TP1 : Modélisation Comportementale VHDL Data-Flow

Objectifs

Le but de cette séance est de modéliser en VHDL, puis de simuler un petit circuit très simple : un additionneur accumulateur que nous appellerons *addaccu*.

Vous pouvez obtenir la spécification fonctionnelle de ce circuit en cliquant ici.

L'outil de simulation utilisé aujourd'hui est le compilateur/simulateur VHDL **asimut**, qui fait partie de la chaîne de CAO *ALLIANCE* développée au laboratoire LIP6.

Vous pouvez obtenir des informations détaillées sur n'importe quel outil de la chaîne *ALLIANCE* en tapant la commande :

```
> man asimut
```

Pour utiliser cette commande, il faut définir le chemin d'accès aux fichiers *man* dans la variable d'environnement UNIX MANPATH.

Le déroulement de ce premier TP comporte trois étapes: Modélisation comportementale du circuit *adaccu*, description des stimuli en entrée et des résultats attendus en sortie, et simulation effective avec le simulateur **asimut**.

Commencez par créer un répertoire de travail *tp1* pour archiver les fichiers de ce TP.

A) Modélisation zero-delay

Le simulateur **asimut** n'accepte qu'un sous-ensemble du langage VHDL, en particulier pour la description comportementale, qui ne doit contenir que des "assignations concurrentes", ce qui correspond au style *data-flow*.

Ces assignations peuvent contenir des informations temporelles, grace à la construction *after*:

```
c <= a and b  after 1 ns;
```

Puisque la sémantique du langage VHDL repose sur les *événements*, l'assignation ci-dessus signifie qu'un événement sur le signal a (c'est à dire un changement de valeur à une date t) peut entraîner un événement sur le signal c à la date t + 1 ns.

La construction *after* peut être utilisée dans des constructions plus complexes, comme la construction *with* x *select* qui décrit un multiplexeur :

```
with sel select
```

```
x <= a after 2 ns when '0',
          b after 2 ns when '1';
```

Mais elle ne peut pas être utilisée dans la construction qui décrit l'écriture dans un registre:

```
accu : block(ck='1' and not ck 'stable)
        begin
        accu <= guarded s ;
        end block ;
```

La construction *after* peut être omise, ce qui correspond à un temps de propagation nul. Une description comportementale qui ne contient pas de constructions *after* est une description dite "zero-delay": on considère que tous les temps de propation sont négligeables par rapport au temps de cycle. C'est donc une description où on cherche à valider la fonctionnalité logique, mais où on ne cherche pas à simuler le comportement temporel.

Ecrire une description comportementale "zero-delay" pour le composant *addaccu*.

Le fichier comportera l'extension ".vbe" qui est l'extension usuelle pour indiquer un fichier VHDL comportemental de type data-flow (.vbe pour Vhdl BEhaviour).

Vous pouvez obtenir des indications sur la syntaxe du format .vbe en tapant la commande

```
>man vbe
```

Vous pourrez vérifier que votre fichier *addaccu.vbe* est syntaxiquement correct en utilisant l'outil **asimut** pour compiler ce fichier, sans lancer la simulation.

```
>asimut -b -c addaccu
```

- l'option -b indique qu'il s'agit d'une description purement comportementale (extension .vbe), car **asimut** accepte également en entrée des descriptions structurelle (dans ce ca le fichier possède l'extension .vst).
- l'option -c permet de compiler sans simuler.

Comme tous les outils de la chaîne ALLIANCE*, **asimut** utilise des variables d'environnement UNIX, qui permettent de paramètrer le simulateur. La signification des 8 variables d'environnement utilisées par **asimut** est à découvrir dans la page man de l'outil **asimut**. Vous pouvez consulter la valeur courante de ces variables en tapant la commande*

```
>echo $VH_MAXERR
```

Vous pouvez modifier ces variables d'environnement en utilisant la commande suivante:

```
>export VH_MAXERR=10
```

Votre description est syntaxiquement correcte lorsque vous n'obtenez plus aucun message d'erreur.

B) Définition des stimuli

Lorsque la description comportementale du composant *addaccu* compile sans erreurs, il faut écrire le fichier décrivant les stimuli qui vont être appliqués sur les entrées du circuit. Ce fichier permet également de déclarer quels signaux on souhaite observer : les signaux observés peuvent être soit des sorties du circuits, soit des signaux internes.

Le format du fichier de stimuli accepté par *asimut* est défini dans la page man du format .pat :

```
>man pat
```

Ce fichier *stimuli.pat* doit en particulier décrire le signal d'horloge. On choisira un signal périodique de période 20 ns, et de rapport cyclique 50% (cela signifie que l'état bas et l'état haut ont des durées égales à 10ns).

Vous trouverez ci-dessous un exemple de fichier *stimuli.pat*

```
-- controled inputs :
in       a (3 downto 0)                X;;;
in       b (3 downto 0)                X;;;
in       sel                           B;;;
in       ck                            B;;;
in       vdd                           B;;;
in       vss                           B;;;

-- observed outputs & internal signals :
register addaccu.accu (3 downto 0)     X spy;;;
out      s (3 downto 0)                X spy;;;

begin
-- Pattern description :
--               a  b  s  c  v  v  a  s
--                     e  k  d  s  c
--                     l     d  s  c
--                                 u
<0   ns>       : 4  3  0  0  1  0 ?* ?* ;
<+10 ns>       : 4  3  0  1  1  0 ?* ?* ;
<+10 ns>       : 0  1  1  0  1  0 ?* ?* ;
<+10 ns>       : 0  1  1  1  1  0 ?* ?* ;
end;
```

On utilise l'attribut *spy* pour les signaux qu'on souhaite observer. Les signaux internes sont désignés par la notation *composant.signal*. Les attributs B et X précisent si on souhaite un affichage de type binaire ou hexadecimal.

Attention : Comme les registres internes au composant *addaccu* sont des bascules à échantillonnage sur front montant, les signaux d'entrée ne doivent pas changer de valeur lorsque le signal d'horloge passe de 0 à 1.

Vous pouvez vérifier le chronogramme défini par votre fichier *stimuli.pat*, en utilisant l'outil de visualisation graphique **xpat**:

```
>xpat -l stimuli
```

C) Simulation logique (zero-delay)

Vous pouvez maintenant lancer la simulation sous **asimut**, en applicant les stimuli définis dans le fichier *stimuli.pat* sur la description comportementale définie dans le fichier *addaccu.vbe*:

```
>asimut -b addaccu stimuli result
```

Le dernier argument sur la ligne de commande est le nom du fichier résultat au format .pat. Ce fichier contient, en plus des valeurs des signauxd' entrée - les valeurs des signaux calculées par le simulateur. Vous pouvez visualiser ces résultats en utilisant l'outil **xpat**.

D) Simulation logico-temporelle

Pour terminer ce TP, vous allez maintenant complêter la description comportementale du composant *addaccu*, en utilisant la construction *after* pour introduire des informations temporelles dans la description comportementale, de

façon à modéliser les temps de propagation dans les différents opérateurs combinatoires. Ces temps de propagation dépendent de la complexité des expressions Booléennes qu'on trouve dans le membre de droite des assignations concurrentes de la description comportementale. Vous utiliserez les valeurs suivantes:

x xor y xor z	2ns
multiplexeur	1ns
x.y + x.z + y.z	1ns

Après modification vous sauvegarderez cette description comportementale temporisée sous le nom *addaccu_time.vbe*.

Puisque qu'on décrit explicitement les temps de propagations, on peut maintenant à observer la propagation du report dans l'additionneur. Il faut modifier le fichier *stimuli.pat* pour introduire les directives permettant d'observer les signaux internes carry[3:0] et mux[3:0]. On affichera ces signaux bit par bit en utilisant les directives suivantes:

```
register addaccu.accu (3 downto 0)      X spy;;;
out      s (3 downto 0)                 X spy;;;
signal   addaccu.carry (0)              B spy;;;
signal   addaccu.carry (1)              B spy;;;
signal   addaccu.carry (2)              B spy;;;
signal   addaccu.carry (3)              B spy;;;
signal   addaccu.mux (0)                B spy;;;
signal   addaccu.mux (1)                B spy;;;
signal   addaccu.mux (2)                B spy;;;
signal   addaccu.mux (3)                B spy;;;
```

Il faut également modifier la description des "patterns" puisqu'on veut maintenant observer 10 signaux:

```
-- Pattern description :
--               a  b  s  c  v  v  a  s  c  c  c  c
--                     e  k  d  s  c     0  1  2  3
--                     l     d  s  c
--                              u

<0  ns>        : 4  3  0  0  1  0 ?* ?* ?* ?* ?* ?* ?* ?* ?* ?* ;
<+10 ns>       : 4  3  0  1  1  0 ?* ?* ?* ?* ?* ?* ?* ?* ?* ?* ;
```

Il faut enfin appliquer sur les entrées des valeurs qui sensibilisent

la chaîne de propagation des reports, telles que

- a[3:0] = 1111
- b[3:0] = 0001
- sel = 0

On sauvegardera ce fichier de stimuli sous le nom *stimuli_time.pat*.

```
>asimut -b addaccu_time stimul_time result_time
```

En analysant le chronogramme obtenu, donnez une estimation de la fréquence maximale de fonctionnement de ce circuit.

Compte-Rendu

Vous rédigerez un compte-rendu d'une page maximum pour ce TP, et vous joindrez en annexe le fichier *addaccu_time.vbe*, ainsi que le chronogramme résultant de la simulation logico-temporelle pour le fichier *stimuli_time.pat*.