INF3500 : Conception et réalisation de systèmes numériques

Examen intra
14 mars 2008

Durée : 2h.
Pondération : 20 %.
Documentation : Toute permise.
Calculatrice : Programmable permise.
Directives particulières : Ordinateurs interdits (une calculatrice exécutant du VHDL est un ordinateur), répondre à toutes les questions, la valeur de chaque question est indiquée.

Tous les codes VHDL font : `library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;`

Question 1. [9 points]

Nous voulons faire un comparateur séquentiel série, qui compare X et Y (nombres binaires non signés), entrés de manière sérielle à partir des bits de poids faible, et qui nous dit si X<Y, X=Y ou X>Y. Les entrées du circuit sont l'horloge, le signal d'initialisation, X_i et Y_i , et les sorties sont X<Y et X=Y (les deux sorties étant fausses si X>Y).

a. [2 pts] Donner le VHDL synthétisable d'un circuit combinatoire qui compare un seul bit de X et Y et dit si X_i=Y_i ou X_i<Y_i (respectivement `egali` et `petiti`). N'utilisez pas de `process` pour décrire ce circuit simple.

```
entity compareBit is
   port (
      Xi, Yi : in std_logic;
      egali, petiti : out std_logic
   );
end compareBit;

architecture arch of compareBit is
...
```

b. [2 pts] Donner le diagramme d'états pour un circuit séquentiel qui, à tout moment, nous donne la comparaison des nombres jusqu'à présent. Autrement dit `egal` sera vrai si $X_{i..0}$=$Y_{i..0}$ et `petit` sera vrai si $X_{i..0}$<$Y_{i..0}$. Les entrées sont `egali` et `petiti`, venant du module `compareBit`, et les sorties sont `egal` et `petit`. (Vous pouvez faire le VHDL en c. puis en extraire le diagramme d'états si vous croyez que c'est plus simple dans ce sens.)

c. [4 pts] Compléter le VHDL synthétisable suivant, pour le circuit séquentiel précédent.

```
entity compareSerie is
   port (
      Xi, Yi, clk, init : in std_logic;
      egal, petit : out std_logic
   );
end compareSerie;

architecture arch of compareSerie is
   component compareBit
      port( Xi, Yi : in std_logic; egali, petiti : out std_logic );
   end component;
...
```

d. [1 pt] Donner un avantage et un inconvénient de faire la comparaison sérielle à partir des bits de poids faible, par rapport à la faire à partir des bits de poids fort.

Question 2. [9 points]

Nous voulons un circuit combinatoire qui prend la sortie d'un compteur de Johnson (registre à décalage circulaire ayant un inverseur sur la boucle) et nous donne un compte binaire standard. La sortie sera 0 lorsque le compteur a des zéros partout, 1 lorsque le compteur a un 1 dans son bit de poids faible, 2 lorsque le compteur a deux 1 dans ses bits de poids faibles, etc., jusqu'à *nétats*-1 lorsque le compteur a un 1 dans son bit de poids fort (où *nétats* est le nombre d'états du compteur de Johnson).

a. [5 pts] Compléter le code VHDL générique ci-dessous, pour qu'il se synthétise en un circuit combinatoire correct pour toute taille de compteur. Votre circuit peut donner une sortie quelconque si l'entrée n'est pas un compte de Johnson valide.

```
entity johnsonAbinaire is
   generic (
      n : positive := 4;  -- nombre de bits du compteur de Johnson
      nB : positive := 3  -- nombre de bits en sortie
   );
   port (
      J : in std_logic_vector(n-1 downto 0);   -- l'entrée (compte de Johnson)
      B : out std_logic_vector(nB-1 downto 0)  -- la sortie (compte binaire)
   );
end johnsonAbinaire;

architecture arch of johnsonAbinaire is
...
```

b. [4 pts] Compléter le banc d'essai pour vérifier le circuit `johnsonAbinaire` de façon exhaustive (pour les entrées valides) pour un `n` donné. Le banc d'essai doit indiquer, à la console seulement, pour quels cas le circuit à vérifier donne une sortie incorrecte. Définir `nB` comme le plus petit nombre de bits nécessaire pour représenter la sortie.

```
use ieee.math_real.all;

entity johnsonAbinaireTB is
   generic ( n : positive := 4 );
end johnsonAbinaireTB;

architecture arch of johnsonAbinaireTB is
   component johnsonAbinaire
      generic( n : positive; nB : positive );
      port( J : in std_logic_vector(n-1 downto 0); B : out std_logic_vector(nB-1 downto 0) );
   end component;

   constant nB : positive := ...; -- nombre de bits de sortie
   signal J : std_logic_vector(n-1 downto 0);
   signal B : std_logic_vector(nB-1 downto 0);
begin
   UUT : johnsonAbinaire  generic map (n => n, nB => nB)  port map (J, B);
...
```

Question 3. [2 points]

Nous voulons un circuit séquentiel qui décode du morse. Un premier circuit analogique à numérique détecte la tonalité émise et donne en sortie un 1 lorsqu'elle est détectée et 0 lorsqu'elle ne l'est pas. Le circuit numérique va ensuite échantillonner la sortie du détecteur, pour compter le temps passé à 1, pour savoir si c'est un bip court (point) ou long (trait), et le temps passé à 0 pour savoir si c'est l'espace entre deux bips d'une même lettre, ou si c'est la fin de la lettre.

Expliquer à quoi il faut faire attention lors de l'échantillonnage, pourquoi, et comment le faire correctement.