

## Objectives

The goal of this proposal is to introduce case statements inside assertions , for example:

```
property p1(logic a,b);
s(a,b) |=>
  case (val)
    3'b011: a ##1 b;
    3'b100: a |> b;
    default: a ##2 b;
  endcase
endproperty
```

Change log:

Addressing comments from JH (12-Feb-08):

1. Added `property_expr ::= property_statement`, this would allow code such that in the example above ( in the objective) without semicolon after the `endcase`. Adding this however basically says that you are free to add semicolon at the ends of `property_expr` or `property_statement` but you don't have to put them after `case` or `if-else` (so "if (b) p;;; else q" is legal), this is backward compatible but also allows inseting semicolons inside `if-else`.
2. Operator precedence for "case" isn't defined, this is consistent with the procedural statements operator precedence.

Addressing comments from DB (13-Feb-08):

1. At F.2.3.5: the `≡` symbol is missing

Addressing comments from DK (18-Feb-08):

1. removed : [The case expression given in parentheses shall be evaluated exactly once and before any of the case item statements.](#) From [16.12.17](#) since it seems to be related only to expression with side effects.
2. Fix fonts in h) and in 16.5.1 on page 4.- made "*(b!=b1 && b!=b2)*" *Italic in "h"*, and keywords to *Courier /Italic/9* in 16.5.1
3. changed h) on page 4 to refer to `expression_` instead of `b_`

18-Feb-08: shifted "16.12.17 case" to be before "16.12.16 recursive properties" in 1932 by note to editor

Addressing comments from BT (18-Feb-08):

1. "change isn't backwards compatible in terms of VPI" – made changes such that original VPI diagram is kept and add support for case operator in the property expr VPI diagram, the proposal now doesn't depend on more extension to VPI.

Addressing comments from BT (24-Feb-08):

1. add `#define vpiCaseProperty` <editor to fill in> and `#define vpiCasePropertyItem` <....> -- Note also rename it "case property item" instead ...

Addressing comments from DK (24-Feb-08):

1. Syntactical meta-symbols [] and {} are shown in new productions in black—made them blue
2. Page 4—" . The set of semantic leading clocks of case (b) b1:q1 b2:q2 endcase is {inherited}." The statement should have general form with n clauses and optional default.
3. F.2.3.5 Other derived operators (last page)  
I would prefer to provide a recursive definition:  
"( case ( b ) b1: P1 ... bn: Pn endcase ) ≡  
 ( if (b===b1) P1 ... else if (b===bn) Pn)" -->  
 "( case ( b ) b1: P1 b2: P1 ... bn: Pn endcase ) ≡  
 ( if (b===b1) P1 else case ( b ) b2: P2 ... bn: Pn endcase )", e.t.c.
4. note fonts, subscripts are sometimes rendered as regular fonts.

Addressing comments from SB (10-Mar-08):

1. "I don't think the BNF footnote is needed. Regular cases have the same restriction with such a BNF footnote." – removed the footnote

Addressing comments from JH (12-Mar-08):

1. extra semicolon in assertion statement –A possible solution is to have both property\_spec and property\_statement\_spec, where the former is unchanged, i.e.

property\_spec ::= [clocking\_event] [disable iff (expression\_or\_dist)] property\_expr

and the latter requires a property\_statement in place of the property\_expr.  
Then use property\_spec in the assertion statements (unchanged) and use  
property\_statement\_spec in the property declaration, getting rid of the  
explicit semicolon there.

Splitting the proposal to 3 as decided in SV-AC (18-Mar-08):

1. VPI (in Mantis 2326)
2. fundamental syntax and semantics (Mantis 2173)
3. changes raised by John for the Champions meeting (13-Mar-08) (Mantis 2327)

Addressing comments from SV-BC (12-May-08)

1. BNF appears to have problems between property\_statement and property\_expr
2. 'if' is not in bold/courier font in Annex F changes. Likely should be.
3. Does expression sizing follow same rules as case statement? Likely not stated in use with rand sequence (17.17.3 in Draft).

Aligning to Draft 6 (29-Jun-08):

1. changed 16.12 to 16.13 and Syntax box 16-14 to 16-16
2. "16.12.16 Case" to "16.13.16 Case"
3. Syntax box 16-17 to 16-19
4. F.2.3.5 to F.3.4.5

Addressing SV-BC comments (06-July-08)

1. changed "wait" which is used as a variable though it is a keyword to "delay"

The proposal has been aligned to P1800-2008draft6

It should also be noted that proposals 2326 and 2327 add VPI ,vacuity and multy clocks definition for the case property syntax.

## 16.13 Declaring properties

REPLACE in Syntax 16-16

```
property_declaration ::=  
    property property_identifier [ ( [ property_port_list ] ) ] ;  
        { assertion_variable_declaration }  
        property_spec ;  
    endproperty [ : property_identifier ]
```

```
property_expr ::=  
    ...  
    | if ( expression_or_dist ) property_expr [ else property_expr ]  
    ...
```

### WITH

```
property_declaration ::=  
    property property_identifier [ ( [ property_port_list ] ) ] ;  
        { assertion_variable_declaration }  
        property_spec ; property_statement_spec  
    endproperty [ : property_identifier ]
```

```
property_statement_spec ::=  
    [ clocking_event ] [ disable iff ( expression_or_dist ) ] property_statement
```

```
property_statement ::=  
    property_expr ;  
    | case ( expression_or_dist ) property_case_item { property_case_item } endcase  
    | if ( expression_or_dist ) property_expr [ else property_expr ]
```

```
property_case_item ::=  
    expression_or_dist { , expression_or_dist } : property_statement  
    | default [ : ] property_statement
```

```
property_expr ::=  
    ...  
    | if ( expression_or_dist ) property_expr [ else property_expr ]  
    | property_statement  
    ...
```

Add:

Note to editor: please insert before "16.13.16 Recursive properties" and shift the numbers accordingly

### 16.13.16 case

The *case* property statement is a multiway decision that tests whether a boolean expression matches one of a number of other boolean expressions and branches accordingly.

---

```
property_statement ::= //from A.2.10
...
| case ( expression_or_dist ) property_case_item { property_case_item } endcase
...

property_case_item ::=
    expression_or_dist { , expression_or_dist } : property_statement
    | default [ : ] property_statement
```

---

*Syntax 16-19—property statement case syntax (excerpt from Annex A)*

(Note to editor – please shift number of tables accordingly)

The *default* statement shall be optional. Use of multiple default statements in one property case statement shall be illegal.

A simple example of the use of the case property statement is the decoding of variable *delay* to produce a delay between the check of two signals as follows:

```
property p_delay(logic [1:0] delay);
    case (delay)
        2'd0: a && b;
        2'd1: a ##2 b;
        2'd2: a ##4 b;
        2'd3: a ##8 b;
        default: 0; // cause a failure if delay has x or z values
    endcase
endproperty
```

During the linear search, if one of the case item expressions matches the case expression given in parentheses, then the property statement associated with that case item shall be evaluated, and the linear search shall terminate. If there is a default case item, it is ignored during this linear search. If all comparisons fail and the default item is given, then the default item property statement shall be executed. If the default property statement is not given and all of the comparisons fail, then none of the case item property statements shall be evaluated and the evaluation of the case property statement from that start point succeeds and returns true (vacuously).

The rules for comparing the case expression to the case item expressions are described in [Note to editor – insert reference to 12.5 – case statement].

## A.2.10 Assertion declarations

### REPLACE

```
property_declaration ::=
    property property_identifier [ ( [ property_port_list ] ) ] ;
        { assertion_variable_declaration }
        property_spec ;
    endproperty [ : property_identifier ]

property_expr ::=
    ...
    | if ( expression_or_dist ) property_expr [ else property_expr ]
    ...
```

### WITH

```
property_declaration ::=
    property property_identifier [ ( [ property_port_list ] ) ] ;
        { assertion_variable_declaration }
        property_spec ; property_statement_spec
    endproperty [ : property_identifier ]

property_statement_spec ::=
    [ clocking_event ] [ disable iff ( expression_or_dist ) ] property_statement

property_statement ::=
    property_expr ;
    | case ( expression_or_dist ) property_case_item { property_case_item } endcase
    | if ( expression_or_dist ) property_expr [ else property_expr ]

property_case_item ::=
    expression_or_dist { , expression_or_dist } : property_statement
    | default [ : ] property_statement

property_expr ::=
    ...
    | if ( expression_or_dist ) property_expr [ else property_expr ]
    | property_statement
    ...
```

### F.3.4.5 Other derived operators

Note to editor : in Draft4 this was F.2.3.5

REPLACE

— ( **if** ( *b* ) *P*<sub>1</sub> **else** *P*<sub>2</sub> ) ≡ ( ( *b* | -> *P*<sub>1</sub> ) **and** ( !*b* | -> *P*<sub>2</sub> ) )

WITH

— ( **if** ( *b* ) *P*<sub>1</sub> **else** *P*<sub>2</sub> ) ≡ ( ( *b* | -> *P*<sub>1</sub> ) **and** ( !*b* | -> *P*<sub>2</sub> ) )

— ( **case** ( *b* ) *b*<sub>1</sub>: *P*<sub>1</sub> **endcase** ) ≡ ( **if** ( *specify*(*b*) === *specify*(*b*<sub>1</sub>) ) *P*<sub>1</sub> )

— ( **case** ( *b* ) **default**: *P*<sub>d</sub> **endcase** ) ≡ ( *P*<sub>d</sub> )

— ( **case** ( *b* ) *b*<sub>1</sub>: *P*<sub>1</sub> **default**: *P*<sub>d</sub> **endcase** ) ≡ ( **if** ( *specify*(*b*) === *specify*(*b*<sub>1</sub>) ) *P*<sub>1</sub> **else** *P*<sub>d</sub> )

— ( **case** ( *b* ) *b*<sub>1</sub>: *P*<sub>1</sub> ... *b*<sub>n</sub>: *P*<sub>n</sub> **endcase** ) ≡ ( **if** ( *specify*(*b*) === *specify*(*b*<sub>1</sub>) ) *P*<sub>1</sub> **else** **case** ( *specify*(*b*) ) *b*<sub>2</sub>: *P*<sub>2</sub> ... *b*<sub>n</sub>: *P*<sub>n</sub> **endcase** )

— ( **case** ( *b* ) *b*<sub>1</sub>: *P*<sub>1</sub> ... *b*<sub>n</sub>: *P*<sub>n</sub> **default**: *P*<sub>d</sub> **endcase** ) ≡ ( **if** ( *specify*(*b*) === *specify*(*b*<sub>1</sub>) ) *P*<sub>1</sub> **else** **case** ( *specify*(*b*) ) *b*<sub>2</sub>: *P*<sub>2</sub> ... *b*<sub>n</sub>: *P*<sub>n</sub> **default**: *P*<sub>d</sub> **endcase** )

Where *specify*(*b*) is a function which expands a Boolean expression *b* and treats it as signed or unsigned according to the rules mentioned in [Note to editor – insert reference to 12.5 – case statement ] for performing expression comparison while evaluating case statement.