INF3500 : Conception et réalisation de systèmes numériques

Examen intra

Vendredi 25 février 2011

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 sur le questionnaire et le remettre.
- Ne posez pas de questions. En cas de doute sur le sens d'une question, énoncez clairement toute supposition que vous faites.

Question 1. (6 points)

La plupart des systèmes numériques utilisent le système binaire pour les opérations arithmétiques. Cependant, lors de l'affichage pour des êtres humains, le système décimal est utilisé. Les circuits de conversion entre les deux sont complexes et lents. La représentation décimale encodée en binaire (BCD – *binary coded decimal*) a pour but de remédier à cette situation. En BCD, chaque chiffre d'un nombre peut prendre une valeur entre 0 et 9, inclusivement, et est encodé sur quatre bits : {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001}.

Pour additionner deux chiffres BCD, on utilise un additionneur binaire à quatre bits et une retenue. Il y a deux cas: soit que la somme est un chiffre BCD valide (0 à 9) ou bien la somme est égale ou supérieure à dix. Dans le premier cas, aucune action n'est requise. Dans le deuxième cas, il faut ajouter +6 à la somme obtenue. La combinaison de la retenue et de la somme finale produit alors un résultat correct.

Exemple : addition 3 + 4 en BCD :

3	0 0 1 1	
+ 4	+ 0 1 0 0	
07	**0** | 0 1 1 1	← somme inférieure à dix, résultat correct (retenue 0, somme 7)

Exemple : addition 7 + 8 en BCD :

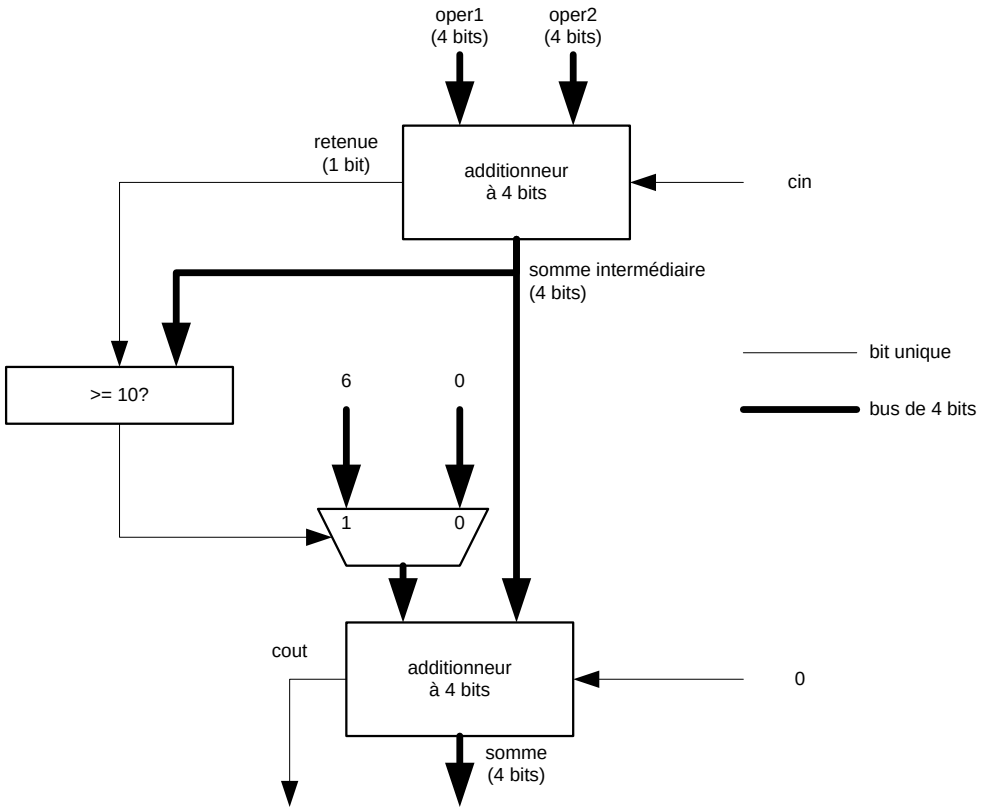
7	0 1 1 1	
+ 8	+ 1 0 0 0	
15	**0** | 1 1 1 1	← somme supérieure à 9, il faut ajouter +6
	+ 0 1 1 0	+ 6
	1 | 0 1 0 1	← résultat correct, retenue de 1, somme de 5

Exemple : addition 9 + 8 en BCD :

9	1 0 0 1	
+ 8	+ 1 0 0 0	
17	**1** | 0 0 0 1	← somme supérieure à 9, il faut ajouter +6
	+ 0 1 1 0	+ 6
	1 | 0 1 1 1	← résultat correct, retenue de 1, somme de 7

a. (3 points) Donnez un diagramme d'un circuit combinatoire pour un additionneur BCD à un chiffre. Vous pouvez utiliser des opérations logiques et arithmétiques (mais pas la division ni le modulo), la comparaison, des multiplexeurs, décodeurs et encodeurs, et toute autre porte logique de base. Identifiez bien chaque bloc et donnez la largeur en bits de tous les signaux. Votre circuit doit être purement combinatoire, il ne doit pas inclure d'éléments à mémoire.

Solution :



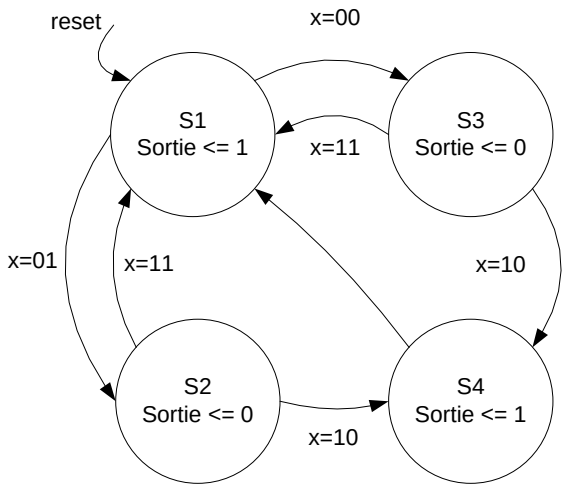
b. (3 points) Donnez le code VHDL correspondant à votre diagramme.

Solution :

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

entity addBCD is
    port (
        oper1, oper2 : in unsigned(3 downto 0);
        somme : out unsigned(3 downto 0);
        cout : out std_logic
    );
end addBCD;

architecture arch1 of addBCD is
begin
    process(oper1, oper2)
    variable t : unsigned(4 downto 0);
    begin
        t := ('0' & oper1) + ('0' & oper2);
        if (t > 9) then
            t := t + 6;
        end if;
        somme <= t(3 downto 0);
        cout <= t(4);
    end process;
end arch1;
```

Question 2. (4 points)

Considérez le diagramme d'états suivant, ainsi que du code VHDL partiel.
Complétez le code VHDL pour correspondre au diagramme d'états.




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

entity controle201101Q2 is
    port (
        reset, CLK : in STD_LOGIC;
        x : in STD_LOGIC_VECTOR(1 downto 0);
        sortie : out STD_LOGIC
    );
end controle201101Q2;

architecture arch of controle201101Q2 is
-- Votre code ici

-- Solution
type type_etat is (S1, S2, S3, S4);
signal etat : type_etat := S1;

begin
-- Votre code ici

-- Solution
    process(CLK, reset) is
    begin
        if (reset = '0') then
           etat <= S1;
        elsif (rising_edge(CLK)) then
           case etat is
              when S1 =>
                 if x = "00" then
                    etat <= S3;
                 elsif x = "01" then
                    etat <= S2;
                 end if;
              when S2 | S3 =>
                 if x = "10" then
                    etat <= S4;
                 elsif x = "11" then
                    etat <= S1;
                 end if;
              when S4 =>
                 etat <= S1;
              when others =>
                 etat <= S1;
           end case;
        end if;
    end process;

    process(x, etat)
    begin
        case etat is
           when S1 | S4 =>
              sortie <= '1';
           when others =>
              sortie <= '0';
        end case;
    end process;


end arch;
```

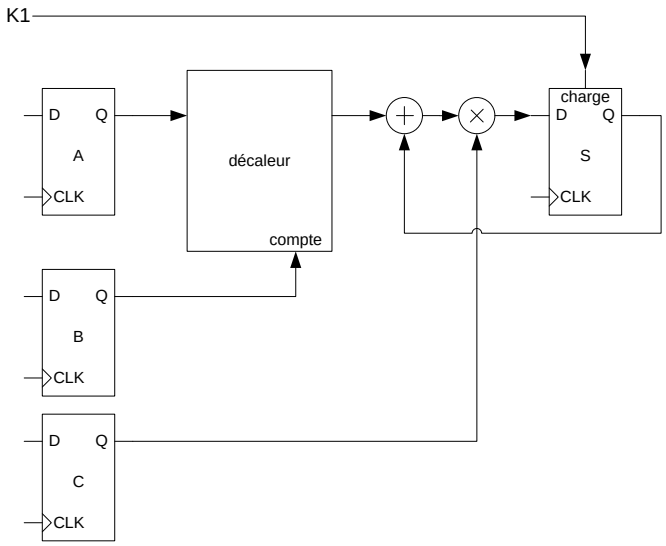
Question 3. (2 points)

Donnez un diagramme d'un chemin des données implémentant les micro-opérations suivantes.

a. K1: S ← (S + sll A, B) * C;

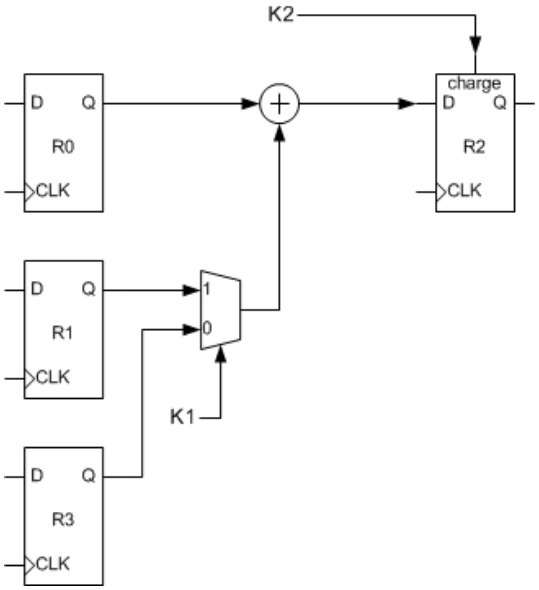
(où sll X, Y veut dire : faire un décalage logique vers la gauche du registre X du nombre de bits spécifié par le registre Y)

Une solution possible :



b. K2K1: R2 ← R0 + R1, K2K1': R2 ← R0 + R3

Une solution possible :



Question 4. (2 points)

Considérez le code VHDL suivant.

Expliquez les étapes suivies par un synthétiseur pour interpréter le code et inférer les structures matérielles nécessaires pour réaliser la fonction décrite.

```
library ieee;
use ieee.std_logic_1164.all;
entity decodeur38 is
   port(
      A : in std_logic_vector(2 downto 0);
      F: out std_logic_vector(7 downto 0)
   );
end decodeur38;
architecture flotDeDonnees of decodeur38 is
begin
   with A select F <=
      "00000001" when "000",
      "00000010" when "001",
      "00000100" when "010",
      "00001000" when "011",
      "00010000" when "100",
      "00100000" when "101",
      "01000000" when "110",
      "10000000" when "111",
      (others => 'X') when others;
end flotDeDonnees;
```

Solution:

1. Le synthétiseur débute par une analyse des entrées et sorties du module. Il y a 3 entrées et 8 sorties. Chaque sortie peut être considérée comme une fonction indépendante.

2. L'énoncé « with-select » spécifie la valeur de chacune des sorties pour toutes les combinaisons possibles d'entrées. La table de vérité de chacune des fonctions de sortie est établie.

3. À partir de la table de vérité, une équation réduite est produite pour chaque fonction.

4. On doit spécifier au synthétiseur les blocs matériels disponibles en spécifiant par exemple la famille de FPGA utilisé. À partir de cette librairie de blocs, chaque fonction est spécifiée en termes de ces blocs et des interconnexions entre eux.

Note : L'assignation (others => 'X') n'est pas considérée par le synthétiseur, cet énoncé n'est pas synthétisable.

Question 5. (2 points) Réponses brèves.

a. (0.5 point) Donnez un diagramme simplifié du flot de conception que vous avez suivi dans les laboratoires du cours à ce jour.

b. (0.5 point) Énumérez quatre (4) éléments fondamentaux qu'on peut retrouver sur un FPGA présentement en vente sur le marché.

_____

_____

_____

_____

c. (0.5 point) Lequel des énoncés suivants est vrai concernant des entités et architectures en VHDL?

i. un module peu avoir plusieurs déclarations d'entité mais une seule architecture;

ii. un module peut avoir seulement une déclaration d'entité mais plusieurs architectures;

iii. un module peut avoir seulement une déclaration d'entité et une seule architecture;

iv. un module peut avoir plusieurs déclarations d'entité et plusieurs architectures.

d. (0.5 point) Lequel/lesquels des énoncés suivants est/sont vrai(s) concernant VHDL?

i. Les processus d'une architecture s'exécutent de façon concurrente.

ii. À l'intérieur d'un processus, les assignations à des signaux sont faites de façon concurrente.

iii. À l'intérieur d'un processus, les assignations à des variables sont faites de façon séquentielle.

iv. Dans une boucle ne comportant pas d'énoncé `wait`, chaque itération prend effectivement un temps nul.

v. Dans une architecture, une assignation concurrente à un signal est une façon compacte d'écrire un processus.

Solutions

a. fig. 4.1 des notes de cours simplifiée : code VHDL et schémas -> vérification par simulation -> synthèse -> implémentation -> génération de fichier de configuration et programmation du FPGA

b. Blocs de logique programmable (CLB), blocs d'entrées-sorties (IOB), réseau d'interconnexions, circuits de routage rapide des retenues, blocs de mémoire intégrée, blocs de fonctions arithmétiques avancées (multiplicateurs, multiplicateurs-accumulateurs), microprocesseurs fixes, circuits de génération et de distribution d'horloge. Éléments moins fondamentaux : LUT, flip-flop, multiplexeur.

c. ii. Une entité, plusieurs architectures

d. i, iii, iv, v.

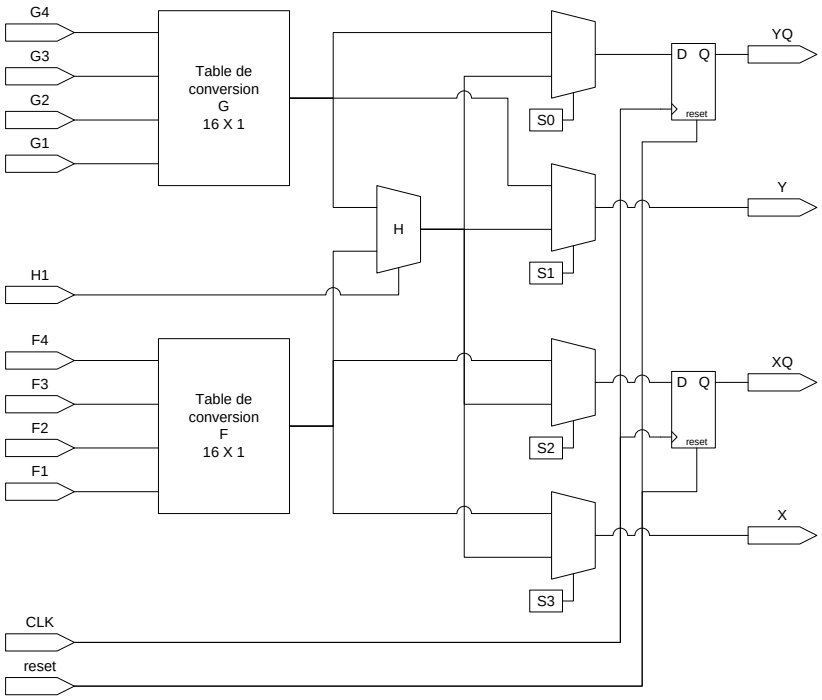
Question 6. (4 points)

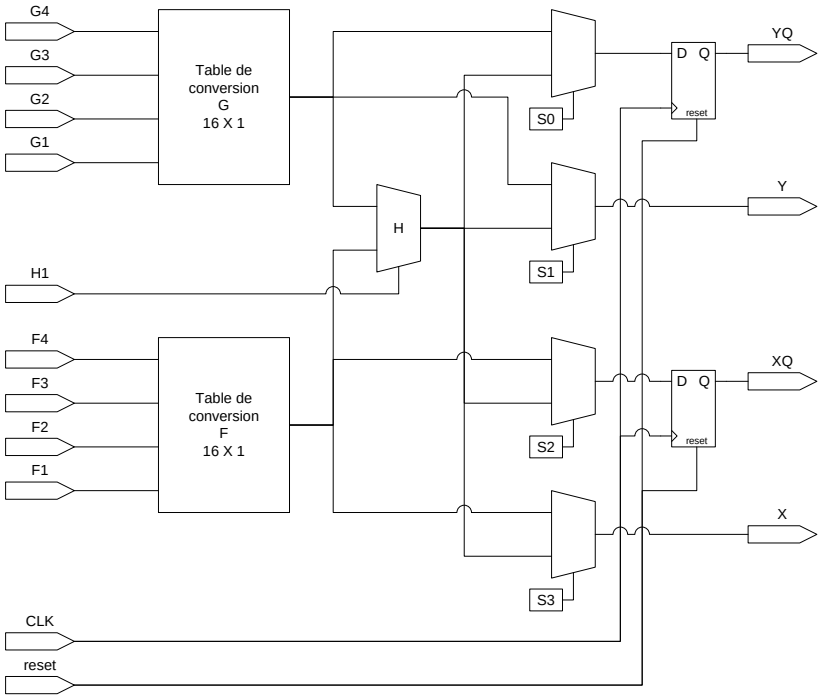
Considérez le code VHDL suivant. Montrez, sur le diagramme fourni, un résultat possible de la synthèse et de l'implémentation de ce code sur un FPGA Virtex II Pro. Indiquez directement sur le dessin où chaque signal se situe ainsi que les interconnections entre les blocs. Indiquez dans les tables de vérité fournies le contenu de chacune des tables de conversion que vous utilisez.

```
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 arch of monModule5 is
signal etat : std_logic_vector(1 downto 0);
begin

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

   sortie(0) <= '0' when (etat = "01" or etat = "10") else '1';

end arch;
```

Question 6

Nom : _____ Matricule : _____





Question 6

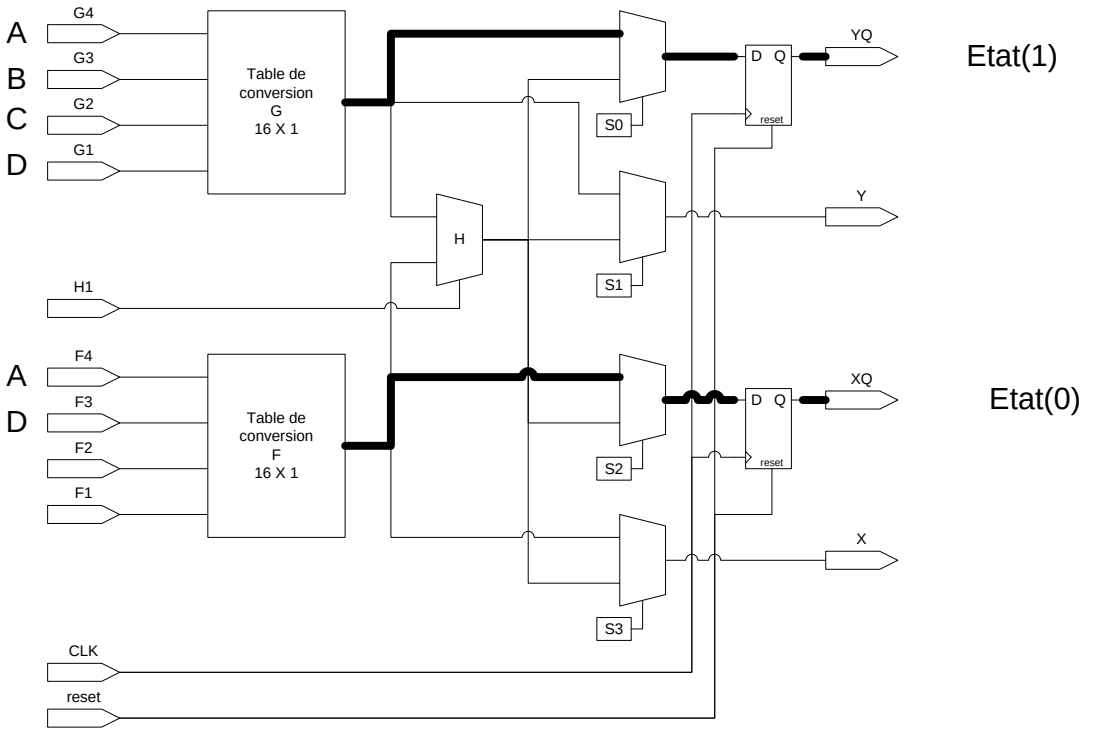
G4 ( )	G3 ( )	G2 ( )	G1 ( )	G ( )
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

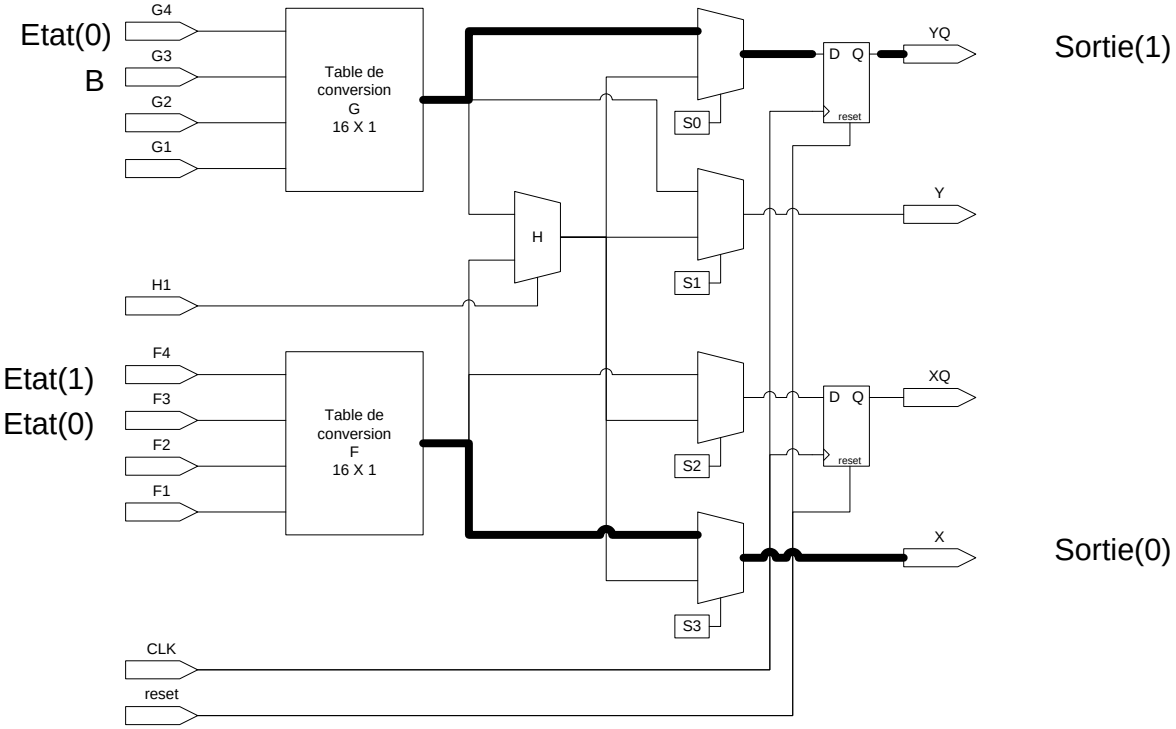
F4 ( )	F3 ( )	F2 ( )	F1 ( )	F ( )
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

G4 ( )	G3 ( )	G2 ( )	G1 ( )	G ( )
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

F4 ( )	F3 ( )	F2 ( )	F1 ( )	F ( )
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Solution


A
B
C
D
G4
G3
G2
G1
Table de conversion G 16 X 1
H
H1
A
D
F4
F3
F2
F1
Table de conversion F 16 X 1
S0
S1
S2
S3
D Q
reset
YQ
Y
XQ
X
CLK
reset
Etat(1)
Etat(0)


Etat(0)
B
Etat(1)
Etat(0)
G4
G3
G2
G1
Table de conversion G 16 X 1
H
H1
F4
F3
F2
F1
Table de conversion F 16 X 1
S0
S1
S2
S3
D Q
reset
YQ
Y
XQ
X
CLK
reset
Sortie(1)
Sortie(0)

Solution

G4 ( )	G3 ( )	G2 ( )	G1 ( )	G ( )
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

F4 ( )	F3 ( )	F2 ( )	F1 ( )	F ( )
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
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	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

G4 ( )	G3 ( )	G2 ( )	G1 ( )	G ( )
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
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	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

F4 ( )	F3 ( )	F2 ( )	F1 ( )	F ( )
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	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1