Une école de l'IMT

SystemVerilog pour la verification

SVA : langage pour assertions

Tarik Graba

Année 2017/2018

Plan

Les assertions en SystemVerilog

Autres langages pour les assertions

Les assertions

Les assertions sont des constructions qui permettent :

- de vérifier (ou de prouver) des propriétés (au sens logique)

En SystemVerilog, ces propriétés peuvent être

- statiques ou
- des séquences temporelles.

Elles permettent de définir un objectif de couverture des simulations effectuées.

Les assertions

Les assertions procédurales

Dans un processus, elles sont évaluées au moment de son exécution.

```
initial/always
begin
...
  assert(xx == aa) else $info("au fait");
...
  assert(xx == yy) else $error("pas bien!");
...
  assert(xx == zz) else $fatal(1,"vraiment pas bien!");
...
```

Sans **`else`** génère une erreur avec un message générique.

Normalement, ignorées par les outils de synthèse.

Les assertions

Les assertions concurrentes

Permettent de vérifier en permanence des règles dans un **`module`**, un **`program`** ou une **`interface`**.

- On définit des propriétés (**`property`**)
- Les propriétés sont forcément liées à un évènement déclencheur.
  - Les simulateur imposent qu'il soit lié à une horloge
- On définit ensuite une assertion (**`assert`**) sur cette propriété.

Les assertions

Les assertions concurrentes

Exemple

```
// au front d'horloge
// a ou b doit être vrai
property P0;
   @(posedge clk)
   a || b;
endproperty

assrt_p0: assert property(P0) else $info("Est-ce normal?");
```

Les assertions

Les assertions concurrentes

Implication (|->)

```
// au front d'horloge si a est vrai
// alor b doit être faux
property P1;
  @(posedge clk)
      a |-> !b;
endproperty

assrt_p1: assert property(P1) else $error("pas bien");
```

Dans la norme, «**Overlapped implication**»

L'évaluation de **b** se fait sur l'évènement pour lequel **a** est vrai.

Les assertions

Les assertions concurrentes

Implication (|=>)

```
// au front d'horloge si a
// alors au prochain front
// c doit être faux
property P2;
   @(posedge clk)
   a |=> !c;
endproperty

assrt_p2: assert property(P2) else $error("vraiment pas bien");
```

Dans la norme, «**Nonverlapped implication**»

L'évaluation de **c** se fait l'évènement suivant l'évènement pour lequel **a** est vrai.

Les assertions

Les assertions concurrentes

Capture

```
// au front d'horloge si req et !ack
// on capture la donnée
// on vérifie quelle reste stable au coup suivant
property P3;
 bit [7:0] s;
   @(posedge clk)
   (req && !ack , s = bus) |=> s == bus;
endproperty

assrt_p3: assert property(P3) else $error("Ça a changé!");
```

Dans l'exemple, la variable locale **s** permet de capturer la valeur de **bus** si la condition **(req && !ack)** est vérifiée. Au cycle suivant on vérifie que **bus** n'a pas changé de valeur.

À chaque fois que la propriété est déclenché, une nouvelle capture est faite.

Les assertions

Les assertions concurrentes

Les séquences

```
// au front d'horloge si stb
// alors ack doit être vrai au cycle suivant
// ou dans les 5 cycles
property P4;
  @(posedge clk)
      stb |-> ##[1:5] ack;
endproperty

assrt_p4: assert property(P4) else $error("Trop tard");
```

Si **stb** alors **ack** doit arriver au cycle suivant ou dans les 5 cycles.

Attention ce code n'est pas efficace.

Les assertions

Les assertions concurrentes

Les séquences

```
property P5;
  @(posedge clk)
     (a ##1 b) |-> (c ##2 d);
endproperty

assrt_p5: assert property(P5) else $error("Trop tard");
```

a suivi de **b** au cycle suivant, implique, **c** suivi de **d** deux cycles plus tard.
Détecter le changement d'état d'un signal peut alors être écrit (**`!stb ##1 stb`**)

On peut avoir des séquences infinies. Par exemple, **`##[1:$]x`** veut dire **x** vrai à partir du cycle suivant.

Des raccourcis

`$rose`	le signal est passé de `0` à `1`
`$fell`	le signal est passé de `1` à `0`
`$stable`	la valeur du signal n'a pas changée
`$changed`	la valeur du signal a changée
…	

Les assertions

Comment les faire intervenir dans un TB

```
module testbench();
  bit   clk;

  foo I(.*);

  tester tester_i(.*);
  slave  DUT(.*);
  monitor monitor_i  (.*);

  always #10ns clk = !clk;

endmodule
```

Les assertions

Comment les faire intervenir dans un TB

```
module monitor( foo.MONITOR I );

   property slave_data_notunknown_when_ready;
      @(posedge I.clk)
      I.ready |-> $isunknown(I.s) == 0;
   endproperty

   assert_slave_data_notunknown_when_ready: assert property (slave_data_notunknown_when_ready)
   else $error("%m: ready is asserted but data from slave is non valid");

   property slave_ready_until_valid;
      @(posedge I.clk)
      $rose(I.ready) |-> I.ready throughout I.valid [->1]; //ou I.ready [*0:$] ##1 I.valid;
   endproperty

   assert_slave_ready_until_valid: assert property(slave_ready_until_valid)
   else $error("%m:slave's ready must be held until valid is set");

   property slave_data_held_when_ready;
      bit [7:0] s;
      @(posedge I.clk) disable iff (I.nrst == 0)
      (I.ready && !I.valid , s = I.s) |=> s == I.s; //ou $stable(I.s);
   endproperty

   assert_slave_data_held_when_ready: assert property(slave_data_held_when_ready)
   else $error("%m: data must be held stable when slave is ready");

endmodule
```

Plan

Les assertions en SystemVerilog

Autres langages pour les assertions

PSL

Property Specification Language

PSL : existe et est standard depuis plus longtemps.
Compatible avec d'autres langages RTL (VHDL, Verilog)
En fonction des outils :

- fichiers indépendants
- commentaires magiques