Nom : _____ Prénom : _____ Matricule : _____

INF3500 : Conception et réalisation de systèmes numériques

Examen intra #2 – lundi 6 novembre 2017

Q1	
Q2	
Q3	
Q4	
Total	

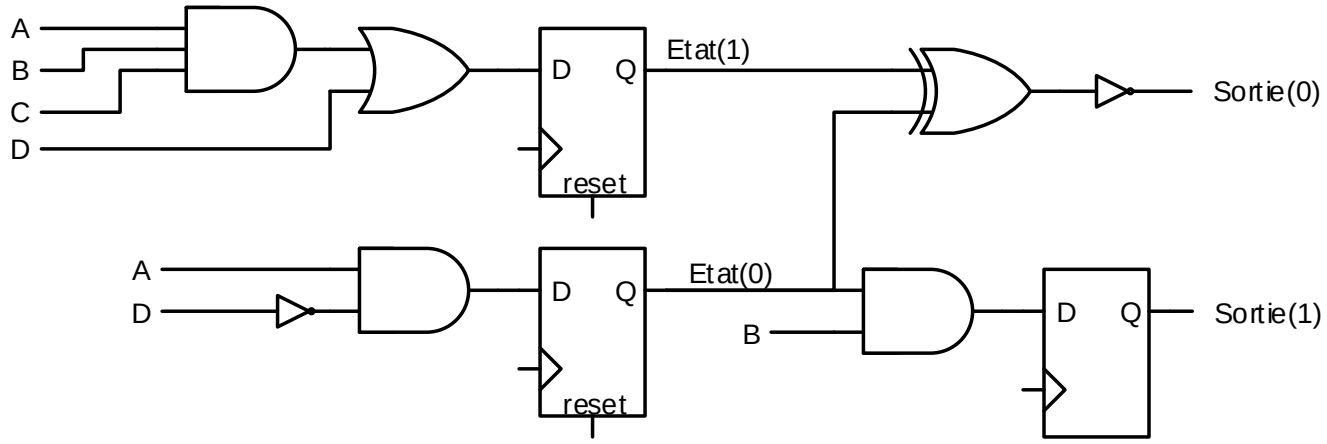
Durée: 1 heure. Documentation: Une feuille recto verso 8.5"×11" ou A4 permise.
Pondération: 10%. Calculatrice: Programmable permise.

Directives particulières:

- Ordinateurs interdits. Appareils mobiles interdits.
- Répondez à toutes les questions, la valeur de chaque question est indiquée.
- Répondez sur le questionnaire et remettez-le.
- Ne posez pas de questions. En cas de doute sur le sens d'une question, énoncez clairement vos suppositions.

Question 1. (2 points)

Décrivez le circuit suivant en code VHDL synthétisable. Les signaux de réinitialisation indiqués doivent être synchrones. Les bascules sont activées par des transitions positives du signal d'horloge.

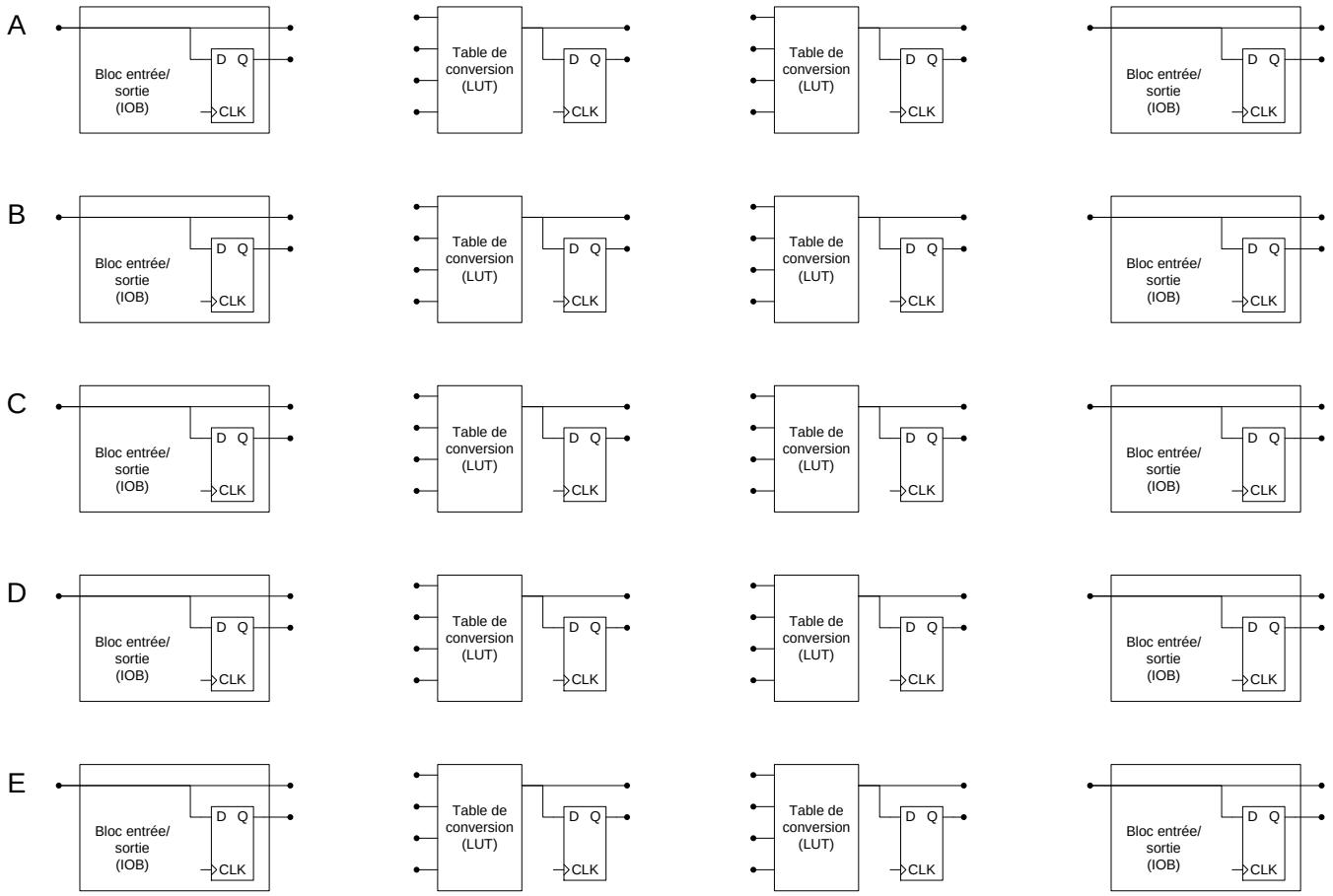


Utilisez le verso si nécessaire)

Question 2. (3 points)

Considérez le code VHDL et le modèle de FPGA suivants. Montrez, sur le modèle du FPGA, un résultat possible de la synthèse et de l'implémentation de ce code. Indiquez directement sur le dessin où les signaux et ports de sortie se situent ainsi que les interconnexions entre les blocs. Les interconnexions peuvent contourner les blocs. Indiquez quand une bascule doit être utilisée. Indiquez par une équation la fonction logique réalisée par chaque LUT que vous utilisez. Respectez l'assignation donnée pour les ports d'entrée.

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

entity module10 is
   port (
      clk, A, B, C, D, E: in std_logic;
      X, Y, Z: out std_logic
   );
end module10;

architecture arch of module10 is

signal F, G, H : std_logic;

begin

   X <= not(A and B and E);
   Y <= G xor H;

   process(clk) is
   begin
      if rising_edge(CLK) then
         F <= A and B and C and D;
         G <= F xor E;
         H <= B or C or D;
      end if;
   end process;

   process(A, B, C)
   begin
      if A = '1' then
         Z <= B or C;
      else
         Z <= B and C;
      end if;
   end process;

end arch;
```



Question 3. (4 points)

L'exponentiation modulaire est une fonction très utile en cryptographie. Elle est définie par $R = B^E \bmod M$. Par exemple, pour $B, E, M = \{5, 3, 32\}$, on a $R = 125 \bmod 32 = 29$. L'exponentiation modulaire est difficile à calculer directement pour de grands nombres (par exemple, quand B, E et M sont exprimés avec 512 bits ou plus). Il existe par contre un algorithme itératif qui permet d'obtenir la réponse en e cycles d'horloge, où e est le nombre de bits de E. On peut décrire cet algorithme avec les micro-opérations suivantes.

init : R ← 1, B ← B0, E ← E0, M ← M0, fini ← 0; init' ET E mod 2 = 1: R ← (R × B) mod M; init' : B ← (B × B) mod M;	init' : E ← E >> 1; init' ET E = 0 : fini ← 1

Donnez un diagramme d'un chemin des données correspondant à ces micro-opérations. Supposez que R, B et E sont exprimés avec W bits, que $M = 2^W$, que B0, E0 et M0 sont les valeurs entrées du système, et que init est un signal de contrôle pour démarrer les calculs.

(utilisez le verso si nécessaire)

Question 4. (1 point)

a. (0.5 point) Expliquez pourquoi un ensemble de vecteurs de tests donnant une couverture de code de 100% n'est pas nécessairement suffisant pour vérifier adéquatement un système numérique. Donnez un exemple.

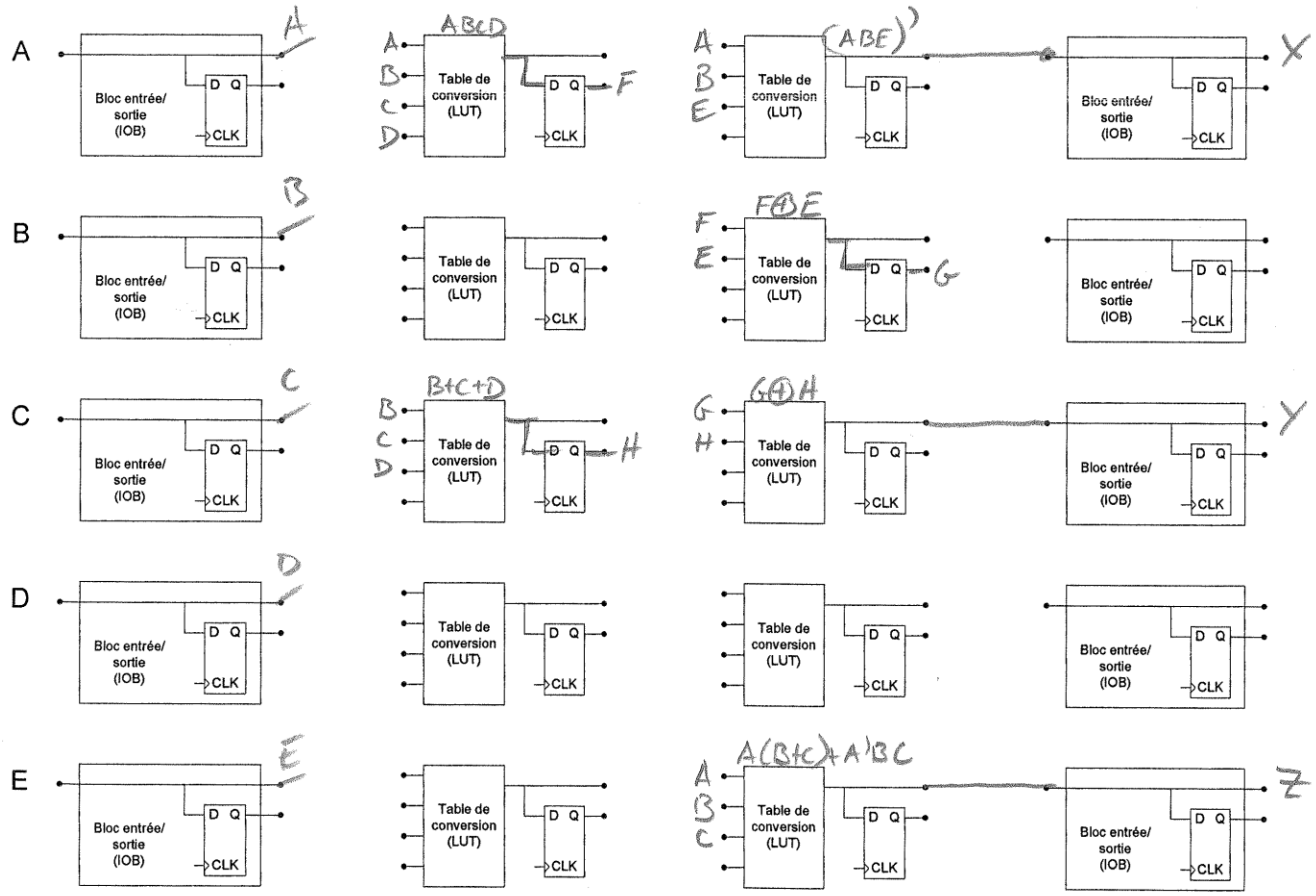
b. (0.5 point) Expliquez les principes du partitionnement en classe pour vérifier un système numérique. Expliquez comment choisir les classes. Expliquez comment composer un ensemble fort de vecteurs de tests à partir des classes.

Solutions

Q1.

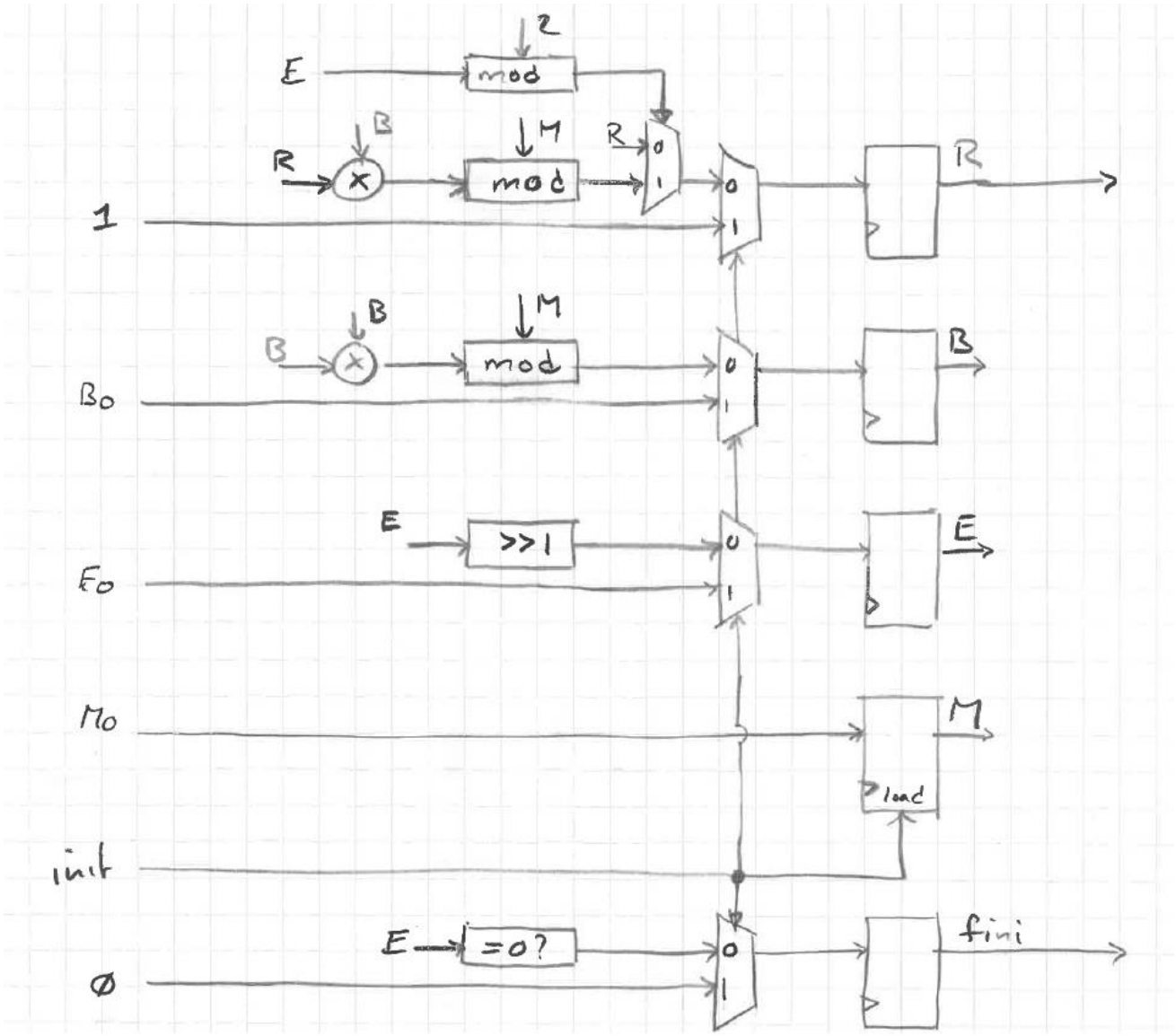
```
library ieee;
use ieee.std_logic_1164.all;

entity monModule5 is
    port (
        reset, clk: in std_logic;
        A, B, C, D : in std_logic;
        sortie : out std_logic_vector(1 downto 0)
    );
end monModule5;

architecture C2_201703 of module5 is
signal etat : std_logic_vector(1 downto 0);
begin

process(CLK, reset) is
begin
    if (rising_edge(CLK)) then
        if (reset = '1') then
            etat <= "00";
        else
            etat(1) <= (A and B and C) or D;
            etat(0) <= A and not(D);
            sortie(1) <= B and etat(0);
        end if;
    end if;
end process;

sortie(0) <= etat(1) xnor etat(0);

end C2_201703;
```

Q2. Code VHDL et modèle de FPGA



Q3. Exponentiation modulaire

Il y a quelques autres réponses possibles, qui utilisent par exemple le signal de chargement des registres en complément de multiplexeurs à l'entrée.



Q4.

a. Couverture de code

La couverture de code n'indique que si certaines situations ont été exercées ou non, sans égard à la fonctionnalité du système.

La métrique de couverture de code complète les autres types de tests et donne une certaine assurance au concepteur que le circuit est bien vérifié. Mais une couverture de 100% pour un ensemble de vecteurs de test ne garantit pas que le circuit rencontre toutes ses spécifications, uniquement que chaque énoncé de la description du modèle a été exécuté au moins une fois. Et comme pour les autres stratégies de sélection de vecteurs de test, la sortie doit être correcte pour chaque vecteur appliqué.

En guise d'exemple, imaginons un module qui doive faire soit l'addition ou la soustraction de deux nombres de 8 bits. Supposons que la partie soustraction n'a pas été incluse dans le modèle. Supposons qu'on construise un ensemble de vecteurs de test qui vérifie l'addition correctement et qui donne 100% de couverture de code. Tout semble correct, sauf que la partie de la spécification concernant la soustraction n'a été ni incluse dans la description ni vérifiée.

b. Partitionnement en classes

Le partitionnement en classes consiste décomposer l'ensemble des valeurs possibles de chaque entrées du système en différentes classes. Le choix des classes dépend principalement de la réponse attendue du système aux éléments de ces classes.

Le principe du partitionnement en classe est qu'un élément d'une classe est réputé être représentatif de tous les éléments de sa classe. Donc si le système fonctionne correctement pour une valeur en entrée dans une des classes, on s'attend à ce qu'il fonctionne correctement pour toutes les autres entrées de la classe.

Un vecteur de test est un ensemble de valeurs appliquées aux entrées du système à un moment donné.

Pour un test faible, chaque classe doit contribuer au moins une fois à un vecteur de test. Le nombre minimal de vecteurs de test est égal au nombre maximal de classes pour chacune des entrées. Pour un test fort, un élément de chaque classe doit être choisi en combinaison avec un élément de chacune des autres classes au moins une fois dans l'ensemble des vecteurs de test. Le nombre minimal de vecteurs de test est égal au produit du nombre de classes pour chaque entrée.