

INF3500 : Conception et réalisation de systèmes numériques

Examen final

Mercredi 18 août 2010

Durée: 2h30.

Pondération: 40%.

Documentation: Une feuille recto verso 8.5"×11" ou A4 permise.

Calculatrice: Programmable permise.

Directives particulières:

- Ordinateurs interdits.
 - Un dictionnaire autorisé.
 - Répondre à toutes les questions, la valeur de chaque question est indiquée.
 - Répondre en français ou en anglais.
 - Répondre dans le cahier fourni et soumettre la page du questionnaire indiquée.
 - Ne pas remettre le questionnaire.
 - Ne posez pas de questions. En cas de doute sur le sens d'une question, énoncez clairement toute supposition que vous faites.
-

Question 1. (8 points)

L'algorithme de Box-Muller permet de générer des variables aléatoires suivant la distribution normale. Pour ce faire, il se fonde sur le principe suivant : soient u_0 et u_1 deux variables indépendantes suivant une distribution uniforme sur l'intervalle $(0,1)$, alors $x_0 = \sin(u_1) \sqrt{\log(u_0)}$ et $x_1 = \cos(u_1) \sqrt{\log(u_0)}$ sont deux variables indépendantes suivant la distribution normale.

Dans un article datant de 2006¹, Lee et al. proposent le circuit de la Figure 1 pour implémenter l'algorithme de Box-Muller sur FPGA. Les lignes horizontales représentent des niveaux de registres.

a) (3 pts) En supposant que le circuit fonctionne à une fréquence de 100 MHz, évaluez les métriques suivantes en justifiant clairement vos réponses :

- La latence du circuit en cycles d'horloge.
- La latence du circuit en secondes.
- Le débit en nombre de résultats par cycle d'horloge.
- Le débit en nombre de résultats par seconde.

On vous demande d'évaluer l'opportunité de remplacer la boîte SIN/COS (et les niveaux de registres qui lui sont associés) par un processeur CORDIC produisant simultanément les deux sorties sinus et cosinus demandées.

b) (3 pts) Sachant que le processeur CORDIC a une latence (en cycles d'horloge) égale à la largeur en bits de l'entrée, et en supposant que la fréquence du circuit passe de 100 MHz à 250 MHz grâce à cette solution, évaluez les métriques suivantes en justifiant clairement vos réponses :

- La latence du circuit en cycles d'horloge.
- La latence du circuit en secondes.
- Le débit en nombre de résultats par cycle d'horloge.
- Le débit en nombre de résultats par seconde.

c) (2 pts) À la lumière de ces résultats, discutez l'intérêt de cette nouvelle solution.

¹ Lee, D.-U et al. *A Hardware Gaussian Noise Generator Using the Box-Muller Method and Its Error Analysis*. IEEE Transactions on Computers, 55(6), June 2006.

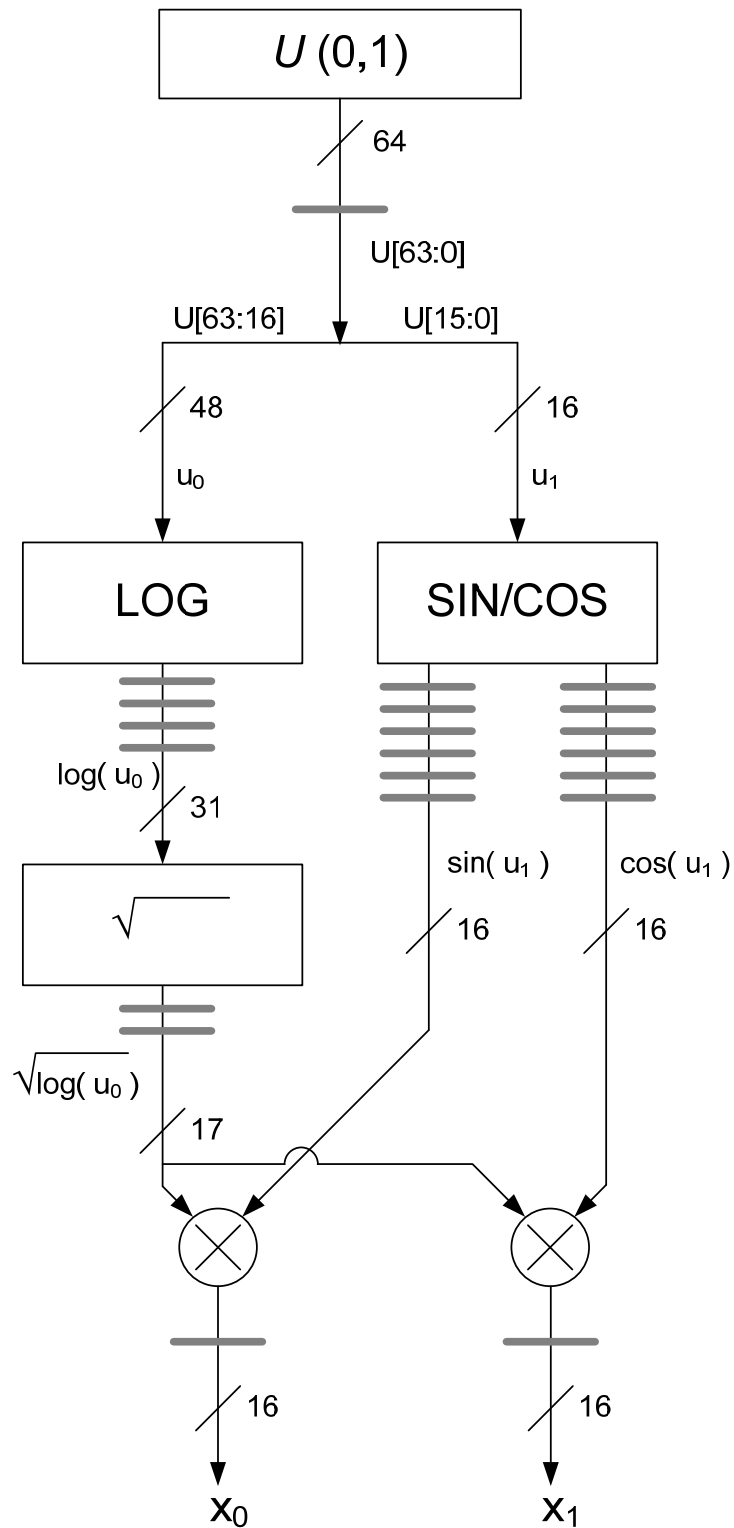


Figure 1 Générateur de variables normales de Lee et al.

Question 2. (11 points)

Un collègue vous suggère d'utiliser le code suivant pour réaliser un circuit effectuant un tri sur quatre nombres en entrées : a, b, c et d. Ainsi, si $(a, b, c, d) = (6, 4, 2, 8)$, les sorties devraient être affectées après 5 cycles d'horloge, de sorte que $(x, y, z, w) = (2, 4, 6, 8)$.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity trieuse is
  generic(
    width : positive := 18
  );
  port(
    clk   : in std_logic;
    a, b, c, d : in unsigned(width-1 downto 0);
    x, y, z, w : out unsigned(width-1 downto 0)
  );
end trieuse;

architecture arch of trieuse is

  type aunsigned is array (0 to 3) of unsigned(width-1 downto 0);
  type a_unsigned is array(0 to 4) of aunsigned;
  signal aus : a_unsigned;

begin

  process( clk ) is
  begin
    if ( rising_edge( clk ) ) then
      for i in 0 to 1 loop
        for j in 0 to 1 loop
          if( aus( 2*i )( 2*j ) > aus( 2*i )( 2*j+1 ) ) then
            aus( 2*i+1 )( 2*j )   <= aus( 2*i )( 2*j+1 );
            aus( 2*i+1 )( 2*j+1 ) <= aus( 2*i )( 2*j );
          else
            aus( 2*i+1 )( 2*j )   <= aus( 2*i )( 2*j );
            aus( 2*i+1 )( 2*j+1 ) <= aus( 2*i )( 2*j+1 );
          end if;
        end loop;

        aus( 2*i+2 )( 0 )   <= aus( 2*i+1 )( 0 );
        aus( 2*i+2 )( 3 )   <= aus( 2*i+1 )( 3 );

        if( aus( 2*i+1 )( 1 ) > aus( 2*i+1 )( 2 ) ) then
          aus( 2*i+2 )( 1 )   <= aus( 2*i+1 )( 2 );
          aus( 2*i+2 )( 2 )   <= aus( 2*i+1 )( 1 );
        else
          aus( 2*i+2 )( 1 )   <= aus( 2*i+1 )( 1 );
          aus( 2*i+2 )( 2 )   <= aus( 2*i+1 )( 2 );
        end if;
      end loop;

      aus( 0 ) <= (a, b, c, d);
    end if;
  end process;

  (x, y, z, w) <= aus( 4 );

end arch;

```

Vous n'êtes pas certain du bon fonctionnement du circuit `trieuse` qui vous a été fourni et vous désirez élaborer une stratégie de test.

- a) **(1 pt)** Quelle est le nombre de vecteurs de test nécessaires pour effectuer un test exhaustif de l'entité `trieuse` ? Justifiez votre réponse par un calcul.
- b) **(1 pt)** Quel est le nombre total de combinaisons de choix de branchements qu'il faut considérer pour vérifier entièrement la couverture de branchements de l'entité `trieuse` ? Justifiez votre réponse par un calcul.
- c) **(1 pt)** Proposez un partitionnement en classes des entrées du circuit `trieuse` permettant de minimiser la taille des vecteurs de tests.
- d) **(1 pt)** Pour chacune des classes identifiées, proposez un exemplaire de donnée pouvant représenter adéquatement les autres membres de la classe.
- e) **(1 pt)** Donnez le nombre minimal de vecteurs de test nécessaires pour effectuer un test faible sur l'entité `trieuse`, et ce en recourant aux classes que vous avez identifiées. Justifiez votre réponse.
- f) **(1 pt)** Donnez le nombre minimal de vecteurs de test nécessaires pour effectuer un test fort sur l'entité `trieuse`, et ce en recourant aux classes que vous avez identifiées. Justifiez votre réponse.
- g) **(2 pts)** Quelle est en pourcentage la couverture de branchements obtenue en appliquant un test fort ? Justifiez votre réponse.

Votre collègue vous fournit le banc d'essai suivant qu'il affirme avoir utilisé pour s'assurer du bon fonctionnement de son circuit `trieuse`.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity trieuse_tb is
    generic(width : positive := 4);
end trieuse_tb;

architecture tb of trieuse_tb is

    component trieuse is
        generic(
            width : positive := 18
        );
        port(
            clk : in std_logic;
            a, b, c, d : in unsigned(width-1 downto 0);
            x, y, z, w : out unsigned(width-1 downto 0)
        );
    end component;

    signal clk : std_logic := '1';
    signal rst : std_logic := '1';

    type aunsigned is array (0 to 3) of unsigned(width-1 downto 0);
```

```

type tb_a_unsigned is array(0 to 7) of unsigned;
constant aus : tb_a_unsigned :=
  ((x"2", x"4", x"6", x"8"), (x"2", x"4", x"8", x"6"), (x"2", x"6", x"4", x"8"),
  (x"2", x"6", x"8", x"4"), (x"2", x"8", x"4", x"6"), (x"2", x"8", x"6", x"4"),
  (x"4", x"2", x"6", x"8"), (x"4", x"2", x"8", x"6"));

signal entrees, sorties : unsigned;
signal a, b, c, d, x, y, z, w : unsigned(width-1 downto 0);

begin

  dut : trieuse generic map(width) port map(clk, a, b, c, d, x, y, z, w);

  rst <= '1' after 0 ns, '0' after 15 ns;
  clk <= not( clk ) after 5 ns;

  (a, b, c, d) <= entrees;
  sorties <= (x, y, z, w);

  process( clk ) is
    variable i : integer := 0;
  begin
    if( rst = '1' ) then
      entrees <= (x"0", x"0", x"0", x"0");
      i := 0;
    elsif( falling_edge( clk ) ) then
      if ( i >= 5 ) then
        assert ( sorties = (x"2", x"4", x"6", x"8") )
          report "Erreur sur les entrees" severity error;
        end if;

        if ( i < 8 ) then
          entrees <= aus( i );
          i := i + 1;
        elsif( i = 13 ) then
          report "Fin de simulation simulation" severity failure;
        else
          i := i + 1;
        end if;
      end if;
    end process;
  end tb;

```

- h) (1 pt) Ce banc d'essai vous assure-t-il hors de tout doute du bon fonctionnement du circuit ? Justifiez clairement votre réponse.
- i) (2 pts) Donnez en pourcentage la couverture de branchements obtenue en appliquant ce banc d'essai. Justifiez votre réponse par un calcul.

Question 3. (8 points)

Le circuit suivant permet d'additionner 4 entiers X , Y , Z et W en parallèle. Le résultat apparaît sur le bus S . Les entiers X , Y , Z , W et S sont exprimés avec n bits. On a donc $X=(x_{n-1}, x_{n-2}, \dots, x_1, x_0)$, $Y=(y_{n-1}, y_{n-2}, \dots, y_1, y_0)$, et de façon similaire pour Z , W et S . Le circuit itératif est composé de n modules identiques connectés en cascade comme montré à la Figure 2.

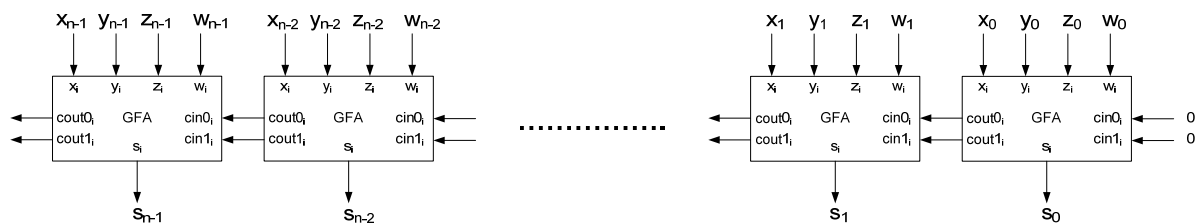


Figure 2 Additionneur itératif à 4 entrées.

Les équations logiques associées à la cellule i du circuit itératif sont données par :

$$\begin{aligned} s_i &= x_i \oplus y_i \oplus z_i \oplus w_i \oplus \text{cin}0_i \\ t_i &= x_i y_i \oplus x_i z_i \oplus x_i w_i \oplus x_i \text{cin}0_i \oplus y_i z_i \oplus y_i w_i \oplus y_i \text{cin}0_i \oplus z_i w_i \oplus z_i \text{cin}0_i \oplus w_i \text{cin}0_i \\ \text{cout}0_i &= t_i \oplus \text{cin}1_i \\ \text{cout}1_i &= x_i y_i z_i w_i \oplus x_i y_i z_i \text{cin}0_i \oplus x_i y_i w_i \text{cin}0_i \oplus x_i z_i w_i \text{cin}0_i \oplus y_i z_i w_i \text{cin}0_i \oplus t_i \text{cin}1_i \end{aligned}$$

On vous demande de compléter le code VHDL donné plus loin pour implémenter un additionneur séquentiel série à n bits utilisant le module itératif et en vous basant sur le chemin de données suivant :

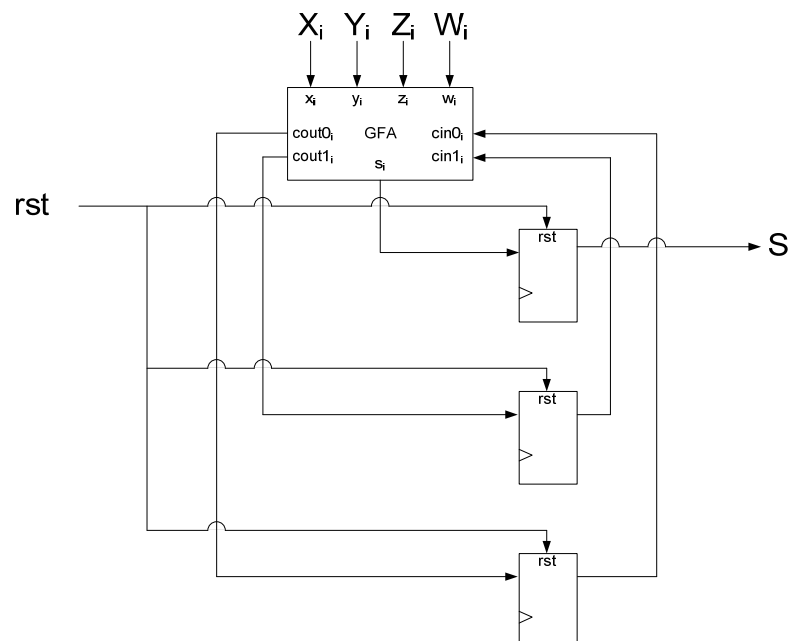


Figure 3 Chemin de données pour un additionneur sériel

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity serial_adder is
  generic(data_width : integer := 8); -- largeur des données
  port(
    clk : in std_logic;           -- signal d'horloge
    rst : in std_logic;           -- signal de réinitialisation asynchrone
    go  : in std_logic;           -- signal de début de traitement
    Xi, Yi, Zi, Wi : in std_logic; -- bits des nombres sériels à additionner
    valid : out std_logic;        -- '1' lorsque la sortie S est valide
    S : out std_logic             -- résultat sériel de l'addition
  );
end serial_adder;

architecture arch of serial_adder is

  component cell_i is port(
    xi, yi, zi, wi, cin0i, cinli : in std_logic;
    si, cout0i, coutli : out std_logic
  ); end component;

  signal si cin0i, cinli : std_logic;
  signal cout0i, coutli : std_logic;

  -- a) Déclaration de vos signaux

begin

  u_iter : cell_i port map(Xi, Yi, Zi, Wi, cin0i, cinli, si, cout0i, coutli);

  -- b) Votre implémentation

end arch;
```


Question 4. (8 points)

On considère un circuit qui réalise le contrôle d'une porte dont l'ouverture est commandée par un code d'accès. Le code d'accès est saisi au moyen d'un clavier numérique, tel qu'illustré à la Figure 4.

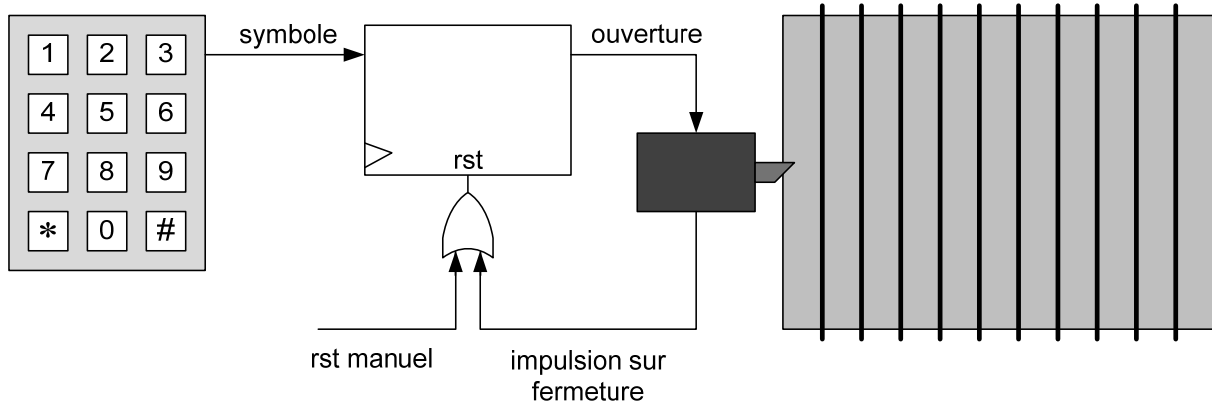


Figure 4 Système d'ouverture de porte

Au départ, la porte est fermée. Pour l'ouvrir, l'utilisateur doit entrer le symbole '*', taper les quatre chiffres de son code et terminer la saisie de son code par le symbole '#'. Si le code est valide, le circuit maintient une tension sur le signal **ouverture** qui force l'ouverture de la porte. Le circuit est réinitialisé lorsque la porte est refermée, ce qui déclenche une impulsion sur le signal asynchrone **rst**.

- (4 pts)** Donnez un diagramme d'états pour le circuit séquentiel implémentant une machine à états commandant cette porte.
- (4 pts)** Donnez en VHDL une description possible d'un circuit séquentiel implémentant la machine à états décrite par votre diagramme d'états. L'entête du code VHDL que vous devez fournir suit :

```

package symboles is
  type symbole is ('#', '*', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9');
  type code is array(0 to 3) of symbole;
end symboles;

library ieee;
use ieee.std_logic_1164.all;
use work.symboles.all;

entity ouverture_porte is
  generic(
    mon_code : code := ('1','8', '6','4')
  );
  port(
    clk      : in std_logic;    -- signal d'horloge
    rst      : in std_logic;    -- signal de réinitialisation asynchrone
    entree   : in symbole;      -- signal de début de traitement
    ouverture : out std_logic   -- signal ouvrant la porte
  );
end ouverture_porte ;

architecture arch of ouverture_porte is

```

Question 5. (5 points)

Questions brèves.

a) **(1 pt)** Donnez quatre fréquences d'horloge qu'il est possible de générer exactement à partir d'une horloge de référence de 100 MHz, telles que :

- la première soit entre 10 kHz et 500 kHz ;
- la seconde entre 500 kHz et 1 MHz ;
- la troisième entre 1 MHz et 10 MHz et
- la quatrième entre 10 MHz et 100 MHz.

b) **(1 pt)** Considérez le processeur à usage spécifique et le processeur à usage général et donnez un avantage et un inconvénient à chacune de ces options pour implémenter un algorithme de tri sur FPGA. Vous pouvez utiliser l'exemple de la question 2 comme référence.

c) **(0.75 pt)** Donnez quatre étapes importantes dans le flot de conception d'un circuit numérique sur FPGA.

d) **(0.75 pt)** Discutez l'option d'utiliser un test pseudo-aléatoire pour tester une machine à états où le vecteur de test aléatoire alimente l'entrée du circuit séquentiel.

e) **(0.75 pt)** Donnez un exemple d'application où la latence initial d'une architecture à pipeline n'a aucun impact sur la performance du système.

f) **(0.75 pt)** Donnez un exemple d'application où la latence initial d'une architecture à pipeline a un impact critique sur la performance du système.