Utilisation des Outils de Développement de `Xilinx ISE`

Objectifs

Le but de ce TP est d'utiliser les outils de développement fourni par `Xilinx ISE`.

1 Design d'un composant en utilisant l'éditeur de schémas

Suivez le tutorial *Xilinx ISE WebPACK Schematic Capture Tutorial* pour découvrir l'éditeur de schémas.

Il faudra adapter certaines parties (le tutorial est fait pour une carte **Nexys 2**) ; en particulier lors de la création du fichier **ucf**, utilisez la bonne nomenclature des pins physiques (4 switchs et 1 LED) en vous inspirant du fichier *Nexys4.ucf*.

2 Mise en œuvre : clignotement d'un LED

Le circuit qui va permettre de faire clignoter un LED avec 4 fréquences différentes peut être représenté de manière schématique par le circuit de la Figure 1.

- les 3 entrées sont : l'horloge, un bouton pour le reset, un bouton de commande ;
- la sortie est la valeur de la LED ;
- le composant *debounce* (**fourni**) évite les rebonds qui peuvent avoir lieu au moment de l'appui sur un bouton ;
- les 4 diviseurs d'horloge (**fournis**) permettent de générer des horloges à 1Hz, 2Hz, 5Hz et 10 Hz ;
- le compteur compte 0, 1, 2, 3 et est commandé par le signal de commande "propre" en sortie du composant *debounce* ;
- le multiplexeur permet de choisir la fréquence de clignotement de la LED en fonction de la valeur en sortie du compteur ;
- le pilote de la LED (**fourni**) permet de faire clignoter la LED suivant une fréquence en entrée.

Nous allons générer les deux composants non fournis (le *multiplexeur* et le *compteur*) en utilisant des fonctionnalités de développement de `Xilinx ISE`.

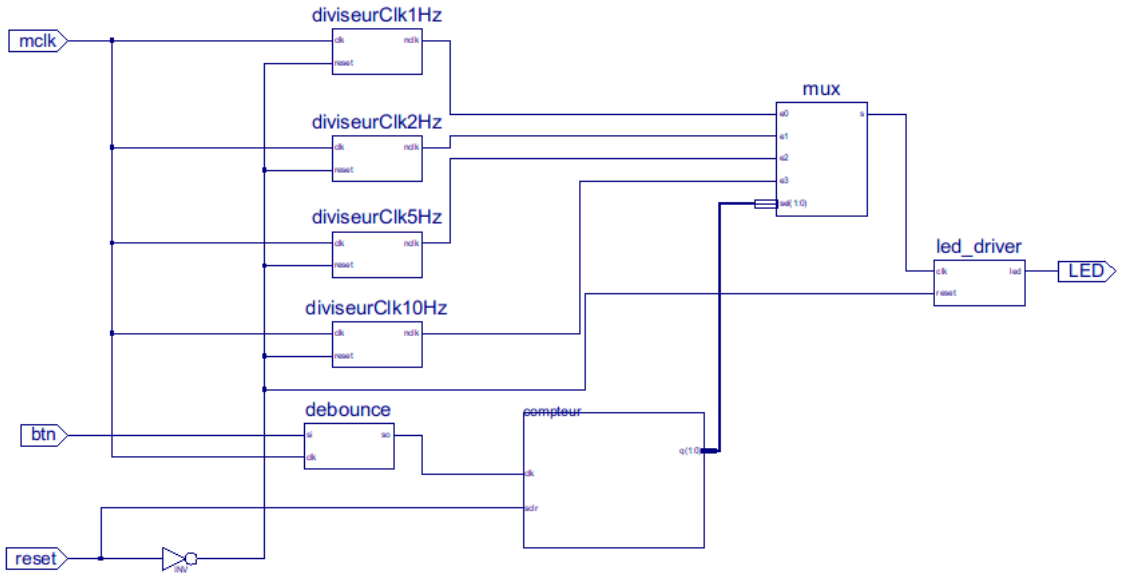

Figure 1 – Circuit de clignotement d'une LED

2.1 Utilisation des *Language Templates*

Pour générer le multiplexeur *4 vers 1*, nous utilisons une fonctionnalité de l'éditeur `Xilinx ISE` : les *Language Templates*.

Commencez par créer un nouveau composant VHDL

- création d'un nouveau *VHDL Module*, `mux4to1` (voir TP1, page 11). Vous devez bien vérifier que le répertoire de sauvegarde (*location*) est un répertoire de **Sources**,
- ce composant a 4 entrées à multiplexer avec 1 entrée pour faire la sélection de la valeur d'entrée qui est positionnée en sortie.

Pour remplir l'architecture du Multiplexeur :

1. sélectionnez **Edit** → **Language Templates**
2. suivez **VHDL** → **Synthesis Constructs** → **Coding Examples** → **Multiplexers** → **4-to-1**
3. positionnez-vous avec l'éditeur dans le fichier `mux4to1.vhd` et placez le curseur après le mot-clé **begin** de l'architecture
4. revenez à l'onglet *Language Templates* et avec un click droit sur **4-to-1**, exécutez la commande **Use in file**
5. adaptez les <noms> pour les faire correspondre aux signaux d'entrée/sortie de votre multiplexeur.

2.2 Génération d'un IP (CORE Generator & Architecture Wizard)

Modification de l'environnement

Pour commencer, modifier votre fichier `.bashrc` en rajoutant l'affectation de la variable d'environnement `XIL_CG_LOAD_ALL_FAMILIES` :

```
export XIL_CG_LOAD_ALL_FAMILIES=TRUE
```

Fermez `Xilinx ISE`, ouvrez un nouveau Terminal et relancez `Xilinx ISE`.

IP

Pour générer le compteur, nous utilisons un nouveau type de source : les IP (Intellectual Property, voir `http://www.xilinx.com/products/intellectual-property.html` pour plus d'information).

En suivant le menu **Project** → **New Source**, sélectionnez **IP** :

— appelez votre composant `compteur`

— choisissez comme *location*, votre répertoire de **Sources** **en créant (ou en conservant) le sous-répertoire ipcore_dir**.

Après un certain temps, une fenêtre de menu s'ouvre, vous donnant accès à un catalogue de composants optimisés pour votre carte.

Génération du compteur

Pour créer le compteur de notre circuit :

1. suivez **Basic Elements** → **Counters** → **Binary Counter**
2. Tapez **Next** puis **Finish**

Un nouvelle application s'ouvre alors : vous pouvez créer votre composant en lui donnant les entrées/sorties désirées :

1. changez la largeur du bus en sortie (2 bits),
2. rajoutez un reset synchrone (il n'y a pas de reset asynchrone ! !),
3. générez le composant avec **Generate** (cela peut prendre un certain temps, un indicateur dans `Xilinx ISE` permet de savoir quand c'est fini).

Si vous regardez maintenant le sous-répertoire **ipcore_dir** indiqué précédemment, un certain nombre de fichiers ont été créés. Voici quelques fichiers utiles :

1. `compteur.vhd`, le code VHDL du compteur,
2. `compteur.vho`, *template* pour intégrer le compteur comme sous-composant,
3. `compteur.sym`, le *Schematic symbol*,
4. `compteur.ngc` à rajouter dans un projet pour toute utilisation du compteur (inutile dans le cas présent, c'est fait automatiquement).

Vous pouvez consulter les 2 premiers fichiers et vérifier par exemple l'utilisation de le bibliothèque **Xilinx** pour le code du compteur.

2.3 Travail

1. si vous avez suivi les instructions, vous venez de créer le multiplexeur et le compteur
2. créez maintenant les *Schematic symbol* pour chaque composant
   — celui du compteur existe déjà,
   — pour les composants fournis et le multiplexeur : sélectionnez un composant et dans la sous-fenêtre **Design**, lancez la commande **Design Utilities → Create Schematic Symbol**,
3. créez le circuit (de nom `circuit`) en utilisant l'éditeur de schémas,
4. créez un fichier des contraintes `circuit.ucf` (partir du fichier *Nexys4.ucf*),
5. programmez la carte et testez le circuit.