ÉCOLE NATIONALE SUPÉRIEURE
de SCIENCES APPLIQUÉES et de TECHNOLOGIE

TECHNOPOLE ANTICIPA LANNION UNIVERSITÉ DE RENNES 1

PROJET VHDL
Réalisation d'un processeur

D. Chillet, D. Ménard et E. Kinvi-boh

Le 7 juillet 2004

ENSSAT - Technopole Anticipa - BP 447 - 6 rue de Kérampont - 22305 LANNION Cedex - Siret 193 509 361 00278 - APE 9215
Téléphone (33) 02 96 46 50 30 - Télécopie (33) 02 96 37 01 99 - Email ...@enssat.fr - Minitel 36 15 ENSSAT - Serveur WWW http://www.enssat.fr/

Table des matières

Chapitre 1

Tutorial V System VHDL

Ce premier chapitre a pour objectif de vous faire prendre en main un compilateur et un simulateur pour le langage VHDL. Il s'agit du logiciel *V_System* de *Model Technology* fonctionnant sur PC. Cet outil est aussi disponible sur station de travail dans l'environnement Mentor.

Après un bref rappel concernant le langage *VHDL*, ce tutorial présente la manière de créer un projet ainsi que la façon de reprendre un projet en cours. Il détaille ensuite les phases de compilation et de simulation de l'outil.

Ce tutorial est basé sur un certain nombre de fichiers qui nous serviront d'exemples lors des différentes phases de compilation ou de simulation. Ces fichiers sont donnés en annexe, mais vous les trouverez sur le site Internet de l'ENSSAT à l'adresse suivante :
`http://r2d2.enssat.fr/enseignements/Vhdl/Vhdl.php`

La première section de ce tutorial présente les différentes manipulations de l'outil V System.
La deuxième section vous guide dans la conception d'une première entité qui vous sera utile pour la suite du projet, l'entité considérée est l'unité arithmétique et logique de notre processeur Von Neumann (UAL).
Enfin la troisième section vous propose de modifier les descriptions qui vous sont fournies de façon à étendre les possibilités de votre UAL.

Bonne lecture et bons développements ...

1.1 Rappel : Qu'est ce que le langage VHDL ?

Prenons le temps de faire un tout petit rappel de ce qu'est VHDL ...

VHDL est un langage de description matériel. À la différence des langages informatiques classiques, VHDL ne vise pas une exécution procédurale, son but est de permettre la description de tout système électronique, d'en valider le fonctionnement avant de passer à la mise en œuvre matérielle.

Rappelons brièvement les trois types de description utilisables en VHDL :
- description comportementale : il s'agit d'une description indiquant le comportement d'un système. Généralement réalisée sous la forme de processus, elle s'apparente à du code procédural classique ;
- description structurelle : il s'agit d'une description schématique d'un système. S'appuyant sur des composants disponibles dans une bibliothèque et des signaux, cette description est l'exacte représentation du schéma électrique du système ;
- description flot de données : il s'agit d'une description indiquant comment un flot de données traverse un système. Le flot des sorties est exprimé en fonction du flot des entrées.

1.2 Manipulation de l'outil V System

L'outil ModelSim est disponible sur PC ainsi que sur station de travail sun. Sur PC, vous lancerez l'outil par le menu "Démarrer", sur station vous lancerez l'outil par la commande "vsim".

Lorsque l'on lance l'outil ModelSim, 2 cas existent, soit vous voulez débuter un nouveau projet, soit vous souhaitez reprendre un projet existant.

1.2.1 Débuter un projet (initialisation du projet)

Le lancement de l'exécutable *ModelSim* ouvre une fenêtre dans laquelle les menus décrits ci-dessous sont accessibles.
Cette fenêtre vous indique en permanence les commandes que vous avez lancées (avec la souris ou manuellement) et vous donne des informations sur la compilation (affichage des fichiers chargés lors de la compilation d'un système par exemple). C'est aussi dans cette fenêtre que sont indiquées les différentes erreurs dans le code VHDL de votre système.
Lorsque vous débutez votre projet, un certain nombre de paramètres est à définir afin que l'outil *V_System* organise correctement les différents fichiers que vous allez créer.

1. À l'aide de *Windows*, créez un répertoire de travail sur votre espace disque :
2. Placement dans le répertoire de travail sur votre espace disque par le menu :

   File —> Change directory

   Sélectionnez le répertoire que vous venez de créer.
3. Création d'une bibliothèque de travail :

   Design —> Create new Library

   Tapez le nom de la bibliothèque de travail *WORK*. C'est dans cette bibliothèque que seront placées toutes les entités compilées sans erreur.
4. Création d'un projet :

   File —> New —> New Project

   Tapez un nom de projet, par exemple `projet`.

Vous pouvez ensuite passer au développement de votre système et aux compilations des différentes spécifications.

1.2.2 Reprise d'un projet

Lorsque vous reprenez votre travail sur un projet que vous avez déjà initialisé, vous devez toujours spécifier deux choses avant de commencer toute nouvelle compilation ou simulation.
Si vous lancez la commande vsim dans le répertoire d'un projet exitant alors ModelSim ouvre ce projet.

1. Après le lancement du logiciel *ModelSim*, placez vous dans votre répertoire de travail sur votre espace disque par le menu :

   File —> Change directory

   Sélectionnez le répertoire de travail.
2. Placez-vous dans votre projet :

   File —> Open —> Open Project

   Sélection du projet.

   Recherchez le fichier `.mpl` qui correspond à votre projet (normalement, ce fichier doit être dans le répertoire que vous avez sélectionné par le menu précédent).

1.2.3 La compilation

Le lancement du compilateur s'effectue par le menu **Design —> Compile** **Compile** du logiciel. Une fenêtre s'ouvre alors et vous donne accès aux fichiers de votre répertoire de travail. Sélectionnez un fichier et cliquez sur **Compile.** La phase de compilation est alors activée.

Remarques

- Utilisez intelligemment les indentations et les commentaires pour que vos fichiers soient lisibles et facilement maintenables.
- Donnez l'extension **.vhd** à tous vos fichiers VHDL (fichiers de déclaration d'entités, d'architectures, de paquetages, de corps de paquetage, etc) ;
- Afin d'éviter une surabondance de fichiers dans votre répertoire de travail, nous vous conseillons de placer le corps d'un paquetage dans le même fichier que la spécification de ce même paquetage. De même, regroupez dans un même fichier, la déclaration d'entité (*entity, generic, port*) et son ou ses architectures.
- Écrivez un fichier par entité ou par paquetage.

Ordre de compilation de vos fichiers

L'ordre de la compilation de vos fichiers est important. Il faut toujours que toutes les unités utilisées (par l'unité à compiler) soient présentes dans la bibliothèque. Ce qui signifie que vous devez respecter l'ordre de compilation suivant :

1. compilation de vos paquetages. Si vous en avez plusieurs, compilez d'abord ceux qui n'utilisent aucun autre paquetage, ensuite vous compilerez les autres ;
2. compilation de vos entités feuilles. On entend par entité feuille, toute entité décrite de façon comportementale ;
3. compilation de vos entités décrivant un sous-système, c'est à dire des entités décrites sous la forme d'une structure de composants ;
4. compilation de votre entité système, c'est à dire de l'entité du plus haut niveau regroupant l'ensemble des sous-systèmes (reliés par des signaux) ;

Résultats de compilation

Dès que la compilation d'une unité est réalisée sans erreur, l'unité en question apparaît dans la bibliothèque. Vous pouvez la visualiser en choisissant le menu :

Design —> View Library Contents

Une fenêtre apparaît alors et vous indique tous les paquetages et entités présents dans la bibliothèque, et pour chaque entité vous pouvez obtenir la liste de toutes les architectures décrites et compilées sans erreur. À partir de cette fenêtre, vous pouvez supprimer un paquetage ou une entité ou une architecture.

Tests des unités réalisées

N'attendez pas que tout votre système soit réalisé pour effectuer des tests sur vos composants. La démarche idéale consiste à tester chaque entité dès que sa compilation est correcte. Pour cela, décrivez une entité de test telle que celle qui vous est donnée en annexe. Compilez alors cette entité et simulez la.

1.2.4 La simulation

Lancez une simulation

Pour lancer le simulateur, sélectionnez le menu **Design —> Load Design**

Une fenêtre est alors ouverte et vous propose l'ensemble des unités présentes dans la bibliothèque. Sélectionnez une entité SIMULABLE (ou une configuration), c'est à dire une entité autonome qui instancie l'entité à simuler et qui gère les signaux d'entrée de l'entité à simuler (pour l'exemple de l'UAL, l'entité à simuler est *TestUal*).

Ouvrez ensuite les fenêtres de simulation par le menu **View —> All** Les fenêtres suivantes sont alors ouvertes :

- fenêtre *Wave* : cette fenêtre permet la visualisation des chronogrammes des signaux de votre système en cours de simulation.
- fenêtre *Dataflow* :
- fenêtre *Source* : cette fenêtre permet la visualisation du code source de l'unité en cours de simulation (fenêtre très utile lors du débuggage, mais inacessible en édition) ;
- fenêtre *List* :
- fenêtre *Structure* : cette fenêtre donne une description semi-graphique de la structure de l'entité en cours de simulation ;
- fenêtre *Process* : cette fenêtre indique les processus actifs lors de la simulation ;
- fenêtre *Variables* : cette fenêtre affiche les valeurs des différentes variables contenues dans l'entité en cours de simulation ;
- fenêtre *Signals* : cette fenêtre affiche les valeurs des différents signaux contenus dans l'entité en cours de simulation

Utilisez judicieusement toutes les fenêtres qui sont mises à votre disposition pour valider votre système (utilisez notamment le fonctionnement en pas à pas).

Par défaut, l'intervalle de simulation est de $100\ ns$, vous pouvez changer cette valeur en choisissant le menu :

Options —> Simulation

Pour recommencer une simulation depuis le temps $0\ ns$, choisissez le menu :

Run —> Restart

Visualisation des chronogrammes

La fenêtre *Wave* vous permet de vérifier que votre système répond bien au cahier des charges qui vous a été défini. Par défaut, cette fenêtre n'affiche aucun signal, pour visualiser les signaux d'entrée/sortie de l'entité simulée, choisissez, dans la fenêtre signal, le menu :

View —> Wave —> Signal in Region

Pour visualiser tous les signaux de la hiérarchie contenus dans votre système, choisissez le menu :

View —> Wave —> Signal in Design

Lancez ensuite la simulation par le menu **Run —> Run 100 ns**

1.3 Conception d'une UAL

Dans ce paragraphe, nous allons détailler la description d'une unité arithmétique et logique ainsi que les phases de compilation et de simulation.

L'UAL que nous allons concevoir peut être représentée par le schéma de la figure 1.1.

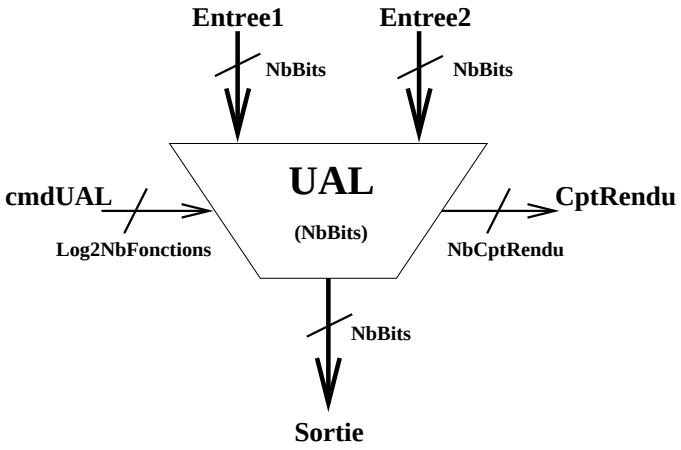

FIG. 1.1 – Vue externe de l'UAL

1.3.1 Initialisation du projet

- À l'aide de *Windows*, créez un répertoire VHDL (répertoire de travail) sur votre espace disque :
- Toujours à l'aide de *Windows*, créez un sous-répertoire *VonNeuman* dans le répertoire VHDL ;
- Placez les 4 fichiers concernant l'UAL dans votre répertoire de travail (`Z :/VHDL/VonNeuman`);
- Lancez le logiciel *ModelSim* :
- Choisissez le menu **File —> Change Directory** et sélectionnez le répertoire `Z :/VHDL/VonNeuman`;
- Choisissez le menu **Design —> Create New Library** et tapez `WORK` (nom de la bibliothèque de travail dans laquelle toutes vos unités seront stockées) ;
- Choisissez le menu **File —> New —> New Project** et tapez `projet` (nom du projet) ;
- Choisissez le menu **Design —> Compile** ;
- Une fois la fenêtre de compilation ouverte, sélectionnez le fichier `PackProj.vhd` et cliquez sur **Compile** ;

1.3.2 Compilation des unités fonctionnelles

La compilation doit être correcte, vérifiez cela dans la fenêtre *Principale* (pas de *warning* et pas d'*erreur*). Les messages affichés dans la fenêtre *Principale* doivent être les suivants :

```
vcom Z:/VHDL/VONNEUMAN/PACKPROJ.VHD
# - Loading package standard
# - Loading package std_logic_1164
# - Loading package std_logic_arith
# - Loading package std_logic_signed
# - Compiling package packproj
# - Compiling package body packproj
# - Loading package packproj
```

- Une fois la fenêtre de compilation ouverte, sélectionnez le fichier `UALPack.vhd` et cliquez sur **Compile** ;
  La compilation doit être correcte, vérifiez cela dans la fenêtre *Principale* (pas de *warning* et pas d'*erreur*). Les messages affichés dans la fenêtre *Principale* doivent être les suivants :

```
vcom Z:/VHDL/VONNEUMAN/UALPACK.VHD
# - Loading package standard
# - Loading package std_logic_1164
# - Loading package std_logic_arith
# - Loading package std_logic_signed
# - Loading package packproj
# - Compiling package ualpack
# - Compiling package body ualpack
# - Loading package ualpack
```

– Sélectionnez ensuite le fichier `UAL.vhd` et cliquez sur **Compile**, aucune erreur ne doit apparaître dans la fenêtre *Principale*. Les messages sont les suivants :

```
vcom Z:/VHDL/VONNEUMAN/UAL.VHD
# - Loading package standard
# - Loading package std_logic_1164
# - Loading package std_logic_arith
# - Loading package std_logic_signed
# - Loading package packproj
# - Loading package ualpack
# - Compiling entity ual
# - Compiling architecture comportementale of ual
```

– Sélectionnez enfin le fichier `UALT.vhd` et cliquez sur **Compile**, aucune erreur ne doit apparaître dans la fenêtre *Principale*. Les messages sont les suivants :

```
vcom Z:/VHDL/VONNEUMAN/UALT.VHD
# - Loading package standard
# - Loading package std_logic_1164
# - Loading package std_logic_arith
# - Loading package std_logic_signed
# - Loading package packproj
# - Loading package ualpack
# - Compiling entity testual
# - Compiling architecture test of testual
# - Loading entity ual
```

1.3.3 Simulation de l'entité TestUAL

– Choisissez ensuite le menu **Design —> Load Design** et choisissez l'entité de simulation `UALT`. La fenêtre *Principale* doit indiquer les messages suivants :

```
vsim
# Loading N:\ARCHITECT\VHDL97\std.standard
# Loading N:\ARCHITECT\VHDL97\ieee.std_logic_1164(body)
# Loading N:\ARCHITECT\VHDL97\ieee.std_logic_arith(body)
# Loading N:\ARCHITECT\VHDL97\ieee.std_logic_signed(body)
# Loading work.packproj(body)
# Loading work.ualpack(body)
# Loading work.testual(test)
# Loading work.ual(comportementale)
```

Ouvrez ensuite les fenêtres de simulation, **View —> All**.

– demandez l'affichage de tous les signaux de l'entité de test en sélectionnant **View —> Wave —> Signals in Region** ;

– Lancez un cycle de simulation en choisissant le menu **Run —> Run 100 ns**. Le cycle de simulation s'arrête au bout de $100\ ns$;

– Vérifiez le bon fonctionnement de l'UAL à l'aide des chronogrammes ;

Prenez le temps d'ouvrir l'ensemble des fenêtres qui vous sont proposées par V System, vous en aurez besoin par la suite pour la mise au point de vos descriptions. Vous observerez notamment le fonctionnement du simulateur lorsque vous effectuez une simulation en mode pas à pas.

1.4 Amélioration de l'UAL de base

L'UAL qui vous est proposée n'effectue qu'un faible nombre d'opérations (addition, et logique, nop).
Vous compléterez cette UAL de façon à ce qu'elle réalise les fonctions suivantes :

– soustraction ;

- multiplication ;
- ou logique ;
- nand logique ;
- nor logique ;
- xor logique.

Avant de vous lancer dans la modification des descriptions, prenez le temps de comprendre comment l'UAL fournie a été décrite. Nous avons volontairement supprimé les commentaires des descriptions, une première étape dans la compréhension du fonctionnement de cette UAL pourrait être de commenter de façon précise les 4 fichiers qui vous sont fournis.

Vous remarquerez que les descriptions que nous avons faites sont suffisamment génériques pour être facilement adaptables. Vous apporterez un soin tout particulier, dans la suite du projet, à l'aspect généricité de vos descriptions.

Chapitre 2

Réalisation d'une file de registres

L'objet de ce chapitre est de vous guider dans la réalisation d'une file de registres générique.
Cet élément de base pourra prendre place par la suite dans une unité de traitement de façon à constituer une partie d'un processeur par exemple.
Après la présentation du système à réaliser, ce document abordera la modélisation du système et vous guidera jusqu'à la conception et à la description des différents éléments qui le compose.

Vous aurez à réaliser les différentes entités du système, à les tester et à les instancier dans le schéma global de la file de registres.

La principale difficulté liée à cette file de registres tient à la façon d'assurer la généricité par rapport aux deux paramètres que sont le nombre de registres et le nombre de bits des registres. Ce document vous amènera à réaliser une description structurelle en utilisant la boucle `generate` du langage VHDL.

Ce chapitre est basé sur un certain nombre de fichiers que vous pourrez trouver sur le site Internet de l'ENSSAT à l'adresse suivante : `http://r2d2.enssat.fr/enseignements/Vhdl/Vhdl.php`

2.1 Présentation du système à concevoir

2.1.1 Cahier des charges

Le cahier des charges du système à concevoir est constitué des points suivants :
- le système doit pouvoir stocker des valeurs venant de l'environnement et restituer ces mêmes valeurs pour l'environnement ;
- la capacité de stockage du système sera limitée à `NbReg` valeurs ;
- chaque élément stocké aura une taille maximale égale à `NbBits` éléments binaires de type `Std_Logic` ;
- le système doit pouvoir fournir, à l'environnement, 2 opérandes simultanément en vue de la réalisation d'un calcul sur un opérateur binaire ;
- un élément particulier du système doit pouvoir être chargé avec une valeur particulière venant de l'environnement ;
- le système doit pouvoir supporter un transfert entre deux éléments internes sans que cela ne perturbe le fonctionnement de l'environnement ;

2.1.2 Remarques

L'ensemble de ces indications nous permet de faire une première analyse et d'aboutir aux remarques suivantes :
- le système sera constitué de 2 sorties (`Sortie1` et `Sortie2`) permettant de fournir 2 opérandes à l'environnement. Les valeurs à présenter sur les sorties seront codées à l'aide de 2 signaux `OutReg1` et `OutReg2`, et

on associera à chaque sortie un signal *OutPutEnable* (`Oe1` et `Oe2`) permettant de contrôler la mise en haute ou en basse impédance des 2 sorties correspondantes.

– la file de registres disposera d'une entrée qui permettra de charger une valeur dans un registre, le registre à charger sera spécifié par le signal `LoadReg` et un signal `Load` permettra de valider effectivement le chargement. Toutefois, le chargement sera effectivement réalisé au front montant d'horloge (système synchrone).
– la file de registres possède un signal `Transf` qui permettra la réalisation d'un transfert de registre à registre. Ce type de transfert sera réalisé de façon interne à la file de registres et placera les sorties du système dans un état de haute impédance. Le transfert s'effectuera entre le registre codé par la commande `OutReg1` et le registre codé par la commande `LoadReg`.

2.2 Vue externe du système – Description de l'entité RegisterFile

La vue externe de la file de registres est donnée par la figure ci-dessous.

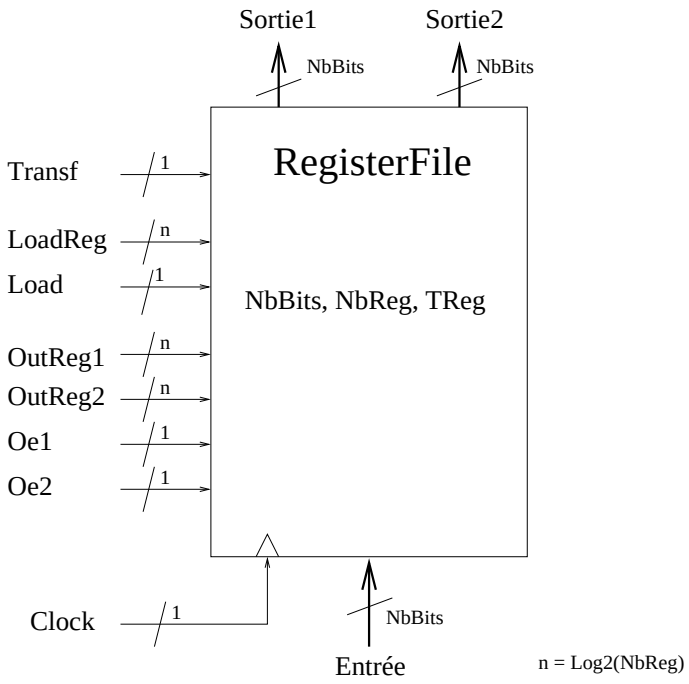

FIG. 2.1 – Vue externe de la file de registre

On en déduit la description de l'entité du système en VHDL :

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

USE work.PackProj.All;

ENTITY RegisterFile IS
        GENERIC (
                NbBits : INTEGER ;
                NbReg : INTEGER ;
                Log2NbReg : INTEGER ;
                TReg : TIME );
        PORT (
                Transf : IN Std_Logic ;
                LoadReg : IN Std_Logic_Vector (Log2NbReg - 1 DOWNTO 0) ;
                Load : IN Std_Logic ;
                OutReg1 : In Std_Logic_Vector (Log2NbReg - 1 DOWNTO 0) ;
                OutReg2 : In Std_Logic_Vector (Log2NbReg - 1 DOWNTO 0) ;
```

```
                Oe1 : IN Std_Logic ;
                Oe2 : IN Std_Logic ;
                Clock : IN Std_Logic ;
                Entree : IN Std_Logic_Vector (NbBits - 1 DOWNTO 0);
                Sortie1 : OUT Std_Logic_Vector (NbBits - 1 DOWNTO 0);
                Sortie2 : OUT Std_Logic_Vector (NbBits - 1 DOWNTO 0)
        );
END RegisterFile;
```

Listing 2.1 – Description de la vue externe de l'entité *Systeme*

Notons, dans cette description, que nous avons ajouté un générique supplémentaire `Log2NbReg`. En effet, le langage de description VHDL ne permet pas la déclaration d'un port en faisant appel à une fonction, déclaration du type :
`Load : IN Std_Logic_Vector (`*`Log2(NbReg)`*` -1 DOWNTO 0)`.
Donc il nous faut trouver un moyen de spécifier le nombre de bits de commande des signaux `LoadReg`, `OutReg1` et `OutReg2`. A priori inutile, ce générique est donc nécéssaire pour rendre l'entité générique.

Comme pour toute conception, nous débutons par une description comportementale de la file de registres. Cette description nous servira ensuite de référence pour la suite de la conception.

2.3 Vue interne du système — Description comportementale de RegisterFile

La première étape, avant d'entreprendre la conception, consiste à décrire le comportement du système. La phase suivante concernera la conception à proprement parlé et aboutira à l'obtention d'un schéma.
Il s'agit de spécifier le fonctionnement de ce boîtier en fonction des actions intervenants sur ses entrées. Pour réaliser cette description, nous allons utiliser un modèle comportemental et la notion de processus du langage VHDL.

```
ARCHITECTURE Comportementale OF RegisterFile IS

BEGIN
        PROCESS
                TYPE Tableau IS ARRAY(INTEGER RANGE 0 TO NbReg -1)
                        OF Std_Logic_Vector(NbBits - 1 DOWNTO 0) ;
                VARIABLE tab : Tableau;
                VARIABLE indice1, indice2, indice3 : INTEGER;
        BEGIN
                WAIT ON Clock, Oe1, Oe2, OutReg1, OutReg2;
                indice1 := CONV_POSITIF(OutReg1);
                indice2 := CONV_POSITIF(OutReg2);
                indice3 := CONV_POSITIF(LoadReg);
                IF Clock = '1' AND Clock'Event THEN

                        IF Load = '1' THEN
                                tab(indice3) := Entree;
                        END IF;
                        IF Transf = '1' THEN
                                tab(indice3) := tab(indice1);
                        END IF;

                END IF;
                IF Oe1 = '1' AND Transf = '0' THEN
                        Sortie1 <= tab(indice1) AFTER TReg;
                ELSE
```

```
                        Sortie1 <= (OTHERS =>'Z');
                END IF;

                IF Oe2 = '1'  AND Transf = '0' THEN
                        Sortie2 <= tab(indice2) AFTER TReg;
                ELSE
                        Sortie2 <= (OTHERS =>'Z');
                END IF;


        END PROCESS;
END Comportementale;
```

Listing 2.2 – Description de l'architecture comportementale de la file de registres

2.4 Test du système — Description de l'entité et de l'architecture de TestRegisterFile

Il s'agit de s'assurer que le système décrit permet bien de réaliser les fonctionnalités désirées. Pour cela nous allons effectuer une simulation du système. Cette simulation est assurée par un simulateur VHDL qui gèrera les événements des signaux d'entrées et affichera les niveaux (ou valeurs) des signaux de sortie.

Pour réaliser cette simulation, nous décrivons un composant de test. Pour tout composant à tester, nous procédons de la même manière, c'est à dire que nous décrivons un composant (une entité) n'ayant ni *generic* ni *port* d'entrées. Ce composant réalise une instanciation du composant à tester puis gère, sous la forme d'un processus, les niveaux (ou valeurs) des signaux d'entrées. L'avantage du composant de test par rapport à une simulation par fichier script du simulateur, est qu'il peut être repris sur n'importe quelle plate forme VHDL, ce qui n'est pas le cas du fichier script qui dépend étroitement du simulateur choisi.

Nous donnons ci-dessous la vue externe du fichier de test ainsi que son architecture alliant à la fois le comportemental et le structurel. Notons que les tests de l'entité `RegisterFile` ne sont que partiellement réalisés, vous compléterez ce test de façon à ce qu'il soit complet.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TestRegisterFile IS
END TestRegisterFile ;

ARCHITECTURE Test OF TestRegisterFile IS

        COMPONENT RegisterFile
                GENERIC (
                        NbBits : INTEGER ;
                        NbReg : INTEGER ;
                        Log2NbReg : INTEGER ;
                        TReg : TIME );
                PORT (
                        Transf : IN Std_Logic ;
                        LoadReg : IN Std_Logic_Vector (Log2NbReg - 1 DOWNTO 0) ;
                        Load : IN Std_Logic ;
                        OutReg1 : In Std_Logic_Vector (Log2NbReg - 1 DOWNTO 0) ;
                        OutReg2 : In Std_Logic_Vector (Log2NbReg - 1 DOWNTO 0) ;
                        Oe1 : IN Std_Logic ;
```

```
                        Oe2 : IN Std_Logic ;
                        Clock : IN Std_Logic ;
                        Entree : IN Std_Logic_Vector (NbBits - 1 DOWNTO 0);
                        Sortie1 : OUT Std_Logic_Vector (NbBits - 1 DOWNTO 0);
                        Sortie2 : OUT Std_Logic_Vector (NbBits - 1 DOWNTO 0)
                );
        END COMPONENT;

        SIGNAL s_entree : Std_Logic_Vector(7 DOWNTO 0) := (OTHERS => 'Z');
        SIGNAL s_sortie1 : Std_Logic_Vector(7 DOWNTO 0) := (OTHERS => 'Z');
        SIGNAL s_sortie2 : Std_Logic_Vector(7 DOWNTO 0) := (OTHERS => 'Z');

        SIGNAL s_loadreg : Std_Logic_Vector(3 DOWNTO 0) := (OTHERS => 'Z');
        SIGNAL s_out1 : Std_Logic_Vector(3 DOWNTO 0) := (OTHERS => 'Z');
        SIGNAL s_out2 : Std_Logic_Vector(3 DOWNTO 0) := (OTHERS => 'Z');

        SIGNAL s_load : Std_Logic := '0';
        SIGNAL s_transf : Std_Logic := '0';
        SIGNAL s_oe1 : Std_Logic := '0';
        SIGNAL s_oe2 : Std_Logic := '0';
        SIGNAL s_clock : Std_Logic := '0';

        FOR RegFile : RegisterFile USE ENTITY work.RegisterFile(Comportementale);
--      FOR RegFile : RegisterFile USE ENTITY work.RegisterFile(Structurelle);

BEGIN
        RegFile : RegisterFile
                GENERIC MAP (8, 16, 4, 10 ns)
                PORT MAP (
                        s_transf, s_loadreg, s_load, s_out1, s_out2,
                        s_oe1, s_oe2, s_clock, s_entree, s_sortie1, s_sortie2);

        ProcessHorloge : PROCESS
        BEGIN
                s_clock <= '0';
                WAIT FOR 50 ns;
                s_clock <= '1';
                WAIT FOR 50 ns;
        END PROCESS ProcessHorloge;

        ProcessSimulation : PROCESS
        BEGIN
                WAIT FOR 10 ns;

                s_entree <= "01010101";
                s_loadreg <= "1000";
                s_load <= '1' ;
                WAIT FOR 100 ns;

                s_entree <= "00000000";
                s_loadreg <= "0000";
                WAIT FOR 100 ns;

                s_entree <= "00000001";
                s_loadreg <= "0001";
                WAIT FOR 100 ns;

                WAIT ;
```

```
        END PROCESS ProcessSimulation ;
END Test;
```

Listing 2.3 – Description de l'entité de test : *TestRegisterFile*

2.5 Vue interne de la file de registres — Description structurelle de RegisterFile

Après une phase de conception, dont nous n'exposons pas les différents mécanismes et étapes qui permettent d'aboutir à une solution, nous aboutissons à une solution structurelle dont le schéma est donné à la figure 2.2.

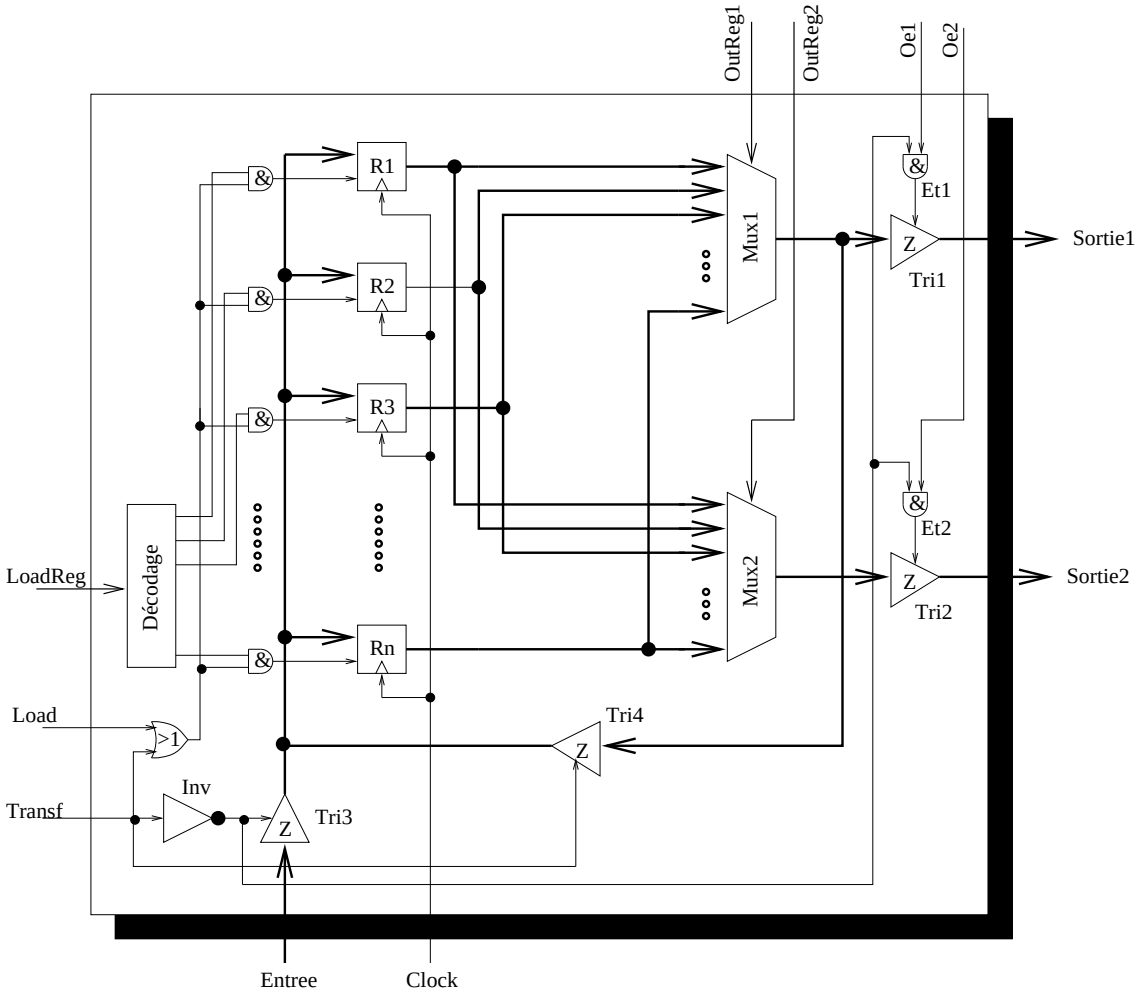

FIG. 2.2 – Vue interne de la file de registre

Dans ce schéma, nous observons qu'un motif de base est répété `NbReg` fois. Il s'agit du motif associant le registre et la porte ET logique. `NbReg` n'étant pas connu à la compilation, il nous faut utiliser un mécanisme permettant de reproduire le motif de façon automatique. VHDL propose une solution pour ce type de description, l'utilisation de la boucle `generate`. Le motif est instancié par ce type de structure VHDL. Les autres éléments du schéma, ne faisant pas partie du motif de base, seront placés de manière classique par des instanciations de composant (ou encore par un *mapping* classique).

Pour réaliser la description sous forme schématique de la file de registres, 2 démarches s'offrent à nous :
- démarche descendante : on effectue une description structurelle de l'architecture, puis on s'intéresse ensuite à la description des composants de base ;

– démarche ascendante : on modélise d'abord les différents composants de base en effectuant pour chacun d'eux un test complet, puis on effectue la description schématique de la file de registres.

2.5.1 Descriptions des éléments de base nécessaires

Nous allons procéder par une approche ascendante, cette approche a été largement illustrée dans le document *"De la description d'un système à la description des transistors"* qui se trouve sur le site Internet de l'ENSSAT à l'adresse suivante : `http://r2d2.enssat.fr/enseignements/Vhdl/Vhdl.php`

Ce schéma fait apparaître un certain nombre de nouveaux composants qu'il nous faut donc décrire. Nous donnons ci-dessous la liste des entités de base à réaliser ainsi que leur cahier des charges. Nous donnons aussi la vue externe des différentes entités (afin qu'il y ait une certaine homogénéité des noms des génériques et des ports dans vos descriptions).

– Entité `ET` : Cette entité possède 2 entrées (de type `Std_Logic`) et fournit une sortie (de type `Std_Logic`) correspondant à la fonction ET logique entre les 2 entrées.

```
ENTITY ET
        GENERIC (
                TEt : TIME );
        PORT (
                Entree1 : IN Std_Logic ;
                Entree2 : IN Std_Logic ;
                Sortie : OUT Std_Logic
        );
END ET;
```

Listing 2.4 – Description de l'entité ET

– Entité `OU` : Cette entité possède 2 entrées (de type `Std_Logic`) et fournit une sortie (de type `Std_Logic`) correspondant à la fonction OU logique entre les 2 entrées.

```
ENTITY OU
        GENERIC (
                TOu : TIME );
        PORT (
                Entree1 : IN Std_Logic ;
                Entree2 : IN Std_Logic ;
                Sortie : OUT Std_Logic
        );
END OU;
```

Listing 2.5 – Description de l'entité OU

– Entité `Inverseur` : Cette entité possède 1 entrée (de type `Std_Logic`) et fournit une sortie (de type `Std_Logic`) correspondant à la fonction NON logique de l'entrée.

```
ENTITY Inverseur
        GENERIC (
                TInv : TIME
        );
        PORT (
                Entree : IN Std_Logic ;
                Sortie : OUT Std_Logic
        );
END Inverseur ;
```

Listing 2.6 – Description de l'entité Inverseur

– Entité `TroisEtat` : Cette entité possède 1 entrée `Oe` (de type `Std_Logic`) qui permet de piloter la mise en haute impédance de la sortie (de type `Std_Logic_Vector`). Si le signal `Oe` est au niveau `1` alors la

sortie (de type `Std_Logic_Vector`) est la recopie de l'entrée (de type `Std_Logic_Vector`) sinon la sortie est en haute impédance.

```
ENTITY TroisEtats
        GENERIC (
                NbBits : INTEGER ;
                TTri : TIME );
        PORT (
                Entree : IN Std_Logic_vector(NbBits - 1 DOWNTO 0) ;
                Sortie : OUT Std_Logic_vector(NbBits - 1 DOWNTO 0) ;
                oe : IN Std_Logic
        );
END TroisEtats;
```

Listing 2.7 – Description de l'entité TroisEtats

– Entité `Registre` : Cette entité possède 1 entrée `Load` (de type `Std_Logic`) qui permet de piloter le chargement de la valeur présente sur l'entrée `Entree` (de type `Std_Logic_Vector`). Le chargement sera réalisé de façon synchrone avec le signal d'horloge `Clock` (de type `Std_Logic`). La valeur présentée sur la sortie `Sortie` (de type `Std_Logic_Vector`) est la valeur contenue dans le registre.

```
ENTITY Registre
        GENERIC (
                NbBits : INTEGER;
                TReg : TIME );
        PORT (
                Load : IN Std_Logic ;
                Clock : IN Std_Logic ;
                Entree : IN Std_Logic_Vector (NbBits -1 DOWNTO 0);
                Sortie : OUT Std_Logic_Vector (NbBits - 1 DOWNTO 0)
        );
END Registre;
```

Listing 2.8 – Description de l'entité Registre

– Entité `Mux` : Cette entité réalise le multiplexage des entrées (de type `Std_Logic_Vector`) sur une sortie de type `Std_Logic_Vector`. Il existe une difficulté pour décrire cette entité. Dans une première approche, on pourrait imaginer de déclarer les entrées du système dans un tableau à 2 dimensions. La dimension 1 correspondrait au nombre de registres `NbReg` alors que la dimension 2 correspondrait au nombre de bits de chaque registre, `NbBits`. On décrirait alors le système de la façon suivante :

```
 TYPE Tableau IS ARRAY (INTEGER RANGE <>) OF Std_Logic_Vector ;
 ENTITY Mux
        GENERIC (
                NbBits : INTEGER ;
                Log2NbIn : INTEGER ;
                TMux : TIME );
        PORT (
                Entrees : IN Tableau(2**Log2NbIn-1 DOWNTO 0, NbBits - 1 DOWNTO 0);
                Sortie : OUT Std_Logic_vector(NbBits - 1 DOWNTO 0) ;
                cmd : IN Std_Logic_Vector(Log2NbIn - 1 DOWNTO 0)
        );
END Mux
```

Listing 2.9 – Description de l'entité Mux

PS : Attention, le type `Tableau` doit être écrit dans un package.

Vous étudierez cette solution, vous expliquerez pourquoi elle ne convient pas. Quel le problème VHDL sous jacent ?

Vous proposerez alors une solution permettant de résoudre le problème tout en conservant la généricité de l'entité sur les 2 paramètres `NbReg` et `NbBits`.

– Entité `Decodage` : Cette entité réalise le décodage d'une entrée (de type `std_Logic_Vector`) et place la sortie (de type `Std_Logic_Vector`) correspondante dans l'état `1` et toutes les autres sorties dans l'état `0`. Par exemple, si `cmd = "110"` alors `Sortie = "01000000"`.

```
ENTITY Decodage
        GENERIC (
                Log2NbOut : INTEGER ;
                TDecodage : TIME );
        PORT (
                Sorties : OUT Std_Logic_vector(2**Log2NbOut - 1 DOWNTO 0) ;
                cmd : IN Std_Logic_Vector(Log2NbOut - 1 DOWNTO 0)
        );
END Decodage;
```

Listing 2.10 – Description de l'entité Decodage

Vous réaliserez l'ensemble de ces entités de base en réalisant à chaque fois le test complet de l'entité.

Attention, ne pas tester correctement vos entités au moment où vous les décrivez, peut engendrer de graves disfonctionnements lors de l'assemblage du système, et dans ce cas il devient très difficile de détecter quel(s) composant(s) ne fonctionne(nt) pas.

2.5.2 Description structurelle de la file de registres

Ici, nous devons réaliser la description du schéma de la file de registres. Cette description utilise nécessairement la boucle `generate` du langage VHDL afin d'instancier `NbReg` fois le motif de base, puis l'ensemble des autres composants est instancié de façon classique.
Nous vous donnons ici le début de l'architecture structurelle de l'entité `RegisterFile`, vous la compléterez de façon à respecter le schéma qui vous est donné :

– cette description reprend la déclaration de l'ensemble des composants qui seront utilisés (partie à compléter) ;
– tous les signaux internes à l'entité doivent être déclarés (partie à compléter) ;
– le schéma de l'architecture est décrit par :
  – une boucle `generate` contenant le motif qui se répète (à compléter) ;
  – des instanciations des composants hors du motif se répétant (à compléter).

```
ARCHITECTURE Structurelle OF RegisterFile IS

        COMPONENT Registre
                GENERIC (
                        NbBits : INTEGER;
                        TReg : TIME );
                PORT (
                        Load : IN Std_Logic ;
                        Clock : IN Std_Logic ;
                        Entree : IN Std_Logic_Vector (NbBits -1 DOWNTO 0);
                        Sortie : OUT Std_Logic_Vector (NbBits - 1 DOWNTO 0)
                );
        END COMPONENT;


        COMPONENT TroisEtats
                GENERIC (
                        NbBits : INTEGER ;
                        TTri : TIME );
                PORT (
                        Entree : IN Std_Logic_vector(NbBits - 1 DOWNTO 0) ;
```

```
                Sortie : OUT Std_Logic_vector(NbBits - 1 DOWNTO 0) ;
                oe : IN Std_Logic
        );
END COMPONENT;

....
....
....
....

SIGNAL s_cmds : Std_Logic_Vector (NbReg-1 DOWNTO 0);
SIGNAL s_loads : Std_Logic_Vector (NbReg-1 DOWNTO 0);

SIGNAL s_oe1 : Std_Logic ;
SIGNAL s_oe2 : Std_Logic ;


....
....
....
....


BEGIN
        bloc : FOR i IN 0 TO NbReg-1 GENERATE
                ....
                ....
                ....
                ....
                ....
        END GENERATE bloc;

        Ou1 : OU
                GENERIC MAP (2 ns)
                PORT MAP (...........);

        Et1 : ET
                GENERIC MAP (2 ns)
                PORT MAP (...........);

        ....
        ....
        ....
        ....
        ....
        ....
        ....
        ....

        Decod : Decodage
                GENERIC MAP (Log2NbReg, 3 ns)
                PORT MAP (...........);

        Inv1 : Inverseur
                GENERIC MAP (1 ns)
                PORT MAP (...........);

        ....
        ....
```

```
        ....
        ....

END Structurelle ;
```

Listing 2.11 – Description de l'architecture structurelle de la file de registres

Chapitre 3

Contrôleur du processeur

Vous réaliserez le contrôleur de votre processeur en décrivant le schéma de la figure 3.1.

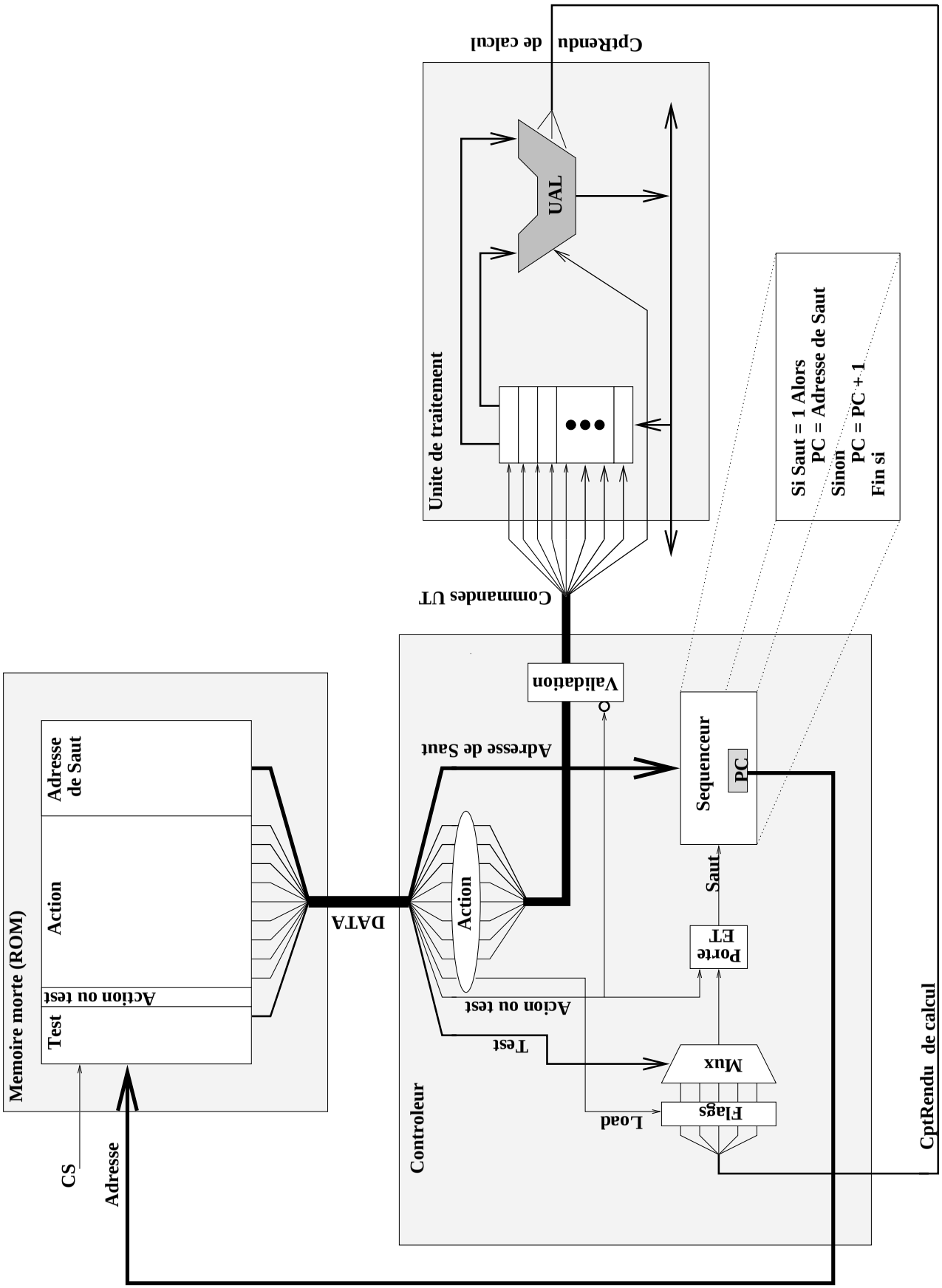

FIG. 3.1 – Schéma du processeur Von Neumann