Généricité des descriptions VHDL

Objectif : *Description de composants paramétrables*

Introduction

Dans les langages de description de matériel, la généricité peut être vue comme un moyen d'extraire les invariants d'un composant. Ces invariants sont fixés dans le code du composant tandis que les évolutions sont définies en tant que paramètres de la description et transmis au moment de l'instanciation du composant. Il peut s'agir du contenue d'une mémoire, de la taille d'un registre, de valeurs de timing, etc) définies au niveau de l'entité. Mais l'utilisation d'attributs sur les tableaux, types, signaux assurent également une meilleure généricité et par voie de conséquence, un développement plus rapide des modèles les exploitant. Dans cette manipulation, on s'intéresse à la description de 2 composants paramétrables :

- Un opérateur multi fonction, bloc qui sera décrit structurellement à partir d'une cellule élémentaire et paramétrable au niveau de la taille des opérandes.
- Une FIFO (first in first out) permettant la communication de messages entre 2 processeurs mutuellement asynchrones et paramétrables au niveau du nombre d'éléments et de la taille des éléments mémorisés.

I Les composants à modéliser

A. Opérateur arithmétique

On souhaite décrire un opérateur arithmétique, de tailel d'opérande paramétrable, capable de réaliser 5 opérations principales : transfert de l'opérande A (simple copie), addition de A et B, soustraire de B à A, incrémentation, décrémentation de la valeur de l'opérande A d'une unité. Il est également possible d' additonner A et B incrmentée de 1 ou encore d'additionner A avec le complément à 1 de B, bien que ces opérations soient marginales. Cet opérateur, de type combinatoire, peut être défini par la table d'opération de la figure ci-dessous. On notera que le transfert peut être exécutée à partir de 2 combinaisons différentes de contrôle. Une solution *mux-based* peut être envisagée en combinant un additionneur, un incrémenteur, un décrémenteur un multiplexeur 7 voies ainsi que quelques portes logiques pour la logique de contrôle. Une solution plus optimisée, dite *custom based* peut s'obtenir à partir d'une cellule élémentaire (un étage) tel que le montre également la figure.

Table fonctionnelle

S_1	S_0	C_{in}	Opération
0	0	0	**G = A** (transfert)
0	0	1	**G = A + 1** (inc)
0	1	0	**G = A + B** (add)
0	1	1	**G = A + B +1** (add+1)
1	0	0	**G = A +** $\overline{\mathbf{B}}$ (A+compl1 B)
1	0	1	**G = A +** $\overline{\mathbf{B}}$ **+1** (soust.)
1	1	0	**G = A - 1** (dec)
1	1	1	**G = A** (transfert)

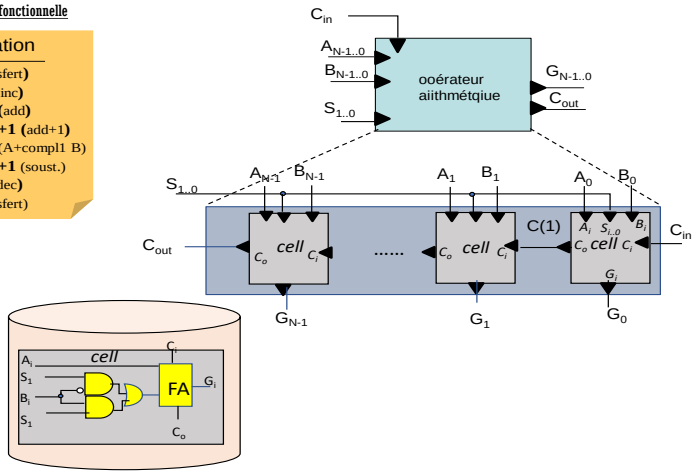

Figure 1: Table d'opération et structure de l'opérateur arithmétique

On peut voir que le full adder est l'élément central aux 4 opérations. Une logique en entrée permet de déterminer la valeur à appliquer sur l'entrée Y en fonction des entrées de contrôle S.

B. Le composant FIFO

On souhaite décrire et tester le fonctionnement d'un circuit de stockage de données de type FIFO (First in, First out). Comme le montre le symbole de la Figure 2, la FIFO dispose des signaux :

- *Datain* : il s'agit de l'entrée de donnée à écrire dans la FIFO
- *Dataout :* ce signal reçoit la valeur de la plus ancienne valeur non encore lue de la file
- *R/W :* ce signal indique le type d'opération à réaliser sur la file (à 1 pour une écriture)
- *Enable :* signal autorisant l'opération
- *Full, empty :* ce sont des indicateurs d'états indiquant si la file est pleine (respectivement vide)

Figure 2: Symbole du circuit

Comme le montre la Figure 3, la fifo peut être aisément construite à partir de 2^{deep} registres reliés pour former un tableau de registres à décalage. Cette mémoire (en gris sur la figure) est combinée avec un compteur dénombrant le nombre d'emplacements valides de la file. Chaque fois qu'une donnée est enfilée, l'ensemble des données des registres sont décalées d'une position et le compteur est incrémenté. Le compteur pointe ainsi toujours sur la dernière valeur entrée mais la sortie *dataout* est à haute impédance. Lors d'une opération de lecture, la donnée pointée par le compteur apparait sur la sortie *dataout* et le compteur est décrémenté au front d'horloge suivant. On notera que la donnée n'est pas réellement effacée de la file, elle n'est simplement plus accessible.

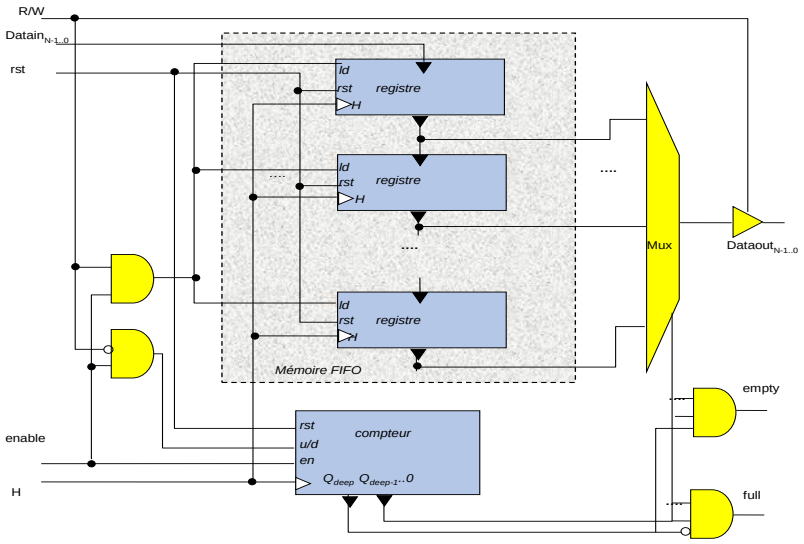

Figure 3: Bloc diagramme du composant FIFO 2^{deep} x wide

Pour déterminer si la file est vide ou non, le compteur dispose d'un bit supplémentaire. La figure montre que lorsque la valeur est négative (poids fort du compteur à 1), la file est déclarée *empty*. Lorsque le compteur atteint la valeur maximale (2^{deep}), la fifo est déclarée pleine. Par conséquent, pour

un fonctionnement correct de l'architecture, le compteur est initialisé (*rst*) avec une valeur -1, de manière que le compteur contienne 0 dès l'insertion de la première donnée.

II Préparation au TP

A. L'opérateur arithmétique

Bien que cet opérateur puisse se décrire à partir d'un couple entity/architecture unique, comme précédemment, un partitionnement de la description peut s'avérer utile pour faciliter la mise au point ou dans un objectif de reuse d'une partie de la description. La description va donc s'opérer à partir d'un premier couple entité-architecture pour décrire la cellule élémentaire. Un second couple permettra l'assemblage des cellules en nombre paramétrable pour former l'opérateur complet.

A.1 Décrivez la cellule élémentaire de l'opérateur arithmétique à partir d'un couple entity-architecture en utilisant un style d'écriture de type dataflow. On pretera attention au fait que le résultat de l'addition des trois bits est fourni sur 2 bits et qu'il est nécessaire de concatner les signaux à additionner avec un '0 en tête pour respecter la taille (ex : '0' & a) Décrivez l'opérateur complet à partir d'un couple entity-architecture en définissant la taille des opérandes comme paramètre générique du composant.

A.2 Décrivez un test bench permettant de valider fonctionnellement votre opérateur sur différentes opérations.

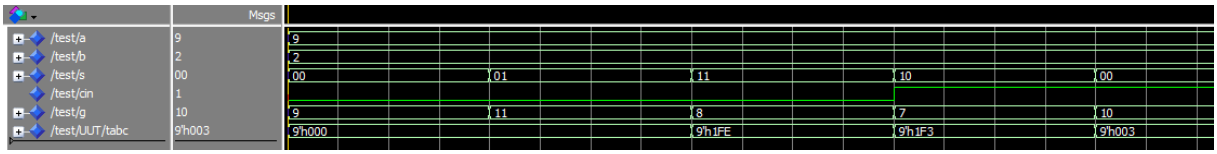

Figure 4: Exemple de chronogramme pour quelques opérations

B. Le composant FIFO

B.1 Définissez un couple entité/architecture permettant de décrire la FIFO à l'aide d'instructions concurrentes**.** Le composant sera paramétrable au niveau de :

- la taille des données élémentaires (*wide*)
- Le nombre d'éléments de la file (2^{deep}).

B.2 Définissez un testbench permettant de valider le modèle (8 places, éléments de 8 bits) en vous inspirant de la figure ci_dessous. Le testbench sera écrit en adoptant une écriture séquentielle effectuant les différentes opérations d'écriture/lecture en séquence, séparée par des *wait for*.

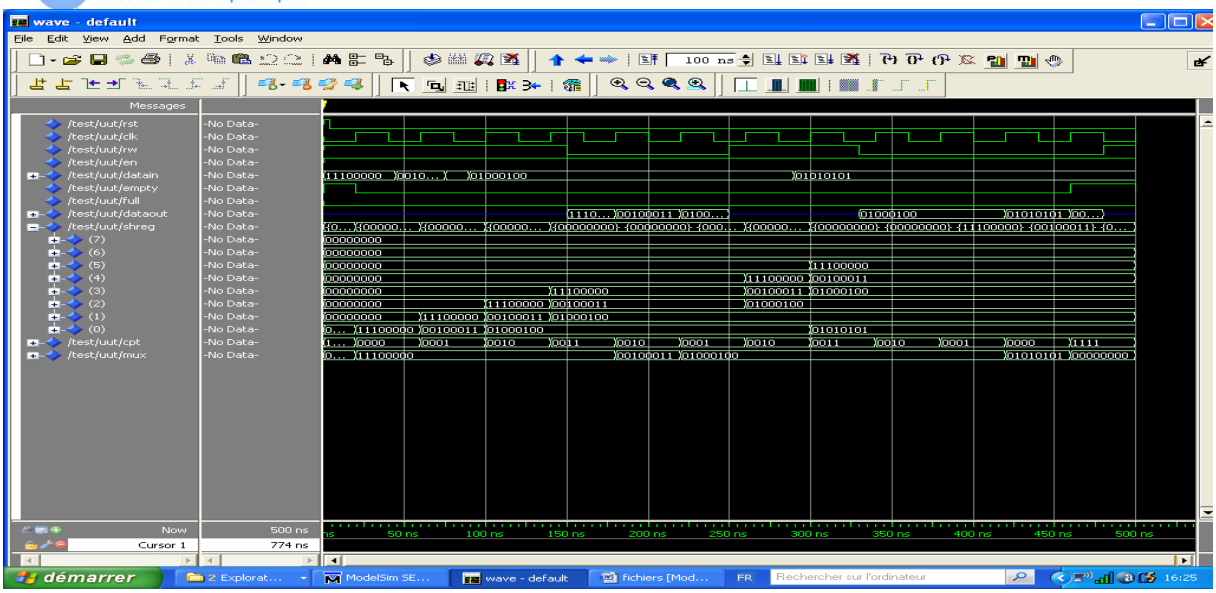

Figure 5: Trace d'une simulation pour le test de la FIFO

III Travail en séance

A. Validez le couple entité/architecture de la cellule élémentaire de l'opérateur multifonction

B. Validez le couple entité/Architecture de l'opérateur arithmétique en vérifiant son fonctionnement sur deux tailles d'opérandes différentes.

C. En vous aidant du scénario ci-dessous, validez le couple entité/architecture de la FIFO en vérifiant son fonctionnement sur différentes valeurs de paramètres génériques