SystemVerilog pour la vérification

Assertions

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Septembre 2024

1. Assertions
2. Combinaisons
3. Fonctions utiles
4. Exemples choisis

Assertions: Concept

- Les assertions permettent de spécifier le comportement du système
  - De manière interne
  - Au niveau des entrées/sorties
- Servent à
  - La vérification formelle
  - Vérifier le bon fonctionnement du DUV pendant la simulation
- *Assertion Based Verification (ABV)*
  - Complémentaire au banc de test classique
  - Peut être fait en partie par l'équipe de développement et en partie par l'équipe de vérification
    - Dépend du type d'assertion

Types d'assertions

- Il existe deux types d'assertions:
  - Immédiates
    - Placées dans du code procédural
  - Concurrentes
    - Placées dans le domaine concurrent

Assertions immédiates

- Permettent de vérifier une expression booléenne à un instant précis de la simulation

Exemple

```
assert1 : assert (a==b);
assert2 : assert (a==b)
            else $display("Zut");
assert3 : assert (a==b)
            $display("Cool");
            else $display("Zut");
```

Affiche un message si erreur

Affiche un message si ok

Affiche un message si erreur

- En cas de validation ou non il est possible d'exécuter des instructions

Sévérité

- Il est possible d'indiquer le degré de sévérité d'une erreur
  - `$fatal`
  - `$error`
  - `$warning`
  - `$info`
- Ces fonctions peuvent être appelées avec un message en argument
- `$fatal` prend un argument supplémentaire (numéro entre 0 et 2)
- S'il n'y a pas de `else`, le mode par défaut est `$error`

Exemple

```
assert1 : assert (a==b) else $fatal;
assert2 : assert (a==b) else $fatal(1,"Ach, c'est triste");
assert3 : assert (a==b) else $warning;
assert4 : assert (a==b) else $warning("Pas top, mais pas grave");
```

Assertions immédiates: usage intéressant

- Possibilité de compter des occurences

Exemple

```
assert1 : assert (a==b) egalite = egalite+1; else nbErreur++;
```

- Possibilité de signaler des erreurs

Exemple

```
assert2 : assert (a==b) egalite = egalite+1; else signifierErreur();
```

Assertions concurrentes

- Assertions combinatoires
  - Semblables aux assertions VHDL
  - Avec en plus le concept d'implication ($A \Rightarrow B$)
- Assertions temporelles
  - Permettent de vérifier le comportement au cours du temps
  - Assertions de type: Si séquence A, alors on doit observer séquence B

Assertions concurrentes

- Les propriétés peuvent être exploitées par 3 instructions:
  - `assert` permet de vérifier qu'une propriété est vérifiée
  - `assume` permet de spécifier des hypothèses sur l'environnement de simulation. Est notamment utile pour la vérification formelle (permet de limiter le nombre de cas).
  - `cover` permet de monitorer l'observation de propriétés

Exemples

```
a0: assert property (b == c);

a1: assume property never (a == b);

a2: cover property (a ##1 b ##1 c) counter = counter + 1;
```

Assertions concurrentes

- Une liste de sensibilité est obligatoire

Exemple

```
assert1 : assert property
            (@(negedge clk) (a==b));
assert2 : assert property
            (@(a,b) (a==b));
```

Evalué sur flanc descendant de clk

Evalué au changement de a ou b

- Il est conseillé de travailler avec une horloge (assert1)
- Si un clocking bloc est déclaré, il n'est pas nécessaire de spécifier l'horloge `@(posedge clk)` dans chaque propriété

Propriété d'implication

- Une implication peut s'écrire sous la forme d'une propriété:

Exemple: $a \Rightarrow b$

```
property prop_ab;
    a |-> b;
endproperty
```

Exemple: $(mode = 1) \Rightarrow (c = a + b)$

```
property prop_mode1;
    (mode==1) |-> (c==a+b);
endproperty
```

Séquences

- Il est possible de définir des séquences

Exemple

```
// a suivi de b
sequence seq_a;
    a ##1 b;
endsequence

// c suivi de d à 0
sequence seq_b;
    c ##1 !d;
endsequence
```

Propriétés

- Des propriétés peuvent être définies à partir des séquences

Exemple

```
property prop;
    seq_a |=> seq_b;
endproperty
```

- Des assertions peuvent être définies à partir des propriétés

Exemple

```
assert property (
    @(posedge clk)        // agit sur le flanc montant de l'horloge
    disable iff (rst==1) // annule l'assertion lors d'un reset
    prop
);
```

Assertion

- L'assertion précédente peut être écrite sous forme plus compacte

Exemple

```
assert property (
    @(posedge clk)        // agit sur le flanc montant de l'horloge
    disable iff (rst==1) // annule l'assertion lors d'un reset
    a ##1 b |=> c ##1 !d
);
```

- L'utilisation de séquences est pertinente s'il y a réutilisation possible

Annulation d'assertion

- `disable iff expr` permet d'annuler l'assertion sur une condition
- Très utile en cas de Reset
- Devrait même toujours être annulée en cas de Reset

Exemple

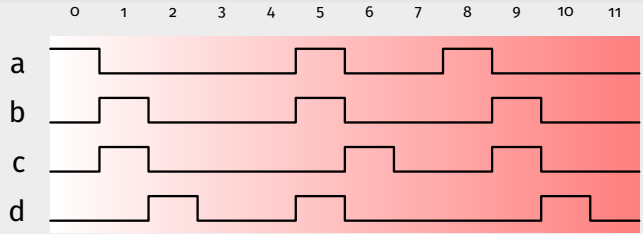
```
assert property (        // une horloge a déjà été définie
    disable iff (rst==1) // annule l'assertion lors d'un reset
    a ##1 b |=> c ##1 !d
);
```

Types d'implication

- Implication différée
  - |=>
  - L'évaluation de la séquence de droite commence après la fin de celle de gauche
- Implication directe
  - |->
  - L'évaluation de la séquence de droite commence au moment de la dernière phase de celle de gauche

Opérateurs d'implication

Exemple

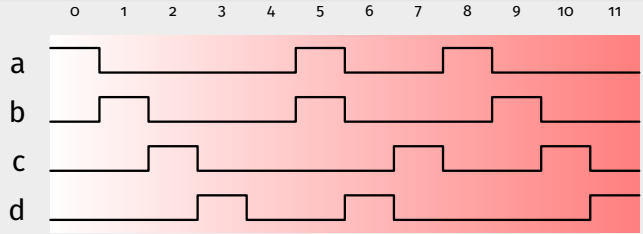


Assertions

```
assert property (a ##1 b |-> c ##1 d);

assert property (a ##1 b |=> c ##1 d);

assert property (a & b |-> d ##1 c);
```

Opérateurs d'implication

Exemple



Assertions

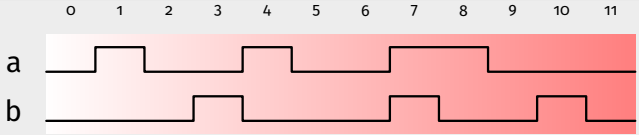
```
assert property (a ##1 b |-> c ##1 d);

assert property (a ##1 b |=> c ##1 d);
```

Délais

- Il est possible de spécifier un délai correspondant à un intervalle de temps
  - `a |-> ##[n:m] b`
    - *b* est vrai après un temps compris entre *n* et *m*
  - `a ##[n:$] b |=> c`
    - Lorsque *a* est observé et que *b* l'est après un temps d'au moins *n*, alors *c* doit l'être un temps après *b*

Délais

Exemple



Assertions

```
assert property (a |-> ##2 b);

assert property (a |-> ##[2:3] b);
```

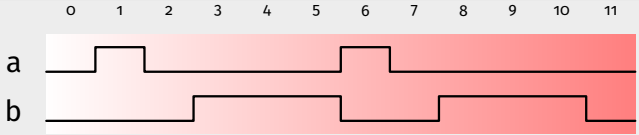
Répétitions

- Il est possible de spécifier la répétition d'un événement

Opérateur	Description
`[*n]`	Répétition un nombre *n* de fois
`[*n:m]`	Répétition un nombre de fois compris entre *n* et *m*
`[*n:$]`	Répétition un nombre de fois compris entre *n* et ∞
`[=n]`	Répétitions non consécutives
`[->n]`	Répétitions non consécutives, mais terminant sur la séquence

Répétitions

Exemple

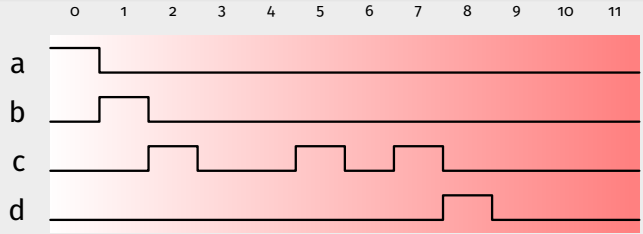


Assertions

```
assert property (a |-> ##2 b);

assert property (a |-> ##2 b[*3]);
```

Répétitions non consécutives

Exemple

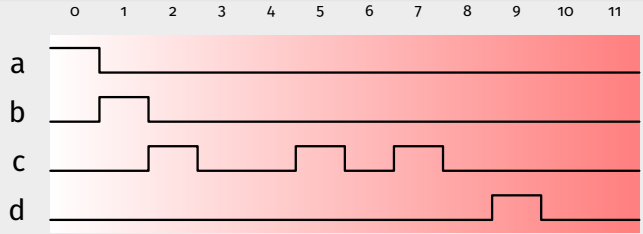


Assertions

```
assert property ((a)|=>(b ##1 c[=3] ##1 d));

assert property ((a)|=>(b ##1 c[->3] ##1 d));
```

Répétitions non consécutives

Exemple



Assertions

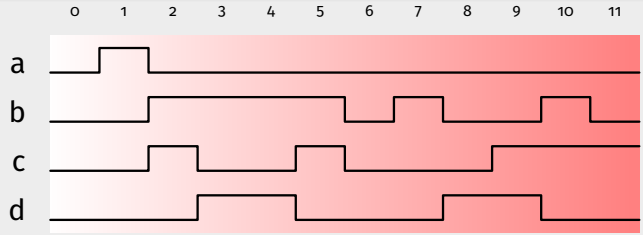
```
assert property ((a)|=>(b ##1 c[=3] ##1 d));

assert property ((a)|=>(b ##1 c[->3] ##1 d));
```

Combinaisons

Fonction	Description
`s1 and s2`	Les deux séquences doivent être observées
`s1 intersect s2`	Les deux séquences doivent être observées, et doivent se terminer en même temps
`s1 or s2`	L'une ou l'autre des séquences doit être observée
`s1 throughout s2`	s1 doit être vraie durant toute la durée de s2
`s1 within s2`	s1 doit être observée durant l'exécution de s2

s1 and s2

Exemple

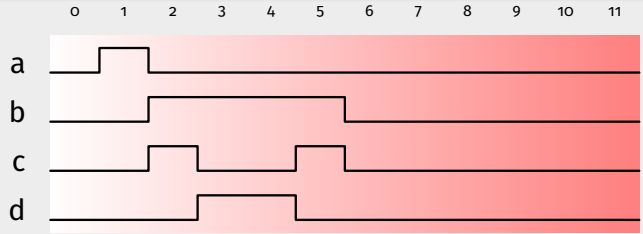


Assertions

```
assert property ((a)|=>((b[*4]) and (c ##1 d)));

assert property ((d[*2])|=>((c) and (b ##1 !b)));
```

s1 intersect s2

Exemple

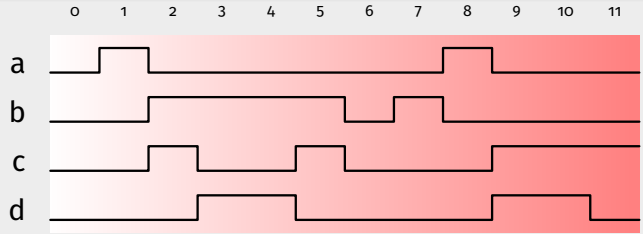


Assertions

```
assert property ((a)|=>((b[*4]) intersect (c ##1 d[*2] ##1 c)));

assert property ((a)|=>((b[*4]) intersect (c[1:2] ##1 d[1:3])));
```

s1 or s2

Exemple

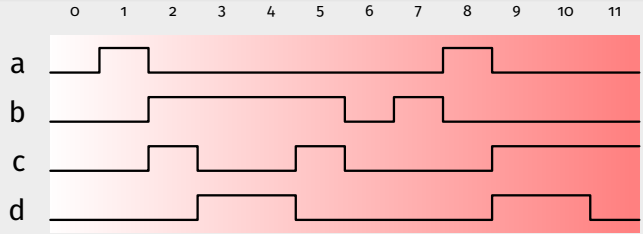


Assertions

```
assert property ((a)|=>((b[*4]) or (d ##1 c)));

assert property ((a)|=>((##1 d ##1 b) or (d[2*])));
```

s1 throughout s2

Exemple

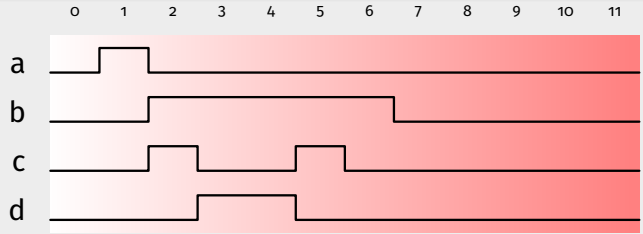


Assertions

```
assert property ((a)|=>((b) throughout (c ##1 d[*2] ##1 c)));

assert property ((b ##1 a)|=>((c) throughout (d[*2])));
```

s1 within s2

Exemple



Assertions

```
assert property ((a)|=>((c ##1 d[*2] ##1 c) within b[*5]));

assert property ((a)|=>((d[*2]) within (c ##3 c)));
```

Fonctions Système

Fonction	Description
`$onehot(exp)`	Retourne vrai si un et un seul bit de l'expression est à 1
`$onehot0(exp)`	Retourne vrai si au plus un bit de l'expression est à 1
`$isunknown(exp)`	Retourne vrai si au moins un bit de l'expression est à X ou Z
`$countones(exp)`	Retourne le nombre de 1 de l'expression

Fonctions utiles

Fonction	Description
`$rose(exp)`	Retourne vrai s'il y a eu un flanc montant
`$fell(exp)`	Retourne vrai s'il y a eu un flanc descendant
`$stable(exp)`	Retourne vrai si le signal est resté stable durant un cycle
`$past(exp)`	Retourne la valeur précédente de l'expression
`$past(exp,n)`	Retourne la valeur prise par l'expression *n* cycles avant

Binding

- De la même manière que pour les groupes de couverture, les propriétés et assertions peuvent être placées dans un module qui est ensuite *bindé* avec le module à tester

Bonne pratique: couverture

- Un des problèmes des assertions est que si la partie gauche d'une implication n'est jamais observée, la partie droite ne sera jamais évaluée
- Donc, de la couverture sur les parties gauches des implications permet de vérifier que chaque assertion a au moins été évaluée une fois

Exemple

```
// Assertion
acknowledge : assert property (disable iff(rst) ack |-> $past(req));


// Couverture
cover_acknowledge : cover property (~rst & ack);
```

Exemple 1

- Lorsqu'une requête (activation de `req`) est émise, alors un acknowledge (activation de `ack`) doit arriver au plus tard 5 cycles d'horloge après

Exemple 2

- Les entrées `a` et `b` doivent rester stables tant que `valid` est actif

Exemple 3 : FIFO


```
entity fifo is
generic (
    FIFOSIZE    : integer := 8;
    DATASIZE    : integer := 8
);
port (
    clk_i       : in    std_logic;
    rst_i       : in    std_logic;
    full_o      : out   std_logic;
    empty_o     : out   std_logic;
    wr_i        : in    std_logic;
    rd_i        : in    std_logic;
    data_i      : in    std_logic_vector(DATASIZE-1 downto 0);
    data_o      : out   std_logic_vector(DATASIZE-1 downto 0)
);
end fifo;
```

Exemple 3 : FIFO

- Après une écriture la donnée doit immédiatement être disponible
- `empty_o` est à '1' si le FIFO est vide
- `full_o` est à '1' si le FIFO est plein
- Il ne faut pas écrire une donnée si le FIFO est plein
- Il ne faut pas lire une donnée si le FIFO est vide

Exemple 3 : FIFO

Après une écriture la donnée doit immédiatement être disponible

```
// If a write occurs, the FIFO can not be empty the next clock cycle
assert_not_empty_after_write : assert property ( @(posedge clk_i)
  ( (wr_i) |=> !empty_o) );
```

Exemple 3 : FIFO

Pour la preuve, déclaration de deux compteurs

```
int wcnt = 0;
int rcnt = 0;


always @(posedge clk_i or posedge rst_i)
    if (rst_i)
        wcnt = 0;
    else if (wr_i)
        wcnt = (wcnt + 1);

always @(posedge clk_i or posedge rst_i)
    if (rst_i)
        rcnt = 0;
    else if (rd_i)
        rcnt = (rcnt + 1);
```

Exemple 3 : FIFO

Vérification de full et empty

```
property p_full;
    @(posedge clk_i)
        ( full_o == (wcnt == rcnt + FIFOSIZE) );
endproperty

assert_full : assert property (p_full);


property p_empty;
    @(posedge clk_i)
        ( empty_o == (wcnt == rcnt) );
endproperty

assert_empty : assert property (p_empty);
```

Exemple 3 : FIFO

Vérification de l'intégrité des données

```
property p_data_integrity;
    int cnt;
    logic[DATASIZE-1:0] data;
    @(posedge clk_i)
        (wr_i, cnt=wcnt, data=data_i) |=>
        ((##[0:$] (rd_i & (rcnt==cnt))) |->
         (data_o==data));
endproperty

assert_data_integrity: assert property (p_data_integrity);
```

Exemple 3 : FIFO

assume: hypothèses sur les entrées

```
// The following disable reads when FIFO is empty
assume property ( @(posedge clk_i) (!(rd_i & empty_o)));

// The following disable write when FIFO is full
assume property ( @(posedge clk_i) (!(wr_i & full_o)));

// The following assume ensures the end of the evaluation process
// Without it the solver can not formally check the design
assume property ( @(posedge clk_i) (wcnt < 4*FIFOSIZE));
```

Utilisation de QuestaFormal

- Les assertions peuvent être exploitées en simulation ou en preuve formelle
- QuestaFormal permet la preuve formelle
- Possibilité d'avoir des scripts pour la vérification formelle

Conclusion

- Les assertions permettent de:
  - Spécifier le comportement d'un système d'une autre manière
  - formellement prouver le bon fonctionnement du système
- Elles sont complémentaires au banc de test classique
- Peuvent être appliquées en black box ou white box
- Peuvent causer des chutes de cheveux