TP2 : 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 : Prise en main sur un exemple connu

Introduction

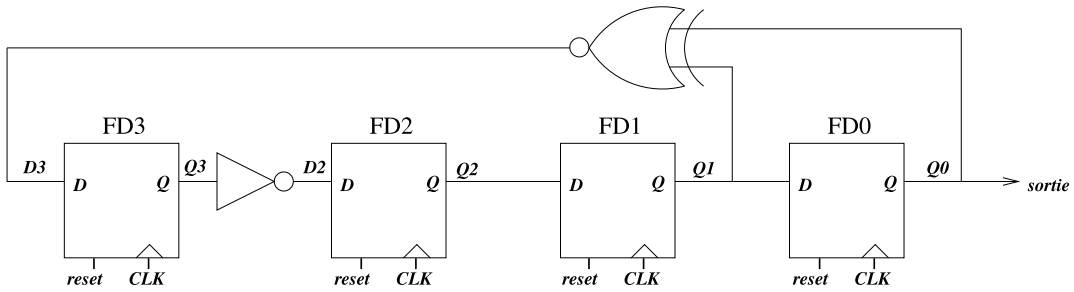
Contrairement à ce que nous avons fait dans la première séance, un ingénieur souhaitant concevoir un circuit numérique n'en assemble pas directement les transistors ou les portes logiques, car la complexité de la tâche le limiterait à des circuits très simples. Il s'appuie donc sur des outils de CAO (Conception Assistée par Ordinateur) pour décrire, simuler et générer son circuit pour la cible matérielle retenue (ASIC, FPGA). Nous utilisons à l'Ensimag 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. 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 et 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].

Conception, simulation et implantation d'un circuit simple

On va prendre comme exemple un circuit étudié lors des TD et TP précédents : le LFSR, dont on rappelle le schéma logique ci-dessous.

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



Description du circuit LFSR en VHDL

Dans ce TP, nous utilisons le langage de description d'architecture (HDL, Hardware Description Language) nommé VHDL. Attention la syntaxe du VHDL est très proche de celle de ADA. Gardez à l'esprit que la finalité de ce langage est de décrire des circuits et non des algorithmes. C'est pourquoi il est impératif d'avoir en tête le circuit que l'on veut représenter ou de l'avoir dessiné, avant d'écrire du code VHDL.

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 à l'endroit de votre choix).

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 /Xilinx/env-14.7.sh
```

Question 1 Le fichier *vhd/lfsr.vhd* contient un squelette de description pour le circuit LFSR. Les commentaires fournis forment un mini-tutoriel qui vous permettra d'appréhender aisément le vaste monde du VHDL. Ouvrez ce fichier avec votre éditeur de texte préféré et complétez le.

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=lfsr
```

Nous allons maintenant vérifier le circuit généré avec le logiciel ISE.
Tapez `ise` dans votre console, puis ouvrez le fichier *lfsr.ngr* généré par la synthèse (File, open...). Dans la fenêtre de dialogue qui apparaît, sélectionnez *Start with a schematic of the top level block*. Vous devriez obtenir une vue graphique de votre composant *lfsr*, avec ses deux entrées et sa sortie. Double-cliquez sur le composant pour observer son contenu (adaptez le niveau de zoom avec F6). Vérifiez que le schéma obtenu correspond au schéma donné dans l'énoncé.

Question 3 Fermez ISE puis reprenez le fichier *vhd/lfsr.vhd*. Inversez l'ordre de 2 affectations concurrentes dans la description (par exemple le `xnor` et le `not`), et observez le circuit généré (n'oubliez pas de relancer la synthèse, puis de fermer et rouvrir les fichiers générés). Que constatez-vous ?

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 4 Ouvrez le fichier *vhd/tb_lfsr.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 5 Lancez la simulation avec l'outil `ISim` sur ce banc d'essai en utilisant la commande :

```
$ make run_simu TOP=lfsr
```

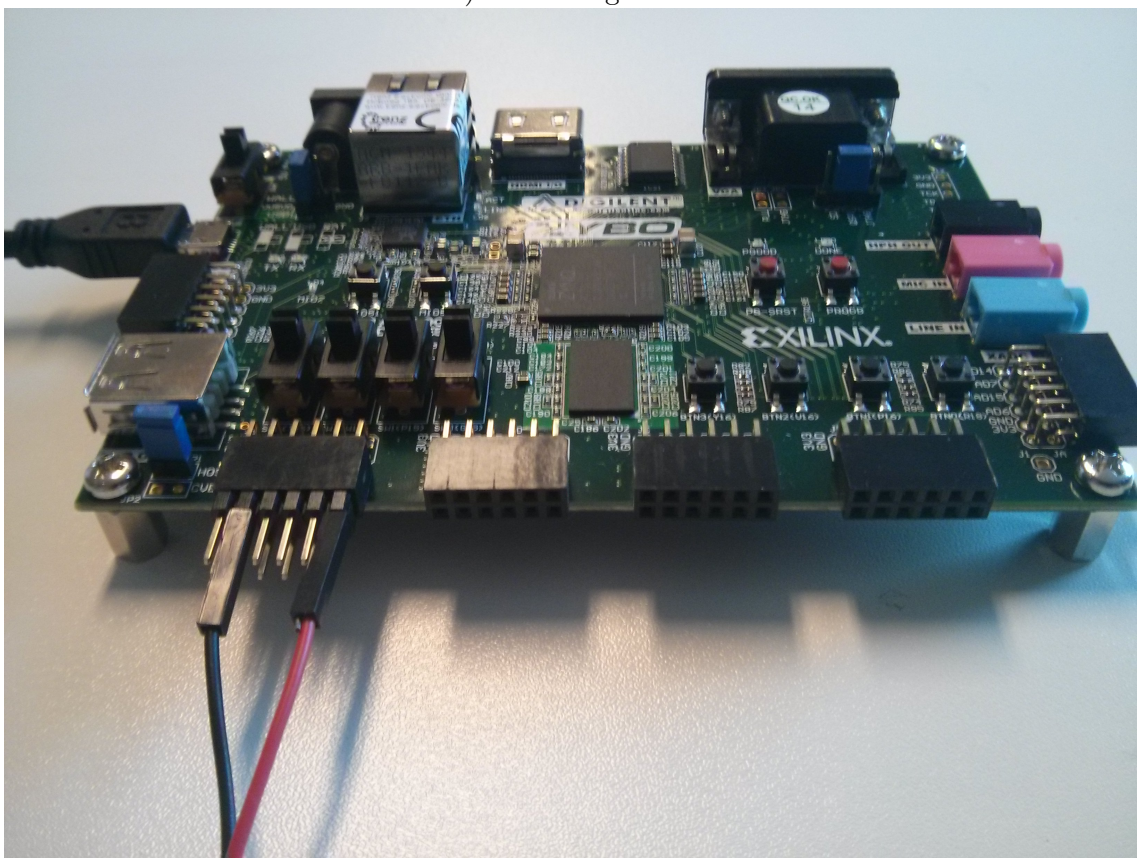
et appuyez sur F6 pour adapter le niveau de zoom. Vérifiez sur le chronogramme que le résultat attendu est correct.

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 et sorties logiques des schémas (*e.g.* les entrées `clk` et `reset` et la sortie `s` du LFSR) et les numéros des pattes correspondant aux périphériques à utiliser. Le fichier *lfsr.ucf* permet à l'utilisateur de spécifier ces associations. 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. Pour cela :

- Connectez par câble USB votre PC au connecteur mini-USB PROG de la carte ZYBO.
- Connectez l'horloge du GBF (sortie TTL/CMOS) sur le connecteur PMOD `JE` (signal sur l'entrée 1 et masse sur l'entrée 5). Voir la figure ci-dessous.



- Positionnez l'interrupteur sur ON.

Programmez le FPGA en tapant la commande :

```
$ make run_fpga TOP=lfsr
```

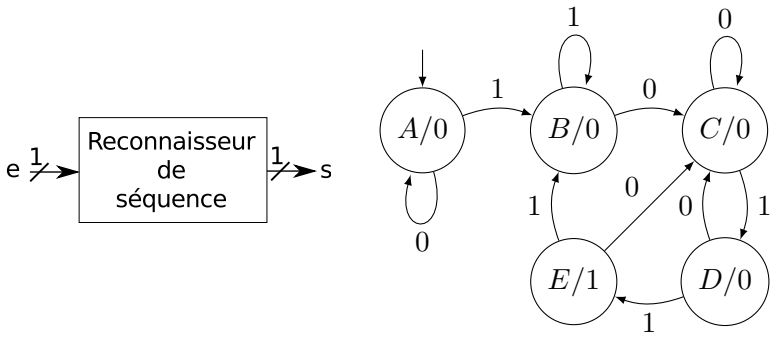
Une fois la carte programmée avec succès, réglez la fréquence du GBF sur $2Hz$ pour avoir un

2. ajoutez *VERB=* au bout de vos lignes make permet de mieux voir ces étapes et d'en comprendre l'utilité. A ne faire qu'après avoir corrigé vos erreurs.

affichage lisible sur les LEDS de la carte FPGA.

Ex. 2 : Implantation d'un automate reconnaisseur de séquence

De manière similaire à ce qui a été vu en TD, on peut construire un automate pour reconnaitre la séquence 10^+11. En voici l'interface, le graphe d'états (Moore) et un codage des états :



État	q_2	q_1	q_0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0

Par la méthode des tableaux de Karnaugh, on peut obtenir les équations minimisées des transitions et de la sortie de l'automate.
$d_2 = q_1.q_0.e$
$d_1 = q_1.\overline{q_0} + q_2.\overline{e} + q_0.\overline{e} = q_1.\overline{q_0} + \overline{e}.(q_2 + q_0)$
$d_0 = \overline{q_0}.e + \overline{q_1}.e = e.(\overline{q_0} + \overline{q_1})$
$s = q_2$

Question 1 Dessinez le circuit correspondant à cet automate.

Question 2 Complétez le fichier *auto.vhd* afin d'y décrire l'automate dessiné.

Question 3 Vérifiez le circuit généré en tapant :

```
$ make synthese TOP=auto
```

puis ouvrez le fichier *auto.ngr* dans ISE. Simulez son fonctionnement grâce au testbench *tb_auto.vhd* en tapant :

```
$ make run_simu TOP=auto
```

Le testbench *tb_auto.vhd* fourni génère une suite de stimuli qu'il applique en entrée de votre automate. Cette suite est construite de manière à tester toutes les transitions de l'automate. Pour chaque entrée, le testbench regarde la sortie de l'automate et la compare à une valeur de référence. Si les deux valeurs diffèrent, cela signifie que vous avez une erreur dans votre automate. Le simulateur vous préviendra par un message d'erreur dans sa console, du type

```
at 120 ns: Error: ==== ERREUR: Sortie de l'automate inattendue ====
```

En revanche, le message `** Failure:Simulation terminée` n'est pas une erreur en soit, mais indique simplement que la suite de stimuli a été entièrement appliquée.

Ex. 3 : Assemblage du LFSR et de l'automate reconnaisseur de séquences

Question 1 Complétez le fichier *global.vhd* de manière à y instancier un composant LFSR connecté à un composant reconnaisseur de séquences.

Question 2 Vérifiez en simulation le circuit global à l'aide de la commande

```
$ make run_simu TOP=global
```

Il faut que la sortie de l'automate passe bien à 1 lorsqu'on détecte la séquence $10^{+}11$ en sortie du LFSR.

Implantation sur la carte FPGA

Pour générer le circuit global sur la carte FPGA, le flot utilise le fichier de contraintes `global.ucf`.

Question 3 Adaptez au besoin ce fichier et programmez la carte en tapant

```
$ make run_fpga TOP=global
```

Une fois la carte programmée, on doit voir s'afficher sur la LED 0 la sortie de l'automate, qui s'allumera donc quand la séquence a été détectée.

Question subsidiaire 1

S'il vous reste du temps, reliez la carte FPGA à l'oscilloscope pour afficher la sortie du LFSR et celle de l'automate reconnaisseur de séquences. Pour cela, vous pouvez rajouter une nouvelle sortie dans le fichier `global.vhd` en décommentant les 2 lignes impliquant `lfsr_o`. Décommentez deux lignes de fichier `global.ucf` de manière à relier la sortie `lfsr_o` à la patte 2 du PMOD `JE` et la sortie `s` à sa patte 3. Branchez deux sondes sur les pattes du PMOD `JE` et utilisez la troisième entrée de l'oscilloscope pour l'horloge sortant du GBF, réglé à $2KHz$.

Il va de soi qu'il faut reprogrammer la carte avec vos modifications.

Question subsidiaire 2

S'il vous reste du temps, refaites le circuit de l'automate reconnaisseur de séquences en choisissant cette fois-ci un codage 1 parmi N.