1.1 Syntax on Page 150, Section 17.5, Syntax Box 17-2

Change

goto_repetition ::= **[*>** const_or_range_expression **]**

To

goto_repetition ::= **[*->** const_or_range_expression **]**

1.2 Typo on Page 178 in Section 17.11.1, first paragraph

Change

desalination

To

destination

1.3 Syntax on Page 176, Section 17.10, Syntax Box 17-14

disable iff (expression) should be within the brackets, not just disable iff

Change

property_spec ::=

[clocking_event] [**disable iff**] **(** expression **)** [**not**] property_expr

| [**disable iff (** expression **)**] [**not**] multi_clock_property_expr

To

property_spec ::=

[clocking_event] [**disable iff (** expression **)**] [**not**] property_expr

| [**disable iff (** expression **)**] [**not**] multi_clock_property_expr

1.4 Similar change as 1.3 on Page 280, Annex A - A.2.10

Change

property_spec ::=

[clocking_event] [**disable iff**] **(** expression **)** [**not**] property_expr

| [**disable iff (** expression **)**] [**not**] multi_clock_property_expr

To

property_spec ::=

[clocking_event] [**disable iff (** expression **)**] [**not**] property_expr

| [**disable iff (** expression **)**] [**not**] multi_clock_property_expr

1.5 Figure on Page 160, Section 17.7.4, Figure Box 17-5

Add a dashed box around the two green arrows showing matches on the last grid line

1.6 Figure on Page 164, Section 17.7.6, Figure Box 17-10

Add a dashed box around the five green arrows showing matches on the last grid line

1.7 Typo related to [* operator on Page 172-173 in Section 17.8

In this typo's Replace * [to [*

On Page 172 in the second example, change

```
sequence data_check;
   int x;
   a ##1 !a, x = data_in ##1 !b*[0:$] ##1 b && (data_out == x);
endsequence
property data_check_p
   int x;
   a ##1 !a, x = data_in |=> !b*[0:$] ##1 b && (data_out == x);
endproperty
```

To

```
sequence data_check;
   int x;
   a ##1 !a, x = data_in ##1 !b [*0:$] ##1 b && (data_out == x);
endsequence
property data_check_p
   int x;
   a ##1 !a, x = data_in |=> !b [*0:$] ##1 b && (data_out == x);
endproperty
```

On Page 173 in the second example, change

```
sequence sub_seq1;
   int v1;
   a ##1 !a, v1 = data_in ##1 !b*[0:$] ##1 b && (data_out == v1);
endsequence
```

To

```
sequence sub_seq1;
   int v1;
   a ##1 !a, v1 = data_in ##1 !b [*0:$] ##1 b && (data_out == v1);
endsequence
```

On Page 173 in the third example, change

```
sequence sub_seq2(lv);
   a ##1 !a, lv = data_in ##1 !b*[0:$] ##1 b && (data_out == lv);
endsequence
```

To

```
sequence sub_seq2(lv);
   a ##1 !a, lv = data_in ##1 !b [*0:$] ##1 b && (data_out == lv);
endsequence
```

On Page 173 in the fourth example, change

```
sequence sub_seq3(lv);
   int lv; // illegal since lv is a formal argument
   a ##1 !a, lv = data_in ##1 !b*[0:$] ##1 b && (data_out == lv);
endsequence
```

To

```
sequence sub_seq3(lv);
   int lv; // illegal since lv is a formal argument
   a ##1 !a, lv = data_in ##1 !b [*0:$] ##1 b && (data_out == lv);
endsequence
```

1.8 Typo on on Page 172 in Section 17.8

Change

1) When `valid_in` is true, `x` is assigned to pipe_in. Property e is true if five cycles later, `x` is equal to `(x+1)`.
   Property e is false if `pipe_out1` is not equal to `(x+1)`.

To

1) When `valid_in` is true, `x` is assigned to pipe_in. Property e is true if five cycles later, `pipe_out1` is equal to `(x+1)`.
   Property e is false if `pipe_out1` is not equal to `(x+1)`.

1.9 Unmatched parenthesis typo's on Page 157 in Section 17.7.2

Change

This is equivalent to:

```
a ##1 ((!b [*0:$] ##1 b)) [*min:max]) ##1 c
```

To

```
a ##1 ((!b [*0:$] ##1 b) [*min:max]) ##1 c
```

Change

This is equivalent to:

```
a ##1 ((!b [*0:$] ##1 b)) [*min:max]) ##1 !b[*0:$] ##1 c
```

To

```
a ##1 ((!b [*0:$] ##1 b) [*min:max]) ##1 !b[*0:$] ##1 c
```

1.10 Example on Page 167, Section 17.7.9

Last example in Section 17.7.9

Change

`trdy[*7]` **within** `(($fell irdy) ##1 irdy[*8])`

To

`!trdy[*7]` **within** `(($fell irdy) ##1 !irdy[*8])`

1.11 Examples on Page 183-184, Section 17.12.2

Change(page 183)

```
property r3;
   @(posedge sclk)(q != d);
endproperty
```


To

```
property r3;
   @(posedge mclk)(q != d);
endproperty
```

Change(page 183)

```
property r4;
   @(posedge sclk)(q != d);
endproperty
```

To

```
property r4;
   @(posedge mclk)(q != d);
endproperty
```

Change (page 184)

```
property r4;
   @(posedge sclk)(a==1) |-> (q != d);
endproperty
```

To

```
property r4;
   @(posedge mclk)(a==1) |-> (q != d);
endproperty
```

1.12 Semantic rules on Page 157, Section 17.7.2

Change

— expression `[*n:m]`, where `n` is the minimum, `m` is the maximum

To

— expression `[*m:n]`, where `m` is the minimum, `n` is the maximum, and m >= 0, m<= n or n is $

1.13 Complete the example on Page 184, Section 17.13

Change

— default clock, for example

default clocking `master_clk @(`**posedge** `clk);`

To

— default clock, for example:

```
default clocking master_clk @(posedge clk);
property p4; (a ##2 b); endproperty
assert property (p4);
```

1.14 Changes on Page 159 in Section 17.7.4

Change

The following example is an expression with the and operator, where the two operands are single sequence evaluations. The operation is illustrated in Figure 17-4.

To

The following example is an expression with the **and** operator, where the two operands are single sequence evaluations.

Change

Here, The two operand sequences are `(te1 ##2 te2)` and `(te3 ##2 te4 ##2 te5)`. The first operand sequence requires that first `te1` evaluates to true followed by `te2` two clock ticks later. The second sequence requires that first `te3` evaluates to true followed by `te4` two clock ticks later, followed by `te5` two clock ticks later. Figure 17-4 shows the evaluation attempt at clock tick 8.

To

The operation as illustrated in Figure 17-4 shows the evaluation attempt at clock tick 8. Here, the two operand sequences are `(te1 ##2 te2)` and `(te3 ##2 te4 ##2 te5)`. The first operand sequence requires that first `te1` evaluates to true followed by `te2` two clock ticks later. The second sequence requires that first `te3` evaluates to true followed by `te4` two clock ticks later, followed by `te5` two clock ticks later.

1.15 Change on Page 166 in Section 17.7.8

Change (last sentence on pp. 166)

If signal `burst_mode` were to be maintained low until clock tick 10, the expression would result in a match as shown in Figure 17-12.

To

If signal `burst_mode` were to be maintained low until atleat clock tick 10, the expression would result in a match as shown in Figure 17-12.

1.16 Change on Page 170 in Section 17.7.11

Change (paragraph just before Figure 17.14)

specifies looking for the rising edge of `frame` within two clock ticks in the future. After `frame` toggles high, `irdy` must also toggle high after one clock tick. This is illustrated in Figure 17-14. `'data_end_exp` is acknowledged at clock tick 6. Next, `frame` toggles high at clock tick 7. Since this falls within the timing constraint imposed by `[1:2]`, it satisfies the sequence and continues to monitor further. At clock tick 8, `irdy` is evaluated. Signal `irdy` transitions to high at clock tick 8, satisfying the sequence specification completely for the attempt that began at clock tick 6.

To

specifies looking for the rising edge of `frame` within two clock ticks in the future. After `frame` toggles high, `irdy` must also toggle high after one clock tick. This is illustrated in Figure 17-14 for the evaluation attempt at clock tick 6. `'data_end_exp` is acknowledged at clock tick 6. Next, `frame` toggles high at clock tick 7. Since this falls within the timing constraint imposed by `[1:2]`, it satisfies the sequence and continues to monitor further. At clock tick 8, `irdy` is evaluated. Signal `irdy` transitions to high at clock tick 8, satisfying the sequence specification completely for the attempt that began at clock tick 6.

1.17 Change on Page 182 in Section 17.12.2

Change (First example)


```
always @(posedge clk) begin
   <statements>;
   assert property (rule);
end
```

To

```
always @(posedge clk) begin
   <statements>;
   assert property (rule);
end
```

1.18 Change on Page 182 in Section 17.12.1

Change (last example)

```
module top(input bit clk);
   logic a,b,c;
   sequence seq3;
      @(posedge clk) b ##1 c;
   endsequence
   c1: cover property (seq3);
   ...
endmodule
```

To

```
module top(input bit clk);
   logic a,b,c;
   sequence seq3;
      @(posedge clk) b ##1 c;
   endsequence
   c1: cover property (seq3);
   ...
endmodule
```

1.19 Change on Page 185 in Section 17.13.

Change (Above Table 17-2)

These example sequences are used in Table 17-2 to explain the clock resolution rules for a sequence definition. The clock of any sequence when explicitly specified is indicated by X. The absence of a clock is indicated by a dash.

To

These example sequences are used in Table 17-2 to explain the clock resolution rules for a sequence definition. The clock of any sequence when explicitly specified is indicated by X. Otherwise, it is indicated by a dash.

1.20 Change on Page 188 in Section 17.14.

Change (last example on pp 188); remove bold font from “enable”

```
interface range (input clk,enable, input int minval,expr);
   property crange_en;
```



```
      @(posedge clk) enable |-> (minval <= expr);
   endproperty
   range_chk: assert property (crange_en);
endinteface
bind cr_unit range r1(c_clk,c_en,v_low,(in1&&in2));
```

To

```
interface range (input clk,enable, input int minval,expr);
   property crange_en;
      @(posedge clk) enable |-> (minval <= expr);
   endproperty
   range_chk: assert property (crange_en);
endinteface
bind cr_unit range r1(c_clk,c_en,v_low,(in1&&in2));
```

