Synthèse logique d'un circuit avec Alliance

L'objectif de ce TME est de vous faire faire un peu de modélisation de circuit. Dans le TP suivant, vous allez pouvoir placer et router le circuit que vous aurez décrit et validé.

Dans ce TME, vous allez travailler sur deux modèles: un PGCD et un CORDIC. Le PGCD va servir d'échauffement pour la synthèse et également un peu de modèle pour la validation de CORDIC. Le code du PGCD présenté dans le cours est presque complet, vous devez le compléter, le tester et en faire la synthèse. Le code de CORDIC est complet, mais vous allez devoir le faire évoluer. Il y a plusieurs degrés d'évolutions possibles. Si vous n'y arrivez pas, vous pouvez quand même expliquer votre démarche. L'idée, c'est comme si on vous donnait la version 1.0 d?un circuit et que vous devez fabriquer la version 2.0.

Pour le TP sur le placement-routage, vous routerez la version de base ou votre version.

PGCD

- Récupérez l'archive de PGCD.
- Ecrivez un fichier Readme.md donnant une explication succincte (sur une ligne) du rôle de chaque fichier.
- Complétez le fichier `pgcd_core.vhd`
- Validez-le avec le Makefile
- Que devriez-vous faire pour augmenter la précision des nombres ? (Vous pouvez le faire, mais ce n?est pas obligatoire)
- Faites la synthèse sur SXLIB avec BOOM, BOOG et LOON. Vous devez faire évoluer le Makefile et là vous avez deux possibilités:
    1. Vous ajoutez une règle `synthesis:` avec une cible sans dépendance dans laquelle vous mettez la séquence des outils à lancer (boom, boog, loon, asimut , etc.). Cette règle est donc comme un script shell a l?instar des règles déjà présentes dans le Makefile.
    2. Vous décrivez un Makefile dans lequel les règles ont la forme `fichier_produit : liste de fichiers sources`. Chaque règle ne fait qu?une étape de la conception ou de la vérification, une règle pour `boom`, une règle pour `boog`, etc.

       La seconde possibilité offre l?avantage d?être explicite du point de vue des fichiers produits et permet une reconstruction partielle. Cependant, elle est plus complexe à décrire et finalement pas très utile. Parce que l?intérêt principal du Makefile, c?est de décrire la séquence des outils et leur arguments, ce que fait mieux la première possibilité. En effet, c?est plus compact et donc plus simple à comprendre. J?ai donc une préférence pour la première possibilité.

- Vous devez aussi vérifier la synthèse par simulation, d?où le lancement d?asimut après la synthèse logique.

CORDIC

Fonction

Le circuit réalise la rotation d'un vecteur (x,y) par un angle a et produit le vecteur (nx,ny)

- Le circuit prend en entrée
    - ♦ les coordonnées x_p et y_p qui sont des nombres entiers signés de -127 à +127.
    - ♦ L'angle a_p est exprimé en radian et il est représenté par un nombre en virgule non signé 3-5.
        - ◊ À l'intérieur du circuit, c'est un nombre en virgule fixe 1-8-7.

    - ◊ Mais, à l'interface, j'ai choisi une représentation non signée 3-5 (port a_p) pour avoir des angles entre 0 et presque 8 radians. La conversion se fait dans le circuit en recopiant les 8 bits de a_p dans les 8 bits de poids faible d'un registre de 16 bits représentant l'angle, puis en complétant avec des `0`à gauche. C'est un choix pour réduire le nombre de broches.
- le circuit reçoit aussi une horloge et un signal reset.
- Le circuit produit en sortie les coordonnées (nx_p, ny_p) du vecteur après rotation.
- Le protocole de communication en entrée et en sortie est FIFO.

Question

- Récuperez l'archive de CORDIC
- Ecrivez un fichier Readme.md donnant une explication succincte (sur une ligne) du rôle de chaque fichier.
- Que fait `make plot`?
- Expliquez quel est le principe de la validation utilisé pour CORDIC et la différence avec celle utilisée pour PGCD.
- Pourquoi celle utilisée pour PGCD est préférable ?

- compléter le modèle VHDL du circuit CORDIC à partir d'une description de l'algorithme en décrit en C.
- valider le fonctionnement avec des patterns que vous pouvez produire à la main, ou avec genpat, pour en créant un *test bench* en vhdl.
- faire la synthèse logique sur SXLIB.
- écrire le compte-rendu de vos travaux et décrire les résultats obtenus.
- Le même circuit sera repris les deux semaines suivantes alors vous pourrez ne rendre qu'un seul compte rendu reprenant les étapes.

Fichiers fournis

- **Spécification de haut niveau**

  La spécification de haut niveau (non RTL) est donnée dans un programme C.
  C'est ce programme qui sert de référence pour le modèle RTL.

    - ♦ cercle.c : contient deux fonctions qui calculent un cercle en faisant tourner vecteur (127,0). La première fonction utilise les fonctions `sin()` et `cos()`, la seconde utilise l'algorithme cordic(). Les deux fonctions produisent un fichier .dat avec les coordonnées que gnuplot peut afficher.

    - ♦ Makefile : compile et execute cercle.c puis lance gnuplot pour afficher les deux cercles.

- **Modèle VHDL 1 core**

  Vous allez modéliser le circuit, mais comme ce n?est pas si simple (ça prend du temps), je vous donne un fichier vhdl incomplet, qui va vous aidez (en principe). Dans ce fichier, j'ai fait des choix que vous êtes libres de changer, mais vous devez avoir en tête que ce modèle 1 core, sera découpé en deux parties la semaine prochaine : une partie contrôle et une partie chemin de données. Le modèle que je vous propose me semble assez simple à couper.

  Parmi les choix qui peuvent faire débat, il y a la description de l'automate. J'utilise un codage One-Hot parce que je trouve que c'est lisible et c'est très efficace (en fait tous les circuits que auxquels j'ai participé ont utilisé cette approche...).

  Je ne vous donne pas de patterns. Je vous conseille de créer un *test benchs* qui produit et consomme des données et que vous devez connecter à votre circuit. Si vous n'y arrivez pas, je vous donnerai de quoi

valider le circuit la semaine prochaine.

Notez que si vous avez une manière plus élégante de faire des shifters, allez-y, mais il faut que `vasy`accepte, or j'ai bien peur que les for generate (solution possible) ne soient pas acceptés.

- cordic_cor.vhd