

## TP n° 8 : L'interface processeur-mémoire en VHDL

Avant tout, cette séance de TP est l'occasion de poser des questions sur le DM, et en particulier de *bootstraper*<sup>1</sup> ceux qui en sont un peu avant le début.

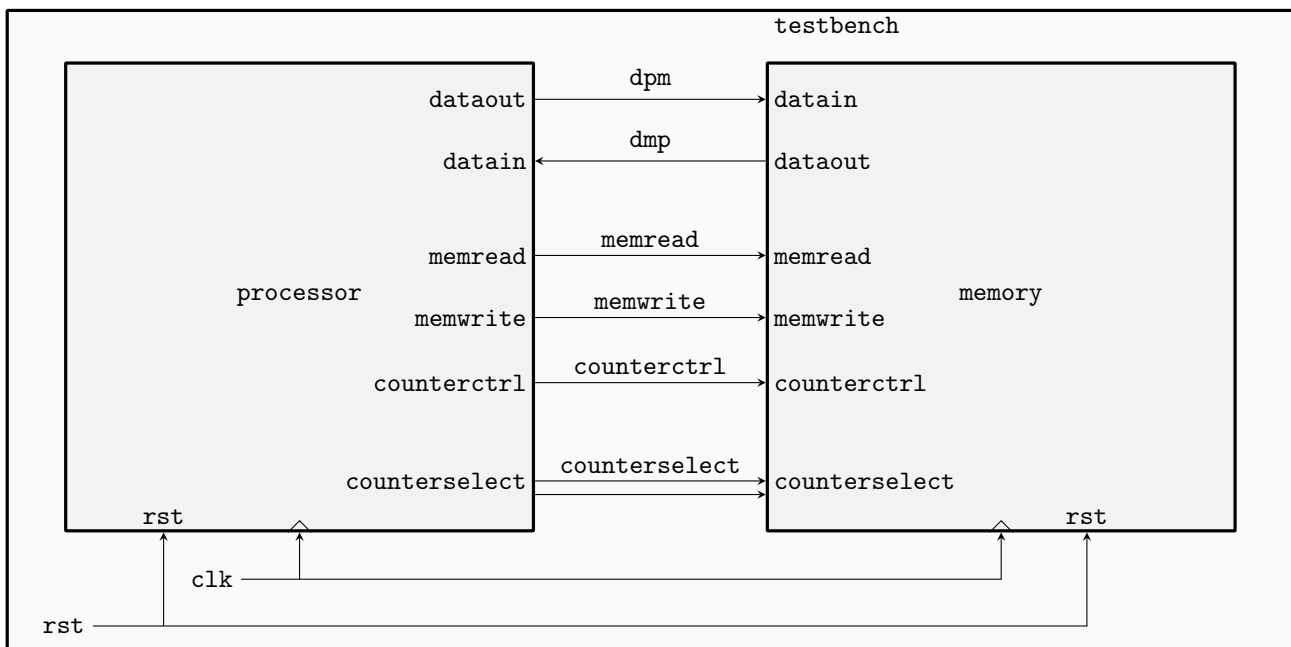
Sinon, l'objectif de ce TP est de mettre en place en VHDL l'interface processeur/mémoire du processeur du DM. On va donc parler de transmission de donnée en série, et donc d'automates.

Ce TP nécessite que vous ayez avancé le TP précédent (introduction au VHDL) jusqu'à avoir observé les chronogrammes du compteur. Si ce n'est pas le cas, terminez tranquillement le TP précédent.

### 1 Les boîtes noires

Dans la suite on va vivre avec juste 3 fichiers : `processor.vhdl`, `memory.vhdl` et `testbench.vhdl`.

FIGURE 1 – L'interface processeur-mémoire. La nouveauté de ces dessin, c'est de distinguer les noms des *ports d'entrées/sortie* (par exemple `datain`) des noms des *signaux* (par exemple `dmp`). Et parfois ce sont les même.



**Question 1** Récupérez `tp2-vhdl.tgz`. Vous y trouverez dans `memory.vhdl` une mémoire sérielle qui n'a qu'un compteur (PC). Simulez-la en utilisant les commandes fournies au début de `testbench.vhd`. Affichez dans GTKWave `clk`, `memread` et `dmp`. Vous devez observer le passage des premiers bits de la mémoire sur `dmp`. Observez aussi que `memread` n'a aucun effet.

**Question 2** Changez le contenu initial de la mémoire (en éditant `memory.vhd`) et recommencez. Un jour, vous y mettrez du binaire sortant de votre `asm.py`<sup>2</sup>...

**Question 3** Redessinez l'intérieur de la boîte noire "memory" en étiquetant les fils avec le nom qu'ils ont dans le `vhdl`. Essayez de tout comprendre dans le VHDL fourni, et posez des questions.

Remarques : vous observerez avec consternation qu'on n'utilise pas l'additionneur laborieusement construit au TP précédent. Pourquoi? D'une part par paresse (la nouvelle manière d'exprimer une addition en utilisant la bibliothèque

1. Traduction littérale : botter le cul.

2. Et c'est pour cela qu'il sort une chaîne de caractères 0 et 1 et non quelquechose de plus compact.

numeric\_std est plus compacte). D'autre part et surtout, parce que cette spécification comportementale est plus abstraite, et par exemple les compilateurs de VHDL vers circuits seront capable de la compiler en un additionneur gros mais rapide (surface  $n \log n$  et temps  $\log n$ ) ou en un additionneur petit mais lent (celui de la semaine dernière, surface et temps en  $n$ ).

Dans toute la suite on ne travaille au début qu'avec un compteur mémoire, le PC. On ignore donc counterselect. Si vous avez vraiment le temps vous ajouterez les trois autres compteurs à la fin.

## 1.1 Prise en compte de memread

**Question 4** Modifiez memory.vhdl pour que la lecture et l'incréméntation du pc n'aient lieu que lorsque le signal memread vaut 1. Lorsque memread vaut 0, dataout doit valoir 0. Indice : il y a plein de façons de faire, celle qui est la plus simple pour la suite est d'ajouter un signal pcenable au registre du PC.

Petits coups de main sur le VHDL qui vous serviront ici et dans toute la suite :

- le multiplexeur s'écrit quelquechose comme cela : `titi <= toto when select = '0' else tata;`
- Pour les constantes, c'est *single quote* pour les std\_logic, *double quotes* pour les vecteurs de bits.
- pour le enable sur le registre, il faut ajouter un if dans le process. Rappel : à l'intérieur du process, on a du bon vieux code séquentiel.

Testez que cela marche.

## 1.2 Prise en compte de memwrite

**Question 5** Modifiez testbench.vhdl pour tester une écriture en mémoire. Pour le test, comme on n'a pas encore implémenté ce qu'il faut pour modifier le PC de l'extérieur, vous vous contenterez d'utiliser le signal rst pour remettre à 0 le PC après votre écriture.

(rst ne remet pas à zéro la mémoire, vous pouvez vérifier dans le VHDL).

## 1.3 Prise en compte de counterctrl

Lorsque counterctrl vaut 1 et memwrite vaut 1, les bits présentés sur datain ne sont pas enregistrées dans la mémoire mais dans PC.

Plus précisément, dans ce cas, un registre à décalage de addresswidth bits (appelez-le pcshiftrég), est décalé d'un bit à gauche et son bit de poids faible reçoit la valeur de datatin. Lorsque counterctrl passe de 1 à 0, pcshiftrég est recopié dans le pc.

Remarque : cela ne marchera que si le processeur laisse counterctrl à 1 pendant exactement addresswidth cycles. Dans notre ISA, les instructions setctr et getctr transfèrent tous les addresswidth bits d'un compteur depuis/vers le registre<sup>3</sup>.

Lorsque counterctrl vaut 1 et memread vaut 1, les bits envoyés sur dataout sont les bits du PC. À nouveau, vous pouvez faire l'hypothèse que counterctrl restera à 1 pendant exactement addresswidth cycles.

**Question 6** Dessinez (sur du papier) les deux dessins suivant en parallèle :

- L'architecture du contenu de la boîte noire memory, en ajoutant des multiplexeurs là où il faut.
- un automate qui prend en entrée counterctrl, memread et memwrite, et dont les sorties sont les signaux suivants du dessin précédent :
  - les signaux de commande des multiplexeurs
  - les signaux enable des registres (registre à décalage et compteur)

C'est une bonne idée de faire valider par un enseignant.

**Question 7** Ensuite (ou en attendant) attaquez le VHDL. A ce sujet, j'encourage de rester dans une seule entité. Le registre à décalage y sera bien isolé dans un nouveau process. Mais si vous préférez créer une entité pour lui, n'hésitez pas. C'est une affaire de goût.

Vous trouverez un exemple de syntaxe VHDL pour un automate sur la page web <http://perso.citi-lab.fr/fdedinec/enseignement/2017/>.

Autre petit coup de main sur le VHDL :

- extraction d'un sous-vecteurs de bits : `toto(6 downto 2)` est un vecteur de 5 bits.
- concaténation de vecteurs de bits : `titi <= toto & tata;` Il faut que la taille de titi soit la somme des tailles de toto et tata.

Si vous êtes coincés sur memory.vhdl, fignez testbench.vhdl en attendant de l'aide.

---

3. Ce qui veut dire que les sauts, même relatifs et proches, vont coûter beaucoup de transitions sur l'interface. On réfléchira un jour à les faire gérer par la mémoire elle-même...

#### **1.4 Ya plus qu'à tout refaire 4 fois**

On veut à présent avoir le choix entre l'un des 4 compteurs contrôlés par `countersselect`. Réfléchissez (sur dessin) où il faut mettre des multiplexeurs, puis corrigez votre VHDL

## **2 Et côté processeur**

Écrivez dans `processor.vhdl` l'entité du processeur.

Le but à présent est de construire, côté processeur, le squelette du cycle de von Neumann : l'automate qui lit les instructions bits à bits et les décode au fur et à mesure pour savoir quels sont les bits restant à lire.

En principe vous savez tout faire, mais en principe vous n'êtes pas arrivé jusqu'ici.