Les scripts de simulation

Objectif: *Description et validation d'une architecture permettant la réalisation d'un multiplicateur binaire hardware sous forme séquentielle.*

Introduction

Modelsim, comme beaucoup d'outil de simulation de descriptions HDL, fournit un langage de commandes macro pour entrer des commandes de simulation sous forme manuelle dans la fenêtre de la console. Ces commandes permettent de forcer des valeurs de signaux, d'affecter des formule ou encore d'exécuter des étapes de simulation. Ces commandes peuvent être exécutées à partir d'un fichier (on parle de script), ce qui permet une economie de temps vis-à-vis d'une entrée manuelle ou depuis le GUI. Cela permet une automatisation complète du processus de vérification. On peut ainsi lancer plusieurs simulations à partir de différents testbench, l'un après l'autre. Les scripts de macro peuvent exécuter des programmes externes tel que des programmes de synthèse. Dans cette manipulation, nous verrons l'écriture d'un script pour la simulation d'un circuit de multiplication binaire effectuée séquentiellement.

I Le circuit de multiplication binaire

La multiplication binaire s'effectue manuellement sous la forme de décalages successifs à gauche du multiplicande et d'addition comme ci-dessous:

	23	10111	multiplicande
x	19	10011	multiplieur
		10111	
		10111	
		00000	
		00000	
		10111	
	437	110110101	

L'implémentation combinatoire de la multiplication est très couteuse au niveau matériel. Pour effectuer cette opération de manière séquentielle (bit par bit) La multiplication binaire s'effectue plus facilement avec quelques changements. Tout d'abord, au lieu d'avoir un circuit numérique additionnant *n* nombres binaires simultanément, on utilise un circuit additionnant simplement 2 nombres.

23	10111	multiplicande
x 19	10011	multiplieur
	00000	produit partiel initial
	10111	ajout du multiplicande, bit du multiplieur étant à 1 (1)
	10111	produit partiel après addition et avant décalage (2)
	01011*1*	produit partiel après décalage (3)
	10111	(1)
	100010*1*	(2)

	1000101	(3)	
	01000101		(3)
	001000101		(3)
	10111		(1)
	110110101		(2)
437	0110110101		produit final

Figure 1: Principe de la multiplication binaire

Par conséquent, à chaque copie du multiplicande ou entrée de 0 dans l'addition, ceux-ci sont immédiatement additionnés à un produit partiel. Celui-ci est mémorisé dans un registre en attente d'action de décalage.

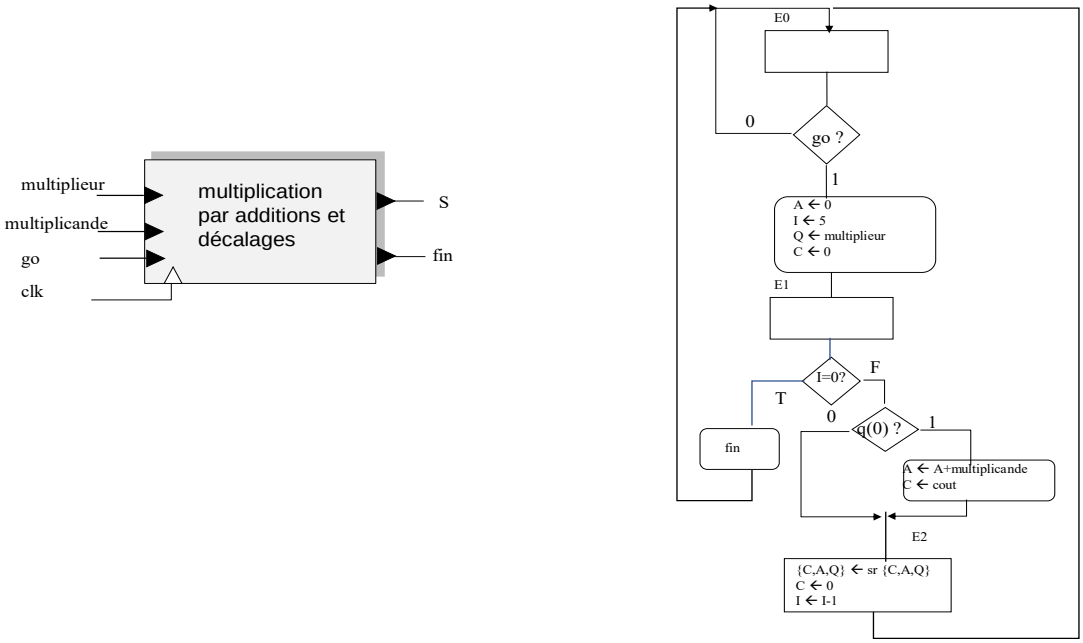

Figure 2: Symbole et ASM du circuit de multiplication

Ensuite, comme le montre la figure 2, au lieu de décaler à gauche les copies du multiplicande, le produit partiel est décalé (*sr*) à droite (en prenant en compte la retenue de l'additioneneur *cout)*. Le produit partiel et la copie du multiplicande conservent ainsi leur position relative. Mais, mieux encore, cela signifie qu'un additionneur suffit pour les *n* bits au lieu de *2n* positions

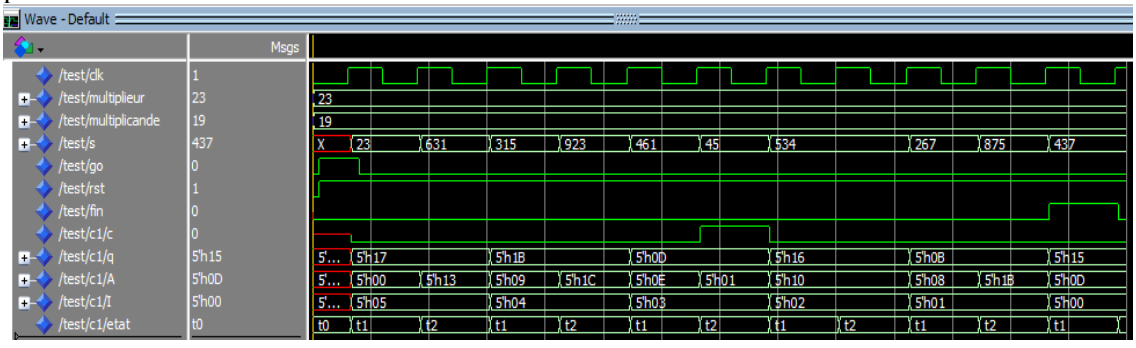

Figure 3: chronogramme de l'ASM pour 23*19

L'addition se déroule toujours à l'intérieur des *n* mêmes positions, au lieu de se décaler à gauche d'un bit à chaque fois. Finalement, lorsque le bit correspondant du multiplieur est un 0, nul besoin d'additionner des zéros au produit partiel, cela ne changerait rien. La figure ci-dessus montre l'évolution de la sortie pour une multiplication de 23 par 19.

Le modèle interne architectural est un modèle hiérarchique. A ce titre, il permet une représentation d'une solution d'implémentation à différents niveaux d'abstraction afin de réduire la complexité des constituants ou réutiliser des constituants ici de projets antérieurs. C'est le principe de la conception modulaire La solution architecturale fait apparaître une conception hiérarchique à partir de constituants de différentes complexité. La hiérarchisation de la conception permet de faire abstraction des détails d'implémentation de certains modules. Ces modules peuvent également être issus de développement préalables et se voir réutilisés dans plusieurs conceptions.

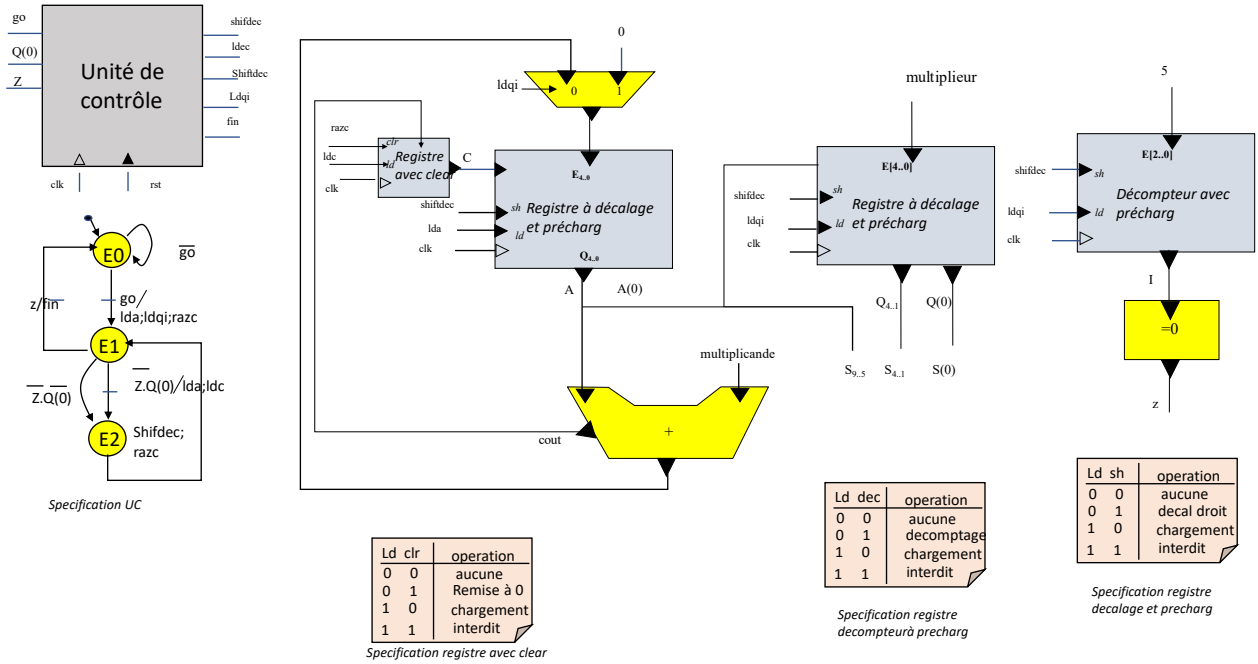

Figure 4: Solution RTL du circuit

La solution architecturale de la multiplication est basée sur 2 instances du module de décalage (registre *A* et *Q*), une instance du module décompteur (*I*), un registre 1 bit avec remise à zéro (*C*), un opérateur d'addition et un multiplexeur.

Le module registre à décalage droit et préchargement permet la réalisation des deux opérations mentionnée dans la spécification: chargement ou décalage vers la droite (selon la valeur de *sel*), sous réserve que l'entrée d'autorisation *en* soit à un.

Un module registre à chargement et raz permet de sélectionner une opération de chargement d'entrée *D* ou une opération de remise à zéro selon l'entrée de sélection, sous réserve de *en*=1.

Le module compteur à préchargement permet également la réalisation de deux opérations: chargement ou décrémentation (selon la valeur de sel).

Le dernier bloc concerne l'unité de contrôle dont le rôle est de positionner les signaux de commandes des registres en fonction des entrées du circuit et des indicateurs d'état.

Ce bloc est décrit par un automate à état fini de type machine de mealy. Celui-ci débute lorsque l'entrée *go* vaut 1et émet un signal de *fin* lorsque l'opération est terminée. Le circuit consommateur de la donnée (un microprocesseur par exemple) peut alors exploiter le résultat formé de la juxtaposition des données A et Q.

Preparation

A. Dans un fichier *run_file.tcl*, tapez et complétez les commandes du script ci-dessous,

```
#On quitte la simulation en cours éventuelle
quit -sim
#directives de compilation des fichiers sources
vcom multiplication.vhd
vcom testmult.vhd
#lancement du simulateur (résolution de 1ns)
vsim -t 1ns work.test(bench)
#ajout des signaux dans la fenêtre de simulation
add wave -noupdate -divider {entrees du bloc multiplication}
add wave -noupdate -radix unsigned -radixshowbase 0 test/multiplieur

 # PARTIE A COMPLETER POUR LE RESTE DES SIGNAUX

#simulation de 1us
run 1 us
```

B. Dans un fichier *multiplication.vhd*, définissez une entité *mult* dont les ports seront de type *std_logic_vector* de 8 bits pour les données en entrée et 16 bits pour le résultat. Le reste des signaux sera de type *std_logic*. On inclura les packages *std_logic_1164* et *std_logic_unsigned* pour cela.

C. Toujours dans le même fichier, donnez une description des différents blocs du chemin de données dans une architecture unique *rtl*_à l'aide d'instructions d'affectation concurrente (éventuellement conditionnelles ou sélectives).

D. Complétez l'architecture avec une description du séquenceur à l'aide de 2 processus:

1. un processus séquentiel de mémorisation de l'état suivant
2. un processus combinatoire pour la génération des sorties et de l'état suivant

E. Dans un fichier *testmult.vhd*, définissez une architecture *test* pour valider cette architecture.

II Travail en séance

Dans la fenêtre modelsim, cliquez sur *tools→tcl→execute macro* et entrez le nom du fichier de commandes. Vérifiez la conformité du résultat de l'exécution de ces commandes .