

Méthodologie de conception et langages de description matérielle

VHDL

Support de cours

RISC-V

Master SIAME

F
P
G
A

Daniela Dragomirescu
François Thiebolt



Université
Paul Sabatier
TOULOUSE III



INSA

INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE





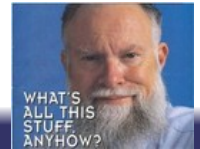
What's all this VHDL stuff, Anyhow* ?

Part I - VHDL language

- Introduction,
- VHDL basics,
- Use case: FIFO.

Part II will focus on synthesis for FPGA targets (Xilinx)

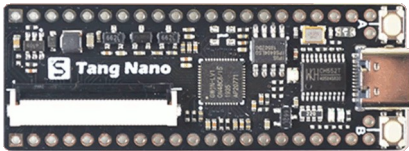
**Robert Allen Pease [1940 - 2011] was a famous TI's analog engineer,
his favourite programming language was ... solder!*



Foreword

For those who can't stand waiting for things to come ...

Tang Nano fpga board



You also have online resources like <https://edaplayground.com>

Rappels :

- Circuits logiques combinatoires

Leur sortie est déterminé par la valeur courante de l'entrée:
portes logiques, multiplexeurs, demultiplexeurs, décodeurs, circuits arithmétiques

- Circuits logiques séquentielles

Leur sortie est partiellement déterminée par l'évolution de l'entrée. La réponse du circuit dépend de l'histoire plus ou moins récente du fonctionnement du circuit.

Fonction **MEMOIRE** interne:

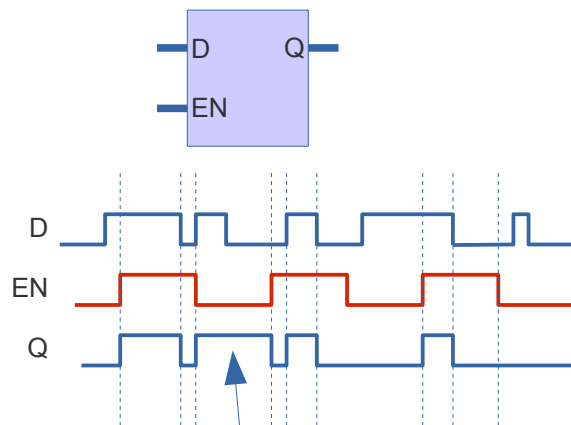
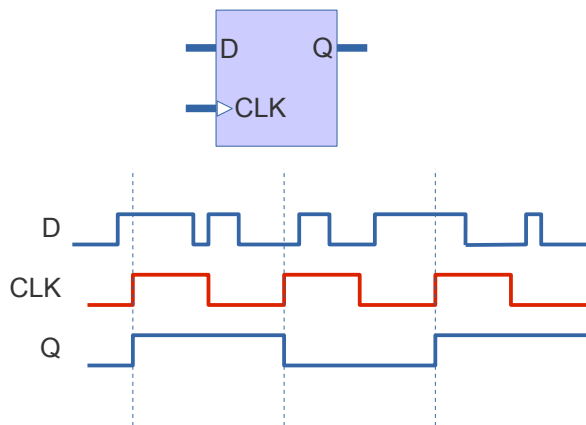
Bascule D, registres, mémoires RAM, compteurs

- Exercice : réalisez un compteur 4bits en utilisant des bascules D en logique séquentielle synchrone



Introduction

Rappels : D flip-flop vs D latch

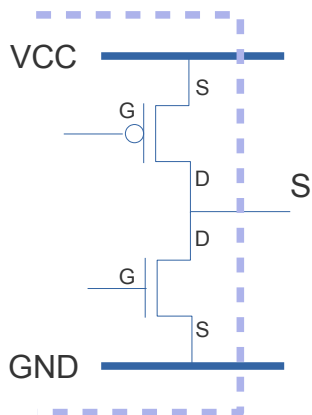


*Unsure about the output ...
guess why ??*

Introduction

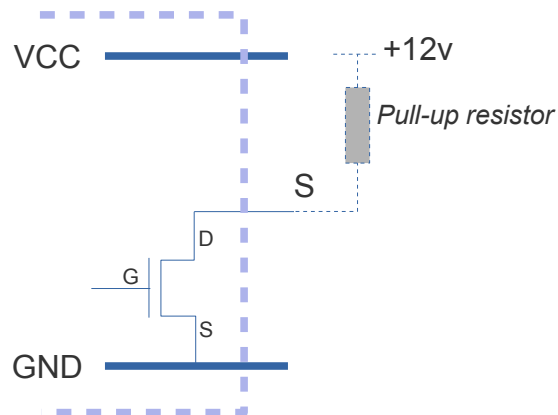
Rappels :

● Totem-pole (*push-pull*) output



Note: same design with bipolar transistors

● Open-Drain / Open-Collector output



Note: same design with bipolar transistors

Introduction

● Évolution des puces

Chips	Year	Frequency	Transistors	Tech.
Intel 8086	1978	8 MHz	29K	3.2μm
Intel 80486 DX	1989	25 MHz	1.2M	1μm
Pentium III XEON	1999	700 MHz	28M	180nm
Pentium IV	2001	1.7 GHz	42M	180nm
Core i7 Westmere	2010	3.2 GHz	1.17B	32nm
nVidia GT400	2010		3B	40nm
nVidia Tesla V100	2017		21.1B	12nm
ARM Cortex A-76 (Kirin 990)	2019	2.86 GHz	10.3B	7nm
H2020, EPI*, EU CPU	2021	??	???	6nm

2018

* European Processor Initiative; it will get manufactured by TSMC.



Introduction

● Accroissement de la complexité des puces:

Impose une évolution des méthodes des conception:

→ *la demande viendra du département de la défense des États-Unis en 1980:
ADA pour le logiciel, VHDL pour le matériel.*

Apparition des langages HDL - Hardware Description Language

Verilog
VHDL



Même principe, syntaxe différente.

Sont des langages de niveau Register Transfer Level

Ces langages 'décrivent' le matériel, ils n'exécutent rien!

Travail avec des cellules standard - briques de base appartenant à une bibliothèque spécifique à chaque fondeur de circuits intégrés



Standard cells - design kit

● VHDL: pro

VHDL est standardisé - standard IEEE

pérennité assurée par la norme

conception modulaire et hiérarchique

VHDL est un langage moderne, puissant et général

haute modularité

unités de compilation séparées

sécurité d'emploi

typage fort

fiabilité

généricité

notion de temps bien définie

En utilisant VHDL on minimise le risque d'erreur sur le circuit intégré en silicium - on diminue le coût de production

● VHDL: cons

langage de simulation



tout n'est pas synthétisable !

● Standardization

- ✦ IEEE VHDL'87 - 1987
- ✦ IEEE VHDL'93 - 1993
- ✦ IEEE VHDL'03 - 2003
- ✦ IEEE VHDL'08 - 2008

Specifications

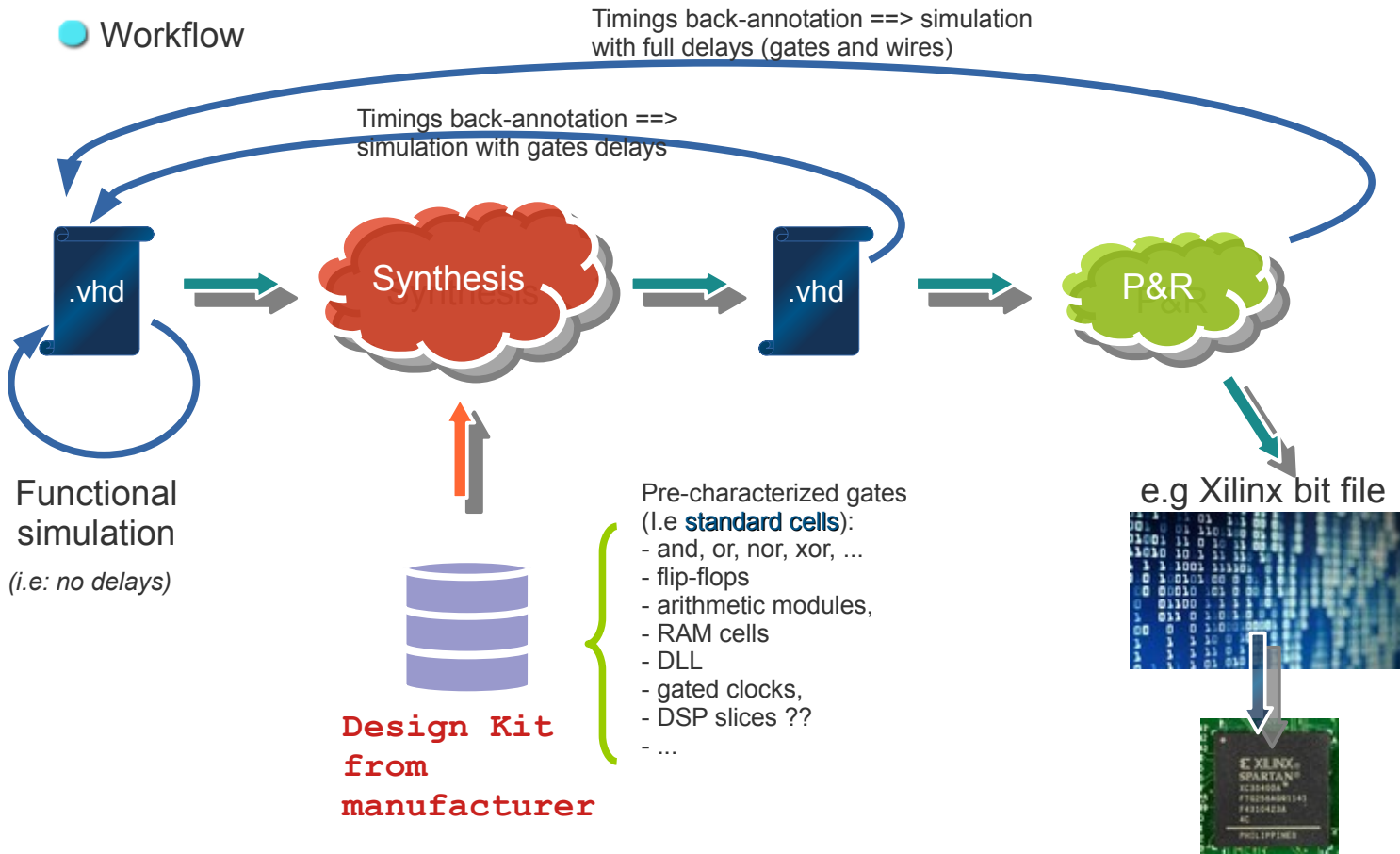
Simulator

Synthesis tools ! →

Never forget the overall objective: to produce chips through the use of synthesis tools!

Introduction

Workflow



Introduction

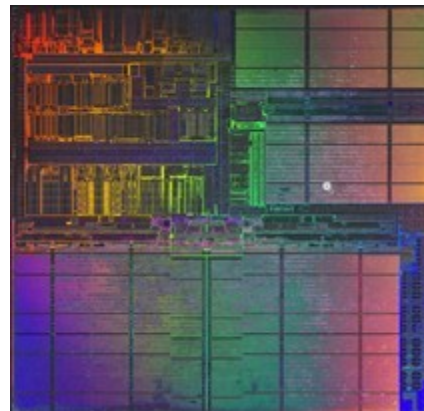
● Synthesis targets

FPGA



€€, Moderate speed, convert to ASIC for high volumes, partially customisable, errors ==> re-flash it :)

ASIC



(HP PA-RISC 8500 die)

€€€€, Highest speed, low-cost on volumes, fully customisable, errors ==> you're dead



Introduction

● Hardware configurable targets

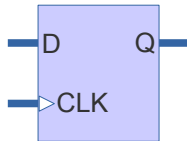
- ✦ CPLD → huge crossbar with surrounding I/O pads
- ✦ FPGA → CLBs spread over the entire chip with huge interconnexion matrix

High-End FPGA may contains either
Hard-IP processors (e.g dual ARM9 in Xilinx Zynq)
Softcore processor (e.g RISC-V or Leon)

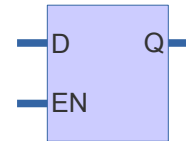
CPLD features less logic blocks but it exhibits deterministic propagation delays.

Quick start

● VHDL: D flip-flop vs D latch



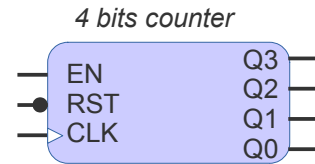
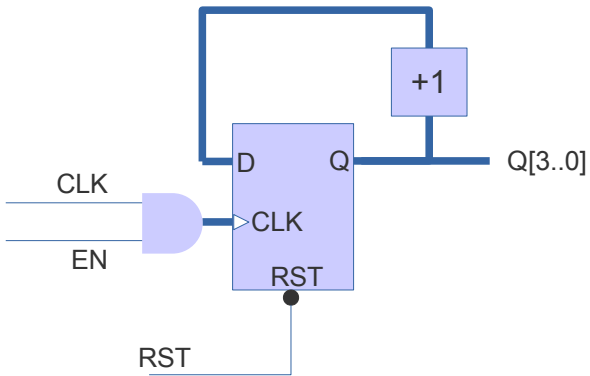
```
process( CLK )  
begin  
    if CLK'event and CLK = '1' then  
        Q <= D;  
    end if;  
end process;
```



```
process( D, EN )  
begin  
    if EN = '1' then  
        Q <= D;  
    end if;  
end process;
```

Quick start

● VHDL: 4 bits counter



```
process( RST,CLK )  
    variable cpt:std_logic_vector(3 downto 0);  
begin  
    if RST='0' then  
        cpt := (others=>'0');  
    elsif CLK'event and CLK = '1' then  
        cpt := cpt + '1';  
    end if;  
end process;
```

Some adjustment required ... we'll discover later.

Part I - VHDL language

- Introduction,
- VHDL basics,
- Use case: FIFO.



VHDL basics

Language features

- Packages

Along with package_body will hold all of your **types**, **constants**, **functions** and **procedures** definitions.

- Libraries

IEEE_std_logic_1164 is a compiled package provided as a system library. Your compiled stuff will go in your **WORK** library.

- Entity + Architecture of components

For each of your components: **entity** + **architecture** descriptions.

- Configurations

For each instantiated component, you can specify an architecture;
e.g: behaviour, synthesized, routed.

- Tests architecture

Yes, it's like a component with an empty **entity** (usually) along with instantiated components you want to test in its **architecture**.

● Libraries

Langage modulaire - unités petites et hiérarchisées

Unités de conception - peuvent être compilées séparément

Description VHDL correcte - bibliothèque de travail - **WORK**
portabilité

Utilitaires ou modèles généraux - bibliothèques de ressources
facilite le travail en équipe

Bibliothèques : IEEE , STD

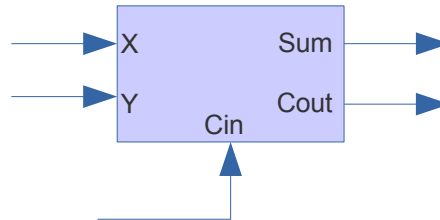
package IEEE.std_logic_1164 et **package** IEEE.std_logic_arith
package STD.textio et **package** STD.STANDARD

VHDL basics

- Unités de conception: **entity**

Entité - model VHDL

Vue externe



```
entity myadder is
    port (
        X, Y, Cin:    in std_logic;
        Sum, Cout:    out std_logic );
end myadder;
```

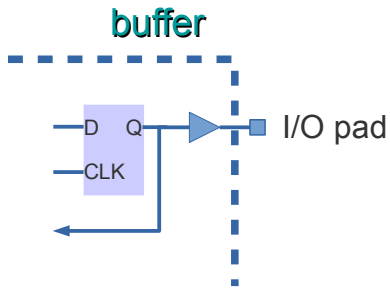
VHDL basics

ports modes

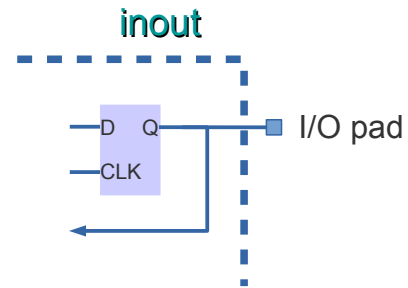
Formal def. vs Connected object	Mode IN	Mode OUT	Mode INOUT	Mode BUFFER
Port de mode IN	✓			
Port de mode OUT		✓		
Port de mode INOUT	✓	✓	✓	
Port de mode BUFFER	✓			✓
Local signal	✓	✓	✓	✓
open keyword (not connected)		✓	✓	✓

VHDL basics

ports modes: **BUFFER** vs **INOUT**



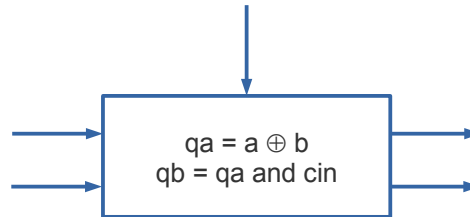
Output pin is buffered: you can **read what you wrote** but you can't read the value of the pin.



The value on the pin may differ from what you wrote !

Unités de conception: **architecture**

Vue interne d'une entité - **ARCHITECTURE**



Style de description de l'architecture :

- Description **structurelle** : interconnexion de composants
- Description **flot de données** : comportement sous forme d'équations
- Description **comportementale** : comportement sous forme algorithmique

On peut combiner les différents styles de descriptions dans une même architecture.

La synthèse produit une description structurelle.

VHDL basics

● Architecture **structurelle** de l'additionneur

```
architecture vue_structurelle of myadder is
```

```
    signal a,b,c : std_logic;
```

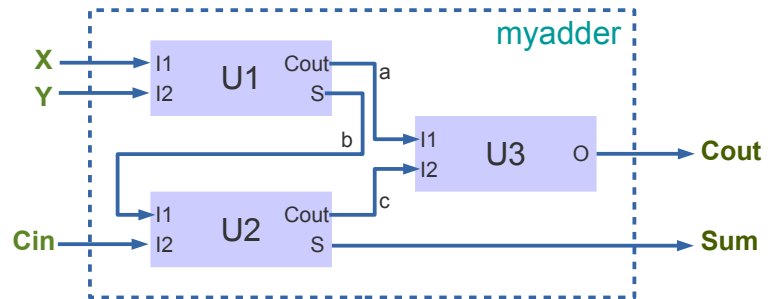
```
begin;
```

```
    U1:entity work.adder(behaviour)  
        port map(X,Y,a,b);
```

```
    U2:entity work.adder(behaviour)  
        port map(b,Cin,c,Sum);
```

```
    U3:entity work.or_gate(behaviour)  
        port map(a,c,Cout);
```

```
end vue_structurelle;
```



● Architecture **flot de données** de l'additionneur

$$S = X \oplus Y$$

$$\text{Somme} = S \oplus \text{Cin}$$

$$\text{Cout} = XY + S\text{Cin}$$

Equations booléennes de
l'additionneur

```
architecture data_flow of myadder is
    signal S : std_logic;
begin;
    Somme <= S xor Cin after 10 ns;
    S <= X xor Y after 10 ns;
    Cout <= (X and Y) or (S and Cin) after 20 ns;
end data_flow;
```


VHDL basics

- Architecture **comportementale** de l'additionneur
décrite par une table de vérité

```
architecture vue_comportementale of myadder is  
begin;
```

```
  process -- instruction concurrente
```

```
    variable N: integer;
```

```
    constant sum_vector: std_logic_vector(3 downto 0) := "1010";
```

```
    constant carry_vector: std_logic_vector(3 downto 0) := "1100";
```

```
  begin
```

```
    N:=0;
```

```
    if X= '1' then N:=N+1; end if;
```

```
    if Y= '1' then N:=N+1; end if;
```

```
    if Cin=1 then N:=N+1; end if;
```

```
    Somme <= sum_vector(N) after 20 ns;
```

```
    Cout <= carry_vector(N) after 30 ns;
```

```
    wait on X, Y, Cin;
```

```
  end process;
```

```
end vue_comportementale;
```

X	Y	Cin	Sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Configurations

Donne la correspondance entre le composant et le modèle dont il est instancié

Permet de faire la liaison entre des composants utilisés dans une architecture et leur réalisation effective

Ex: Soit une entité « trois_registres » avec une architecture « beh » ayant 3 registres R1, R2 et R3

```
use work.trois_registres;  
configuration alpha of trois_registres is  
  for beh  
    for R1,R2 : registre_8bit use entity work.registre_8bit(arch);  
    for R3 : registre_8bit use entity work.registre_8bit(decalage);  
  end for;  
end alpha;
```

alternative to apply the same architecture for all instantiated components:

```
FOR ALL : registre_8bit use entity work.registre_8bit(arch);
```

● Configuration example for our adder

```
library work;
entity myadder is
    port (X, Y, Cin : in std_logic;
          Sum, Cout : out std_logic);
end myadder;
architecture vue_structurale of myadder is

    for all : adder use entity work.adder(beh);
    for all : or_gate use entity work.or_gate(struct);

    signal a,b,c : std_logic;
    begin
        U1:entity work.adder(behaviour)
            port map(X,Y,a,b);

        U2:entity work.adder(behaviour)
            port map(b,Cin,c,Sum);

        U3:entity work.or_gate(behaviour)
            port map(a,c,Cout);
    end vue_structurale;
```

```
library ma_lib;

for all : adder use entity ma_lib.adder(beh);
```

● Packages

Ensemble des algorithmes, sous-programmes, nouveau types et sous-types, des objets: signaux et constantes (pas de variables)

Un ou plusieurs paquetage dans la même bibliothèque

2 unités de conception :

Spécification d'un paquetage - vue externe

présente tous ce qu'exporte le paquetage (algorithmes, objets, types)

Corps du paquetage - vue interne - *optionnel*

contient la description des algorithmes, déclarations locales des types, objets

Pour avoir accès à un paquetage il faut le référencé par une clause **use**

```
library IEEE;  
    use IEEE.std_logic_1164.all ;  
library ma_lib;  
    use ma_lib.mon_paquetage.all;
```

● Packages

Spécification du paquetage :

```
package SIMPLE is
```

```
    constant TAILLE_MAX: integer :=1024;
```

```
    type mon_integer is INTEGER range 0 to TAILLE_MAX;
```

```
    signal addition_10bits : mon_integer ;
```

```
    function MIN(A,B:INTEGER) return INTEGER;
```

```
    function MAX(A,B:INTEGER) return INTEGER;
```

```
end SIMPLE;
```

Les déclarations faite dans la spécification du paquetage sont connues dans le corps du paquetage.

● Packages - **body**

Corps du paquetage

```
package body SIMPLE is
  function MIN(A,B:INTEGER) return INTEGER is
  begin
    if A<B then return A;
    else return B;
    end if;
  end MIN;
  function MAX(A,B:INTEGER) return INTEGER is
  begin
    if B<A then return A;
    else return B;
    end if;
  end MAX;
end SIMPLE;
```

● Concurrent and Sequential domains

La description d'un système matériel est naturellement concurrente.

Le fond d'une description VHDL est **concurrent**.

Les 2 domaines cohabitent en VHDL

Domaine **concurrent**

La zone de déclarations des **entités**

La zone de déclarations des **architectures**

Domaine **séquentiel**

La zone de déclarations de **processus**

Le **corps du paquetage**

● Tests

Pour **vérifier** (**simuler**) le comportement du composant décrit en VHDL il faut le **tester**

Le programme de test - un programme VHDL, avec la même structure (entity, architecture)

Applique les stimuli en entrée et regarder les sorties du composant sous test

Si possible - test exhaustif

Difficile à mettre en œuvre pour des systèmes complexes

Preuve formelle



VHDL basics

Exemple: test de l'additionneur

```
entity test_adder is
end test_adder;
architecture bench of test_adder is
    For all : adder use entity work.adder(vue_structurale);

    signal data1, data2, data3 : std_logic;
    signal data_out, carry_out : std_logic;
begin
    additionneur: entity work.adder PORT MAP (data1,data2, data3,data_out,carry_out);
    data1 <= '0', '1' after 30 ns;
    data2 <= '1', '0' after 50 ns, '1' after 60 ns;
    data3 <= '0', '1' after 12 ns;
end bench;
```

Test non exhaustif !!

Opérateurs et littéraux

Classes d'opérateurs par ordre de priorité croissante

1. Logiques : and or nand nor xor;

- défini sur les types booléen et BIT

2. Relationnels: = /= < <= > >=

- défini sur tous les types sauf le type file

3. Arithmétiques et concaténation: + - &

4. Signe: + -

- les opérateurs de signe sont moins prioritaires que les opérateurs de multiplication !!!!!

5. Multiplication: * / mod rem

6. Exposant, valeur absolue, complément: ** abs not

- ** élévation à la puissance : opérande de droite (la puissance) doit être entière

Il est possible de **surcharger** les opérateurs ; ceci ne change pas leur priorité

● Opérateurs et **littéraux**

Classifications:

numérique - notation décimale - entier (1345 ou 1_345) ou réel (13.0)

- notation basée - base 16 : X"A2"

caractères ou chaîne de caractères ('a' , "Bonjour", "1234")

énumérés (type ALU_OPS is (ADD,SUB,ADDI,SUBU, ...)

chaînes de bits ("1010")

null

WARNING:

1 - nombre entier

'1' - bit

Expressions - portent un type, et au moment de l'exécution une valeur

Identificateurs - commencent avec une lettre; pas de différence entre majuscules et minuscules

● Objets

Les objets contiennent des valeurs. Il y a 4 classes d'objets en VHDL :
constantes, variables, fichiers et signaux.

Les constantes ont une valeur unique fixe

Les variables ont une valeur unique modifiable

Les fichiers contiennent des séquences de valeurs qui peuvent être lues ou écrites

Les **signaux** conservent l'histoire des valeurs passées, de la valeur présente et des valeurs prévues dans le futur : seules les valeurs futures peuvent être modifiées par affectation de signal

Tous les objets ont un type

● Types

VHDL est un langage typé

Un type est un ensemble de valeurs ordonnées

Le type est **statique** : il ne peut pas être modifié

Il est possible de définir de nouveaux types en utilisant les types prédéfinis et des constructeurs de types

Classification :

- Types scalaires

- Types composites

- Types **access** (pointeur) - seules **les variables** peuvent être de type acces

- Types fichiers - mot clé **file**

● Types scalaires

Les entiers

```
type mon_integer is INTEGER range -65_536 to 65_535 ;
```

Les flottants

```
type mon_flottant is REAL range 5.36 downto 2.15;
```

Les types énumérés

```
type COULEUR is (BLEU, VERT, ROUGE);
```

```
type BOOLEAN is (FALSE, TRUE);
```

```
type LOGIC4 is ('X', '0', '1', 'Z');
```

Les types physiques

TIME - défini dans le paquetage STANDARD

TIME - la notion de temps que connaît le simulateur

Sont caractérisés par l'unité de base, l'intervalle de ses valeurs autorisées, collection des sous-unités et leur correspondance.

Types physiques

TIME

```
type TIME is range -9_223_372_036_854_775_808 to 9_223_372_036_854_775_807
```

```
-- codage sur 64 bits ;
```

```
units fs ;
```

```
ps = 1000 fs;
```

```
ms = 1000 us;
```

```
ns = 1000 ps;
```

```
sec = 1000 ms;
```

```
us = 1000 ns;
```

```
min = 60 sec ;
```

```
hr = 60 min;
```

```
end units;
```

DISTANCE

```
type DISTANCE is range 0 to 1E16
```

```
units A ;
```

```
nm = 10 A;
```

```
um = 1000 nm;
```

```
mm = 1000 um;
```

```
cm = 10 mm;
```

```
m = 1000 mm;
```

```
km = 1000 m;
```

```
end units;
```

VHDL basics

● Type **STD_LOGIC**

Se trouve dans le paquetage IEEE.std_logic_1164

Paquetage IEEE.std_logic_unsigned

Paquetage IEEE.std_logic_arith

u stands for unsolved ... not unsigned!

std_logic est le type **résolu** de std_u**l**ogic

'1' – 1 *strong*
'0' – 0 *strong*
'Z' – high impedance
'U' – uninitialized
'X' – undetermined
'H' – 1 **weak**
'L' – 0 **weak**
'W' – **weak** unknown

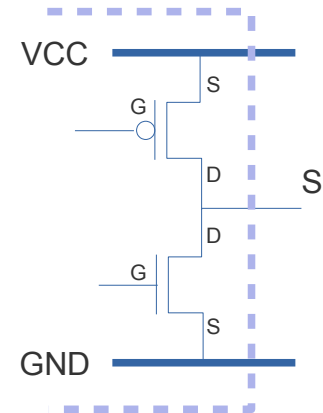
std_logic resolution function

A	B	out
'1'	'Z'	'1'
'0'	'H'	'0'
'0'	'1'	'X'
'H'	'L'	'W'
...

STD_LOGIC_VECTOR

```
signal A,B : std_logic_vector(7 downto 0);  
signal mycpt: std_logic_vector(0 to 3);
```

array of std_logic



Types composites

Classification :

Tableaux : collection d'objets de même type - **array**

Articles : collection d'objets de types différents - **record**

Tableaux

Ses éléments sont désignés par un indice

Contraints (un entier)

type MOT **is array** (0 to 31) **of** STD_LOGIC;

Non-contraints

type STD_LOGIC_VECTOR **is array** (NATURAL **range** <>) **of** STD_LOGIC;

Définition d'un intervalle d'indilage lors de l'exécution

signal bus : STD_LOGIC_VECTOR (63 **downto** 0);

Attributs de tableaux :

LEFT, RIGHT, HIGH, LOW, RANGE, REVERSE_RANGE, LENGTH

```
type FILE_REGS is array(0 to (2**3)-1) of std_logic_vector(31 downto 0);  
signal regs : FILE_REGS;
```

● Tableaux - **attributs**

Exemple :

```
type INDEX1 is INTEGER range 1 to 20;  
type INDEX2 is INTEGER range 7 downto 3;  
type VECTEUR1 is array (INDEX1) of STD_LOGIC;  
type VECTEUR2 is array (INDEX2) of STD_LOGIC;
```

VECTEUR1'**LEFT** rendra 1 et VECTEUR2'**RIGHT** rendra 3

VECTEUR1'**HIGH** rendra 20 et VECTEUR2'**HIGH** rendra 7

VECTEUR1'**LOW** rendra 1 et VECTEUR2'**LOW** rendra 3

VECTEUR1'**RANGE** rendra 1 to 20 - utilisation pour des boucles

VECTEUR1'**REVERSE_RANGE** rendra 20 **downto** 1

VECTEUR1'**LENGTH** rend le nombre d'éléments du tableau - donc 20

Types composites - **articles**

Articles - mot clé **record** - **end record**

Contient des éléments de types différents

Ses éléments sont désignés par un **nom**

Exemple :

```
type BIT_COMPLET is record  
    valeur : std_logic;  
    sortance_min, sortance_typ, sortance _max: integer;  
end record;
```

```
signal A : BIT_COMPLET;  
variable B : BIT_COMPLET;
```

alors A.valeur est de type std_logic, **A.sortance_typ** est de type entier

Affectation

```
A <= B ;  
ou  
A.valeur <= B.valeur after 50 ns;
```

● Aggrégats

Un agrégat est une façon d'indiquer la valeur d'un type composite

Un agrégat se note entre parenthèse, les éléments étant séparés par des virgules

Vérification de type faite par le compilateur

Exemple : soit

```
type TAB is array (1 to 3) of INTEGER ;
```

```
type ART is record
```

```
    Champ1 : NATURAL;
```

```
    Champ2 : STD_LOGIC;
```

```
    Champ3 : NATURAL;
```

```
end record;
```

```
signal A : TAB;    signal B : ART;
```

3 notations distinctes de l'agrégat :

notation positionnelle

```
A <= (12, 13, 14);
```

```
B <= (5, '0', 15);
```

notation par denomination

```
A <= (1=>12, 2=>13, 3=>14);
```

```
B <= (Champ3 =>15, Champ1 => 5, Champ2 =>'0' );
```

notation mixte (positionnelle -denomination)

```
A <= (5, others =>0);
```

```
B <= (Champ2 =>'0', others =>0);
```

```
A <= (others =>0);
```

● Sous-types

Déclaration de sous-type quand on souhaite restreindre les valeurs d'un certain type

Un sous-type est toujours compatible avec son type de base

Exemples:

```
subtype integer_8bits is INTEGER range 0 to 255 ;
```

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
```

Sous-types dynamiques:

Font intervenir des expressions calculables à la simulations

```
Subtype MOT is STD_LOGIC_VECTOR (1 to MAX);
```

où MAX est un paramètre d'un sous-programme ou un paramètre générique d'une entité

● Notions de signal et affectations du signal

Un signal correspond à la représentation matérielle du support de l'information en VHDL.

Un signal a une évolution en temps (une variable - non)

Toute déclaration de signal se fait dans le **domaine concurrent** - la zone de déclaration des architectures, des entités et la spécification du paquetage

Tout signal peut conserver l'historique de ses valeurs passées si celles-ci sont utilisées par ailleurs, la valeur présente et les valeurs prévues dans le futur - le «pilote» du signal

Affectation du signal :

connexion à un port de sortie d'un composant

affectation du signal dans le domaine concurrent

affectation du signal dans le domaine séquentiel

- Affectation inconditionnelle de signal

```
S <='0', '1' after 20 ns;
```

La valeur doit avoir un type compatible avec celui du signal affecté

Chaque élément de l'expression a une partie valeur et une partie délai de type physique TIME (délai nul par défaut sans la clause **after**)

```
S <= A after 10 ns ;
```

Tous les signaux utilisés dans l'expression sont soit locaux, soit des ports définis en entrée

- Exécution d'une affectation de signal

L'affectation du signal ne change pas la valeur présente du signal, mais modifie les valeurs futures que ce signal sera susceptible de prendre

$A \leq B$; ou $A \leq B$ after 0 ns;

Le signal A prend la valeur présente du signal B après un delta délai

Delta délai - est un délai qui est nul pour la simulation et ne représente que la causalité des évènements.

● Exécution d'une affectation de signal

$S \leq A+B$ **after** 20 ns;

L'expression affectée à un signal est évaluée à chaque changement de valeur sur l'un de ses signaux d'entrée (réponse événementielle)

Chaque valeur calculée par l'expression est mémorisée dans le signal destination avec son délai d'apparition

Si l'affectation est de nouveau évaluée, les anciennes valeurs prévues pour le futur peuvent être remplacées par les nouvelles valeurs - voir exemple affectation conditionnelle

● Affectation conditionnelle de signal

Condition simple:

```
S <= A when condition else  
      B;
```

Instruction conditionnelle concurrente

```
S <= A after 20 ns when condition else  
      B after 10 ns;
```

Condition multiple avec ordre de précedence

```
S <=      5 when A='1' and B='1' else  
          4 when A='1' else  
          0;
```

Instruction conditionnelle concurrente

Quand n'importe quel signal d'entrée change de valeur,
l'ensemble des conditions est évalué dans l'ordre (de gauche à droite)



La première valeur produite par la première expression dont la condition est vraie est affectée au signal !

● Affectation sélective de signal

Une condition unique détermine quelle expression sera affectée au signal

```
type alu_ops is (add, sub, and, xor) ;
```

```
signal opcode: alu_ops;
```

```
signal result, A, B, : std_logic;
```

```
.....
```

```
with opcode select    – instruction de choix concurrente
```

```
    result <= A + B when add,  
              A - B when sub,  
              A and B when and,  
              A xor B when xor,  
              '0' when others;
```

```
result <= A + B when opcode==add else  
        A - B when opcode==sub else  
        A and B when opcode==and else  
        A xor B when opcode==xor;
```

L'affectation sélective modélise les composants de type multiplexeurs et décodeurs

*not anymore a muxer ...
guess why ?*

VHDL basics

Attributs

L'attribut est une caractéristique associée à un type ou à un objet

L'utilisateur peut définir des nouveaux attributs

Prédéfini :

S'**quiet**(T) - TRUE si le signal S est "tranquille" pendant au moins T

S'**stable**(T) - TRUE si aucun événement sur le signal S depuis T

S'**delayed**(T) - signal S différé de T après **NOW**

des affectations au
delà du temps T

S'**event** - TRUE si un événement vient d'arriver sur S

des affectations mais qui
ne changent pas la valeur

S'**last_value** - dernière valeur de S avant le dernier événement

S'**last_event** - valeur du temps écoulé depuis le dernier événement de S

S <= **transport** A and B after 20ns;

also **inertial** and **reject** to model propagation on wires.

● Variable vs Signal

Les variables se déclarent dans le **domaine séquentiel des processus ou des sous-programmes**

Les variables s'utilisent et s'affectent dans le domaine séquentiel uniquement `a:='1'`

Les signaux se déclarent dans le **domaine concurrent** des entités, architectures, spécification du paquetage, des blocs

Les signaux s'utilisent dans les 2 domaines séquentiel et concurrent `a <='1'`

Une affectation de variable est **immédiate**, alors qu'une affectation de signal est différée jusqu'à la fin de l'évaluation de toutes les affectations.

e.g lorsque l'on arrive sur une instruction `wait` dans un `process`

● Instructions concurrentes

On ne simule pas en temps « réel » - on simule en temps virtuel
(temps de simulation)

Ces fonctionnements seront décrits par des instructions concurrentes
s'exécutant de manière asynchrone

Assignations de signaux (conditionnelle ou sélective ou inconditionnelle)

Instanciations de composants

Instruction **processus**

Appels concurrents de procédures

Instructions d'assertion

Instructions de bloc

Instruction **generate**

● Instructions concurrentes

Processus

est l'objet fondamental manipulé par le simulateur

toute instruction concurrente peut toujours être traduite par un processus

un processus est sensible aux signaux

il contient que des instructions séquentielles

la durée de vie d'un processus est celle de la simulation; un processus est cyclique

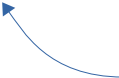
e.g affectation dans le domaine concurrent:

```
S <= A + B;
```

est en fait un processus avec en liste de sensibilité les signaux situés en partie droite de l'affectation (i.e A, B)

● Instructions concurrentes - **Processus**

```
{label: } process { liste_des_signaux_surveillés}
  Déclarations
  begin
    Instructions séquentielles
  end process {label};
```



liste de sensibilité

Les déclarations sont élaborés une seule fois, à l'initialisation
un processus s'exécute au moins une fois à l'initialisation jusqu'à
l'instruction **WAIT** (si elle existe)
à chaque événement intervenant sur un des signaux surveillés, le
processus va s'exécuter

Restrictions:

l'instruction **wait** bloque le processus – voir instruction **wait** plus loin

l'instruction **wait** ne peut s'utiliser à l'intérieur d'un processus que si
celui-ci ne possède pas de liste de signaux surveillés

● Instructions concurrentes - **Processus**

Processus avec liste de sensibilité :

```
{label: } process  
  {liste_des_signaux_surveillés}  
  Déclarations  
  begin  
    Instructions séquentielles  
  end process {label};
```

S'exécute à l'initialisation une fois

Processus sans liste de sensibilité:

```
{label: } process  
  Déclarations  
  begin  
    wait on {signaux_surveillés }  
    Instructions séquentielles  
  end process {label};
```

Ne s'exécute pas lors de
l'initialisation

● Instructions concurrentes

Block

réuni des instructions **concurrentes**

est la base de hiérarchie en VHDL

toute hiérarchie VHDL se ramène à un ou plusieurs blocs imbriqués

permet de garder des affectations de signaux - circuits synchrones

```
label : block  
    { généricité_et_port}  
    ... Déclarations ...  
    begin  
    ....Instruction concurrentes  
    end block;
```

Appel concurrent de procédure

a la même syntaxe que l'appel séquentiel de procédure

**domaine concurrent - paramètres de la procédure ne peuvent être que de signaux
ou de constantes**

```
procedure alu(A:in std_logic_vector, ..., signal S: out std_logic,... );
```

Instruction concurrente d'assertion

● Instructions séquentielles

Domaine d'utilisation des instructions séquentielles : dans le corps d'un **processus** ou sous-programme

Instruction **WAIT**

Instruction **ASSERT**

Instructions d'affectation de variables et de signaux

Appel de procédures

Instructions conditionnelles

Instructions de contrôle

Instruction nulle - **null**

● Instructions séquentielles

Instruction **WAIT**

L'exécution de processus ou de programme peut être suspendue, en fonction d'une condition donnée et pour un temps indiqué par l'instruction **wait**

wait { **on** liste _signaux } { **until** condition_booléenne } { **for** temps };

Tout événement arrivant sur un signal de la liste donnée provoque l'évaluation de la condition booléenne.

Exemple:

Clock generator in a test architecture:

```
signal clk: std_logic;
.....
clock: process
begin
  clk <= '1';
  wait for 5ns;
  clk <= '0';
  wait for 5ns;
end process clock;
```

```
process
begin
  wait on CLK;
  if clk'event and CLK='1' then
    q <= d after 10 ns;
  end if;
end process;
```

```
process
begin
  wait until CLK'event and CLK ='1';
  q <= d after 10 ns;
end process;
```

wait until <condition>
attend un évènement sur l'un des signaux
ET teste ensuite la condition !!

● Instructions séquentielles

Instruction **ASSERT**

Surveille une condition et émet un message dans le cas où celle-ci est **fausse**.

L'exécution du programme reprend alors immédiatement derrière l'instruction d'assertion.

```
assert condition {report mesg} {severity niveau}
```

```
assert NOW<1 min report "Fin simu" severity ERROR;
```

Type Severity_Level is (note, warning, error, faillure);

Timeout generator in a test architecture:

```
timeout: process
begin
    wait for 150ns;
    assert false report "Timeout !!!" severity failure;
end process timeout;
```

● Instructions séquentielles

Instructions d'affectation de variables et de signaux

Variable1 := 2;

Signal1 <= 2 after 20 ns;

Appel de procedure

procedure alfa (nr : integer, msg : string) --définition de la procédure

Alfa(4, "GO");

--appel positionnel

Alfa(nr =>4, msg =>"GO");

Instruction return

Réservée aux sous-programmes

Instruction nulle

null

l'exécution passe à la ligne suivante

L'instruction nulle n'est pas nécessaire à la compilation de processus ou de corps de procédures vides

VHDL basics

● Instructions séquentielles

➤ Instructions conditionnelles

```
if condition1 then traitement1
elsif condition2 then traitement2
else traitement3
end if;
```

Instructions de choix:

```
case expression is
  when val1 => instructions1
  when val2 => instructions2
  when others => instr3
end case;
```

e.g procedure alu

```
type ALU_OPS is (ALU_ADD, ALU_SUB, ALU_AND ...);
signal CTRL_ALU : ALU_OPS;
.....
case CTRL_ALU is
  when ALU_ADD | ALU_SUB | ALU_SLT =>
    b_in := B;
    c_in := '0';
    if (CTRL_ALU /= ALU_ADD) then
      b_in := not(B);
      c_in := '1';
    end if;
    adder_cla(A,b_in,c_in,tmp_S,tmp_C,tmp_V);
    if (CTRL_ALU = ALU_SLT) then
      tmp_S := .....
      tmp_C := '0';
      tmp_V := '0';
    else
      tmp_C := not(SIGNED_OP) and tmp_C;
      tmp_N := SIGNED_OP and tmp_S(DATA_WIDTH-1);
      tmp_V := SIGNED_OP and tmp_V;
    end if;
  when ALU_AND =>
    tmp_S := A and B;
    .....
  when others =>
end case;
```

VHDL basics

● Instructions séquentielles

Instructions de boucle

loop
 instructions séquentielles;
end loop;

for indice **in** intervalle **loop**
 instructions séquentielles;
end loop;

while condition **loop**
 Instructions séquentielles;
end loop;

next label_de_boucle **when** condition --arrête l'itération en cours

exit label_de_boucle **when** condition

zero detector

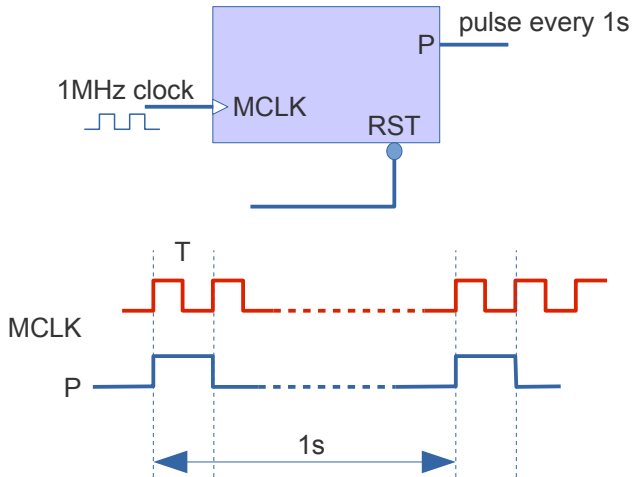
```
signal reg:std_logic_vector(7 downto 0);
.....
process( reg )
    variable zero:std_logic := '1';
begin
    zero:='1';
    for i in reg'range loop
        if reg(i)/='0' then
            zero:='0';
            exit;
        end if;
    end loop;
    if zero='1' then ...
    end if;
end process;
```

... with vector comparison

```
if reg/=conv_std_logic_vector(0,reg'length) then
    .....
end if;
```


Exercises

Pulse generator



```
entity pulse_gen is
  port( RST,MCLK:in std_logic;
        P:out std_logic );
end pulse_gen;
```

Whenever a reset occurs (i.e. $RST = '0'$), system resets itself. Normal operation restarts when reset is no more active.

Retrieve and edit the following files:

- pulse_gen.vhd → entity + architecture of your component
- test_pulse_gen.vhd → test architecture.

Compilation:

```
ghdl -a --ieee=synopsys -fexplicit \
pulse_gen.vhd test_pulse_gen.vhd
```

Elaborate:

```
ghdl -e --ieee=synopsys -fexplicit test_pulse_gen
```

Run:

```
ghdl -r --ieee=synopsys -fexplicit test_pulse_gen \
--wave=test_pulse_gen.ghw
```

Visualisation:

```
gtkwave test_pulse_gen.ghw
```

Note: google [ghdl makefile](#) to ease things a bit.

● log2 function

Allows you to get the number of bits required to encode a value:

$\log_2(1)=0$
 $\log_2(2)=1$
 $\log_2(4)=2$
 $\log_2(8)=3$

```
function log2(V:natural) return natural is
begin
    .....
end log2;
```

VHDL basics

● D flip-flop

What's wrong here ??

```
entity basculeD is
port(
  D, CLK: in std_logic;
  Q: inout std_logic;
  Qb: out std_logic);
end basculeD;
architecture beh1 of basculeD is
begin
  process
  begin
    wait until clk'event and clk='1';
    Q <= d;
    Qb <= not Q;
  end process;
end beh1 ;
```

```
entity basculeD is
port(
```

```
  D, CLK: in std_logic;
  Q: inout std_logic;
  Qb: out std_logic);
```

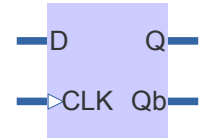
```
end basculeD;
architecture beh2 of basculeD is
Begin
```

```
  Qb <= not Q; process001
```

```
  process process002
```

```
  begin
    wait until clk'event and clk='1';
    Q <= d;
  end process;
```

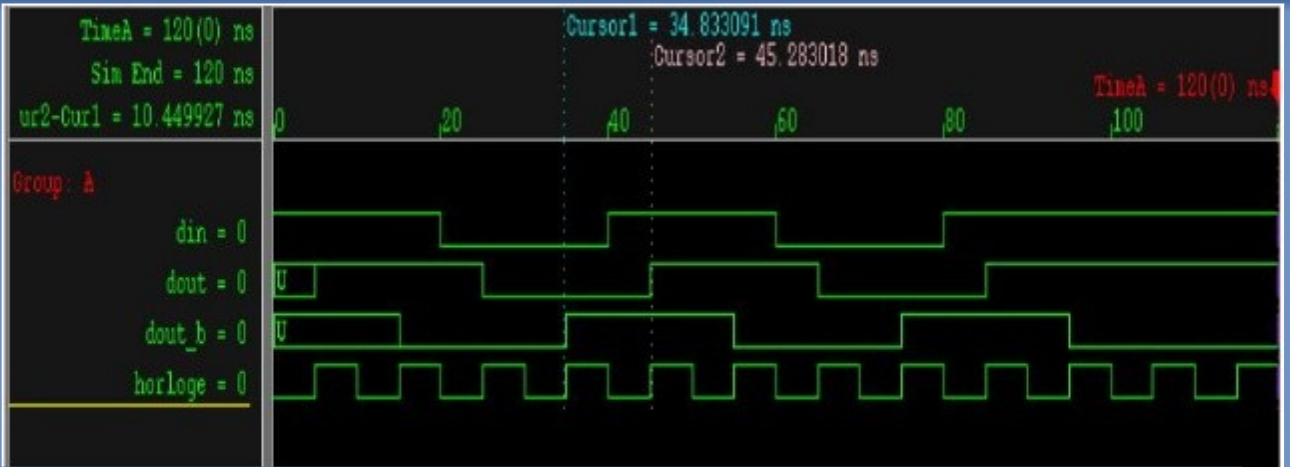
```
end beh2 ;
```



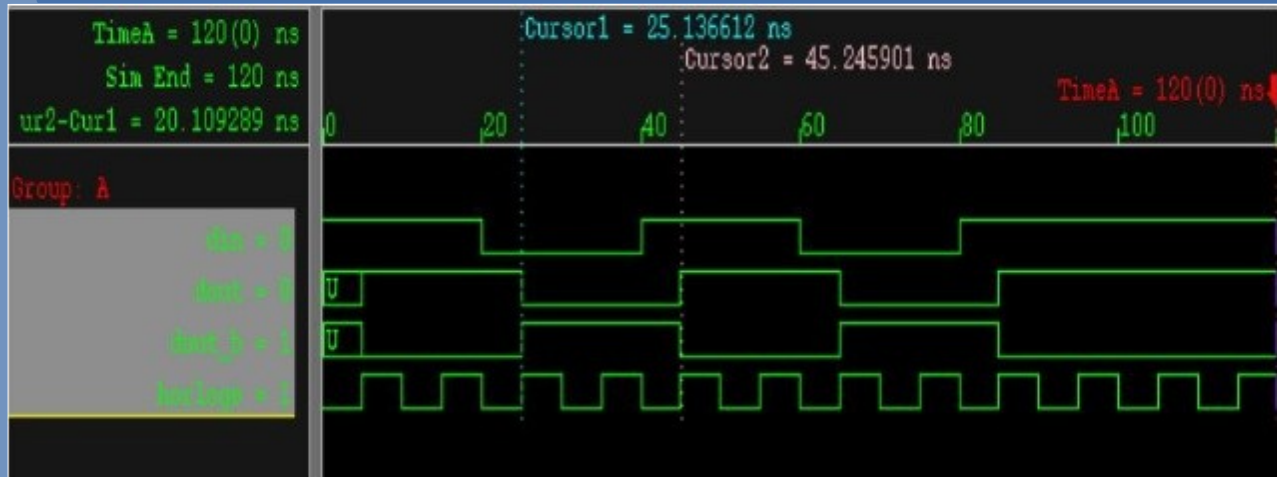
really ??

VHDL basics

beh1 architecture



beh2 architecture



VHDL basics

● D flip-flop – **Reset** featured

asynchronous

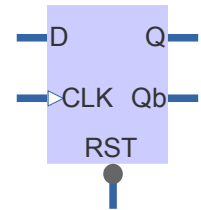
```
entity basculeD is
  port(
    D, CLK: in std_logic;
    Q: out std_logic;
    Qb: out std_logic );
end basculeD;
```

```
architecture beh3 of basculeD is
begin
  process( RST, CLK )
  begin
    if RST='0' then
      Q <= '0'; Qb <= '1';
    elsif rising_edge( CLK ) then
      Q <= d;
      Qb <= not D;
    end if;
  end process;
```

synchronous

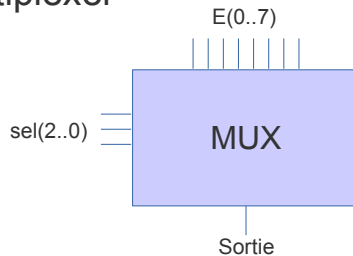
```
entity basculeD is
  port(
    D, CLK: in std_logic;
    Q: out std_logic;
    Qb: out std_logic );
end basculeD;
```

```
architecture beh3 of basculeD is
begin
  process( CLK )
  begin
    if rising_edge( CLK ) then
      if RST='0' then
        Q <= '0'; Qb <= '1';
      else
        Q <= d;
        Qb <= not D;
      end if;
    end if;
  end process;
```



Exercises

8 bits multiplexer



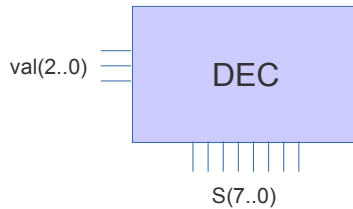
```
entity mux is
  port (
    sel : in integer range 0 to 7 ;
    entree : in std_logic_vector(0 to 7);
    Sortie : out std_logic);
end mux;
architecture beh of mux is
begin
  with sel select -- instruction de choix concurrente
    sortie <= entree(0) when 0,
              entree(1) when 1,
              entree(2) when 2,
              entree(3) when 3,
              entree(4) when 4,
              entree(5) when 5,
              entree(6) when 6,
              entree(7) when 7;
end beh;
```

Ré-écrivez ce multiplexeur en utilisant les affectations conditionnelles :

- dans le domaine combinatoire,
- puis dans un process.

Exercises

3 → 8 decoder



val	S 0	S 1	S 2	S 3	S 4	S 5	S 6	S 7
0	1							
1		1						
2			1					
3				1				
4					1			
5						1		
6							1	
7								1



Only one output activated for each input value ... **now it's up to you !**

VHDL basics

● Généricité

une entité est générique, jamais une architecture.

permet de définir une famille de composants par rapport à une caractéristique donnée.

valeur fixée lors de l'instance de l'entité

```
entity mydecoder is
  generic ( size: natural:=4 );
  port ( S: out STD_LOGIC_VECTOR (size-1 downto 0);
        val: in STD_LOGIC_VECTOR( _____ downto 0) );
end myadder;
```

default value

Instanciation du composant :

```
dec1 : entity work.mydecoder(behaviour)
  generic map (8)
  port map (val=> xxx, S=> yyy);
```

guess what to set here ??

● Généricité – instruction concurrente **generate**

Permet l'élaboration itérative ou conditionnelle de lignes de code

2 formes : conditionnelle et itérative

Forme conditionnelle

```
label : if condition_booléenne generate  
    suite d'instructions concurrentes  
end generate {label};
```

Forme itérative

```
label : for nom_du_paramètre_de_la_génération in intervalle discret generate  
    suite d'instructions concurrentes  
end generate {label};
```

● Généricité

Exemple: registre à décalage **générique (N)**

entrées Data et Clock - std_logic;

sortie S est un std_logic_vector (N-1 downto 0)

architecture structurelle en utilisant un composant de type bascule D

```
entity reg_decalage is
```

```
  GENERIC( N: natural :=8 );
```

```
  port ( Data, clock : in std_logic;
```

```
         S : out std_logic_vector(N-1 downto 0) );
```

```
end reg_decalage;
```

● Généricité

```
library work;  
use work.basculeD.all;
```

```
ARCHITECTURE structurelle OF reg_decalage IS
```

```
    for all : basculeD use entity work.basculeD(beh3);
```

```
begin
```

```
    gauche : basculeD port map (data, clock, S(N-1), open);
```

```
    boucle : for i IN 1 to N-1 generate
```

```
        circ : basculeD PORT MAP (S(N-i), clock, S(N-i-1), open);
```

```
    END GENERATE boucle;
```

```
END structurelle ;
```

What's wrong here ??

● Généricité

```
library work;  
use work.basculeD.all;
```

```
ARCHITECTURE structurelle OF reg_decalage IS
```

```
    signal aux : std_logic_vector (N-1 downto 0);
```

```
    for all : basculeD use entity work.basculeD(beh3);
```

```
begin
```

```
    gauche : basculeD port map (data, clock, aux(N-1), open);
```

```
    boucle : for i IN 1 to N-1 generate
```

```
        circ : basculeD PORT MAP (aux(N-i), clock, aux(N-i-1), open);
```

```
    END GENERATE boucle;
```

```
    S <= aux;
```

```
END structurelle ;
```

Now it's ok :)

● Généricité

```
ARCHITECTURE beh OF reg_decalage IS
```

```
begin
```

```
    comport : process
```

```
    begin
```

```
        wait until clock'event and clock ='1';
```

```
        s(n-1) <= Data ;
```

```
        copie : For i IN n-1 to 1 LOOP
```

```
            s(n-i-1) <= s(n-i);
```

```
        end loop copie;
```

```
    end process comport ;
```

```
END beh;
```

What's wrong here ??

● Généricité

ARCHITECTURE beh **OF** reg_decalage **IS**

signal aux : std_logic_vector (N-1 downto 0);
begin

s <= aux ;

comport : **process**

begin

wait until clock'event and clock ='1';

aux(n-1) <= Data ;

copie : **For** i **IN** n-1 **to** 1 **LOOP**

aux(n-i-1) <= aux(n-i);

end loop copie;

end process comport ;

END beh;

Now it's ok :)

● Compilation, élaboration, exécution, exploitation

Utilisation du VHDL :

- compilation et éditions de liens
 - aspects statiques de la description
- élaboration
 - aspects paramétrables de la description;
 - effectue l'instanciation de la description VHDL;
 - si la structure est hiérarchique il faut commencer par le plus haut niveau
- exécution
 - pour un simulateur \Leftrightarrow simulation
 - pour un synthétiseur \Leftrightarrow la synthèse
- exploitation des résultats plus spécifique de la simulation

VHDL basics

concurrent domain

```
Q <= value1 when condition else  
  value2;
```

affectation conditionnelle

sequential domain

```
if condition then  
  Q <= value1;  
else  
  Q <= value2;  
end if;
```

boucles

```
for indice in intervalle generate  
  affectations concurrentes;  
end generate;
```

```
for indice in intervalle loop  
  instructions séquentielles;  
end loop;
```

```
while condition loop  
  Instructions séquentielles;  
end loop;
```




END

strong empathy for hardware

perseverance

soldering sessions

a bit of electronics

ghdl + gtkwave

secret ingredient*

linux



path to YOUR success!

**Po's father knows it!*