INF3500 : Conception et réalisation de systèmes numériques

Examen final

7 décembre 2008, 13h30-16h00

Durée: 2h30.
Pondération: 40%.
Documentation: Une feuille recto-verso 8.5”×11” ou A4 manuscrite permise.
Calculatrice: Programmable permise.
Directives particulières: Ordinateurs interdits, répondre à toutes les questions, la valeur de chaque question est indiquée.
Ne posez pas de questions. En cas de doute sur le sens d’une question, énoncez clairement toute supposition que vous faites.

Question 1. (8 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 10. 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 10, 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

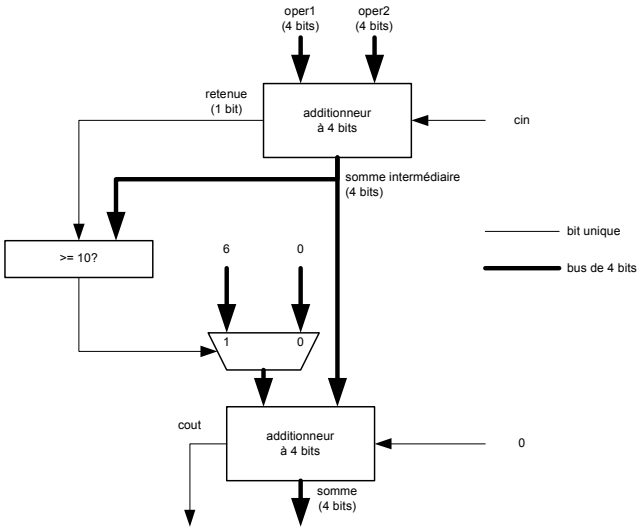
a. (4 points) Donnez un diagramme d'un circuit combinatoire pour un additionneur BCD à un chiffre. Vous pouvez utiliser des opérations logiques et arithmétiques (mais pas la division ni le modulo), la comparaison, des multiplexeurs, décodeurs et encodeurs, et toute autre porte logique de base. Identifiez 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.

b. (2 points) En utilisant des instances d'un module additionneur BCD à un chiffre développé à la partie a., donnez un diagramme d'un circuit combinatoire pour un additionneur BCD à 8 chiffres. Vous pouvez supposer que les additionneurs à 4 bits de votre module peuvent accepter une retenue en entrée en plus de produire une retenue de sortie. 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.

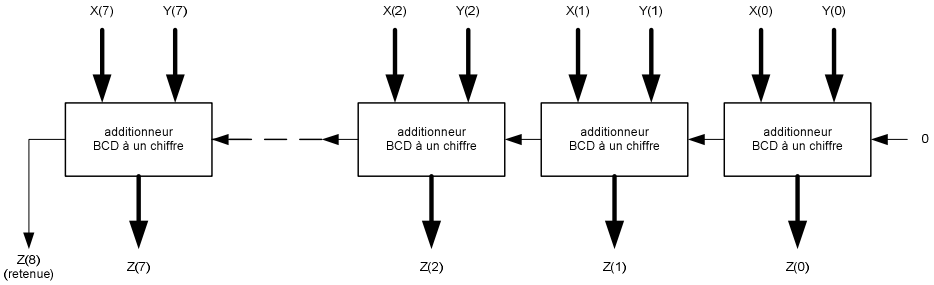
c. (2 points) En utilisant une seule instance du module additionneur BCD à un chiffre développé à la partie a., donnez un circuit séquentiel itératif pour un additionneur BCD à 8 chiffres. Vous pouvez supposer que les additionneurs à 4 bits de votre module peuvent accepter une retenue en entrée en plus de produire une retenue de sortie.

Solution :

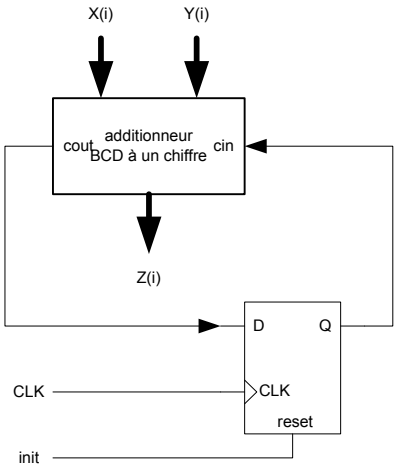
a.



b.



c.



Les entrées X et Y sont présentées un chiffre à la fois, en série. La somme Z est disponible elle aussi un chiffre à la fois en série.

Dans le diagramme, les traits gras représentent des bus de 4 bits; les autres traits sont des signaux de 1 bit.

Le signal init doit être activé avant le calcul d'une nouvelle somme.

La question spécifie des nombres de 8 chiffres, mais le circuit montré ici fonctionnerait pour toutes les largeurs de nombres.

Question 2. (8 points)

Considérez le problème de la vérification d'un circuit combinatoire qui doit rencontrer les spécifications de la question 1a.

Complétez le squelette de banc d'essai suivant en VHDL en incluant une génération algorithmique de vecteurs de tests ainsi qu'une évaluation automatisée des réponses du circuit. Vous devez effectuer un test exhaustif, et la simulation doit s'arrêter dès qu'une erreur survient ainsi que lorsque tous les vecteurs de test ont été appliqués. Vous pouvez supposer que seulement des entrées BCD valides doivent être appliquées au circuit. Indiquez clairement dans quelle partie du code existant votre code doit être placé.

Dans l'entité à vérifier, les ports `oper1` et `oper2` sont des chiffres BCD exprimés sur quatre bits. Le port `somme` est un chiffre BCD exprimé sur quatre bits, représentant la somme. Le port `cout` représente la retenue, et occupe un bit unique.

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

entity addbcdTB is
end addbcdTB;

architecture tbexhaustive1 of addbcdTB is
   component addbcd
   port(
      oper1, oper2 : in UNSIGNED(3 downto 0);
      somme : out UNSIGNED(3 downto 0);
      cout : out std_logic );
   end component;
   signal oper1, oper2, somme : UNSIGNED(3 downto 0);
   signal cout : std_logic;

   -- vous pouvez ajouter du code ici si nécessaire ...

begin

   UUT : addbcd port map (oper1, oper2, somme, cout);

   -- vous pouvez ajouter du code ici si nécessaire ...

end tbexhaustive1;
```

Solution :

```
architecture tbexhaustive1 of addbcdTB is
   component addbcd
   port(
      oper1, oper2 : in UNSIGNED(3 downto 0);
      somme : out UNSIGNED(3 downto 0);
      cout : out std_logic );
   end component;

   signal oper1, oper2, somme : UNSIGNED(3 downto 0);
   signal cout : std_logic;

begin

   UUT : addbcd port map (oper1, oper2, somme, cout);

   process
   variable lasomme : integer := -1;
   begin

      for k1 in 0 to 9 loop
         for k2 in 0 to 9 loop
            oper1 <= to_unsigned(k1, 4);
            oper2 <= to_unsigned(k2, 4);

            wait for 10 ns;

            if cout = '1' then
               lasomme := 10 + to_integer(somme);
            else
               lasomme := to_integer(somme);
            end if;

            assert lasomme = k1 + k2 report "erreur!" severity failure;

         end loop;
      end loop;

      report "simulation terminée!" severity failure;

   end process;

end tbexhaustive1;
```

Question 3. (8 points)

Un processeur spécialisé doit être conçu pour calculer la moyenne arithmétique d'une plage de cellules dans une mémoire RAM de 1024 cellules de 8 bits chacune. Les données en mémoire sont exprimées en complément à deux. La moyenne arithmétique est donnée par la somme des valeurs des cellules, divisée par le nombre de cellules. Avant d'effectuer le calcul, le processeur doit d'abord recevoir, sur deux ports distincts et simultanément, l'adresse de la première cellule de la plage ainsi que le logarithme en base 2 du nombre de cellules entrant dans le calcul de la moyenne. Le nombre de cellules est donc toujours une puissance positive de 2. Le chargement est effectué quand un signal de contrôle spécial 'go' est activé, puis le processeur débute ses opérations.

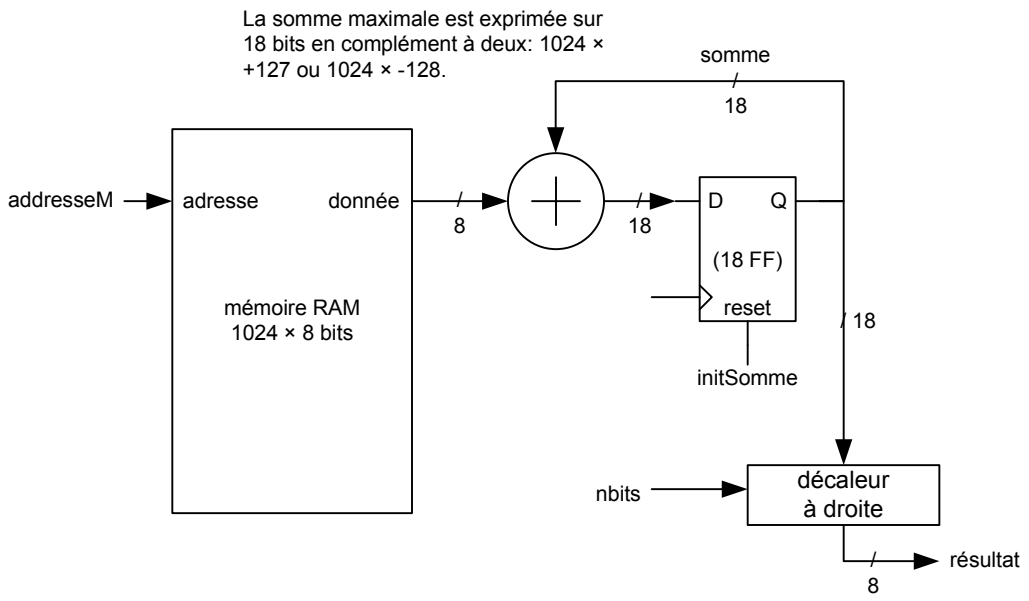
Le processeur doit effectuer le calcul, puis activer un signal spécial lorsque le résultat est appliqué à son port de sortie. Il doit aussi activer un signal d'erreur spécial à la sortie si la cellule de départ et le nombre de cellules spécifiés sont tels que la plage spécifiée dépasse la dernière cellule de la mémoire. La moyenne doit être exprimée sur 8 bits.

a. (4 points) Donnez un diagramme montrant le chemin des données de ce processeur. Identifiez bien chaque bloc et indiquez la largeur en bits de tous les ports et de tous les signaux.

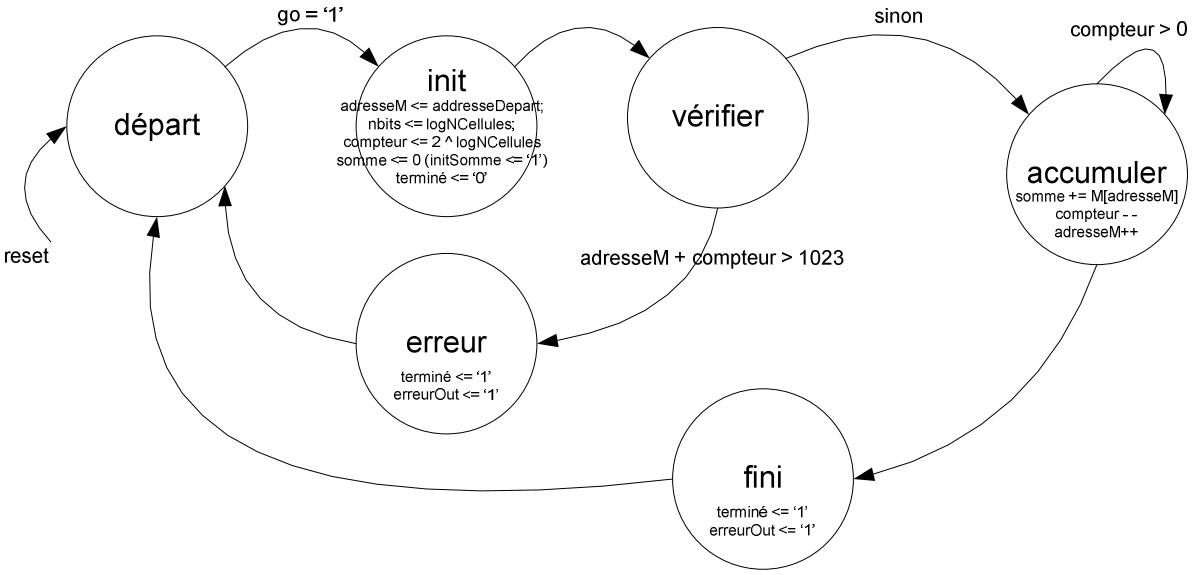
b. (4 points) Spécifiez l'unité de contrôle de ce processeur par un diagramme d'états. Identifiez bien les états et les conditions pour les transitions. Identifiez les signaux de contrôle appliqués au chemin des données.

Solution :

a.


La somme maximale est exprimée sur 18 bits en complément à deux: 1024 × +127 ou 1024 × -128.
somme
18
addresseM
adresse
donnée
8
18
D
Q
(18 FF)
reset
initSomme
mémoire RAM
1024 × 8 bits
18
nbits
décaleur à droite
résultat
8

b.


go = '1'
sinon
compteur > 0
départ
init
adresseM <= addresseDepart;
nbits <= logNCellules;
compteur <= 2 ^ logNCellules
somme <= 0 (initSomme <= '1')
terminé <= '0'
vérifier
accumuler
somme += M[adresseM]
compteur - -
adresseM++
reset
adresseM + compteur > 1023
erreur
terminé <= '1'
erreurOut <= '1'
fini
terminé <= '1'
erreurOut <= '1'

Question 4. (4 points)

Considérez le code VHDL suivant pour un arbitre de bus. L'arbitre permet à trois processeurs de partager un bus unique. Quand un processeur veut avoir accès au bus, il active un des trois signaux de contrôle `r` (pour « requête »). Pour informer un des processeurs qu'il peut se servir du bus, l'arbitre l'informe grâce à l'un des trois signaux de contrôle `g` (pour « go »). Quand le processeur correspondant a terminé du bus, il en informe l'arbitre de bus en désactivant son signal de requête `r`.

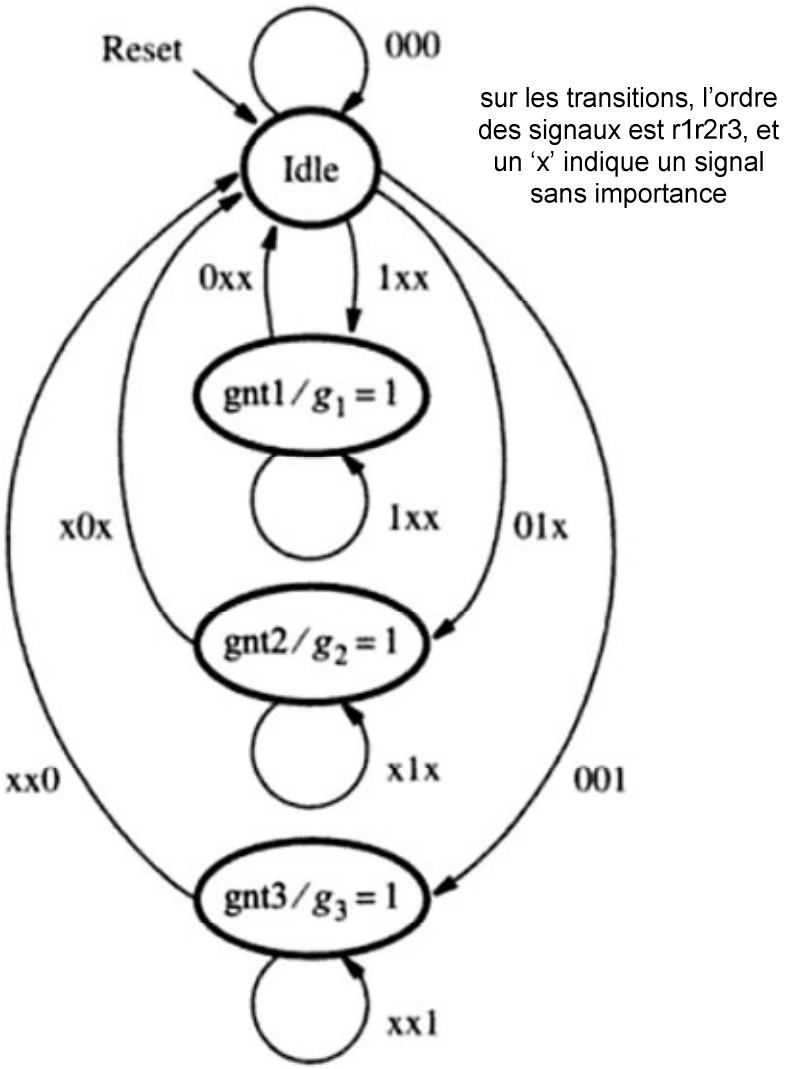
Donnez un diagramme d'états illustrant le fonctionnement de ce module. Identifiez clairement chaque état, les transitions entre les états, et les conditions de transition entre les états.

```
-- code tiré de S. Brown & Z. Vranesic,
-- Fundamentals of Digital Logic, McGraw-Hill, 2005.
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY arbiter IS
   PORT (    Clock, Resetn       : IN   STD_LOGIC ;
             r                : IN   STD_LOGIC_VECTOR(1 TO 3) ;
             g                : OUT  STD_LOGIC_VECTOR(1 TO 3) ) ;
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
   TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
   SIGNAL y : State_type ;
BEGIN
   PROCESS ( Resetn, Clock )
   BEGIN
      IF Resetn = '0' THEN y <= Idle ;
      ELSIF (Clock'EVENT AND Clock = '1') THEN
         CASE y IS
            WHEN Idle =>
               IF r(1) = '1' THEN y <= gnt1 ;
               ELSIF r(2) = '1' THEN y <= gnt2 ;
               ELSIF r(3) = '1' THEN y <= gnt3 ;
               ELSE y <= Idle ;
               END IF ;
            WHEN gnt1 =>
               IF r(1) = '1' THEN y <= gnt1 ;
               ELSE y <= Idle ;
               END IF ;
            WHEN gnt2 =>
               IF r(2) = '1' THEN y <= gnt2 ;
               ELSE y <= Idle ;
               END IF ;
            WHEN gnt3 =>
               IF r(3) = '1' THEN y <= gnt3 ;
               ELSE y <= Idle ;
               END IF ;
         END CASE ;
      END IF ;
   END PROCESS ;
   g(1) <= '1' WHEN y = gnt1 ELSE '0' ;
   g(2) <= '1' WHEN y = gnt2 ELSE '0' ;
   g(3) <= '1' WHEN y = gnt3 ELSE '0' ;
END Behavior ;
```

Solution :



(tiré de S. Brown & Z. Vranesic, Fundamentals of Digital Logic, McGraw-Hill, 2005.)

Question 5. (4 points)

Questions à développement.

a. (1 point) Énumérez quatre (4) éléments fondamentaux qu'on peut retrouver sur un FPGA présentement en vente sur le marché.

b. (1 point) Expliquez, à l'aide d'un diagramme, le principe de la programmation d'un dispositif programmable EEPROM ou Flash par grille flottante.

c. (1 point) Expliquez en deux ou trois phrases chacune les étapes de synthèse, d'association (*mapping*), de placement et de routage.

d. (1 point) Considérez l'énoncé suivant : « Pour un circuit numérique implémenté sur FPGA, il n'y a pas de désavantage à utiliser le principe du pipeline au maximum, parce que les bascules sont à toutes fins pratiques gratuites. » Dites si vous êtes en accord ou en désaccord avec cet énoncé, et expliquez en quelques lignes pourquoi.

Solution :

a. Pour 0.25 point chacun:

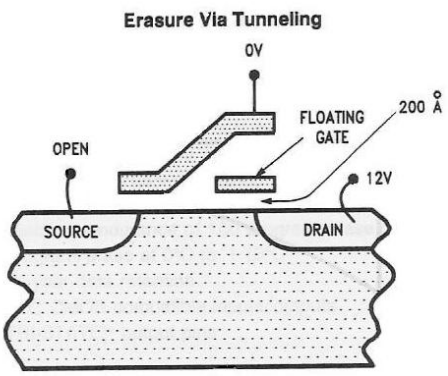
blocs de logique programmable (CLB), blocs d'entrées-sorties (IOB), réseau d'interconnexions, circuits de routage rapide des retenues, blocs de mémoire intégrée, blocs de fonctions arithmétiques avancées (multiplicateurs, multiplicateurs-accumulateurs), microprocesseurs fixes, circuits de génération et de distribution d'horloge

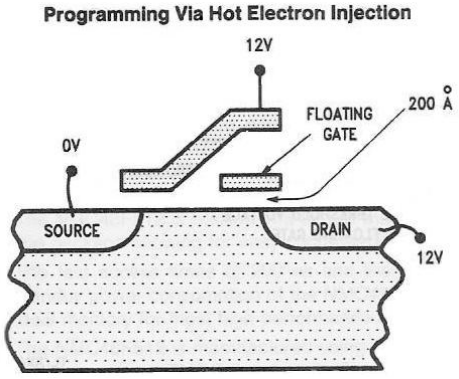
Pour 0.125 point chacun :

LUT, flip-flop, multiplexeur

b. Dans son état normal, la grille flottante est déchargée et le transistor fonctionne normalement, c'est-à-dire qu'une tension appliquée à la grille du transistor induit un canal et permet au courant de passer entre la source et le drain du transistor. Pour 'programmer' la cellule, on place une tension élevée sur la grille de contrôle et le drain du transistor, comme montré dans la figure. Comme un courant élevé circule dans le canal, des électrons sont attirés par la grille de contrôle et vont s'emmagasiner sur la grille flottante. La tension négative de la grille flottante est alors suffisante pour neutraliser toute tension normale appliquée à la grille de contrôle, et le transistor ne conduit plus. Le transistor peut conserver cet état programmé plusieurs années.

Pour 'déprogrammer' le transistor, on applique une tension élevée uniquement au drain de celui-ci, comme montré ici. La tension est suffisante pour attirer les électrons emmagasinés sur la grille flottante, à travers de l'isolant. Le transistor fonctionne alors normalement.





c. Le synthétiseur produit une description du circuit en termes d'éléments logiques simples à partir d'une description dans un langage de description matérielle comme VHDL. Le synthétiseur indique aussi les interconnexions entre ces composantes. Le produit du synthétiseur est communément appelé « liste des interconnexions (*netlist*)».

Le processus d'association (*mapping*) consiste à associer des composantes de base à des blocs ou des groupes de blocs logiques du FPGA. Par exemple, un groupe de portes logiques peut être combiné en une seule table de conversion d'un bloc logique.

Le placement consiste à disposer les composantes du circuit en rangées et en colonnes. Ce placement est souvent effectué de façon à respecter certaines contraintes de temps et/ou d'espace imposées par l'utilisateur.

Le routage consiste à choisir les chemins suivis par les fils d'interconnexions entre les composantes du circuit. Cette étape est soumise aussi à des contraintes, habituellement de temps.

d. Plutôt en accord. Une fois qu'on a choisi un FPGA en particulier, il n'y a pas de coût associé à l'utilisation de chacune des ressources de la puce. Si le débit d'information doit être maximisé, alors le fait de pipeliner autant que possible devrait être encouragé, en utilisant toutes les bascules disponibles. Cependant, ajouter un grand nombre d'étages de pipeline augmente la latence du circuit, ce qui pourrait aller à l'encontre des spécifications.

Question 6. (8 points)

Une pile est une structure de données à laquelle on ne peut ajouter des éléments que sur le dessus, et de laquelle on ne peut enlever des éléments que du dessus aussi. Une pile est donc une structure de la forme DAPS: Dernier Arrivé, Premier Servi. Quand la pile est pleine, il n'est pas possible d'y rajouter un élément. Quand la pile est vide, il n'est pas possible d'y lire un élément.

Considérez la déclaration d'entité suivante en VHDL pour une pile:

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

entity pile is
   generic (
      N : positive := 3; -- la profondeur (le nombre d'éléments) de la pile
      W : positive := 8 -- la largeur (en bits) de la pile
   );
   port (
      clk : in std_logic;
      reset : in std_logic;      -- Actif haut: un '1' réinitialise la pile.
      din : in std_logic_vector(W - 1 downto 0); -- Données entrant dans la pile.
      dout : out std_logic_vector(W - 1 downto 0); -- Données sortant de la pile.
      wr_en : in std_logic; -- Write-enable: si actif, une donnée sera lue de din et
                            -- placée sur la pile au prochain front montant de clk,
                            -- si la pile n'est pas pleine.
                            -- Le signal wr_en a préséance sur le signal rd_en.
      rd_en : in std_logic; -- Read-enable: si actif, la donnée du dessus de la pile
                            -- sera retirée et placée sur dout au prochain front
                            -- montant de clk, si la pile n'est pas vide.
                            -- Quand wr_en est actif, le signal rd_en n'est pas
                            -- considéré.
      empty : out std_logic; -- Indique que la pile est vide.
      full : out std_logic -- Indique que la pile est pleine.
   );
end pile;
```

Donnez une architecture pour cette entité qui respecte les spécifications données dans les commentaires de la déclaration des ports.

Solution :

```
architecture arch of pile is

type memoire is array (0 to N - 1) of std_logic_vector(W - 1 downto 0);
signal lapile : memoire;

begin

   process(clk, reset)
   variable pointeurdepile : natural range 0 to N := 0;
   begin
      if (reset = '1') then
         pointeurdepile := 0;
      elsif (clk'event and clk = '1') then

         if (wr_en = '1') then
            if (pointeurdepile /= N) then
               lapile(pointeurdepile) <= din;
               pointeurdepile := pointeurdepile + 1;
            end if;
         elsif (rd_en = '1') then
            if (pointeurdepile /= 0) then
               pointeurdepile := pointeurdepile - 1;
               dout <= lapile(pointeurdepile);
            end if;
         end if;

      end if;

      if (pointeurdepile = 0) then empty <= '1'; else empty <= '0'; end if;
      if (pointeurdepile = N) then full <= '1'; else full <= '0'; end if;

   end process;

end arch;
```