TP1 : Conception de circuits simples avec des outils de CAO pour FPGA

Travail Préparatoire : Lire l'énoncé avant l'arrivée en séance de TP.
Objectifs du TP :

- Apprendre à représenter des circuits simples avec un langage de description d'architecture (Hardware Description Language, HDL)
- Comprendre les étapes fondamentales des flots de conception de circuits numériques : simulation, synthèse logique, programmation de la carte FPGA

Ex. 1 : Découverte et prise en main de l'environnement de TP

Introduction

Comme beaucoup de tâches d'ingénieries complexes, la conception de circuit numérique repose sur des outils de CAO (Conception Assistée par Ordinateur) pour décrire, simuler et générer le circuit pour la cible matérielle retenue (ASIC, FPGA). Dans le cadre de ces TPs, nous utilisons le langage de description d'architecture VHDL pour décrire les circuits et nous les implantons ensuite sur une cible FPGA, c'est à dire un réseau programmable de portes combinatoires (principalement des petites tables de vérité nommées LUTs) et de portes séquentielles. Le FPGA utilisé est intégré dans la puce ZYNQ au milieu de la carte ZYBO mise à votre disposition pour les TPs (Lien vers la documentation). Il est composé de 18K LUT à 6 entrées, de 35K bascules flip-flop, de 240Ko de mémoires embarquées et de 80 opérateurs arithmétiques (multiplieur 18 bits en complément à 2, additionneur/soustracteur/accumulateur sur 48 bits). Outre ce FPGA, la carte embarque aussi un processeur cortex A9 de ARM (deux cœurs tournant à 650MHz), 512 Mo de mémoire DDR3, des interfaces d'entrée-sortie (5 LED, 4 interrupteurs et 6 boutons poussoirs) et une connectique riche : HDMI, VGA, Ethernet, USB OTG, jacks audio, un port série, un port JTAG (sur USB) pour la programmation du FPGA et des ports d'extension PMOD.

La prise en main des outils de CAO pour FPGA est fastidieuse et peu intéressante au vu des objectifs des TP. Pour vous l'épargner, nous avons créé un flot simplifié automatique utilisant make [1]. Toutes les commandes données dans l'énoncé sont à taper dans un terminal positionné dans le dossier fourni sur Chamilo (téléchargez l'archive et décompressez-la dans un répertoire archi/tp1 sur votre compte info).

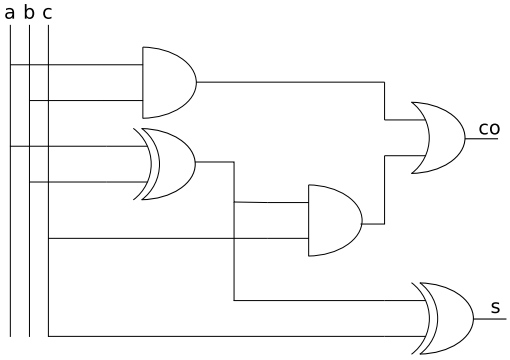
Si vous lancez le make dans un mauvais répertoire, ce dernier vous dira *Pas de règles pour fabriquer la cible* $\cdots$. Pour fonctionner correctement, votre terminal doit initialiser les variables de l'environnement de CAO. Pour cela, utiliser comme suggéré :

```
source /bigsoft/Xilinx/Vivado/2019.1/settings64.sh
```

Conception, simulation et implantation d'un circuit simple

Pour illustrer le flot de conception, nous allons prendre comme exemple le circuit suivant :

1. Outil communément utilisé pour automatiser des tâches de génération.



Description du circuit en VHDL

La syntaxe VHDL vous sera introduite au fur et à mesure dans les commentaires. Une fiche récapitulative est à votre disposition sur Chamilo (*Documents/TP/cheatsheetVHDL.jpeg*).

Bien que la syntaxe du VHDL soit très proche de celle d'un langage de programmation, il faut garder à l'esprit que la finalité du langage VHDL est de décrire des circuits et non des algorithmes. Il est donc impératif d'avoir en tête le circuit que l'on veut représenter ou de l'avoir dessiné, avant d'écrire du code VHDL.

Question 1 Le fichier *vhd/AC.vhd* contient un squelette de description pour le circuit à étudier. Ouvrez ce fichier avec votre éditeur de texte préféré. Lisez le et complétez la partie manquante.

Question 2 Effectuez la synthèse logique de votre circuit, i.e. sa traduction en portes logiques telles qu'elles vous ont été présentées en TD, en utilisant la commande :

```
$ make synthese TOP=AC
```

Analysez le résumé des rapports de synthèse affiché dans la console, et plus particulièrement les parties "Needed Primitives" et "Real Utilization", pour vérifier les ressources utilisées.

Simulation comportementale

Un des grands avantages des langages HDL réside dans la possibilité de simuler le comportement des circuits décrits. Outre les entrées-sorties du circuit, la simulation permet d'accéder à l'ensemble des signaux internes du circuit. Un atout incontournable pour déverminer un circuit ! Afin de simuler un composant, il faut créer un banc d'essai (testbench) qui instancie ce composant et lui fournir en entrée un ensemble de stimuli.

Question 3 Ouvrez le fichier *vhd/tb_AC.vhd* pour étudier le banc d'essai proposé et en comprendre les subtilités (mini-tutorial inclus). Quelles sont dans ce fichier les entrées générées ?

Question 4 Lancez la simulation avec l'outil `Vivado` sur ce banc d'essai en utilisant la commande :

```
$ make run_simu TOP=AC
```

Comme prévu dans le banc de test, le simulateur a atteint l'assertion finale et a basculé sur cette dernière dans le code source. Cliquez sur la fenêtre du chronogramme (onglet juste à côté) puis sur le bouton Zoom Fit (ou dans le menu View -> Zoom fit). Vérifiez sur le chronogramme que les résultats obtenus sont corrects.

Implantation sur la carte FPGA

Après avoir vérifié le bon fonctionnement de notre circuit, on peut l'implanter sans crainte sur la cible matérielle, ici un FPGA. Le FPGA est relié aux différents périphériques par des pattes (*i.e.* les petits fils qu'on voit sortir de la puce) numérotées. Si on veut utiliser ces périphériques, il faut donc préciser l'association entre le nom des entrées/sorties logiques de notre circuit et les numéros des pattes correspondant aux périphériques à utiliser. Le fichier *AC_xc7z010-clg400-1.xdc* permet à l'utilisateur de spécifier ces associations.

Question 5 D'après ce fichier, à travers quelle interface pourra t'on modifier a, b et c, et visualiser co et s ?

Avec ces informations, le flot de conception est capable de générer un fichier décrivant la programmation du FPGA. Cette opération prend un temps conséquent au vu des nombreuses étapes à réaliser [2]. Il faut ensuite transférer le fichier généré vers la carte pour la programmer.

Question 6 Branchez maintenant la carte FPGA en connectant par câble USB votre PC au connecteur mini-USB PROG de la carte ZYBO, puis positionnez l'interrupteur sur ON. Programmez le FPGA en tapant la commande :

```
$ make run_fpga TOP=AC
```

Actionnez les interrupteurs et vérifiez le bon fonctionnement en observant les LEDs.

Si tout cela ne fonctionne pas comme attendu, voici des pistes pour chercher d'où vient le problème.

1. Pour vérifier que la carte ET le câble sont opérationnels, taper la commande :
   ```
   $ lsusb
   ```
   et devrait s'afficher :
   ```
   1Bus 001 Device 004: ID 0403:6010 Future Technology Devices International, Ltd FT2232C Dual USB/FIFO IC
   ```
   Sinon changer le câble ou la carte.
2. Pour vérifier que le driver digilent est installé et communique bien avec la carte utilisée taper la commande :
   ```
   $ djtgcfg enum
   ```
   et devrait s'afficher :
   ```
   djtgcfg enum
         Found 1 device(s)

         Device Zybo
         Product Name: Digilent Zybo
         User Name: Zybo
         Serial Number: XXXXX
   ```
   Si ce n'est pas le cas, changer de port... Sinon tout est bon et on peut passer à la programmation.

2. ajoutez *VERB=1* au bout de vos lignes make permet de voir le détail de ces étapes. Pratique pour corriger vos erreurs.

Ex. 2 : Diviseur de fréquence

Pour travailler la logique séquentielle, nous allons nous intéresser au circuit diviseur de fréquence, fourni dans le fichier *vhd/div_freq.vhd*.

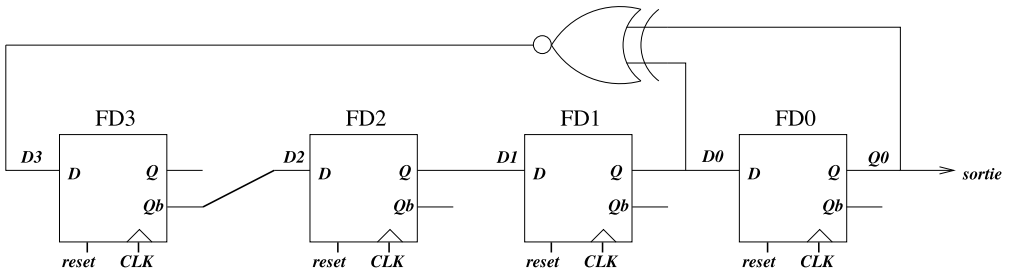
Question 1 Lisez le code fourni et dessinez le circuit correspondant.

Question 2 Simulez (`make run_simu TOP=div_freq`) et retrouvez le fonctionnement du diviseur de fréquence.

Question 3 L'entrée `reset` fonctionne de manière asynchrone. Modifiez le code pour la rendre synchrone et vérifiez en simulation.

Ex. 3 : Réalisation d'un circuit LFSR

Un LFSR (c.f. TD3) intègre à la fois de la logique séquentielle et de la logique combinatoire. On utilisera le circuit ci-dessous, qui produit la séquence cyclique "000100110101111" :



Question 1 Décrivez en VHDL le circuit ci-dessus en utilisant le squelette de code *vhd/lfsr.vhd*.

Question 2 Simulez (make run_simu TOP=lfsr) pour vérifier le bon fonctionnement. Si vous le souhaitez, vous pouvez ajouter dans le banc de test (fichier *vhd/tb_lfsr.vhd*) des assertions pour vérifier automatiquement la séquence (comme montré dans le banc de test de l'exercie 1). Lorsque le résultat d'une simulation n'est pas correct, il faut regarder le comportement à l'intérieur du circuit. Le simulateur nous permet d'observer des signaux internes. Pour ajouter un signal interne, faites apparaître si besoin l'encart "Scope" et "Objects" (dans le menu Window), puis cliquez sur la flèche du banc de test (ici, "tb_lfsr"), puis cliquez sur le composant sous test (ici, "C_lfsr"), ajoutez enfin les signaux intéressants par glisser-poser de l'encart "Objects" vers le chronogramme. Pour mettre à jour le chronogramme, il faut redémarrer la simulation ("restart" puis "run for" avec sablier). Attention, à ne jamais utiliser le bouton "run all". Ce dernier lance le simulateur indéfiniment, ce qui va très rapidement saturer la mémoire de votre espace de travail.

Question 3 Effectuez la synthèse logique de votre circuit. Comparez le rapport de synthèse à celui de l'exercice précédent. Que constatez-vous ?

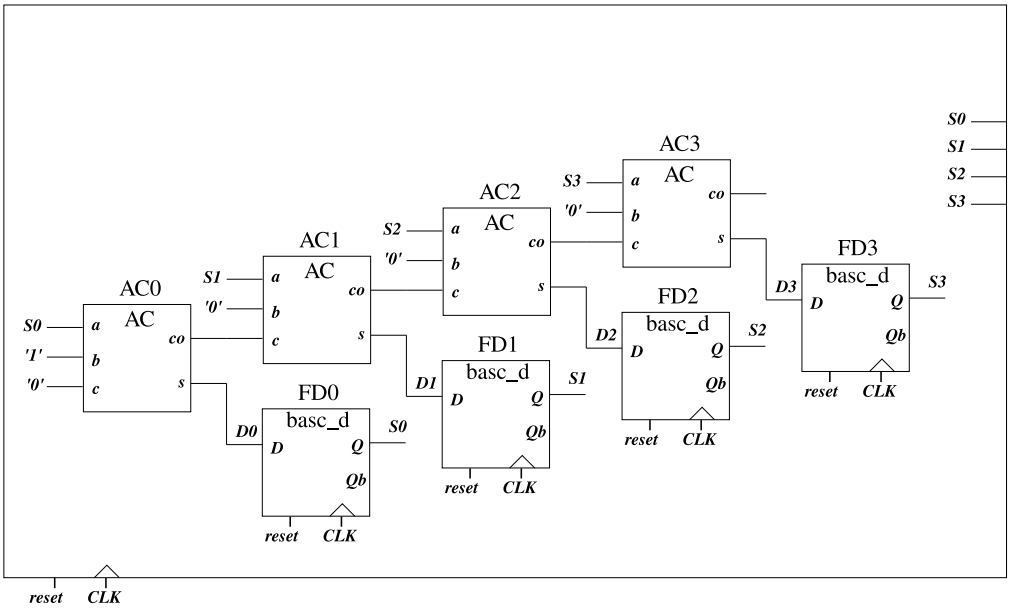
Question 4 Pour passer le composant lfsr sur carte, nous l'avons encapsulé dans le fichier *vhd/lfsrfpga.vhd*. Ouvrez ce fichier pour comprendre le problème traité, puis testez sur carte avec la commande : `$ make run_fpga TOP=lfsrfpga`

Pour aller plus loin...

Ex. 4 : Circuit bonus

Voilà un petit exercice pour occuper les plus rapides, qui permet de reprendre les notions abordées dans la séance et de préparer la prochaine séance de TD.

Question 1 Implantez le circuit décrit dans le schéma suivant dans le squelette de code *vhd/bonusfpga.vhd*. Avec le fichier *xdc* fourni, vous pouvez le tester sur carte. Si sa fonction ne vous apparaît pas clairement, c'est surement que vous avez fait une erreur. Il faut alors créer un banc de test (fichier *vhd/tb_bonusfpga.vhd* pour simuler votre composant.


AC0
AC
a
b
c
co
s
S0
'1'
'0'
AC1
S1
D0
FD0
basc_d
D
Q
Qb
reset
CLK
S0
AC2
S2
D1
FD1
S1
AC3
S3
D2
FD2
S2
FD3
D3
S3