Lab 7 : Machines d'états finies

Université de Cergy-Pontoise

Partie I

On désire dans cet exercice réalier une machine d'états finis (ou finite state machine, FSM) qui reconnaisse deux séquences spécifiques de symbôles d'entrée. Ces deux séquences correspondent à quatre 1 ou quatre 0 consécutifs. Le composant à réaliser a 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 chevauchement de séquences sont autorisées, ainsi si $w = 1$ pendant 5 cycles consecutifs, la sortie z doit être égale à 1 après le 4e et 5e cycle. La figure 1 illustre la relation entre w et z.

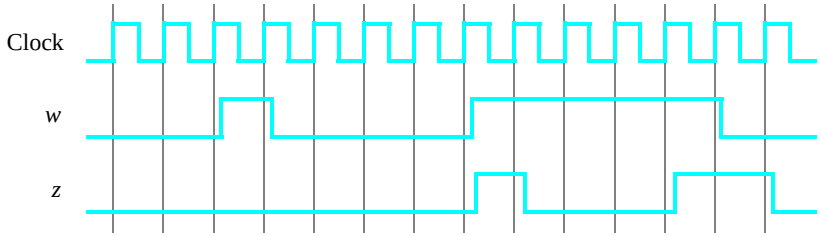

Figure 1: Les timing requis de la sortie z.

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 $y_8, \ldots, y_0$, décrit dans la Table 1.

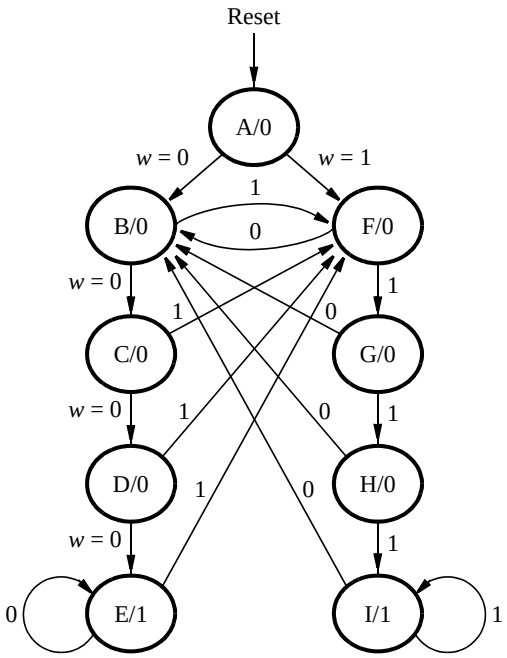

Figure 2: Diagramme d'états de la FSM.

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

Table 1: Codage One-hot de la FSM.

Réaliser le code VHDL de cette FSM :

1. Créer un nouveau projet Quartus-II,

2. Ecrire le code d'un composant flipflop (1bit) à 4 entrées (D, setn, resetn, clock) et une sortie (Q). Ecrire ensuite le code VHDL qui fait l'instanciation des neufs flip-flops dans le circuit, ainsi que les expressions des fonctions de transition (état futur) et de génération (z). Pour cela écrivez la table de vérité correspondant au diagramme de la figure 1. C'est à partir de cette table que vous retrouverez les expressions logiques de chaque bit du registre d'états.

   Utiliser le switch SW_0 de la carte DE2 comme un reset asycnhrone actif à l'état bas. Utiliser le switch SW_1 comme entrée w, et le bouton poussoir KEY_0 comme clock manuelle. Utiliser enfin la sortie $LEDG_0$ comme sortie z, ainsi que les leds $LEDR_8$ à $LEDR_0$ pour afficher l'état courant de la FSM.

3. Simulez le comportement de votre circuit.

4. Puis testez le fonctionnement sur la carte.

5. Finallement, considérez la modification de codage indiquée dans la Table 2. 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 préter attention à l'état de reset. Les flipflops de cet état doivent être à 0 lorsque l'état correspond au 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.

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

Table 2: Codage one-hot modifié.

Partie II

Dans cet exercice nous allons décrire la machine d'états finis de la Figure 2 à plus haut niveau. Dans cette version, il ne s'agira plus d'écrire les expressions logiques de chaque flipflop. 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 3.

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

Table 3: Codage binaire de la FSM.

Nous vous fournissons un squelette de code VHDL pour la structure CASE en Figure 3. On observe que l'état présent et l'état futur sont représentés comme des types énumérés utilisant les symbôles A à I. C'est ensuite le synthétiseur VHDL qui détermine le nombre de flip-flops nécessaire.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY part2 IS
   PORT (  . . . define input and output ports
           . . .);
END part2;

ARCHITECTURE Behavior OF part2 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;
```

Figure 3: Squellette VHDL pour la description de la FSM.

Implémenter le circuit en suivant les étapes suivantes :

1. Créer un nouveau projet.
2. Inclure et adapter le code de la Figure 3. Utiliser les switch SW_0 sur les cartes DE2 comme reset asynchronous à état bas, le SW_1 comme entrée w et le bouton KEY_0 comme horloge manuelle. Utiliser la led $LEDG_0$ comme sortie z, et les $LEDR_3$ à $LEDR_0$ pour afficher l'état.
3. 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 cliquez sur Analysis and Synthesis, puis sur More Setting. Comme indiqué dans la Figure 4, modifiez le paramètre State Machine Processing sur User-Encoded.
4. Pour visualiser le circuit produit par Quartus II, lancez l'outil RTL viewer. Double-cliquez sur la boîte representant la FSM, et vérifier qu'elle correspond bien à celle de la Figure 2. Pour voir les codes des états ouvrir le rapport de Compilation (section Analysis and Synthesis, puis State Machines).

5. Simuler le comportement du circuit.

6. Vérifier enfin le fonctionnement sur la carte.

7. Modifier maintenant le codage manuel avec un codage One Hot (modifier les paramètres indiqués dans l'étape 3). Relancer la synthèse. Comparer les résultats de synthèse à ceux obtenus en étape 3. Comparer les codages d'états obtenus.

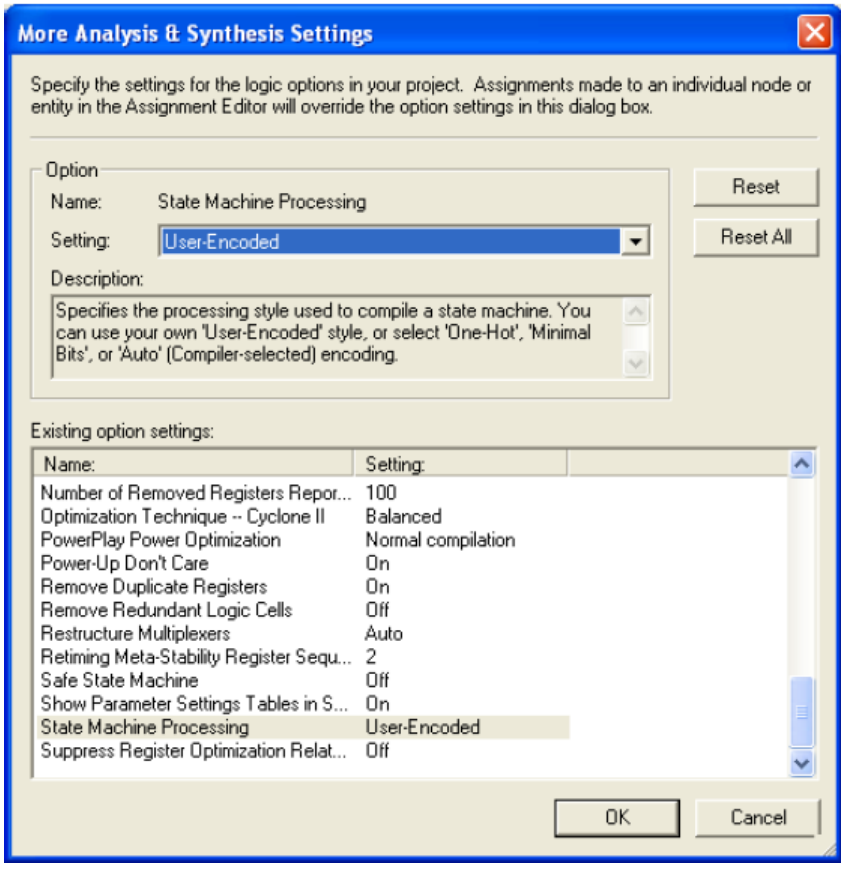

Figure 4: Spécification de la méthode de codage des états dans Quartus II.

Partie III

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 switchs SW_{2-0} (000 for A, 001 for B, etc.) et affichera le code Morse correspondant sur les leds rouges. Lorsque l'utilisateur appuie sur le bouton KEY_1, 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 boutton KEY_0 fonctionne comme reset asycnhrone.

Un code VHDL partiel de l'encodeur vous est fourni.

1. Observez le code. Quelle est la fréquence de l'horloge a l'entrée du Top ?
   Quels sont les valeurs des paramètres passées au composant *modulo_counter_ser* ?
   Que fait donc le composant *modulo_counter_ser* ?

2. 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.

3. Tester le circuit directement sur la carte.

4. (optionnel) Essayer de rassembler ces huits FSM en une seule FSM homogène en utilisant les longeurs fournies en paramètre générique.