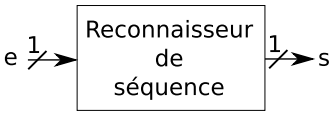
TP2 : Automates et compteurs

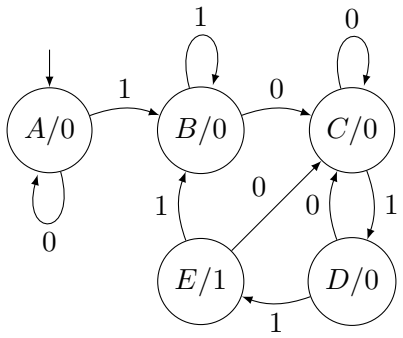
Travail Préparatoire : Lire l'énoncé avant l'arrivée en séance de TP.
Objectifs du TP :

- Maîtriser le codage systématique des automates (machines de Moore) en circuits
- Réaliser des compteurs, qui seront souvent réutilisés
- Apprendre à utiliser et fabriquer des composants *génériques* en VHDL

Ex. 1 : Implantation d'un automate reconnaisseur de séquence (codage compact)

De manière similaire à ce qui a été vu en TD, on peut construire un automate pour reconnaître la séquence 10^+11. En voici l'interface, le graphe d'états (Moore) et une proposition de codage compact 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 (rappel : on utilise 3 bascules correspondant aux bits q_0, q_1, q_2 du codage compact, dont les entrées de données sont d_0, d_1, d_2) :

$$
\begin{aligned}
d_2 &= q_1.q_0.e \\
d_1 &= q_1.\overline{q_0} + q_2.\overline{e} + q_0.\overline{e} = q_1.\overline{q_0} + \overline{e}.(q_2 + q_0) \\
d_0 &= \overline{q_0}.e + \overline{q_1}.e = e.(\overline{q_0} + \overline{q_1}) \\
s &= q_2
\end{aligned}
$$

Question 1 Dessinez le circuit correspondant à cet automate.

Question 2 On utilise la même bascule que dans le TP1. Voir le fichier `vhd/basc_D.vhd`. Complétez le fichier `auto.vhd` afin d'y décrire l'automate dessiné.
Indication : si vous utilisez un composant fourni sans avoir besoin de toutes ses sorties, vous pouvez déclarer qu'un fil de sortie est "*en l'air*", c'est-à-dire connecté à rien. Dans un `port map` on écrit par exemple : `qb => open`, où `open` est un mot clé de VHDL.

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
```

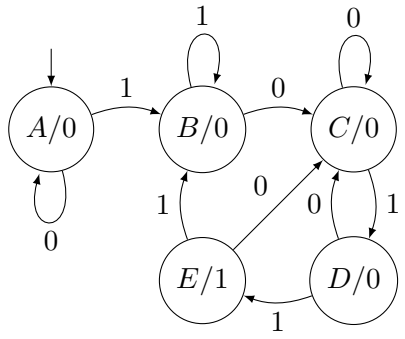
Le testbench `tb_auto.vhd` fourni génère une suite de stimuli qu'il applique en entrée de votre automate. Cette suite est construite de manière à tester toutes les transitions de l'automate. Pour chaque entrée, le testbench regarde la sortie de l'automate et la compare à une valeur de référence. Si les 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. 2 : Implantation d'un automate reconnaisseur de séquence (codage *1 parmi n*)

Le but de cet exercice est de reprendre l'exercice précédent, avec un codage *1 parmi n* à la place du codage compact. Voici de nouveau l'automate, et une proposition de codage *1 parmi n* :



État	q_4	q_3	q_2	q_1	q_0
A	0	0	0	0	1
B	0	0	0	1	0
C	0	0	1	0	0
D	0	1	0	0	0
E	1	0	0	0	0

Les équations découlent directement de l'automate (on utilise autant de bascules que d'états, donc 5 ici, dont les entrées de données sont d_4, d_3, d_2, d_1, d_0) :

$$
\begin{aligned}
d_0 &= q_0.\overline{e} \\
d_1 &= q_0.e + q_1.e + q_4.e = (q_0 + q_1 + q_4).e \\
d_2 &= q_1.\overline{e} + q_2.\overline{e} + q_4.\overline{e} + q_3.\overline{e} = (q_1 + q_2 + q_3 + q_4).\overline{e} \\
d_3 &= q_2.e \\
d_4 &= q_3.e \\
s &= q_4
\end{aligned}
$$

On vous fournit maintenant deux bascules, l'une initialisée à 0, l'autre à 1 (voir fichiers `vhd/basc_D.vhd` et `vhd/basc_D_set.vhd`).

Question 1 Modifiez le fichier `vhd/auto_unparmin.vhd` afin d'y coder l'automate selon les équations ci-dessus :
— ajoutez des signaux et instanciez 5 bascules D pour le codage 1 parmi n, attention à la bascule qui correspond à l'état initial
— écrivez les équations ci-dessus

Question 2 Reprenez toute la démarche de l'exercice 1 (synthèse, observation du circuit, simulation avec `tb_auto_unparmin.vhd`

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 (Cf. TP 1) connecté à un automate reconnaisseur de séquences. Vous devez récupérer votre composant LFSR du TP1 et le mettre dans `vhd/lfsr.vhd`, ainsi que reprendre le codage de votre automate dans `vhd/auto.vhd`.

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.

Ex. 4 : Compteurs

Compteur 4 bits

Dans l'exercice 4 du TP1, on vous demandait de réfléchir à un circuit composé de 4 composants AC et 4 bascules. La connexion entre les 4 composants AC doit être rapprochée de l'exercice 2 du TD 4 sur les opérations arithmétiques.

Dans cet exercice nous allons introduire les vecteurs et les opérateurs arithmétiques, qui permettent de décrire de tels circuits de manière beaucoup plus abstraite, sans toujours revenir à la connexion des additionneurs 1 bit.

Observez le fichier fourni `vhd/compteur4.vhd` (documentation incluse). C'est un circuit avec deux entrées `clk, reset` et une sortie `cpt` qui est un vecteur de 4 bits interprétable comme un entier non signé, dans $[0, 15]$ donc. On fournit un composant `reg4b` (registre 4 bits) pour mémoriser un tel entier 4 bits. Si l'entrée `reset` est toujours fausse, le compteur doit compter modulo 2^4 : 0, 1, 2, ..., 14, 15, 0, 1, 2, ... Quand l'entrée `reset` est à 1, il revient à 0 même s'il n'avait pas atteint 15.

Question 1 Réfléchir à l'effet de l'entrée `reset` : que vaut la sortie du compteur au moment où `reset` passe à 1 ? Et à l'instant suivant ?

Question 2
— Complétez le fichier `vhd/compteur4.vhd` en instanciant un registre 4 bits, et en écrivant les équations nécessaires pour réaliser le compteur 4 bits avec reset.
Remarque : les variables `dd` et `curval` sont des entiers (type `unsigned`) donc on peut utiliser des opérateurs arithmétiques.
— Si vous avez le temps, même question en maintenant toujours à 0 l'entrée `reset` du registre 4 bits : comment implanter un `reset` sans passer par celui du registre ?
— Réalisez la synthèse (`make synthèse TOP=compteur4`) et observez `compteur4.ngr` dans ISE.

Pour tester le circuit, on utilise le testbench `vhd/tb_compteur4.vhd`, qui provoque un reset avant que le compteur ait atteint la valeur 15.

Question 3
— Pouvez-vous deviner quelle séquence de valeurs vous allez observer en sortie du compteur avec ces stimuli d'entrée ?
— Simulez votre circuit (`make run_simu TOP=compteur4`) et vérifiez si vous avez deviné juste.

Lecture : réalisation du compteur 4 bits en utilisant un registre générique

Vous aurez constaté dans l'exercice précédent que la taille choisie (4) intervient souvent mais de manière très systématique dans le code VHDL. Nous allons ici introduire la notion de *generic* en VHDL, qui est faite pour généraliser des descriptions de circuits. L'idée ici est de remplacer 4 par n partout.

On étudie ici le composant décrit dans `vhd/reggene.vhd`. Ce composant a un paramètre de plus que le composant `reg4b` utilisé précédemment (fichier `vhd/reg4b.vhd`), spécifié avec le mot-clé `generic`.

Question 4 Comparez `vhd/reg4b.vhd` et `vhd/reggene.vhd` pour voir comment on remplace 4 par n.

On observe maintenant `vhd/compteur4avecreggene.vhd`, qui montre comment utiliser un tel composant générique. Notez bien l'utilisation de `generic map` pour spécifier la valeur du paramètre de taille. Contrairement à ce qui apparaît dans le `port map`, n ne correspond pas à un *fil* du circuit à construire. Il ne s'agit pas de connecter des ports. C'est juste une indication pour spécifier quelle taille on choisit pour l'instanciation du composant `reggene`.

Compteur n bits générique

Il s'agit maintenant de définir un compteur lui-même de taille générique, fabriqué avec le composant générique registre n bits.

Question 5
— Complétez le fichier `vhd/compteurgene.vhd`.
— Réalisez la synthèse (`make synthese TOP=compteurgene`) et observez `compteurgene.ngr` dans ISE pour constater l'absence de *fil* correspondant au paramètre n.
— On fournit de quoi tester ce compteur générique dans `vhd/tb_compteurgene.vhd`. Simulez avec `make run_simu TOP=compteurgene`.

Ex. 5 : Pour aller plus loin...

Nous donnons ici quelques idées pour prolonger les exercices de ce TP.

Exploitation des sorties `qb` des bascules dans le codage compact des automates

Il est probable que dans le premier exercice vous ayez systématiquement connecté les sorties `qb` des bascules D à "open". Reprenez vos équations pour exploiter les sorties `qb`, au lieu d'écrire des sous-expressions de la forme `not q` dans les équations.

Codage *1 parmi n* des automates avec une bascule initialisée à 0

Nous avons utilisé deux bascules différentes pour le codage *1 parmi n*. Si maintenant on ne vous fournit que la bascule initialisée à 0, comment faire ? Le codage de l'état initial n'est pas 0 partout, et pourtant il faut que la bascule correspondant à cet état soit à 1 dans l'état initial. On décide donc que la bascule qui code l'état initial contiendra en fait en permanence la négation de q_0.

Reprendre le codage 1 parmi n pour le coder avec cette idée.

Comparaison des codages d'automates

On a construit deux (et même trois si vous avez fait l'extension ci-dessus) codages du même automate. On peut les faire marcher en parallèle sur les mêmes entrées, pour vérifier qu'ils produisent toujours la même sortie.

Vous pouvez ajouter ce qu'il faut pour mettre en oeuvre cette idée ddans un testbench inspiré de `vhd/tb_auto.vhd`.