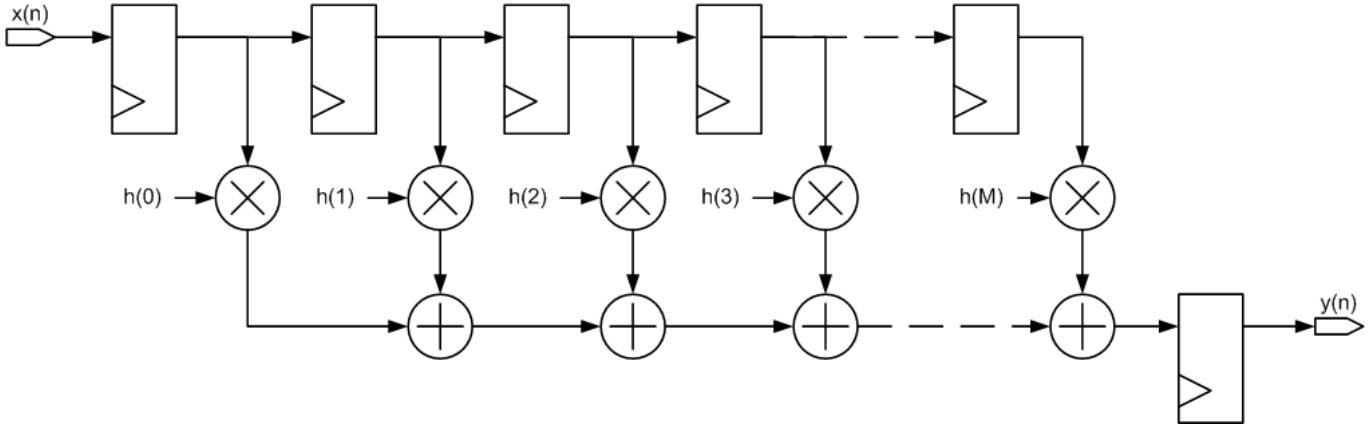
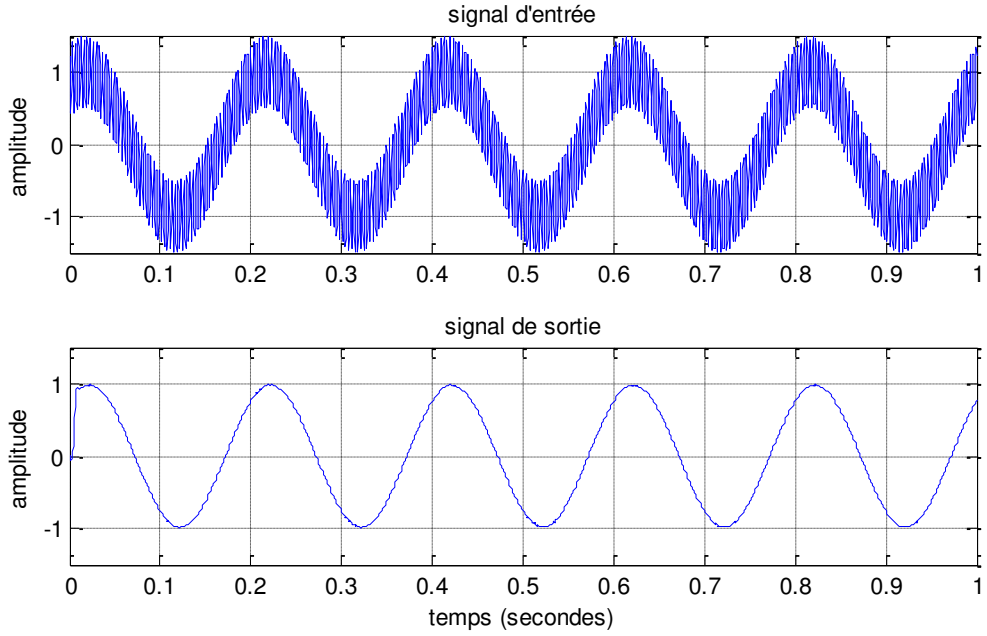
Exemple de design d'un filtre numérique

Exemple de conception: réduction du bruit dans un signal audio à l'aide d'un filtre

- On veut réduire le bruit dans un signal audio.
- On doit faire l'implémentation d'un filtre numérique à réponse impulsionnelle finie.
- Nous allons considérer trois designs:
  1. Un design de base
  2. Un design pipeliné qui maximise le débit
  3. Un design compact qui minimise la surface

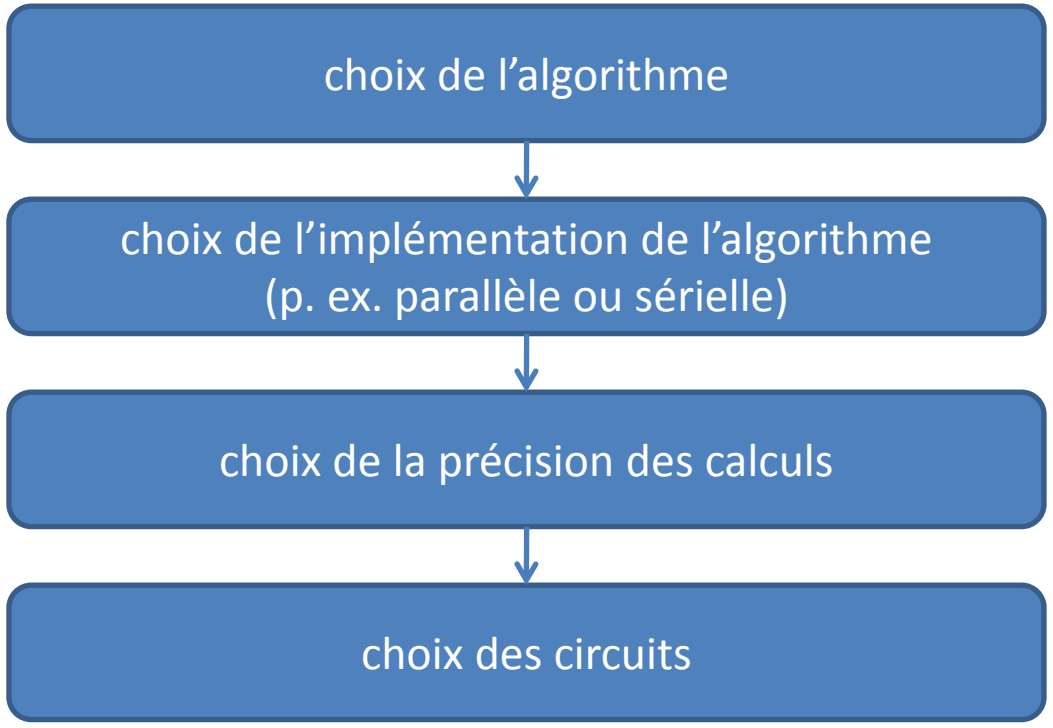




$$y(n) = \sum_{k=0}^{M} h(k)x(n-k)$$

Niveaux de compromis de design et leur impact

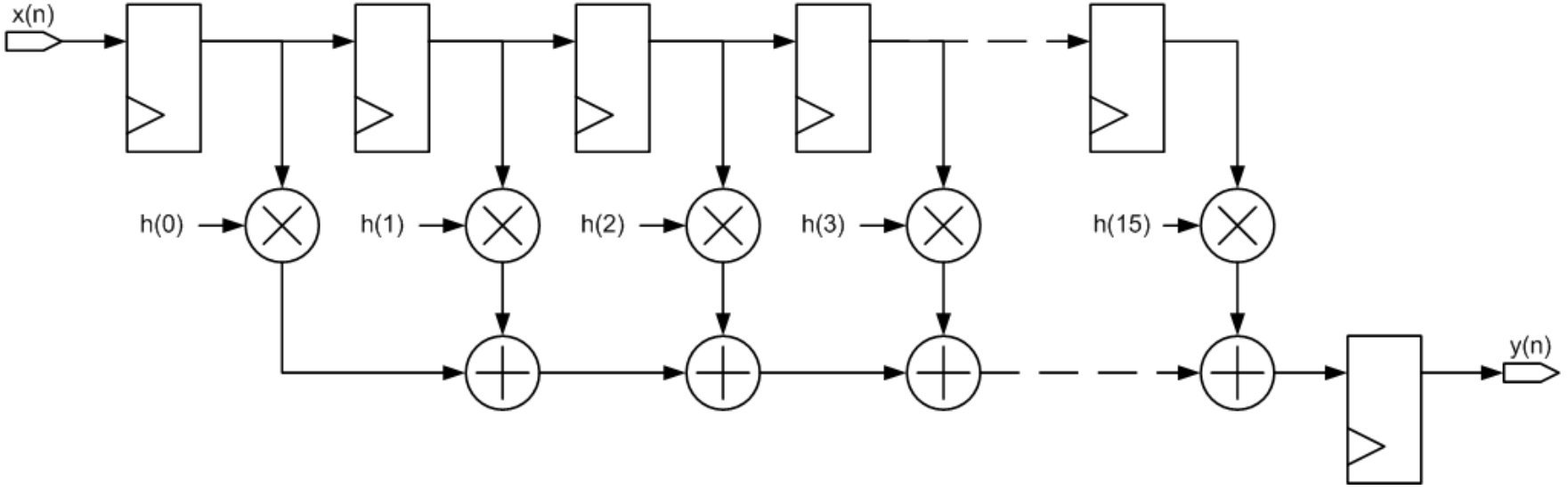
- En règle générale, plus une décision de design est prise à un haut niveau d'abstraction et plus son impact sera grand sur la performance et la surface d'un système.
- Une bonne décision prise à un bas niveau d'abstraction ne peut pas en général compenser pour une mauvaise décision prise à un haut niveau d'abstraction.



Design #1: forme directe de base

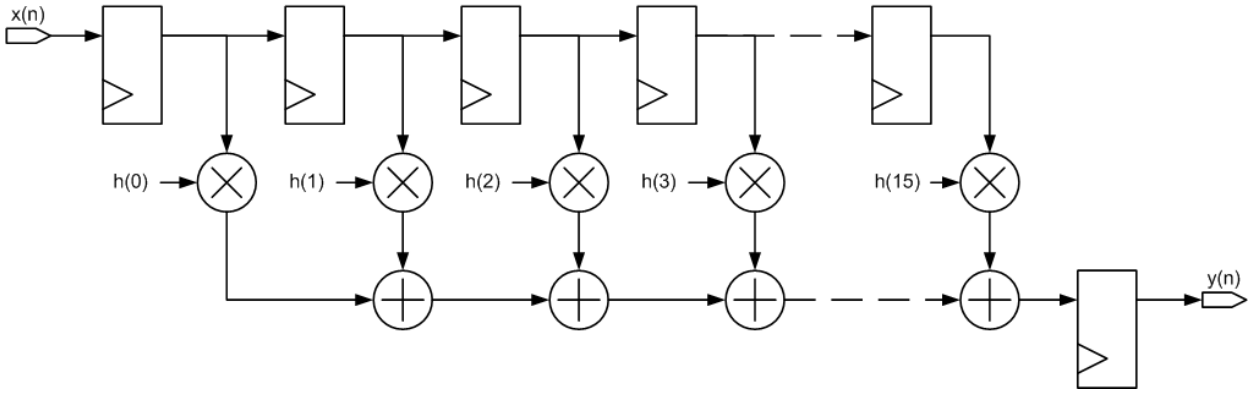
- L'entrée est exprimée sur 8 bits, la sortie sur 20.
- Au fil des calculs, plus de bits sont nécessaires pour représenter les résultats intermédiaires.
- Par exemple deux nombres de 8 bits multipliés ensemble donnent un résultat exprimé sur 16 bits.
- Les coefficients h(i) sont des constantes.

```
entity fir is
   generic (
      Win : positive := 8;
      Wout : positive := 20
   );
   port (
      reset_n, clk : in std_logic;
      x : in signed(Win - 1 downto 0);
      y : out signed(Wout - 1 downto 0)
   );
end fir;
```



Design #1: forme directe de base

- Il existe des algorithmes pour obtenir la valeur des coefficients h(k) en fonction de la réponse en fréquence désirée pour le filtre.
- Les coefficients montrés ici ont été obtenus à l'aide de MATLAB.

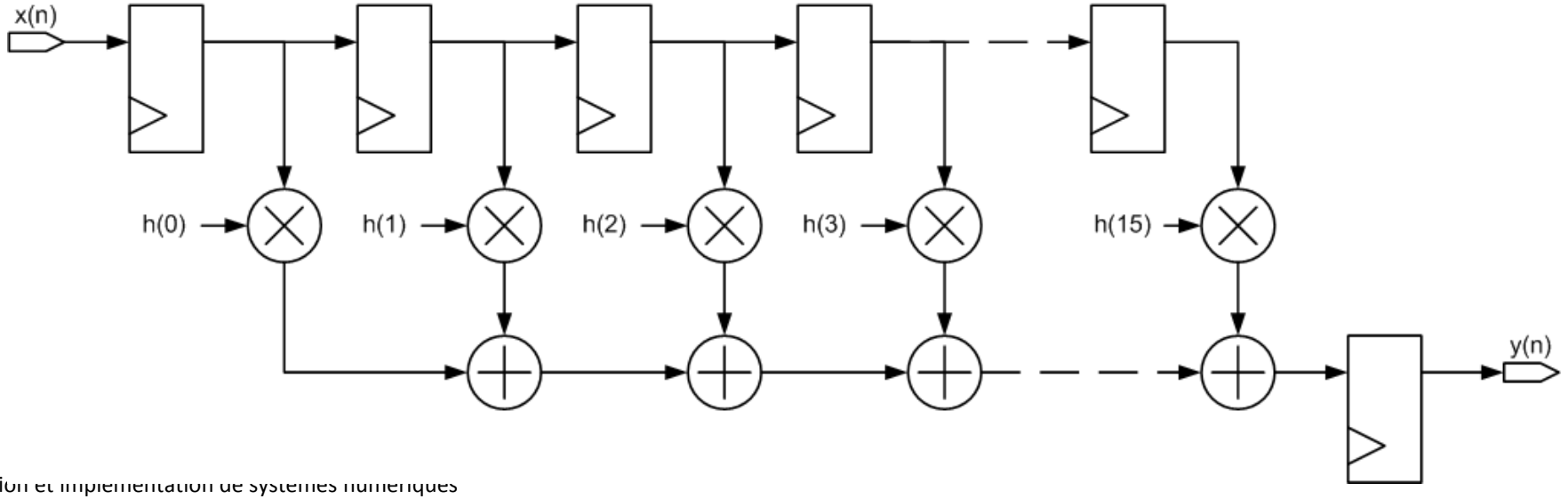



```
library IEEE;
use ieee.numeric_std.all;

package firpack is

   constant Win : integer := 8;
   constant Wout : integer := 20;

   type coefficients is array(natural range <>) of integer range -(2 ** (Win - 1)) to 2 ** (Win - 1);
   constant h : coefficients := (-12, 8, 17, 4, -20, -11, 47, 107, 107, 47, -11, -20, 4, 17, 8, -12);

   type ligneDelais is array(0 to h'length - 1) of signed(Win - 1 downto 0);
   type produits is array(0 to h'length - 1) of signed(2 * Win - 1 downto 0);

end firpack;
```

Design #1: forme directe de base modélisation de la ligne de délais

- La ligne de délai est composée de 16 registres en série pour entreposer des versions retardées de x(n), x(n-1), x(n-2), …, x(n- 15).

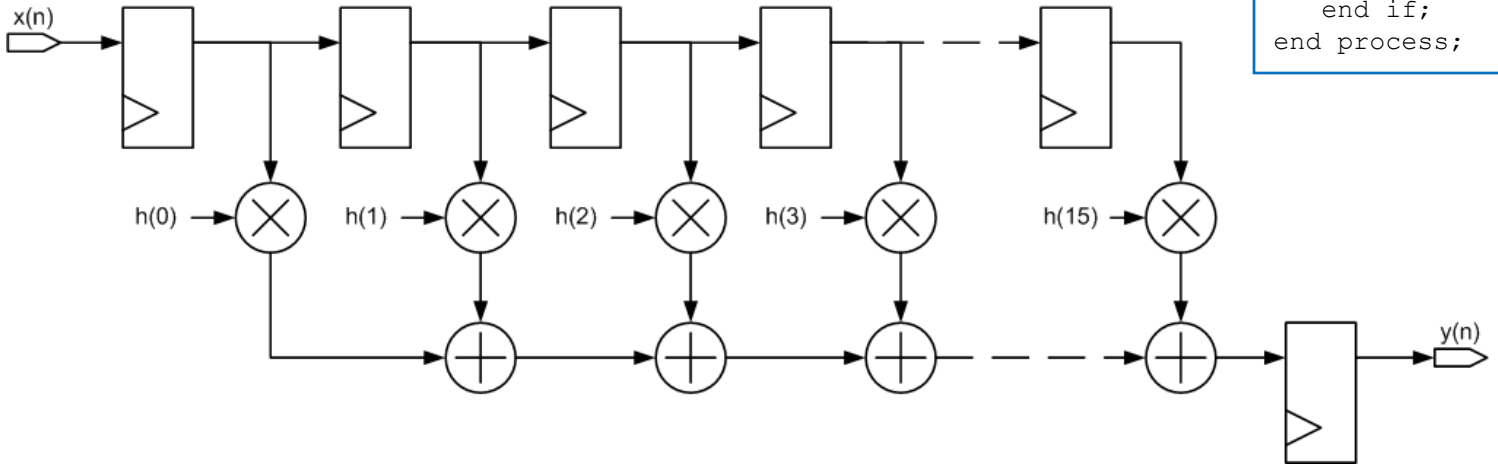
```
-- ligne de délais
process(clk)
begin
   if rising_edge(CLK) then
      if (reset_n = '0') then
         xDelai <= (others => (others => '0'));
      else
         for k in xDelai'length - 1 downto 1 loop
            xDelai(k) <= xDelai(k - 1);
         end loop;
         xDelai(0) <= x;
      end if;
   end if;
end process;
```



Design #1: forme directe de base modélisation des produits et de leur accumulation

- Les produits sont calculés et accumulés.
- Une boucle simplifie l'écriture mais n'implique pas de séquence de calcul: la boucle peut être déroulée de façon statique et tout est concurrent.

```
-- génération et addition des produits, et sortie finale
-- y <= h(0) * xDelai(0) + h(1) * xDelai(1) + ... + h(15) * xDelai(15);

process(clk)
variable somme : integer := 0;
begin
   if rising_edge(CLK) then
      if reset_n = '0' then
         y <= (others => '0');
      else
         somme := 0;
         for k in 0 to xDelai'length - 1 loop
            somme := somme + to_integer(h(k) * xDelai(k));
         end loop;
         y <= to_signed(somme, y'length);
      end if;
   end if;
end process;
```

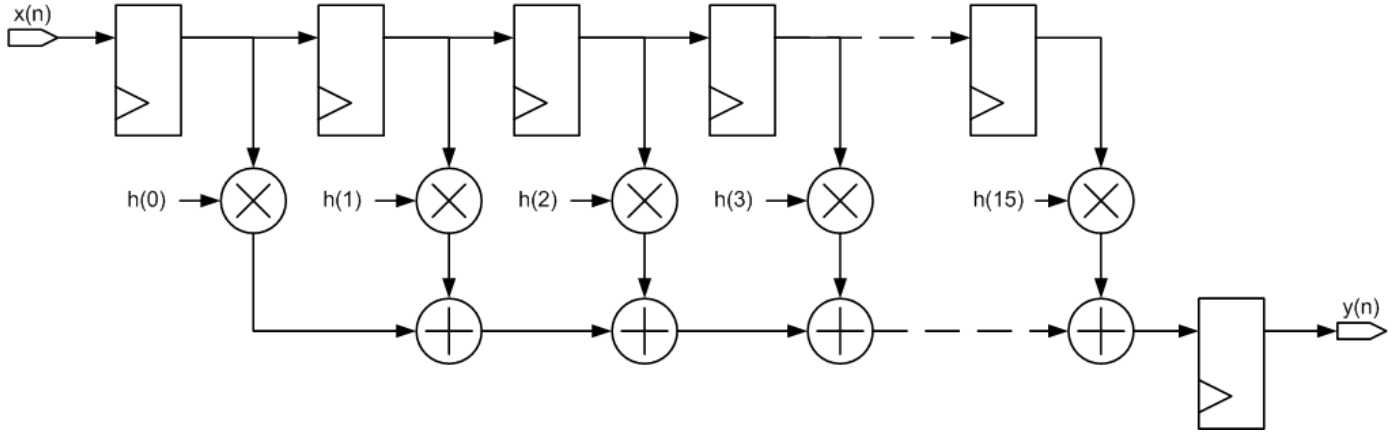


Latence de 1 registre, 1 cycle.
(On ne compte pas le premier registre dans lequel on place la donnée en entrée)

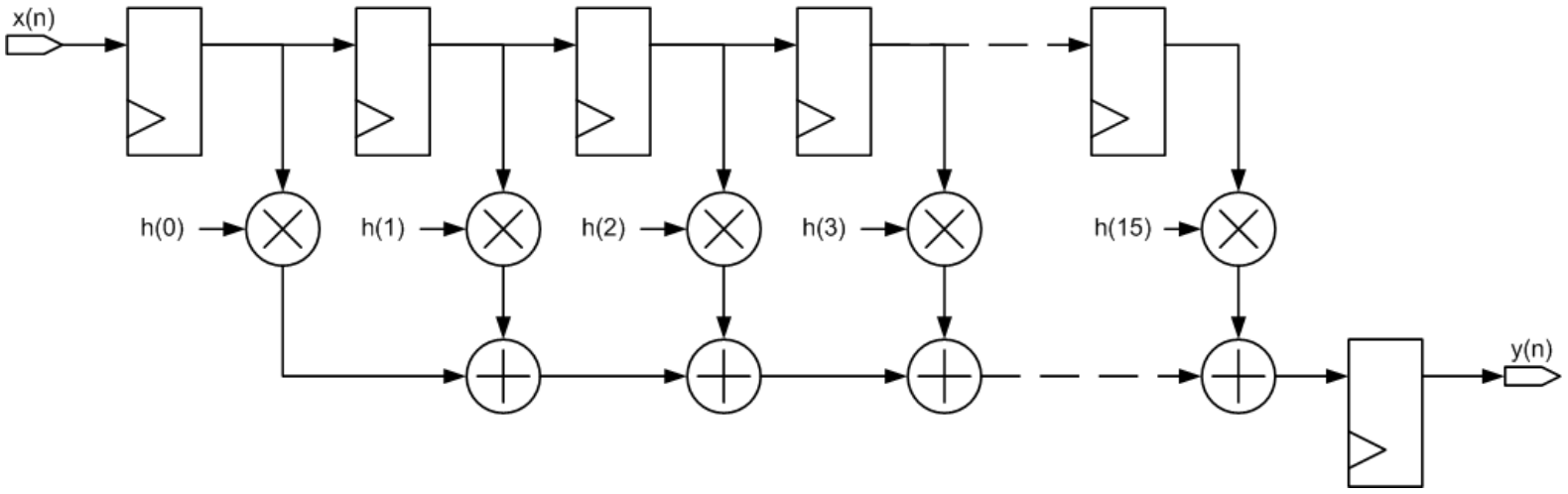
Design #1: forme directe de base résultats de synthèse

- Ressources utilisées:
  - Number of Slices: 190 out of 13696
  - Number of Slice Flip Flops: 148 out of 27392
  - Number of 4 input LUTs: 270 out of 27392
  - Number of MULT18X18s: 12 out of 136
- Période minimale d'horloge et chemin critique
  - Minimum period: 25.436ns (Maximum Frequency: 39.314MHz)
  - Source: xDelai_0_0 (FF)
  - Destination: y_19 (FF)
  - Data Path Delay: 25.436ns (Levels of Logic = 40)
- Latence:
  - 1 cycle

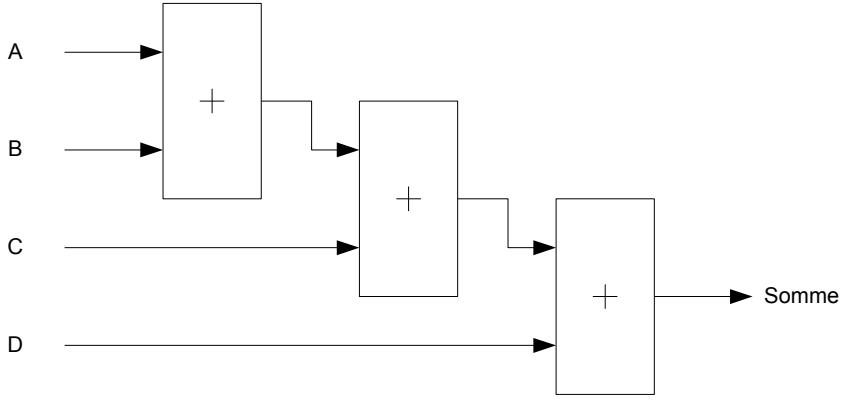
```
library IEEE;
use ieee.numeric_std.all;

package firpack is

    constant Win : integer := 8;
    constant Wout : integer := 20;

    type coefficients is array(natural range <>) of integer range -(2 ** (Win - 1)) to 2 ** (Win - 1);
    constant h : coefficients := (-12, 8, 17, 4, -20, -11, 47, 107, 107, 47, -11, -20, 4, 17, 8, -12);

    type ligneDelais is array(0 to h'length - 1) of signed(Win - 1 downto 0);
    type produits is array(0 to h'length - 1) of signed(2 * Win - 1 downto 0);

end firpack;
```

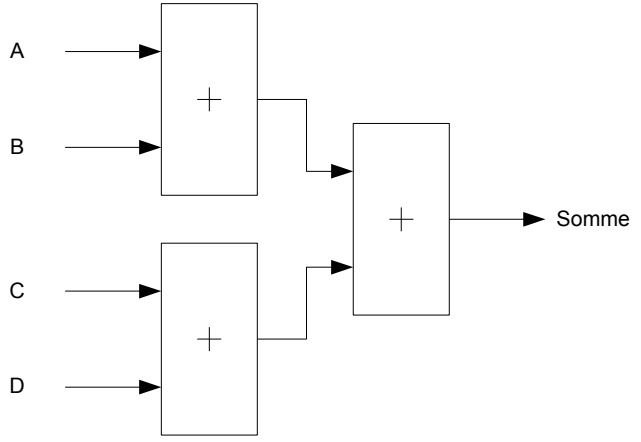


Design #2: maximiser le débit



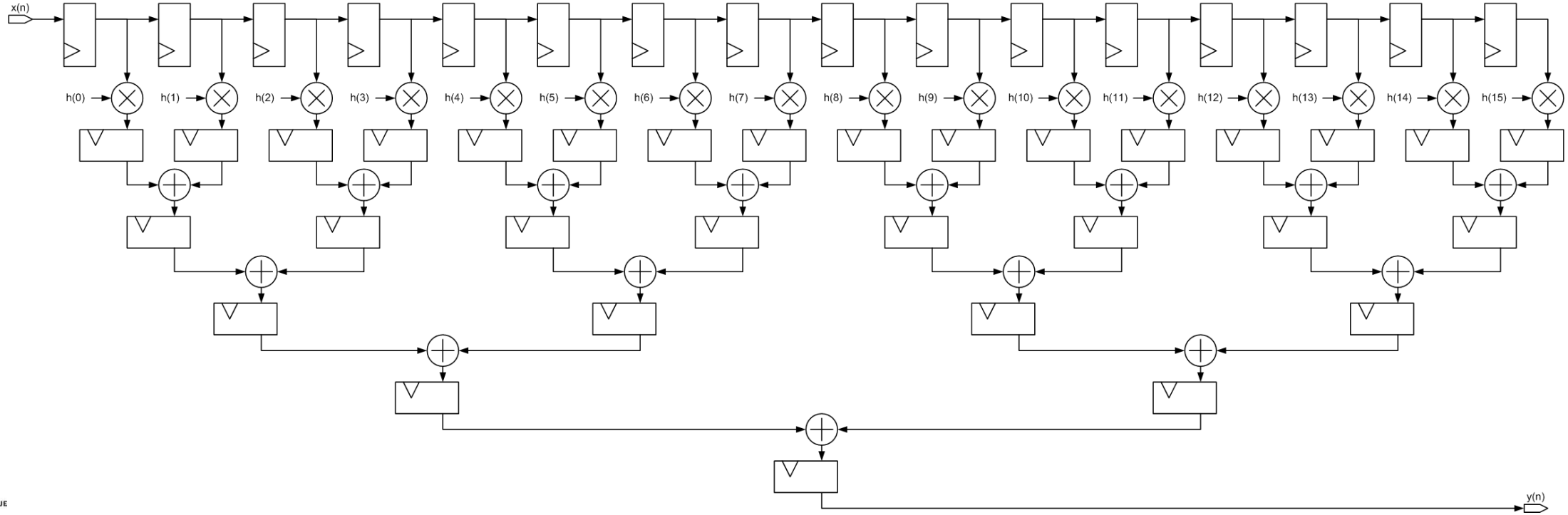
Défi:
Proposer une architecture pour maximiser le débit.





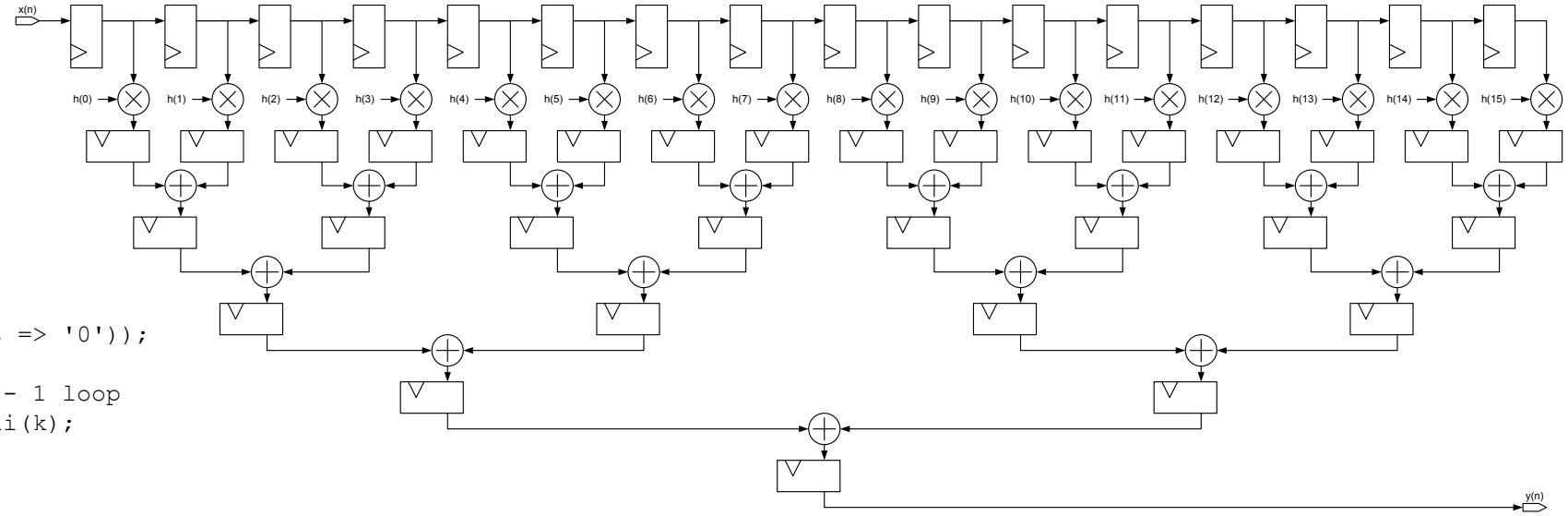
Design #2: forme directe avec les produits pipelinés et un arbre binaire d'additions pipeliné

- Chaque produit est placé dans un registre.
- Les produits sont accumulés deux à la fois, ce qui correspond aux modules d'addition sur les FPGA.
- Chaque somme est placée dans un registre.
- Dans cette version, le chemin critique est le plus court possible, mais il n'est pas clair à partir du schéma où celui-ci se trouve.



Design #2: forme directe avec les produits pipelinés et un arbre binaire d'additions pipeliné

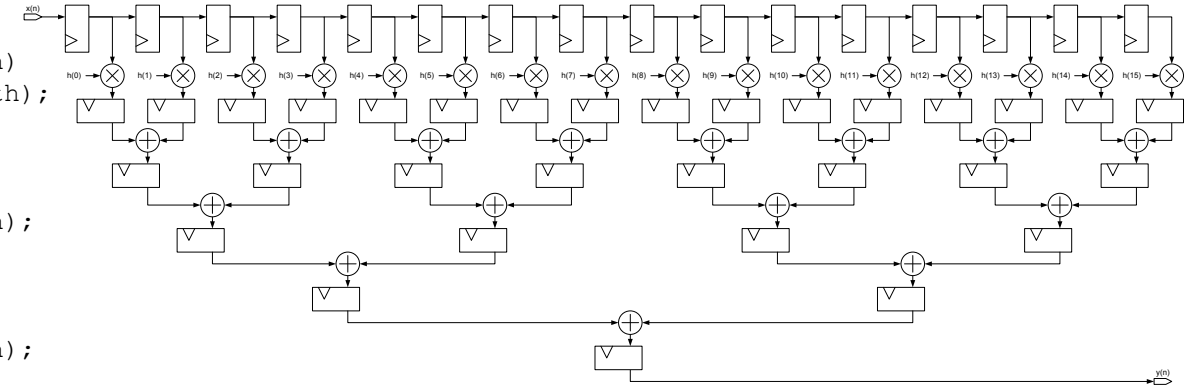
```
-- ligne de délais
process(clk)
begin
   if rising_edge(CLK) then
      if (reset_n = '0') then
         xDelai <= (others => (others => '0'));
      else
         for k in xDelai'length - 1 downto 1 loop
            xDelai(k) <= xDelai(k - 1);
         end loop;
         xDelai(0) <= x;
      end if;
   end if;
end process;


-- registres des produits
process (clk)
begin
   if rising_edge(CLK) then
      if (reset_n = '0') then
         lesproduits <= (others => (others => '0'));
      else
         for k in 0 to lesproduits'length - 1 loop
            lesproduits(k) <= h(k) * xDelai(k);
         end loop;
      end if;
   end if;
end process;
```



Design #2: forme directe avec les produits pipelinés et un arbre binaire d'additions pipeliné

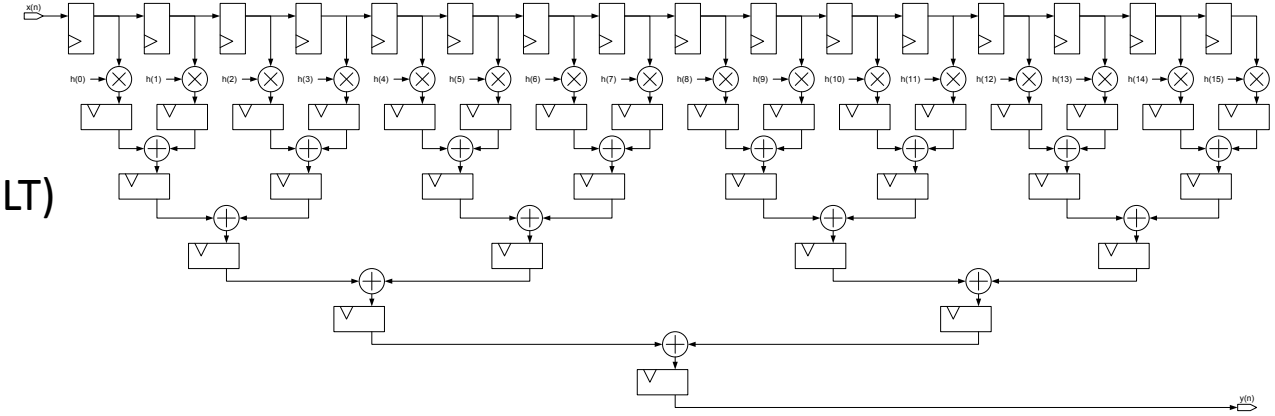
```
type addniv1 is array(0 to h'length / 2 - 1) of signed(2 * Win - 1 + 1 downto 0);
type addniv2 is array(0 to h'length / 4 - 1) of signed(2 * Win - 1 + 2 downto 0);
type addniv3 is array(0 to h'length / 8 - 1) of signed(2 * Win - 1 + 3 downto 0);
```

```
-- addition des produits avec un arbre binaire pipeliné et sortie finale
process(clk)
variable somme : integer := 0;
begin
   if rising_edge(clk) then
      if reset_n = '0' then
         lesaddniv1 <= (others => (others => '0'));
         lesaddniv2 <= (others => (others => '0'));
         lesaddniv3 <= (others => (others => '0'));
      else
         for k in 0 to h'length / 2 - 1 loop
            lesaddniv1(k) <= resize(lesproduits(2 * k), lesaddniv1(k)'length)
                           + resize(lesproduits(2 * k + 1), lesaddniv1(k)'length);
         end loop;
         for k in 0 to h'length / 4 - 1 loop
            lesaddniv2(k) <= resize(lesaddniv1(2 * k), lesaddniv2(k)'length)
                           + resize(lesaddniv1(2 * k + 1), lesaddniv2(k)'length);
         end loop;
         for k in 0 to h'length / 8 - 1 loop
            lesaddniv3(k) <= resize(lesaddniv2(2 * k), lesaddniv3(k)'length)
                           + resize(lesaddniv2(2 * k + 1), lesaddniv3(k)'length);
         end loop;
      end if;
   end if;
   somme := to_integer(lesaddniv3(0) + lesaddniv3(1));
   y <= to_signed(somme, y'length);
end process;
```

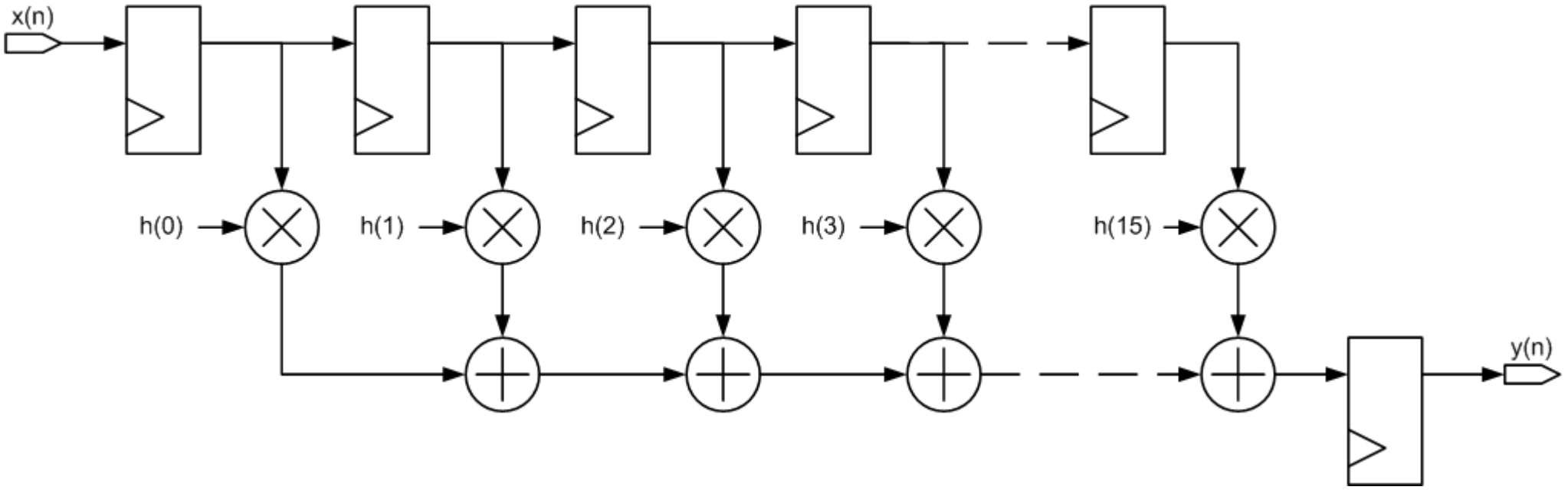


Design #2
Résultats de la synthèse

- Ressources utilisées:
  - Number of Slices: 204 out of 13696 1%
  - Number of Slice Flip Flops: 360 out of 27392 1%
  - Number of 4 input LUTs: 242 out of 27392 0%
  - Number of MULT18X18s: 12 out of 136 8%
- Période minimale d'horloge et chemin critique
  - Minimum period: 3.166ns
    (Maximum Frequency: 315.856MHz)
  - Source: Mmult_lesproduits_8_mult0000 (MULT)
  - Destination: lesaddniv1_4_16 (FF)
  - Data Path Delay: 3.166ns (Levels of Logic = 18)
- Latence:
  - 5 cycles



Design #3:
forme compacte

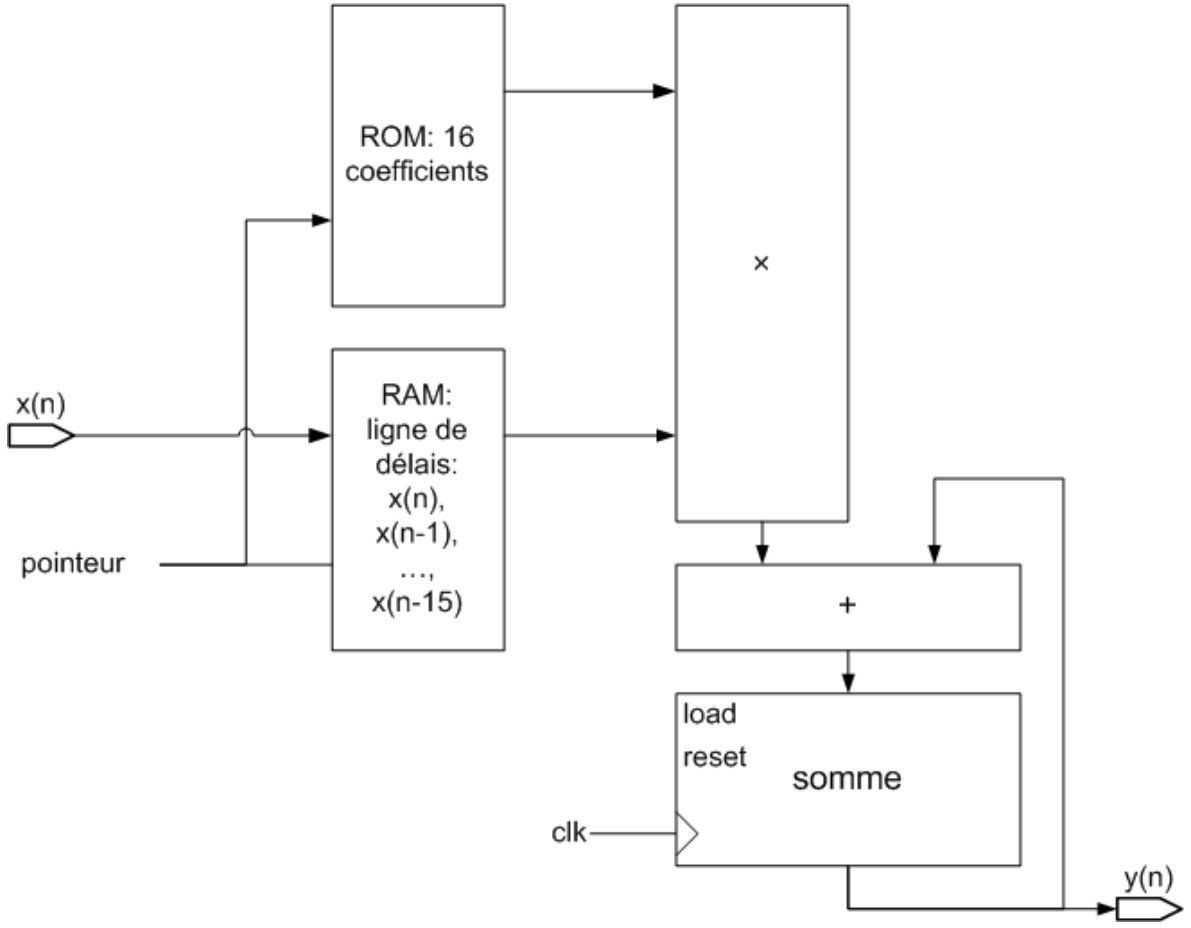


Défi:
Proposer une architecture pour minimiser la surface.
Conseil: utiliser un seul multiplicateur et un seul additionneur.

Design #3: forme compacte chemin des données

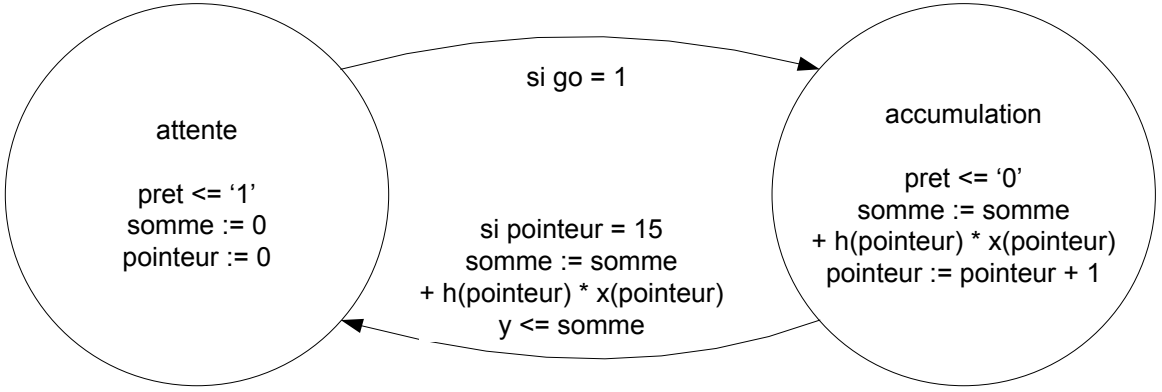
- Un seul multiplicateur.
- Même ligne de délais que les deux autres designs.
- Un pointeur permet d'accéder chaque donnée x et chaque coefficient, un à la fois.
- Une unité de contrôle doit régler la séquence des calculs ainsi que l'échange des données x(n) et y(n) avec le monde extérieur.

```
-- ligne de délais
process(clk)
begin
   if rising_edge(CLK) then
      if (reset_n = '0') then
         xDelai <= (others => (others => '0'));
      elsif (etat = attente) and (go = '1') then
         for k in xDelai'length - 1 downto 1 loop
            xDelai(k) <= xDelai(k - 1);
         end loop;
         xDelai(0) <= x;
      end if;
   end if;
end process;
```



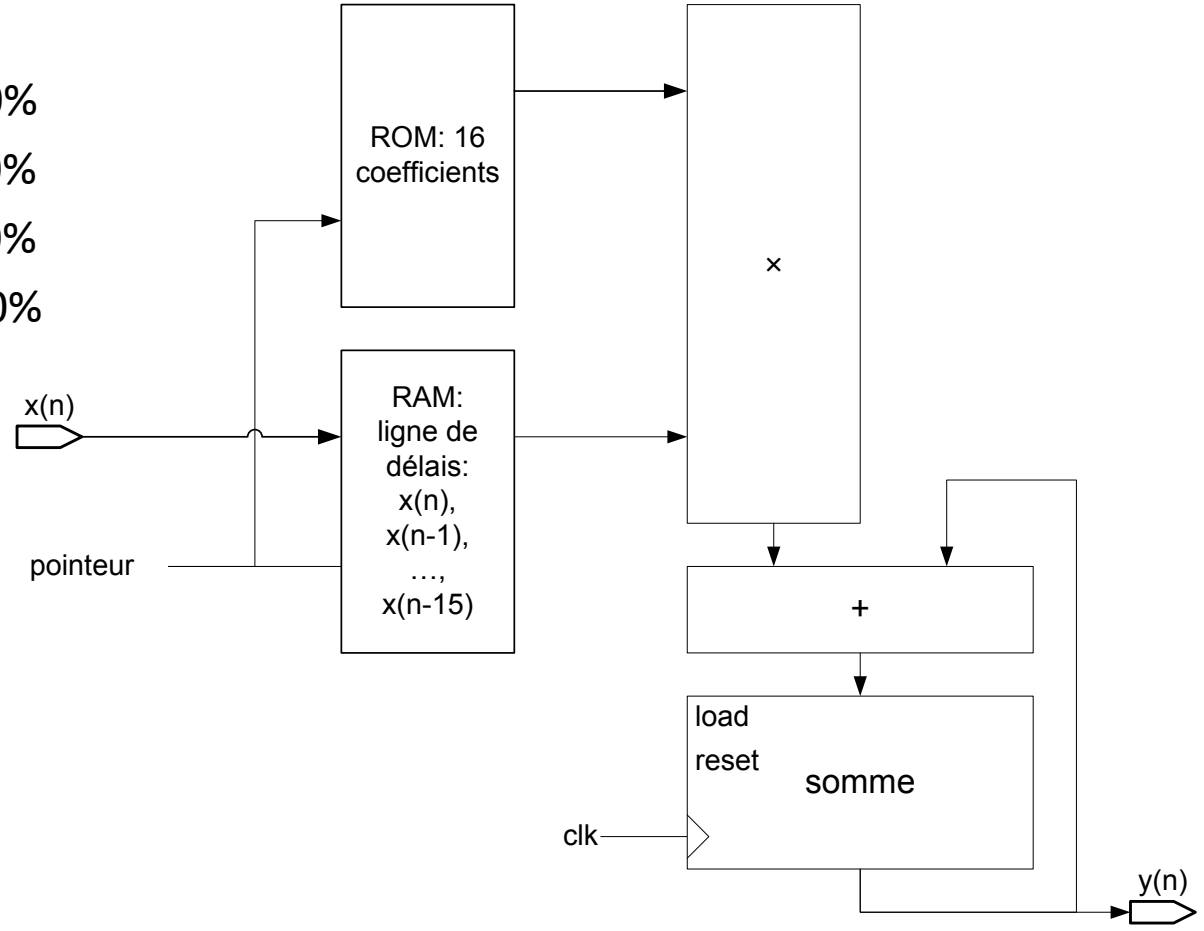
Design #3: forme compacte modélisation VHDL

```
type type_etat is (attente, accumulation);
signal etat : type_etat;
   -- ...

-- chemin des données intégré à l'unité de contrôle
controle : process (CLK)
variable pointeur : integer range 0 to 15;
variable somme : integer;
begin
   if rising_edge(CLK) then
      if reset_n = '0' then
         etat <= attente;
         y <= (others => '0');
      else
         case etat is
            when attente =>
               pointeur := 0;
               somme := 0;
               if go = '1' then
                  etat <= accumulation;
               end if;
            when accumulation =>
               somme := somme
               + to_integer(h(pointeur) * xDelai(pointeur));
               if pointeur = 15 then
                  etat <= attente;
                  y <= to_signed(somme, y'length);
               else
                  pointeur := pointeur + 1;
               end if;
            when others =>
               etat <= attente;
            end case;
      end if;
   end if;
end process;

pret <= '1' when etat = attente else '0';
```



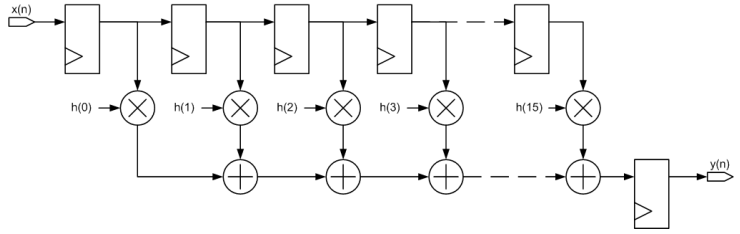
Design #3: forme compacte
Résultats de la synthèse

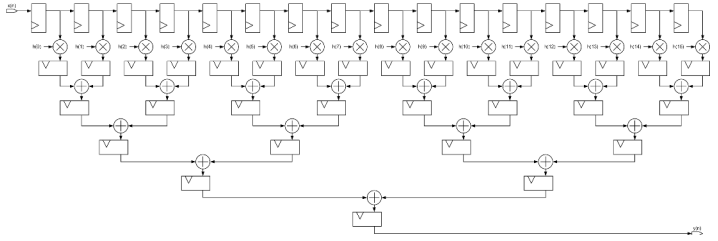
- Ressources utilisées:
  - Number of Slices: 127 out of 13696 0%
  - Number of Slice Flip Flops: 173 out of 27392 0%
  - Number of 4 input LUTs: 124 out of 27392 0%
  - Number of MULT18X18s: 1 out of 136 0%
- Période minimale d'horloge et chemin critique
  - Minimum period: 7.420ns (Maximum Frequency: 134.768MHz)
  - Source: pointeur_0 (FF)
  - Destination: y_19 (FF)
  - Data Path Delay: 7.420ns (Levels of Logic = 11)
- Latence:
  - 17 cycles

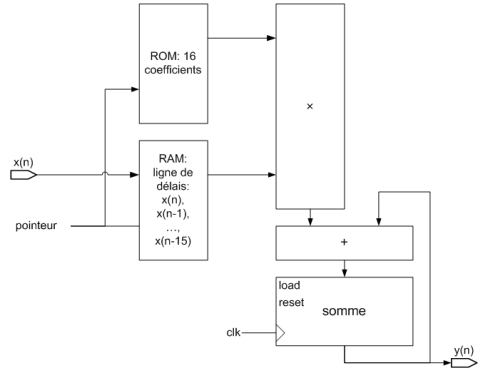


Sommaire

- Métrique Coût × Débit^{-1} (un petit nombre est bon).
- v.1: Coût = nombre de LUTs.
  Le design #2 est le meilleur selon cette métrique.
- v. 2: Coût = #LUT + #FF + 100 × #Mul.
  Le design #2 est encore le meilleur.
- La morale: pour les FPGA, il est très avantageux de pipeliner un design au maximum, à moins d'avoir à rencontrer des contraintes très strictes d'utilisation des ressources.



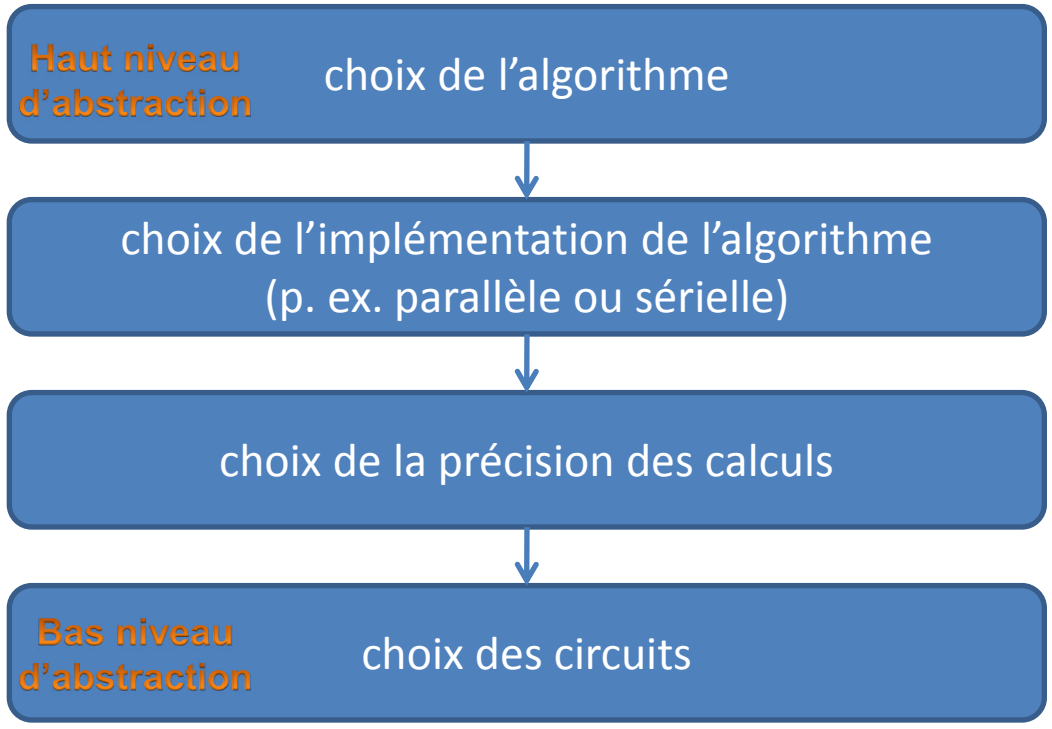




design	détails	LUT	FF	Mul	T_{min} (ns)	Débit^{-1} (cycles/ résultat)	Débit^{-1} (ns/résultat)	Latence (cycles)	Latence (ns)	C×D v. 1	C×D v.2
1	base	270	148	12	24.4	1	24.4	1	24.4	6.59	39
2	pipeliné	242	360	12	3.17	1	3.17	5	15.9	0.77	5.7
3	compact	124	173	1	7.42	17	126	17	126.1	15.6	50

En conclusion …

- Pour le problème de l'implémentation d'un filtre, on peut considérer les décisions à prendre aux quatre niveaux vus précédemment:



Quel type de filtre?

Forme pipelinée ou compacte?

Combien de bits de précision pour les entrées et pour les résultats intermédiaires?

Quels types d'additionneurs et de multiplicateurs?