périodique diffusé dans tout le circuit
    - Tel que les éléments de mémorisation ne peuvent changer d'état qu'à des instants définis par l'horloge
    - Ilo ne peut prendre que les valeurs '0' et '1'
  - Trois possibilités pour définir ces instants de changement d'état :
    - Uniquement quand front montant sur l'horloge
    - Uniquement quand front descendant sur horloge
    - (Uniquement quand horloge au niveau haut : très difficile)

3. Circuits synchrones

- VHDL ne possède pas de notion d'horloge
- Pour décrire une horloge il faut utiliser au choix :
  - **1 instruction qui détecte un front**
  - 1 composant de bibliothèque
- Détection d'un front montant :
  - clock = '1' and clock'event
  - Dans library ieee : rising_edge (clock)
- Détection d'un front descendant :
  - clock = ’0' and clock'event
  - Dans library ieee : falling_edge (clock)

- VHDL ne possède pas de notion de registre ou de bascule
- Pour placer une bascule dans le circuit il faut utiliser au choix :
  - **1 description comportementale que VHDL interprète comme une bascule**
  - 1 bascule de bibliothèque

4. Bascule D

```
library ieee;
Use ieee.std_logic_1164.all;
Constant T_PD : time := 5 ns ;
entity dff is
    port (
        q           : out std_logic ;
        d           : in std_logic ;
        clock       : in std_logic ;
        ) ;
    end entity dff ;
```

```
architecture behaviour of dff
    is
    begin process (clock) is
        begin
        if clock = '1' and clock'event
            then
                q <= d after T_PD ;
            end if ;
        end process;
    end architecture behaviour ;
```

5. Bascule D avec reset synchrone

```
library ieee;
use ieee.std_logic_1164.all;
Constant T_PD : time := 5 ns ;
    entity dff_syn is
     port (
        q                    : out std_logic ;
        d                    : in std_logic ;
        clock, reset_n       : in std_logic ;
        ) ;
     end entity dff_syn ;
```

```
architecture behaviour of dff_syn
    is
     begin
          process (clock) is
          begin
          if clock = '1' and clock'event then
                    if reset_n = ’0'
                      then q <= '0’ after T_PD ;
                       else q <= d after T_PD ;
                       end if;
             end if ;
          end process;
    end architecture behaviour ;
```

6. Bascule D avec reset asynchrone

```
library ieee;
use ieee.std_logic_1164.all;
Constant T_PD : time := 5 ns ;
    entity dff_asyn is
    port (
        q                   : out std_logic ;
        d                   : in std_logic ;
        clock, reset_n      : in std_logic ;
        ) ;
     end entity dff_asyn ;
```

```
architecture behaviour of
   dff_asyn is
    begin process (clock, reset_n) is
        begin
        if (reset_n = ’0')
           then q <= '0’ after T_PD ;
        elsif clock = '1' and clock'event
           then q <= d after T_PD ;
        end if ;
        end process;
    end architecture behaviour ;
```

7. Note sur la synthèse

- **La synthèse d'une instruction if incomplète**

```
If condition
    then s <= d ;
    end if ;
```

- **Produit la génération d'un point mémoire.**

- **La synthèse d'une instruction if complète**

```
If condition
    then s <= d ;
    else s <= e ;
    end if ;
```

- **Produit la génération d'un multiplexeur 2 vers 1.**

8. Essentiel

- Dans l'écriture des instructions conditionnelles if et case
- Il faut veiller à ce que :
  - Les points mémoires soient présents :
    - Dans les mémoires.
    - Dans les registres.
  - Il n'y ait aucun point mémoire dans les circuits combinatoires.

9. Circuits combinatoires en VHDL

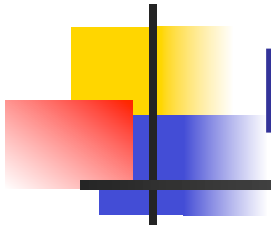
- Dans le corps de l'architecture comportementale d'un circuit combinatoire, la liste de sensibilité du process doit inclure toutes les entrées du circuit.
- Exemple addc : a, b, cin

10. Circuits séquentiels synchrones en VHDL

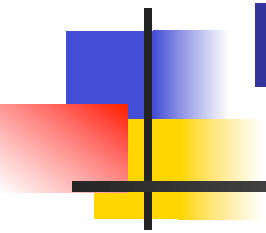
- Dans le corps de l'architecture comportementale d'un circuit séquentiel, la liste de sensibilité du process doit être :
  - SOIT (clock)
  - SOIT (clock, reset_n)
- Aucune autre liste de sensibilité ne conduira à un circuit séquentiel correctement synthétisé

11. Conception synchrone

- Pour éviter les erreurs de conception, il faut respecter les règles de conception synchrone (synchronous design) :
  - 1 seule horloge et un seul type de front de cette horloge
  - Aucune porte logique sur le chemin de propagation du signal d'horloge.
  - Cette horloge provoque le déclenchement de toutes les bascules.
  - Uniquement des bascules D.
  - Toutes les bascules sont déclenchées par cette horloge et par aucun autre signal.
    - Pas de signaux de données pour contrôler le déclenchement d'une bascule.



Exercice



III. Signaux

Patrick.Garda@upmc.fr

1. Signaux

- Les signaux représentent les données physiques échangées entre les modules (anglais data signal)
- Chaque signal sera matérialisé dans le circuit final par une équipotentielle
- Exemples :
  - entrées et sorties de addc : a, b, cin, sum, cout
  - signaux internes à structure : s1, c1, c2

- Instruction d'affectation de signaux :
  - s <= d after delay ;
  - s est le signal
  - d est le driver
  - delay est le délai
- Affectation de variables
  - :=

2. Type d'un signal

- Chaque signal a un type, comme dans un langage structuré
- Le type définit l'ensemble des valeurs que peut prendre le signal.
- Les signaux de type synthétisable sont synthétisables.

- Déclaration du type d'un signal
  - soit section "signal" d'une architecture
  - soit section "port" d'une entité
  - les types des signaux d'une déclaration d'entité ou de composant et d'une instanciation doivent se correspondre
  - le signal et le driver d'une affectation de signal doivent avoir le même type ou des types compatibles

3. Différents types de signaux

- Les types peuvent être définis pas l'utilisateur ou prédéfinis.
- Deux paquetages de types prédéfinis :
  - standard ou std, norme IEEE 1076 de 1987.
  - IEEE, norme IEEE 1164 de 1992, usage recommandé.

4. Types prédéfinis de std

- Types énumérés
  - type boolean is (false, true) ;
  - type character is (liste_des_caractères) ;
    - Tous les caractères ISO 8 bits
    - Une constante caractère est notée 'A'
  - type bit is ('0', '1') ;
  - type severity_level is (note, warning, error, failure) ;
- Types physiques
  - type time is range $- to $+
    units fs ;
    ps = 1000 fs ;
    ns = 1000 ps ;
    us = 1000 ns ;
    ms = 1000 us ;
    sec = 1000 ms ;
    min = 60 sec ;
    hr = 60 min ;
    end units ;

5. Types scalaires prédéfinis de std

- Entiers :
  - type integer is range -2_147_483 to 2_147_482 ;
  - subtype positive is integer range 1 to integer'high ;
  - subtype natural is integer range 0 to integer'high ;
- Remarques sur entiers :
  - Valeurs négatives codées en complément à 2
  - Entiers synthétisés comme bus 32 bits par défaut
- Intervalles :
  - Subtype byte is integer range -128 to 127 -- codé 8 bits C2
- Réels :
  - type real is range $- to $+
  - Réels pas synthétisables

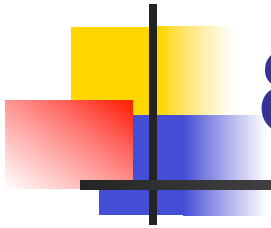
6. Constantes scalaires

- Déclaration de constantes
  - Constant length : integer := 32 ;
- Constantes entières :
  10 123_456 98E+3
- 253 dans une autre base :
  - 2#1111_1101#
  - 8#0375#
  - 16#FD#
  - 16#0fd#
- 1024 dans une autre base :
  - 2#1#E10
  - 16#4#E2
  - 10#1024#E+00
- Constantes réelles :
  1.0 7.456_23 9.8E+4
- 0.5 dans une autre base :
  - 2#0.100#
  - 8#0.4#
  - 12#0.6#
- Remarque :
  - La base est toujours écrite en décimal.
  - Les constantes de type synthétisable sont synthétisables.

7. Opérateurs

- Différentes classes d'opérateurs scalaires :
- arithmétiques :
  - + - * /
  - ** : exponentiation
  - mod : quotient division euclidienne
  - rem : reste division euclidienne
  - Abs : valeur absolue
- comparaisons :
  - < <= > >=
  - Egalité :
    - VHDL =
    - C = =
    - /= (différent)
- logiques :
  - not and or nand nor xor xnor

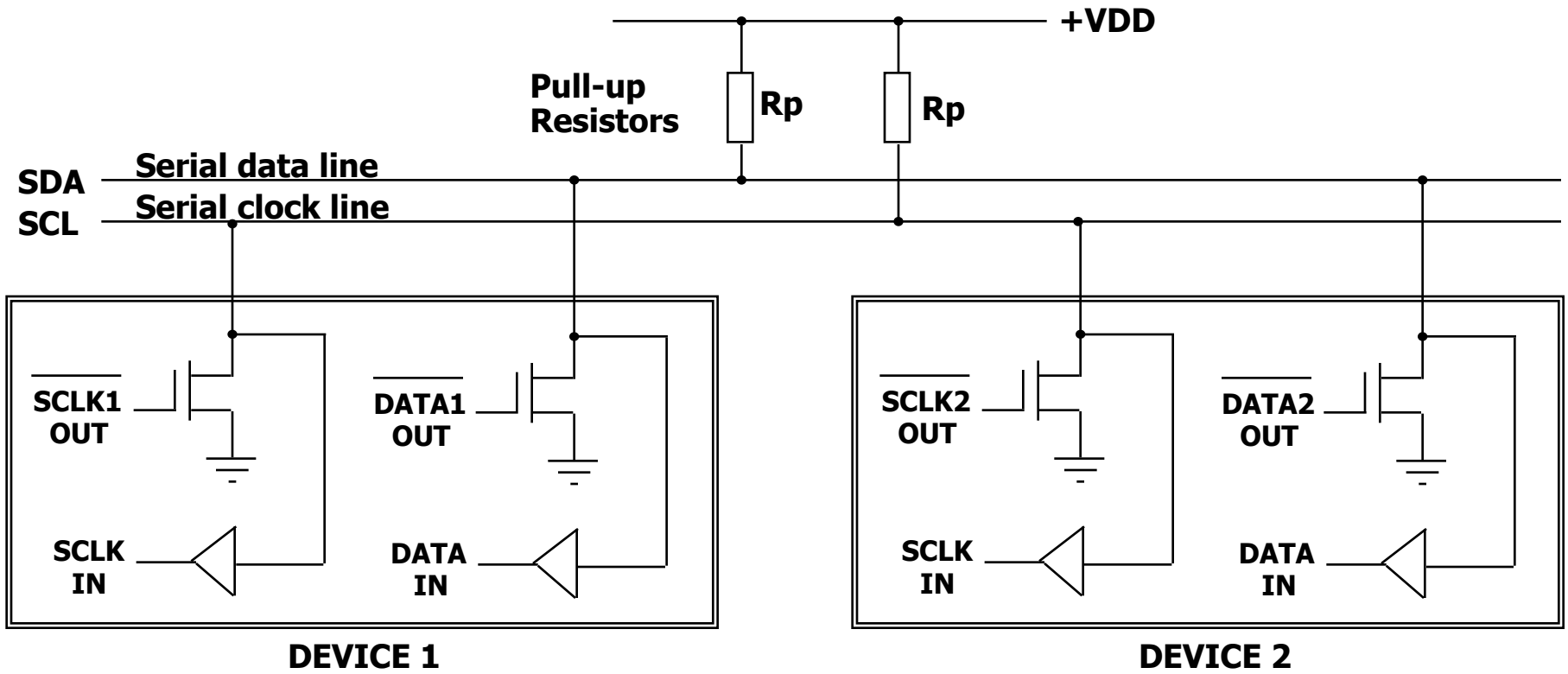
- If (a = b) then
- If (a == b)
- A = (A mod b) * b + (a rem b)



8. Tableaux

- Déclaration
  - type nom is array (intervalle) of type_de_base ;
- Utilisation
  - Modélisation de bus.
  - Modélisation de mémoires.
- Synthèse
  - Tableaux à 1 dimension
  - Indices entiers ou sous-type entiers
  - Eléments du tableau synthétisables

9. Modélisation d'un bus : Exemple du bus I2C : Connections par Drains ouverts



10. Modélisation d'un bus : conflits

- Conflit :
- Le bus Data_bus est relié aux ports d'entrées-sorties de 3 instances :
  - DEVICE_1 : SDA <= data_1 after t1 ;
  - DEVICE_2 : SDA <= data_2 after t2 ;
- Quelle est la valeur sur le bus ?

- Résolution du conflit
  - Le conflit doit être résolu par une fonction de résolution associée au type
  - C'est dans ce but qu'a été introduite la norme IEEE 1164

11. Types de base de la norme IEEE 1164

- Les types standards se sont avérés insuffisants et insuffisamment normalisés
- Ceci a conduit à la définition de nouveaux types en 1992 :
  - Std_ulogic
  - Std_logic
- Ainsi que des opérateurs sur ces types.
- Ces types sont synthétisables.

12. Type std_ulogic

- type std_ulogic is (

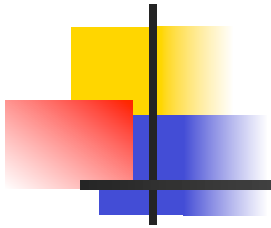
```
    'U',    -- Uninitialized,      Synthesis : No
    'X',    -- Forcing Unknown,    Synthesis : No
    '0',    -- Forcing 0,          Synthesis : 0
    '1',    -- Forcing 1 ,         Synthesis : 1
    'Z',    -- High impedance,     Synthesis : Z
    'W',    -- Weak unknown,       Synthesis : No
    'L',    -- Weak 0 ,            Synthesis : 0
    'H',    -- Weak 1 ,            Synthesis : 1
    '-'     -- Don't care          Synthesis : No
) ;
```

Table de résolution de conflits

	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'
'X'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
'0'	'U'	'X'	'0'	'X'	'0'	'0'	'0'	'0'	'X'
'1'	'U'	'X’	'X’	'1'	'1'	'1’	'1'	'1’	'X'
'Z'	'U'	'X'	'0'	'1'	'Z'	'W'	'L’	'H’	'X'
'W'	'U'	'X'	'0’	'1'	'W‘	'W'	'W'	'W'	'X'
'L'	'U'	'X'	'0'	'1'	'L'	'W'	'L'	'W'	'X'
'H'	'U'	'X'	'0'	'1’	'H'	'W’	'W'	'H'	'X'
'-'	'U'	'X'	'X'	'X’	'X’	'X’	'X’	'X'	‘X’

13. Sous-types résolus

- **function resolved (s : std_ulogic_vector) ;**
  return std_ulogic ;
- **subtype std_logic is resolved std_ulogic ;**
- ('X', '0', '1')
  subtype X01 is resolved std_ulogic range 'X' to '1' ;
- ('X', '0', '1', 'Z')
  subtype X01Z is resolved std_ulogic range 'X' to 'Z' ;
- ('U', 'X', '0', '1')
  subtype UX01 is resolved std_ulogic range 'U' to '1' ;
- ('U', 'X', '0', '1', 'Z')
  subtype UX01Z is resolved std_ulogic range 'U' to 'Z' ;



14. Constantes vectorielles

- Permet de donner la valeur présente sur un bus.
- Une constante chaîne de caractères est notée "AB"
- Une constante bit_vector est notée "0100"
- Une constante std_logic_vector est notée "01ZHWL"

- Constantes binaires : préfixe qui précise la base
  - B : 2
  - O : 8
  - X : 16
- b"0101"
- B"0100_0011"
- o"00" -- B"000_000"
- O"372" -- B"011_111_010"
- x"0d" -- B"0000_1101"
- X"FA" -- B"1111_1010"

15. Opérateurs sur les tableaux

- Tranches de tableaux
  - Exemple précédent
  - SRAM_Cells (0 to 32267) ;
- Concaténation
  - Opérateur &
    - ampersand
  - Exemples :
    - "abc" & ‘d’ = "abcd"
    - ‘e’ & ‘f’ = "ef"
    - B"0000" & B"1111" = B"00001111"
- Opérateurs sur les tableaux de bits et de booléens :
  - Not, or, nor, and, nand, xor, xnor
  - Sll : shift left logical
  - Srl : shift right logical
  - Sla : shift left arithmetic
  - Sra : shift right arithmetic
  - Rol : rotate left
  - ror : rotate right

16. Codage des nombres

- Signe
  - SVA
  - C1
  - C2
- Endianness
  - Little endian
    - Bits de poids faible rangés aux adresses les plus petites
    - Ex. Intel
  - Big endian
    - Bits de poids fort rangés aux adresses les plus petites
    - Ex. Motorola

17. IEEE 1076.3

- Ambiguités std et 1164
  - Bit_vector et std_logic_vector codés comme « paquets de bits »
  - Poids fort et poids faible non précisés
- Paquetages Numeric_bit et Numeric_std de 1076.3
  - Type unsigned : vecteurs de bits avec poids fort à gauche et poids faible à droite
  - Type signed : vecteurs de bits codés en C2 avec même convention de poids

18. Opérateurs sur tableaux de 1076.3

- Types synthétisables
  - Std_ulogic_vector
  - Std_logic_vector
  - Unsigned
  - Signed
- Extension par surcharge des opérateurs aux types :
  - integer
  - unsigned
  - signed
- Opérateurs :
  Not and nand or nor xor xnor
  + - * /
  Rem mod abs
  = /= < <= > >=
  shift_left, shift_right
  Rotate_left, rotate_right
- Alignements
  - resize
- Conversions
  - to_integer
  - To_signed
  - To_unsigned

19. Exemple : opérateur arithmétique

- library ieee ;
- use ieee.std_logic_1164.all ;
- use ieee. numeric_std.all ;

- entity add_32 is

```
port (
    c : out signed (31 downto 0) ;
    a : in signed (31 downto 0);
    b : in signed (31 downto 0)
    ) ;
    end entity  add_32 ;
```

- architecture behaviour of add_32 is

```
begin
c <= a + b after 50 ns ;
end architecture behaviour ;
```

IV. Modèles génériques et registres

Patrick.Garda@upmc.fr

Modèles génériques

- Problème
  - Modèles envisagés jusqu'à présent ont un comportement et une structure figée dans leur description
- Modèles évolutifs : 3 mécanismes en VHDL
  - Paramètres génériques
  - Instruction generate
  - Tableaux non contraints

Paramètres génériques

- Constantes dont la valeur peut être définie lors de leur utilisation
- Permettent de paramétrer une architecture :
  - Comportementale
  - structurelle

Comportement générique

- Les paramètres génériques sont définis dans l'entité
  - Avant les ports d'entrées-sorties
- Ils peuvent être utilisés :
  - Dans l'entité après leur déclaration
  - Dans le corps de toute architecture associée à l'entité

- Exemple
- Entity or2 is
  Generic (T_pd : time) ;
  Port (a, b : in std_logic ;
  Y : out std_logic)
  End entity or2 ;
- Architecture behaviour of or2 is
  Begin
  Y <= a or b after T_pd ;
  End architecture behaviour ;

Instanciation d'un paramètre générique

- La valeur du paramètre générique est précisée lors de l'instanciation de l'entité
- Des instances différentes peuvent utiliser des valeurs différentes

- Exemple
- Gate1 : entity work.or2(behaviour)
  Generic map (2 ns) ;
  Port map (in1, in2, y_out) ;
- Gate2 : entity work.or2(behaviour)
  Generic map (T_pd => 3 ns) ;
  Port map (a => a1, b => b1, y =>y1) ;

Valeurs par défaut

- Une valeur par défaut du paramètre générique peut être indiquée lors de sa déclaration
- Lors de l'instanciation, cette valeur peut être :
  - Utilisée telle quelle : open
  - Remplacée par une autre valeur

- Exemple
- entity dff is

  Generic (
  T_pd, T_su : time := 2 ns ; T_h : time := 0 ns) ;

  Port (clock, d : in std_logic ; q : out std_logic)
- Request_dff : entity work.dff (behaviour)

  Generic map (4 ns, 3 ns, open) ;

  Port map (system_clock, request, prending_request) ;

Structure générique

Les constantes génériques permettent aussi de paramétrer les structures

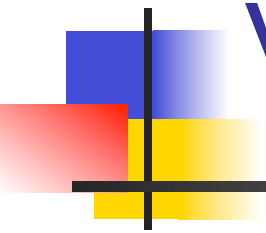
Exemple

```
Entity generic_register is
    Generic (w : positive := 32) ;
    Port (reset, clock : in std_logic ;
        d : in std_logic_vector (w - 1
        downto 0) ;
    q : out std_logic_vector (w - 1 downto
        0) ;
    End entity generic_register ;
```

```
Architecture behaviour of
    generic_register is
  Process (clock, reset) is
  Constant zero : std_logic_vector (w - 1
      downto 0) := (others => '0') ;
  Begin
  If reset = '0' then q <= zero ;
  Elsif clock'event and clock = '1' then q
      <= d ;
  End if ;
  End process ;
  End architecture behaviour ;
```

Aggrégats

- Permettent de donner les valeurs d'une mémoire
  - Cf instruction case
- Exemple
  - Subtype octet is integer range 0 to 255 ;
  - Subtype ram_address is integer range 0 to 65535 ; -- 64 kbytes
  - Type ram is array (ram_address) of octet ;
  - Signal coef_ram : ram := (0|1 => 17,
    2 to 1023 => 255, others => 0) ;
- Type ram is array (ram_address) of octet ;



V. VITAL

Patrick.Garda@upmc.fr

VITAL : une application des paramètres génériques

- Initiative VITAL (VHDL Initiative Towards ASIC Libraries) devenue standard IEEE 1076.4-1995
- Règles de modélisation VHDL pour le développement de bibliothèques ASIC.
- Les modèles VHDL de VITAL définissent les fonctionnalités des portes logiques ainsi que les noms et les significations des paramètres génériques représentant les délais associés aux portes.
- Les modèles VHDL de VITAL utilisent les types du paquetage standard STD_LOGIC_1164.

Intérêt de VITAL

- Les valeurs des délais dépendent de la technologie et sont toujours calculés en-dehors des modèles eux-memes.
- Elles sont stockées dans un fichier au format SDF et utilisées dans les modèles VHDL par l'intermédiaire des paramètres génériques.
- La simulation des circuits avec les modèles VITAL permet de valider une conception pour la fabrication (sign-off simulation).
- Elle peut être en plus accélérée grâce au codage des modèles au cœur du simulateur et à l'aide de variables.

Fichiers SDF

```
(DELAYFILE
 (SDFVERSION            "3.0")
 (DESIGN    "DigPart")
. . . .
  (CELLTYPE "DigPart")
  (INSTANCE)
  (DELAY
   (ABSOLUTE
    (INTERCONNECT sch\|B1.Q sch\|B2.A (0.0000:0.0000:0.0000) . . )
   )))
 (CELL
  (CELLTYPE "CLKBU2")
  (INSTANCE sch\|B1)
  (DELAY
   (ABSOLUTE
    (IOPATH A Q  (0.3314:0.3314:0.3314) (0.4205:0.4205:0.4205))
   )))
 (CELL
  (CELLTYPE "CLKBU2")
  (INSTANCE sch\|B2)
  (DELAY
   (ABSOLUTE
    (IOPATH A Q  (0.3221:0.3221:0.3221) (0.4363:0.4363:0.4363)) ))))
```

- Standard Delay Format
- Fichier SDF contient des informations sur les délais pour les librairies VITAL.
- Il sert à la rétroannotation des modèles après Placement et Routage.

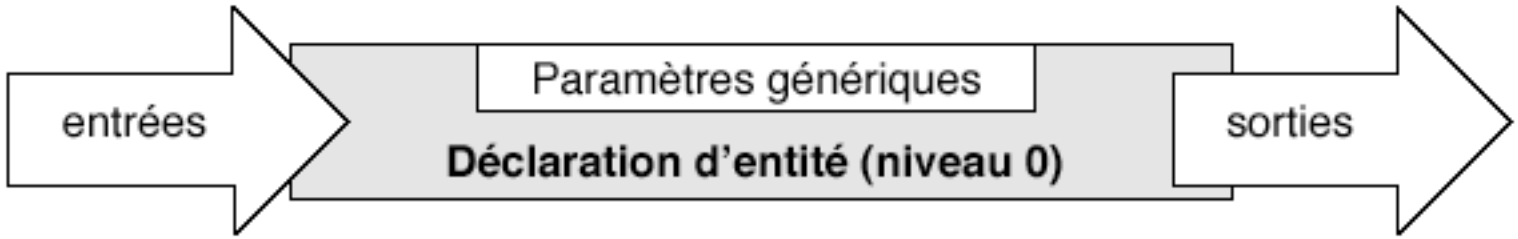
Bibliothèques VITAL

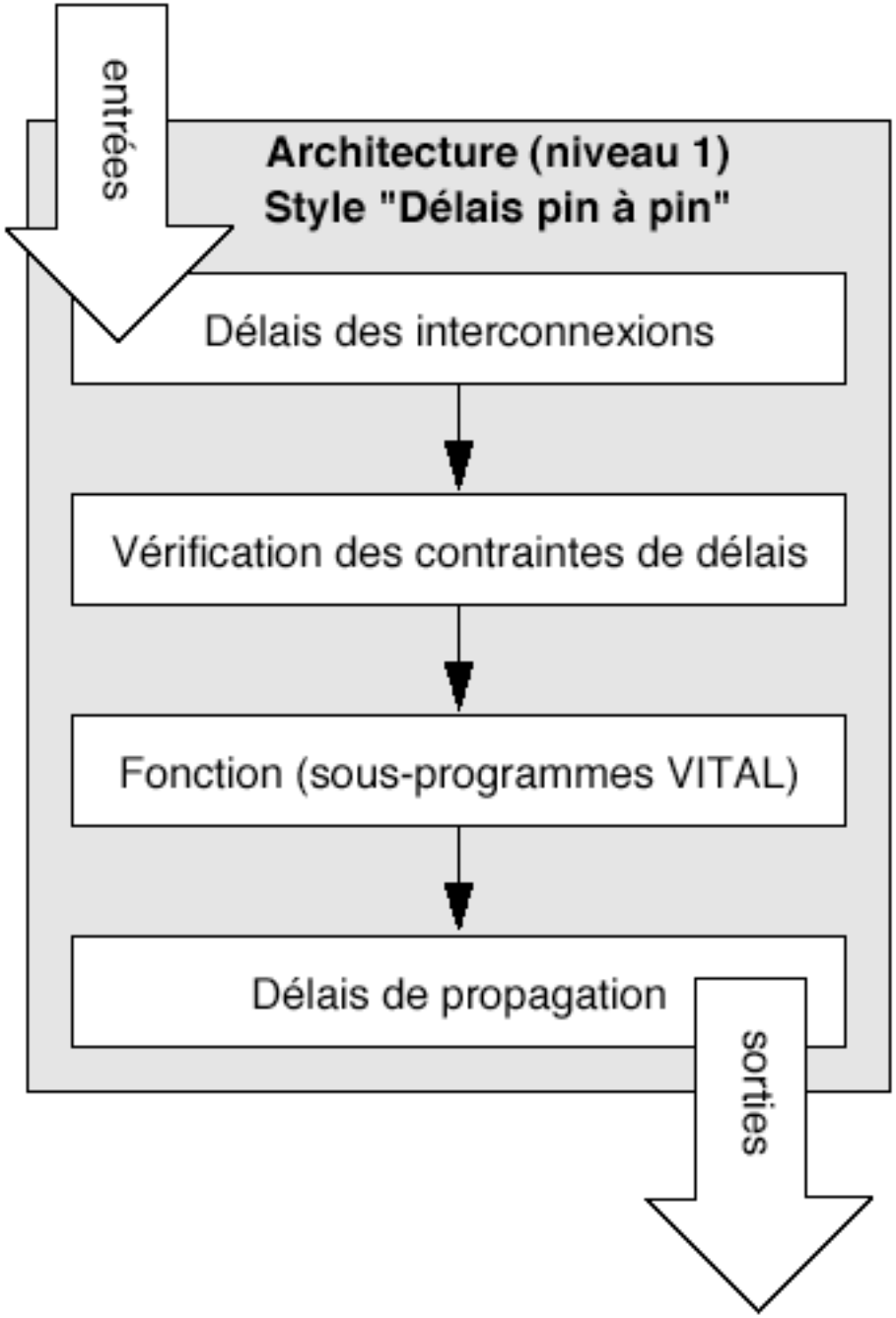
- Une bibliotheque de modeles VITAL pour des portes logiques standard est normalement incluse dans tout environnement de conception (design kit) fourni par les fondeurs.
- Les outils de synthèse sont tous capables de générer une description VHDL synthétisée au niveau porte supportant la norme VITAL.

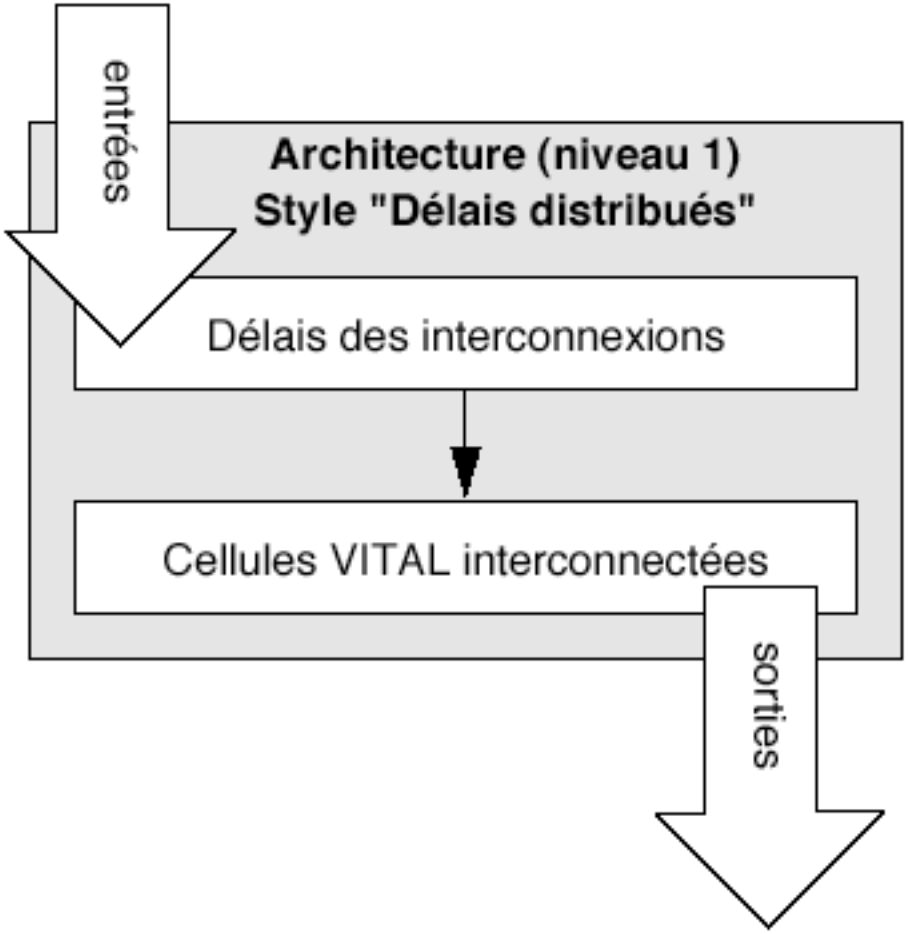
VITAL : niveaux 0 et 1

- La norme VITAL definit deux niveaux de modélisation.
- Le niveau 0 définit une déclaration d'entité standard qui inclut les noms et les types standard pour les paramètres génériques et les ports.
- Le niveau 1 ajoute des règles de modélisation pour le corps d'architecture. Il propose deux styles de modélisation :
  - le style "délais pin to pin" (pin-to-pin delay style)
  - le style "délais distribués" (distributed delay style).

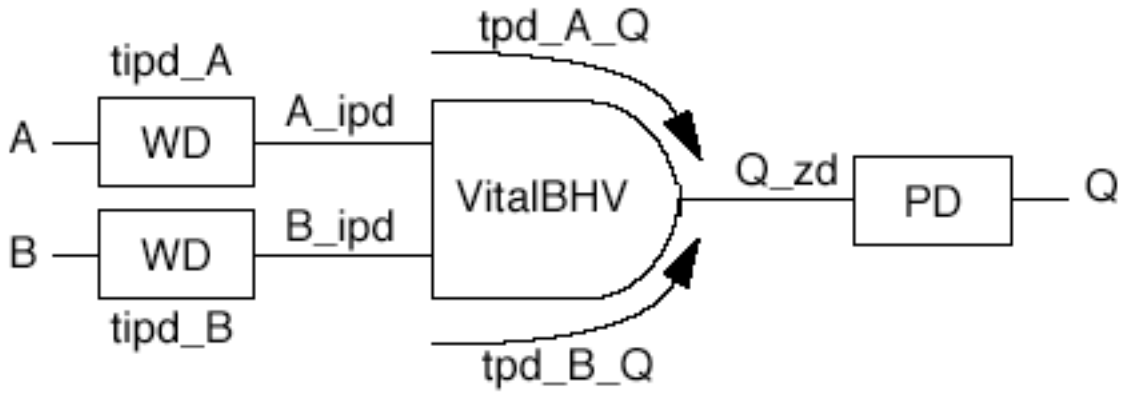
VITAL : niveau 0




entrées
Architecture (niveau 1)
Style "Délais pin à pin"
Délais des interconnexions
Vérification des contraintes de délais
Fonction (sous-programmes VITAL)
Délais de propagation
sorties


entrées
Architecture (niveau 1)
Style "Délais distribués"
Délais des interconnexions
Cellules VITAL interconnectées
sorties

Exemple :
AND2 délai « pin to pin »



Entité AND2 Délai « Pin to pin »

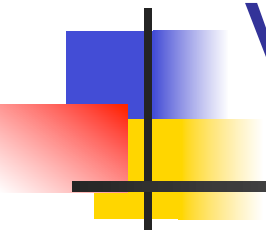
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.vital_timing.all;     -- paquetage VITAL
entity AND2 is
generic (               -- délais d'entrée et de propagation
      tipd_A              : VitalDelayType01 := (tr01 => 0 ns, tr10 => 0 ns);
      tipd_B                        : VitalDelayType01 := (tr01 => 0 ns, tr10 => 0 ns);
      tpd_A_Q             : VitalDelayType01 := (tr01 => 1.2 ns, tr10 => 1.1 ns);
      tpd_B_Q             : VitalDelayType01 := (tr01 => 1.2 ns, tr10 => 1.1 ns);
      TimingChecksOn: Boolean := FALSE); -- vérifications actives
port (Q                   : out std_logic;
        A, B : in  std_logic);
attribute VITAL_LEVEL0 of AND2: entity is TRUE;  -- la déclaration d'entité est du niveau 0
end AND2;
```

Architecture AND2 : début

```
use IEEE.vital_primitive.all;    -- paquetage VITAL
architecture level1 of AND2 is
attribute VITAL_LEVEL1 of level1: architecture is TRUE;  -- l'architecture est de niveau 1
-- signaux locaux prenant en compte les délais d'interconnexion
signal A_ipd, B_ipd: std_ulogic := 'U';
begin
WireDelay: block -- application des délais d'interconnexion
begin
    VitalWireDelay(A_ipd, A, tipd_A);
    VitalWireDelay(B_ipd, B, tipd_B);
end block WireDelay;
```

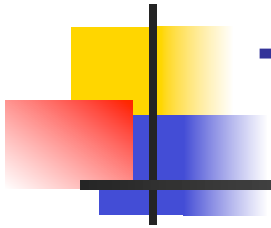
Architecture AND2 : fin

```
VITALBehavior: process (A_ipd, B_ipd)
variable GlitchData_Q: GlitchDataType; -- détection de courses
variable Q_zd: std_ulogic; -- valeur de sortie non retardée
begin
-- application de la fonction logique
Q_zd := VitalAND2(A_ipd, B_ipd, DefaultResultMap);
-- application des delais de propagation
VitalPathDelay01(                   OutSignal => Q,
                                    OutSignalName => "Q",
                                    OutTemp => Q_zd,
                                    Paths => (
                                                0 => (
                                                        InputChangeTime => A_ipd'last_event,
                                                        PathDelay       => tpd_A_Q,
                                                        PathCondition   => TRUE
                                                        ),
                                                1 => (
                                                        InputChangeTime => B_ipd'last_event,
                                                        PathDelay       => tpd_B_Q,
                                                        PathCondition   => TRUE
                                                        )
                                                ),
                                    GlitchData => GlitchData_Q,
                                    GlitchMode => NoGlitch,
                                    GlitchKind => OnEvent);
        end process VITALBehavior;
end level1;
```



VI. Textio

Patrick.Garda@upmc.fr



Textio

- Ce paquetage permet d'effectuer des entrées-sorties avec des fichiers de texte.
- Ceci permet de lire, par exemple :
  - Signaux (waveforms) pour la simulation et le test
  - Paramètres par exemple contenu de ROM, programme exécutable pour un cœur de micro,…
- Un paquetage d'entrées-sorties pour std_logic est en cours de normalisation à partir du paquetage std_logic textio de Synopsis.

Textio package

- **TEXTIO** fait partie de la library **STD** [VHDL LRM14.3]
- Ci-dessous les entêtes du package **TEXTIO** montrent les déclarations de types, subtypes, et l'utilisation des procédures

```
package Part_TEXTIO is        -- VHDL-93 version.
type LINE         is access STRING;  -- LINE est un pointeur vers un STRING.
type TEXT         is file of STRING; -- File est un fichier d'enregistrement ISO.
type SIDE         is (RIGHT, LEFT);  -- SIDE est un type énuméré
                                     -- qui définit la justification.
subtype WIDTH     is NATURAL;        -- définit la largeur des champs de sortie.
file INPUT        : TEXT open READ_MODE is "STD_INPUT";  -- Entrée par défaut.
file OUTPUT       : TEXT open WRITE_MODE is "STD_OUTPUT"; -- Sortie par défaut.
```

Textio package (suite)

```
-- Les procédures ci-dessous sont définies pour un type T :
-- BIT BIT_VECTOR BOOLEAN CHARACTER INTEGER REAL TIME STRING
procedure READLINE(file F: TEXT; L: out LINE);
procedure READ(L: inout LINE; VALUE: out T);
procedure READ(L:inout LINE; VALUE:out T; GOOD:out BOOLEAN);
procedure WRITELINE(F: out TEXT; L: inout LINE);
procedure WRITE(
   L: inout LINE;
   VALUE: in T;
   JUSTIFIED: in SIDE:= RIGHT;
   FIELD: in WIDTH := 0;
   DIGITS: in NATURAL := 0; -- for T = REAL only
   UNIT: in TIME:= ns);     -- for T = TIME only
function ENDFILE(F: in TEXT) return BOOLEAN;
end Part_TEXTIO;
```

Exemple : écriture à l'écran

```
library std;
use std.textio.all;
entity Text is end;
architecture Behave of Text is
signal count : INTEGER := 0;
begin count <= 1 after 10 ns, 2 after 20 ns, 3 after 30 ns;
process (count) variable L: LINE; begin
   if (count > 0) then write(L, now); -- Write time.
   write(L, STRING'(" count="));
-- STRING' is a type qualification.
   write(L, count); writeline(output, L);
   end if;
end process;
end;
```