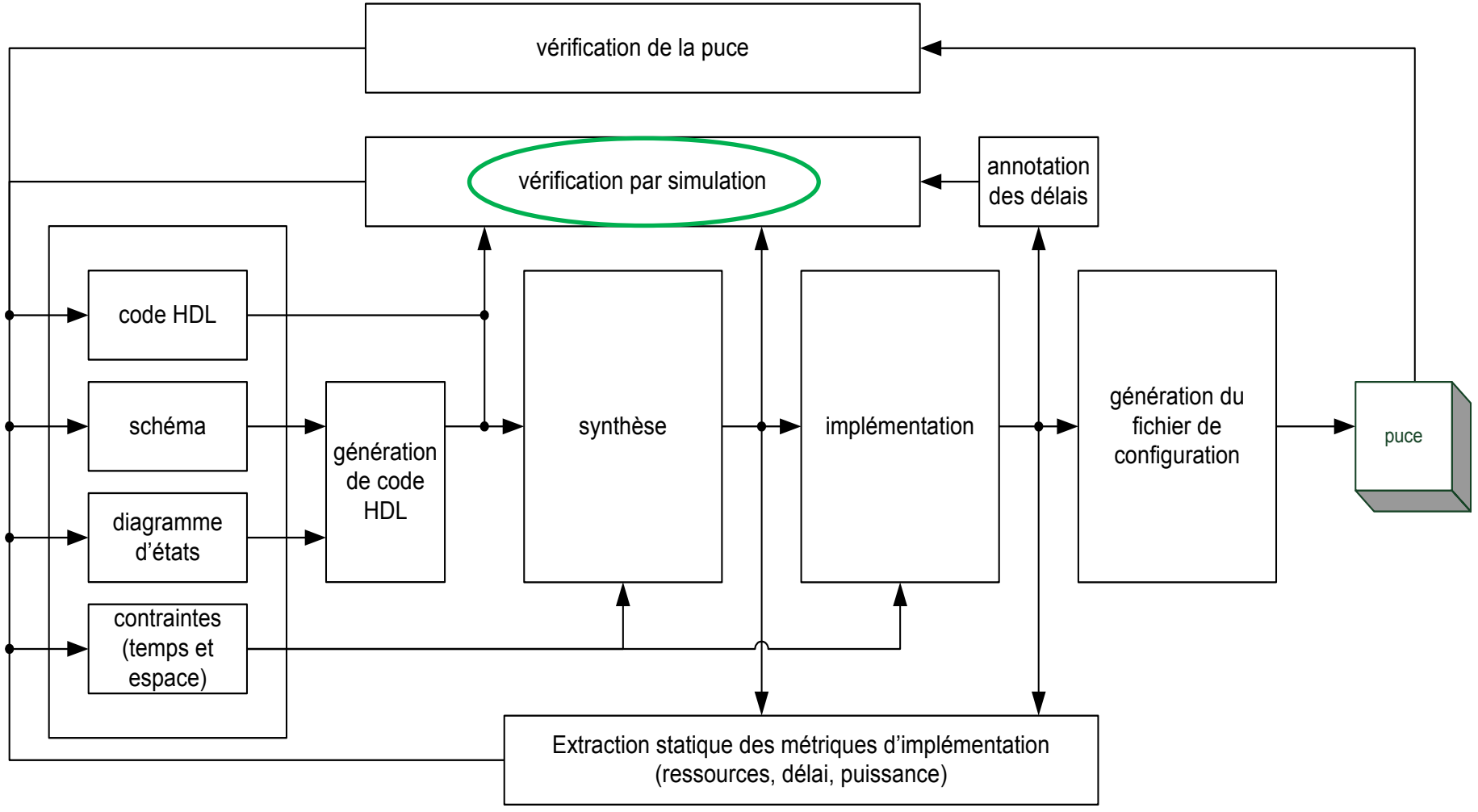
Simulation d’un modèle VHDL

Simulation d’un modèle VHDL
Sujets de ce thème

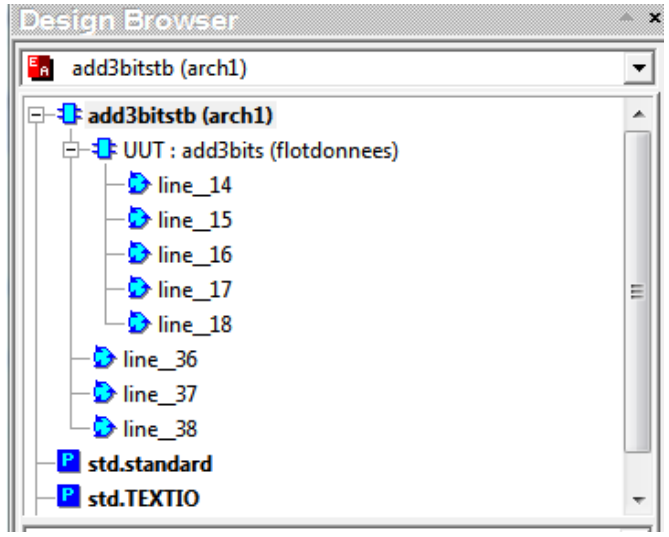
- Simulation: rappel et vue d’ensemble
- Le problème de la simulation séquentielle d’événements concurrents
- Fonctionnement du simulateur:
  - Liste d’événements
  - Liste des dépendances
  - Délais delta
- Exemples
- Valeurs par défauts d’objets en VHDL

Simulation: vue d'ensemble

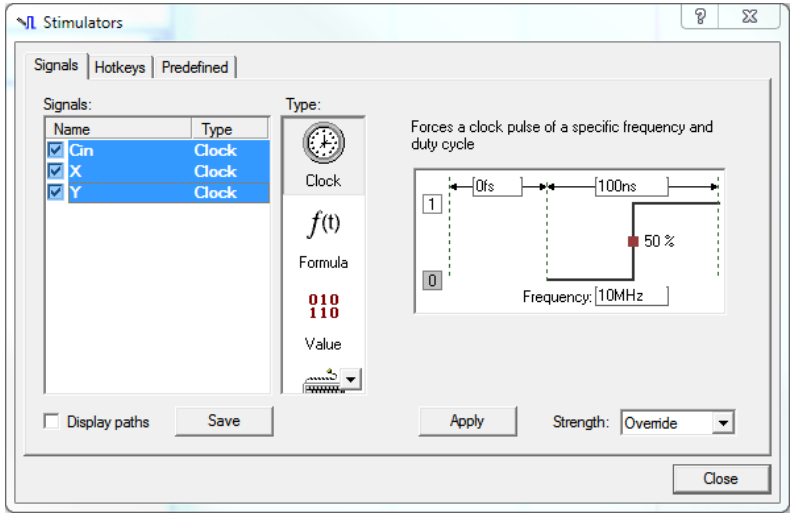


Simulation: vue d'ensemble

- Un simulateur travaille avec la description intermédiaire produite par un compilateur à partir de code HDL.
- Le simulateur nécessite qu'on force la valeur de certains signaux (habituellement les ports d'entrée du circuit à simuler), par le biais d'un banc d'essai ou d'outils de simulation.
- Du point de vue du simulateur, le banc d'essai et le modèle à simuler sont fusionnés.

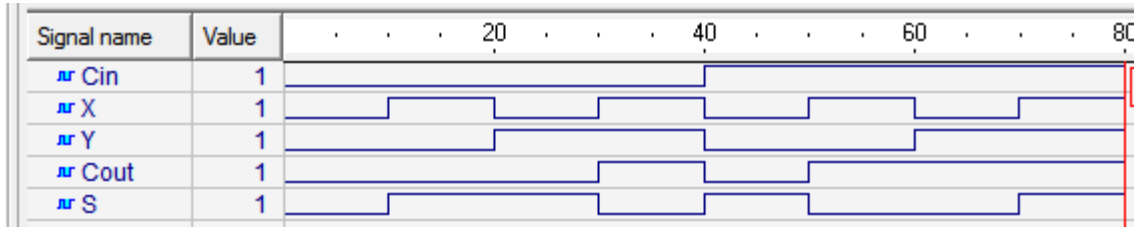



```
Cin <= '0' after 0 ns, '1' after 40 ns;
Y <= '0' after 0 ns, '1' after 20 ns;
X <= '0' after 0 ns, '1' after 10 ns;
```



Simulation: vue d'ensemble

- Le simulateur évalue comment les signaux forcés se propagent dans le circuit et calcule la valeur de tous les signaux qui en dépendent.
- Le simulateur présente les résultats de la simulation dans:
  - un chronogramme;
  - un tableau;
  - des messages à la console; et/ou,
  - un fichier.



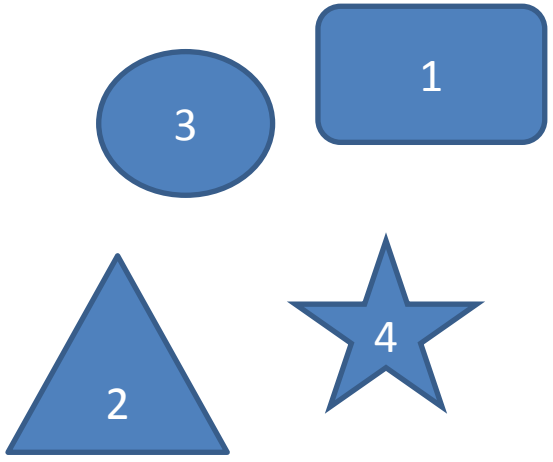
Time	Delta	Cin	X	Y	Cout	S
0 ps	3	0	0	0	0	0
10000 ps	3	0	1	0	0	1
20000 ps	1	0	0	1	0	1
30000 ps	3	0	1	1	1	0
40000 ps	3	1	0	0	0	1
50000 ps	4	1	1	0	1	0
60000 ps	1	1	0	1	1	0
70000 ps	3	1	1	1	1	1

```
#  Simulation has been initialized
#  Selected Top-Level: add3bitstb (arch2)
run
# EXECUTION:: NOTE    : simulation terminée
# EXECUTION:: Time: 80 ns,  Iteration: 0,  TOP instance,  Process: line__82.
# KERNEL: stopped at delta: 0 at time 80 ns.
>
```

Console

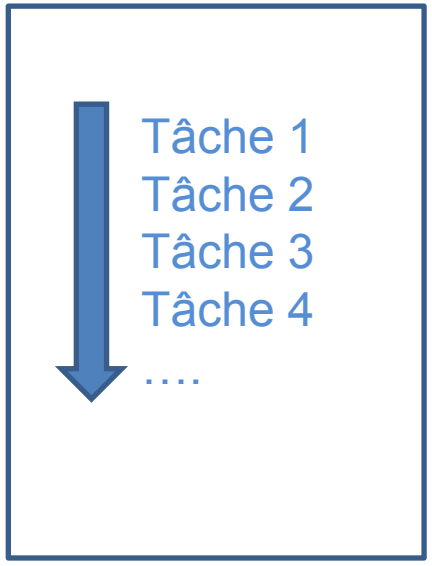
Problème fondamental de la simulation

- VHDL permet de modéliser des systèmes où plusieurs événements peuvent se produire concurremment.
- La simulation est faite sur une station de travail dotée d'un microprocesseur qui traite l'information de façon séquentielle.



- Le problème fondamental de la simulation de code VHDL sur une station de travail consiste à simuler la concurrence d'événements sur une machine séquentielle.

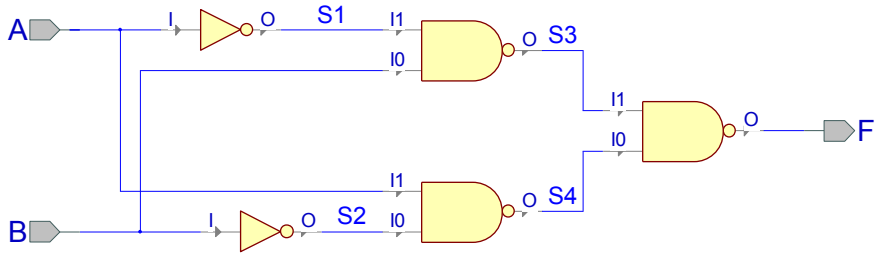
Comment simuler ceci avec cela?

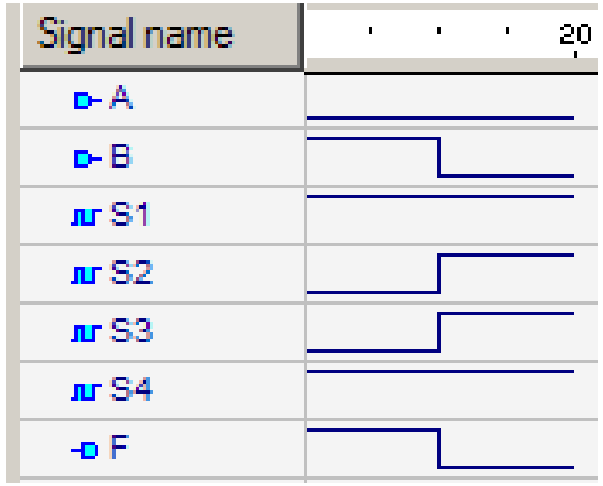


Simulation d'un modèle avec son banc d'essai

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

entity demodelaidelta is
  port(
       A : in STD_LOGIC;
       B : in STD_LOGIC;
       F : out STD_LOGIC
  );
end demodelaidelta;

architecture flot of demodelaidelta is
signal S1, S2, S3, S4 : STD_LOGIC;
begin
S3 <= not(B and S1);
S4 <= not(S2 and A);
F <= not(S4 and S3);
S1 <= not(A);
S2 <= not(B);
end flot;
```

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

entity demodelaidelta_tb is
end demodelaidelta_tb;

architecture TB_ARCHITECTURE of demodelaidelta_tb is
signal A, B, F : STD_LOGIC;
begin
UUT : entity demodelaidelta(flot) port map (A, B, F);
A <= '0' after 0 ns;
B <= '1' after 0 ns, '0' after 10 ns;
end TB_ARCHITECTURE;
```





À t = 10 ns, un changement sur l'entrée B entraîne des changements simultanés sur les signaux S2 et S3, puis sur la sortie F.

Comment simuler cette concurrence d'événements?

Fonctionnement du simulateur

- Le fonctionnement du simulateur repose sur trois concepts fondamentaux :
  1. une liste d'événements;
  2. une liste des dépendances des signaux; et,
  3. le concept des délais delta.

Fonctionnement du simulateur

- Le fonctionnement du simulateur repose sur trois concepts fondamentaux :
  1. une liste d'événements;
  2. une liste des dépendances des signaux; et,
  3. le concept des délais delta.

  – Le simulateur maintient en tout temps une liste d'événements à simuler.
  – Un événement est un changement de valeur à apporter à un signal, comme le faire passer de '1' à '0'.

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

entity demodelaidelta_tb is
end demodelaidelta_tb;

architecture TB_ARCHITECTURE of demodelaidelta_tb is
signal A, B, F : STD_LOGIC;
begin
UUT : entity demodelaidelta(flot) port map (A, B, F);
A <= '0' after 0 ns;
B <= '1' after 0 ns, '0' after 10 ns;
end TB_ARCHITECTURE;
```

Liste d'événements

À t = 0, initialisation de la simulation (tout à 'U')
…
À t = 0, assignation de 0 à A
À t = 0, assignation de 1 à B
…
À t = 10 ns, assignation de 0 à B

Fonctionnement du simulateur

- Le fonctionnement du simulateur repose sur trois concepts fondamentaux :
  1. une liste d'événements;
  2. une liste des dépendances des signaux; et,
  3. le concept des délais delta.

  – La liste des dépendances des signaux indique pour chaque signal, la liste des signaux dont il dépend.
  – Cette dépendance est indiquée explicitement ou implicitement dans le code par la liste de sensitivité des processus.
  – Lorsqu'un événement se produit sur un signal, le simulateur ajoute l'évaluation des signaux qui dépendent de ce signal à la liste des événements à simuler.

```
library ieee;
use ieee.std_logic_1164.all;
entity demodelaidelta is
  port(
       A, B : in STD_LOGIC;
       F : out STD_LOGIC
  );
end demodelaidelta;
architecture flot of demodelaidelta is
signal S1, S2, S3, S4 : STD_LOGIC;
begin
S3 <= not(B and S1);
S4 <= not(S2 and A);
F <= not(S4 and S3);
S1 <= not(A);
S2 <= not(B);
end flot;
```

Signal	Dépend de
S3	B, S1
S4	S2, A
F	S4, S3
S1	A
S2	B

```
...
process (A, T1)
begin
   T1 <= A and B;
   T2 <= not(T1);
end process;
```

Signal	Dépend de
T1	A, T1
T2	A, T1

Fonctionnement du simulateur

- Le fonctionnement du simulateur repose sur trois concepts fondamentaux :
  1. une liste d'événements;
  2. une liste des dépendances des signaux; et,
  3. le concept des délais delta.
  - Le délai delta (Δ) permet la simulation séquentielle d'événements concurrents
  - Un délai delta est un temps infinitésimalement court nécessaire à l'évaluation d'un événement.
  - Il est possible que plusieurs événements soient simulés tour à tour en plusieurs délais delta consécutifs avant que la valeur de tous les signaux ne se stabilise.
  - Du point de vue du temps de simulation, la somme de tous ces délais delta reste nulle.

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

entity demodelaidelta_tb is
end demodelaidelta_tb;

architecture TB_ARCHITECTURE of demodelaidelta_tb is
signal A, B, F : STD_LOGIC;
begin
UUT : entity demodelaidelta(flot) port map (A, B, F);
A <= '0' after 0 ns;
B <= '1' after 0 ns, '0' after 10 ns;
end TB_ARCHITECTURE;
```

Liste d'événements

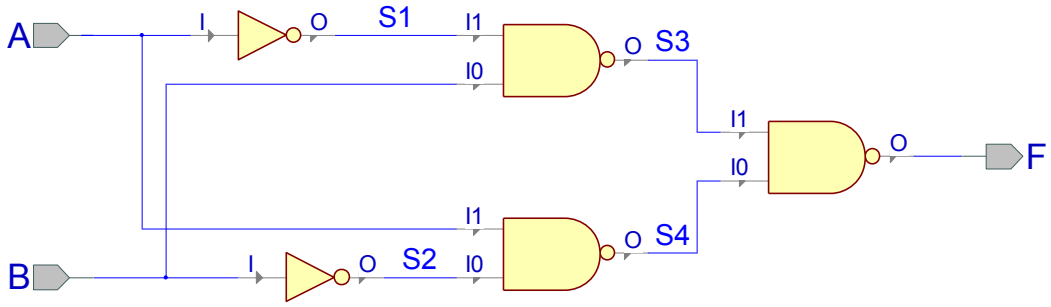
t = 0 + 0 Δ, initialisation de la simulation (tout à 'U')
…
À t = 0 + 1 Δ, assignation de 0 à A
À t = 0 + 2 Δ, assignation de 1 à B
…
À t = 10 ns + 0 Δ, assignation de 0 à B

Exemple du déroulement de la simulation

À t = 0 + 0 Δ, initialisation de la simulation (tout à 'U')
À t = 0 + 1 Δ, assignation de 01 à AB
À t = 0 + 2 Δ, S1, S2 et S4* prennent leur valeur
À t = 0 + 3 Δ, S3 prend sa valeur
À t = 0 + 4 Δ, F prend sa valeur

À t = 10 ns + 0 Δ, assignation de 0 à B
À t = 10 ns + 1 Δ, S2 et S3* prennent leurs nouvelles valeurs
À t = 10 ns + 2 Δ, F prend sa nouvelle valeur

```
...
A <= '0' after 0 ns;
B <= '1' after 0 ns, '0' after 10 ns;
...
```

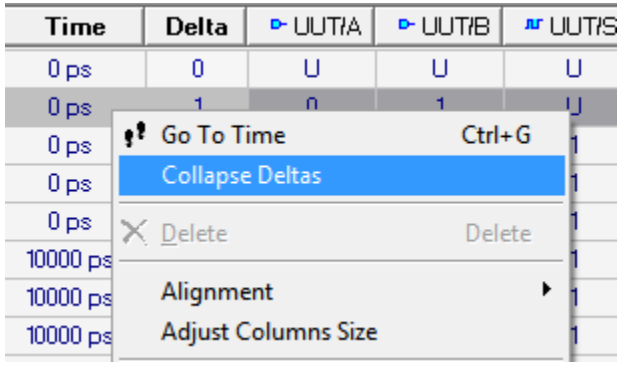


* À ces moments, le simulateur a assez d'information pour calculer la valeur du signal parce qu'un '0' à l'entrée d'une porte NON-ET implique automatiquement un '1' à sa sortie

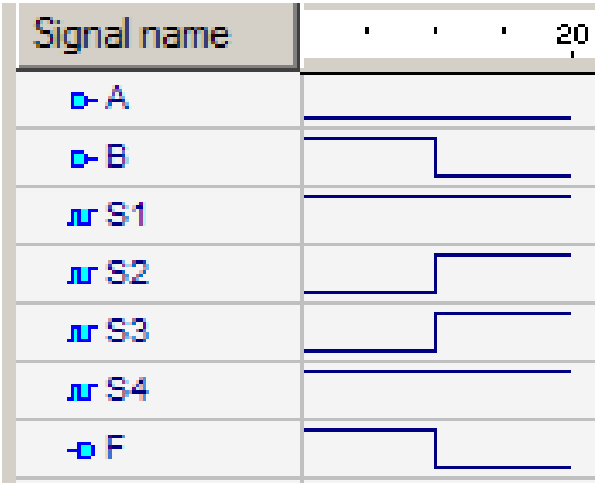
Time	Delta	UUT/A	UUT/B	UUT/S1	UUT/S2	UUT/S3	UUT/S4	UUT/F
0 ps	0	U	U	U	U	U	U	U
0 ps	1	0	1	U	U	U	U	U
0 ps	2	0	1	1	0	U	1	U
0 ps	3	0	1	1	0	0	1	U
0 ps	4	0	1	1	0	0	1	1
10000 ps	0	0	0	1	0	0	1	1
10000 ps	1	0	0	1	1	1	1	1
10000 ps	2	0	0	1	1	1	1	0

n × Δ = 0 unités de temps:
« collapse deltas »

Time	Delta	UUT/A	UUT/B	UUT/S1	UUT/S2	UUT/S3	UUT/S4	UUT/F
0 ps	0	U	U	U	U	U	U	U
0 ps	1	0	1	U	U	U	U	U
0 ps	2	0	1	1	0	U	1	U
0 ps	3	0	1	1	0	0	1	U
0 ps	4	0	1	1	0	0	1	1
10000 ps	0	0	0	1	0	0	1	1
10000 ps	1	0	0	1	1	1	1	1
10000 ps	2	0	0	1	1	1	1	0

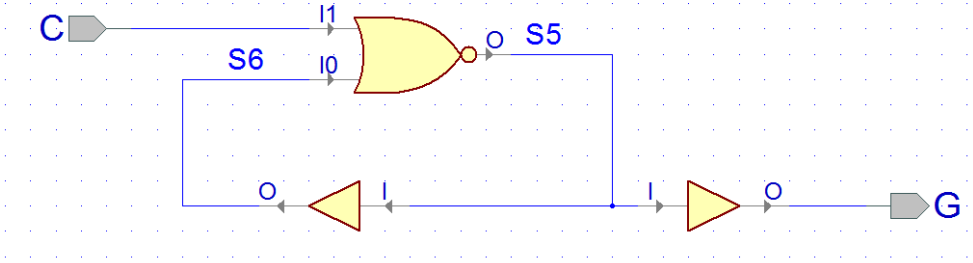


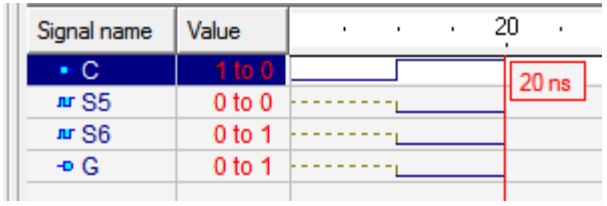
Time	Delta	UUT/A	UUT/B	UUT/S1	UUT/S2	UUT/S3	UUT/S4	UUT/F
0 ps	4	0	1	1	0	0	1	1
10000 ps	2	0	0	1	1	1	1	0

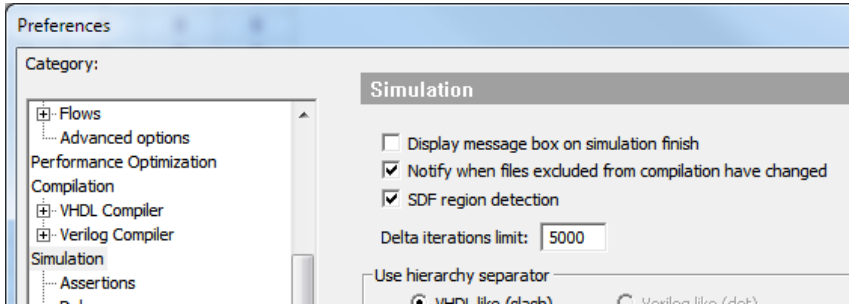


Et si la simulation n'avance que par des deltas?

- Il est possible, lors de la simulation d'un modèle, que la valeur de certains de ses signaux ne se stabilise pas.
- Par exemple, un circuit peut inclure un chemin de rétroaction qui le fait entrer en oscillation. Le temps de simulation n'avance alors qu'en deltas.
- Un simulateur doit pouvoir détecter cet état de chose, s'arrêter lui-même et afficher un message d'erreur.







Time	Delta	C	S5	S6	G
0 ps	0	0	U	U	U
10000 ps	0	1	U	U	U
10000 ps	1	1	0	U	U
10000 ps	2	1	0	0	0
20000 ps	0	0	0	0	0
20000 ps	1	0	1	0	0
20000 ps	2	0	1	1	1
20000 ps	3	0	0	1	1
20000 ps	4	0	0	0	0
20000 ps	5	0	1	0	0
20000 ps	6	0	1	1	1
20000 ps	7	0	0	1	1
20000 ps	8	0	0	0	0

```
#  Simulation has been initialized
#  Selected Top-Level: demodelaideltamax (arch)
run
# KERNEL: stopped at delta: 5000 at time 20 ns.
# KERNEL: Error: KERNEL_0160 Delta count overflow. Increase the iteration limit using
-i argument for asim or the matching entry in simulation preferences.
# Error: Fatal error occurred during simulation.
endsim
```

Déroulement global de la simulation

- Au temps 0, le simulateur place, sur la liste des événements:
  - les processus;
  - les assignations concurrentes; et,
  - les instanciations de composantes.
- Un des événements est choisi et est exécuté. L'exécution de cet événement peut engendrer de nouveaux événements qui sont placés sur la liste des événements, au temps 0 + Δ ou plus tard, tel que spécifié par une clause `after`.
- Les autres événements du temps 0 sont exécutés à leur tour, et d'autres événements sont ajoutés à la liste des événements au temps 0 + Δ ou plus tard.
- Quand tous les événements du temps 0 ont été exécutés, le temps de simulation est avancé au temps 0 + Δ, et le processus recommence.
- Le cycle se répète tant que la liste des événements contient encore des événements à traiter à un temps 0 + $n\Delta$.
- Ensuite, le temps de simulation est avancé selon le moment du prochain événement dans la liste des événements.
- Le cycle se répète tant que la liste des événements n'est pas vide.

Un deuxième exemple

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add3bits is
   port (
      Cin : in std_logic;
      X : in std_logic;
      Y : in std_logic;
      Cout : out std_logic;
      S : out std_logic
   );
end add3bits;

architecture flotdonnees2 of add3bits is
signal T1 : std_logic;
signal T2 : std_logic;
signal T3 : std_logic;
begin
   S <= T1 xor Cin;
   Cout <= T3 or T2;
   T1 <= X xor Y;
   T2 <= X and Y;
   T3 <= Cin and T1;
end flotdonnees2;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity add3bitsTBsimple is
end add3bitsTBsimple;

architecture arch1 of add3bitsTBsimple is
signal Cin, X, Y, Cout, S : std_logic;
begin
UUT : entity add3bits(flotdonnees2) port map (Cin, X, Y, Cout, S);
Cin <= '0' after 0 ns;
X <= '0' after 0 ns, '1' after 10 ns;
Y <= '0' after 0 ns;
end arch1;
```

Un deuxième exemple

Liste des événements

À t = 0 + 0 Δ, initialisation de la simulation (tout à 'U')
À t = 0 + 1 Δ, assignation de 000 à X, Y, Cin
À t = 0 + 2 Δ, évaluation des signaux T1, T2, T3 et S (provoquées par les changements sur X, Y, Cin)
À t = 0 + 3 Δ, évaluation de T3, S et Cout (provoquées par les changements sur T1, T2, T3)
Aucun nouvel événement n'est ajouté à la liste d'événements.

À t = 10 ns + 0 Δ, assignation de 1 à X
À t = 10 ns + 1 Δ, évaluation de T1 et T2 (provoquées par le changement sur X)
À t = 10 ns + 2 Δ, évaluation de T3 et de S (provoquées par le changement sur T1)
Aucun nouvel événement n'est ajouté à la liste d'événements.

```
-- module add3bits:
S <= T1 xor Cin;
Cout <= T3 or T2;
T1 <= X xor Y;
T2 <= X and Y;
T3 <= Cin and T1;
...
```

Signal	Dépend de
S	T1 , Cin
Cout	T3, T2
T1	X, Y
T2	X, Y
T3	Cin, T1

```
-- dans le banc d'essai:
Cin <= '0' after 0 ns;
X <= '0' after 0 ns, '1' after 10 ns;
Y <= '0' after 0 ns;
```

Time	Delta	UUT/T1	UUT/T2	UUT/T3	UUT/X	UUT/Y	UUT/Cout	UUT/S	UUT/Cin
0 ps	0	U	U	U	U	U	U	U	U
0 ps	1	U	U	U	0	0	U	U	0
0 ps	2	0	0	0	0	0	U	U	0
0 ps	3	0	0	0	0	0	0	0	0
10000 ps	0	0	0	0	1	0	0	0	0
10000 ps	1	1	0	0	1	0	0	0	0
10000 ps	2	1	0	0	1	0	0	1	0

Valeurs par défaut d'objets en VHDL

- Lors de la simulation, un objet déclaré sans initialisation de valeur se voit assigner une valeur par défaut qui dépend de son type.
- La valeur par défaut correspond à la borne gauche du type (type T, valeur T'left).
- La borne gauche dépend de la déclaration du type.

```
type STD_ULOGIC is (
'U',              -- Uninitialized
'X',              -- Forcing  Unknown
'0',              -- Forcing  0
'1',              -- Forcing  1
'Z',              -- High Impedance
'W',              -- Weak     Unknown
'L',              -- Weak     0
'H',              -- Weak     1
'-'               -- Don't care
);
subtype STD_LOGIC is resolved STD_ULOGIC;
type STD_ULOGIC_VECTOR is array (NATURAL range <>) of STD_ULOGIC;
subtype STD_LOGIC_VECTOR is (resolved) STD_ULOGIC_VECTOR;
```

Type	Valeur par défaut
`boolean`	FALSE
`natural`	0
`positive`	1
`integer`	typiquement -2^{31}
`std_logic`	'U'
`std_logic_vector`	« UUU….U »

Vous devriez maintenant être capable de …

- Expliquer le fonctionnement de la simulation d'un modèle VHDL. Expliquer les principes de la liste d'événements, de la liste de dépendances et des délais delta. (B2)
- Étant donné un modèle VHDL et son banc d'essai, montrer le déroulement de la simulation. (B3)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance – mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.