INF3500 : Conception et réalisation de systèmes numériques

Examen intra #2 – 31 octobre 2019

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

Question 1. (1 point)

a. (0.5 point) Donnez la représentation de la fraction +43/13 en code binaire, avec 4 bits pour la partie entière et 8 bits pour la partie fractionnaire :

_____ , _____

b. (0.5 point) Considérez le code VHDL suivant qui modélise une machine à états. Combien de vecteurs de tests seraient nécessaires, au minimum, pour faire un test exhaustif? Justifiez complètement votre réponse.

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity module13 is
 port (
 clk, reset : std_logic;
 A, B, C : in std_logic;
 unnombre : in signed(2 downto 0);
 unentier : in integer range 0 to 15;
 unbool : in boolean
 );
end module13;
```

```
architecture arch of module13 is
type type_etat is (s1, s2, s3, s4, s5, s6);
signal etat : type_etat;
begin

 --
 -- des énoncés cachés!
 --
 --

end arch;
```

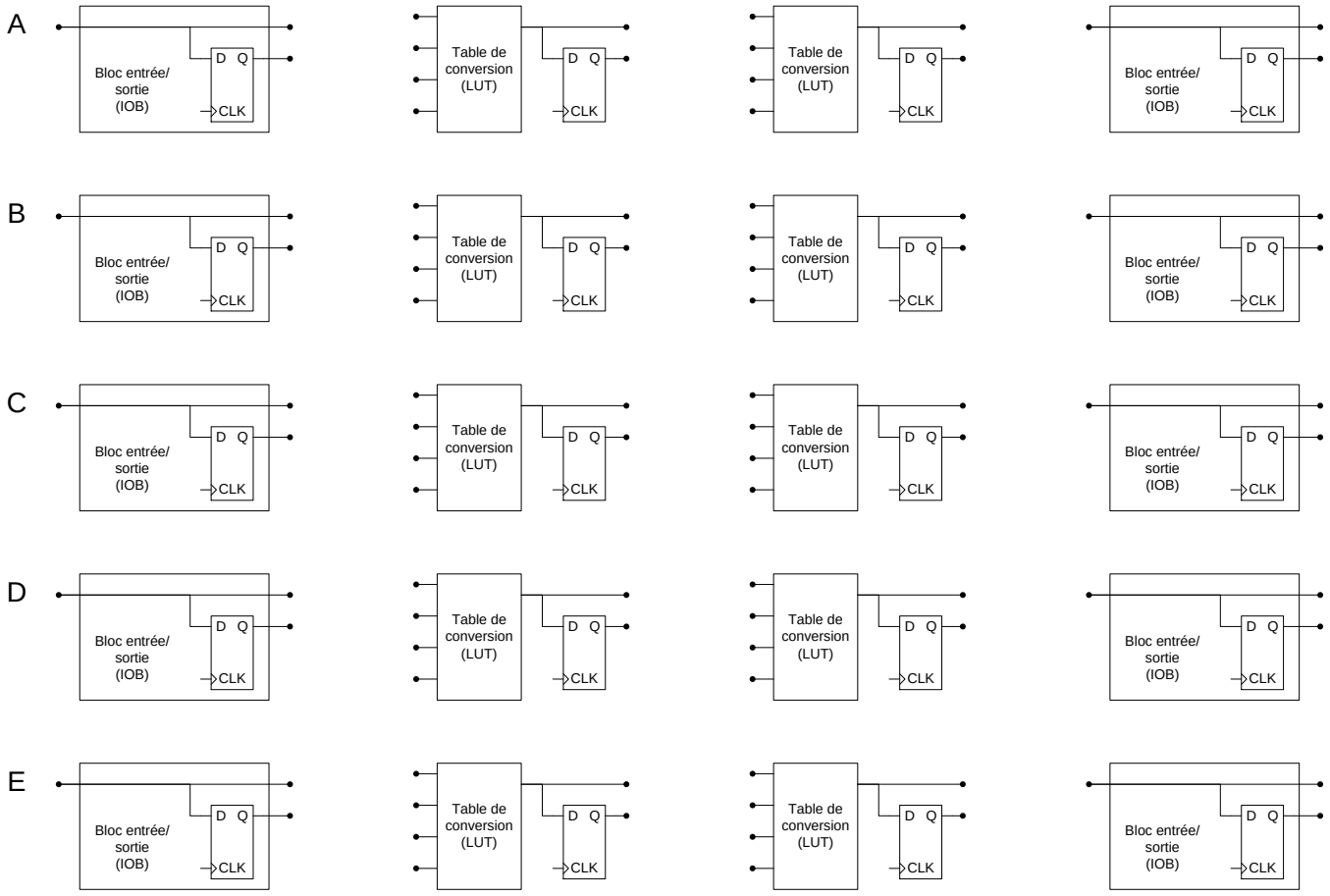
Question 2. (2 points)

Considérez le code VHDL et le modèle de FPGA suivants.

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

entity module11 is
 port (
    clk, A, B, C, D, E: in std_logic;
    X, Y, Z: out std_logic
 );
end module11;

architecture arch of module11 is

signal Q : std_logic_vector(1 downto 0) := "10";
signal R, S, T : std_logic := '1';

begin

 R <= '1' when A & B & C = "011" else '0';
 X <= T or Q(0);
 Z <= R or S or T;
```

```
 process(clk) is
 begin
    if rising_edge(CLK) then
       Q(1) <= Q(0) and R and S;
       Q(0) <= R xor S;
       Y <= Q(1) and Q(0);
    end if;
 end process;

 process(D, E)
 variable V : std_logic := '0';
 begin
    V := D;
    S <= V or E;
    V := not(D);
    T <= V and E;
 end process;

end arch;
```

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 chaque signal et port de sortie se situe ainsi que les interconnexions entre les blocs. Les interconnexions peuvent contourner les blocs. Indiquez quand une bascule doit être utilisée. Indiquez clairement, par une équation ou une porte logique, la fonction logique réalisée par chaque LUT que vous utilisez.

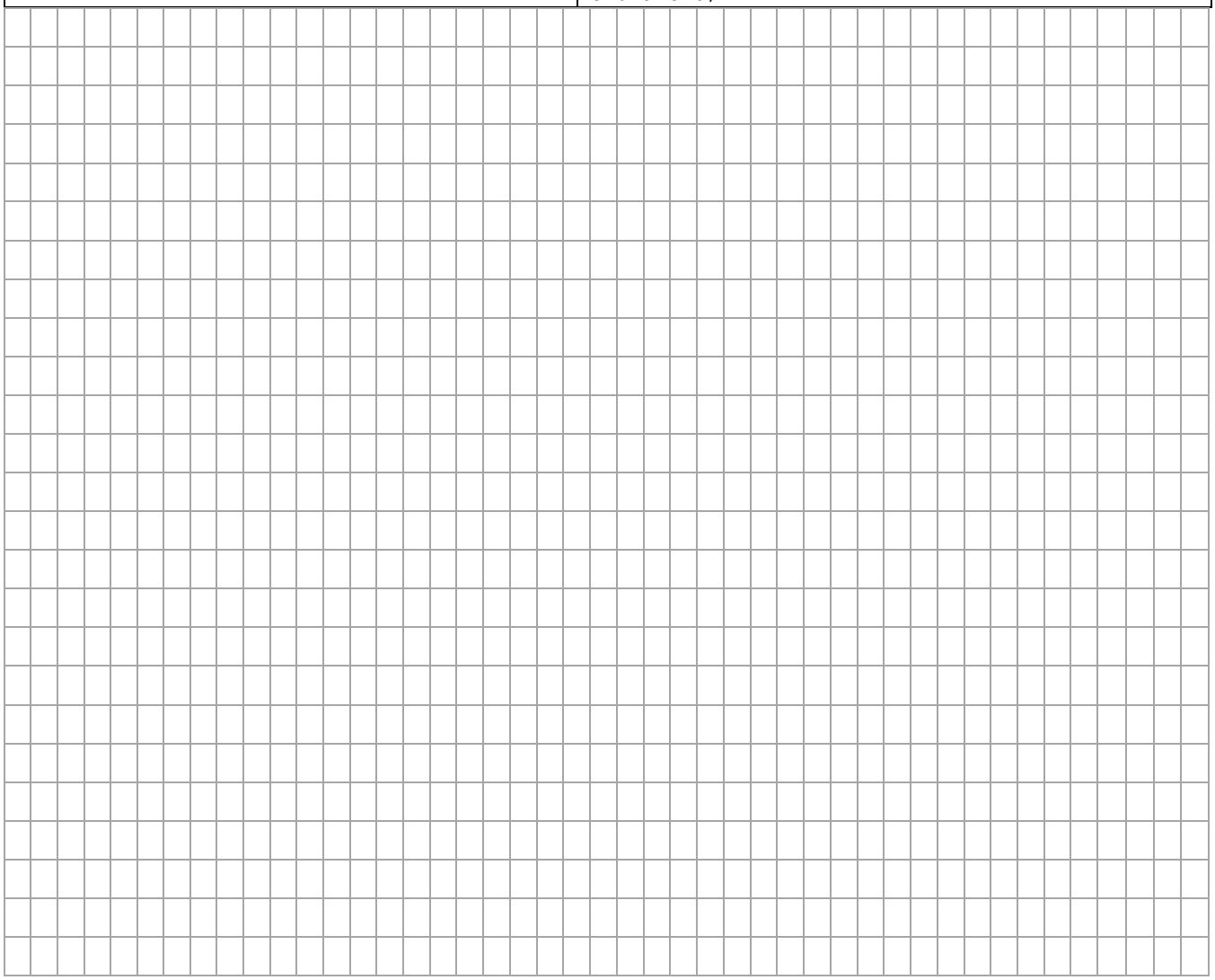


Question 3. (4 points)

a. (3 points) Donnez le diagramme d'un chemin des données correspondant au code VHDL suivant.

```
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity pgfc2 is
 generic (
    W : integer := 12
 );
 port (
    CLK : in STD_LOGIC;
    A0, B0 : in unsigned(W - 1 downto 0);
    init : in std_logic;
    sortie : out unsigned(W - 1 downto 0);
    fini : out std_logic
 );
end pgfc2;


architecture arch3 of pgfc2 is

signal A, B : unsigned(W - 1 downto 0);
```

```
begin
 process(CLK) is
 begin
    if rising_edge(CLK) then
       if init = '1' then
          A <= A0;
          B <= B0;
          fini <= '0';
       elsif A > B then
          A <= A - B;
       elsif A < B then
          B <= B - A;
       else
          if A = B then
             fini <= '1';
          else
             fini <= '0';
          end if;
       end if;
    end if;
 end process;
 sortie <= A;
end arch3;
```



b. (1 point) Estimez la quantité de tables de conversion (LUT) et de bascules (FF) nécessaires pour implémenter le module pgfc2 de la partie a. dans un FPGA de la série 7 de Xilinx. Justifiez complètement votre estimation.

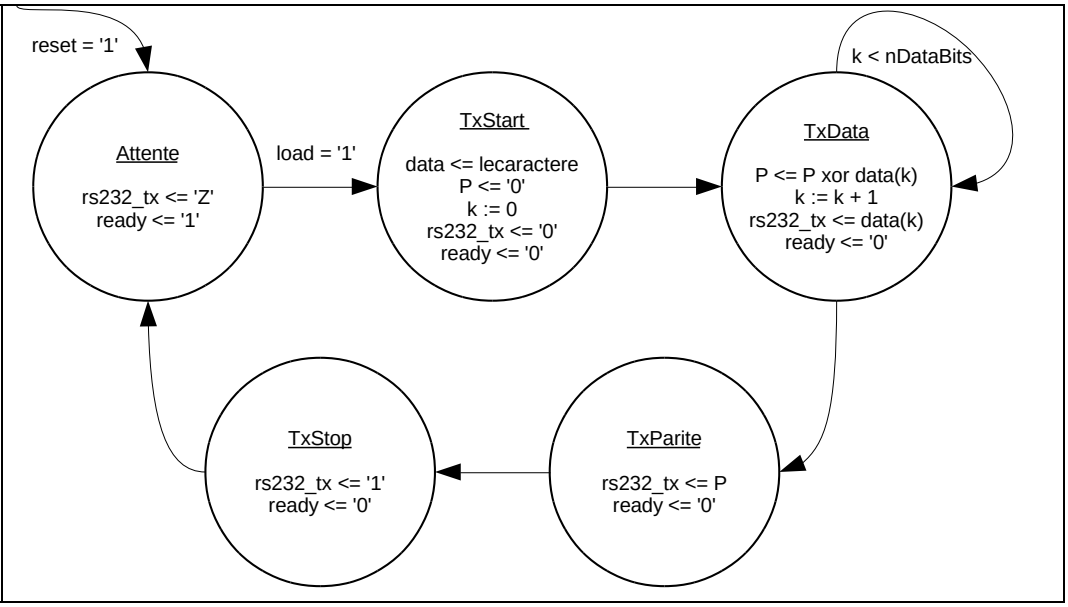
Type	Quantité	Justification
LUT		
FF		

Question 4. (3 points)

Donnez une architecture en VHDL synthétisable correspondant au diagramme d'états suivants, pour la déclaration d'entité donnée.

Le diagramme d'états montre le comportement d'un transmetteur pour des communications numériques sérielles RS-232.

`P` et `data` sont des signaux internes, et `k` est une variable dans un processus.




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity rs232tx is
 generic (nDataBits : positive := 8); -- nombre de bits de données
 port(
    reset : in std_logic;
    bitClk : in std_logic; -- horloge des bits
    lecaractere : in std_logic_vector(nDataBits - 1 downto 0); -- caractère à transmettre
    load : in std_logic; -- il faut charger le caractère et débuter la transmission
    ready : out std_logic; -- le système est prêt à transmettre un nouveau caractère
    rs232_tx : out std_logic -- signal de transmission RS-232
 );
end rs232tx;
```

Solutions

1. a. 43 / 13 = 3 + 4 / 13

partie entière : 0011

partie fractionnaire sur 8 bits : 4 / 13 ≈ 79 / 256 = (64 + 8 + 4 + 2 + 1) / 256 = 0,01001111 en base 2

b.

Test exhaustif

Il y a effectivement 11 bits d'entrée : 3 pour A, B, C; 3 pour « unnombre »; 4 pour « unentier »; et 1 pour « unbool ». On ne compte pas l'horloge ni le reset.

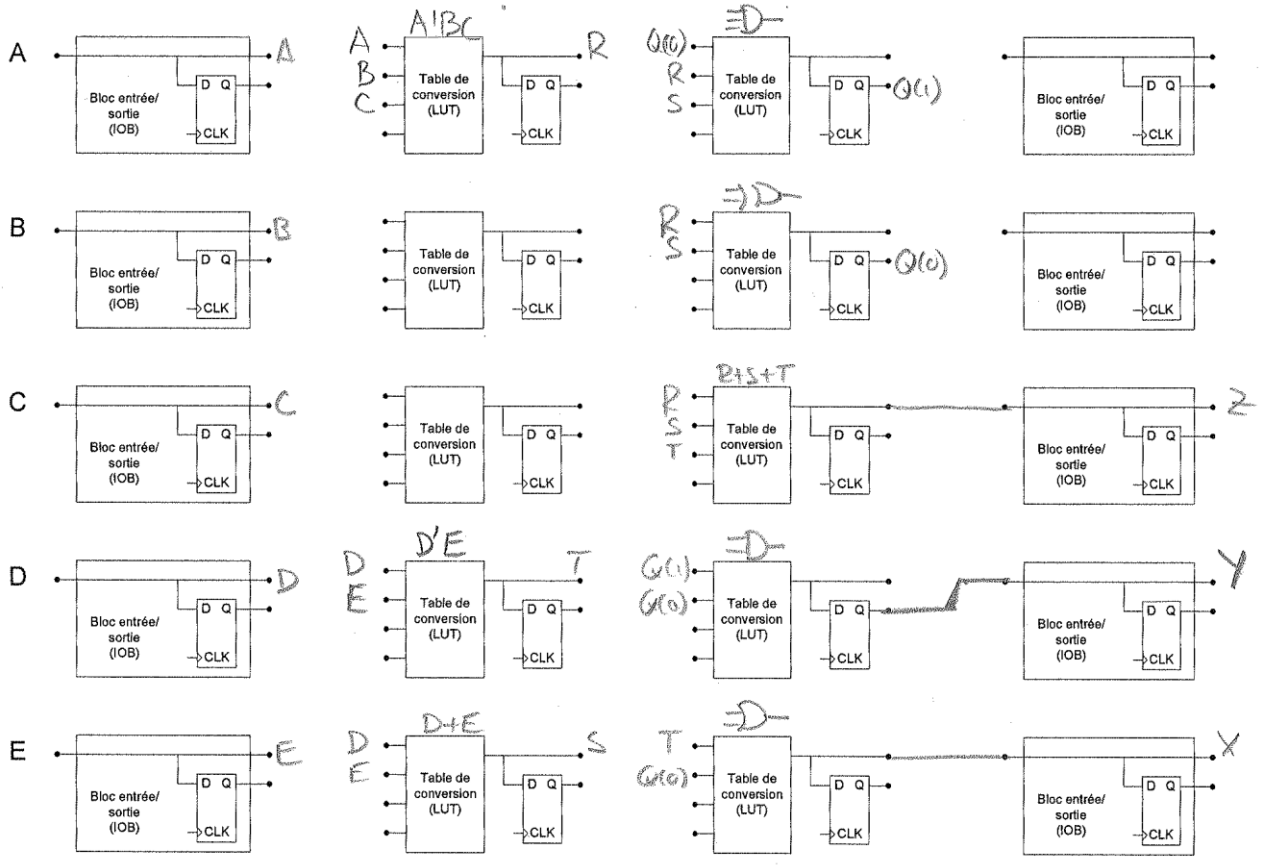
Il y a 6 états.

Dans chaque état, il faut essayer chacune des combinaisons possibles d'entrée. Donc au total on a $6 \times 2^{11} = 12288$ vecteurs de test.

Si on suppose que les 6 états sont encodés sur 3 bits, alors le nombre de combinaisons possibles serait $8 \times 2^{11} = 16384$ vecteurs de test.

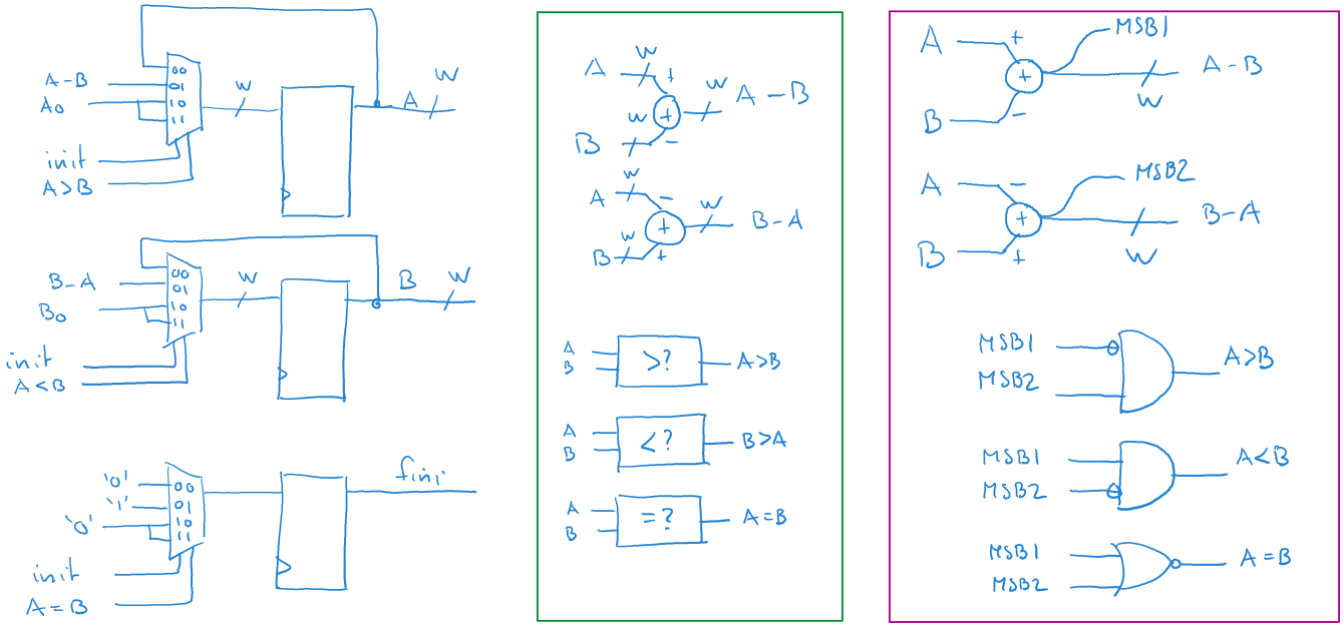
2. Implémentation sur FPGA

Une solution possible. Pour la sortie Y, on peut aussi utiliser la bascule qui est dans le bloc d'entrée-sortie plutôt que la bascule liée à la table de conversion.

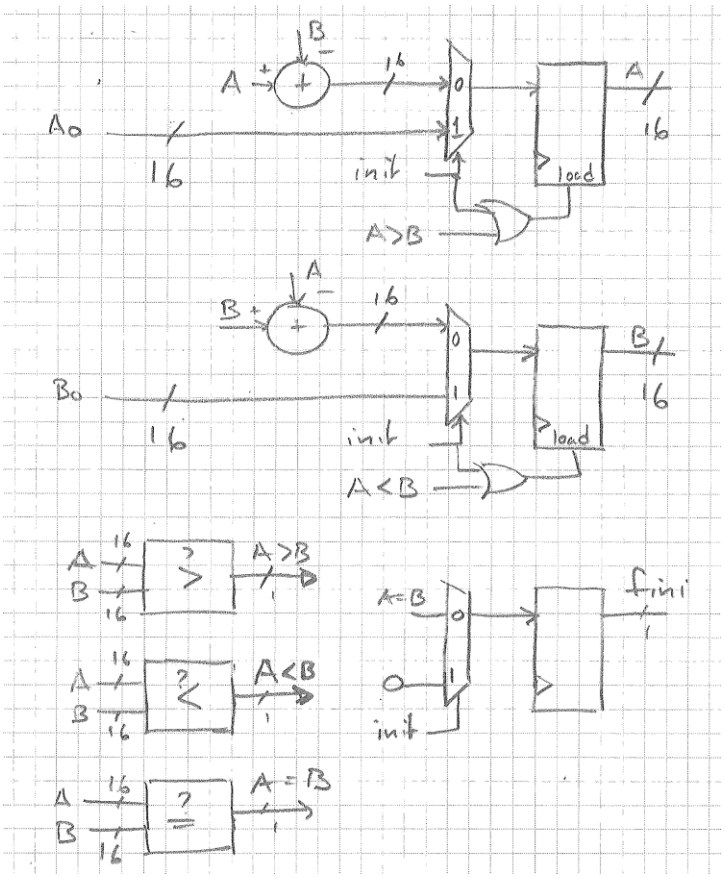


3. PGFC

a. Une solution possible. Le contenu des deux rectangles est équivalent et un seul est nécessaire. Le rectangle de droite (rouge) montre une façon d'implémenter les fonctions de comparaisons du rectangle de gauche (vert).



Une autre possibilité (montrée ici pour W = 16)



b. Le nombre de ressources nécessaires dépend de la façon de réaliser les fonctions dans le chemin des données.

Le décompte montré ici suppose la solution montrée avec le rectangle en vert pour les différences et les comparaisons indépendantes.

LUT :

Soustractions : 2 × 12 = 24 LUT

Comparaisons (<, >, =) : si on ne réutilise pas les soustractions, alors il faut 3 × 12 = 36 LUT additionnelles.

Multiplexeurs : dans le pire des cas, chaque multiplexeur nécessite une LUT par bit (c'est une fonction à 6 entrées), donc a besoin de 2 × 12 LUT = 24 pour A et B, et 1 LUT pour le mux de fini;

Total : 24 + 36 + 24 = 84 LUT environ

FF : il faut 25 registres : 12 pour A, 12 pour B et 1 pour fini

DSP48E : on n'utilise aucune tranche DSP48E pour ce circuit

Le décompte montré ici suppose la solution montrée avec le rectangle en rouge.

LUT :

Soustractions : 2 × 12 = 24 LUT

Comparaisons (<, >, =) : en réutilisant les soustractions déjà faites on n'a besoin que de 3 LUT pour les fonctions qui prennent en entrée les MSB des deux soustractions.

Multiplexeurs : dans le pire des cas, chaque multiplexeur nécessite une LUT par bit (c'est une fonction à 6 entrées), donc a besoin de 2 × 12 LUT = 24 pour A et B, et 1 LUT pour le mux de fini;

Total : 24 + 3 + 24 = 51 LUT environ

FF : il faut 25 registres : 12 pour A, 12 pour B et 1 pour fini

DSP48E : on n'utilise aucune tranche DSP48E pour ce circuit

4. RS232

Le code montré ici est du style 'à un seul processus'. Il a pour conséquence d'instancier des bascules supplémentaires pour les sorties rs232_tx et ready. Ces sorties seront donc effectivement un cycle en retard sur les états. Il est possible d'écrire une solution avec un style à un seul processus sans ce retard, mais alors il faut spécifier les valeurs des ports de sorties dans les états précédents, en ajoutant des conditions … c'est possible mais pas élégant.

```
architecture arch of rs232tx is

constant startBit : std_logic := '0';
constant stopBit : std_logic := '1';

type etat_type is (attente, txstart, txdata, txparite, txstop);
signal etat : etat_type := attente;

signal data : std_logic_vector(nDataBits - 1 downto 0);
signal parite : std_logic;

process(reset, bitClk)
 variable indice : natural range 0 to nDataBits;
 begin
    if reset = '1' then
       ready <= '0';
       rs232_tx <= 'Z';
       etat <= attente;
    elsif rising_edge(bitClk) then
```

```
       case etat is

          when attente =>
          ready <= '1';
          rs232_tx <= 'Z';
          if load = '1' then
             data <= lecaractere;
             etat <= txstart;
          end if;

          when txstart =>
          ready <= '0';
          rs232_tx <= startbit;
          parite <= '0';
          indice := 0;
          etat <= txdata;

          when txdata =>
          ready <= '0';
          rs232_tx <= data(indice);
          parite <= parite xor data(indice);
          indice := indice + 1;
          if indice = nDatabits then
             etat <= txparite;
          end if;
```

```
          when txparite =>
          ready <= '0';
          rs232_tx <= parite;
          etat <= txstop;

          when txstop =>
          ready <= '0';
          rs232_tx <= stopBit;
          etat <= attente;

          when others =>
          ready <= '0';
          rs232_tx <= 'Z';
          etat <= attente;

       end case;
    end if;
 end process;

end arch;
```

La solution suivante a un style 'à deux processus' et elle est préférable bien qu'elle soit plus verbeuse. Les sorties rs232_tx et ready sont spécifiées dans un processus séparé, qui ne dépend pas de l'horloge, mais seulement de l'état présent et des signaux data, indice et parite. Les deux sorties n'auraient alors pas de registre supplémentaire et elles seraient synchronisées avec les états. Il faut alors que le compteur indice soit un signal – et non une variable – afin d'être écrit par un processus et lu par l'autre.

```
architecture deuxprocessus of rs232tx is

constant startBit : std_logic := '0';
constant stopBit : std_logic := '1';

type etat_type is (attente, txstart, txdata, txparite, txstop);
signal etat : etat_type := attente;

signal data : std_logic_vector(nDataBits - 1 downto 0);
signal parite : std_logic;
signal indice : natural range 0 to nDataBits;

begin
```

```
 process(reset, bitClk)
 begin

    if reset = '1' then
       etat <= attente;

    elsif rising_edge(bitClk) then

       case etat is

          when attente =>
          if load = '1' then
             data <= lecaractere;
             etat <= txstart;
          end if;

          when txstart =>
          parite <= '0';
          indice <= 0;
          etat <= txdata;

          when txdata =>
          parite <= parite xor data(indice);
          indice <= indice + 1;
          if indice = nDatabits - 1 then
             etat <= txparite;
          end if;

          when txparite =>
          etat <= txstop;

          when txstop =>
          etat <= attente;

          when others =>
          etat <= attente;

       end case;

    end if;
 end process;
```

```
--  process(etat, data, indice, parite)
 process(all)
 begin

    case etat is

       when attente =>
       ready <= '1';
       rs232_tx <= 'Z';

       when txstart =>
       ready <= '0';
       rs232_tx <= startbit;

       when txdata =>
       ready <= '0';
       rs232_tx <= data(indice);

       when txparite =>
       ready <= '0';
       rs232_tx <= parite;

       when txstop =>
       ready <= '0';
       rs232_tx <= stopBit;

       when others =>
       ready <= '0';
       rs232_tx <= 'Z';

    end case;

 end process;


end deuxprocessus;
```