ÉCOLE NATIONALE SUPÉRIEURE
de SCIENCES APPLIQUÉES et de TECHNOLOGIE

TECHNOPOLE ANTICIPA LANNION UNIVERSITÉ DE RENNES 1

Réalisation d'une file de registres génériques en VHDL

D.Chillet

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

Conception d'une file de registres générique en VHDL

.1 Introduction

L'objet de ce document 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 document est basé sur un certain nombre de fichiers que vous pourrez trouver le site Internet de l'ENSSAT à l'adresse suivante : `http://archi.enssat.fr/Enseignement/Vhdl/`

.2 Présentation du système à concevoir

.2.1 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 ;

.2.2 Remarques

L'ensemble de ces indications nous permet de faire une première analyse et d'aboutir aux remarques suivantes :

- le système sera constituée 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 effective 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`.

.3 Vue externe du système – Description de l'entité RegisterFile

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

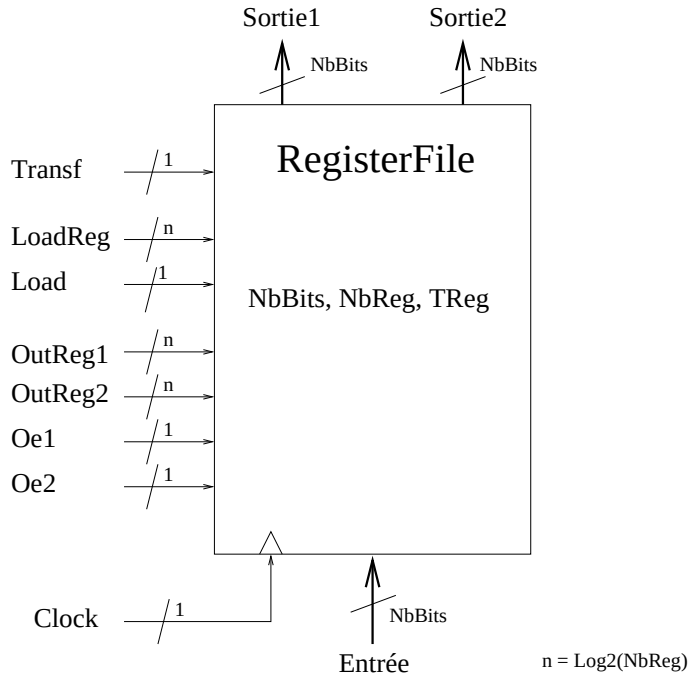

FIG. .1 – *Vue externe de la file de registre*

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;
```

LIST. 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.

.4 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 boitier 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;
```

LIST. 2 - *Description de l'architecture comportementale de la file de registres*

.5 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 procèdons 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 choisit.

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;
```

LIST. 3 - *Description de l'entité de test :* TestRegisterFile

.6 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.

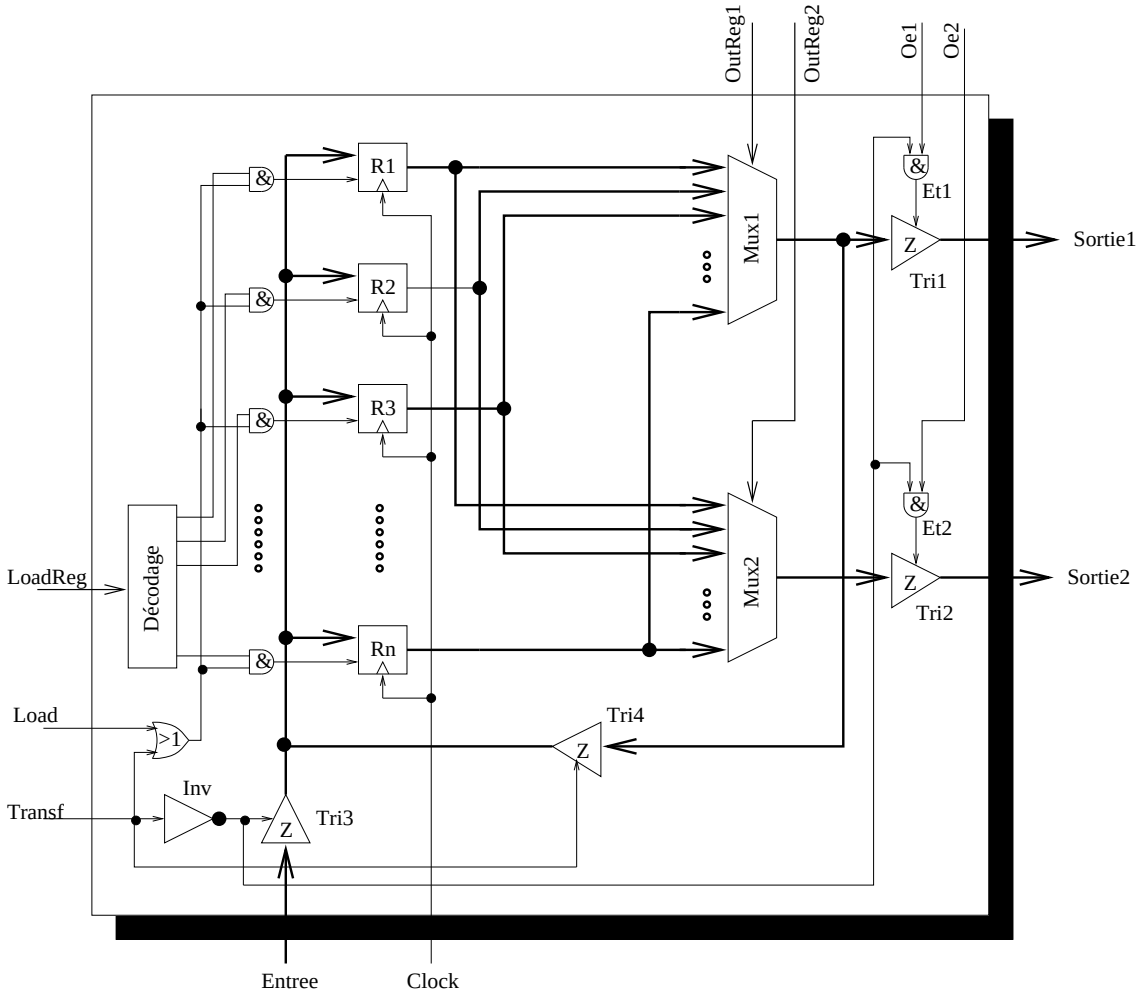

FIG. .2 – *Vue interne de la file de registre*

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.

.6.1 Descriptions des éléments de base nécéessaires

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"*.

Ce schéma fait appaître 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 le vue externe des différentes entités (afin qu'il y ait une certaine homogénéïté 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;
```

LIST. 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;
```

LIST. 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 ;
```

LIST. 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;
```

LIST. 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 contenu 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;
```

LIST. 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 déclarer les entrées du système dans un tableau à 2 dimensions. La dimension 1 correspondrait au nombre de registres `NbReg` alors que la dimension correspondrait au nombre de bits de chaque registres `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
```

LIST. 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;
```

LIST. 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'entitée.

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

.6.2 Description structurelle de la file de registres

Ici, nous devons réaliser la description du schéma de la file de registres. Cette description utilise nécéssairement la boucle `generate` du langage VHDL afin d'instancier `NbReg` fois le motif de base, puis l'ensemble des autres composants sont instanciés 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 decrit par :
  - une boucle `generate` contenant le motif qui se répète (à compléter) ;
  - des instanciations des composants hors du motif se répètant (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 ;
```

LIST. 11 - *Description de l'architecture structurelle de la file de registres*