INF3500 : Conception et réalisation de systèmes numériques

Examen final

Décembre 2013

Durée: 2h30.

Pondération: 40%.

Documentation: Une feuille recto-verso 8.5”×11” ou A4 permise.

Calculatrice: Programmable permise.

Directives particulières:

- 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. (8 points)

La représentation d'un nombre binaire *n* en virgule flottante comporte trois parties : le signe $s = \{0, 1\}$, la mantisse *m* et l'exposant *e*. On a alors $n = (-1)^s \times m \times 2^e$.

Le produit *p* de deux nombres n_1 et n_2 peut être décomposé en trois étapes, soit le produit des signes, le produit des mantisses et la somme des exposants :

$$p = n_1 \times n_2 = (-1)^{s_1} \times (-1)^{s_2} \times m_1 \times m_2 \times 2^{e_1+e_1}$$

Les signes sont représentés avec un seul bit, '0' pour positif et '1' pour négatif. Le produit des signes des nombres correspond à l'opération ou-exclusif de leur bit de signe.

Les exposants sont représentés avec *E* bits en complément à deux.

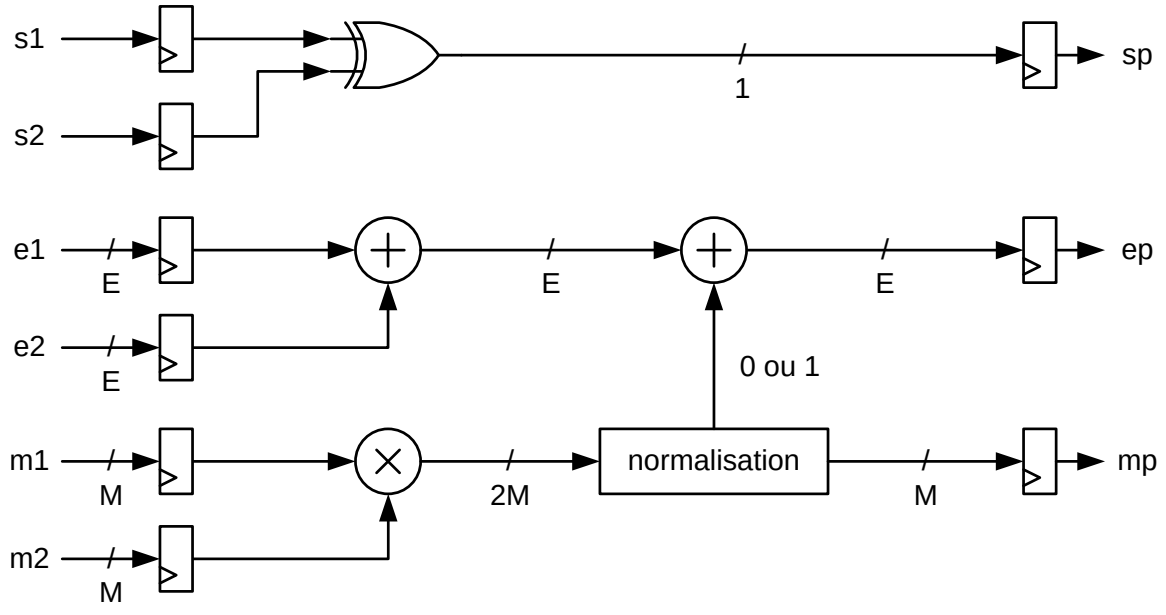
Les mantisses sont des nombres entiers positifs représentés avec *M* bits dans l'intervalle [1, 2[. Leur bit le plus significatif est toujours '1', les autres bits sont fractionnaires. Par exemple, avec $M = 4$, on pourrait avoir $m = 1.011_2 = 1 + 3/8 = 1.375_{10}$. Le produit de deux mantisses doit toujours être gardé dans l'intervalle [1, 2[. Si un produit de mantisses donne un nombre égal ou supérieur à 2, il faut diviser ce produit par deux (donc faire un décalage vers la droite d'une position) et incrémenter l'exposant d'une unité. On appelle ce processus la *normalisation* du produit avant sa représentation finale.

Par exemple, pour $M = 4$, $E = 4$, $n_1 = -1 \times 1.375 \times 2^3 = -11_{10}$ et $n_2 = 1 \times 1.875 \times 2^{-1} = 0.9375_{10}$, on a :

$$p = (-1 \times 1) \times (1.375 \times 1.875) \times (2^{3-1})$$
$$= -1 \times 2.578125 \times 2^2$$
$$= -1 \times 1.2890625 \times 2^3$$
$$= -10.3125.$$

L'étape de normalisation doit aussi inclure la troncation des bits les moins significatifs du produit pour exprimer la mantisse avec *M* bits.

Le diagramme suivant montre un circuit permettant d'effectuer la multiplication de nombres à virgule flottante, et le code d'une entité est donné à la page suivante. Donnez le code VHDL synthétisable d'une architecture correspondant à cette entité telle qu'elle est décrite ici.



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

entity mul_uvf is
   generic (
      E : natural := 7; -- nombre de bits des exposants
      M : natural := 8  -- nombre de bits des mantisses
   );
   port(
      CLK, reset : in std_logic;
      s1, s2 : in std_logic;                -- bits de signe des entrées
      sp : out std_logic;                   -- bit de signe du produit
      e1, e2 : in signed(E - 1 downto 0);   -- exposants
      ep : out signed(E - 1 downto 0);      -- exposant du produit
      m1, m2 : in unsigned(M - 1 downto 0); -- mantisses
      mp : out unsigned(M - 1 downto 0)     -- mantisse du produit
   );
end mul_uvf;
```

Question 2. (8 points)

Considérez le diagramme du multiplicateur de nombres à virgule flottante de la Question 1. Une instance du circuit avec $E = 8$ et $M = 23$ est implémentée pour un FPGA de la famille Virtex-5. Après implémentation, on a caractérisé les différentes parties du circuit comme suit.

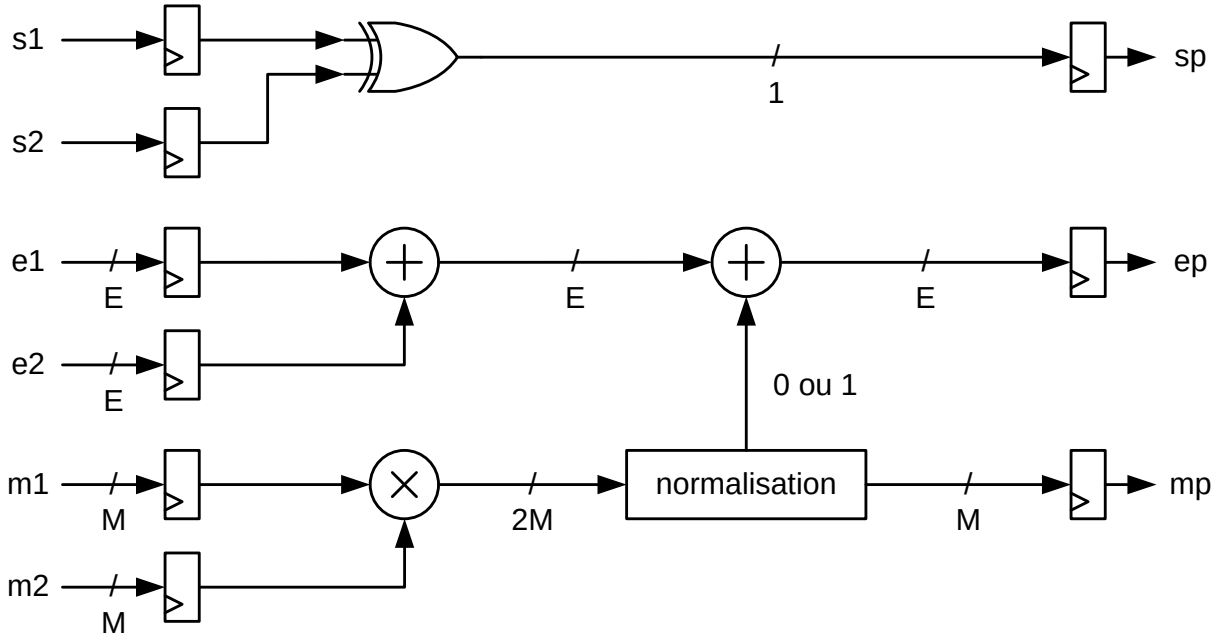
Les bascules ont un délai de 0.4 ns, un temps de préparation t_{su} de 0.4 ns et un temps de maintien t_h de 0.2 ns. La porte OUX est implémentée par une LUT dont le délai est de 0.1 ns. Les additionneurs ont des délais de 1.75 ns. Le multiplicateur est implémenté sur un des blocs DSP et a un délai de 3.25 ns. La fonction de normalisation a un délai de 2.5 ns.

Suite au placement et routage, on estime que les fils d'interconnexion entre chacun des blocs ont des délais de 0.1 ns chacun.

a. (2 points) Identifiez le chemin critique du circuit et donnez la fréquence maximale d'opération.

b. (1 point) Donnez la latence du circuit en secondes et le débit du circuit en résultats par seconde.

c. (2 points) On désire atteindre un débit minimal de 175×10^6 résultats par seconde en gardant la latence la plus faible possible. Indiquez comment pipeliner le circuit pour satisfaire ces spécifications. Annotez le diagramme suivant. Justifiez complètement votre réponse.



d. (3 points) Indiquez clairement comment modifier votre code VHDL donné en réponse à la Question 1 pour inclure votre pipeline de la partie c.

Question 3. (4 points)

Considérez le module du calcul du produit de deux nombres en virgule flottante de la Question 1 avec $E = 4$ et $M = 6$.

a. (1 point) Combien de vecteurs de test sont nécessaires pour une vérification exhaustive de cette entité?

b. (2 points) Pour chacune des entrées, donnez les valeurs à tester selon l'analyse des valeurs limites.

c. (0.5 point) Pour les valeurs limites de la partie b), quelle est la taille minimale d'un ensemble de vecteurs de tests pour effectuer un test faible? Justifiez complètement votre réponse.

d. (0.5 point) Pour les valeurs limites de la partie b), quelle est la taille minimale d'un ensemble de vecteurs de tests pour effectuer un test fort? Justifiez complètement votre réponse.

Question 4. (8 points)

a. (0.5 point) Expliquez pourquoi un ensemble de vecteurs de tests donnant une couverture de code de 100% n'est pas nécessairement suffisant pour vérifier adéquatement un module.

b. (1.5 point) Donnez trois qualités d'un bon ensemble de vecteurs de test.

c. (1 point) On vous propose de participer au développement d'un nouveau langage de description matérielle, PolyHDL. Ce langage devra servir à modéliser des circuits numériques qui seront implémentés sur FPGA. Donnez deux caractéristiques que vous considérez essentielles pour ce langage.

d. (1 point) Expliquez comment un double tampon peut permettre de synchroniser l'échange d'informations entre deux domaines d'horloge différents.

e. (1 point) Expliquez pourquoi la division générale n'est pas synthétisable par la plupart des synthétiseurs de code HDL ciblant des FPGA.

f. (1 point) Nommez et décrivez deux principes à suivre pour la conception et l'implémentation de circuits numériques sur FPGAs.

g. (1 point) Donnez deux causes possibles du déphasage d'horloge.

h. (1 point) Donnez deux métriques pour comparer différentes solutions de placement d'un circuit numérique sur un FPGA.

Question 5. (4 points)

Considérez le code VHDL suivant. Encerclez les quatre erreurs détectées au moment de la compilation et, pour chaque ligne comportant une erreur, donnez le code corrigé correspondant.

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

entity unitearithmetique is
   generic (
      W : positive := 8 -- largeur des opérandes
   );
   port (
      A, B : in integer(W - 1 downto 0);
      choix : in std_logic_vector(2 downto 0);
      F : out signed(W - 1 downto 0)
   );
end unitearithmetique;

architecture arch of unitearithmetique is
begin
   process(A, B, choix)
   variable t : integer range 0 to 2 ** W - 1;
   begin
   t <= to_integer(A * B);
   case to_integer(unsigned(choix)) is
      when 4 => F <= signed(abs(A));
      when 6 => F <= t;
      when others => F <= 'X';
   end case;
   end process;
end arch;
```

Question 6. (8 points)

Une opération communément utilisée en traitement d'images est le calcul de la somme des différences absolues (SDA) entre deux blocs de pixels A et B. Cette somme est une mesure de la similarité entre les deux images. Supposons deux blocs de 16 × 16 pixels (256 pixels), entreposés dans deux mémoires distinctes de 256 octets chacune. Chaque octet de chaque mémoire contient l'intensité d'un seul pixel sous la forme d'un nombre non signé entre 0 et 255 inclusivement. Une version en pseudocode d'un algorithme permettant de calculer la SDA des blocs A et B est donné ici.

```
SDA = 0;
SDAprete = vrai;
tanque(vrai) {
    tantque (go = 0) {
        ; // attendre
    }
    adresse = 255;
    SDA = 0;
    SDAprete = faux;
    tant que (adresse >= 0) {
        SDA = SDA + abs(A[adresse] – B[adresse]);
        adresse = adresse – 1;
    }
    SDAprete = vrai;
}
```

Le signal de contrôle `SDAprete` est activé par le module quand le calcul est terminé. Le signal de contrôle `go` est activé par le reste du système quand les mémoires sont chargées et que les calculs peuvent débuter.

a. (4 points) Donnez un diagramme montrant le chemin des données du processeur. Indiquez clairement toutes les ressources requises, leur nombre, leur largeur en bits et leur signaux de contrôle.

b. (4 points) Donnez la machine à états du processeur. Identifiez bien chaque état, les actions à prendre dans chaque état, et les conditions pour les transitions entre les états.

INF3500 : Conception et réalisation de systèmes numériques

Solutionnaire d'examen final

Décembre 2013

Q1.

```
architecture arch of mul_uvf is

signal s1r, s2r : std_logic;
signal e1r, e2r : signed(E - 1 downto 0);
signal m1r, m2r : unsigned(M - 1 downto 0);

begin

   -- registres d'entrée
   process (CLK, reset)
   begin
      if reset = '0' then
         s1r <= '0'; s2r <= '0';
         e1r <= (others => '0'); e2r <= (others => '0');
         m1r <= (others => '0'); m2r <= (others => '0');
      elsif rising_edge(CLK) then
         s1r <= s1; s2r <= s2;
         e1r <= e1; e2r <= e2;
         m1r <= m1; m2r <= m2;
      end if;
   end process;

   -- calculs
   process (CLK, reset)
   variable mt : unsigned(2 * M - 1 downto 0);
   variable et : signed(E - 1 downto 0);
   begin
      if reset = '0' then
         sp <= '0';
         ep <= (others => '0');
         mp <= (others => '0');
      elsif rising_edge(CLK) then
         sp <= s1r xor s2r;
         mt := m1r * m2r;
         et := e1r + e2r;
         if mt >= 2 ** (M - 1) then
            mt := mt / 2;
            et := et + 1;
--       elsif mt < 2 ** (M - 2) then -- ne peut pas arriver!
--          mt := mt * 2;
--          et := et - 1;
         end if;
         mp <= mt(2 * M - 1 downto M);
         ep <= et;
      end if;
   end process;
end arch;
```

Q2.

a. Le chemin critique va d'une des bascules m, à travers le multiplicateur, le bloc de normalisation, l'additionneur, puis la bascule pour ep.

Le délai total est de 0.4 + 0.1 + 3.25 + 0.1 + 2.5 + 0.1 + 1.75 + 0.1 + 0.4 ns = 8.7 ns. La fréquence maximale d'opération est donc de 114.9 MHz.

b. La latence du circuit est de un cycle (sans compter la bascule à l'entrée), donc 8.7 ns, et le débit est de 114.9 M résultats par seconde.

c. Pour atteindre 175 M résultats/seconde, sans paralléliser, il faut un délai maximal de 5.71 ns.

On fait l'essai de placer un registre de pipeline après le multiplicateur. Le chemin critique serait :

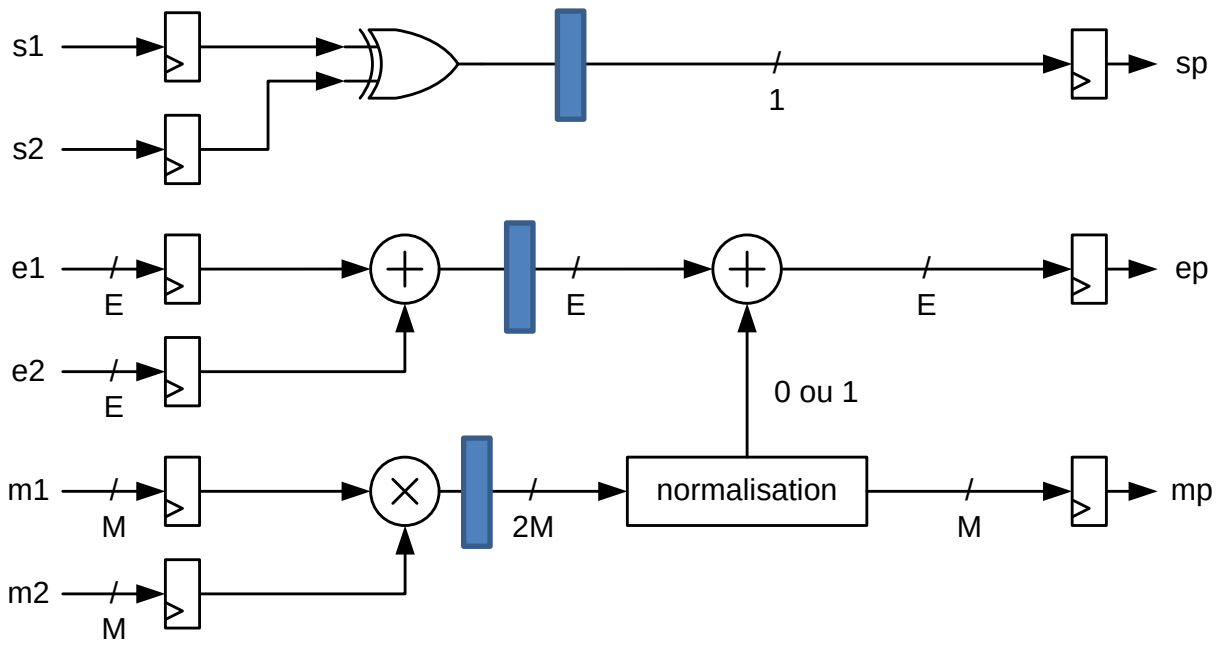
m1 à registre entre multiplicateur et normalisation, délai = 0.4 + 0.1 + 3.25 + 0.1 + 0.4 ns = 4.25 ns.

registre entre multiplicateur et normalisation à registre ep, délai = 0.4 + 0.1 + 2.5 + 0.1 + 1.75 + 0.1 + 0.4 ns = 5.35 ns.

Le chemin critique a donc un délai de 5.35 ns, ce qui correspond à une fréquence maximale d'opération de 186.9 MHz et un débit de 186.9 M résultats par seconde.

La latence serait de deux cycles, soit 10.7 ns.

Pour équilibrer le pipeline, il faut aussi placer un registre de pipeline entre les deux additionneurs et un après la porte OUX.



d.

```
architecture arch_pipe of mul_uvf is

signal sp_pipe : std_logic;
signal m1Xm2 : unsigned(2 * M - 1 downto 0);
signal e1pluse2 : signed(E - 1 downto 0);

signal s1r, s2r : std_logic;
signal e1r, e2r : signed(E - 1 downto 0);
signal m1r, m2r : unsigned(M - 1 downto 0);

begin

   -- registres d'entrée
   process (CLK, reset)
   begin
      if reset = '0' then
         s1r <= '0'; s2r <= '0';
         e1r <= (others => '0'); e2r <= (others => '0');
         m1r <= (others => '0'); m2r <= (others => '0');
      elsif rising_edge(CLK) then
         s1r <= s1; s2r <= s2;
         e1r <= e1; e2r <= e2;
         m1r <= m1; m2r <= m2;
      end if;
   end process;

   -- calculs pipelinés
   process (CLK, reset)
   variable mt : unsigned(2 * M - 1 downto 0);
   variable et : signed(E - 1 downto 0);
   begin
      if reset = '0' then
         sp <= '0'; sp_pipe <= '0';
         ep <= (others => '0'); e1pluse2 <= (others => '0');
         mp <= (others => '0'); m1Xm2 <= (others => '0');
      elsif rising_edge(CLK) then
         sp_pipe <= s1r xor s2r;
         sp <= sp_pipe;

         m1Xm2 <= m1r * m2r;
         mt := m1Xm2;

         e1pluse2 <= e1r + e2r;
         et := e1pluse2;

         if m1Xm2 >= 2 ** (M - 1) then
            mt := m1Xm2 / 2;
            et := e1pluse2 + 1;
         end if;

         mp <= mt(2 * M - 1 downto M);
         ep <= et;
      end if;
   end process;
end arch_pipe;
```

Q3

a. Il y a $2 \times 11 = 22$ bits en entrée, donc 2^{22} vecteurs de test possibles.

b. Pour l'entrée s1, {0, 1}, pour l'entrée s2 {0, 1}.

Pour l'entrée m1, {1.00000, 1.00001, 1.10000, 1.11110, 1.11111}.

Pour l'entrée m2, {1.00000, 1.00001, 1.10000, 1.11110, 1.11111}.

Pour l'entrée e1, {-8, -7, 0, 6, 7}.

Pour l'entrée e2, {-8, -7, 0, 6, 7}.

c. Il faudrait prendre un vecteur de test pour chaque entrée au moins une fois, donc 5 vecteurs de test.

d. Il faudrait prendre $5 \times 5 \times 5 \times 5 \times 2 \times 2 = 2500$ vecteurs de test.

Q4

a. La couverture de code n'indique que si certaines situations ont été exercées ou non, sans égard à la fonctionnalité du système.

b. Efficace pour découvrir des bogues, c'est-à-dire que chaque vecteur de test vérifie plusieurs fonctionnalités en même temps, et donc que peu de vecteurs de tests sont nécessaires.

Identifie la source des bogues, pour aider à leur éradication.

Reproductible, donc il est facile de recréer le bogue.

Automatisé à l'aide d'un banc d'essai.

Exécutable dans un temps raisonnable

c. pouvoir représenter la concurrence, pouvoir modéliser des composantes matérielles, permettre la simulation du modèle, permettre la synthèse du modèle, inclure des opérateurs correspondant à des structures disponibles sur FPGA, permettre la description de bancs d'essais, favoriser l'abstraction et la hiérarchie, etc.

d. Un double tampon consiste de deux bascules en série. L'entrée asynchrone est appliquée à la première bascule. Il est possible que celle-ci entre dans un état métastable. Elle finira en pratique par se stabiliser sur un 1 ou un 0. La deuxième bascule permet d'accorder le plus long temps possible à la première bascule pour se stabiliser, soit une période d'horloge.

e. La division générale n'est pas synthétisable par choix des concepteurs de synthétiseurs. Puisque les FPGA ne possèdent pas de bloc implémentant la division, il faudrait choisir une architecture de diviseur en particulier et la réaliser avec un arrangement de LUT, FF, etc. Les concepteurs de synthétiseurs préfèrent laisser ce choix aux concepteurs de systèmes numériques qui écrivent du code HDL.

f. Quelques solutions possibles : conception synchrone, ne pas utiliser de loquets, signal d'initialisation global, routage spécial du signal d'horloge, exploitation de toutes les ressources de la puce, gestion manuelle de la disposition, arithmétique en virgule fixe plutôt que flottante.

g. signaux d'horloge avec des chemins de longueur différente, signal d'horloge avec un débalancement de la charge du signal, contrôle du signal d'horloge avec de la logique combinatoire (*clock gating*).

h. longueur totale des interconnexions, chemin le plus long, congestion des interconnexions.

Q5

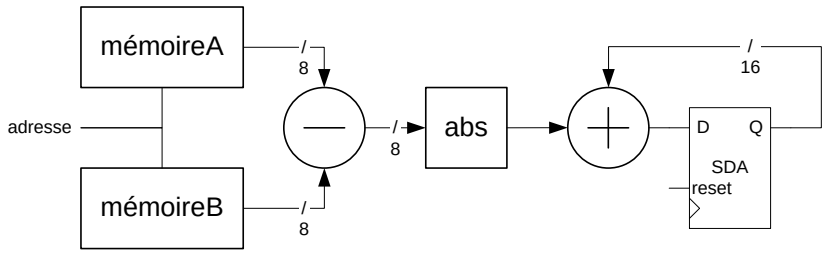
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity unitearithmetique is
   generic (
      W : positive := 8
   );
   port(
--    A, B : in integer(W - 1 downto 0);
      A, B : in signed(W - 1 downto 0);
      choix : in std_logic_vector(2 downto 0);
      F : out signed(W - 1 downto 0)
   );
end unitearithmetique;

architecture arch of unitearithmetique is
begin
   process(A, B, choix)
   variable t : integer range 0 to 2 ** W - 1;
   begin
-- t <= to_integer(A * B);
   t := to_integer(A * B);
   case to_integer(unsigned(choix)) is
      when 4 => F <= signed(abs(A));
--    when 6 => F <= t;
      when 6 => F <= to_signed(t, W);
--    when others => F <= 'X';
      when others => F <= (others => 'X');
   end case;
   end process;
end arch;
```

Q6

a.



b.