TD
Langages de description de matériels

SEI- 2015

Objectifs de ces TP :

- TP1 :
  - comprendre et compiler une description VHDL et son testbench
  - apprendre à modifier un composant VHDL purement combinatoire et son testbench
  - apprendre à écrire un composant séquentiel en VHDL (compteur)
  - Simuler et observer des traces de simulation, prise en main des outils
- TP2
  - Ecrire la description VHDL d'un registre à décalage et la tester
  - Commencer la synthèse d'un circuit combinatoire, prise en main des outils
- TP3
  - Ecrire et tester un automate (FSM)
  - Faire une synthèse de FSM afin d'aboutir à une implémentation physique

Documents à utiliser lors du TP

- Poly du cours
- Liens internet de vhdl de votre choix pour la vérification de syntaxe. Fortement recommandé en séance !
- Document joint sur la simulation et la synthèse de programmes VHDL. Lire intégralement et attentivement le document introductif joint. La présentation de l'outil de simulation et de synthèse s'y trouve !…

Notes

Ce TP se fait en binôme et un compte rendu global devrait être rendu à la fin de la dernière séance.

Les machines sont en libre service, vous pouvez y revenir à volonté, mais la porte bloque de l'extérieur à 20h00. Vous pouvez ressortir, mais pas retourner.

Questions et exercices pour le TP 1

1) Analyser et comprendre les fichiers présents dans le répertoire TP_VHD concernant la description d'une unité arithmétique – logique (ALU). Le composant se trouve en répertoire /vhd alors que son testbench est nommé test_ALU_assert et se trouve dans le répertoire /bench). Tous les programmes VHDL qui vous sont donnés devront être commentés. (voir répertoires /vhd et /bench).

2) Créer une bibliothèque ayant pour nom **lib_COMP** dans le répertoire **vhd** et compiler la description VHDL de l'ALU dans la bibliothèque **lib_COMP**.

3) Créer une bibliothèque **lib_BENCH** dans le répertoire bench et compiler le "test_ALU_assert" dans la bibliothèque **lib_BENCH**. L'ordre dans lequel les deux fichiers sont compilés a une importance ou pas ?

Vous devez respecter ces noms de bibliothèque, ils sont déclarés par ailleurs dans des fichiers de configuration des outils.

4) Simuler l'unité arithmétique - logique afin de vérifier le bon fonctionnement (à l'aide de l'outil **Modelsim – voir l'autre poly**). Simuler avec run –all. Que se passe t'il à la fin de la simulation ? Pourquoi ? Quels sont les délais de propagation ?

5) Modifier le test bench pour simuler et vérifier l'opération arithmétique 100-200.

6) A l'aide de votre imagination et du poly de cours, faire en sorte que les « asserts » fonctionnent (au moins un) !

7) Modifier la description ALU en proposant une architecture basée sur l'utilisation d'un process; et compléter avec l'opération arithmétique A+1, pour CMD ="111" et A pour CMD ="000";.

8) Rajouter un signal d'entrée de type **reset**, et une **clock** afin de transformer cette architecture dans une architecture possédant un reset asynchrone, alors que et les opérandes d'entrée et le résultat seront clockées. Changer le testbench afin de prendre en compte les nouveaux signaux et vérifier que tout fonctionne comme prévu.

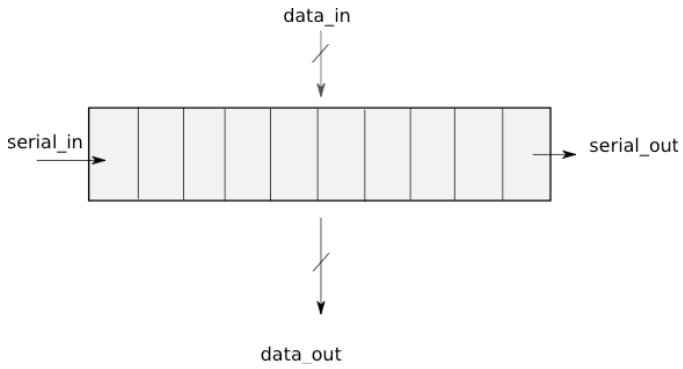
9) Dans le répertoire **/vhd** créer un compteur UP/DOWN sur 8 bits. Ce compteur à une entrée **resetn** asynchrone – active à '0', une clock (notée **clk**), une entrée **enable** synchrone (notée **en**), une entrée **UP/DOWN** et une sortie (**count**) sur 4 bits.

10) Compiler cette description et d'après le textbench de l'ALU créer un testbench pour ce compteur. (ne pas utiliser des asserts). Valider le fonctionnement **up/down** par la simulation.

Questions et exercices pour le TP 2

11) Dans le répertoire /vhd écrire et compiler un registre à décalage sur 11 bits.
Définition des IN/OUT: Ce registre spécial à une entrée reset – asynchrone, une entrée clock (notée **clk**), une entrée shift_right (notée **sr**) synchrone, une entrée **LOAD** synchrone**,** une entrée **serial_in,** une entrée **data_in** sur 11 bits, une sortie **data_out** sur 11 bits et une sortie **serial_o** sur un bit.
Fonctionnement:

- Si reset '1' alors la sortie **data_out** est mise à zero.
- Si **LOAD** est à '1', l'entrée **data_in** est chargée parallèlement dans le registre.
- Si **sr** vaut '1', le registre effectue un décalage à droite d'une position (division par 2) et l'entrée **serial_in** entre dans le registre. La sortie **serial_o** prendra pour valeur le bit de poids faible.



12) D'après le testbench de l'ALU/compteur créer un testbench pour ce registre à décalage.
13) Valider le fonctionnement par la simulation.
14) Synthétiser **l'UAL du TP1** (à l'aide de l'outil **PRECISION**, **voir l'autre poly**) avec un codage one-hot. Relever la fréquence et la surface estimées par l'outil.
15) Générer une « netlist » VHDL du résultat de synthèse. Enregistrer ce résultat de la synthèse sous le nom ALU_synth.vhd dans le répertoire **synth**. Ouvrir avec un editeur de texte et regarder le résultat. Commenter la différence entre la description initiale et celle-ci.
16) Créer une bibliothèque **lib_SYNTH** dans le répertoire /**synth** et y compiler ALU_synth.
17) Dans le fichier test_ALU_assert adapter les noms des bibliothèques pour simuler l'ALU synthétisée.
18) Simuler à nouveau **l'UAL** après la synthèse. Vérifier le fonctionnement.

Questions et exercices pour le TP 3

Arbitre à priorité fixe, ARBITER

On veut réaliser un circuit FSM de type arbitre, fonctionnant en style tourniquet (round robin). Cet arbitre contrôle l'accès de 3 composants de type processeurs (P1, P2, P3) à une ressource partagée de type mémoire (M).

Spécifications:

- ✓ Un seul processeur peut accéder la mémoire à un instant donné, selon les priorité suivantes: d'abord P1, puis P2, sinon P3. Si plusieurs processeurs tentent d'accéder à la ressource, la priorité est donnée en ordre au P1 (la plus haute priorité), P2, puis P3 (la plus faible priorité).
- ✓ Chaque processeur envoie à l'arbitre un signal de type 'request' à '1' (R1, ou R2 ou R3), lorsqu'il souhaite accéder à la mémoire M.
- ✓ L'arbitre produit les signaux ack1, ack2, ou ack 3 (égaux à '1') aux processeurs, leur signalant lequel de 3 processeurs aura accès à la mémoire.
- ✓ Lorsque le processeur finit l'accès à la mémoire, il remet son signal 'request' à '0'. Autrement, ce signal restera à '1' durant l'utilisation de la ressource partagée.
- ✓ Le système possède un signal de type reset, actif à '0' qui initialise le système en mode 'Idle'. Autrement tous les signaux du système changent au front montant de l'horloge.
- ✓ La représentation sous forme de graphe de transition d'états est présenté par la suite.

1). Ecrire une description VHDL synthétisable de cet arbitre (/vhd/arbiter.vhd); la compiler.

2). Ecrire un testbench (/bench/test_arbiter.vhd) qui permettra un test exhaustif de cet arbitre (veillez à ce que chaque état soit parcouru au moins une fois, et que toute transition d'un état à l'autre se fasse au moins une fois). Justifiez par commentaires dans le code du testbench. Vérifier les sorties par des asserts. Simuler.
3). Faire des synthèses successives avec des codages binaires et one-hot, relever la surface et interpréter les réponses.