INF3500 : Conception et réalisation de systèmes numériques

Examen intra

Jeudi 22 octobre 2009

Durée: 2h.
Pondération: 20%.
Documentation: Une feuille recto verso 8.5”×11” ou A4 permise.
Calculatrice: Programmable permise.
Directives particulières:
Ordinateurs interdits
Répondre à toutes les questions, la valeur de chaque question est indiquée.
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. (4 points)

Donnez une architecture pour les entités suivantes en VHDL synthétisable, en respectant les spécifications données en commentaires.

a.

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

-- bascule T
-- La sortie de la bascule T passe de 0 à 1 ou de 1 à 0 sur une
-- transition positive d'horloge quand l'entrée enable = 1.
-- Sa sortie est inchangée quand enable = 0.
-- La sortie est remise à 0 de façon asynchrone quand reset_n = 0.
entity basculeT is
port (
    reset_n, clk, enable : in std_logic;
    T : out std_logic
);
end basculeT;
```

Une solution possible :

```
architecture arch of basculeT is
signal Ttemp : std_logic;
begin
    T <= Ttemp;
    process(reset_n, clk) is
    begin
        if reset_n = '0' then
            Ttemp <= '0';
        elsif rising_edge(clk) then
            if enable = '1' then
                Ttemp <= not(Ttemp);
            end if;
        end if;
    end process;
end arch;
```

b.

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

-- compteurSecondes
-- La sortie secondes de ce compteur est incrémentée à chaque transition positive
-- d'horloge, allant de 0 à 59 puis recommençant à 0.
-- La sortie uneminute est menée à 1 pour une période d'horloge quand le compte
-- passe de 59 à 0, et à 0 autrement.
-- Le compte est remis à zéro de façon synchrone quand reset_n = 0.
entity compteurSecondes is
port (
reset_n, clk1Hz : in std_logic;
secondes : out unsigned(5 downto 0);
uneminute : out std_logic
);
end compteurSecondes;
```

Une solution possible :

```
architecture arch of compteurSecondes is
signal secondes_t : natural range 0 to 59;
begin
   secondes <= to_unsigned(secondes_t, secondes'length);
   process(clk1Hz) is
   begin
      if rising_edge(clk1Hz) then
         if reset_n = '0' then
            secondes_t <= 0;
            uneminute <= '0';
         else
            if secondes_t = 59 then
               secondes_t <= 0;
               uneminute <= '1';
            else
               secondes_t <= secondes_t + 1;
               uneminute <= '0';
            end if;
         end if;
      end if;
   end process;
end arch;
```

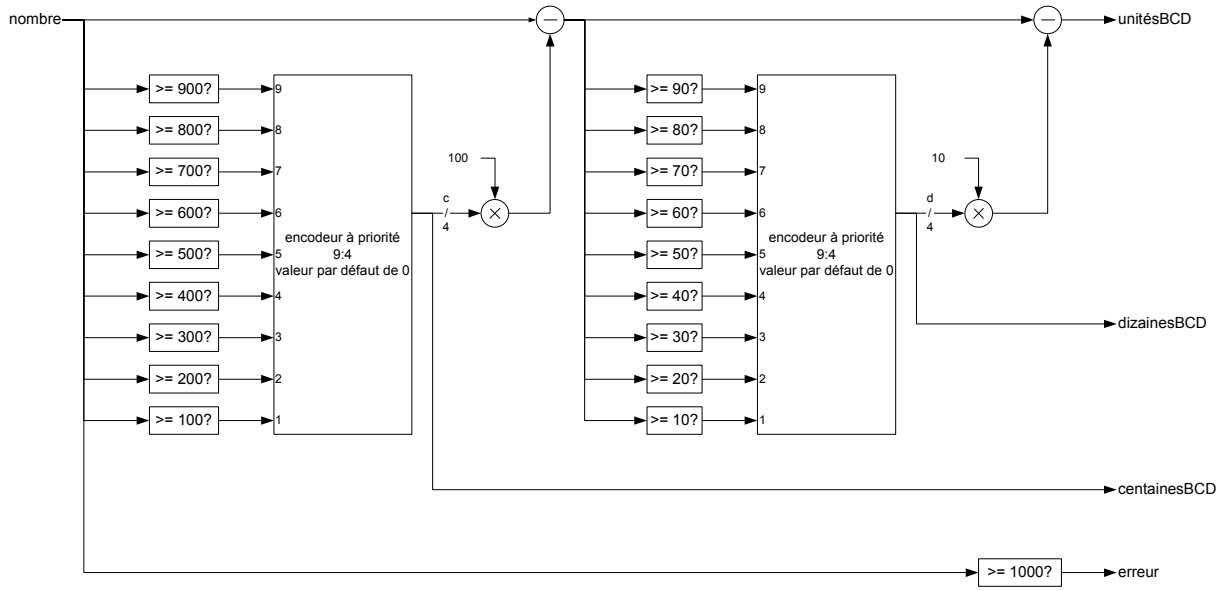
Question 2. (3 points)

Considérez le code VHDL suivant. Le circuit accepte en entrée un nombre non signé encodé sur 10 bits, et il a trois sorties. Ces sorties donnent le nombre de centaines, de dizaines et d'unités (de 0 à 9 inclusivement, encodées sur 4 bits) du nombre. La sortie erreur est activée si le nombre est égal ou supérieur à 1000.

Donnez un diagramme correspondant composé de modules combinatoires (multiplexeurs, décodeurs, etc.) et d'unités fonctionnelles (additionneurs, multiplicateurs, comparateurs, etc.) tel qu'il pourrait être généré par un processus de synthèse.

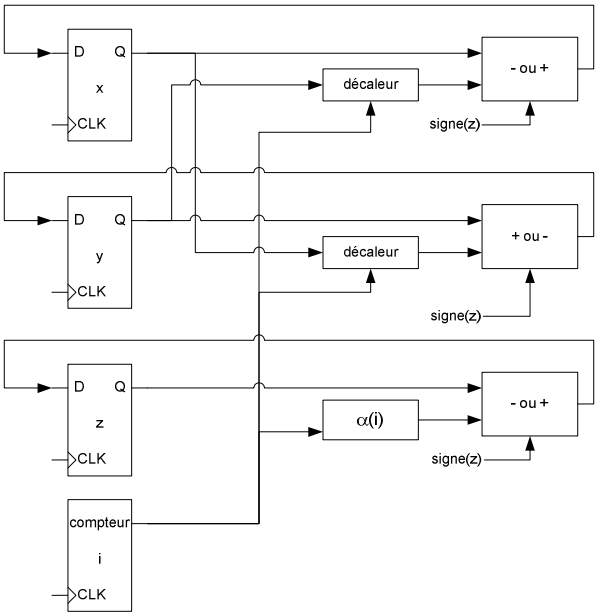
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity unsigned2dec is
    port(
        nombre : in unsigned(9 downto 0);
        centainesBCD, dizainesBCD, unitesBCD : out unsigned(3 downto 0);
        erreur : out std_logic
    );
end unsigned2dec;

architecture arch of unsigned2dec is
begin

    erreur <= '1' when nombre >= 1000 else '0';

    process(nombre)
    variable n, c, d, u : natural := 0;
    begin

        n := to_integer(nombre);

        for centaines in 9 downto 1 loop
            c := 0;
            if n >= centaines * 100 then
                c := centaines;
                exit;
            end if;
        end loop;

        n := n - c * 100;

        for dizaines in 9 downto 1 loop
            d := 0;
            if n >= dizaines * 10 then
                d := dizaines;
                exit;
            end if;
        end loop;

        u := n - d * 10;

        centainesBCD <= to_unsigned(c, 4);
        dizainesBCD <= to_unsigned(d, 4);
        unitesBCD <= to_unsigned(u, 4);

    end process;

end arch;
```

Solution :


nombre
>= 900?
>= 800?
>= 700?
>= 600?
>= 500?
>= 400?
>= 300?
>= 200?
>= 100?
9
8
7
6
5
4
3
2
1
encodeur à priorité
9:4
valeur par défaut de 0
100
c
/
4
>= 90?
>= 80?
>= 70?
>= 60?
>= 50?
>= 40?
>= 30?
>= 20?
>= 10?
encodeur à priorité
9:4
valeur par défaut de 0
10
d
/
4
unitésBCD
dizainesBCD
centainesBCD
>= 1000?
erreur

Question 3. (2 points)

Considérez le diagramme suivant, qui représente le chemin des données d'un processeur qui implémente l'algorithme CORDIC. L'algorithme CORDIC permet entre autres de calculer les fonctions trigonométriques à l'aide d'un processus itératif. Le diagramme comporte quatre registres, x, y, z et i; les autres blocs sont combinatoires.



Les registres x, y et z contiennent des valeurs signées en complément à deux représentées sur 12 bits. Les deux décaleurs produisent des versions décalées des contenus des registres x et y, et le nombre de positions de décalage est donné par la valeur du registre i. Les trois blocs d'addition/soustraction sont contrôlés par le signe de z : quand z est positif, celui du milieu additionne et les autres soustraient, et inversement quand z est négatif. Le compteur i débute à 0 et compte jusqu'à 7 puis s'arrête. Le bloc α(i) est une mémoire ROM de huit valeurs signées représentées sur 12 bits.

Donnez le code pour un processus qui implémente les registres x, y, z et i tels que montrés dans le diagramme. Vous n'avez pas à écrire de code pour le chargement des registres, pour les sorties du système, ni pour le contrôle du chemin des données. Supposez que x, y, z et i sont déclarés comme des signaux de type `signed`.

Solution :

```
process(clk)
begin
    if rising_edge(clk) then
        if z(z'left) = '0' then
            x <= x - shift_right(y, i);
            y <= y + shift_right(x, i);
            z <= z - alpha(i);
        else
            x <= x + shift_right(y, i);
            y <= y - shift_right(x, i);
            z <= z + alpha(i);
        end if;
        if (i < 7) then
            i <= i + 1;
        end if;
end if;
```

Question 4. (3 points)

Pour l'entité `unsigned2dec` décrite à la question #2, composez un banc d'essai qui effectue un test exhaustif et qui vérifie que la sortie est valide en tout temps. Une sortie valide indique correctement les unités, les dizaines et les centaines pour des nombres entre 0 et 999, inclusivement. Pour les autres nombres, le signal d'erreur doit être actif. Dans le cas d'une erreur, votre banc d'essai doit simplement afficher le message « erreur » à la console et se terminer.

Complétez le code VHDL suivant en donnant l'architecture.

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

entity unsigned2dec_TB is
end unsigned2dec_TB;

architecture TB_ARCHITECTURE of unsigned2dec_TB is
--- votre code ici

begin
--- votre code ici

end TB_ARCHITECTURE;
```

Une solution possible:

```
architecture TB_ARCHITECTURE of unsigned2dec_TB is

signal nombre : unsigned(9 downto 0);
signal centainesBCD, dizainesBCD, unitesBCD : unsigned(3 downto 0);
signal erreur : std_logic;

begin

UUT : entity unsigned2dec(arch)
port map (nombre, centainesBCD, dizainesBCD, unitesBCD, erreur);

process
variable n : natural;
begin
   for k in 0 to 1023 loop
      nombre <= to_unsigned(k, nombre'length);
      wait for 10 ns;
      n := to_integer(centainesBCD) * 100
      + to_integer(dizainesBCD) * 10
      + to_integer(unitesBCD);
      if (k <= 999) then
         assert erreur = '0' report "erreur!" severity failure;
         assert to_integer(centainesBCD) <= 9 report "erreur!" severity failure;
         assert to_integer(dizainesBCD) <= 9 report "erreur!" severity failure;
         assert to_integer(unitesBCD) <= 9 report "erreur!" severity failure;
         assert n = k report "erreur!" severity failure;
      else
         assert erreur = '1' report "erreur!" severity failure;
      end if;
   end loop;
   report "simulation terminée" severity failure;
end process;

end TB_ARCHITECTURE;
```

Question 5. (3 points)

Réponses brèves.

a. Considérez le code VHDL suivant. Quelle valeur prendra le port F lors de la simulation? Pourquoi?

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity exempleSignaux is
    port (F : out std_logic);
end exempleSignaux;
architecture arch1 of exempleSignaux is
begin
    F <= '1';
    F <= '0';
end arch1;
```

Solution : le port F prendra la valeur 'X' parce qu'il a deux sources de deux valeurs différentes.

b. Considérez le code VHDL suivant. Quelle valeur prendra le port F lors de la simulation si A = '1'? Pourquoi?

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity exempleSignaux is
    port (A : in std_logic; F : out std_logic);
end exempleSignaux;
architecture arch2 of exempleSignaux is
begin
    process(A)
    begin
        F <= A;
        F <= not(A);
    end process;
end arch2;
```

Solution : le port F prendra la valeur '0' parce que dans le processus c'est la dernière assignation à F qui sera gardée.

c. En VHDL, les types `STD_LOGIC_VECTOR`, `UNSIGNED` et `SIGNED` sont très utilisés pour la simulation et la synthèse de système numériques. Les trois types ont exactement la même définition :

```
type STD_LOGIC_VECTOR is array ( NATURAL range <>) of STD_LOGIC;
type UNSIGNED is array ( NATURAL range <>) of STD_LOGIC;
type SIGNED is array ( NATURAL range <>) of STD_LOGIC;
```

Expliquez la nécessité d'avoir ces trois types et ce qui rend l'utilisation de chacun distincte des autres.

Solution : Les trois types ont des utilisations différentes. Le type STD_LOGIC_VECTOR est utilisé pour les valeurs logiques, les deux autres pour des valeurs numériques. L'utilisation de chaque type est distincte à cause des fonctions, procédures et opérateurs qui sont définis et/ou surchargés pour chaque type.

d. Il y a quatre considérations fondamentales pour l'implémentation d'un système numérique, dont la taille du système. Énumérez les trois autres.

Solution : Taux de traitement, puissance consommée et précision des calculs.

e. Nommez et décrivez brièvement une des trois technologies de programmation les plus populaires pour FPGA.

Solution : SRAM, antifusibles et Flash – voir les notes de cours, section 3.7.7.

f. Pour le FPGA XC2VP30-FF896 utilisé dans les laboratoires, il y a deux façons d'implémenter de la mémoire RAM : la mémoire RAM distribuée (Distributed RAM) et la mémoire RAM bloc (Block RAM). Décrivez brièvement ces deux ressources et expliquez leurs différences.

Solution : La mémoire bloc est composée de modules spécifiques intercalés à travers les blocs de logique configurable du FPGA. La mémoire distribuée utilise les ressources internes de blocs de logique configurable.

Question 6. (2 points)

Illustrez, avec un diagramme, le flot de conception que vous avez suivi dans les laboratoires du cours INF3500 à ce jour, de la description du design à la vérification du fonctionnement de la puce.

Solution : figure 4-1 des notes de cours

Question 7. (3 points)

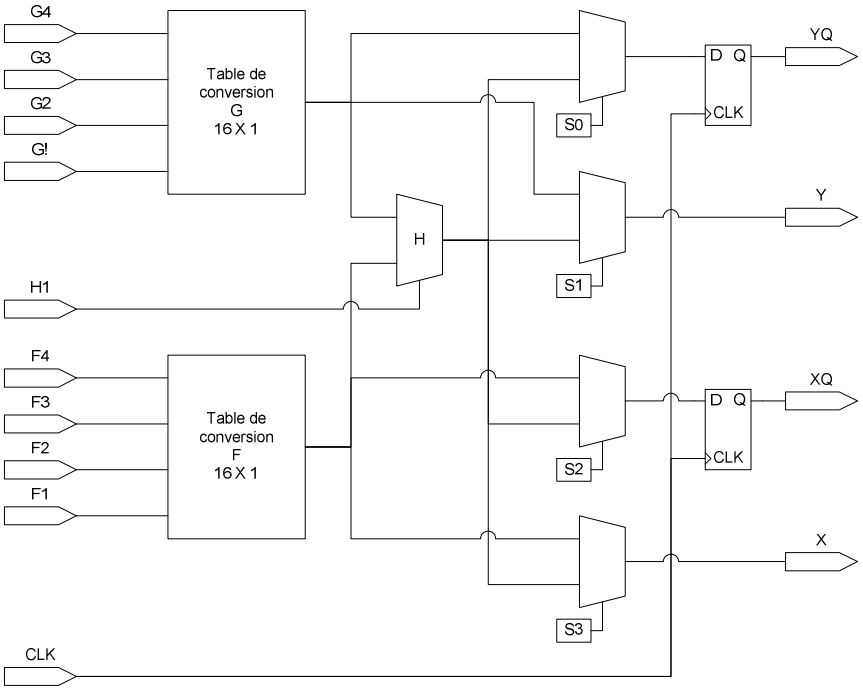
Considérez le code VHDL suivant. Montrez, sur le diagramme suivant d'une partie d'un FPGA, un résultat possible de la synthèse et de l'implémentation de ce code. Indiquez directement sur le dessin où chaque signal se situe, les interconnections entre les blocs, et le contenu de chacune des tables de conversion que vous utilisez.

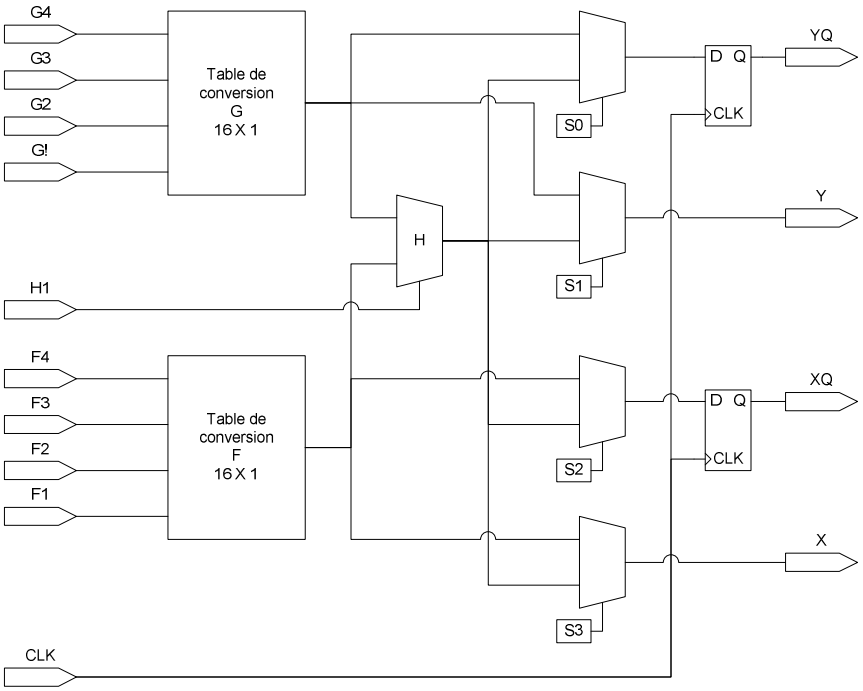
Remettez la feuille dans votre cahier d'examen.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity cctsequentielex1 is
    port (
        reset : in STD_LOGIC;
        CLK : in STD_LOGIC;
        X : in STD_LOGIC;
        Z : out STD_LOGIC
    );
end cctsequentielex1;

architecture arch1 of cctsequentielex1 is
signal A : STD_LOGIC; -- bascule A (en haut)
signal B : STD_LOGIC; -- bascule B (en bas)
begin

    process(CLK, reset) is
    begin
        if (reset = '0') then
           A <= '0';
           B <= '0';
        elsif (rising_edge(CLK)) then
           A <= A xor B;
           B <= x or not(B);
        end if;
    end process;

    -- signal de sortie
    z <= not(A or B);

end arch1;
```

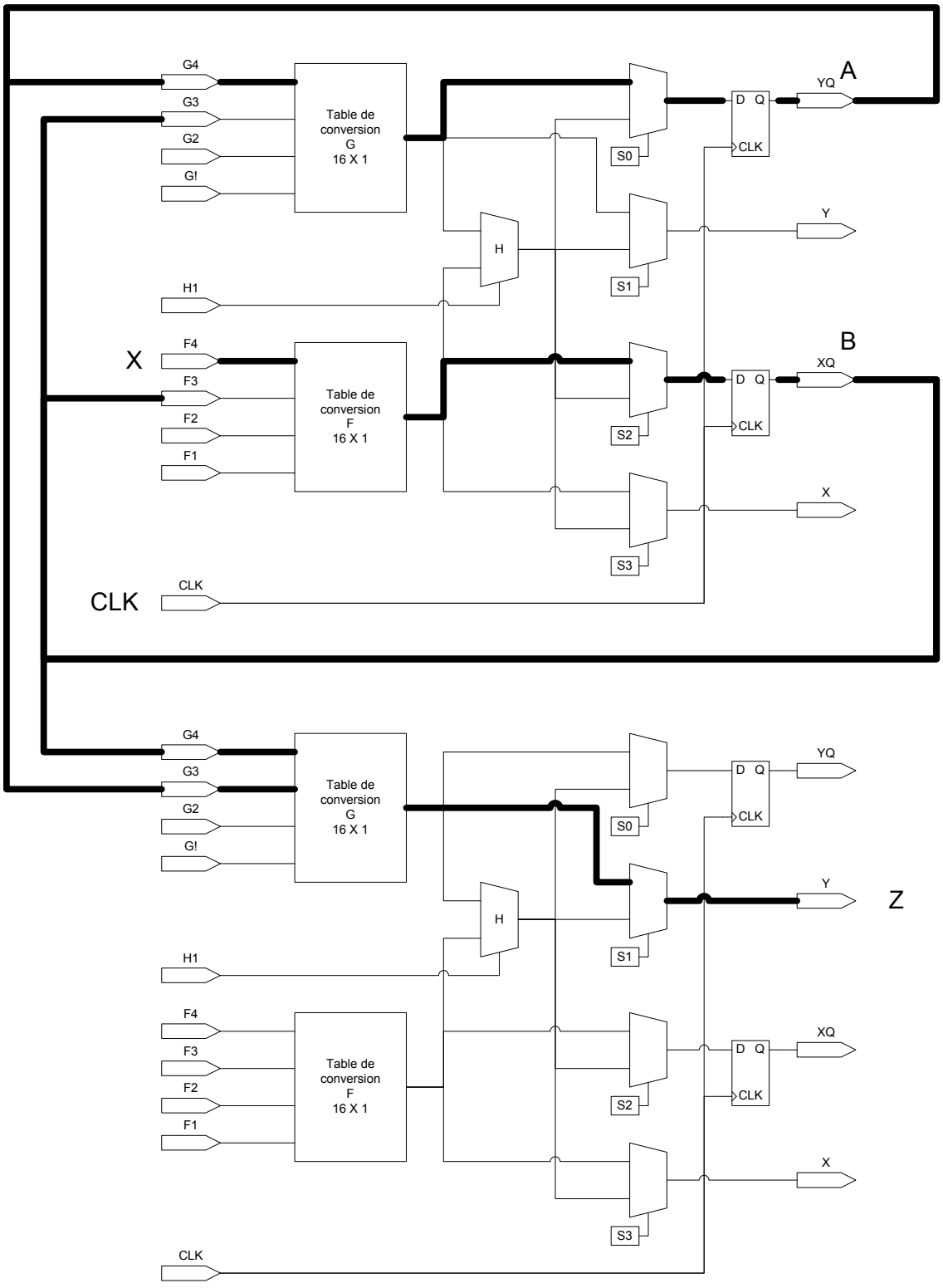
Annoter cette page et remettez-la avec votre cahier d'examen

Nom : _____ Matricule : _____





Solution :


G4
G3
G2
G!
Table de conversion G 16 X 1
H
H1
F4
F3
F2
F1
Table de conversion F 16 X 1
X
CLK
S0
S1
S2
S3
D Q
CLK
YQ
A
Y
XQ
B
X
Z

G4 (A)	*G3 (B)*	*G2 (NC)*	*G1 (NC)*	*G (A)*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

F4 (X)	*F3 (B)*	*F2 (NC)*	*F1 (NC)*	*F (B)*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

G4 (B)	*G3 (A)*	*G2 (NC)*	*G1 (NC)*	*G (Z)*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0