INF3500 – Hiver 2022
Exercices #5 – Conception de chemins des données

0501 Processeurs et chemins des données

1. Donnez le diagramme de chemins des données pour implémenter les micro-opérations suivantes.
   a. R2 ← R3 ET R5
   b. K1: R0 ← R1; K1' : R0 ← R2
   c. K2: R4 ← min(R5, R6)
   d. R4 == 0 : R9 ← R3 + R2; R4 != 0 : R9 ← R3 – R2

2. Donnez un diagramme de chemins des données pour implémenter les micro-opérations suivantes.

init : N ← N0; init' ET N > 0 : N ← N - 1
init : fini ← 0; init' ET N == 0 : fini ←1

Pour ces opérations, N et fini sont gardés dans des registres, et init est un signal de contrôle qui vient de l'extérieur. N0 est une valeur qui vient de l'extérieur.

On peut aussi représenter ces micro-opérations par le pseudo-code suivant. Chaque if-else peut correspondre à l'action d'un multiplexeur.

```
si init == 1 {
    N ← N0;
} else {
    si N > 0 {
        N ← N - 1;
    } else {
        -- N reste inchangé
   -- N ← N;
   }
}

si init == 1 {
    fini ← 0;
} else {
    si N == 0 {
        fini ← 1;
    } else {
        -- fini reste inchangé
   -- fini ← fini;
   }
}
```

3. Donnez le diagramme d'un chemin des données pour les micro-opérations suivantes. On suppose que tous les registres ont 16 bits de large. Vous pouvez utiliser les composantes suivantes : registres, fonctions logiques, additionneurs, soustracteurs, décaleurs, multiplexeurs, décalage et ROM. La ROM retourne une valeur gardée en mémoire, e.g. R9 ← ROM[5] place la valeur entreposée à l'adresse 5 de la mémoire dans le registre R9. R8 ← ROM[R1] place la valeur entreposée à l'adresse pointée par le registre R1 dans le registre R8.

`R3 ← R3 + 1;`

`K1: R0 ← R0 – R1 srl R3; K1': R0 ← R0 + R1 srl R3;`
(où Rx srl Ry veut dire décalage logique du contenu du registre Rx vers la droite de Ry positions)

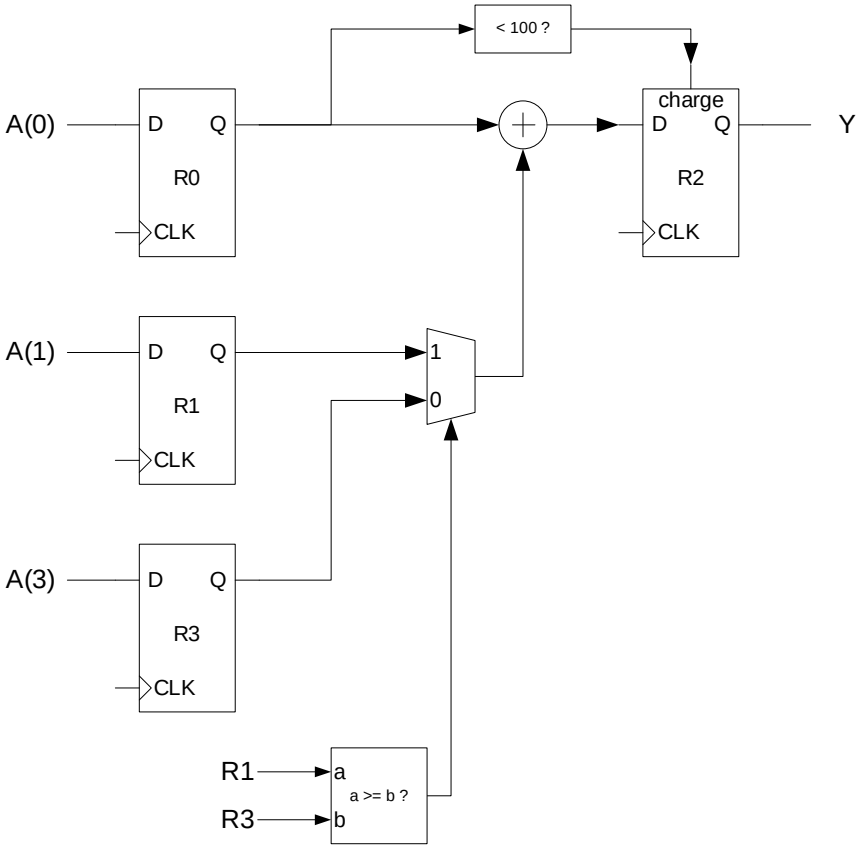
`K1: R1 ← R1 + R0 srl R3; K1': R1 ← R1 – R0 srl R3;`

`K1: R2 ← R2 – ROM[R3]; K1': R2 ← R2 + ROM[R3];`

`K1: t ← signe(R2);` (le signe d'un nombre est donné par son bit le plus significatif)

0502 Modélisation VHDL d'un chemin des données

4. Donnez le code VHDL qui correspond au chemin des données suivant, pour lequel les ports A(2:0) et Y et tous les bus de données sont de type signed et sont exprimés sur 16 bits.



5. Donnez le code VHDL synthétisable d'un compteur BCD à deux chiffres. Chaque chiffre doit être exprimé sur 4 bits et représenter un chiffre décimal de 0 à 9. La réinitialisation (à 0) doit être synchrone avec l'horloge. À chaque coup d'horloge, le compteur doit incrémenter d'une unité et les deux chiffres doivent passer par les nombres entiers {0, 1, 2, 3, … 97, 98, 99, 0, 1, 2, …} en continu.

Complétez les déclarations d'entité et d'architecture suivantes. Un banc d'essai vous est fourni. Assurez-vous que votre code compile correctement et qu'il s'exécute sans erreur avant de le copier-coller dans votre réponse ici.

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

entity compteurBCD is
    generic (
        N : positive := 2 -- nombre de chiffres du compteur
    );
    port (
        reset, clk : in std_logic;
        sortie : out unsigned(N * 4 - 1 downto 0)
    );
end compteurBCD;
architecture arch of compteurBCD is
begin
    assert N = 2 report " ne fonctionne que pour N = 2" severity failure;

    -- votre code ici

end arch;
```

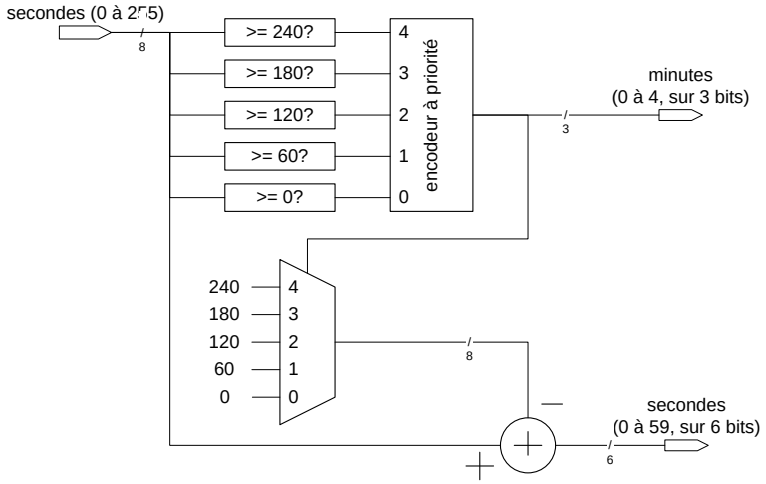
0503 Représentation binaire de nombres entiers et opérations arithmétiques de base

6. Donnez la valeur des expressions suivantes. Montrez les opérations à faire en binaire. Utilisez, selon le cas, le décalage ou l'extraction de bits de poids faible.

-6×4	$121 \% 16$
$48 \div 2$	$-99 \% 32$

0505 Décodeurs et encodeurs

7. Considérez le diagramme suivant pour un circuit numérique combinatoire qui accepte en entrée un vecteur de 8 bits représentant un nombre positif de secondes. Le circuit a deux sorties donnant le nombre de minutes et de secondes correspondantes. Par exemple, pour une entrée de 181 secondes, on devrait avoir en sortie 3 minutes et 1 seconde. Donnez une architecture en VHDL synthétisable correspondant à cette entité et à ce diagramme.

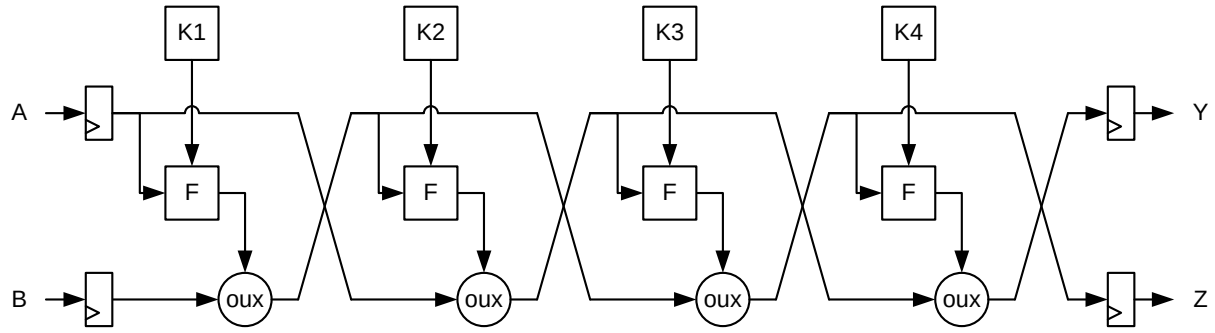



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity convSecondes is
port (
secondesIn : in unsigned(7 downto 0);
minutesOut : out unsigned(2 downto 0);
secondesOut : out unsigned(5 downto 0)
);
end convSecondes;
```

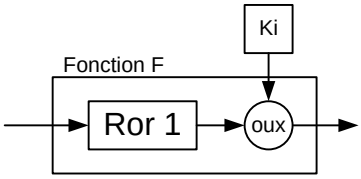
Objectifs spécifiques de la semaine

8. Le réseau de Feistel est utilisé dans les algorithmes de chiffrement par bloc. Plusieurs algorithmes utilisent le réseau de Feistel, dont DES, Blowfish et RC5. Le diagramme suivant illustre un réseau de Feistel simple à quatre étages. Les algorithmes cryptographiques basés sur un réseau de Feistel diffèrent principalement dans le nombre d'étages et dans la nature de la fonction F.



Le message à chiffrer est décomposé en un flux de nombres appliqués aux entrées A et B du réseau à chaque coup d'horloge. Les quatre clés secrètes K1, K2, K3 et K4 restent normalement constantes pour le chiffrement du message. Les sorties Y et Z sont un flux de nombres (une sortie par coup d'horloge) représentant le message chiffré. On constate qu'à chaque étage le signal du haut est combiné à la clé par la fonction F. On effectue ensuite un ou-exclusif bit à bit avec le signal du bas. À la fin de l'étage, les signaux du haut et du bas sont interchangés pour le prochain étage.

Supposez ici que tous fils du diagramme représentent un signal de 16 bits. Supposez que la fonction F consiste à appliquer une rotation de l'entrée de 1 bit vers la droite puis à effectuer l'opération ou-exclusif bit à bit avec la clé K_i exprimée sur 16 bits.



Considérez la déclaration d'entité suivante pour le diagramme du réseau de Feistel. Donnez une architecture en VHDL synthétisable correspondant à cette entité et au diagramme.

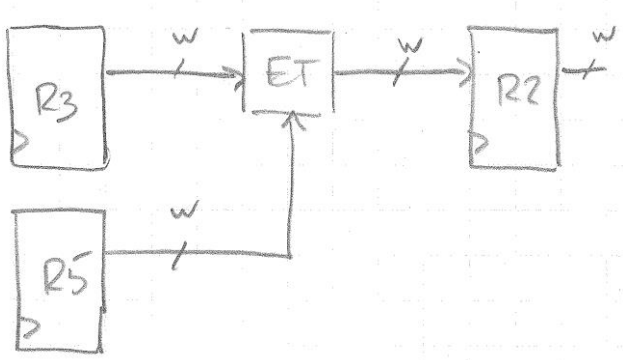
```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
entity feistel4 is
   generic (W : positive := 16);
   port (
      clk, reset : in std_logic;
      A, B : in unsigned(W - 1 downto 0);
      Y, Z : out unsigned(W - 1 downto 0);
      K1, K2, K3, K4: in unsigned(W – 1 downto 0)
   );
end feistel4;
```

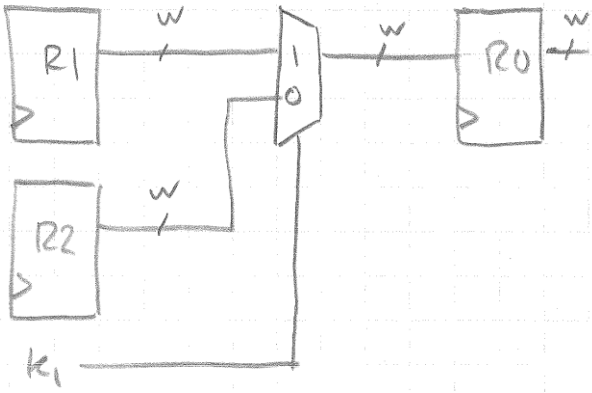
Solutions

1.

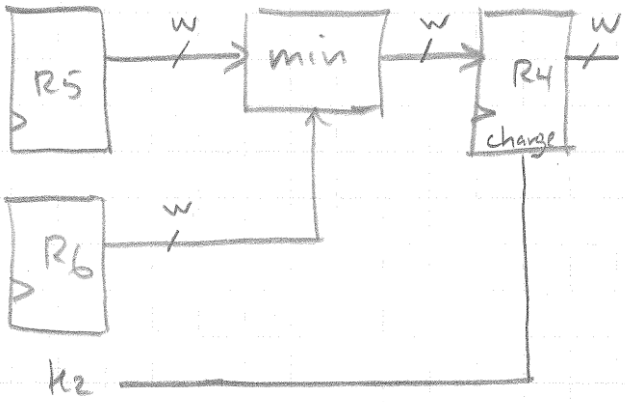
a.



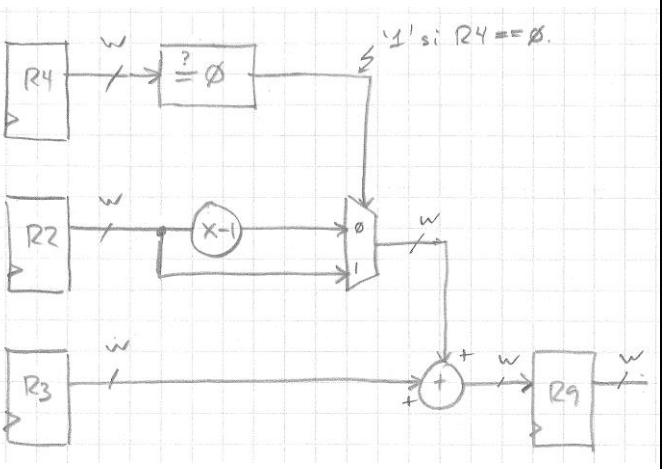
b.



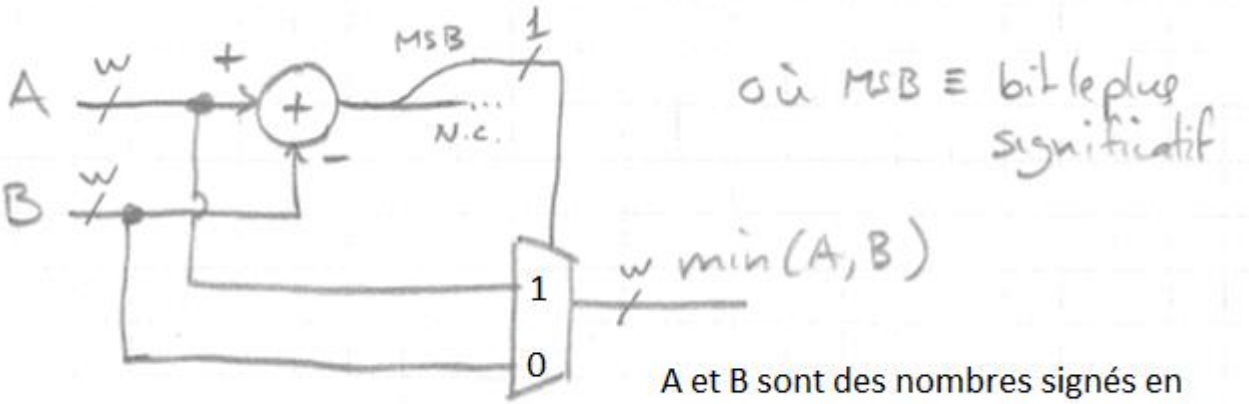
c.



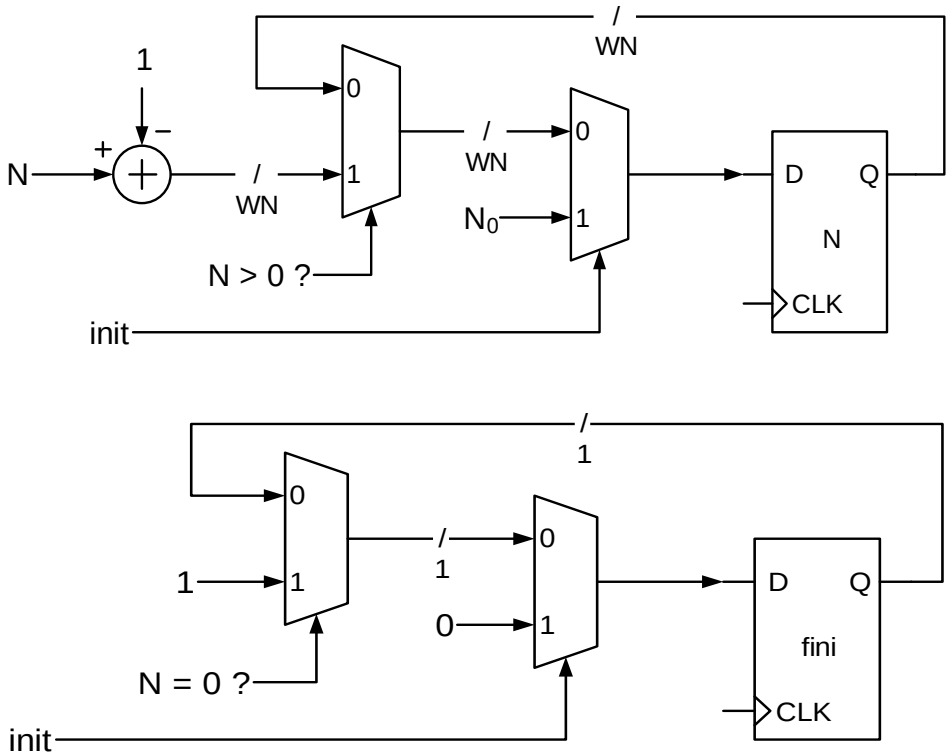
d.



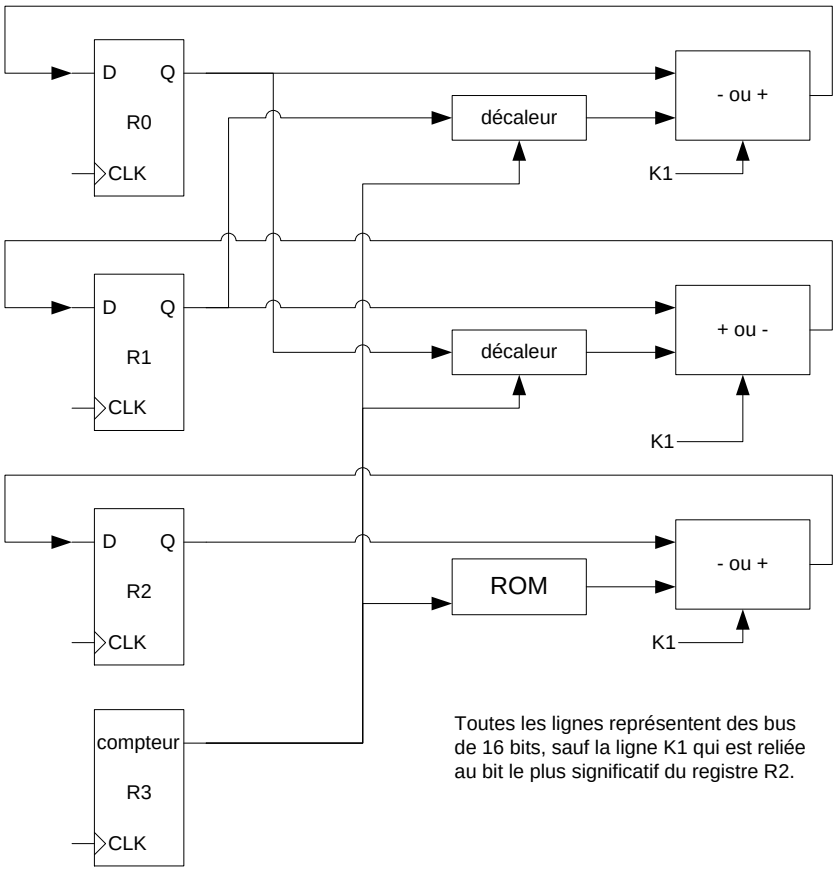
c. détail



2. Solution :



3. Solution



4. Solution à venir

5. Solution

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

entity compteurBCD is
    generic (
        N : positive := 2 -- nombre de chiffres du compteur
    );
    port (
        reset, clk : in std_logic;
        sortie : out unsigned(N * 4 - 1 downto 0)
    );
end compteurBCD;

architecture arch of compteurBCD is

begin

    assert N = 2 report "cette architecture ne fonctionne que pour N = 2" severity failure;

    process(clk) is
    variable lecompte_0 : natural range 0 to 9;
    variable lecompte_1 : natural range 0 to 9;
    begin
        if rising_edge(clk) then
            if reset = '1' then
                lecompte_0 := 0;
                lecompte_1 := 0;
            else
                if lecompte_0 = 9 then
                    lecompte_0 := 0;
                    if lecompte_1 = 9 then
                        lecompte_1 := 0;
                    else
                        lecompte_1 := lecompte_1 + 1;
                    end if;
                else
                    lecompte_0 := lecompte_0 + 1;
                end if;
            end if;
        end if;
    sortie(7 downto 4) <= to_unsigned(lecompte_1, 4);
    sortie(3 downto 0) <= to_unsigned(lecompte_0, 4);
    end process;

end arch;
```

6. Solutions

a. 1010 << 2 = 101000, soit -24 ok.

b. 0110000 >> 1 = 011000, soir +24, ok.

c. 01111001 % 16 = 0111[1001] = 9 ok

d. 10011101 % 32 = 100[11101] = 11101 = 29. C'est le bon résultat parce que -99 = -4 × 32 + 29. Attention, il y a plusieurs définitions du modulo quand des nombres négatifs sont en jeu. La sélection de bits de poids faible donne le bon résultat quand le dividende est un nombre positif. Pour plus d'informations voir entre autres http://fr.wikipedia.org/wiki/Modulo_(opération).

7. Solution, conversion de secondes

```
architecture arch of convSecondes is
begin
   process(secondesIn) is
   variable minutesT : natural range 0 to 4;
   variable secondesT : natural range 0 to 59;
   begin
      if secondesIn >= 240 then
         minutesT := 4;
         secondesT := to_integer(secondesIn) - 240;
      elsif secondesIn >= 180 then
         minutesT := 3;
         secondesT := to_integer(secondesIn) - 180;
      elsif secondesIn >= 120 then
         minutesT := 2;
         secondesT := to_integer(secondesIn) - 120;
      elsif secondesIn >= 60 then
         minutesT := 1;
         secondesT := to_integer(secondesIn) - 60;
      else
         minutesT := 0;
         secondesT := to_integer(secondesIn);
      end if;
      minutesOut <= to_unsigned(minutesT, 3);
      secondesOut <= to_unsigned(secondesT, 6);
   end process;
end arch;
```

8. Feistel solution :

Solution a 4 étages :

```
arch monarch of feistel4 is
type unsigned2D is array(natural range <>) of unsigned(W - 1 downto 0);
signal A0, A1, A2, A3, A4, B0, B1, B2, B3, B4 : unsigned(W - 1 downto 0);
begin
   -- registres à l'entrée pour A et B et à la sortie pour Y et Z
   process (clk, reset)
   begin
      if reset = '1' then
         Y <= (others => '0');
         Z <= (others => '0');
         A0 <= (others => '0');
         B0 <= (others => '0');
      elsif rising_edge(clk) then
         A0 <= A;
         B0 <= B;
         Y <= A4;
         Z <= B4;
      end if;
   end process;

   -- description du réseau comme tel
   -- on pourrait aussi paramétrer A et B, puis utiliser une description
   -- avec une boucle (dans un process) ou bien avec énoncé generate
   B1 <= A0;
   A1 <= (A0 ror 1) xor K1 xor B0;

   B2 <= A1;
   A2 <= (A1 ror 1) xor K2 xor B1;

   B3 <= A2;
   A3 <= (A2 ror 1) xor K3 xor B2;

   B4 <= A3;
   A4 <= (A3 ror 1) xor K4 xor B3;

end monarch;
```

Solution a nombre d'étages arbitraire :

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

entity feistel is
    generic (W : positive := 16  ;
              N : positive := 4) ; -- nombre étage
    port (clk   : in std_logic ;
          reset : in std_logic ;
          A, B  : in unsigned(W-1 downto 0);
          K     : in unsigned(N*W-1 downto 0) ;
          Y, Z  : out unsigned(W-1 downto 0)) ;
end feistel;

architecture ng of feistel is
    signal A0, B0 : unsigned(W - 1 downto 0) ;
begin
    process (clk, reset)
        variable Ai, Bi, Fo : unsigned(W - 1 downto 0) ;
    begin
        if reset = '1' then
            A0 <= (others => '0');
            B0 <= (others => '0');
            Y  <= (others => '0');
            Z  <= (others => '0');
        elsif rising_edge(clk) then
            --
            A0 <= A;
            B0 <= B;
            -- Valeurs de sortie calculées itérativement au moyen de variables
            Fo := (A0 ror 1) xor K(W-1 downto 0) xor B0;
            Bi := A0 ;
            Ai := Fo;
            if N > 1 then
                for j in 2 to N loop
                    Fo := (Ai ror 1) xor K(W*j-1 downto W*(j-1)) xor Bi ;
                    Bi := Ai ;
                    Ai := Fo ;
                end loop ;
            end if ;
            Y <= Ai;
            Z <= Bi;
     end if;
end process;

end ng ;
```