Native Testbench Coding Guide

Version C-2009.06
June 2009

Comments?
E-mail your comments about this manual to:
vcs_support@synopsys.com.

SYNOPSYS®

Contents

1

Introduction

The purpose of this document is to guide users as they develop OpenVera code for use with Synopsys verification products. In this document, NTB refers to the Native Testbench technology integrated in VCS and Pioneer NTB products.

Though NTB integration is usually transparent, some of the semantics need minor modification in order to be consistent with the Verilog language.

This document includes the following sections:

- Vera to NTB Conversion Strategy
- NTB Syntax Differences
- NTB Semantic Differences
- NTB Verilog/VHDL (MX) Limitations

- SystemVerilog Considerations

You can find details for each tool and command-line option in the respective product’s user guide.

Details of all constructs are available in the respective LRM (language reference manual).

2

Vera to Native Testbench Conversion Strategy

This chapter contains the following topics:

- Estimating the Speedup with a Profile
- Vera Compile Strategy and Vera 2005.06
- Vera Port Names and Skew
- Editing Testbench Files
- NTB Syntax Check
- Debugging any Simulation Mismatches (Semantic Changes)

Estimating the Speedup with a Profile

NTB speeds up the testbench portion of the design, and when using the VCS interface between the testbench and RTL. Running a simulation profile reports the time consumed by Vera, and enables a speedup estimate to be made.

If the design spends 10% of the time in Vera, you can expect a small speedup; if the design spends 80% of the time in Vera, you can expect a large speedup.

When using VCS for the RTL and Testbench portions of the design, an additional speedup may be also available due to VCS optimizing the design and testbench together.

Vera Compile Strategy and Vera 2005.06

Change the Vera compile strategy to use the `vera -cmp -dep_check -f filelist` option, or a single file containing a series of `#include` statements. Verify that you can compile and run the design properly using Vera and the new compile strategy before compiling with NTB.

Validate the design using Vera 2005.06. Refer to the Vera documentation to resolve any issues.

Vera Port Names and Skew

Add the `–ntb_opts vera_portname` option to the testbench compile-time command to preserve Vera naming conventions. This causes NTB to name the vera-shell module `vera_shell`, and to name the ports `.interfacename_portname`. These names are compatible with Vera.

Zero skew interfaces can expose race conditions between the design and the testbench. Add a `#-1` input skew, if required, and verify that the testbench operates correctly.

Editing Testbench Files

Using the workarounds detailed in this document, modify the testbench code to enable compilation in both Vera and NTB.

In general, code should be modified so that it runs in both Vera and NTB. Some code may be different for each tool, or not present for NTB. For these changes, use a preprocessor construct as shown below. NTB defines `SYNOPSYS_NTB` to OpenVera codes for any compilation containing `–ntb` options.

```
#ifdef SYNOPSYS_NTB
// NTB specific code here
#else
// Vera specific code here
#endif
```

To use the same `SYNOPSYS_NTB` macro definition in the Verilog code, add `+define+SYNOPSYS_NTB` to VCS compilation options.

NTB Syntax Check

In some cases, NTB syntax checking is stricter than Vera, as described in the following sections:

- NTB with Big Endian
- Illegal Tokens Between ifdef and endif
- Naming Variables in NTB

NTB with Big Endian

For ports defined as `[0:31]`, Vera converts it to `[31:0]` in the generated template interface file. NTB keeps it as `[0:31]` which will introduce a compilation error.

Illegal Tokens Between ifdef and endif

Illegal tokens between `#ifdef` and `#endif` cause errors (generated by the NTB preprocessor). For example:

```
class test{
bit a;
#ifdef FALSE
static char char_array[16]={'0','1'};
#endif
}
```

Naming Variables in NTB

NTB, unlike Vera, does not support variables to be named the same as the type. In this case, `pc_format` is both a type and a variable. Change it to `pc_format pc`, and it compiles fine.

For example:

```
#include <vera_defines.vrh>
enum pc_format = PC1,PC2,PC3;
class A
{
   string c;
   pc_format pc_format;//NTB syntax error. Change variable
name to others than type name “pc_format”
   task new(integer i=3,pc_format pc_format)
   {
      this.pc_format = pc_format;
   }
}
program t
{
 A a = new(3,PC1);
}
```

Debugging any Simulation Mismatches (Semantic Changes)

Simulation mismatches can occur due to the different ways Vera and NTB handle low-level signal interactions, thread ordering, and random generation.

You can debug these differences using waveforms, the debugger, and by following the guidelines specified in section entitled, "Coding Style Recommendations (Semantics)".

3

Coding Style Workarounds (Syntax)

NTB does not support certain OpenVera coding constructs that Vera supports. This section lists these constructs and provides workarounds.

Features Not Currently Supported

NTB in the VCS2006.06 release stream does not support the following Vera features. However, these features may be supported in a future release.

Vera 6.x 2005.06 Language Features

NTB does not currently support the following language features:

- object finalize
- global save and restore of random number generation state
- Vera-SystemC TLI
- Task/function overloading
- Distribution function system calls:

  - `Random_range()`
  - `Rand_normal()`
  - `Rand_poisson()`
  - `Rand_t()`
  - `Rand_exponential()`
  - `Rand_erlang()`
  - `Rand_chi_square()`

Dynamic signal_connect() and vcon

NTB does not support signal connects to an arbitrary HDL node using cross-module references at runtime (referred to as dynamic signal connects). However, NTB supports the mechanism of using signal connects to connect a port variable to an existing OpenVera interface variable (referred to as static signal connects).

The dynamic signal connect feature requires that the simulator keep hooks to all the Verilog variables as the OpenVera code could access any of these variables at runtime. As a result, it turns off some simulator optimizations and reduces the performance. For this reason, NTB does not support this feature and you should avoid using it in OpenVera.

Note:

> You should also avoid assigning objects to virtual ports using `cast_assign`. This might cause runtime errors if null handles are present.

NTB does not support OpenVera `.vcon` files. You can use `hdl_connect` in the interface instead.

```
//Vera .vcon file
connect ifc.in1 = "dut_test_top.dut.sig1"
//NTB
interface ifc1 {
…
     input in1 PSAMPLE… hdl_connect "dut_test_top.dut.sig1";
…
}
```

Simulation Command-line Arguments

Vera uses the `get_plus_arg` function to check for the presence of a runtime plus argument and to retrieve the value of a runtime plus argument. NTB supports `get_plus_arg()` with CHECK, HNUM and NUM. NTB does not support STR mode.

Verilog uses the `test_plusargs()` function to determine if a plus argument is specified, and uses the `value_plusargs()` function to retrieve the value of a plus argument.

Example

```
program test {
   bit [2047:0] b; // 256 char bit-string
   string s;
   integer i;
// Native Testbench Code
   #ifdef SYNOPSYS_NTB
      if (value_plusargs("my_str=%s", s)) {
         printf("got my_str with value = %s\n", s);
      }
   #else
      if (get_plus_arg(CHECK, "my_str")) {
         b = get_plus_arg(STR, "my_str");
         s.bittostr(b);
         printf("got my_str with value = %s\n", s);
      }
}
```

Variable Width Part Selects Corner Cases

NTB supports part selects with non-constant width. However, the following limitations apply:

- NTB does not support variable part select in concatenation/ replication operators, including:

```
result = {..,inp[m:l],..}
result = .. + {8{inp[m:l]}};
```

- NTB does not support variable part selects as arguments to reduction operators, including:

```
unary bit negation result = ~inp[m:l]
unary bit AND result = &inp[m:l]
unary bit NAND result = ~&inp[m:l]
unary bit XOR result = ^inp[m:l]
unary bit XNOR result = ~^inp[m:l]
```

- NTB does not support variable part selects in `@( )` statements.

Restricted Expects

NTB does not support restricted expect statements. Recode the restricted expect as a simple expect with a check on the first value change.

```
//OpenVera
@@@0,100 bus.$data == 2’b01;  //restricted expect
//NTB
fork
@0,100 bus.$data === 2’b01;
@(bus.$data) if (bus.$data !== 2’b01) printf(“error”);
join any
```

The timeout Construct

NTB does not support the `timeout` construct. You can replace the timeout construct by a `fork/join` with a `delay` or `wait` task for a specific clock domain. This solution works in both Vera and NTB.

```
// OpenVera Code
event e;
timeout (e, 1000);
sync(ANY, e);
// Native Testbench Code (works in Vera)
event e;
fork
{
repeat (1000) @(posedge CLOCK);
error ("Sync timed out\n");
}
sync(ANY, e);
join any
terminate(); // Kill all child processes.
```

Alternatively, you can create a macro to wrap the code.

The sync Function with an ORDER Argument

NTB does not support the `sync()` with an ORDER argument. If the design triggers the events in the correct order, you can change the code to use several calls to `sync(ALL, )` in the desired sequence.

```
// Old code
sync(ORDER, event_a, event_b, event_c);

// New code
sync(ALL, event_a);
sync(ALL, event_b);
sync(ALL, event_c);
```

If the design triggers the events in a different order and Vera produces a runtime error, you will need to rewrite the code using `fork/join` and `if` statements to check that the events occur in the correct order.

Outside coverage_group Definitions Are Not Supported

In OpenVera you can declare a `coverage_group` inside a class and define it outside the class as shown in the example below:

```
//OpenVera code
Class A {
     Coverage_group Cov;
}

Coverage_group A::Cov {
     Statement_here
}
```

This kind of coding style is not supported yet in NTB. You need to move the definitions into the class.

Non-constant Expressions in Accessing Multi-dimensional Arrays

NTB-OV does not support the use of non-constant expressions to access multi-dimensional arrays. But you can workaround this limitation as follows:

```
 a = b[i-:2];
```

In this example, the construct inside the range `VEC[EXPR+:C]` or `VEC[EXPR-:C]` is called using the indexed part select, where:

- `VEC` is any vector ID.
- `EXPR` is any valid Verilog expression representing the starting index.
- `+/-` is used depending on ascending/descending declaration range for `VEC`.
- `C` is the constant width of the select.

Features With No Planned Support

NTB does not support the following Vera features.

Region

NTB and SystemVerilog do not support region constructs. A region class library that has similar functionality is available for NTB. An example is provided in section 3.3.1.

Functional Coverage Sampled Argument Port Slices

OpenVera functional coverage supports sampling of passed-in parameters whose actual arguments are bit slices of port members. NTB does not support this corner case.

An example of functional coverage is as follows:

```
// interface I with signal w
// virtual-port P with members pw and pwSel
// bind B binds pw to I.w, and pwSel to I.w[6:3]
coverage_group MyCov (sample bit [7:0] param1,
sample bit [4:0] param2,
sample bit [7:0] param3,
sample bit [4:0] param4,
sample bit [4:0] param5,
sample bit [2:0] param6) {
sample I.w, I.w[6:3], P.$pw, P.$pw[6:3], P.$pwSel[2:1];//
Supported in 7.2
sample param1, param2, param3, param4, param5, param6;
}
MyCov cObj = new (I.w, // Supported since 7.2
I.w[6:3], // Supported since 7.2
b.$pw, // Supported since 7.2
b.$pwSel, // Supported since 7.2
b.$pw[6:3], // Not supported in NTB
b.$pwSel[2:1] // Not supported in NTB
);
```

Single Virtual Port Used for Monitor and Driver

In OpenVera, members of instances of a given port type can be bound to interface signals of different depths and widths while NTB issues a warning and sets the depth and width to the higher value.

```
interface ifc {
input a_in PSAMPLE;
output b_out PHOLD;
input [2:0] c_in PSAMPLE;
output [2:0] d_out PHOLD;
}
// Avoid this code style with 1 port
port my_port { sig; }
bind my_port bind1 { sig ifc.a_in; }
bind my_port bind3 { sig ifc.c_in; } // NTB Width Warning
```

OpenVera System Functions

NTB does not support the following OpenVera system functions:

- `error_mode()`
- `get_shell_param()`
- `string.get_status`
- `string.get_status_msg`
- `get_env`

For the `rand_mode` and `constraint_mode`, the string specifying the variable or constraint name must be a constant. For example, replace the variable string with a case statement and fixed strings:

```
case (string) {
CONSTRAINT_NAME1: constraint_mode(OFF,
“CONSTRAINT_NAME1”);
CONSTRAINT_NAME1: constraint_mode(OFF,
“CONSTRAINT_NAME2”);
CONSTRAINT_NAME1: constraint_mode(OFF,
“CONSTRAINT_NAME3”);
}
```

semaphore/mailbox_get() – constant wait mode

In NTB, the first argument to a `semaphore_get()` or `mailbox_get()` command must be a constant. This is not requested in Vera.

The workaround is to add a case statement; the new code will work in Vera and NTB. You can add `check` and `data` arguments as required.

```
// Original Code
// result = mailbox_get(mode, id, data);
case (mode) {
WAIT: result = mailbox_get(WAIT, id, data);
NO_WAIT: result = mailbox_get(NO_WAIT, id, data);
COPY_WAIT: result = mailbox_get(COPY_WAIT, id, data);
COPY_NO_WAIT: result = mailbox_get(COPY_NO_WAIT, id, data);
}
```

Value Change Alert

NTB does not support the value change alert (VCA) construct. You must perform value change alert using procedural code inside a `fork/join` construct.

Dynamic VRO Loading

NTB compiles OpenVera code with the design using VCS technology, therefore, it does not use or support VRO format files. NTB supports separate compilation of OpenVera code to shared object (`libtb.so`). It is necessary to load this shared object with `simv` during runtime.

Vera System Verifier (VSV)

NTB does not support the VSV functionality. If you plan to use VSV with NTB, contact VCS Customer Support.

Vera UDF

NTB does not support Vera UDF functionality. However, NTB does support calls to C code using DPI.

Mailbox receive with WAIT

NTB does not support `mailbox_receive()` with the WAIT argument. You can use the `mailbox_put()` and `mailbox_get()` functions, which can accomplish the same functionality.

With Port Syntax

NTB does not support the `'with port'` syntax. You should recode this using virtual ports. You can use virtual ports in both Vera and NTB.

New Features in Vera 2005.12

Function Calls in Constraints

This feature is not supported yet in NTB. However, VCS 2008.09 does support this feature.

Enhancement for solve-before and solve-before-hard Constructs

In Vera 2005.12, random numeric arrays (`bit vector`, `integer`, or `enum`) and object members can be passed as arguments to the `solve-before` and `solve-before-hard` constructs. All array types (fixed size, dynamic and associative) and smart queues are supported.

Currently, NTB does not support this feature, however, VCS 2008.09 does currently support this feature.

Smart Queue pick() and pick_index()

`Pick()`

In Vera 2005.12, the `smart queue method pick()` supports an optional `with` expression that determines the random element selection criteria and finds the random element satisfying the expression.

Currently, NTB does not support this feature.

`Pick_index()`

In Vera 2005.12, the `smart queue method pick_index()` supports an optional `"with"` expression that determines the random element selection criteria and finds the random index satisfying the expression.

Currently, this feature is not supported in NTB.

Cross Coverage Bin Enhancement

Vera2005.12 includes improved cross coverage modeling. The `cross bin` definition syntax and the semantics for interpreting the expression in the definition have changed. Refer to the Vera2005.12 release notes for detailed information.

Version VCS2006.06 and above support the updated syntax and semantics.

SystemVerilog Auto Binning

In Vera2005.12, the automatic coverage bin creation feature for creating state bins follows the SystemVerilog style.

Version VCS2006.06 and above function the same as Vera2005.12 for auto bin creation.

Guards Added for Cross and Sample Levels

In Vera2005.12, in addition to guards being specified at the bin levels, you can specify guards at the sample and cross levels. For more information refer to the Vera2005.12 release notes.

VCS2006.06 provides the same capabilities.

Impact of Guard Changes on Auto Cross Bin Creation

Version Vera2005.12 includes changes to the behavior of auto bin creation with guard. For more information refer to the Vera2005.12 release notes.

Version VCS2006.06 and above function the same as Vera2005.12.

Coverage Shapes in Instance Merging

In Vera2005.12, the concept of `shapes` was introduced to match identical instances of coverage groups so that merging produces useful results. When instances of coverage groups are to be merged, a mechanism is needed not only to ensure that the targeted instances are of the same coverage group, but also that the instances have the same parameter(s), because instances of the same coverage group can be instantiated with different parameters. For more information refer to the Vera2005.12 release notes.

Version VCS2006.06 and above provide the same feature.

New Default Value for coverage_goal

In Vera 2005.12, when specified for a `coverage_group`, the `coverage_goal` attribute designates the desired coverage percentage for the group. The default value is now 100%.

Version VCS2006.06 and above set the same value (100%) as the default value for the `coverage goal`.

New Features in Vera 2006.12

Object Allocation in Randomize

Version Vera2006.12, now includes a mechanism to grow and shrink arrays within the layered framework of the constraint solver. Additionally, version Vera2006.12 includes new keyword constructs to dynamically allocate objects in the constraint solver. For more information, refer to the Vera2006.12 release notes.

- Dynamic Scalar Arrays

Vera2006.12, includes a mechanism to grow and shrink scalar arrays within the layered framework of the constraint solver. No code change is required, the rand associative, dynamic and SmartQ arrays will grow and shrink based on the solution of the `size()` constraints within a randomize call.

Currently, version VCS2006.06 does not include this mechanism.

- Dynamic Object Arrays

Vera2006.12 includes an extensible enhancement to grow and shrink arrays of objects. For more information, refer to the Vera2006.12 release notes.

Version VCS2006.06 does not support this feature.

Modified Default Constraint Behavior

Consistent with SystemVerilog, Vera2006.12 includes changes to the behavior of disabling a default constraint expression when it contains multiple random variables. For more information, refer to the Vera2006.12 release notes.

VCS2006.06 deals with default constraint as the previous versions of Vera.

VCS 2008.09 deals with the default constraint behavior as per the enhancements in Vera 2006.12.

Keyword `“with default”` for Constraint Blocks

In Vera2006.12, constraint block expressions can be appended with the keywords `with default` and are defined as constraints inside a `with default` constraint block. These constraints have the property so they do not override default constraints like normal constraints. For more information, refer to the Vera2006.12 release notes.

VCS2006.06 does not support this feature.

Final Constraint Blocks

Vera2006.12 constraint blocks have been enhanced with a final keyword. Using this keyword before the block name specifies that this constraint block cannot be overridden in any of its derived objects.

VCS2006.06 does not support this feature. VCS 2008.09 does support this feature.

Coverage Transition Bin Definition Using State Bin Names

In Vera2006.12, the transition bin specification syntax has been enhanced to take state bin names as states in a transition sequence. For more information, refer to the release notes.

VCS2006.06 does not support this feature.

Ignored Added for States at Sample Levels in Coverage

In Vera2006.12, in addition to ignored bins being supported at the cross levels, they can now be specified at state levels in samples. For more information, refer to the release notes.

VCS2006.06 does not support this feature.

BETA FEATURE: OOP and Aspect Extensions for Coverage Group and Sample

As a beta feature, Vera 2006.12 adds AOP and OOP support for coverage groups and samples. For more information, refer to the release notes.

VCS2006.06 does not support this feature.

Semaphore Checking Key Function

In Vera2006.06, the `semaphore_get()` function has been enhanced with the `NUM_CHECK` option to query the number of keys available in the semaphore.

VCS2006.06 does not support this feature.

System Function vera_bits()

Vera2006.06 introduces a new system function, `vera_bits()`, which returns the number of bits required to hold a variable as a bit stream.

VCS2006.06 does not support this system function.

Others

Expect Statements with x/z Comparison

Comparing x’s and z’s in expect statements is incompatible with NTB and Verilog. In Vera, expect statements that contain the operators "==" or "!=", x’s and z’s are treated as "don’t cares". This is unlike Verilog behavior, and is not consistent with Vera x/z behavior outside of expects. Vera does not allow any other operator, such as "===" in expect statements, to explicitly check x and z behavior. This leads to mismatches when the expected value is x or z. NTB allows all operators except the "<=" operator in expect statements so that you can specify explicit behavior. If you want to specifically check for x or z, the recommended style is to use the "===" operator. To ignore x’s and z’s, you can use wild card operators "=?=".

Note: NTB is compatible with Verilog.

OpenVera DirectC

NTB supports both DirectC and DPI interfaces, but DPI is recommended with NTB.

Sample Region Class Code for NTB

```
#include <vera_defines.vrh>
class Region_Control {
   static integer region_id_counter ;
   integer  my_region_id ;
integer sa,sb,sc;   //semapore a,b,c

task new() {
   if (region_id_counter === 32'hx)
      region_id_counter = 1;
   else
      region_id_counter++;
   my_region_id = region_id_counter;

   sa = alloc(SEMAPHORE,0,1,1);
   sb = alloc(SEMAPHORE,0,1,1);
   sc = alloc(SEMAPHORE,0,1,1);
}

function integer region_alloc(
      integer kind, integer initial, integer num ) {
   if (region_id_counter === 32'hx ||
         my_region_id === 32'hx) {
      printf("You need to create the object with
         new() task first\n");
      exit(1);
   }
   region_alloc = my_region_id;
}

task region_exit(integer kind, reg [31:0] a  = 32'hx ,
      reg [31:0] b =32'hx, reg [31:0] c = 32'hx) {
   // Assume 3 optional args are typical, probably better
   // to have 10
   if (kind != my_region_id)
      error("User Region_Control: This is not the
         correct object reference \n");

   if (a !== 32'hx ) {
      semaphore_put(sa,1);
   }
   if (b !== 32'hx ) {
      semaphore_put(sb,1);
   }
   if (c !== 32'hx ) {
      semaphore_put(sc,1);
   }
```

```
}

function integer region_enter(integer wait_nowait,
      integer kind, reg [31:0] a  = 32'hx ,
      reg [31:0] b =32'hx, reg [31:0] c = 32'hx) {
   // Assume 3 optional args are typical, probably better
   // to have 10
   integer avail_a, avail_b,avail_c;

   avail_a = 1; avail_b = 1; avail_c = 1;
   region_enter = 1;

   if (kind != my_region_id)
      error("User Region_Control: This is not the correct
         object reference \n");

   if (wait_nowait == NO_WAIT ) {
      if (a !== 32'hx ) {
         avail_a = semaphore_get(NO_WAIT,sa,1);
      }
      if (b !== 32'hx ) {
         avail_b = semaphore_get(NO_WAIT,sb,1);
      }
      if (c !== 32'hx ) {
         avail_c = semaphore_get(NO_WAIT,sc,1);
      }

      region_enter = avail_a&&avail_b&&avail_c;
   }
   else {  // assume WAIT
      if (a !== 32'hx) {
         if(b !== 32'hx) {
            if(c !== 32'hx) {
               fork
                  semaphore_get(WAIT,sa,1);
                  semaphore_get(WAIT,sb,1);
                  semaphore_get(WAIT,sc,1);
               join
            }
            else {
               fork
                  semaphore_get(WAIT,sa,1);
```

```
                     semaphore_get(WAIT,sb,1);
                  join
               }
            }
            else {
               semaphore_get(WAIT,sa,1);
            }
         }
      }
   }
}
```

4

Coding Style Recommendations (Semantics)

Print Format [Semantic Difference]

Vera allows `printf` statements to have more format specifiers than arguments. This is incompatible with Verilog. In NTB, the number of format specifiers must match the number of arguments.

NTB/Verilog prints an unknown value for an integer data type as `x` (lowercase) while Vera prints `X` (uppercase).

You should use the subset of the `printf()` functionality supported by the Verilog `$write` function in the code.

Data Type integer (NTB Compatibility with SV)

In Vera, an integer is an X or a 2-state variable, while in Verilog an integer is a 4-state variable equivalent to a signed reg[31:0]. Therefore, Vera evaluates expressions with any `X` or `Z` for an integer-type operand to `X` whereas NTB follows Verilog semantics by evaluating bit by bit for `&`, `|`, and `^`.

A similar difference arises in the case of assignment of any `bx` or `bz` to integers in Vera versus NTB. Vera fills the entire integer with `x`, and NTB follows Verilog semantics in interpreting the integer as a `reg[31:0]`; therefore, only those bits that are actually assigned are made x or z. For example:

```
integer a;
reg [31:0] r = 32'hx10z0000;
a = r;
printf("%h\n", a);
// Vera prints : X
// Native Testbench prints: x10z0000
```

To be compatible with Verilog, avoid using bit-fields of integers in OpenVera. When you need to use a bit-field, you can use a `reg[31:0]` to remain Verilog-compatible.

When you pass an integer to a large bit vector array, NTB performs sign extension, while Vera adds leading zeroes as needed. The solution is to explicitly pad the call with a leading 0 for NTB.

```
my_task(bit [63:0] arg); // task definition
my_task({1'b0, intger_arg}); // task call
```

Initialization Order of Enumerated Types (NTB Compatible with SV)

NTB implicitly initializes an uninitialized variable of an enumerated data type to 0. However, Vera initializes such a variable to `X`.

```
enum Color { Red = 10, Green = 20, Blue = 30 };
program main {
   Color c; // uninitialized
   printf("%d\n", c);
   // NTB prints 0;
   // Vera prints (ENUM_VAR:X)
   printf("%s\n", c);
   // NTB prints empty;
   // Vera prints ** UNDEFINED **
}
```

Ternary Operator Handling with X and Z (NTB Compatible with SV)

Ternary statements such as `o = control ? a : b`; behave differently in Vera and NTB.

- If `control` does not contain an `X`, Vera and NTB behave the same.
- If `control` contains an `X`, Vera always treats the `control` as false.
- If `control` contains an `X`, NTB examines the other bits.
  - If any bit contains a `1`, control is true.
  - If all other bits are `0`, control is false.

- If all bits are `X`, control is false.

Control	Vera	NTB
0000	False	False
1111	True	True
000X	False	False
111X	False	True
XXXX	False	False

Avoid X in control signals, or explicitly checking whether using an X is possible. Use `===` or `=?=` to explicitly check if an X is possible.

Initial Drive [NTB Compatible with SV]

With Vera, interface output signals are Z at time 0. In NTB, output signals are Z, and become Z at a time given by the output skew. You must modify the code if a check at startup time expects a Z value. You should ensure that checks, during the initial simulation skew time, check for either X or Z.

Differences in Soft Drive Behavior

In Vera, driving signal is `strong` by default, and `soft` when specified. Vera resolves drives using signal strengths. If a signal is driven with two conflicting values, a runtime error occurs.

In NTB, if two strong drives occur in the same timestep, the last drive persists, no signal resolution is performed, and no runtime errors occur.

For NTB, you must modify code that has several threads driving a signal with a soft value, and one thread driving with a strong value. Several scenarios are possible:

Example 4-1   Strong and soft drive in the same timestep

Vera: The strong drive persists (a = 0 below)

NTB: The last drive persists (a = 1 below)

```
fork
{
   intf.a = 0;
}
{
suspend_thread();
intf.a = 1 soft;
}
join all
```

Example 4-2   Strong drive and Z drive in the same timestep

Vera: The strong drive persists (a = 0 below)

NTB: the last drive persists (a = Z below)

```
fork
{
   intf.a = 0;
}
{
   suspend_thread();
   intf.a = 'bz;
}
join all
```

Example 4-3   Conflicting strong drives

Vera: A runtime error occurs

NTB: The last drive persists (a = 1 below)

```
fork
{
   intf.a = 0;
}
{
   suspend_thread();
   intf.a = 1;
}
join all
```

Example 4: Conflicting Soft Drives

Vera: a = X without a simulation error

NTB: The last drive persists (`a = 1` below)

```
fork
{
   intf.a = 0 soft;
}
{
   suspend_thread();
   intf.a = 1 soft;
}
join all
```

Differences in the Fork/Join Thread Order Behavior

In Vera, `join any` and `join all` constructs have an implicit `suspend_thread` task call after the join. Vera suspends the parent thread and executes it later in the current time instead of executing it immediately.

In NTB, `join any` and `join all` constructs do not have an implicit `suspend_thread` after the join. The parent thread starts execution immediately after the join condition is met.

The following example illustrates that NTB would print `BCDA` and Vera would print `BCAD`.

```
program main {
   fork
      fork
         printf("A");
      join none
      printf("B");
      printf("C");
   join all
   // Vera has implicit suspend_thread(); here
   printf("D");
   wait_child();
}
```

This typically results in log files with messages from the same timestep being issued in a different order. Do not rely on thread ordering as the OpenVera language standard does not specify the execution order of threads.

Differences in Events, Sync, Wait_var Behavior

Vera event synchronization uses a LIFO mechanism (the last sync to wait on an event is the first sync to wake up). NTB event synchronization uses a FIFO mechanism (first sync to wait on an event wakes up first). Interaction between ON or OFF and ONE_SHOT or HAND_SHAKE are different in NTB and Vera. In NTB, level values (ON or OFF) override the edge values (ONE_SHOT, HAND_SHAKE). In Vera, this is not the case. The following example illustrates the difference between NTB and Vera. NTB loops forever, but Vera finishes:

```
#include <vera_defines.vrh>
program main {
   event e;
   trigger(ON, e);
   trigger(HAND_SHAKE, e);
   trigger(OFF, e);
   sync(ALL, e); // e is OFF but in Vera this sync unblocks!
}
```

ONE_BLAST Corner Case [Semantic Difference]

Vera supports the ONE_BLAST event. In NTB, a ONE_BLAST event stays ON until the end of the timestep, whereas in Vera, it stays ON for the current invocation of Vera. If you invoke Vera multiple times in the same timestep (unlikely, but a possible scenario), there will be differences in behavior with NTB. For example, if you call a Verilog task in OpenVera code, this Verilog task will return from Verilog to OpenVera in the same timestep. The ONE_BLAST is not ON for Vera, but is still ON for NTB after you return from Verilog.

Differences in Behavior of Repeat Loops with Negative Arguments

Vera treats a count in a repeat loop as an unsigned number, whereas NTB treats a count in a repeat loop as a signed number. In Vera, `repeat (-1)` loops forever, whereas NTB does not enter the loop.

Differences in Bit Reverse Operator (><) Behavior

In Vera, there is a subtle difference between the bit reverse task and operator:

- The `vera_bit_reverse` task reverses bits first and then sizes the result as needed.
- The `><` operator sizes the argument first and then reverses the bits.

The NTB `vera_bit_reverse` task and `><` operator both match the behavior of the `vera_bit_reverse` task.

Non-Blocking pre/post pack [Semantic Difference]

In NTB, the following tasks cannot block or call tasks/functions that block.

- `pre_pack()`
- `post_pack()`

Garbage Collection [Semantic Difference]

Both Vera and NTB perform automatic garbage collection on objects that they no longer use. Garbage collection happens at an indeterminate time during simulation and it is possible that coverage objects will continue to collect coverage until the garbage collection occurs. This occurs if the coverage trigger is outside the object where coverage is defined and coverage continues to be triggered after the object is deleted.

For this reason, the event that triggers coverage should only trigger coverage when the coverage object has a valid handle, or when the event is part of the object containing coverage declarations. Another solution is to disable coverage just before removing the last handle to the coverage object, as in the following example:

```
my_cov.inst_set_collect(OFF);
my_cov = null; // last reference to My_cov removed here
```

Vera Final Report

Unlike Vera, NTB does not print a final report on drives and expects. If you want to compare the output between Vera and NTB, use the Vera runtime switch option, `+vera_disable_final_report`, to turn this feature off.

Note: No support is planned for this feature.

Bad Coding Style Errors

Reading and modifying the same variable in different parts of the same statement can produce unexpected results. These differences only occur if the same variable is read and modified in two different parts of the same statement. You must avoid any dependency on such side effects. For example, the `j` variable in the following has several different values depending on which instance of it is being read:

```
printf("%d %d %d %d %d\n", j, ++j, j, j++, j);
```

Sign Extension

NTB follows the Verilog rules when performing arithmetic extension prior to applying arithmetic operators. The following table contains an example of the semantic differences:

Code	Vera	NTB
16’h0 === 32’h0	1	1
(~16’hffff) === 32’h0	1	0
16’hffff === 32’hffffffff	0	0
(~16’h0000) === 32’hffffffff	0	1

The solution is to ensure that the sizes of the arguments on both sides of the comparison are consistent.

5

VCS MX Limitations

VCS MX with NTB has the following limitations:

- Import and export tasks cause a delta cycle to execute in the simulation.
- If the design has a top-level VHDL, NTB cannot import Verilog tasks.
- In the VCS 7.2 and 2005.06 releases, there is an additional level of hierarchy around the testbench visible during debugging.

6

NTB to SystemVerilog NTB Recommendations

Data Types [Syntax Difference]

OpenVera does not support the following SystemVerilog data types. You should not use these data types in classes that will be used in both languages:

- Real
- Unpacked union
- Unpacked structure

Blocking Functions [Syntax Difference]

SystemVerilog does not allow functions to block, therefore, you will need to modify any code that you transition to SystemVerilog to contain blocking tasks instead of functions. This is a major problem if you transition the entire testbench to SystemVerilog, as you will need to replace all calls to the function with task calls.

String Routines [Syntax Difference]

The following routines are supported by NTB:

```
$psprintf
string::string.match()
string::replace
```

SystemVerilog does not support these string routines. VCS does supports these string routines, but they are not part of the current SystemVerilog standard.