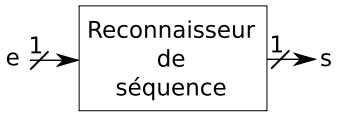
TP3 : Implantations d'un automate reconnaisseur de séquence

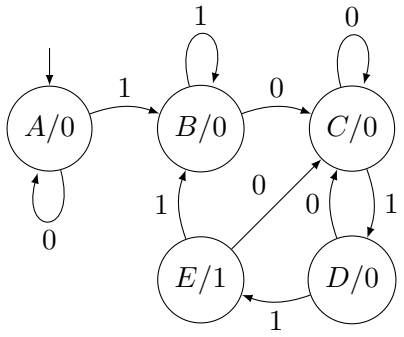
Travail Préparatoire : Lire l'énoncé avant l'arrivée en séance de TP.
Objectifs du TP :

- Réviser les notions de VHDL déjà introduites sur des machines d'états
- Découvrir le codage implicite des machines d'états
- Approfondir les descriptions par processus

Ex. 1 : Implantation en 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 Complétez le fichier `auto.vhd` afin d'y décrire l'automate dessiné. Vous utiliserez le composant `reggene` vu dans le dernier TP.

Question 3 Simulez le bon fonctionnement grâce au testbench `tb_auto.vhd` (`$ make run_simu TOP=auto`), puis réalisez la synthèse (`make synthese TOP=auto`) et analysez le résumé des rapports de synthèse affiché dans la console.

Attention : le message `** Failure:Simulation terminée` (et, dans Vivado, l'affichage du testbench à la ligne `assert (i < 32) report "Simulation terminée" severity failure;`) n'est pas une erreur en soit, mais indique simplement que la simulation s'est correctement terminée.

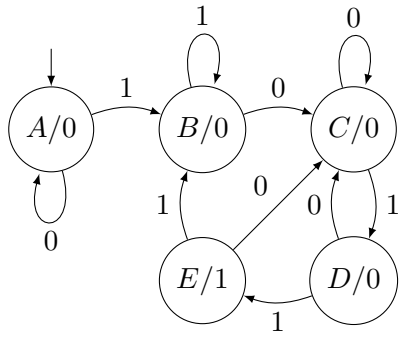
En effet, le testbench `tb_auto.vhd` fourni génère une suite de stimuli qu'il applique en entrée de votre automate. Cette suite est construite de manière à tester toutes les transitions de l'automate. Pour chaque entrée, le testbench regarde la sortie de l'automate et la compare à une valeur de référence. 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 ====
```

Pour terminer la simulation et éviter quelle ne dure infiniment, le testbench génère l'erreur `** Failure:Simulation terminée`.

Ex. 2 : Implantation en 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, on décrit la même entité, mais avec une architecture différente. 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) :

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

Question 1 Complétez le fichier `vhd/auto_unparmin.vhd` afin d'y coder l'automate selon les équations ci-dessus en contraintes imposées (signaux, utilisation d'un process).

Question 2 Simulez le bon fonctionnement (`$ make run_simu TOP=auto_unparmin TOP_ENTITY=auto`), puis réalisez la synthèse (`make synthese TOP=auto_unparmin TOP_ENTITY=auto`) et analysez le résumé des rapports de synthèse affiché dans la console. (on remarque ici qu'il est nécessaire de préciser l'entité car le nom des fichiers n'est pas le même que le nom de l'entité décrit)

Ex. 3 : Implantation en codage implicite

Dans les 2 versions précédentes, nous avons du fixer le codage (et donc le circuit) à l'implantation. En pratique, on va plutôt laisser la synthèse logique faire le choix de codage à notre place. Pour ça, il faut définir nouveau type, énumérant les différents états possibles de notre machine d'états. C'est plus lisible, plus simple à implanter et plus facile à debugger.

Question 1 Complétez le fichier `vhd/auto_implicite.vhd` afin d'y coder l'automate en utilisant les nouveaux éléments de syntaxe introduits.

Question 2 Simulez le bon fonctionnement (`$ make run_simu TOP=auto_implicite TOP_ENTITY=auto`). Observez en particulier votre signal d'état. Qu'a t'on gagné par rapport aux implantations précédentes ? Réaliser la synthèse (`make synthese TOP=auto_implicite TOP_ENTITY=auto`) Quelle est l'implémentation que le compilateur a mis en place ?

Pour aller plus loin...

Ex. 4 : Décodeur NRZI

Dans cet exercice, on ne vous fourni aucun fichier de départ, vous avez tout à écrire, c'est un bon exercice qui récapitule toutes les notions du langage vhd vu jusqu'à maintenant.

On se propose ici de décrire et tester le circuit Décodeur " Non-Retour à Zéro Inversé " étudié dans l'exercice 2 du TD 5.

Question 1 En vous inspirant des fichiers `vhd/auto.vhd`, `vhd/tb_auto.vhd` et `tb_auto.prj` et en reprenant les éléments du TD 5, écrire les fichiers v̈hd/decod_nrzi.vhd(en utilisant la méthode de votre choix : codage logarithmique ou codage un parmi n ou codage implicite), v̈hd/tb_decod_nrzi.vhd (en décrivant les valeurs en entrée et les valeurs attendues en sortie, conformément au chronogramme du TD) et `tb_decod_nrzi.prj`

Question 2 Simuler et vérifier la justesse de votre réalisation. Ajouter tous les signaux internes pertinent, et présenter cette simulation de manière claire (en ajoutant des séparateurs, des couleurs, ...)

Question 3 Faire la synthese de votre réalisation et justifier les composantes nécessaires indiqués par le rapport de synthèse