

© ALSE - Sept 2001

# *Réussir la Conception des FPGA Complexes.*

-do it right, the first time-



Bertrand CUZEAU  
Technical Manager - ALSE  
ASIC / FPGA Design Expert  
Doulos HDL Instructor (Verilog-VHDL)

info@ALSE-FR.COM

<http://www.alse-fr.com>

☎ : 33.(0)1 45 82 64 01

# *Qu'est-ce qu'un FPGA "Complexe" ?*



NB : Complexité  $\neq$  Densité :

- Un FPGA de 1 Million de portes peut être simple à concevoir...

Quelques Facteurs aggravants :

- Asynchronisme,
- Définition insuffisante,
- Environnement externe complexe / mal modélisé,
- Proximité des Limites technologiques,
- Algorithmes complexes en peu de cycles, ...

Indicateur : Plan de Test ou Phase de Recette.

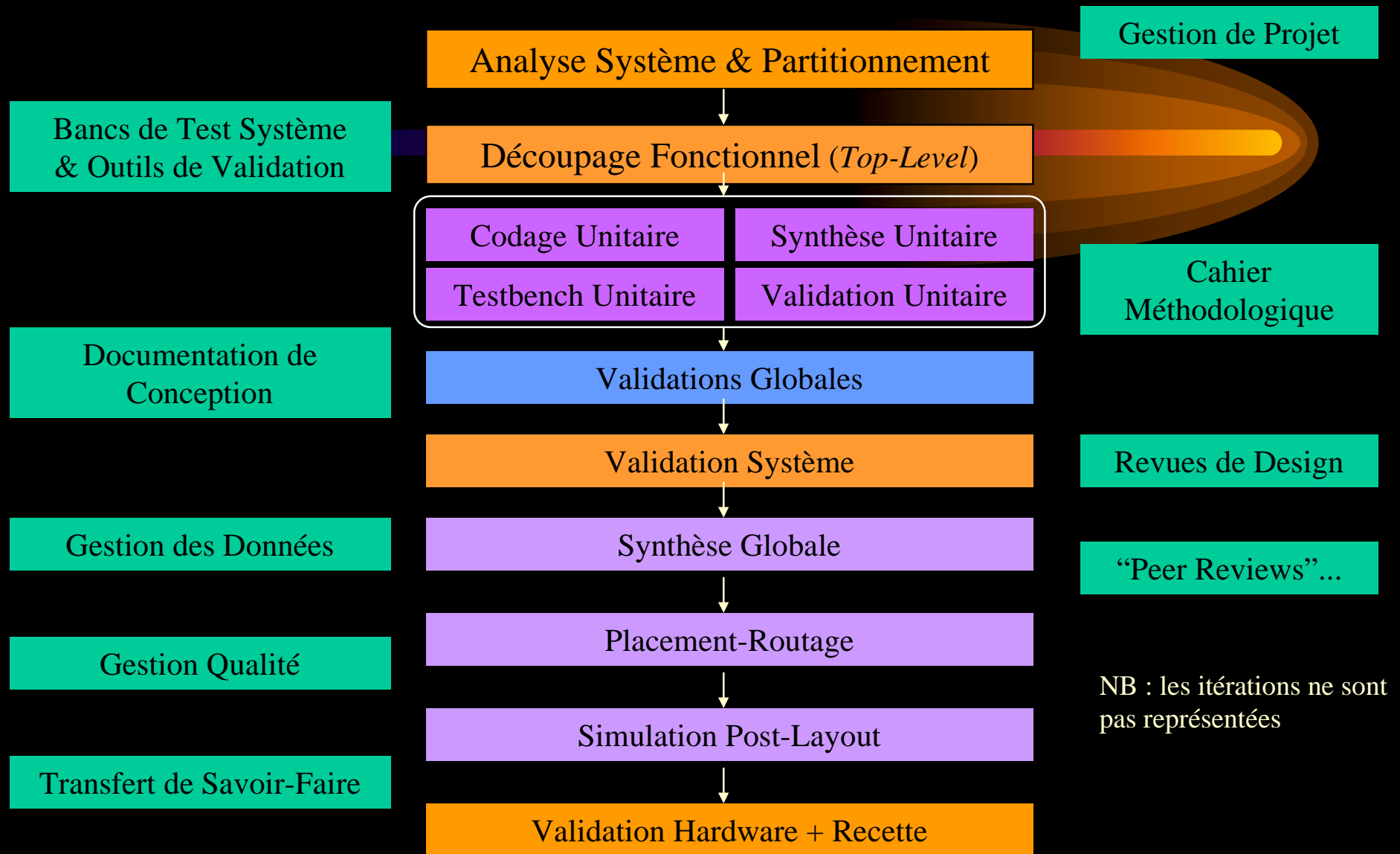
# *Les Ingrédients du Succès*



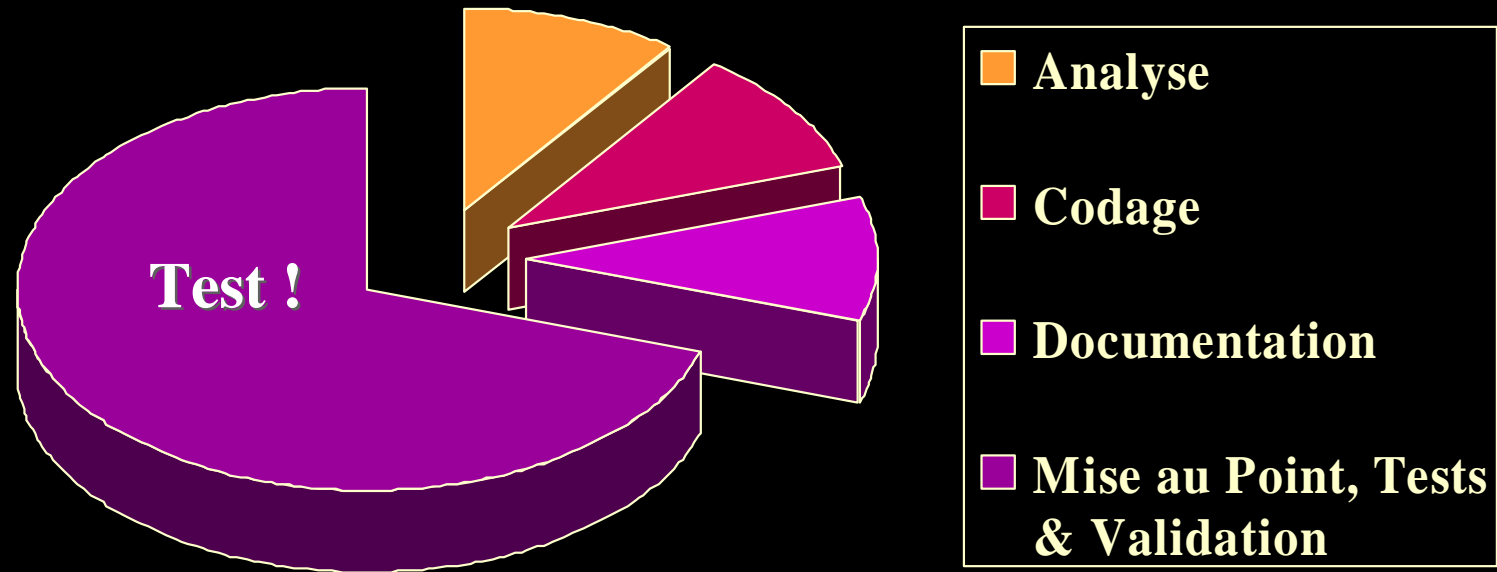
## **Savoir-Faire**



# Design Flow (Flot de Conception)



# *Le Développement*



Conséquence : Optimiser l'étape de codage est souvent inutile (parfois néfaste)

# *Le Rôle de la Simulation*

## **La Simulation Fonctionnelle évoluée est au Coeur de la Méthodologie gagnante :**

- Réduire, simplifier, et sécuriser la phase de Mise au Point.
- Faciliter les tests de non-régression
- Faciliter la recette

Un gain d'un facteur 3 est courant

=> délais & coûts de développement divisés par 2 !

# *Accélérer la Mise au Point*

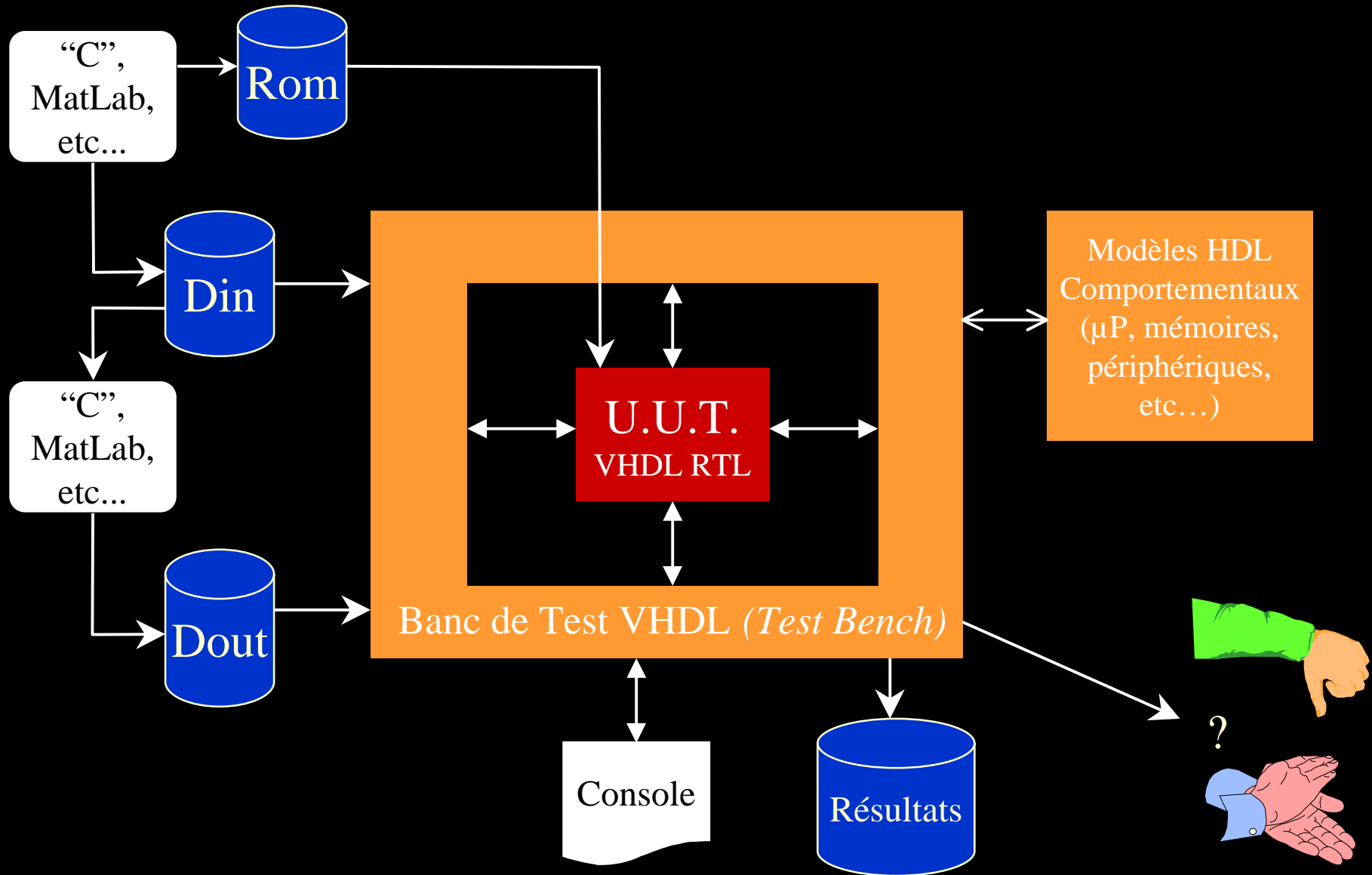


On peut accélérer la phase de Simulation.

Voici quelques astuces pratiques :

- Supprimer les tâches répétitives (GUI → Scripts),
- Automatiser les tests (push-button, self-testing),
- Eliminer l'analyse des Waveforms,
- Utiliser les E/S Fichiers,
- Comparaison au vol (“on-the-fly”),
- Tests interactifs, ...

# Banc de Test évolué



## Exemple de Générateur de Code VHDL en 'C'

```
fprintf(fdest, "  -- Synthesized Waveform Table\n");
fprintf(fdest, "  -- (c) Bertrand Cuzeau - alse@compuserve.com\n");
fprintf(fdest, "  subtype SLV8 is std_logic_vector (7 downto 0);\n");
fprintf(fdest, "  type Rom64x8 is array (0 to 63) of SLV8;\n");
fprintf(fdest, "  constant Sinus_Rom : Rom64x8 := (\n  ");

for (n=0; n<n_ech; n++)
  {
    slong = sin (deuxpi *n/(4 * n_ech));
    slong *= 127;  // for 7 bits
    sint = (int) slong;
    fprintf (fdest, "  x\"%02x\"", sint);
    if (n != n_ech-1) fprintf (fdest, ", ");
    if ((n % 8)==7) fprintf(fdest, "\n  ");
  }
fprintf(fdest, "  ); ");
fclose(fdest);
```

### Résultat obtenu :

```
subtype SLV8 is std_logic_vector (7 downto 0);
type Rom64x8 is array (0 to 63) of SLV8;
constant Sinus_Rom : Rom64x8 := (
  x"00", x"03", x"06", x"09", x"0c", x"0f", x"12", x"15",
  x"18", x"1b", x"1e", x"21", x"24", x"27", x"2a", x"2d",
  x"30", x"33", x"36", x"39", x"3b", x"3e", x"41", x"43",
  x"46", x"49", x"4b", x"4e", x"50", x"52", x"55", x"57",
  x"59", x"5b", x"5e", x"60", x"62", x"64", x"66", x"67",
  x"69", x"6b", x"6c", x"6e", x"70", x"71", x"72", x"74",
  x"75", x"76", x"77", x"78", x"79", x"7a", x"7b", x"7b",
  x"7c", x"7d", x"7d", x"7e", x"7e", x"7e", x"7e", x"7e" );
```

# *Méthode de Description*

**La méthode de description doit être adaptée :**

- **aux nouveaux outils (HDL)**
- **aux composants modernes**
- ~~FFs simples, logique discrète !~~
- ~~Equations logiques complexes !~~
- ~~Macros TTL !!!~~
- + Machines à Etats complexes
- + RTL comportemental

# *Les Atouts du Silicium*

Certains composants peuvent être mieux adaptés que d'autres à certaines applications. Citons en vrac quelques points qui peuvent faciliter le développement :

- Qualité de la Librairie des macro-fonctions difficilement inférables.  
ex : Multiplieurs pipelinés (suivant synthétiseur), Rams / Roms / Dual Ports, opérateurs spéciaux, etc...
- Prédicibilité et stabilité des performances,
- Outils de Synthèse et de Simulation VHDL,
- Phase de Placement-Routage simple et bien intégrée,
- etc...

# *Le Savoir-Faire*

- Une Formation excellente s'impose (cf Annexe B)  
+ remises à niveau régulières.
- Bonne maîtrise des règles de Conception Digitale !
- Rigueur et Méthode,
- Expérience et Pratique (du design et des outils)

N.B : Excellence du concepteur  $\Rightarrow$  Simplicité du code...

## Attention !

- La littérature et les formations sont rarement à la hauteur des exigences (et à jour avec l'état de l'art) ...
- Le code existant est souvent de mauvaise qualité ...

# Conclusion

Aujourd'hui plus que jamais, la réussite dépend du savoir-faire et de l'intelligence du concepteur, mais :  
une méthodologie rigoureuse est incontournable,  
(la "bidouille" n'est plus viable).

- Le(s) langage(s) HDL,

- Les outils et

- Le silicium

sont aujourd'hui au rendez-vous !

En y associant Méthodologie et Savoir-Faire,  
les applications les plus "complexes" vous tendent les bras...

# - Annexe A -

## UART comportementale Emis./Récep. simple

### Emission (N-8-1)

```
TX <= '0'; -- Start bit
wait for BITperiod;

for i in 0 to 7 loop
    TX <= RSDData(i);
    wait for BITperiod;
end loop;

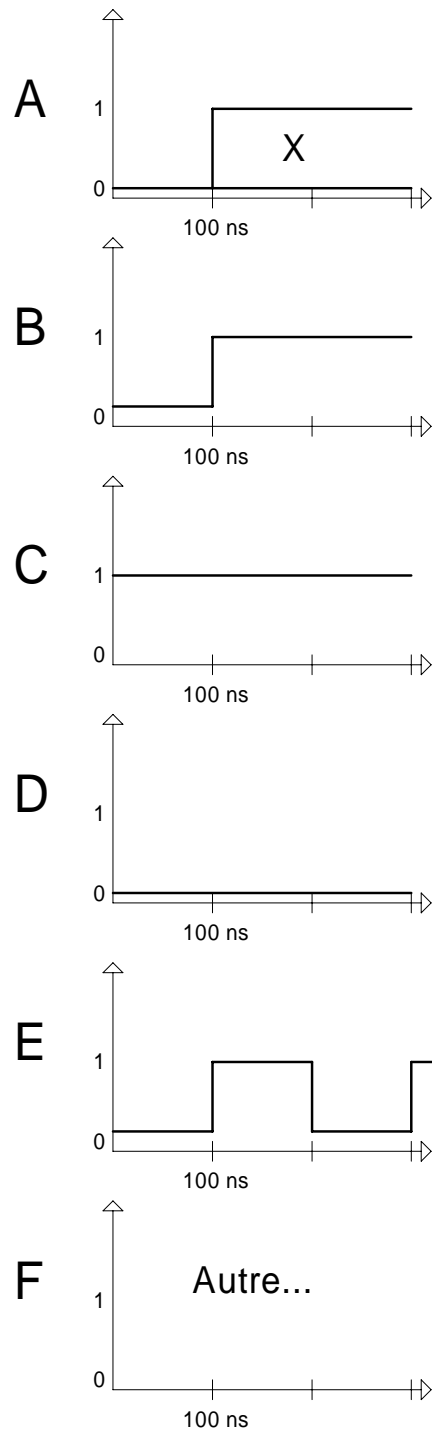
TX <= '1'; -- Stop bit
wait for BITperiod;
```

### Réception (N-8-1)

```
loop
    wait until RX='0'; -- get falling edge
    wait for (0.5 * BITperiod); -- Middle of Start bit
    assert RX='0'
        report "Start Bit Error ???" severity warning;
    wait for BITperiod; -- First Data Bit
    for i in 0 to 7 loop -- Get word
        MOT(i) := RX;
        wait for BITperiod;
    end loop;
    wait for BITperiod; -- Stop bit
    assert RX='1'
        report "Stop bit Error ???" severity warning;
    WRITE (L, ... (MOT)); -- send the received char to
    WRITELINE (F, L); -- file or transcript
end loop;
```

Conclusion : un langage HDL (Verilog/VHDL) permet de construire très simplement des modèles comportementaux que l'on pourra utiliser pour spécifier ou pour valider.

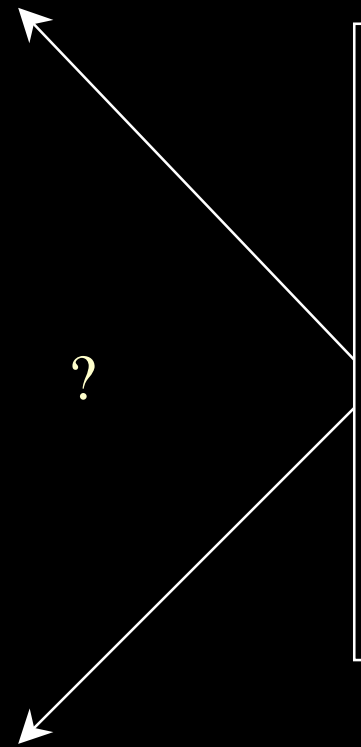
# - Annexe B - Quizz VHDL



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

entity quizz is
end quizz;

architecture test of quizz is
    signal nRST : std_logic;
begin
    process
    begin
        nRST <= '0';
        nRST <= '1' after 100 ns;
        wait;
    end process;
end test;
```



La réponse est "F". Etait-ce évident ?

Et si la dernière ligne était :  
nRST <= '0' after 100 ns; ?