**CONCEPTION DES CIRCUITS PROGRAMMABLES**

Conception des circuits programmables

# Introduction

Les applications spécifiques en conception électronique font appel de plus en plus à la logique programmée : microprocesseurs, microcontrôleurs, les composants programmables…

2 grandes familles de circuits logiques programmables :

* PROM
* PLD

# Les différentes familles de PROM

Mémoires mortes

ROM prog par masque PROM prog par l’utilisateur

 PROM a fusibles EPROM OPTOM ou

EPROM prog une seule fois

UVPROM EEPROM ou E2PROM EPROM flash

## Principe d’une PROM à fusible

Structure à :

- ET fixes (les circuits de décodage d’adresse)

- OU programmables (les données placées dans la mémoire)

* Lorsque la mémoire est livrée, tous les fusibles sont intacts. Elle ne contient alors que des 1
* La programmation va consister à faire sauter les fusibles aux emplacements ou on souhaite mémoriser des 0 en utilisant une haute tension (généralement 12 Volts), pour faire sauter le ou les fusibles.
* Les PROM à fusibles sont en voie de disparition.

Fusible métallique ou Si

Lignes métalliques : interconnexions

 Intact "Programmé"

Même structure en ligne poly silicium

Ou transistor "grillable" joue le rôle de fusible

Remarque : antifusible

Antifusible : au début on a que des 0, et on grille pour avoir des 1

## Principe des UVPROM

Grille de contrôle

Isolants

Grille isolée

* Chaque cellule mémoire élémentaire est constituée d'un transistor MOS dont la grille de commande est totalement isolée dans une couche d'oxyde.
* Par application d'une tension suffisamment élevée, qui est appelée tension de programmation, on crée des électrons chauds ou électrons ayant une énergie suffisante pour passer au travers de cet isolant.
* Ces charges s'accumulent alors sur cette grille isolée où elles se trouvent piégées. La cellule mémoire est programmable.

Un rayon UV décharge la PROM (les photons font franchie les électrons dans le sens inverse).

## Principe de EEPROM

* Ce sont des PROM effaçables électriquement octet par octet si nécessaire.
* Les EPROM flash sont effaçables globalement et sont plus rapides et plus simples (1 CMOS par bit) que les EEPROM.

Toutes deux sont programmables en circuit

|  |  |  |  |
| --- | --- | --- | --- |
| **Nom** | **Reprogrammable** | **Volatile** | **Technologie** |
| Fuse | Non | Non | Bipolaire |
| EPROM | Oui | Non | UVCMOS |
| EEPROM | Oui | Non | EECMOS |
| SRAM | Oui | Oui |  |
| Antifuse | Non | Non |  |

# Différentes familles de circuits logiques programmables

## Quelques définitions

EEPROM/E2PROM : Mémoire programmable à lecture seule, effaçable électriquement.

EPLD : Circuits logiques reprogrammables.

FPGA : Réseau des portes programmables.

GAL : Circuits logiques PAL reprogrammables à technologie CMOS.

ISP : Circuits que l'on peut programmer même lorsqu'il est en place sur l'application.

PAL : Circuits logiques programmables dans lesquels seules les fonctions ET sont programmables, pas les fonctions OU.

PLD : Famille des circuits programmables qui comprend les PAL, GAL, EPLD et FPGA.

## Différentes familles de PLD

PAL : bipolaires non effaçable : petite taille 10 à 100 (nombre de porte).

EPLD : circuits logiques effaçables : taille moyenne 1000 à 5000.

FPGA : réseaux de portes programmables : grande taille + de 5000.

### Les PAL

* Ils possèdent des matrices ET programmables, et des matrices OU fixes.
* La fusion des fusibles est obtenue en appliquant à leur bornes une tension de 11.5V pendant 10 à 50 s (leur tension de fonctionnement est de 5V).
* Cette opération est sûre effectuée en utilisant un programmateur adapté.

Structure de base avec les normes des constructeurs

Q0

Q1

b

a

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

x

PLD : programmé :

Q0 = 

Q1 = 

X : fusible intact

Configuration des entrées/sorties

* Certaines broches de ce circuit peuvent être utilisées aussi bien en entrées qu'en sortie grâce à un système de logique 3 états.
* La commande de cette dernière est configurée au moment de la programmation.
* La structure de sortie permet aussi de réinjecter les sorties en entrées (feed-back).

Référence des PAL

|  |  |
| --- | --- |
| L | Combinatoire active bas |
| H | Combinatoire active haut |
| C | Combinatoire active complémentaire |
| R | Registre synchrone |
| RA | Registre asynchrone |
| X | OU exclusif |
| Y | **????** |

* Nombre d'entrées : entre 10 et 22
* Nombre de sorties : entre 1 et 10
* La puissance est indiquée par 1 lettre code
* La vitesse indique le temps de propagation en ms

Exemple : PAL 16 L 8 H 15 PC

### Les GAL

* Les GAL sont des PAL à technologie CMOS, c'est à dire programmables et surtout effaçables électriquement.
* On retrouve les mêmes références qu'en PAL.

Protection contre la duplication

Les GAL sont dotés d'un bit de sécurité (empêchant la lecture du contenu du circuit). Il sont constitué de 8 octets appelés signature qui contiennent des infos sur le produits.

### Les EPLD

* Densité d'intégration supérieur aux PAL.
* Fonctionner à une vitesse au moins égale aux PAL bipolaire.

EPLD : description fonctionnelle

EPLD de la famille MAX :

* Logic Array broches(LABs)
* Macro cellules
* Expanseur
* Réseaux d'Interconnections Programmables (PIA)
* I/O control blocks

Les séries MAX incluent des entrées dédiées telles que des "horloges","clear",… pour chaque macro cellule.

### Les FPGA (Field Programmable Gate Arrays)

L'architecture, retenue par Xilinx, se présente sous forme de 2 couches :

* Une couche appelée circuit configurable.
* Une couche réseau mémoire SRAM.

La couche dite circuit configurable est constituée d'une matrice de blocs logiques configurables CLB permettant de réaliser des fonctions combinatoires et des fonctions séquentielles.

La SRAM

* La configuration du circuit est mémorisée sur la couche réseau SRAM et stockée dans une ROM externes
* Un dispositif interne permet à chaque mise sous tension de charger la SRAM interne à partir de la ROM

Programmation des PLDs

1. Cahier des charges.
2. Mise en équation.
3. Résolution du problème sous forme d'équation logique de logigramme ou d'algorithme.
4. Choix du PLD en fonction du nombre d'entrée(s) et de sortie(s).
5. Saisie des équations logiques, de la table de vérité, du logigramme ou de l'algorithme avec le logiciel.
6. Simplification logique.
7. Génération d'un fichier au format JEDEC.
8. Simulation.
9. Programmation du PLD à l'aide du fichier JEDEC et du programmateur.
10. PLD programmé.

Les étapes 6, 7 et 8 sont effectuée par le logiciel.

Graphe d'états

* Abel (langage de première génération)
* Verilog
* VHDL

V : VHSIC : Very High Intergrated Circuit

H : Harware

D : Description

L : Language

ASIC : Application Specific Integrated Circuits (Prédiffusés/Précaractérisé/Full custom)

* Les ASIC répondent à un besoin unique
* Il est réalisé par les fondeurs
* Ce système offre dans certains cas, la solution la plus rentable à la réalisation d'un cahier des charges.

Prototype : FPGA, PLD

En grand nombre : ASIC

* Prédiffusés
	+ Circuit génériques au "catalogue"
	+ Développement/production rapides.
	+ Pas d'utilisation optimale du Si
* Précaractérisés (standart cells) :
	+ Rien dans le Si
	+ Les cellules sont développées, caractérisées (géométrie électrique) accessible par leur schéma.
	+ Schéma +HDL
	+ Synthèse logique ou instanciations basés sur les ressources en BD fondeur.
	+ Permet le développement de circuits mixtes
	+ Envoi d'une description des masques au fondeur
* A la demande (Full custom)
	+ Liberté totale : optimisation complète.
	+ Quasiment plus utilisé en numérique
	+ Reste d'actualité en analogique
	+ Très long à mettre au point

Comparaison de coûts pour les séries

Coût

Nombre de pièces

Technologie 1

Technologie 3

Technologie 2

 PLD(1) Rediff(2) Précaractérisation(3)

Total ASIC = 70%

1. Synthèse logique du VHDL

# Structure d'un programme en VHDL

## Introduction

 Entity1 Entity2

P1

P2

P3

S1

S2

S3

E1

E2

Instanciation

Une description VHDL est composée de deux parties indissociables à savoir l'entité qui définit les signaux en E/S et permet de reconnaître lors des instanciations les différences entres les différents blocs

L'architecture contient les instructions VHDL, et elle est décrite(comportementale). Déclaration des bibliothèque : nommées par l'IEEE, elles contiennent les définitions des types de signaux électroniques, des fonctions et des sous programme utiles à la définition de l'architecture.

Library IEEE;

USE IEEE.std\_logic\_1164.all;

USE IEEE.numeric\_std.all;

USE IEEE.std\_logic\_unsigned.all;

La directive "USE" permet de sélectionner les bibliothèques à utiliser.

Remarque : le logiciel "activ HDL" de ALS Design permet d'aller chercher directement les bibliothèques nécessaires.

## La déclaration de l'entité

L'entité permet de définir le nom de la description VHDL, les entrées, les sorties et l'instruction qui est défini est PORT

Entity nom\_de\_l'entite is

Port (description des signaux);

End nom\_de\_l'entite ;

Remarque : pour "description des signaux" on note :

 Nom\_du\_signal : sens type ;

Exemple : Une bascule

Clock

 D

RESET

Q

Bus de 2 bits

Entity bascule is

Port ( CLOCK : in std\_logic;

 D : in std\_logic;

 RESET : in std\_logic; 0 to 1

 Q : out std\_logic\_vector(1 downto 0));

End bascule;

* Le nom du signal est compose de caractères : les premier caractère est une lettre, VHDL n'est pas sensible à la casse (t/T)
* Le sens du signal :
	+ in = signal d'entrée
	+ out = signal de sortie
	+ inout : signal en entrée/sortie
* le type du signal :

les types prédéfinis par le langage VHDL sont assez nombreux et le langage permet la création de ses propres types ou sous types. Pour le moment, on va utiliser std\_logic pour un signal et std\_logic\_vector pour un bus.

Exemple : Faire un bus de 5 bits

CCP : inout std\_logic\_vector(4 downto 0);

CCP(4) est le MSB

CCP(0) est le LSB

Un signal de type std\_logic peut prendre 4 valeurs :

'0' ou 'L'

'1' ou 'H'

'Z' haute impédance

'-' indifférent

Quand on a un bit on le met entre cote (')

Plus d'un bit on le met entre guillemets (")

Les déclarations d'architecture

L'architecture décrit le fonctionnement souhaité pour un circuit ou pour une partie du circuit. Dans un circuit, il y aura plusieurs couples entre entités affectées. Une architecture peut décrire une fonction combinatoire, séquentielle ou les deux.

Architecture nom\_de\_l'architecture of nom\_de\_l'entite is

Begin

 … instructions … ;

End nom\_de\_l'architecture;

Exemple : Réaliser le programme VHDL d'une porte NAND

 Entity (NAND)

a

b

S

On donne : - l'affectation : <=

 - les opérateurs logiques : and, not

Library ieee;

USE iee.std\_logic\_1134.all

Entity NAND is

Port ( a : in std\_logic;

 b : in std\_logic;

 s : out std\_logic );

End NAND;

Architecture desc\_NAND of NAND is

Begin

S <= not (a and b);

End desc\_NAND;

# Les instruction de base du mode concurrent : logique combinatoire

Dans toute description VHDL, toutes les instructions sont évaluées en même temps et affectent les signaux de sortie en même temps également. L'ordre dans lequel les instructions sont écrites n'a aucune importance puisque VHDL génère les structures électroniquement ce qui le différencie d'un langage informatique classique.

## Les opérateurs

### L'affectation : <=

Elle permet de modifier l'état d'un signal en fonction d'autres signaux

Exemple : S <= '0';

 S2 <= "001";

Binaire : BUS <="010";

Hexadécimal : BUS <= X "0AF";

Octal : BUS <= O "11";

### Opérateur de concaténation : &

Ex : A <= "001";

 B <= "101";

 C <= A & B & "01" *C <= 00110101*

### Opérateurs logiques

|  |  |  |
| --- | --- | --- |
| **Opérateur** | **VHDL** | **Explication** |
| ET | **AND** |  |
| NON ET | **NAND** |  |
| OU | **OR** |  |
| NON OU | **NOR** |  |
| OU EXCLUSIF | **XOR** |  |
| NON OU EXCLUSIF | **XNOR** |  |
| NON | **NOT** |  |
| DECALAGE A GAUCHE | **SLL** | Remplacement par des 0 |
| DECALAGE A DROITE | **SRL** | Remplacement par des 0 |
| ROTATION A GAUCHE | **ROL** | Le caractère de gauche est mis à droite |
| ROTATION A DROITE | **ROR** | Le caractère de droite est mis à gauche |

Les opérateurs logiques ont priorités sur la concaténation.

Exercice : A <= "1010011"

 Donner le résultat des opérateurs suivants :

B <= "01" & A SLL 2 *B <= 011001100*

C <= A SRL 3 & "11" *C <= 000101011*

D <= A ROL 4 *D <= 0111010*

E <= AROR 2 & "00" *E <= 111010000*

### L'opérateur arithmétique

Le langage VHDL utilise les opérateurs arithmétiques classiques (+, -, \*, /), cependant pour les utiliser il faut utiliser les bibliothèques numériques suivantes :

USE ieee.numeric\_std.all

USE ieee.std\_logic\_arith.all

Remarque : l'utilisation de ces opérateurs avec des signaux de grandes tailles peut générer de grandes structures électronique.

### Opérateurs relationnels

Il permet de modifier l'état d'un signal ou de signaux suivant le résultat d'un test ou d'une condition. En logique combinatoire, ils sont souvent utilisés avec les instructions when, select, with.

|  |  |
| --- | --- |
| **Opérateur** | **VHDL** |
| Egal | **=** |
| Différent | **/=** |
| Inférieur | **<** |
| Inférieur ou égal | **<=** |
| Supérieur | **>** |
| Supérieur ou égal | **>=** |

## Les instructions du mode concurrent

### Affectation conditionnelle

Signal <= expression when condition

 else expression when condition

 else expression when condition

 else expression ;

Remarque : la dernière ligne "else expression" n'est pas obligatoire mais elle est fortement conseillée.

Exercice : Réaliser le programme VHDL d'un multiplexeur 4 vers 1 avec entrée de sélection

E1 (1bit)

E2

E3

E5

S (1 bit)

SEL

|  |  |
| --- | --- |
| **SEL** | **S** |
| 00 | **E1** |
| 01 | **E2** |
| 10 | **E3** |
| 11 | **E4** |

Library ieee;

USE iee.std\_logic\_1134.all

Entity multiplex is

Port ( E1,E2,E3,E4 : in std\_logic;

 SEL : in std\_logic\_vector (1 downto 0);

 S : out std\_logic );

End multiplex;

Architecture desc\_multiplex of multiplex is

Begin S <= E1 when SEL = "00"

 Else E2 when SEL = "01"

 Else E3 when SEL = "10"

 Else E4;

End desc\_multiplex;

On aurais pu écrire aussi :

Library ieee;

USE iee.std\_logic\_1134.all

Entity multiplex is

Port ( E : in std\_logic\_vector (3 downto 0);

 SEL : in std\_logic\_vector (1 downto 0);

 S : out std\_logic );

End multiplex;

Architecture desc\_multiplex of multiplex is

Begin S <= E(0) when SEL = "00"

 Else E(1) when SEL = "01"

 Else E(2) when SEL = "10"

 Else E(3);

End desc\_multiplex;

### Affectation sélective

Cette instruction permet d'affecter différentes valeurs à un signal selon les valeurs prises par un signal de sélection

With signal\_de\_selection select

 signal <= expression when valeur\_de\_selection

 expression when valeur\_de\_selection

 expression when others;

Exercice : Réaliser le programme d'un multiplexeur 1 vers 4 avec entrée de sélection SEL et sortie de validation Enable active sur l'état bas

S1 (1bit)

S2

S3

S5

|  |  |
| --- | --- |
| **SEL** | **S** |
| 00 | **S1** |
| 01 | **S2** |
| 10 | **S3** |
| 11 | **S4** |

Enable

SEL

E (1 bit)

Library ieee;

USE iee.std\_logic\_1134.all

Entity demultiplex is

Port ( SEL : in std\_logic\_vector (1 downto 0);

 E : in std\_logic

 Enable : in std\_logic;

 S1,S2,S3,S4 : out std\_logic;);

End demultiplex;

Architecture desc\_demultiplex of demultiplex is

With Enable select

 S1 <= E when 0

???

???

End desc\_demultiplex;

## Les instructions du mode séquentiel

### Définition d'un process

Un process est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement. L'exécution d'un process est déclenché par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans la liste de sensibilité lors de la déclaration du process.

[nom\_du\_process :] Process ( liste\_de\_sensibilite )

 Begin

 … instruction\_du\_process … ;

 End process;

Le nom du process est facultatif.

### Règle de fonctionnement d'un process

#### L'exécution d'un process

* L'exécution d'un process a lieu à chaque changement d'état d'un signal de la liste de sensibilité
* L'instruction du process s'exécute séquentiellement
* Les changements d'état des signaux par les instructions du process sont prises en compte à la fin du process
* 2 process d'une architecture ayant la même liste de sensibilité vont être exécutés en même temps

Exemple :

Process (C,D)

Begin

 A <= 2 ;

 B <= A + C ;

 A <= D + 1 ;

 E <= A \* 2 ;

End process;

A = 1 A <= D + 1 ;

B = 1 B <= A + C ;

C = 1

D = 1

E = 1 E <= A \* 2 ;

On commence en mettant D à 2

On trouve les résultats suivants :

A = 3 B = 2 E = 2

Remarque : Les valeurs ne sont affectées qu'après la fin du process

#### Les principales structures du mode séquentiel

* A l'intérieur d'un process, on peut utiliser des "if" et/ou des "case". L'exécution est alors séquentielle.
* De même on peut utiliser des boucles pour contrôler l'exécution. Les "for" sont à bornes fixes, alors que les "while" ne le sont pas.

##### Le fonctionnement du "If"

Syntaxe : *If condition then instructions ;*

 *Elsif condition then instructions ;*

 *Else instructions ;*

 *End if ;*

Exemple du démultiplexeur 1 vers 4

|  |  |
| --- | --- |
| **SEL** | **S** |
| 00 | **S1** |
| 01 | **S2** |
| 10 | **S3** |
| 11 | **S4** |

SEL

E (1 bit)

S1 (1bit)

S2

S3

S5

Enable

Library ieee;

USE iee.std\_logic\_1134.all

Entity demultiplex is

Port ( SEL : in std\_logic\_vector (1 downto 0);

 E : in std\_logic

 Enable : in std\_logic;

 S1,S2,S3,S4 : out std\_logic;);

End demultiplex;

Architecture desc\_demultiplex of demultiplex is

Begin

 Process (SEL, ENABLE, E) --liste de sensibilité

 Begin

 If SEL ="00" and ENABLE = '0' then S1 <= E ;

 Elsif SEL ="01" and ENABLE = '0' then S2 <= E ;

 Elsif SEL ="10" and ENABLE = '0' then S3 <= E ;

 Elsif SEL ="11" and ENABLE = '0' then S4 <= E ;

 Else S1<='Z', S2<='Z', S3<='Z', S4<='Z' ;

 End if;

 End process ;

End desc\_demultiplex ;

##### Le fonctionnement du "Case"

Syntaxe : Case signal de selcteion is

 When valeur\_de\_selection => instruction;

 When valeur\_de\_selection => instruction;

 …

 Whenothers => instruction;

 End case;

Même exercice avec case :

Library ieee;

USE iee.std\_logic\_1134.all

Entity demultiplex is

Port ( SEL : in std\_logic\_vector (1 downto 0);

 E : in std\_logic

 Enable : in std\_logic;

 S1,S2,S3,S4 : out std\_logic;);

End demultiplex;

Architecture desc\_demultiplex of demultiplex is

Begin

 Process (SEL, ENABLE, E) --liste de sensibilité

 Begin

 If ENABLE = '0' then

 Case SEL is

 When "00" => S1 >= E ;

 When "01" => S2 >= E ;

 When "10" => S3 >= E ;

 Whenothers S4 >= E ;

 End case

 Else S1<='Z', S2<='Z', S3<='Z', S4<='Z' ;

 End if ;

 End process ;

End desc\_demultiplex ;

##### Le fonctionnement du "for"

Syntaxe : FOR parametre IN intervalle (Ex : 0 to 5) LOOP

instruction;

 End LOOP;

Exemple de codage de données :

Le cahier des charges est le suivant :

 s0 = a0

 Pour i > 0 si = ai . ai-1 + ai . ai-1

ENTITY codage IS

 a : IN std\_logic\_vector (3 downto 0) ;

 s : OUT std\_logic\_vector (3 downto 0) ;

END codage

Architecture archi of codage is

Begin

 Process

 Begin

 s(0) <= a(0);

 FOR I IN 1 to 3 LOOP

 s(i)<=(a(i) AND a(i-1)) OR ((NOT a(i) AND NOT a(I-1));

 END LOOP;

 END process ;

END archi;

##### Le fonctionnement du "wait"

* Il est possible de définir un process sans liste de dépendance. Chaque process est toujours exécuté au moins une fois, au début.
* En ajoutant des énoncés "wait", il devient possible d'indiquer que le process sera réveillé à un certain endroit, selon une certaine condition :
* Wait for time
* Wait on signal
* Wait until condition
1. Le VHDL

# Les classes d'objets

Il y a trois classes d'objets utilisées dans les zones de déclarations des programmes :

* Signaux
* Variables
* Constantes

Notion de signal

* Représentation matérielle du support de l'information en VHDL
* L'affectation de signal (a <= '0') permet de faire évaluer au cours du temps les valeurs présentes sur le signal.

Remarques : le signal se déclare sans l'architecture, les variables elle se déclarent dans le process

Exemple d'affectation de signal

a <=1 after 4ns, 0 after 8 ns, 1 after 12 ns;

Attention les expressions doivent être ordonnées selon un ordre croissant des délais.

 2 4 6 8 10 12 14 temps (ns)

a

b

s1

s2

z

Il faut maintenant trouver b, s1, s2 et z

b <= 1, 0 after 2ns, 1 after 6ns, 0 after 10ns ;

s1 <= not (a);

s2 <= not (b);

z <= 1, 0 after 2ns, 1 after 4ns, 0 after 6ns, 1 after 8ns, 0 after 10ns, 1 after 12ns ;

## Propriété des signaux

* Chaque signal à 3 propriétés : son type, sa valeur et son histoire temporelle.
* Les signaux sont déclarées dans les entités et la partie déclarative des architecture.
* Attention on ne peut pas les déclarer dans un process.
* Les signaux ne reçoivent leur assignation définitive que lorsque le process a fini son exécution.

## Les variables

Les variables se déclarent dans le domaine séquentiel d'un process, d'une procédure ou d'un fonction

Syntaxe :

Variable var\_name {, var\_name} : TYPE [ := value ] ;

Exemple

Variable i : INTEGER range 0 to 3 ; -- valeur initiale 0

Variable x : std\_ulogic ; -- valeur initiale 0

i := 3 ;

x := 0 ;

Remarque : une affectation de variable est immédiate dans un process

Exercice

Process (C,D)

Begin

 Av := 2 ; Av = 2

 Bv := Av + C ; Bv = 2+1 = 3

 Av := D + 1 ; Av = 2+1 = 3

 Ev := Av \* 2 ; Ev = 3\*2 = 6

 A <= 2 ; A = 3

 B <= A + C ; B = 3

 E <= A \* 2 ; E = 6

End process;

A = 1

B = 1

C = 1

D = 1

E = 1

On commence en mettant D à 2

Résultat

A = 3 B = 3 E = 6

## Les constantes

* Les constantes sont utilisées pour référencer une valeur ou un type spécifique.
* Elles permettent une meilleure lecture et la maintenant d'une source.
* De tous type, elles sont utilisées dans les architectures ou les packages.

Syntaxe :

Constant const\_name {, const\_name} : TYPE [ := value ] ;

Exemple

Constant BUS\_WIDTH : INTEGER := 8 ;

La déclaration generic

La section "GENERIC\_DECLARATION" dans l'entête "ENTITY" permet de définir des paramètres exploitables dans l'architecture. Cette méthode d'écriture permet un maintenance plus aisée.

Exemple :

ENTITY entity\_name IS

GENERIC (TP : time := 30 ns) ;

Port (declarations) ;

END [entity\_name] ;

# Les délais

VHDL permet de modéliser différents types de délais, qui sont utiles lors des simulations :

Délais d'inertie :

* Il agit comme un filtre, c'est à dire que les impulsions de durée inférieure à celle indiquée sont supprimées.
* Celles qui sont égales ou supérieures sont retardées de cette valeur. Utile pour tenir compte du temps de montée/descente.
* Ex : *sum <= inertial (a xor b) after 5ns ;*

Le mot inertial est facultatif

Délais d'inertie :

* Il représentent le délai encouru par les interconnections
* Utile pour tenir compte des délais RC dans les longs fils
* Ex : *sum <= transport (a xor b) after 5ns ;*

# Typage des objets

Le typage des objets permet en premier lieu de protéger les affectations d'objets en restreignant les possibilités d'affectations aux objets du même type.

## Les types prédéfinis

Les types prédéfinis reconnaissent six objets scalaires ayant une valeure unique à chaque instant :

* Bit
* Bit\_vector
* Boolean
* Real
* Integer
* Character

## Le type énuméré

Ce type permet de définir un type utilisateur auquel on peut assigner un objet.

Tous les objets doivent être assignés à un type.

Chaque type doit être unique.

Syntaxe :

Type identifier is (enumeration\_type\_literals) ;

Exemple

Type couleur is (rouge, vert, jaune) ;

Type op\_type is (opadd, opor, opand, opxor) ;

Type letters is ('A','a','R','r') ;

## Le type numérique

Cette définition de type doit être dans une échelle de valeur. La base par défault est le décimale.

Syntaxe :

Type identifier is Range implementation\_defined ;

Exemple

Type byte is Range 0 to 255 ; -- byte variant de 0 à 255

Type index is Range 7 downto 0 ; -- index variant de 7 à 0

## Le type physique

Cette définition de type représente une quantité physique. La base des unités doit être spécifiée. Les valeurs sont calculées à partir de cette base.

Syntaxe :

Type identifier is Range implementation\_defined ;

Units

Base\_unit\_declaration;

[secondary\_unit\_declaration]

End units;

Exemple

Type time is Range implementation\_defined ;

Units

fs;

ps = 1000 fs;

ns = 1000 ps;

us = 1000 ns;

ms = 1000 us;

sec = 1000 ms;

min = 60 fs;

hr = 60 min;

End units;

## Le type tableau

Le type tableau est un groupement de types identiques. Les tableaux peuvent être multidimensionnels.

Il existe deux sortes de tableaux :

* Tableau avec contraintes : à dimensions figées (rangées colonnes)
* Tableau sans contraintes : sans préciser les dimensions (dynamique)

Syntaxe :

Type identifier is Array ;

[unconstrained\_array\_definition] ;

[constrained\_array\_definition] ;

Exemple

Type data\_bus is Array (0 to 31) of bit; --tableau de 32 bits

Type bit4 is Array (3 doxnto 0) of bit; --tableau de 4 bits

Type string is Array (positive RANGE <>) of character;

Type dim2 is Array (0 to 7,0 to 7) of bit; --tableau à 2 dim

Type bit\_vector is Array (natural range <>);

## Le type trois états

Std\_logic et std\_logic vector

## Les pointeurs

Ils ne sont pas synthétisables du fait de leur caractère dynamique. Ils sont uniquement affectable sur les variables et constante pas les signaux. Ils sont strictement réservés au domaine fréquentiel.

Syntaxe :

Type pointeur is Access typeobjet ;

Exemple

Variable pt : pointeur ;

Pt : = new typeobjet ; -- allocation dynamique d'un objet en mémoire

La libération du pointeur se fait avec : dellocate (pt)

## Les structures : RECORD

Les structures permettent de regrouper un ou plusieurs champs de type différents et fournissent un accès au champs par leur nom. Un type record peut être associé à une variable comme un signal.

Exemple :

TYPE data is RECORD

Bus1, bus2, bit ;

Duree : time ;

Donnees : integer range 0 to 255 ;

End record ;

Signal enregistrement1 : data ;

enregistrement1 . bus1 <= '1' ;

enregistrement1 <= ('0', '1', duree, donnees)

## les sous types : SUBTYPE

SUBTYPE est une restriction d'un type à un sous domaine de ce type.

Exemple :

TYPE color is (red, yellow, blue, green, black) ;

SUBTYPE primary IS colors RANGE red TO blue ;

TYPE integer is range –2147483647 to 2147483647;

SUBTYPE absolu IS integer 0 TO 2147483647;

# Les entités (ENTITY)

EXERCICE :

Réaliser un compteur synchrone comptant sur front montant d'horloge (en entrée l'horloge, en sortie un compteur sur 4 bit)

L'instruction de front montant est réalisé par l'attribut 'event :

CLOCK = '1' and CLOCK 'event

S(3)

S(2)

S(1)

S(0)

CLOCK

Entity compteur is

Port (CLOCK : in std\_logic ;

 S :inout std\_logic\_vector (3 downto 0)) ;

End compteur;

Architecture cbo of compteur is

Begin

 Process (CLOCK)

 Begin

 If (CLOCK = '1' and CLOCK 'event)

 Then S <= S + 1;

 End if ;

 End process ;

End cbo;