INF3500 : Conception et réalisation de systèmes numériques

Examen intra

Mercredi 20 février 2013

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.
- Remettre le questionnaire avec votre réponse à la question 2.
- Ne posez pas de question durant l'examen. En cas de doute sur le sens d’une question, énoncez clairement toute supposition que vous faites.

---

**Question 1. (4 points)**

Considérez le code VHDL suivant décrivant un circuit combinatoire:

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

entity question_1 is
port(
   entree : in signed(9 downto 0);
   sortie : out signed(9 downto 0)
);
end question_1;

architecture arch of question_1 is


   -- déclaration de fonction
   function gen_response(int_value : integer) return signed is
      constant cst_1 : integer :=  7;
      constant cst_2 : integer :=  9;
      constant cst_3 : integer := 16;
      constant w : positive := 10;
   begin
      return to_signed( (cst_1 * int_value + cst_2 * int_value)
         mod
         cst_3,
         w);
   end function gen_response;

   -- déclaration de signal
   signal a : signed(9 downto 0);

begin

   -- process combinatoire
   process( entree, a ) is
      variable v : signed(9 downto 0);
   begin

      a <= gen_response( to_integer(entree) );
      v := gen_response( to_integer(a) );
      sortie <= a + v;

   end process;

end arch;
```

a) Quelle est la valeur prise par le signal `sortie` lorsque `entree` vaut `"1000010000"`. Rép. `"0000000000"`

b) Quelle est la valeur prise par le signal `sortie` lorsque `entree` vaut `"0100000101"`.Rép. `"0000000000"`

c) Pour quelle(s) valeur du signal `entree`, `sortie` subit-elle un débordement suite à l'addition de `a` et `v` ?

Aucune. a = v = 0.

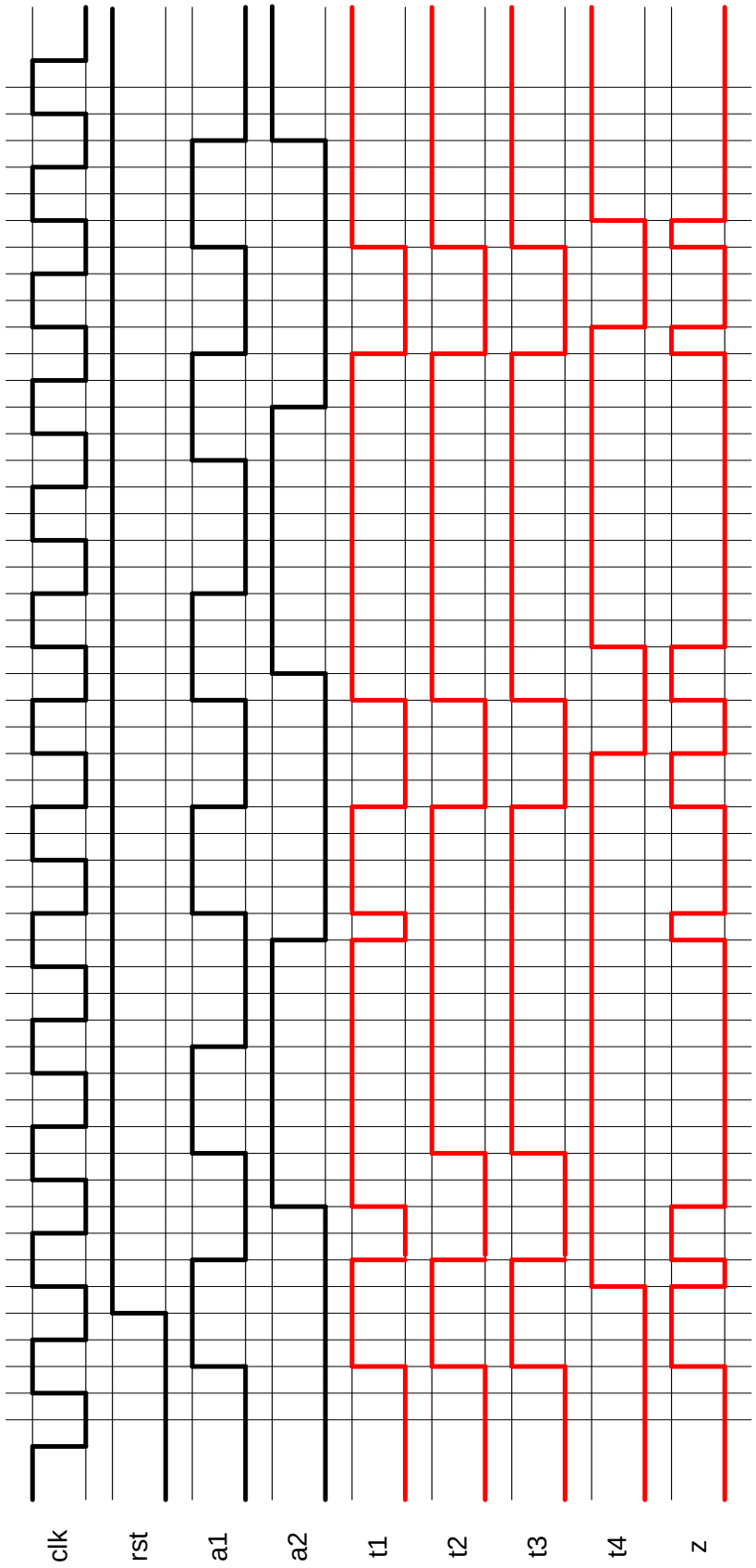
d) Ce code VHDL est-il synthétisable ? Justifiez votre réponse.

Le code est synthétisable. Il s'agit d'un circuit combinatoire impliquant addition, multiplication et modulo sur une puissance de 2. De plus, le synthétiseur devrait se rendre compte que la sortie vaut toujours 0.

**Question 2. (4 points)**

Considérez le code VHDL suivant. Complétez le chronogramme issu de cette description matérielle.

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

entity question_2 is
port(
   rst_n : in std_logic;
   clk : in std_logic;
   a1, a2 : in std_logic;
   z : out std_logic
);
end question_2;

architecture arch of question_2 is
   -- déclaration de signaux
   signal t1, t2, t3, t4 : std_logic;
begin
   -- process 1
   process( a1, a2 ) is
   begin
      t1 <= a1 or a2;
   end process;

   -- process 2
   process( a1 ) is
   begin
      t2 <= a1 or a2;
   end process;

   -- process 3
   process( a1 ) is
      variable v : std_logic;
   begin
      v := a1;
      t3 <= a1 or a2;
   end process;

   -- process 4
   process( rst_n, clk) is
   begin
      if( rst_n = '0' ) then
         t4 <= '0';
      elsif( rising_edge( clk ) ) then
         t4 <= a1 or a2;
      end if;
   end process;

   -- sortie
   z <= t1 xor t2 xor t3 xor t4;

end arch;
```


clk
rst
a1
a2
t1
t2
t3
t4
z

**Question 3. (4 points)**

Considérez le code VHDL suivant décrivant un circuit séquentiel:

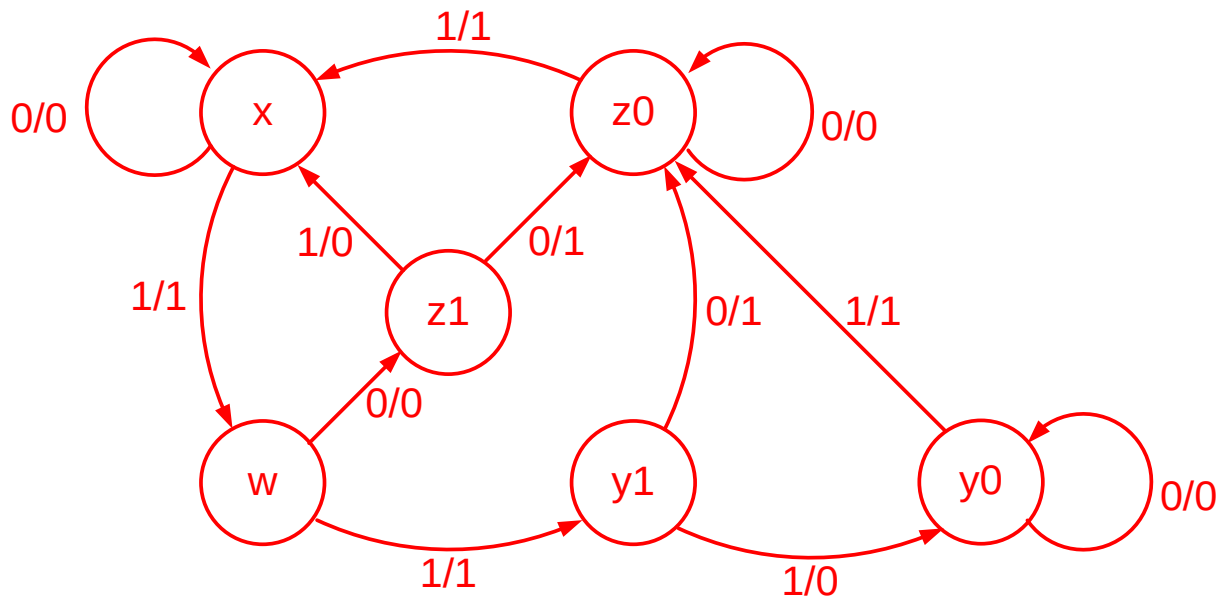
```
library ieee;
use ieee.std_logic_1164.all;

entity question_3 is
port (
   reset  : in  std_logic;
   clk    : in  std_logic;
   entree : in  std_logic;
   sortie : out std_logic
);
end question_3;

architecture arch of question_3 is

   type etats is (x, y, z, w);
   signal etat : etats := x;
   signal t_sortie : std_logic := '0';

begin

   process(clk, reset) is
   begin
      if reset = '0' then
         etat <= x;
         t_sortie <= '0';
      elsif rising_edge(clk) then
         case etat is
            when x =>
               if ( entree = '1' ) then
                  etat <= w;
               end if;
            when y =>
               if ( entree = '0' ) then
                  etat <= z;
               end if;
               t_sortie <= '0';
            when z =>
               if ( entree = '1' ) then
                  etat <= x;
               end if;
               t_sortie <= '0';
            when others => -- w
               if ( entree = '0' ) then
                  etat <= z;
               else
                  etat <= y;
               end if;
               t_sortie <= '1';
         end case;
      end if;
   end process;

   sortie <= entree xor t_sortie;

end arch;
```

a) S'agit-il d'une machine de Moore ou d'une machine de Mealy ? Justifiez votre réponse.

Il s'agit d'une machine de Mealy puisque la sortie dépend de l'entrée.

b) Dessinez le diagramme d'états de la machine à états.

Ce circuit peut s'interpréter comme un machine à états à 4 états auquel est adjoint un circuit séquentiel constitué d'une bascule pour entreposer l'état du signal `t_sortie`. En supposant que les états de la première machine sont encodés sur 2 bits, alors la machine à états totale (la machine de Mealy qui nous intéresse) possède jusqu'à 8 états. Une analyse du comportement de cette dernière indique que seuls 6 états de la machine comptent (les autres états étant redondants). Le diagramme suivant illustre le comportement de cette machine de Mealy:




c) Partant de l'état `x`, quel sera la sortie `sortie` de la machine après 3 fronts montant de l'horloge si on maintient l'entrée `entree` à `'1'`. Quel sera l'état à ce moment là?

La sortie sera à 1. On sera à y.

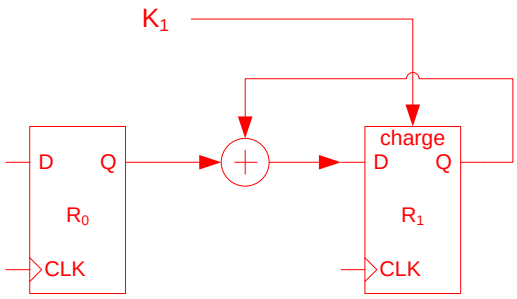
d) Partant de l'état `x`, quel sera la sortie `sortie` de la machine après 3 fronts montant de l'horloge si on maintient l'entrée `entree` à `'0'`. Quel sera l'état à ce moment là?

La sortie sera à 0. On sera à x.

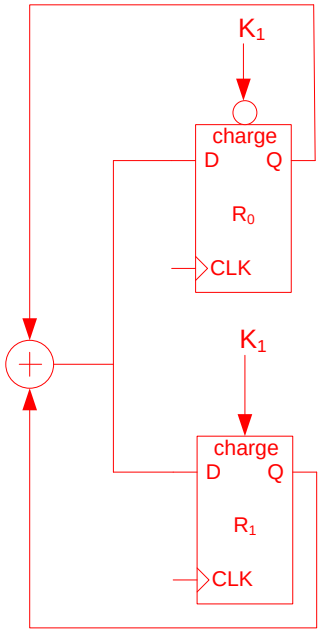
**Question 4. (4 points)**

En utilisant des registres, des multiplexeurs, décodeurs et autres circuits combinatoires vus en classe, dessinez les chemins de données permettant de réaliser chacune des micro-opérations suivantes, où $R_0$, $R_1$ et $R_2$ sont des registres et $K_1$, $K_2$ sont des signaux de contrôle :

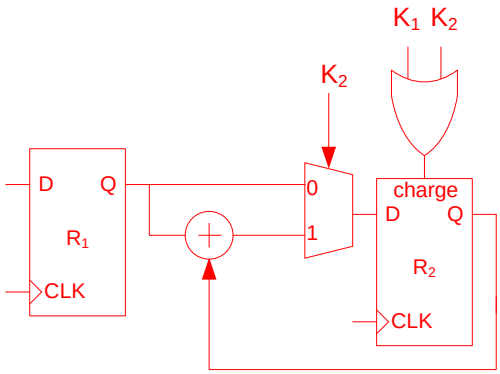
a) $K_1$: $R_1 \leftarrow R_1+R_0$



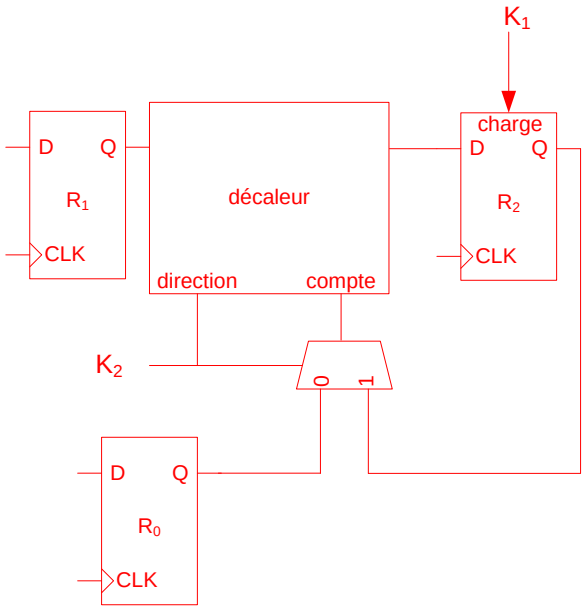

b) $K_1$: $R_1 \leftarrow R_1+R_0$ ; $K_1$': $R_0 \leftarrow R_1+R_0$



c) $K_1K_2'$: $R_2 \leftarrow R_1$ ; $K_2$: $R_2 \leftarrow R_1+R_2$




d) $K_1K_2$: $R_2 \leftarrow$ sll $R_1$, $R_2$ ; $K_1K_2$': $R_2 \leftarrow$ slr $R_1$, $R_0$



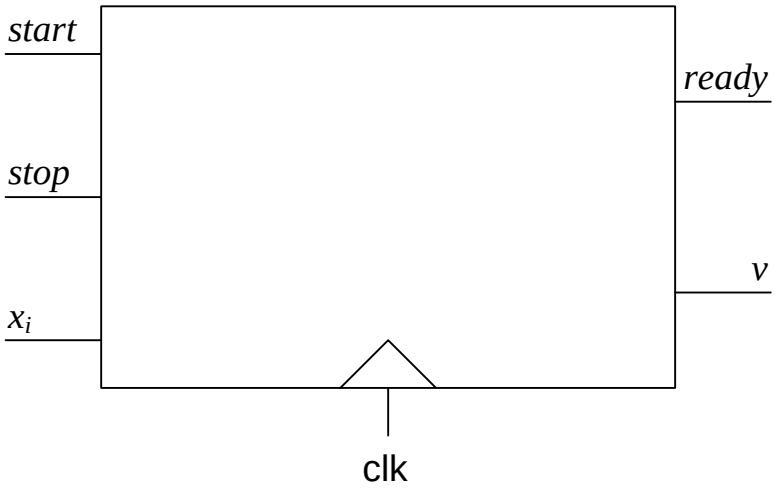
**Question 5. (4 points)**

On désire concevoir un circuit séquentiel qui prend N entrées binaires $x_1$, $x_2$, $x_3$, ..., $x_N$ et qui calcule la fonction logique $v = (x_1 + x_2 + x_3 + ... + x_N)$, qu'il convient de lire comme $v$ est la disjonction (OR) des entrées $x_1$ à $x_N$.

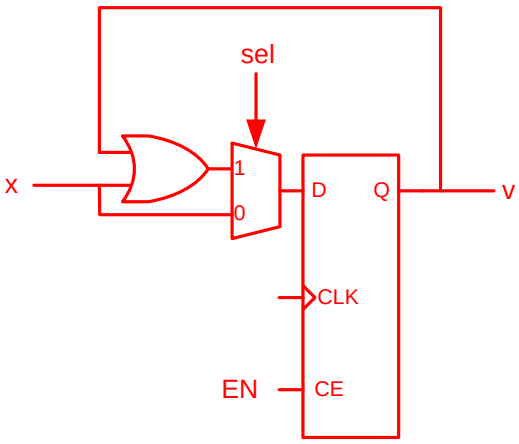
Les entrées $x_i$ arrivent de manière sérielle. Au début de la séquence, qui coïncide avec l'arrivée de $x_1$, le signal d'entrée *start* prend la valeur 1. À la fin de la séquence, qui coïncide avec l'arrivée de l'entrée $x_N$, le signal *stop* prend la valeur 1. Les signaux *start* et *stop* prennent la valeur 1 durant un cycle d'horloge seulement durant une séquence d'entrée $x_i$. Vous êtes assurés d'avoir au moins 2 éléments durant une séquence d'entrée ($N \geq 2$).

Le résultat de l'opération $v$ doit apparaître à la sortie en même temps que l'on met la sortie *ready* à 1. Le signal *ready* demeure à 1 tant qu'une nouvelle séquence de traitement n'a pas débuté. Il vaut 0 quand le circuit traite une nouvelle séquence.

Le schéma suivant devrait vous aider à saisir les ports d'entrées/sorties de l'entité que nous considérons.




a) Proposez un chemin de données permettant de réaliser ce circuit.



b) Décrivez en VHDL ce circuit séquentiel en exploitant le chemin de données proposé à la question 5.a.

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

entity question_5 is
port (
   reset : in  std_logic;
   clk   : in  std_logic;
   start : in  std_logic;
   stop  : in  std_logic;
   x     : in  std_logic;
   ready : out std_logic;
   v     : out std_logic
);
end question_5;

architecture arch of question_5 is
   signal rdy : boolean := true;
   signal sel : std_logic;
   signal en  : std_logic;
   signal q   : std_logic := '0';
begin
   process(clk, reset) is
   begin
      if reset = '1' then
         rdy <= true;
      elsif rising_edge(clk) then
         case rdy is
            when true =>
               if( start = '1' ) then
                  rdy <= false;
               end if;
            when false =>
               if( stop = '1' ) then
                  rdy <= true;
               end if;
         end case;
      end if;
   end process;

   sel <= '0' when rdy else '1';
   en <= '0' when ( rdy and start = '0' ) else '1';
   ready <= '1' when rdy else '0';

   process(clk, reset) is
   begin
      if reset = '1' then
         q <= '0';
      elsif rising_edge(clk) then
         if( en = '1' ) then
            if( sel = '0' ) then
               q <= x;
            else
               q <= q or x;
            end if;
         end if;
      end if;
   end process;

   v <= q;

end arch;
```

Cette solution reprenait dans le détail le chemin de donnée et y ajoutait un contrôle. Voici une façon plus conventionnelle de faire la même chose où les signaux `sel` et `en` sont inférés de la description HDL.

```
architecture arch2 of question_5 is
   signal rdy : boolean := true;
   signal q   : std_logic := '0';
begin

   process(clk, reset) is
   begin
      if reset = '1' then
         rdy <= true;
      elsif rising_edge(clk) then
         case rdy is
            when true =>
               if( start = '1' ) then
                  rdy <= false;
                  q <= x;
               end if;
            when false =>
               if( stop = '1' ) then
                  rdy <= true;
               end if;
               q <= x or q;
         end case;
      end if;
   end process;

   ready <= '1' when rdy else '0';
   v <= q;

end arch2;
```