Projet Intégration des Systèmes

Etude et conception d'un filtre numérique RIF à réponse impulsionnelle programmable

Conception et validation fonctionnelle

1. Procédure de connexion

Login : `xph2sle000 à 025`
Passwd : `sYst1nTeg`

`Changez le mot de passe à la première connexion avec la command yppasswd`

2. Configuration de l'environnement VHDL

La configuration de l'environnement VHDL est réalisée par l'intermédiaire d'un fichier qui contient un ensemble de commandes. Ce fichier est dénommé par défaut `modelsim.ini` (un exemplaire est disponible dans le répertoire `~/FILTRE/config`). Pour que ce fichier soit utilisé par défaut lors de l'exécution d'une commande (compilation, simulation,…), il est nécessaire de définir une variable d'environnement dénommée `MODELSIM`:

```
export MODELSIM=$HOME/FILTRE/config/modelsim.ini
```

Le fichier `~/FILTRE/config/.bashrc` devrait contenir cette commande. Vérifier, sinon la rajouter ! Configurer l'environnement avec cette commande (à effectuer dans chaque nouveau terminal):

```
source ~/FILTRE/config/config_RTL
```

Dans le fichier `modelsim.ini`, doivent apparaître les liens entre bibliothèques logiques et physiques. Par exemple, si vous créez une bibliothèque de nom logique `lib_funct`, il faut définir le répertoire physique dans lequel elle est stockée :

```
lib_funct = $HOME/FILTRE/lib/lib_funct
```

Les principales bibliothèques nécessaires aux projets sont déjà définies dans le fichier `modelsim.ini`. Vous n'avez donc pas forcément à le modifier…

3. Structure du projet

- Répertoires contenant les codes sources des modèles du circuit à concevoir (uC et filtre):

```
~/FILTRE/vhd/
~/FILTRE/vhd/filter
~/FILTRE/vhd/uC
```

- Répertoire contenant les codes sources des environnements de validations :

```
~/FILTRE/bench
```

4. Création d'une bibliothèque

Cette opération est nécessaire avant la première compilation d'une unité de conception dans la bibliothèque `lib_NOM`.

```
vlib ~/FILTRE/lib/lib_NOM
```

5. Compilation

La commande suivante effectue la compilation des unités de conception contenues dans le fichier source `source.vhd`, et stocke les résultats compilés sans erreur dans la bibliothèque de travail `lib_NOM`:

```
vcom –work lib_NOM source.vhd
```

6. Simulation avec ModelSim

La commande suivante lance le simulateur:

```
vsim &
```

Tous les fichiers présents dans le répertoire de travail courant seront accessibles en cours de session, par exemple les fichiers de commande du simulateur (cf. §Créer et utiliser un fichier de commandes).

6.1. Charger une unité de conception à simuler

Dans la fenêtre `Workspace`, qui représente la structure du projet, sélectionner la bibliothèque `lib_bench` et charger la configuration du composant à simuler.
Le chargement d'une nouvelle unité à simuler pourra être fait par le menu `Simulate -> Start simulation`. Dans ce menu se trouve un onglet pour spécifier l'unité de temps à utiliser pendant la simulation ("ns" est l'unité par défaut).

6.2. Observer les signaux de l'unité de conception

La fenêtre `Objects` est ouverte lors du chargement d'une unité de conception. Cette fenêtre montre les signaux d'entrée/sortie du bloc sélectionné dans la fenêtre `Workspace` (signaux définis dans l'entité).
La fenêtre `Objects` peut être obtenue aussi à partir du menu `View -> Debug Windows`.

6.3. Observer le code VHDL source

La visualisation des codes source VHDL est possible à partir de l'onglet `Files` de la fenêtre `Workspace`.

6.4. Observer les chronogrammes (ondes) de simulation (waves)

La définition des signaux à afficher sous forme de chronogrammes peut être réalisée dans la fenêtre `Objects` (menu pop-up obtenu par le bouton droit de la souris, commande `Add to Wave`). La liste de signaux à ajouter peut correspondre à :

`Selected signals` : signaux sélectionnés dans la fenêtre `Objects` avant d'activer la commande.
`Signals in Region` : ports de l'entité.
`Signals in Design` : tous les signaux internes.

Il est aussi possible d'utiliser la commande `add wave nom_du_signal` ou `add wave /*` pour afficher tous les signaux de l'entité de plus haut niveau hiérarchique.

6.5. Construire des stimuli

Bien qu'il soit fortement recommandé de construire les stimuli avec un programme VHDL de test, voici un moyen simple pour créer rapidement des signaux de façon interactive à l'aide du simulateur.

La commande de base pour construire des formes d'ondes est la commande `force` (aussi disponible à partir du menu pop-up de la fenêtre `Objects`).
Exemples :

`force x 000`	positionne le signal `x` à la valeur `000` à partir de l'instant de simulation courant
`force x(1) 1`	positionne le `BIT 1` du signal `x` à la valeur `1` à partir de l'instant de simulation courant

6.6. Exécuter la simulation

L'exécution est lacée à partir du menu `Simulate` →`Run`.
La commande `Step` permet de suivre l'évolution pas à pas de la simulation en visualisant le code source.
La commande `run time` exécute t`ime` unité de temps de simulation (ex : `run 100`).
La commande `run time'high` suivante exécute la simulation pour toujours.

6.7. Analyser les résultats de simulation

Dans la fenêtre des chronogrammes, le curseur peut être déplacé ; les valeurs des signaux sont affichées, ainsi que le temps pointé par le curseur. Il est possible d'ajouter d'autres curseurs, de façon à effectuer des mesures d'intervalles de temps (menu `Add -> Cursor`).

6.8. Sauvegarder les résultats de simulation

Il est possible de sauvegarder les chronogrammes dans un fichier Postscript :
menu `File -> Print Postscript`, impression dans un fichier.

6.9. Créer et utiliser un fichier de commandes

Il peut être utile de pouvoir reproduire les stimuli qui ont été construits pour tester une unité de conception (Attention, cela fait double emploi avec les programmes de test écrits en VHDL). Pour cela, il faut créer un fichier qui reproduit les commandes utilisées pour créer les stimuli. Le nom du fichier n'est pas contraint, l'extension si elle existe peut être quelconque. Il est conseillé d'ouvrir un fichier texte dans le répertoire courant, et de le mettre à jour au fur et à mesure que les stimuli sont élaborés.
La commande `do nom_fichier` permet d'exécuter le fichier de commandes et ainsi reproduire les stimuli.

7. Paquetages standard et documentations

Cette partie présente les paquetages mis à la disposition du concepteur, et qui définissent un environnement standard pour la simulation et la synthèse. Les codes sources sont dans le répertoire `/softs/modeltech/vhdl_src/` :

- Paquetage `STANDARD` : ce paquetage définit les types `boolean`, `bit`, `character`, `severity_level`, `integer`, `real`, `time`, `string`, `bit_vector`, les sous types `natural`, `positive` ainsi que la fonction `now`.
- Paquetage `TEXTIO` : ce paquetage définit les primitives d'entrée/sortie ASCII de VHDL.
- Paquetage `STD_LOGIC_1164` : ce paquetage définit un type `bit` étendu à neuf états dénommé `STD_ULOGIC` et un sous type résolu dénommé `STD_LOGIC`. Il définit également les types composites `STD_ULOGIC_VECTOR` et `STD_LOGIC_VECTOR` ainsi que les fonctions logiques usuelles sur ces types.
- Paquetage `NUMERIC_BIT` et `NUMERIC_STD` : ces paquetages définissent les fonctions arithmétiques qui permettent de travailler sur des vecteurs de `BIT` et `STD_LOGIC_VECTOR` respectivement.
- Paquetage `STD_LOGIC_ARITH` : ce paquetage définit les fonctions arithmétiques de la même façon que les paquetages standard `NUMERIC_BIT` et `NUMERIC_STD`, mais il assure la compatibilité avec les versions antérieures des outils de synthèse.

Il suffit donc d'inclure les clauses suivantes au début des programmes pour avoir accès à ces paquetages :

```
library IEEE ;
use IEEE.STD_LOGIC_1164.all ;
use IEEE.NUMERIC_STD.all ;
```

ou

```
library IEEE ;
use IEEE.STD_LOGIC_1164.all ;
library arithmetic ;
use arithmetic.STD_LOGIC_ARITH.all ;
```

7. Modules de debug

Selon la syntaxe VHDL, seule les signaux locaux à une entité sont visibles. Ce choix garantie un bon partionnement de l'architecture pour la synthèse, mais il est parfois gênant lors de la simulation comportementale parce qu'il ne permet pas d'observer les signaux internes. Par exemple, lorsqu'on applique un delta à l'entrée de notre filtre, on est censé voir défiler les coefficients du filtre en tant que valeurs de sorties. Cependant à cause du gain de filtrage la pluparts de ces coefficients sont tronqués lors des calculs et n'atteignent pas la sortie, ce qui rend impossible d'écrire une routine de vérification directement dans le testbench. Pour pallier à ce problème, Modelsim fourni des fonctions «init_signal_spy » qui permettent de mapper des signaux internes sur des signaux locaux. Dans le bench fourni, il y a donc un « spy_process » qui vous permet d'observer aisément l'accumulateur du filtre et le signal de validation de sortie.
Ces modules ne sont bien sûr pas synthétisables, et ne font pas partie de la norme VHDL : un simulateur autre que Modelsim ne va pas les supporter.

8. Travail demandé

1ère Partie : étude du filtre FIR :

a) Comprendre et commenter l'ensemble du code source donné dans le répertoire `filter`.

b) Faire un chronogramme spécifiant le fonctionnement du système.

c) Extraire la machine à états du chronogramme. La représenter sous forme de graphe des transitions.

d) Compléter le fichier `fsm.vhd`. Il s'agit d'écrire l'architecture associée à l'entité `fsm` du contrôleur du filtre. Le contrôleur doit être tel que le filtre puisse s'interfacer avec le système de conversion analogique/numérique et numérique/analogique (cf. annexe).

e) Le fichier de test `bench.vhd` doit :
- Simuler le fonctionnement du microcontrôleur (charger les coefficients dans la RAM, et lancer le filtrage).
- Simuler le fonctionnement des convertisseurs, avec prise en compte des caractéristiques fonctionnelles et temporelles des signaux (cf. annexe).
- Permettre de vérifier la correction fonctionnelle du filtre (réponse impulsionnelle, réponse indicielle, réponse à une sinusoïde, ou à une somme de sinusoïdes). Le paquetage `ieee.math_real` contient, entre autre la fonction sinus...

Modifier le fichier VHDL de test `bench.vhd` en conséquence, de façon à permettre une grande flexibilité de spécification du signal d'entrée et à réaliser les validations de façon automatisée (clauses `assert`).

f) Simuler le filtre complet dans l'environnement de test développé pour vérifier son bon fonctionnement. Argumenter avec précision pourquoi le fonctionnement du filtre est correct.

2ème Partie : connexion avec le microcontrôleur :

a) Proposer un algorithme qui réalise l'écriture des coefficients dans la RAM du filtre en fonction de l'entrée de configuration du processeur et qui active le filtrage.

b) Coder cet algorithme en langage assembleur.

c) Coder en binaire ce programme afin de générer la ROM du uC.

d) Adapter le fichier `bench_top.vhd` afin de valider par simulation sous Modelsim le fonctionnement du programme.

e) Une fois validé, adapter le fichier `bench_top.vhd` et simuler l'ensemble du système (uC + filtre). A nouveau, le fichier de test doit permettre de:
- simuler le fonctionnement des convertisseurs et envoyer des stimuli sur l'entrée config du microcontrôleur,
- vérifier la correction fonctionnelle du système (réponse impulsionnelle, réponse indicielle, réponse à une sinusoïde, ou à une somme de sinusoïdes).

f) Valider le fonctionnement global. Argumenter avec précision pourquoi le fonctionnement est correct.

Un compte rendu synthétique, décrivant l'ensemble de vos travaux, est à rendre à la fin de la quatrième séance.