Systèmes digitaux

Cours 11

Algorithmes complexes

- Il existe plusieurs opérations définies dans le langage VHDL:
  - +, -, *, division par les puissances de 2, les comparaisons, les fonctions logiques, etc.
- Certaines opérations sont plus complexes et demandent une approche différente:
  - Look-up table
  - Algorithme multi-cycle
  - Algorithmes itératifs

Look-up table

- Certaines opérations sont moins évidentes à implanter:
    - Exponentielle, sinus, cosinus, tangente, etc.
- Il est possible d'effectuer une série de Taylor…

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots$$

- Les calculs sont parfois longs et demandent des ressources

Look-up table

- Dans certaines cas, il est plus simple de créer une look-up table
- Par exemple: sin(x) pour x de 9 bits, où x est en degrés et où sin(x) est de 8 bits format
  - Il faudrait faire un tableau du type CASE/WHEN contenant 512 possibilités

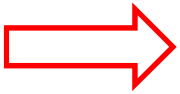
```
CASE x IS
    WHEN “000000000” =>
       sinx <= “00000000”;
    WHEN “000000001” =>
       sinx <= “00000100”;
…
END CASE;
```

Sin(x) allant de 0 à 1 sur 8 bits (0.XXXXXXXX)

Look-up table

- Il est évidemment possible d'écrire les 512 lignes à la main
  - Les informaticiens se serviraient plutôt d'un « script » ou d'un langage qui automatise les choses répétitifs
- En MATLAB, on pourrait commencer par:

```
for x=0:511
        sind(x)
end
```



```
ans =
     0
ans =
    0.0175
ans =
    0.0349
ans =
    0.0523
…
```

Look-up table

- Sachant que:
  - On veut une valeur entre 0 et 255
  - Et que sind(x) donne entre -1 et 1
- On pourrait:
  - Additionner 1 au résultat (pour devenir positif)
  - Multiplier par 127 et arrondir

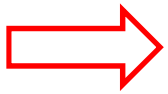
```
for x=0:511
        round((sind(x)+1)*127)
end
```

```
ans =
   127
ans =
   129
ans =
   131
ans =
   134
...
```

Look-up table

- Pour s'en servir en VHDL, il faut avoir le nombre en binaire:

```
for x=0:511
        sin_dec=round((sind(x)+1)*127);
        de2bi(sin_dec,8, 'left-msb')
end
```



```
ans =
     0     1     1     1     1     1     1     1
ans =
     1     0     0     0     0     0     0     1
ans =
     1     0     0     0     0     0     1     1
ans =
     1     0     0     0     0     1     1     0
ans =
     1     0     0     0     1     0     0     0
```

Fonction MATLAB

8 bits

MSB à gauche

Il suffit maintenant d'ajouter du texte de façon automatique…

Look-up table

- La fonction de2bi nous donnera un vecteur de 0 et de 1
- Pour s'en servir, il faut les mettre sous forme de chaîne de caractère en utilisant sprintf
- Ensuite, on l'insère dans une commande:

```
for x=0:511
        sin_dec=round((sind(x)+1)*127);
        vecteur=de2bi(sin_dec,8, 'left-msb');
        sin_bin=sprintf('%d', vect);
        partie_when=sprintf('WHEN %d =>\n', x);
        partie_assign=sprintf('sinx <= "%s";\n', sin_bin);
        disp(partie_when);
        disp(partie_assign);
end
```

Mettre sous forme VHDL

Montrer à l'écran

Look-up table

- Le résultat n'est pas exactement ce qu'on voulait:

```
WHEN 0 =>
sinx <= "01111111";
WHEN 1 =>
sinx <= "10000001";
WHEN 2 =>
sinx <= "10000011";
WHEN 3 =>
…
```

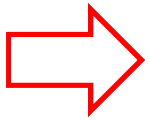
- Il faut changer la valeur du x en binaire…
  - On met x sur 9 bits, MSB à gauche
  - On forme une chaîne de caractère avec

```
x_vecteur = de2bi(x,9, 'left-msb');
x_bin = sprintf('%d', x_vecteur);
```

Look-up table

- Au final, ça donne ceci:
  - Ça semble être beaucoup de travail, mais il suffit de l'apprendre une fois…
  - Les scripts sont souvent très utiles

```
for x=0:511
        sin_dec=round((sind(x)+1)*127);
        vecteur=de2bi(sin_dec,8, 'left-msb');
        sin_bin=sprintf('%d', vect);
        x_vecteur = de2bi(x,9, 'left-msb');
        x_bin = sprintf('%d', x_vecteur);
        partie_when=sprintf('WHEN "%s" =>\n', x_bin);
        partie_assign=sprintf('sinx <= "%s";\n', sin_bin);
        disp(partie_when);
        disp(partie_assign);
end
```



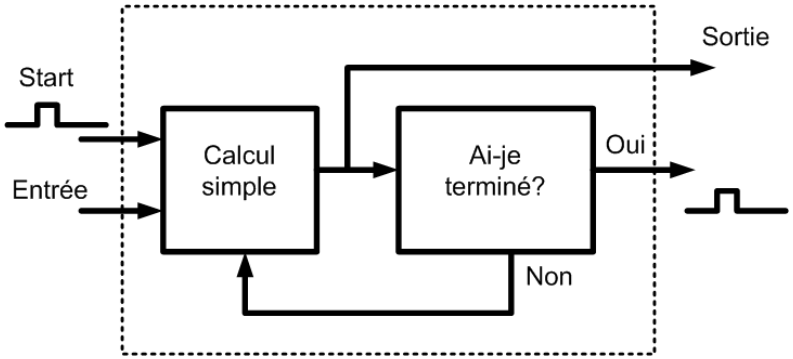
```
WHEN "000000000" =>
sinx <= "10111101";
WHEN "000000001" =>
sinx <= "10111101";
WHEN "000000010" =>
sinx <= "10111101";
WHEN "000000011" =>
sinx <= "10111101";
…
```

Algorithmes multi-cycles

- L'approche avec les look-up tables fonctionne sur 1 cycle d'horloge
- Certaines opérations peuvent être effectuées sur plusieurs cycles
    - On peut penser à la multiplication par addition successif
    - Ou à la division par soustraction successive

Algorithmes multi-cycles

- Les algorithmes multi-cycles peuvent avoir une structure comme celle-ci:



- On entre les données et on active le processus
  - Le processus continue AUTOMATIQUEMENT
  - Lorsque c'est terminé, il le signale et génère la sortie...

Algorithmes multi-cycles

- Un exemple simple serait la multiplication
  - Une multiplication peut se faire directement en VHDL
  - Peut aussi se faire par sucession d'additions
- En faisait 4 x 5, on se retrouve à faire:
  - 5+5+5+5
- Donc, on procèderait comme suit:
  - On additionne 5 (Additionneur)
  - Combien de fois a-t-il additionné 5? (Comparateur)
  - Si ça fait 4 fois, j'arrête.

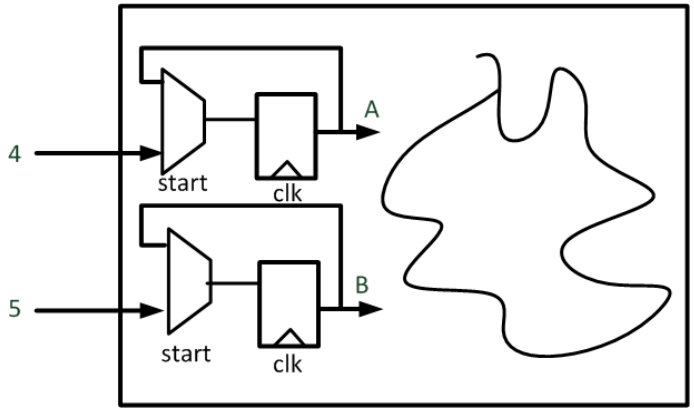
Algorithmes multi-cycles

- De façon plus structurée:
  - Quand Start=1, je sauvegarde les entrées
  - A=4 et B=5
  - Je mets produit=0
- À chaque coup d'horloge:
  - J'additionne B au produit
  - Je soustrais 1 de A
- Je continue jusqu'à ce que A soit 0

Pourquoi sauvegarder?

Algorithmes multi-cycles

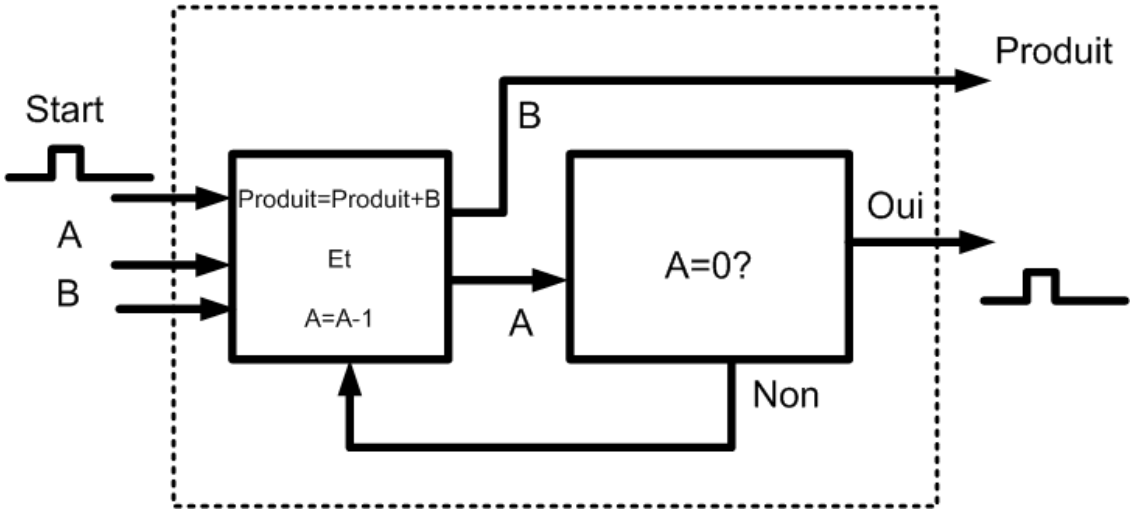
- Start=0
  - A=0, B=0, Produit=0
- Start=1
  - A=4, B=5, Produit=0



- Aux cycles d'horloge subséquents:
  - A=3, B=5, Produit=5
  - A=2, B=5, Produit=10
  - A=1, B=5, Produit=15
  - A=0, B=5, Produit=20

Algorithmes multi-cycles

- Le diagramme ressemblerait à ceci:



Algorithmes multi-cycles

- Pourquoi une pulsation à la sortie?
  - Les algorithmes multi-cycles prennent plusieurs cycles d'horloge
  - Le prochain module aimerait se servir de nos données
  - Mais comment savoir si la donnée est bonne ou pas?
- On peut générer un signal "Fin" pour dire que le calcul est terminé
  - Donc, que la sortie est bonne

Exemple (seul)

- On veut faire une division de deux nombres:
  - Le résultat donne Quotient et Reste
- Pensez à un algorithme multi-cycle pour la division
  - Qu'arrive-t-il quand Start=1?
  - Qu'arrive-t-il à chaque cycle d'horloge subséquent?
  - Faites un exemple numérique avec dividende=27 et diviseur=6

Exemple (seul)

- Quand start=1:
  - A=dividende B=diviseur, Quotient=0
- À chaque cycle:
  - A=A-B
  - Quotient=Quotient + 1
  - On compare si B > A
- On arrête quand B > A
  - Quotient est calculé à chaque cycle
  - Reste = A

Exemple (seul)

- Quand start=1:
    - A=27, B=6, Quotient=0
- À chaque cycle:
    - A=27-6=21, Quotient = 1
    - A=21-6=15, Quotient = 2
    - A=15-6=9, Quotient = 3
    - A=9-6=3, Quotient = 4
    - On arrête
- Quotient = 4, Reste = 3

Algorithme multi-cycle en VHDL

- Les algorithmes multi-cycle suivent un patron assez semblable:
  - Lorsque Start = 1, l'algorithme commence
  - Toutes les entrées sont enregistrées dans des flip flops
- C'est pratique d'avoir une variable qui indique quand l'algorithme roule
  - Quand start=1, cette variable est à 1
  - Quand cette variable est 1, on fait les calculs
  - Quand le critère de fin est arrivé, la variable tombe à 0

Algorithme multi-cycle en VHDL

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        IF start = '1'THEN
            s_terme1 <= entree1;
            s_terme2 <= entree2;
            s_produit <= (OTHERS => '0');
            fin <= '0';
        ELSIF s_terme2 > 0 THEN
            s_produit <= s_produit + s_terme1;
            s_terme2 <= s_terme2 - 1;
            fin <= '0';
        ELSIF s_terme2 = “0000” THEN
            produit <= s_produit;
            fin <= '1';
        END IF;
    END IF;
END PROCESS;
```

1. On enregistre les entrées
2. On initialise le produit

1. On fait l’addition
2. On soustrait de 1

Quand s_terme2 est 0,
la multiplication est terminée

Algorithmes itératifs

- Considérons maintenant une opération plus complexe
- L'approche proposée est l'algorithme itératif:
    - Il sera multi-cycle, mais aura des critères de convergence différents
- Considérons le cas particulier de la racine carrée

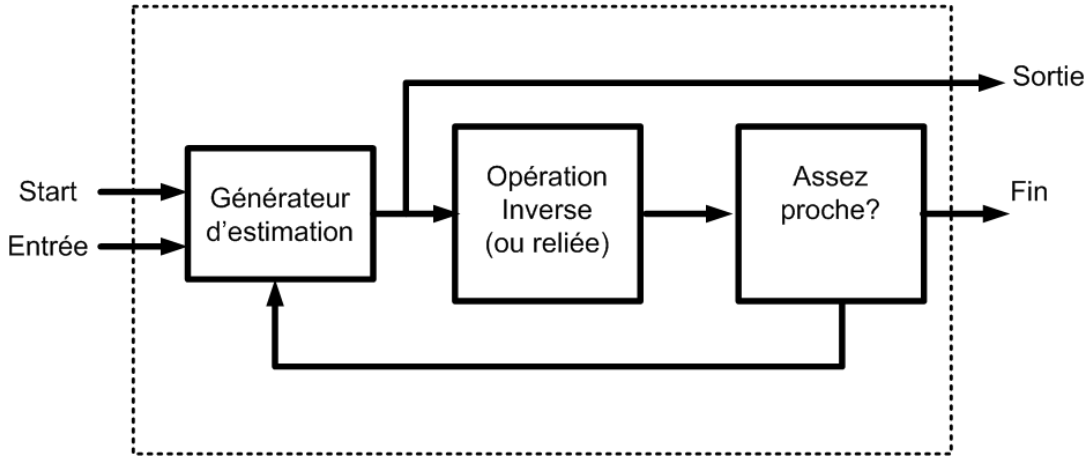
Racine carrée

- Dans l'approche la plus simple, on peut essayer toutes les possibilités:
  - On conserve la valeur qui génère le moins d'erreur
  - Avec la racine carrée, l'erreur se mesure en calculant x^2
- On veut la racine carré de 144
  - On commence avec 0: 0*0=0 Erreur de 144
  - 1*1=1 Erreur de 143
  - 2*2=4 Erreur de 140
  - …

12 donnerait l'erreur la plus faible (0)

Algorithmes itératifs

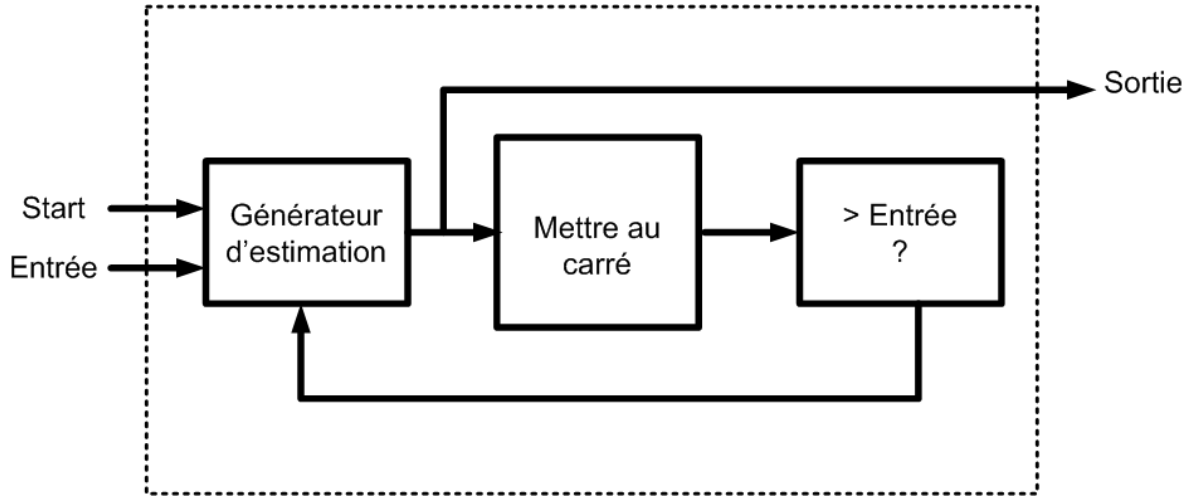
- Une autre façon de fonctionner sera celle-ci:



- On estime et on se rapproche tranquillement vers la bonne réponse…

Racine carrée

- L'ajustement de l'estimation se fait avec x^2
  - On veut savoir si on est trop haut ou trop bas
  - On ajuste notre estimation en conséquence

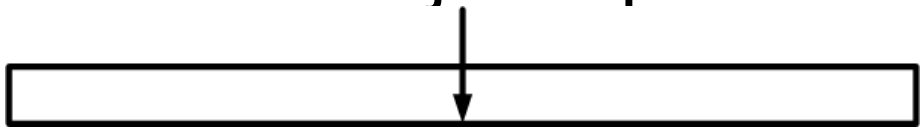


Racine carrée

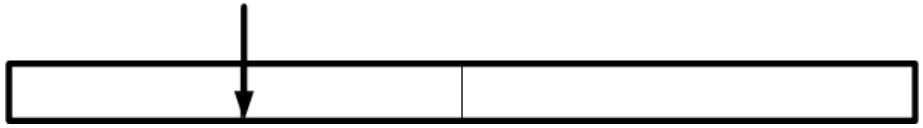
- Comment générer la première estimation?
  - On devine
- Ça nous donne un point de départ
  - Avec le x^2, on s'ajuste
  - Si on cherche 144 et qu'on devine 10: 10*10=100 (+)
  - Si on cherche 144 et qu'on devine 15: 15*15=225 (-)
- Dans notre cas, on fait une recherche binaire
  - L'approche est structurée

Racine carrée

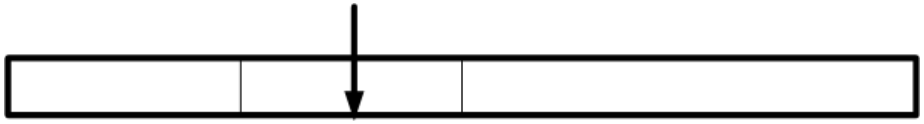
- On commence toujours par le milieu:



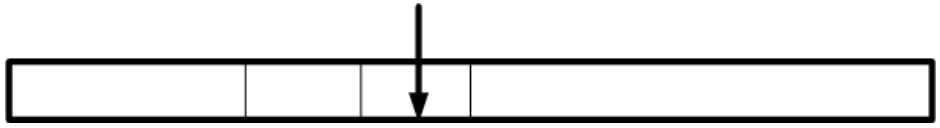
  - L'opération inverse révèle que c'est trop grand



  - L'opération inverse révèle que c'est trop petit



  - L'opération inverse révèle que c'est trop petit

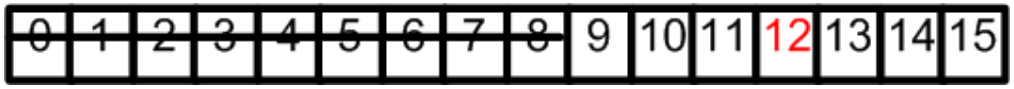


Et on a terminé

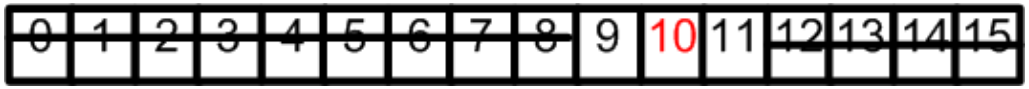
Racine carrée

- Trouvons la racine carrée de 121:
  - Les possibilités vont de 0 à 15 (4 bits)
- Start = 1:
  - Estimation = 8, Carré = 64: Trop petit

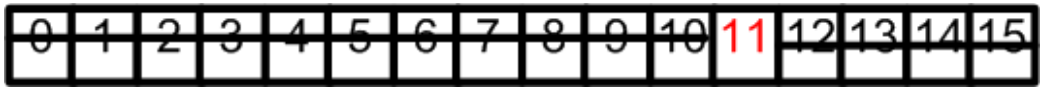
  - Estimation = 12(Milieu: 8-15), Carré = 144: Trop grand



  - Estimation = 10 (Milieu: 8-12), Carré=100: Trop petit



  - Estimation= 11(Milieu:10-12), Carré=121

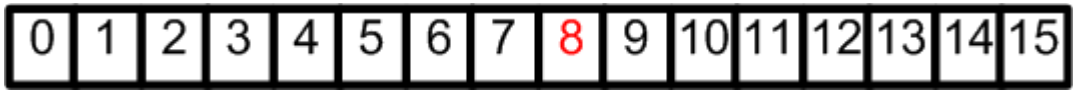


Exemple (seul)

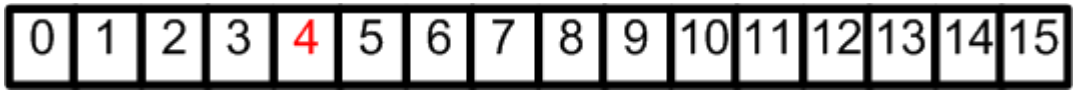
- On veut faire une division de deux nombres:
  - Le résultat donne juste un quotient de 4 bits
- Pensez à un algorithme itératif avec estimation
  - Quelle est l'opération inverse?
  - Qu'arrive-t-il quand Start=1?
  - Qu'arrive-t-il à chaque cycle d'horloge subséquent?
  - Faites un exemple numérique avec dividende=28 et diviseur=4

Exemple (seul)

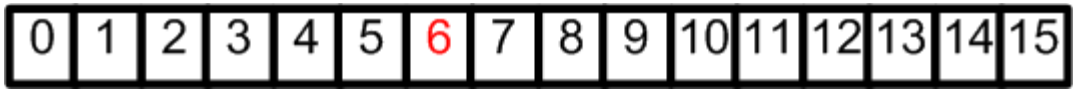
- L'opération inverse est la multiplication
- Quand Start=1
  - Estimation = 8, 8x4=32: Trop grand



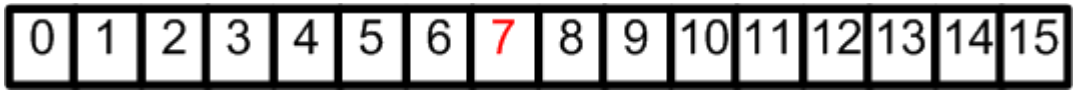
  - Estimation = 4, 4x4=16: Trop petit



  - Estimation = 6, 6x4=24: Trop petit



  - Estimation = 7, 7x4=28



Exemple (seul)

- Dans la recherche binaire, il y a un maximum de N essais pour un nombre de N bits
    - Si la réponse peut avoir 4 bits, on aura la bonne réponse après 4 essais (maximum)
- Une façon de le voir:
    - On commence par mettre estimation(3) = 1
    - Donc estimation="1000"
    - Si c'est trop gros, estimation(3)=0 (sinon, inchangé)
    - On passe au prochain: estimation(2)=1...

Exemple (seul)

- Donc, on recommence l'exemple:
  - Estimation="1000", 8x4=32: trop gros
  - Estimation="0100", 4x4=16: PAS trop gros
  - Estimation="0110", 6x4=24: PAS trop gros
  - Estimation="0111", 7x4=28: PAS trop gros
- Qu'arrive-t-il si on arrive à la bonne réponse avant la fin?
  - Essayons avec un diviseur de 32 et un dividende de 8

Exemple (seul)

- Avec ces nouveaux chiffres:
  - Estimation="1000", 8x8=64: trop gros
  - Estimation="0100", 4x8=32: PAS trop gros
  - Estimation="0110", 6x8=48: trop gros
  - Estimation="0101", 5x8=40: trop gros
  - Donc... Estimation="0100"
- Réponse: ce n'est pas grave si on arrive à la bonne réponse avant le temps...

Algorithme itératif en VHDL

- L'approche ici est semblable:
  - Lorsque Start = 1, l'algorithme commence
  - Toutes les entrées sont enregistrées dans des flip flops
  - Variable INDEX est égale au nombre de bits MOINS 1
- À chaque cycle,
  - On met le bit estimation(INDEX) = 1
  - Si l'estimation est trop grosse, estimation(INDEX)=0
  - On passe au prochain bit (INDEX=INDEX-1)
  - Ça continue jusqu'à ce que INDEX = 0

Dans l'exemple, on le fait un peu différemment...

Algorithme itératif en VHDL

```
PROCESS (clk)
BEGIN
   IF clk'EVENT AND clk = '1' THEN
      IF start = '1' THEN
         s_index <= "100";
         s_entree <= entree;
         s_estimation <= "1000";
         fin <= '0';
      ELSIF s_index > 0 THEN
         IF s_carre > s_entree THEN
            s_estimation(index-1) = '0';
         END IF;
         s_index <= s_index - 1;
         s_estimation(s_index-2)='1';
         fin <= '0';
      ELSE
         estimation <= s_estimation;
         fin <= '1';
      END IF;
   END IF;
END PROCESS;
 s_carre <= s_estimation * s_estimation;
```

1. On enregistre les entrées
2. On initialise l'estimation

1. Si trop gros, on met le bit à 0
2. On passe à l'autre bit
3. On met l'autre bit à 1

On met au carré en tout temps

Note: On utilise s_index = 4 parce qu'on veut arrêter quand il est 0... Plutôt que commencer à 3 et terminer à -1