

IEEE 1800-2009 provided global assert controls that affected all assertions in the design. It did not provide a way to control assertions based on their type. There was no way for the users to only disable **cover** directives but keep the **assert** and **assume** directives as enabled. Similarly, there was no way to only enable concurrent assertions and switch off immediate assertions. IEEE 1800-2012 made enhancements to allow those tight control features. The syntax for the assertion control syntax is as follows:

```

assert_control_task ::=
    assert_task [ ( levels [ , list_of_scopes_or_assertions ] ) ] ;
    | assert_action_task [ ( levels [ , list_of_scopes_or_assertions ] ) ] ;
    | $assertcontrol ( control_type [ , [ assertion_type ] [ , [ directive_type ]
        [ , [ levels ] [ , list_of_scopes_or_assertions ] ] ] ] ) ;
assert_task ::= $asserton | $assertoff | $assertkill
assert_action_task ::= $assertpasson | $assertpassoff | $assertfailon | $assertfailoff |
    $assertnonvacuouson | $assertvacuousoff
list_of_scopes_or_assertions ::= scope_or_assertion { , scope_or_assertion }
scope_or_assertion ::= hierarchical_identifier

```

Compatibility with
1800-2009

4.2.4.1 Assert control

Rule: The **\$assertcontrol** provides finer granularity in how and which types of assertions are controlled. The syntax is repeated below:

```

$assertcontrol ( control_type [ , [ assertion_type ] [ , [ directive_type ]
    [ , [ levels ] [ , list_of_scopes_or_assertions ] ] ] ] ) ;

```

[1] The arguments for the **\$assertcontrol** system task are described below:

— **control_type**: This argument controls the effect of the **\$assertcontrol** system task. This argument shall be an integer expression. Section 4.2.4.1.1 describes the valid values of this argument.

— **assertion_type**: This argument selects the assertion types that are affected by the **\$assertcontrol** system task. This argument shall be an integer expression. Section 4.2.4.1.2 describes the valid values for this argument. If **assertion_type** is not specified, then it defaults to all types of assertions and **expect** statements (i.e., Concurrent, Simple Immediate, Observed-Deferred Immediate, Final Deferred Immediate, and **expect**).

— **directive_type**: This argument selects the directive types (i.e., **assert**, **cover**, **assume**) that are affected by the **\$assertcontrol** system task. This argument shall be an integer expression. Section 4.1.4.1.3 describes the valid values for this argument. If **directive_type** is not specified, then it defaults to all types of directives.

— **levels**: This argument specifies the levels of hierarchy, consistent with the corresponding argument to the **\$dumpvars** system task (see 1800-2012 section 21.7.1.2). If this argument is not specified, it defaults to 0 (i.e., the specified module and in all module instances below the specified module). Example: **\$dumpvars (1, top);** // Because the first argument is a 1,

// this invocation dumps all variables within the module top;

// it does not dump variables in any of the modules instantiated by module top.

— **list_of_scopes_or_assertions**: This argument specifies which scopes of the model to control. These arguments can specify any scopes or individual assertions. For example,

```

module akill;    // /ch4/4.2/akill.sv
    bit clk, a,b;
    default clocking @(posedge clk); endclocking
    ap_kill: assert property (a |>= b) $assertkill(0, akill.ap_test_kill);
    ap_kill0: assert property(a |>= b) $assertcontrol(5, 15, 7, 0, akill.ap_kill0);

```

4.2.4.1.1 Control_type

The control type argument controls the effect of the `$assertcontrol` system task. The type of this argument is `integer`. The valid values for this argument are defined in Table 4.1.4.2.1. The following assertion and code is used in the explanation of the control types. See Section 12.3 for 1800'2018 package related to supporting the assertion control constants.

```
logic clk, a, b, c, d; // /ch4/4.2/asncontrol.sv , see /ch11/11.3/asncntrl.sv
function automatic void pass(); $display("ap1 pass"); endfunction : pass
function automatic void fail(); $display("ap1 fail"); endfunction : fail
// assertion ap1 is used in Table 4.2.4.1.1
ap1: assert property(@ (posedge clk) a |-> b) pass(); else fail();
```

// Assertion controls

```
let LOCK          = 1; // assertion control type
let UNLOCK        = 2; // assertion control type
let ON            = 3; // assertion control type
let OFF           = 4; // assertion control type
let KILL          = 5; // assertion control type
let PASSON       = 6; // assertion control type
let PASSOFF      = 7; // assertion control type
let FAILON       = 8; // assertion control type
let FAILOFF      = 9; // assertion control type
let NONVACUOUSON = 10; // assertion control type
let VACUOUSOFF   = 11; // assertion control type
```

// Assertion types

```
let CONCURRENT    = 1; // assertion_type, concurrent
let S_IMMEDIATE   = 2; // assertion_type, simple immediate
let D_IMMEDIATE   = 12; // assertion_type, Final and Observed deferred immediate
let EXPECT        = 16; // assertion_type, expect
let UNIQUE        = 32; // unique if and case violation
let UNIQUE0       = 64; // unique0 if and case violation
let PRIORITY      = 128; // priority if and case violation
let ALL_ASSERTS   = (CONCURRENT|S_IMMEDIATE|D_IMMEDIATE|EXPECT); // (i.e., 31)
```

See 1800-2012 Section 12.4.2
unique-if, unique0-if, and priority-if


// Assertion directives

```
let ASSERT        = 1; // directive_type for assertion control tasks
let COVER         = 2; // directive_type for assertion control tasks
let ASSUME        = 4; // directive_type for assertion control tasks
let ALL_DIRECTIVES = (ASSERT|COVER|ASSUME); // (i.e., 7)
```

Table 4.2.4.1.1 Control type values for \$assertcontrol

Control type values	Effect	Description
1	Lock	This prevents status change of all specified assertions and expect statements until they are unlocked. Thus, once an assertion is locked its assert control properties cannot be changed until it is first unlocked. Example, \$assertcontrol(LOCK, ALL_ASSERTS, ALL_DIRECTIVES, 0, ap1);
2	Unlock	This removes the locked status of all specified assertions and expect statements. \$assertcontrol(UNLOCK, ALL_ASSERTS, ALL_DIRECTIVES, 0, ap1);
3	On	This re-enables the execution of all specified assertions. This control_type value does not affect expect statements. Example, \$assertcontrol(ON); This enables all the assertions except those that are locked. The locked assertions remain in their current control states.
4	Off	This stops the checking of all specified assertions until a subsequent \$assertcontrol with a control_type of 3 (On). No new attempts will be started. Attempts that are already executing for the assertions, and their pass or fail statements, are not affected. Any queued or pending assertions are not flushed and may still mature. No new instances of assertions are queued. The assertions are re-enabled with a subsequent \$assertcontrol with a control_type of 3 (On). This control_type value does not affect expect statements. For example, \$assertcontrol(OFF); // using default values of all other arguments
5	Kill	This aborts execution of any currently executing attempts for the specified assertions and then stop the checking of all specified assertions until a subsequent \$assertcontrol with a control_type of 3 (On). This also flushes any queued pending reports of deferred assertions or pending procedural assertion instances that have not yet matured. This control_type value does not affect expect statements. For example, \$assertcontrol(KILL, CONCURRENT, ASSERT, 0); Kill currently executing concurrent assertions, but do not kill concurrent covers, assumes and immediate/deferred asserts/covers/assumes. Using the appropriate directive type values for thread second arguments.
6	PassOn	This enables execution of the pass action for vacuous and nonvacuous success of all the specified assertions. An assertion that is already executing, including execution of the pass or fails action, is not affected. For example, \$assertcontrol(PASSON, CONCURRENT, ALL_DIRECTIVES, 0, ap1); If assertion ap1 succeeds vacuously or nonvacuously, the pass() function is called. All other concurrent assertions retain their current assert controls, and are unaffected by this assert control.
7	PassOff	This stops execution of the pass action for vacuous and nonvacuous success of all the specified assertions. Execution of the pass action for both vacuous and nonvacuous successes can be re-enabled subsequently by \$assertcontrol with a control_type value of 6 (PassOn), while the execution of the pass action for only nonvacuous successes can be enabled subsequently by \$assertcontrol with a control_type value of 10 (NonvacuousOn). An assertion that is already executing, including execution of the pass or fails action, is not affected. By default, the pass action is executed. For example, \$assertcontrol(PASSOFF, CONCURRENT, ALL_DIRECTIVES); If any assertion succeeds vacuously or nonvacuously, the pass action block is not executed. Thus, the pass() function will not be called if assertion ap1 succeeds.

Control type values	Effect	Description
8	FailOn	This enables execution of the fail action of all the specified assertions. An assertion that is already executing, including execution of the pass or fails action, is not affected. This task also affects the execution of the default fail action block (i.e., <code>\$error</code> , which is called in case no else clause is specified for the assertion) For example, \$assertcontrol (FAILON, CONCURRENT, ALL_DIRECTIVES); If any concurrent assertion fails, the fail action block is executed. Thus, the fail() function will be called if assertion ap1 fails.
9	FailOff	This stops execution of the fail action of all the specified assertions until a subsequent \$assertcontrol with a control_type value of 8 (FailOn). An assertion that is already executing, including execution of the pass or fails action, is not affected. By default, the fail action is executed. This task also affects the execution of default fail action block. For example, \$assertcontrol (FAILOFF); If any assertion fails, the fail action block is not executed. Thus, the fail() function will not be called if assertion ap1 fails.
10	Non-vacuous On	This enables execution of the pass action of all the specified assertions on nonvacuous success. An assertion that is already executing, including execution of the pass or fail action, is not affected. For example, \$assertcontrol (NONVACUOUSON); // Thus if assertion ap1 succeeds nonvacuously (e.g., a==1, b==1), the pass() function is called.
11	Vacuous Off	This stops execution of the pass action of all the specified assertions on vacuous success until a subsequent \$assertcontrol with a control_type value of 6 (PassOn). An assertion that is already executing, including execution of the pass or fails action, is not affected. By default, the pass action is executed on vacuous success. . For example, \$assertcontrol (VACUOUSOFF,,,ap1); // Thus if assertion ap1 succeeds vacuously (e.g., a==0, b==X), the pass() function is not called.

 **Guideline:** In most simulation cases it is recommended to disable assertion checking until the testing environment is stabilized. Once that point is reached, assertion checking can be enabled. In addition, it is recommended to disable the pass action block because assertions are expected to succeed. The enabling of the action block is only needed for debugging or when the pass action block updates a module variable; thus, supplying pass information is burdensome. Below is an example:

```

event start_sim; // /ch4/4.2/assertion_control.sv
int count1=0, count2=0;
ap_x1: assert property(Px1) count1 <= count1 + 1'b1;
ap_x2: assert property(Px1) count2 <= count2 + 1'b1;
initial begin
    $assertcontrol(KILL); // Stop checking all assertions
    wait (start_sim); // wait for subsystem to be ready to start checking for assertions
    $assertcontrol(ON); // enable all assertions
    // disable all pass action blocks except those needed
    $assertcontrol(LOCK, ALL_ASSERTS, ALL_DIRECTIVES, 0, ap_x1); // lock any changes to ap_x1
    $assertcontrol(PASSOFF); // pass off for ap_x2
    $assertcontrol(UNLOCK, ALL_ASSERTS, ALL_DIRECTIVES, 0, ap_x1, ap_x2);
end

```

pass action block increments module

With above code, do not use the **assert(randomize(object))** because all assertions will be disabled during initialization, and the randomization of the desired variables will not take effect. Instead use:

```

if(!randomize(var1, var2, var3)) $error("randomization failure"); // list of variables
// If classes with rand variables are used, then use : (See Ch9/consecutive.sv for an example)
if (!class_instance.randomize()) $error("randomization failure"); // without UVM
if (!class_instance.randomize()) `uvm_error("MYERR", "error message"); // with UVM

```

4.2.4.1.2 assertion_type

Rule: [1] The `assertion_type` argument selects the assertion types that are affected by the `$assertcontrol` system task. This argument shall be an integer expression. The valid values for this argument are described in Table 4.1.4.1.2. Multiple `assertion_type` values can be specified at a time by Oring different values. For example, a task with `assertion_type` value of 3 (which is the same as Concurrent | SimpleImmediate) shall apply to concurrent and simple immediate assertions. If `assertion_type` is not specified, then it defaults to 31 (Concurrent | SimpleImmediate | Observed-DeferredImmediate | FinalDeferredImmediate | Expect) and the system task applies to all types of assertions and **expect** statements.

Table 4.1.4.1.2 Values for `assertion_type` for assertion control tasks

assertion_type values	Types of assertions affected (See 4.5, 4.6)	Assertion example
1	Concurrent	ap_c: assert property (a => b);
2	Simple Immediate	a_c: assert (a && b);
4	Observed Deferred Immediate	a_d: assert #0 (a && b);
8	Final Deferred Immediate	a_c: assert final (a && b);
16	Expect	ex_c: expect (a => b);
32	Unique	unique if (a==0) c <= b; //ch4/m_unique.sv else if (a== 1) c <= d;
64	Unique0	unique0 if (a==0) c <= b; else if (a== 1) c <= d;
128	Priority	priority if (a==0) c <= b; else if (a== 1) c <= d;

4.2.4.1.3 directive_type

Rule: [1] The `directive_type` argument selects the directive types that are affected by the `$assertcontrol` system task. This argument shall be an integer expression. The valid values for this argument are described in Table 4.1.4.1.3. This argument is checked only for assertions. Multiple `directive_type` values can be specified at a time by OR-ing different values. For example, a task with `directive_type` value of 3 (which is same as Assert|Cover) shall apply to **assert** and **cover** directives. If `directive_type` is not specified, then it defaults to 7 (Assert | Cover | Assume) and the system task applies to all types of directives.

Table 4.1.4.1.3 Values for `directive_type` for assertion control tasks

directive_type values affected	Types of directives
1	Assert directives
2	Cover directives
4	Assume directives

4.2.4.1.4 Equivalent assertion control system tasks

Rule: The `assert` tasks provide backward compatibility to IEEE 1800-2009 and includes the following keywords: `$assertoff`, `$assertkill`, and `$asserton`. Specifically,

- `$assertoff` stops the checking of all specified assertions until a subsequent `$asserton`. An assertion that is already executing, including execution of the pass or fail statement, is not affected. It is equivalent to:
 `$assertcontrol(4, 15, 7, levels [,list])`