# Lab 3 : VHDL

## Machine à états

## • Introduction

L'objectif de ce lab est d'implémenter des machines à états.

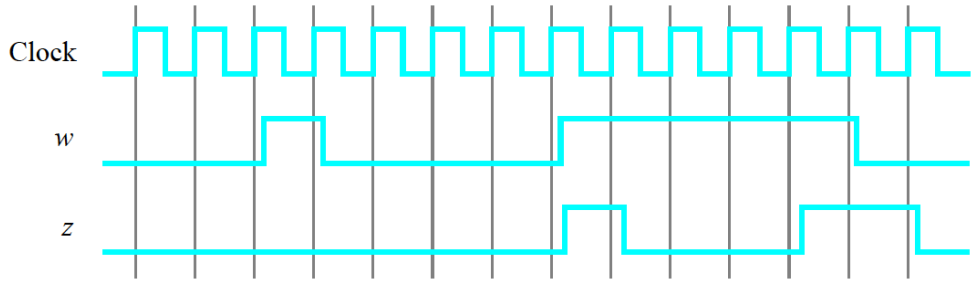
Normalement, vous ne devez créer qu'un seul projet QUARTUS dans un répertoire de votre espace, et **celui-ci a déjà été créé lors de la première séance**. On rajoutera des fichiers VHDL au fur et à mesure des séances directement ce projet.

**Rappel sur le lancement de quartus :**
-Ouvrir le terminal, puis taper cd **/home/embed/intelFPGA_lite/18.1/quartus/bin** puis taper **./quartus** (ne pas oublier le « ./ »)

-Ouvrer ensuite votre projet Quartus placé dans votre répertoire local.

## • Exercice 1 : Détecteur de séquence

On désire dans cet exercice réaliser une machine d'états finis (ou finite state machine, FSM) qui reconnaisse deux séquences spécifiques de symboles d'entrée. Ces deux séquences correspondent à quatre 1 ou quatre 0 consécutifs.
Le composant à réaliser possédera une entrée w et une sortie z. Lorsque w = 1 ou que w = 0 pendant quatre cycles d'horloge consécutifs la valeur de z est placée à 1; autrement z = 0 . Des chevauchements de séquences sont autorisées, ainsi si w = 1 pendant 5 cycles consécutifs, la sortie z reste à 1. Le chronogramme suivant illustre la relation entre w et z :




Le diagramme d'états de cette FSM est donné en Figure 2. Dans cet exercice, vous réaliserez manuellement le circuit qui implémente ce diagramme. Il inclut les expressions logiques des fonctions de transition pour chaque flip-flop du registre d'états. On utilisera ici un codage One Hot, c'est-à-dire sur neuf bits (flip-flops) appelés y8 à y0, décrit dans la Table 1.

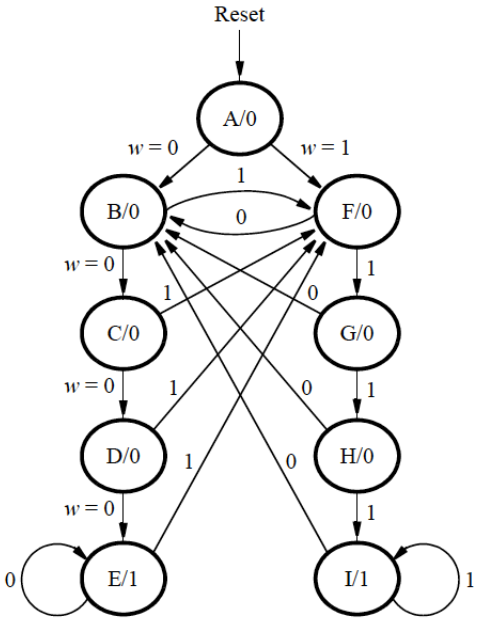



*Diagramme d'états de la FSM*

| Name | State Code $y_8y_7y_6y_5y_4y_3y_2y_1y_0$ |
|---|---|
| **A** | 000000001 |
| **B** | 000000010 |
| **C** | 000000100 |
| **D** | 000001000 |
| **E** | 000010000 |
| **F** | 000100000 |
| **G** | 001000000 |
| **H** | 010000000 |
| **I** | 100000000 |

*Codage one-Hot des états de la FSM*

A faire sur papier :

a. Etablir la table de vérité à partir du diagramme d'état
b. Déterminer les fonctions de transition et la fonction de génération

A faire sous Quartus :

c. Sous Quartus, File -> New, VHDL File
d. Sauvegarder le nouveau fichier immédiatement via **File, Save As**, et nommer la **FlipFlop_D**
e. Ecrire le code VHDL correspondant à une bascule D. Celle-ci aura 4 entrées (D, setn, resetn (asynchrone) et CLK) et 1 sortie (Q).
f. Créer un nouveau fichier VHDL appelé **FSM_detect**.
g. Créer un fichier VHDL appelé **test_FSM_detect** qui sera le testbench du module FSM. Effectuer la simulation en générant les stimuli appropriés et faites valider par l'enseignant.
h. Créer ensuite un fichier **FSM_impl_test** dont on connectera KEY0 à l'entrée resetn, le switch SW0 comme entrée w, le bouton KEY1 pour l'horloge CLK. On utilisera aussi LEDG0 pour la sortie z ainsi que les leds LEDR8 à LEDR0 pour afficher l'état courant de la FSM.
i. Simuler le comportement du circuit
j. Compiler l'entité **FSM_impl_test** et tester sur la carte
k. Faites valider par l'enseignant

Finalement, on souhaite modifier le codage des états de la FSM comme indiquée dans la table suivante :

| Name | State Code $y_8y_7y_6y_5y_4y_3y_2y_1y_0$ |
|---|---|
| **A** | 000000000 |
| **B** | 000000011 |
| **C** | 000000101 |
| **D** | 000001001 |
| **E** | 000010001 |
| **F** | 000100001 |
| **G** | 001000001 |
| **H** | 010000001 |
| **I** | 100000001 |

*Codage one-hot modifié*

En effet, pour l'implémentation sur FPGA, le circuit peut être simplifié. Pour cela, le codage des états revêt un caractère très important. On doit déjà prêter attention à l'état de reset. Les flipflops stockant état doivent être à 0 lorsque l'état correspond au reset. Ce n'est pas le cas avec le codage one-hot (1 bascule est à 1). Avec le nouveau codage, toutes les bascules seront à 0 lors du reset. Cette approche est préférable car les FF du FPGA incluent une entrée reset mais pas d'entrée set. En complément de ce codage, vous modifierez donc votre entité flipflop en supprimant l'entrée set. Les modifications dans le top doivent être minimales.

## • Exercice 2 : FSM à 3 processus

Dans cet exercice nous allons décrire la machine d'états finis précédente à plus haut niveau. Dans cette version, il ne s'agira plus d'écrire les expressions logiques de chaque flipflop. Pour simplifier les choses, nous allons décrire la FSM à l'aide de 3 processus :

-1 processus pour le codage de l'état courant
-1 processus pour le passage à l'état futur
-1 processus pour la sortie

Nous allons utiliser une structure VHDL de type CASE à l'intérieur d'un PROCESS et un autre PROCESS pour représenter les flip-flops. Nous utiliserons un troisième PROCESS pour la fonction de génération de la sortie z. Pour cela, l'état sera codé sur 4 bits comme indiqué dans la table située à droite :

| Name | State Code $y_3y_2y_1y_0$ |
|---|---|
| **A** | 0000 |
| **B** | 0001 |
| **C** | 0010 |
| **D** | 0011 |
| **E** | 0100 |
| **F** | 0101 |
| **G** | 0110 |
| **H** | 0111 |
| **I** | 1000 |

Nous vous fournissons un squelette de code VHDL pour la structure CASE ci-dessous :

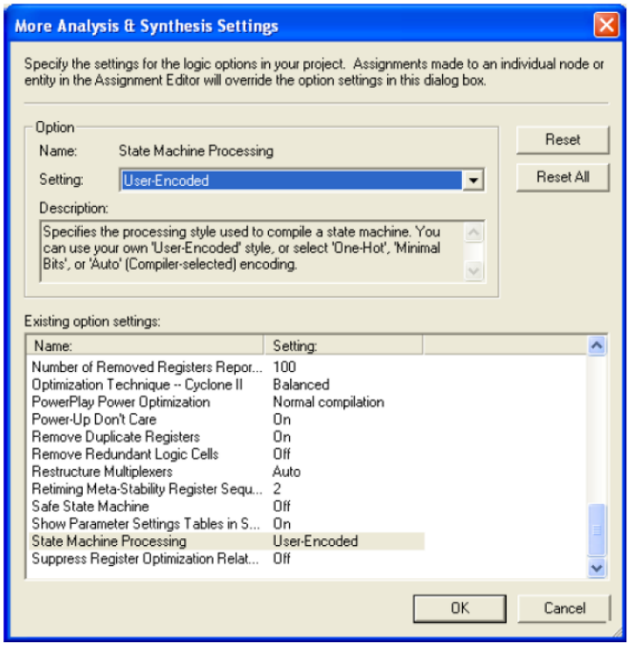
```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
ENTITY FSM IS
PORT (
            --define input and output ports
        );
END FSM;

ARCHITECTURE Behavior OF FSM IS
--declare signals
TYPE State_type IS (A, B, C, D, E, F, G, H, I);
-- Attribute to declare a specific encoding for the states
attribute syn_encoding : string;
attribute syn_encoding of State_type : type is "0000 0001 0010 0011 0100 0101 0110 0111 1000";
SIGNAL y_Q, Y_D : State_type; --y_Q is present state, y_D is next state
BEGIN

PROCESS (w, y_Q) -- state table
BEGIN
  case y_Q IS
     WHEN A =>  IF (w = '0') THEN Y_D <= B;
                ELSE Y_D <= F;
                END IF;
     … other states
     END CASE;
END PROCESS; -- state table

PROCESS (Clock) -- state flip-flops
BEGIN
…
 END PROCESS;
…  assignments for output z and the LEDs
END Behavior;
```

On observe que l'état présent et l'état futur sont représentés comme des types énumérés utilisant les symboles A à I. C'est ensuite le synthétiseur VHDL qui détermine le nombre de flip-flops nécessaires.

A faire :

a. Sous Quartus, File, New, VHDL File
b. Sauvegarder la immédiatement via **File, Save As**, et nommer la **FSM**
c. Ecrire le code VHDL en se basant sur le template donné précédemment
d. Effectuer une simulation fonctionnelle pour vérifier le fonctionnement
e. Créer un nouveau fichier VHDL que l'on nommera **Test_FSM**. Dans ce module, créer une instance du module **FSM**. Utiliser KEY0 pour l'entrée de reset asynchrone, le SW0 comme entrée w et le bouton KEY1 comme horloge manuelle. Utiliser la led LEDG0 pour visualiser la sortie z, et les LEDR3 à LEDR0 pour afficher l'état.
f. Avant la compilation, il faut préciser à l'outil de synthèse les codes choisis pour chaque état. Autrement, il fera sa propre affectation de codes. Indiquez ces choix dans **Assignments > Settings** dans Quartus II, puis cliquer sur **Analysis and Synthesis**, puis sur **More Settings**. Comme indiqué dans la Figure suivante,

modifiez le paramètre State Machine Processing sur User-Encoded .




Visualiser ensuite le circuit produit par Quartus via l'onglet **Tools < Netlist Viewer, RTL Netlist Viewer.** Double-cliquer sur la boîte représentant la FSM, et vérifier qu'elle correspond bien à celle attendue. Pour voir les codes des états ouvrir le rapport de Compilation (section **Analysis and Synthesis**, puis State Machines).

g. Simuler le comportement du circuit
h. Compiler le module **Test_FSM** et le tester sur la plateforme
i. Faites valider par l'enseignant.
j. Modifier maintenant le codage manuel avec un codage One Hot (modifier les paramètres indiqués dans l'étape f précédente). Relancer la synthèse. Comparer les résultats de synthèse à ceux obtenus en étape f. Comparer les codages d'états obtenus.

## Exercice 3 : Unité de contrôle du microprocesseur

Par rapport des spécifications données en cours, on souhaite réaliser l'unité de contrôle du processeur. Réaliser les étapes suivantes :

a. Sur feuille papier, dessiner le diagramme d'état du contrôleur. Faites attention à tenir compte de tous les signaux d'entrée/sortie (Run, …)
b. Créer ensuite un fichier VHDL, nommer la **ControlUnit** et sauvegarder là. Lors de votre description, pensez bien à utiliser 3 processus, cela rendra votre code très clair.
c. Faire une simulation après avoir créé le **testbench tb_ControlUnit.vhd** pour

valider le fonctionnement

d. Faire valider par l'enseignant

## Exercice 4 : Pour aller plus loin, Encodeur de code Morse

Dans cet exercice, on souhaite réaliser un encodeur de code Morse. Ce code Morse utilise des impulsions courtes ou longues pour représenter un message. Chaque lettre correspond à une séquence de points (pulsation courte) ou de traits (longue). Un exemple est indiqué ci-dessous pour les 8 premières lettres de l'alphabet :

| | |
|---|---|
| A | • — |
| B | — • • • |
| C | — • — • |
| D | — • • |
| E | • |
| F | • • — • |
| G | — — • |
| H | • • • • |

L'entrée de notre circuit prendra en entrée un codage binaire d'une de ces 8 lettres sur les switches SW2 à SW0 (000 for A, 001 for B, etc.) et affichera le code Morse correspondant sur les leds rouges. Lorsque l'utilisateur appuie sur le bouton KEY1, le circuit affiche le code. Les temps des impulsions courtes seront de 0.5-seconde (points) et les longues de 1.5-seconde (traits). Le bouton KEY0 fonctionne comme reset asynchrone.
Un code VHDL partiel de l'encodeur vous est fourni.

A faire :

a. Sous Quartus, observer le code. Quelle est la fréquence de l'horloge à l'entrée du Top ? Quelles sont les valeurs des paramètres passées au composant modulo_counter_ser ? Que fait donc le composant modulo_counter_ser ?
b. Le process state_table implémente la machine de plus haut niveau qui sélectionne une des 8 machines à état à déclencher. Ecrivez les codes des 8 machines à états correspondant à chacune des 8 premières lettres. Chaque PROCESS écrit dans un signal qui lui est propre. La LEDR affichant le code Morse fera une opération de ou logique entre ces 8 signaux.
c. Tester le circuit directement sur la carte

## Conclusion

Tirer les conclusions vis-à-vis des différents types de codage et la manière de décrire une FSM.