INF3500 : Conception et réalisation de systèmes numériques

Corrigé de l'examen final

Vendredi 3 mai 2013

Durée: 2h30.

Pondération: 40%.

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.
- Ne pas remettre le questionnaire.
- 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. (9 points)

Au laboratoire, vous avez eu à concevoir un processeur à usage général dont les instructions tiennent sur 28 bits, et dont la déclaration d'entité suit. Le jeu d'instructions de ce processeur est donné à l'Annexe I.

```
entity processeurv20 is
   generic (
      Nreg       : integer := 32; -- nbre de registres
      Wd         : integer := 16; -- largeur du chemin des données en bits
      Wi         : integer := 28; -- largeur des instructions en bits
      Mi         : integer := 12; -- nbre de bits d'adresse de la mém. d'instructions
      Md         : integer :=  8; -- nbre de bits d'adresse de la mém. des données
      resetvalue : std_logic := '1'
   );
   port(
      reset               : in  std_logic;
      clk                 : in  std_logic;
      entreeExterne       : in  signed(Wd-1 downto 0);
      fifoInNotReady      : in  std_logic;
      reqEntreeExterne    : out std_logic;
      sortieExterne       : out signed(Wd-1 downto 0);
      fifoOutNotReady     : in  std_logic;
      ecrireSortieExterne : out std_logic
   );
end processeurv20;
```

a. Quelle sera la valeur du registre `r0` à la fin de l'exécution de ce programme ?

```
constant memoireInstructions : memoireInstructions_type :=
(x"A070003", -- r7    <- +3                / addr x0
 x"A080001", -- r8    <- +1                / addr x1
 x"A000000", -- r0    <- +0                / addr x2
 x"9000000", -- M[r0] <- r0                / addr x3
 x"A010001", -- r1    <- +1                / addr x4
 x"9000101", -- M[r1] <- r1                / addr x5
 x"8020000", -- r2    <- M[r0]             / addr x6
 x"8030100", -- r3    <- M[r1]             / addr x7
 x"3030300", -- r3    <- r3 << 1           / addr x8
 x"0030203", -- r3    <- r2 + r3           / addr x9
 x"7000100", -- r0    <- r1                / addr xA
 x"0010108", -- r1    <- r1 + r8           / addr xB
 x"9000103", -- M[r1] <- r3                / addr xC
 x"1070708", -- r7    <- r7 - r8           / addr xD
 x"C020006", -- jump if not Z to addr x6   / addr xE
 x"7000300", -- r0    <- r3                / addr xF
others => (others => '1'));
```

À la fin de l'exécution de ce programme, r0 aura la valeur 12.

b. Quel est le nombre de cycles d'horloge nécessaire à l'exécution du programme ? Justifiez clairement votre réponse en énonçant vos hypothèses de travail.

Les instructions x0 à x5 ainsi que xF seront exécutées 1 fois. Les instruction x6 à xE seront exécutées 3 fois. On a un total de (6+1) + 3×(9) = 34 instructions exécutées. Chaque instruction nécessitant 3 cycles d'horloges, il faudra donc 102 cycles d'horloge pour que le programme s'exécute.

c. Décrivez dans un pseudo-code le plus lisible possible l'algorithme précédent. On supposera que l'algorithme retourne la valeur du registre `r0`.

```
M[0] = 0;
M[1] = 1;
for i = 1 : 3
        M[i+1] = 2M[i] + M[i-1];
end for
return M[4];
```

L'algorithme qui est exécuté ressemble à qui produit la série des nombres de Fibonacci (elle s'appelle série des nombres de Pell), sauf que la série utilisée ici est décrite par la récurrence:

$$P_n = 2P_{n-1} + P_{n-2}, \text{ où } P_0 = 0 \text{ et } P_1 = 1.$$

e. Quelle est la valeur de n du P_n retourné par ce programme ?

$n = 4$.

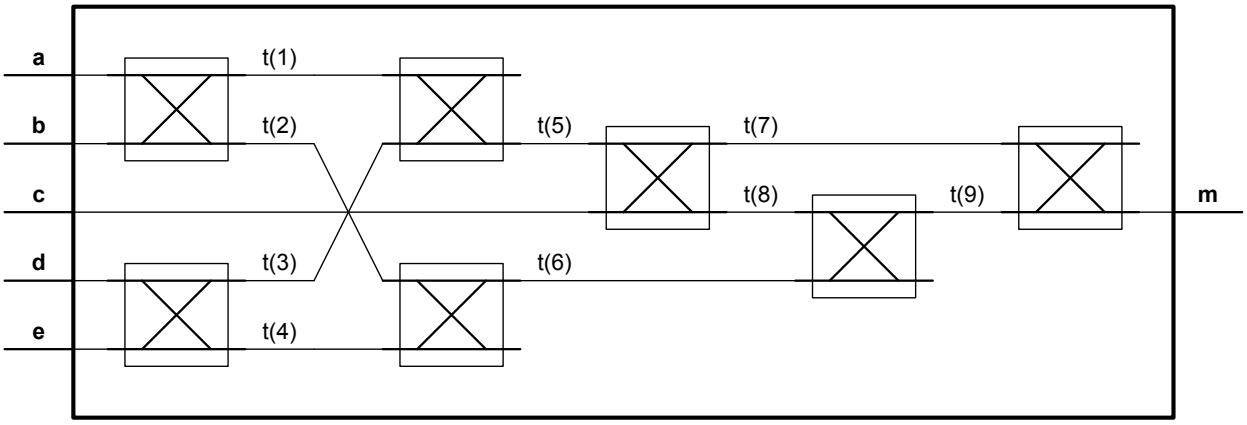
f. Quels changements faut-il apporter au programme pour qu'il retourne P_{25}. Indiquez dans votre réponse la ou les lignes d'instructions à modifier et comment il faut les modifier.

Il suffit de modifier la première ligne :

```
(x"A070018", -- r7    <- +24                  / addr x0
```

Question 2. (8 points)

On vous demande de considérer le circuit permettant de trouver la valeur médiane de cinq nombres signés donnés en entrée : a, b, c, d et e. Ainsi, si (a, b, c, d, e) = (+3, +8, +5, −7, +2), la sortie devrait être m = +3. Le code VHDL de l'entité est fourni à l'Annexe II. L'implémentation de cette dernière utilise un réseau de tri que reproduit la figure suivante.



a) Quelle est le nombre de vecteurs de test nécessaires pour effectuer un test exhaustif de l'entité **`median5`** ? Justifiez votre réponse par un calcul.

Il faut 2^{5W} vecteurs de test.

Pour tester cette entité, on vous propose de définir cinq classes pour chacune des entrées du circuit **`median5`** (on suppose w≥3) :

$$\{ \{-2^{w-1}, -2^{w-1}+1, \ldots -2^{w-2}-1\},$$
$$\{-2^{w-2}, -2^{w-2}+1, \ldots -2^{w-3}-1\},$$
$$\{-2^{w-3}, \ldots +2^{w-3}-1\},$$
$$\{+2^{w-3}, +2^{w-2}+1, \ldots +2^{w-2}-1\},$$
$$\{+2^{w-2}, +2^{w-2}+1, \ldots +2^{w-1}-1\} \}$$

b) Quelle est la taille minimale du vecteur de test (*de l'ensemble de vecteurs de test*) pour effectuer un test faible sur l'entité **`median5`**, et ce en recourant aux classes identifiées précédemment. Justifiez votre réponse.

Il faut au minimum 5 valeurs puisque nous avons cinq classes pour chacune des cinq entrées.

c) Quelle est la taille minimale du vecteur de test (*de l'ensemble de vecteurs de test*) pour effectuer un test fort sur l'entité **`median5`**, et ce en recourant aux classes identifiées précédemment. Justifiez votre réponse.

Il faut au minimum $5^5 = 3125$ vecteurs de test puisque nous avons cinq classes pour chacune des cinq entrées.

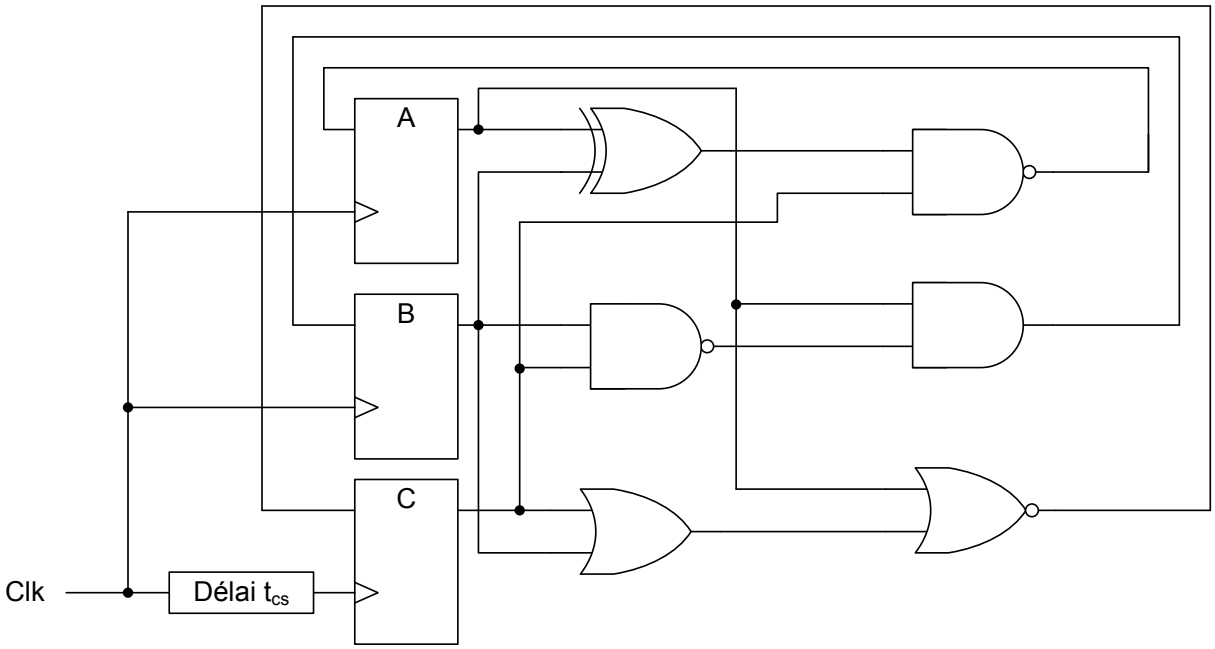
Il vous est proposé d'exploiter le banc de test reproduit à l'Annexe III pour tester le module **`median5`**.

d) Quelle est la couverture de branchement obtenue en utilisant ce banc de test ?

Bien qu'il s'agisse d'un test faible, la façon dont les vecteurs sont définis assure une couverture de branchement de 100%

Question 3. (7 points)

Considérez le circuit suivant. Les délais des portes logiques combinatoires sont respectivement de 4 ns pour un XOR, 3 ns pour un ET/OU, 2 ns pour un NON ET/NON OU. Les bascules ont un temps de préparation de 1 ns, un temps de maintien de 0.5 ns et un temps de propagation de 2 ns. On néglige le délai associé aux fils.



En supposant dans un premier temps que le délai de retard sur l'horloge est absent du circuit (t_{cs} = 0):

a) Donnez le chemin critique du circuit en indiquant les bascules sources et la bascule de destination, ainsi que les composantes sur ce chemin.

Le chemin critique part des bascules A/B et se rend à la bascule A. Il traverse un XOR et un NON-ET.

b) Donnez la fréquence maximale d'horloge correspondante.

La période minimale étant de 4 + 2 + 1 +2 = 9 ns, la fréquence maximale est 111 MHz.

c) Donnez la marge libre de préparation pour les autres chemins.

A-B 3 ns, A-C 4 ns, B-B 1 ns, B-C 1 ns, C-A 3 ns, C-B 1ns, C-C 1ns.

d) En supposant que vous pouvez ajouter autant de registres que vous le voulez, quelle est la fréquence maximale d'opération du circuit si on procédait à son pipelinage ? Justifiez clairement votre réponse.

La fréquence maximale serait relative à la meilleure période d'horloge que l'on puisse obtenir, c'est-à-dire 3+4 ns = 7 ns, soit 142 MHz.

En supposant que le délai de retard du signal d'horloge vers la bascule C est t_{cs}=2 ns:

e) Donnez le chemin critique du circuit en indiquant les bascules sources et la bascule de destination, ainsi que les composantes sur ce chemin.

Le chemin critique part de la bascule C et se termine sur la bascule B. Les portes traversées sont un NON-ET et un ET.

f) Donnez la fréquence maximale d'horloge correspondante.

La période minimal de ce chemin est 8 + 2 = 10 ns. La fréquence maximale serait alors de 100 MHz.

g) Donnez la marge libre de préparation pour les autres chemins.

A-B 1 ns, A-B 4 ns, A-C 7 ns, B-A 1ns, B-B 2 ns, B-C 4 ns, C-A 2 ns, C-C 2 ns.

Question 4. (8 points)

On vous demande de considérer le code VHDL donné à l'Annexe IV qui implémente un arbre d'additionneurs dont le nombre d'entrées (donné par N) est une puissance de 2.

Le Tableau 1 donne les estimés de synthèse pour différentes valeurs de N (W est fixé à 8).

Tableau 1

N	Nombre de registres	Nombre de LUT	Période minimale	Fréquence max.
2	24	8	2.026 ns	493.583 MHz
4	40	21	3.384 ns	295.508 MHz
8	72	50	4.757ns	210.217 MHz
16	136	105	6.115ns	163.532 MHz

a) Expliquez l'évolution du nombre de registres obtenus pour les différentes valeurs de N rapportées au Tableau 1. Justifiez votre réponse par un calcul.

Le nombre de registres est égal à 8N (entrées) + 8 (sortie), puisque W = 8.

b) Sachant que les registres en entrée ne sont pas pris en compte dans le calcul de la latence, donnez la latence de l'arbre d'additionneurs pour les différentes valeurs de N données au Tableau 1.

La latence est de un cycle, soit le registre de sortie.

c) Pour chacune des valeurs de N données au Tableau 1, estimez le débit de cet opérateur.

N	Débit (traitements par seconde)
2	493.583 e+6
4	295.508 e+6
8	210.217 e+6
16	163.532 e+6

Considéré maintenant l'implémentation de l'arbre d'additionneurs donnée à l'Annexe V pour laquelle le Tableau 2 donne les estimés de synthèse pour différentes valeurs de N (W est fixé à 8).

Tableau 2

N	Nombre de registres	Nombre de LUT	Période minimale	Fréquence max.
2	24	8	2.026 ns	493.583 MHz
4	56	24	2.026 ns	493.583 MHz
8	120	56	2.026 ns	493.583 MHz
16	248	120	2.026 ns	493.583 MHz

d) Expliquez pourquoi la période minimale (et par voie de conséquence, la fréquence maximale) est identique pour tous les N considérés au Tableau 2.

Le circuit est maintenant pipeliné, ce qui fait que la fréquence maximal est celle d'un additionneur simple. Ainsi tous les arbres d'additionneurs ont une fréquence maximale d'opération égale à celle de l'arbre à deux entrées.

e) Expliquez l'évolution du nombre de registres obtenus pour les différentes valeurs de N rapportées au Tableau 2. Justifiez votre réponse par un calcul.

Cette fois, le nombre de registres est W(N +N/2+N/4+…+N/N).

Pour N= 2, cela donne 8(2+1) = 24

Pour N= 4, cela donne 8(4+2+1) = 56

Pour N= 2, cela donne 8(8+4+2+1) = 120

Pour N= 2, cela donne 8(16+8+4+2+1) = 248

f) Sachant que les registres en entrée ne sont pas pris en compte dans le calcul de la latence, donnez la latence de cet arbre d'additionneurs pour les différentes valeurs de N données au Tableau 2.

N	Latence
2	1
4	2
8	3
16	4

g) Pour chacune des valeurs de N données au Tableau 2, estimez le débit de l'opérateur.

Le débit est 493.583e+6 pour toutes les valeurs de N.

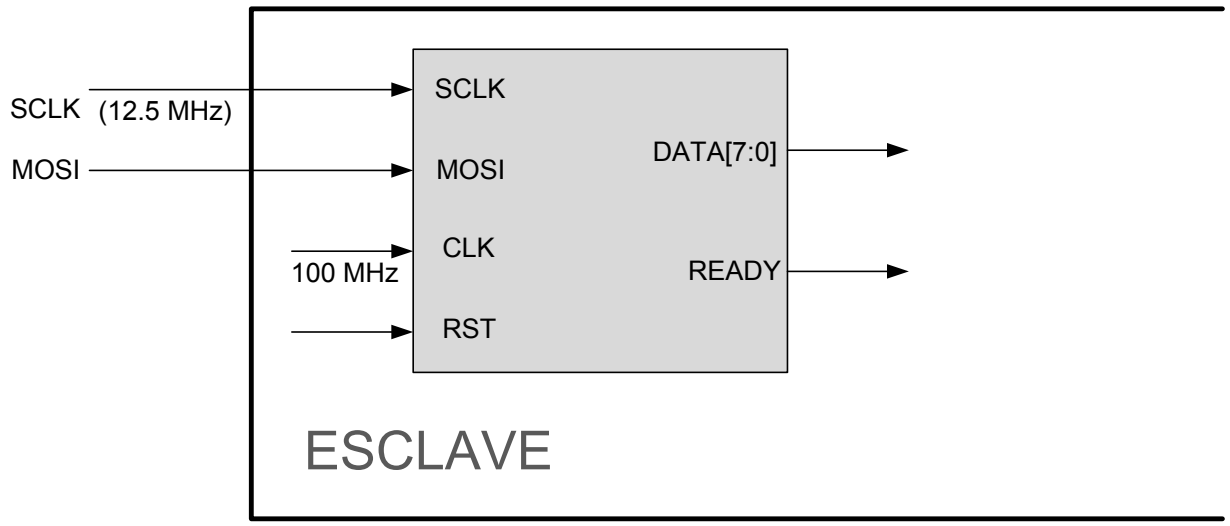
Question 5. (8 points)

On désire concevoir un lien SPI (*serial peripheral interface bus*). Une liaison SPI est un bus synchrone qui opère en full duplex, c'est-à-dire dans les deux sens et en même temps (c'est le cas également de la communication RS232 que vous avez manipulée au laboratoire). La liaison point à point dans un lien de communication SPI se fait entre un "maître" et un "esclave". La figure suivante illustre le lien de communication entre les deux unités:

Voici une description des signaux du lien de communication:

- ✓ SCLK : est un signal d'horloge généré par le maître durant la communication et par lequel les données (sur les lignes MOSI/MISO) sont synchronisées.
- ✓ MOSI : est un signal de données de 1 bit permettant au maître d'envoyer des données à l'esclave. L'acronyme est pour Master Out, Slave In.
- ✓ MISO : est un signal de données de 1 bit permettant à l'esclave d'envoyer des données au maître lorsque le maître initie une communication. L'acronyme signifie Master In, Slave Out.
- ✓ $\overline{SS}$: est un signal actif bas servant à sélectionner l'esclave avec lequel le maître veut communiquer. Il est utile dans une topologie où plusieurs esclaves sont attachés au maître. Nous ne le considérons pas ici.

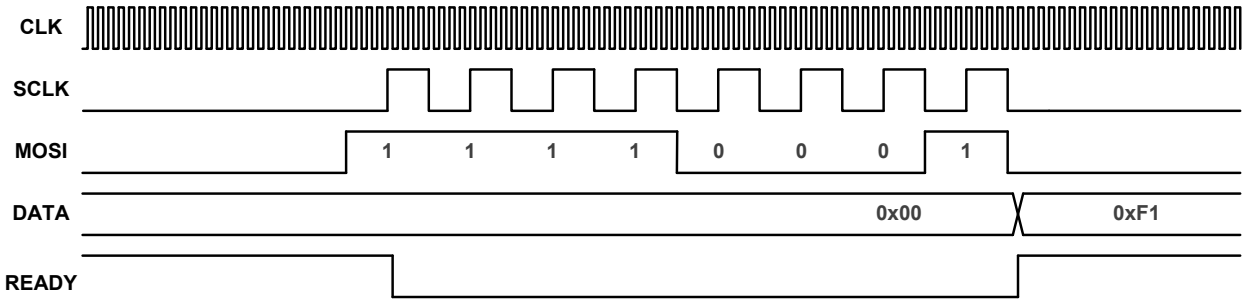
Nous désirons réaliser une version simplifiée de le **module de réception** de l'esclave. Celui-ci se contente des signaux SCLK et MOSI, tel qu'illustré à la figure suivante:



Voici une description des signaux du bloc à réaliser:

- ✓ CLK : signal d'horloge interne au FPGA oscillant à 100 MHz.
- ✓ RST : signal de réinitialisation, synchrone sur CLK.
- ✓ DATA: bus de données permettant de communiquer le message reçu.
- ✓ SCLK: signal d'horloge SPI, ce signal est asynchrone du point de vue de CLK. Il est actif uniquement durant une communication. On suppose que sa fréquence est inférieure à 12.5 MHz.
- ✓ MOSI : signal de donnée SPI permettant au maître d'envoyer des données de manière sérielle vers l'esclave. Les bits de poids fort sont envoyés en premier.
- ✓ READY: signal indiquant que l'unité de réception est soit en attente d'un message, soit qu'elle vient d'en recevoir un nouveau (ce qui se déduit par l'observation d'un front montant sur ledit signal).

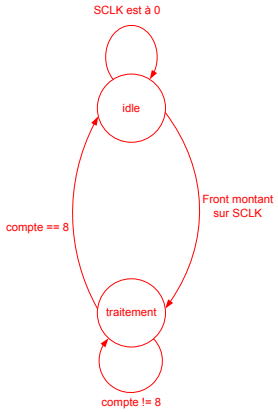
La figure suivante illustre le fonctionnement de la communication. Dans cet exemple, le maître a envoyé le mot 0xF1 sur la ligne sérielle MOSI. On voit également que l'horloge SCLK est activée uniquement durant la transmission (on voit donc 8 fronts montants).



a) Quel sera le nombre d'états de l'unité de contrôle du processeur spécialisé permettant d'implémenter le module de réception de l'esclave. Justifiez clairement votre réponse.

Il suffit de deux états. Un état d'attente, et un état de traitement.

b) Dessinez le diagramme d'états de l'unité de contrôle du processeur spécialisé permettant d'implémenter le module de réception de l'esclave.



c) Proposez une implémentation VHDL permettant de réaliser le module de réception de l'esclave. Vous n'êtes pas obligés de prendre en compte les risques de métastabilité sur les signaux en entrée, mais c'est très fortement recommandé. Votre module doit être cadencé par le signal CLK et se réinitialiser de façon asynchrone sur le signal RST.

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

entity spi_receive is
generic( w : positive := 8 );
port(
   rst, clk, sclk, mosi : in std_logic;
   ready : out std_logic;
   data  : out std_logic_vector(w-1 downto 0)
);
end spi_receive;

architecture arch of spi_receive is
   signal sclk_1, sclk_2, sclk_3, mosi_1, mosi_2: std_logic;
begin
   process(rst, clk) is
      variable compte : natural range 0 to w := 0;
      variable rdy    : boolean := true;
      variable buff   : std_logic_vector(w-1 downto 0);
   begin

      if( rst = '1' ) then
         compte <= '0'
         rdy := true;
         ready <= '1';
      elsif( rising_edge( clk ) ) then
         sclk_1 <= sclk;   mosi_1 <= mosi;
         sclk_2 <= sclk_1; mosi_2 <= mosi_1;
         sclk_3 <= sclk_2;
         if( rdy ) then
            ready <= '1';
            if( sclk_3 = '0' and sclk_2 = '1' )
               ready <= '0';
               rdy := false;
               compte := 1;
               buff := buff(w-2 downto 0) & mosi_2;
            end if;
         else
            ready <= '0';
            if( sclk_3 = '0' and sclk_2 = '1' ) then
               compte := compte + 1;
               buff := buff(w-2 downto 0) & mosi_2;
               if( compte = w ) then
                  ready <= '1';
                  rdy := true;
                  compte := 0;
                  data <= buff;
               end if
            end if
         end if;
      end if;

   end process;
end arch;
```

Annexe I

Jeu d'instructions du processeurv20

Instructions arithmétiques: $r_{dest} \leftarrow A$ op B:

opération	choixCharge	choixA	choixB
IR[27:24] Valeurs permises: x"0" : F ← A + B x"1" : F ← A – B x"2" : F ← A >> 1 x"3" : F ← A << 1 x"4" : F ← not(A) x"5" : F ← A and B x"6" : F ← A or B x"7" : F ← A	IR[20:16] Adresse du registre de destination	IR[12:8] Adresse du registre de l'opérande A	IR[4:0] Adresse du registre de l'opérande B

Instructions pour l'interfaçage de la mémoire de données:

Instructions	choixCharge	choixA	choixB
IR[27:24] Valeurs permises: x"8": *lire* x"9": *écrire*	IR[20:16] Adresse du registre de destination (s'applique à *lire*)	IR[12:8] Adresse du registre de l'opérande A, servant à l'adressage de la mémoire (s'applique autant à *lire* qu'à *écrire*)	IR[4:0] Adresse du registre de l'opérande B qui sert à contenir la donnée à *écrire*

Instruction de chargement immédiat:

Instruction	Valeur à charger
IR[27:24] Valeurs attendue: x"A"	IR[15:0] Représentation signée

Instructions de branchement

Instruction	Type de branchement	Adresse de branchement
IR[27:24] Valeur attendue: x"C"	IR[19:16] Valeurs permises: x"0" (sans condition) x"1" si = 0 x"2" si ≠ 0 x"3" si < 0 x"4" si ≥ 0	IR[11:0] Adresse de l'instruction de destination

Instruction d'arrêt

Instruction	
IR[27:24] Valeur attendue: x"F"	

Annexe II

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

entity median5 is
generic( w : positive := 8 );
port( a, b, c, d, e : in signed(w-1 downto 0); m : out signed(w-1 downto 0) );
end median5;

architecture arch of median5 is
begin
   process(a, b, c, d, e) is
      type tableau is array(1 to 9) of signed(w-1 downto 0);
      variable t : tableau;
   begin

      if( a >= b ) then
         t(1) := a; t(2) := b;
      else
         t(1) := b; t(2) := a;
      end if;

      if( d >= e ) then
         t(3) := d; t(4) := e;
      else
         t(3) := e; t(4) := d;
      end if;

      if( t(1) >= t(3) ) then
         t(5) := t(3);
      else
         t(5) := t(1);
      end if;

      if( t(2) >= t(4) ) then
         t(6) := t(2);
      else
         t(6) := t(4);
      end if;

      if( t(5) >= c ) then
         t(7) := t(5); t(8) := c;
      else
         t(7) := c; t(8) := t(5);
      end if;

      if( t(8) >= t(6) ) then
         t(9) := t(8);
      else
         t(9) := t(6);
      end if;

      if( t(7) >= t(9) ) then
         m <= t(9);
      else
         m <= t(7);
      end if;

   end process;
end arch;
```

Annexe III

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

entity median5_tb is
generic( w : positive := 8 );
end median5_tb;

architecture tb of median5_tb is
   type tableau is array(1 to 5) of integer;
   constant valeurs : tableau := (-2**(w-1), -2**(w-2), 0, +2**(w-3), +2**(w-2));
   signal a, b, c, d, e, m : signed(w-1 downto 0);

   component median5 is
   generic( w : positive := 8 );
   port( a, b, c, d, e : in signed(w-1 downto 0); m : out signed(w-1 downto 0) );
   end component;

begin

   -- instantiation du l'unité sous test
   uut : median5 generic map( W => w)
   port map( a => a, b => b, c=> c, d => d, e => e, m => m );

   process
      variable c1 : natural := 1;
      variable c2 : natural := 2;
      variable c3 : natural := 3;
      variable c4 : natural := 4;
      variable c5 : natural := 5;
   begin

      for i in 1 to valeurs'length loop
         -- assigner les entrées
         a <= to_signed( valeurs( c1 ) , w );
         b <= to_signed( valeurs( c2 ) , w );
         c <= to_signed( valeurs( c3 ) , w );
         d <= to_signed( valeurs( c4 ) , w );
         e <= to_signed( valeurs( c5 ) , w );

         -- incrémenter les indexes
         if( c1 = 5 ) then c1 := 1; else c1 := c1+1; end if;
         if( c2 = 5 ) then c2 := 1; else c2 := c2+1; end if;
         if( c3 = 5 ) then c3 := 1; else c3 := c3+1; end if;
         if( c4 = 5 ) then c4 := 1; else c4 := c4+1; end if;
         if( c5 = 5 ) then c5 := 1; else c5 := c5+1; end if;

         wait for 10 ns;

         -- varifier la sortie
         assert to_integer( m ) = 0
         report "Erreur sur le vecteur de test " & integer'image(i) severity warning;

      end loop;

      assert false
      report "Simulation ended." severity failure;

   end process;

end tb;
```

Annexe IV

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

entity arbre_additionneurs_v1 is
   generic (
      N : positive := 8; -- nombre d'entrées, doit être une puissance de 2
      W : integer  := 8  -- largeur des données );
   port (
      rst              : in  std_logic;
      clk              : in  std_logic;
      entrees          : in  signed(N*W-1 downto 0);
      resultat         : out signed(W-1 downto 0)
   );
end arbre_additionneurs_v1;

architecture arch of arbre_additionneurs_v1 is

   constant log2N : natural := integer( log2( real(N) ) );
   type tableau_de_signed is array( natural range <> ) of signed(W-1 downto 0);
   signal donnees : tableau_de_signed( 2*N-1 downto 1 );

begin

   assert N = 2**log2N
   report "N doit etre une puissance de 2 " severity failure;

   process( rst, clk) is
   begin
      if( rst = '1' ) then
         -- initialise les registres des entrées
         for i in N to 2*N-1 loop
            donnees( i ) <= (others => '0');
         end loop;

         -- initialise la sortie
         resultat <= (others => '0');

      elsif( rising_edge( clk ) ) then
         -- registre les entrées
         for i in N to 2*N-1 loop
            donnees( i ) <= entrees( (i-N+1)*W-1 downto (i-N)*W );
         end loop;

         -- registre la sortie
         resultat <= donnees ( 1 );

      end if;
   end process;

   process( donnees ) is
   begin
      for i in 1 to log2N loop
         for j in 2**(i-1) to 2**(i)-1 loop
            donnees( j ) <=  donnees( 2*j ) + donnees( 2*j+1 );
         end loop;
      end loop;
   end process;

end arch;
```

Annexe V

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

entity arbre_additionneurs_v1 is
   generic (
      N : positive := 8; -- nombre d'entrées, doit être une puissance de 2
      W : integer  := 8  -- largeur des données );
   port (
      rst              : in  std_logic;
      clk              : in  std_logic;
      entrees          : in  signed(N*W-1 downto 0);
      resultat         : out signed(W-1 downto 0)
   );
end arbre_additionneurs_v1;

architecture arch of arbre_additionneurs_v1 is

   constant log2N : natural := integer( log2( real(N) ) );
   type tableau_de_signed is array( natural range <> ) of signed(W-1 downto 0);
   signal donnees : tableau_de_signed( 2*N-1 downto 1 );

begin

   assert N = 2**log2N
   report "N doit etre une puissance de 2 " severity failure;

   process( rst, clk) is
   begin
      if( rst = '1' ) then
         -- initialise les registres
         for i in 1 to 2*N-1 loop
            donnees( i ) <= (others => '0');
         end loop;

      elsif( rising_edge( clk ) ) then
         -- registre les entrées
         for i in N to 2*N-1 loop
            donnees( i ) <= entrees( (i-N+1)*W-1 downto (i-N)*W );
         end loop;

         for i in 1 to log2N loop
            for j in 2**(i-1) to 2**(i)-1 loop
               donnees( j ) <=  donnees( 2*j ) + donnees( 2*j+1 );
            end loop;
         end loop;

      end if;
   end process;

   resultat <= donnees ( 1 );

end arch;
```