Chapitre 5

Opérateurs Câblés :

Les opérateurs câblés sont des composants électroniques qui effectuent des opérations mathématiques et logiques sur des signaux numériques. Ils sont souvent utilisés pour réaliser des calculs et des traitements dans les circuits électroniques. Les opérateurs câblés courants incluent les portes logiques (ET, OU, NON, etc.), les additonneurs, les soustracteurs, les multiplexeurs, les démultiplexeurs, les comparateurs, etc.

5.1 Opérateurs Câblés :

Représentations des Nombres Relatifs : Les nombres relatifs sont utilisés pour représenter des valeurs positives et négatives dans les systèmes numériques. Les principales représentations des nombres relatifs comprennent :

5.1.1 Binaire Décalé :

Utilisé pour représenter des nombres signés en utilisant un bit de signe et des bits pour la valeur absolue.

5.1.2 Signe et Valeur Absolue :

Un bit est utilisé pour indiquer le signe (0 pour positif, 1 pour négatif) et les bits restants représentent la magnitude. Complément à Un : La valeur négative est obtenue en inversant tous les bits d'un nombre binaire positif.

5.1.3 Complément à Deux :

La valeur négative est obtenue en inversant tous les bits d'un nombre binaire positif et en ajoutant 1.

5.1.4 Représentation à Virgule Fixe :

Utilisée pour représenter des nombres réels en fixant un nombre fixe de bits pour la partie entière et la partie fractionnaire.

5.1.5 Représentation à Virgule Flottante :

Utilisée pour représenter des nombres réels avec une mantisse et un exposant, similaire à la notation scientifique.

5.1.6 Opérateurs Mathématiques :

Les opérateurs câblés sont utilisés pour réaliser des opérations mathématiques sur les nombres binaires. Les opérateurs courants incluent les additionneurs (pour l'addition), les soustracteurs (pour la soustraction), les multiplieurs (pour la multiplication), les diviseurs (pour la division), les comparateurs (pour la comparaison de nombres), etc.

Additionneurs :

Les additionneurs, tels que l'additionneur complet, sont utilisés pour ajouter deux nombres binaires. Ils prennent en compte les retenues pour effectuer une addition correcte.

Multiplieurs :

Les multiplieurs effectuent la multiplication de deux nombres binaires. Il existe différentes architectures pour les multiplieurs, telles que le multiplicateur "boîte" ou le multiplicateur "arbre".

Diviseurs :

Les diviseurs effectuent la division de deux nombres binaires. Les divisions peuvent être à virgule fixe ou à virgule flottante.

Comparateurs :

Les comparateurs comparent deux nombres et indiquent si l'un est plus grand, plus petit ou égal à l'autre. Les opérateurs câblés sont essentiels dans les systèmes numériques pour effectuer des calculs et des opérations logiques, que ce soit dans les processeurs, les FPGA, les ASIC ou d'autres circuits électroniques. Ils permettent de réaliser des fonctions complexes en manipulant des signaux binaires de manière précise.

5.2 Structure d'un programme

5.3 Structure d'une description VHDL simple

Une description VHDL est un texte en langage VHDL qui décrit la structure et le comportement d'un circuit numérique. Voici les éléments de base d'une description VHDL simple : L'entité (entity) : c'est la définition du module à implémenter, qui inclut le nom, les entrées et les sorties. L'architecture : c'est une description du comportement interne du module, qui définit les signaux internes et les processus. Les signaux : ce sont des variables qui représentent les valeurs logiques (0 ou 1) ou les niveaux de tension du circuit. Les processus : ce sont des blocs de code qui décrivent le comportement du circuit en réponse aux signaux d'entrée. Les assignations : ce sont des instructions qui permettent de modifier la valeur d'un signal. Voici un exemple de description VHDL simple pour un module qui compare deux entrées et renvoie la plus grande valeur : entity comparateur is

```
port (
a, b : in std logic vector(7 downto 0);
c : out std logic vector(7 downto 0)
); end comparateur;
architecture behav of comparateur is
begin
process (a, b)
begin
if a > b then
c <= a;
```

```
else
c <= b;
end if;
end process;
end behav;
```

Dans cet exemple, l'entité "comparateur" est définie avec deux entrées "a" et "b" de 8 bits, et une sortie "c" de 8 bits. L'architecture "behav" décrit le comportement interne du module en utilisant un processus qui compare les entrées et assigne la valeur maximale à la sortie "c".

5.4 Entité

En VHDL, l'entité (entity) est une structure qui décrit les caractéristiques d'un module ou d'un composant numérique. Elle définit les entrées (ports d'entrée) et les sorties (ports de sortie) du module, ainsi que ses caractéristiques telles que le type de signal ou le type de données. Voici un exemple simple d'une entité VHDL qui décrit un module avec deux entrées et une sortie :

```
entity mon module is
port (
entree1 : in std logic;
entree2 : in std logic;
sortie1 : out std logic
);
end mon module;
```

Dans cet exemple, l'entité "mon module" est définie avec deux entrées "entree1" et "entree2" de type std logic (c'est à dire un signal logique), et une sortie "sortie1" de type std logic. L'entité est définie dans une structure de blocs et se termine avec le mot clé "end" suivi du nom de l'entité. Les ports d'entrée et de sortie définis dans l'entité peuvent être utilisés dans l'architecture pour décrire le comportement interne du module. Les ports peuvent également être utilisés pour connecter le module à d'autres modules ou composants dans le schéma global du système.

5.5 Les différentes descriptions d'une architecture (de type flot de données, comportemental ou procédural, structurel et architecture de test)

En VHDL, une architecture décrit le comportement interne d'une entité (module) définie précédemment. Il existe plusieurs types d'architectures que l'on peut utiliser pour décrire le comportement d'un module : Architecture de type flot de données : cette architecture décrit le comportement du module en termes de flux de données et de traitement des données. Elle utilise généralement des expressions mathématiques pour décrire le comportement du module. Architecture comportementale : cette architecture décrit le comportement du module en termes de fonctions logiques et d'opérations. Elle utilise généralement des instructions logiques (if/then/else, case) pour décrire le comportement du module. Architecture procédurale : cette architecture décrit le comportement du module en termes de processus ou de blocs de code. Elle utilise généralement des processus VHDL pour décrire le comportement du module. Architecture structurelle : cette architecture décrit le comportement du module en termes de composants structurels tels que des portes logiques, des multiplexeurs, des registres, etc. Elle utilise des instances de composants pré-définis pour décrire le comportement du module. Architecture de test : cette architecture est utilisée pour simuler et tester le comportement du module. Elle utilise généralement des signaux de test pour vérifier que le module fonctionne correctement. Chaque type d'architecture a ses avantages et ses inconvénients, et il est important de choisir la bonne architecture pour décrire le comportement du module en fonction des exigences du système.

5.6 Process

En VHDL, un process est une structure de blocs qui est utilisée pour décrire le comportement d'un module. Un process peut être utilisé dans une architecture procédurale pour décrire le comportement d'un module en termes de blocs de code et de séquences d'instructions. Un process est défini par le mot clé "process" suivi d'une liste de signaux (sensibles au changement), qui indique les signaux qui déclenchent le process. Le corps du process est défini à l'aide d'un bloc de code entre les mots clés "begin" et "end process".

Voici un exemple simple de process qui décrit un module avec deux entrées et une sortie :

```
architecture exemple of mon module is
begin
process (entree1, entree2)
begin if (entree1 = '1' and entree2 = '1') then
sortie1 <= '1';
else
sortie1 <= '0';
end if;
end process;
end exemple;
```

Dans cet exemple, un process est utilisé pour décrire le comportement du module "mon module". Le process est défini pour être déclenché par les signaux "entree1" et "entree2". Le comportement du module est défini à l'intérieur du process à l'aide d'une instruction conditionnelle (if/else) qui teste les valeurs des entrées et met à jour la sortie en conséquence. Le process est un outil puissant pour décrire le comportement d'un module en VHDL, car il permet de décrire le comportement du module en termes de séquences d'instructions et de blocs de code. Cependant, il est important de noter que l'utilisation de process doit être judicieuse pour éviter des problèmes de timing et de fiabilité dans les conceptions complexes.

5.7 Les structures de contrôle en VHDL

En VHDL, il existe plusieurs structures de contrôle qui permettent de décrire le comportement d'un module. Ces structures sont utilisées pour contrôler le flux d'exécution du code et pour effectuer des opérations répétitives. Voici quelques exemples de structures de contrôle en VHDL :

Structure conditionnelle :

La structure conditionnelle en VHDL est utilisée pour exécuter des instructions en fonction de l'évaluation d'une expression logique. Elle est définie à l'aide des mots-clés "if", "elsif" et "else". Voici un exemple de structure conditionnelle :

```
if (condition1) then
```

instruction1 ;
elsif (condition2) then
instruction2 ;
else
instruction3 ;
end if ;
Dans cet exemple, les instructions sont exécutées en fonction de la valeur de la condition1 et de la condition2. Structure de boucle :
La structure de boucle en VHDL est utilisée pour effectuer une série d'instructions de manière répétitive. Il existe deux types de boucles en VHDL : la boucle "for" et la boucle "while". Voici un exemple de boucle "for" :
for i in 0 to 10 loop
instruction1 ;
end loop ;
Dans cet exemple, l'instruction1 est exécutée 11 fois (pour i allant de 0 à 10).
Structure de séquence :
La structure de séquence en VHDL est utilisée pour exécuter une série d'instructions dans un ordre spécifique. Elle est définie à l'aide des mots-clés "sequential" et "end sequential". Voici un exemple de structure de séquence :
sequential
instruction1 ;
instruction2 ;
end sequential ;
Dans cet exemple, l'nstruction1 est exécutée avant l'instruction2.
Structure de process :
La structure de process en VHDL est utilisée pour décrire le comportement d'un module. Elle est définie à l'aide des mots-clés "process", "begin" et "end process". Les instructions à l'intérieur du process sont exécutées séquentiellement. Voici un exemple de structure de process :
process (entree1, entree2)
begin
if (entree1 = '1' and entree2 = '1') then

```
sortie1 <= '1';
else
sortie1 <= '0';
end if;
end process;
```

Dans cet exemple, le process est déclenché par les signaux entree1 et entree2. Les instructions à l'intérieur du process sont exécutées en fonction de la valeur de ces signaux. En utilisant ces différentes structures de contrôle, il est possible de décrire le comportement d'un module de manière précise et efficace en VHDL.

5.8 Instructions séquentielles et concurrentes

En VHDL, il y a deux types d'instructions : les instructions séquentielles et les instructions concurrentes. Les instructions séquentielles sont exécutées les unes après les autres, dans l'ordre où elles ont été écrites dans le code. Les instructions séquentielles sont exécutées dans un process ou une architecture séquentielle. Les instructions concurrentes sont exécutées simultanément. Elles sont utilisées pour décrire la structure du circuit et pour décrire les relations entre les différents éléments du circuit. Les instructions concurrentes sont exécutées dans une architecture concurrente. Voici un exemple de code VHDL avec des instructions séquentielles et concurrentes :

```
entity exemple is
port (
clk : in std logic;
a : in std logic vector(3 downto 0);
b : in std logic vector(3 downto 0);
c : out std logic vector(3 downto 0)
);
end exemple;

    architecture Seq of exemple is
signal d : std logicvector(3 downto 0);
begin
```

```
process (clk)
begin
if rising edge(clk) then
d <= a + b;
end if;
end process;


    c <= d;
end Seq;
architecture Conc of exemple is
begin
c(0) <= a(0) and b(0);
c(1) <= a(1) or b(1);
c(2) <= not a(2);
c(3) <= b(3);
end Conc;
```

Dans cet exemple, l'architecture Seq utilise un process pour exécuter une instruction séquentielle (l'addition de a et b) et une instruction concurrente (l'affectation de c à d). L'architecture Conc utilise des instructions concurrentes pour décrire la relation entre a, b et c.

5.9 Les paquetages et les bibliothèques

En VHDL, un paquetage est une collection de types, de constantes, de fonctions et de procédures qui peuvent être utilisés dans plusieurs entités ou architectures. Un paquetage est défini dans un fichier VHDL séparé et est inclus dans le code VHDL à l'aide d'une directive use dans la partie declarative du code.

Voici un exemple de code VHDL avec un paquetage : – Définition du paquetage package mon paquetage is

```
constant pi : real := 3.14;
function carre(x : integer) return integer;
end mon paquetage;
```

```
– Corps du paquetage
package body mon paquetage is
function carre(x : integer) return integer is
begin
return x*x;
end carre;
end mon paquetage;
– Utilisation du paquetage
entity exemple is
port (
clk : in std logic;
a : in integer;
b : out integer
);
end exemple;
architecture Behavioral of exemple is
use work.mon paquetage.all; – inclusion du paquetage
begin
b <= carre(a) + pi;
end Behavioral;
```

En VHDL, une bibliothèque est un conteneur pour les entités, les architectures et les paquetages. Chaque bibliothèque peut contenir plusieurs entités, architectures et paquetages. Les bibliothèques sont utilisées pour organiser les différents modules d'un projet VHDL. Par défaut, un projet VHDL dispose d'une bibliothèque standard appelée ieee qui contient les packages et les types de base nécessaires pour les projets VHDL. Il est possible de créer des bibliothèques supplémentaires pour organiser les modules d'un projet.

Voici un exemple de création et d'utilisation d'une bibliothèque en VHDL :

```
– Création d'une bibliothèque
library mon lib;
use mon lib.mon paquetage.all;
– Définition d'une entité dans la bibliothèque
entity exemple is
```

```
port (
clk : in std logic;
a : in integer;
b : out integer
);
end exemple;
– Définition d'une architecture dans la bibliothèque
architecture Behavioral of exemple is
begin
b <= carre(a) + pi; – utilisation du paquetage inclus
end Behavioral;
– Utilisation de l'entité dans le code principal
library mon lib;
use mon lib.exemple.all;
entity top is
port (
clk : in std logic;
a : in integer;
b : out integer
);
end top; v architecture Behavioral of top is
begin
exemple inst : entity work.exemple v port map (
clk => clk,
a => a,
b => b
);
end Behavioral;
```

Dans cet exemple, une bibliothèque appelée mon lib est créée et un paquetage appelé mon paquetage est inclus dans cette bibliothèque. Une entité et une architecture sont définies dans cette bibliothèque. Dans le code principal, l'entité exemple de la bibliothèque mon lib est utilisée et instanciée dans l'architecture Behavioral de l'entité top.

5.10 Applications : Implémentation de quelques circuits logiques dans les circuits FPGA

L'implémentation de circuits logiques dans les circuits FPGA peut être effectuée à l'aide de divers outils de conception tels que Quartus II, Xilinx ISE ou Vivado. Voici quelques exemples de circuits logiques courants qui peuvent être implémentés dans les circuits FPGA : Additionneur : Un additionneur peut être implémenté dans un FPGA en utilisant des blocs logiques programmables tels que des blocs LUT (Look-Up Table) ou des blocs DSP (Digital Signal Processing). Le circuit peut être décrit en VHDL en utilisant des processus séquentiels ou concurrents pour l'ajout et la propagation des retenues.
Compteur : Un compteur est un circuit séquentiel qui peut être implémenté en utilisant des registres à décalage ou des compteurs binaires. La description VHDL du circuit doit inclure des signaux de comptage, des signaux de synchronisation et des signaux de sortie pour afficher la valeur du compteur.
Multiplexeur : Un multiplexeur peut être implémenté dans un FPGA en utilisant des blocs LUT ou des blocs de mémoire intégrée. La description VHDL doit inclure des signaux d'entrée, un signal de sélection et un signal de sortie pour le multiplexeur.
Décodageur : Un décodageur est un circuit combinatoire qui peut être implémenté en utilisant des blocs LUT ou des blocs de mémoire intégrée. La description VHDL doit inclure des signaux d'entrée pour le code binaire et des signaux de sortie pour les lignes de sortie activées.
Comparateur : Un comparateur peut être implémenté dans un FPGA en utilisant des blocs logiques programmables tels que des blocs LUT ou des blocs DSP. La description VHDL doit inclure des signaux d'entrée pour les valeurs à comparer et un signal de sortie pour indiquer si les valeurs sont égales ou non.
Ces exemples ne sont que quelques-uns des nombreux circuits logiques qui peuvent être implémentés dans les circuits FPGA. Le choix des blocs logiques programmables et la méthode de description VHDL dépendent des spécifications du circuit et des exigences de performance.

5.11 Compteur

Un compteur est un circuit qui permet de compter le nombre d'événements qui se produisent. Le compteur peut être synchrone ou asynchrone, binaire ou décimal, ascendant ou descendant, selon ses caractéristiques. Voici un exemple de description VHDL d'un compteur binaire synchrone 4 bits :

```
entity compteur binaire 4bits is port (
clk : in std ogic ;
reset : in std logic ;
compteur : out std ogic vector(3 downto 0)
) ;
end compteur binaire 4bits ;
architecture behavioral of compteur binaire 4bits is
signal compteur interne : std logic vector(3 downto 0) ;
begin
process(clk, reset)
begin
if reset = '1' then – Si reset est à 1, le compteur est réinitialisé
compteur interne <= (others => '0') ;
elsif rising edge(clk) then – Si un front montant de l'horloge se produit
compteur interne <= compteur interne + 1 ; – On incrémente le compteur
end if ;
end process ;
compteur <= compteur interne ; – On transmet la valeur du compteur en sortie
end behavioral ;
```

Dans cet exemple, l'entité compteur binaire 4bits a une entrée d'horloge clk, une entrée de réinitialisation reset et une sortie compteur de type std logic vector de longueur 4 bits. L'architecture behavioral décrit le comportement du compteur. La ligne signal compteur interne : std logic vector(3 downto 0) ; déclare une variable interne qui stocke la valeur du compteur. Le processus utilise la condition if/elsif pour réinitialiser le compteur lorsque reset est à 1, ou pour l'incrémenter à chaque front montant de l'horloge clk. Enfin, la ligne compteur <= compteur interne ; transmet la valeur du compteur en sortie. Cette description VHDL peut être synthétisée en utilisant un outil de conception FPGA pour

implémenter le circuit dans le FPGA en utilisant les blocs logiques programmables appropriés.

5.12 Registre à décalage

Un registre à décalage est un circuit qui permet de décaler les bits d'un mot binaire vers la gauche ou la droite. Voici un exemple de description VHDL d'un registre à décalage à 8 bits avec une entrée série, une sortie série et une entrée de commande de décalage :

```
entity registre decalage is port (
clk : in std logic;
reset : in std logic;
entree serie : in std logic;
commande decalage : in std logic;
sortie serie : out std logic
);
end registre decalage;
architecture behavioral of registre decalage is
signal registre interne : std logic vector(7 downto 0);
begin
process(clk, reset)
begin
if reset = '1' then – Si reset est à 1, le registre est réinitialisé
registre interne <= (others => '0');
elsif rising edge(clk) then – Si un front montant de l'horloge se produit if commande decalage = '1' then – Si la commande de décalage est à 1, on décale les bits vers la gauche
registre interne <= registre interne(6 downto 0) &entree serie;
else – Sinon, on décale les bits vers la droite
registre interne <= entree serie &registre interne(7 downto 1);
end if;
end if;
end process;
```

```
    sortie serie <= registre interne(0); – On transmet le bit de sortie en série
end behavioral;
```

5.13 Filtre simple

Un filtre simple peut être implémenté dans un circuit FPGA en utilisant une structure de type flot de données. Il peut s'agir, par exemple, d'un filtre passe-bas qui permet de laisser passer les basses fréquences et d'atténuer les hautes fréquences. Voici un exemple de description VHDL pour un tel filtre :

```
    library ieee;
use ieee.std logic 1164.all;
use ieee.numeric std.all;

    entity filtre is
port (
clk : in std logic;
reset : in std logic; input : in std logic;
output : out std logic
);
end entity;
architecture comportementale of filtre is
signal sample : std logic vector(7 downto 0) := (others => '0');
signal acc : signed(15 downto 0) := to signed(0, 16);
begin
process (clk)
begin
if rising edge(clk) then
if reset = '1' then
sample <= (others => '0');
```

```
acc <= to signed(0, 16);
else
sample <= input & sample(7 downto 1);
acc <= acc + signed(sample) - acc/16;
output <= '1' when acc(15) = '0' else '0';
end if;
end if;
end process;
end architecture;
```

5.14 Outils de programmation : Altera Quartus II, Modelsim, Xilinx ISE

Altera Quartus II, ModelSim et Xilinx ISE sont des outils de programmation largement utilisés pour la conception et la programmation de circuits FPGA et CPLD. Voici une brève présentation de ces outils : Altera Quartus II : il s'agit d'un environnement de développement intégré (IDE) pour la programmation de circuits FPGA et CPLD d'Altera/Intel. Quartus II permet de concevoir, simuler, synthétiser et programmer des circuits numériques en utilisant VHDL, Verilog ou un diagramme de blocs. ModelSim : c'est un simulateur de circuits numériques qui peut être utilisé avec Quartus II ou d'autres outils de conception de circuits. ModelSim permet de simuler et de déboguer des circuits numériques à différents niveaux d'abstraction, de l'entrée/sortie jusqu'aux signaux internes. Xilinx ISE : il s'agit d'un environnement de développement intégré pour la conception de circuits FPGA et CPLD de Xilinx. ISE permet de concevoir, simuler, synthétiser et programmer des circuits numériques en utilisant VHDL, Verilog ou un diagramme de blocs. Ces outils offrent des fonctionnalités avancées pour faciliter la conception de circuits numériques complexes. Ils disposent également de bibliothèques de composants prédéfinis, ce qui facilite la conception de circuits. En outre, ces outils offrent des fonctionnalités de débogage et d'optimisation de code pour améliorer les performances des circuits. En somme, Altera Quartus II, ModelSim et Xilinx ISE sont des outils de programmation puissants pour les concepteurs de circuits numériques. Ils permettent de

concevoir, simuler, synthétiser et programmer des circuits FPGA et CPLD, et ils offrent des fonctionnalités avancées pour faciliter la conception et l'optimisation des circuits.