VMM Checker Rules Manual

Version C-2009.06
June 2009

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

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of
>
> _____ and its employees. This is copy number_____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia, Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSIMplus, HSPICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

1

VMM Checker

This chapter provides reference information about the VMM Checker available with VCS. VMM Checker is a static linting tool that performs various checks to make sure that the testbench meets the VMM guidelines. It contains 35 rules that perform semantic analysis and check for coding style issues.

For more information about VMM, see the *Verification Methodology Manual*.

Note:

This manual is part of the VCS Design Checker documentation suite. For more information about Design Checker, see the *VCS/VCSi Design Checker User Guide*.

The VMM Checker rules are organized into the following categories:

- "VMM_ENV"

- “VMM_INT”
- “VMM_XN”
- “VMM_XT”

VMM_ENV

The rules listed under this category check the environment.

VMM_ENV001

Message: Top level module is not found for VMM rule checking

Description	This message is displayed when a top-level module is found missing in the code. VMM requires a testbench structure to contain a top level module.
Language	SystemVerilog
Severity	Lint

VMM_ENV002

Message: Multiple top level modules specified for VMM rule checking

Description	This message is displayed when more than one top level module is present in the code. VMM requires a testbench structure to contain only a single top level module. To limit performing checks on a single top level module, you can use the `-check_module` or `-nocheck_module` option.
Language	SystemVerilog
Severity	Lint

VMM_ENV003

Message: Program is not found for VMM rule checking

Description	Display this message when a program block is missing in the code. VMM requires a testbench structure to contain a program block that contains environment and tests.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input bit clk);
   logic data;
   logic [3:0] data2;
   logic data1;

   clocking ck @(posedge clk);
      input   data;
      output  data1, data2;
   endclocking : ck

   modport myport(clocking ck);

endinterface : dut_itf

module dut(dut_itf itf);
   always @itf.ck
      itf.data <= itf.data1 ? |itf.data2 : &itf.data2;
endmodule // dut

module top;
   bit clk = 0;
```

```
   dut_itf itf(clk);
   dut dut_inst(itf);

   always #5 clk = !clk;

endmodule // top
```

VMM_ENV004

Message: One or more program shouldn't instantiate an extension of vmm_env class

Description	Design Checker flags this rule when one or more than one program instantiate a class extended from `vmm_env` object. If one program has instantiation of more than one `vmm_env` object (either different or same) or more than one program have instantiation of one `vmm_env` object each (either different or same), this rule is flagged.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
class test_env_01 extends vmm_env ;
   vmm_log log = new ("log", "test_env_01");

   function new();
   endfunction
endclass
```

```
class test_env_01_derived extends test_env_01 ;
   vmm_log log = new ("log", "test_env_01_derived");

   function new();
   endfunction
endclass

class test_env_02 extends vmm_env ;
   vmm_log log = new ("log", "test_env_02");
   function new();
   endfunction
endclass

program test;

test_env_01   test01;
test_env_01_derived   test01_derived;
                     // VMM_ENV004 violation for this.

   initial begin
   end
endprogram

program test1;

test_env_02   test02; // VMM_ENV004 violation for this.
test_env_02   test02_another;
               // VMM_ENV004 violation for this.

   initial begin
   end

endprogram
```

VMM_INT

The rules listed under this category checks the SystemVerilog interfaces.

VMM_INT003

Message: All interface signals shall be declared as wire.

Description	This message is displayed when a design signal in a primary interface is not declared as wire. VMM requires that all design signals interfacing with primary interfaces be defined as wires.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input bit clk); // VMM_INT003 flags here
   wire data;
   wire [3:0] data2;
   wire data1;
   bit d1;          // VMM_INT003 flags here
   real r1;          // VMM_INT003 flags here

   clocking ck @(posedge clk);
      input   data;
      output    data1, data2;
   endclocking : ck

   modport myport(clocking ck);

endinterface : dut_itf
```

VMM_INT004

Message: Setup time of a synchronous signal in the design interface is not a parameter

Description	This message is displayed when a synchronous design signal in the primary interface has an input skew specified as a non-parameter constant. VMM requires that input setup time in a clocking block of a primary design interface must be specified as a parameter. Synchronous signals and their skews are specified in clocking blocks.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input clk);
   wire data;
   wire [3:0] data2;
   wire data1;
   wire d1;
   wire r1;
   parameter setup = 5;

   clocking ck @(posedge clk);
      input #setup data;
      input #2ps      d1;    // VMM_INT004 flags here
      input #3ps      r1;    // VMM_INT004 flags here
      output          data1, data2;
   endclocking : ck

   clocking ck1 @(posedge clk);
```

```
      default input #5;       // VMM_INT004 flags here
      input      data;
      input      d1;
      input      r1;
      output     data1, data2;
   endclocking : ck1

   clocking ck2 @(posedge clk);
      default input #setup;
      input      data;
      input      d1;
      input      r1;
      output     data1, data2;
   endclocking : ck2
   modport myport(clocking ck);

endinterface : dut_itf
```

VMM_INT005

Message: Hold time of a synchronous signal in the design interface is not a parameter

Description	This message is displayed when a synchronous design signal in the primary interface has an output skew specified as a non-parameter constant. VMM requires that output hold time in a clocking block of a primary design interface must be specified as a parameter. Synchronous signals and their skews are specified in clocking blocks.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input clk);
   wire data;
   wire [3:0] data2;
   wire data1;
   wire d1;
   wire r1;
   parameter hold = 5;

   clocking ck @(posedge clk);
      input           data;
      input #2ps      d1;
      input #3ps      r1;
      output #1       data1;   // VMM_INT005 flags here
      output #hold    data2;
   endclocking : ck
```

```
   clocking ck1 @(posedge clk);
      default output #5;     // VMM_INT005 flags here
      input       data;
      input       d1;
      input       r1;
      output      data1, data2;
   endclocking : ck1

   clocking ck2 @(posedge clk);
      default output #hold;
      input       data;
      input       d1;
      input       r1;
      output      data1, data2;
   endclocking : ck2

   modport myport(clocking ck);

endinterface : dut_itf
```

VMM_INT006

Message: Modport for a transactor is missing in the design interface

Description	This message is displayed when the definition of a `modport` contained in a command layer transactor is missing in primary interfaces. VMM requires that all modport properties specified in command layer transactors be `modports` of primary interfaces. Primary interfaces interface with design signals from the DUT.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input bit clk);
   logic data;
   logic [3:0] data2;
   logic   data1;

   clocking ck @(posedge clk);
      input    data;
      output   data1, data2;
   endclocking : ck

   modport myport(clocking ck);
endinterface : dut_itf

interface itf1(input bit clk);
   logic data;
   logic [3:0]   data2;
```

```
   logic         data1;

   clocking ck @(posedge clk);
      input      data;
      output     data1, data2;
   endclocking : ck

   modport myport1(clocking ck);

endinterface : itf1

class CmdXactor1 extends vmm_xactor;
   virtual itf1.myport1 myport; // VMM_INT006 flags here.
// This modport is part of dut_itf which is not instantiated
// in the top level module

   Channel cmd_chan;
   // static vmm_log log;

   function new (string instance,
         integer stream_id = -1,
         virtual dut_itf.myport myport,
         Channel channel = null);

      // Call the super task to initialize the xactor
      super.new("CMD", instance, stream_id) ;

      // Save a reference to the interface
   this.cmd_chan = channel;

      // Construct an input channel if needed, save a
      // reference to the channel
      if (channel == null)
         channel = new("CMD channel", instance);
      this.cmd_chan = channel;

   endfunction: new

   task main();
      super.main();
   endtask : main
```

```
   function void start_xactor();
      super.start_xactor();
   endfunction : start_xactor

   function void reset_xactor(reset_e rst_type = SOFT_RST);
      super.reset_xactor(rst_type);
      cmd_chan.flush();
   endfunction : reset_xactor

   function void stop_xactor();
      super.stop_xactor();
   endfunction : stop_xactor

endclass : CmdXactor1

module dut(dut_itf itf);
   wire clk;

   itf1 itf1(clk);
   always @itf.ck
      itf.data <= itf.data1 ? |itf.data2 : &itf.data2;
endmodule // dut

module top;
   bit clk = 0;

   dut_itf itf(clk);
   dut dut_inst(itf);

   always #5 clk = !clk;

endmodule // top
```

VMM_INT007

Message: Signals in a clocking block are listed individually in a modport instead of specifying the clocking block

Description	This message is displayed when a synchronous signal specified in a clocking block is appearing as an individual signal in a `modport`. VMM requires that all synchronous signals be listed in `modports` in groups using the `clocking` construct, instead of individual signals.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input clk);
   wire data;
   wire [3:0] data2;
   wire data1;
   wire d1;
   wire r1;
   wire r2;

   parameter hold = 5;

   clocking ck @(posedge clk);
      input          data;
      input #2ps     d1;
      input #3ps     r1;
      output #1      data1;
      output #hold   data2;
```

```
   endclocking : ck

   clocking ck1 @(posedge clk);
      default output #5;
      input    data;
      input    d1;
      input    r1;
      output   data1, data2;
   endclocking : ck1

   clocking ck2 @(posedge clk);
      default output #hold;
      input    data;
      input    d1;
      input    r1;
      output   data1, data2;
   endclocking : ck2

   modport myport(clocking ck);

   modport myport1(input d1, r1, output data1, data2);
   // VMM_INT007 flags here for the above 4 signals

   modport myport2(output data, input data1);
   // VMM_INT007 flags here for the above 2 signals

endinterface : dut_itf
```

VMM_INT008

Message: Missing asynchronous signals from modports of the design interface

Description	This message is displayed when an asynchronous signal specified in a design interface is not listed in any modport of the design interface. VMM requires that all asynchronous signals specified in a design interface are listed in modports of the interface.
Language	SystemVerilog
Severity	Lint

Example

The following example shows Verilog code that flags this rule:

```
interface dut_itf(input clk); // VMM_INT008 flags here for
                              // asynchronous signal clk
wire data;
   wire [3:0] data2;
   wire data1;
   wire d1;
   wire r1;
   wire r2; // VMM_INT008 flags here for asynchronous
            // signal r2

   parameter hold = 5;

   clocking ck @(posedge clk);
      input          data;
      input #2ps     d1;
      input #3ps     r1;
      output #1      data1;
      output #hold   data2;
```

```
   endclocking : ck

   clocking ck1 @(posedge clk);
      default output #5;
      input    data;
      input    d1;
      input    r1;
      output   data1, data2;
   endclocking : ck1

   clocking ck2 @(posedge clk);
      default output #hold;
      input    data;
      input    d1;
      input    r1;
      output   data1, data2;
   endclocking : ck2

   modport myport(clocking ck);
   modport myport1(input d1, r1, output data1, data2);
   modport myport2(output data, input data1);

endinterface : dut_itf
```

VMM_XN

The rules listed under this category are applicable to classes derived from the base class `vmm_data`.

VMM_XN001

Message: No transaction class derived from vmm_data is found

Description	This message is displayed when none of the transaction class is derived from `vmm_data`. The SV testbench code should contain at least one transaction class derived from `vmm_data` for exchanging data between a command-layer transactor and the upper layers.
Language	SystemVerilog
Severity	Lint

VMM_XN003

Message: Classes derived from vmm_data do not contain a public static vmm_log object

Description	This message is displayed when classes derived from `vmm_data` do not contain a public static `vmm_log` object. Every transaction should output messages about its status using the shared messaging service present in the `vmm_log` class.
Language	SystemVerilog
Severity	Lint

VMM_XN004

Message: Classes derived from vmm_data do not call in the new() function its base class constructor and pass the local vmm_log object

Description	The `new()` function should be redefined and call `super.new` with the local `vmm_log` object instance as an argument.
Language	SystemVerilog
Severity	Lint

VMM_XN005

Message: There is no rand property in the transaction class

Description	This message is displayed if there is no `rand` property in the transaction class. A transaction content should be randomizable. It must contain `rand` variables with the transaction values.
Language	SystemVerilog
Severity	Lint

VMM_XN006

Message: A non-public rand property is found in a transaction

class

Description	This message is displayed when a non-public `rand` property is found in a transaction. A transaction `rand` variable must be accessible.
Language	SystemVerilog
Severity	Lint

VMM_XN007

Message: No constraint block of a transaction class is named <transaction_class_name>_valid

Description	This message is displayed when none of the constraint block of a transaction class is named `<transaction-class-name>_valid`. A constraint with the `_valid` suffix should be defined to delimit the valid range of values of the transaction data (`rand` variables).
Language	SystemVerilog
Severity	Lint

VMM_XN008

Message: Constructor without arguments is not provided for a

transaction class (arguments should have default values)

Description	All arguments to the transaction constructor should have meaningful default values, which allow the constructor to be called without actual arguments, but still create a valid default transaction instance.
Language	SystemVerilog
Severity	Lint

VMM_XN009

Message: Transaction has a non-function method

Description	This message is displayed when a non-function method is found in a transaction. The transaction methods must be non-blocking; they should be implemented using functions.
Language	SystemVerilog
Severity	Lint

VMM_XN010

Message: Function psdisplay is not defined for a transaction

Description	This message is displayed when the function `psdisplay` is not defined for a transaction. You must format the transaction state information using an overloaded `psdisplay` function that is customized for a particular transaction type.
Language	SystemVerilog
Severity	Lint

VMM_XN011

Message: Function is_valid is not defined for a transaction

Description	This message is displayed when the function `is_valid` is not defined for a transaction. You must check the validity of a transaction using an overloaded `is_valid` function that is customized for a particular transaction type.
Language	SystemVerilog
Severity	Lint

VMM_XN012

Message: Function allocate is not defined for a transaction

Description	This message is displayed when the function allocate is not defined for a transaction. You must allocate a new transaction using an overloaded `allocate` function that is customized for a particular transaction type.
Language	SystemVerilog
Severity	Lint

VMM_XN013

Message: Function copy is not defined for a transaction

Description	This message is displayed when the function copy is not defined for a transaction. You must copy a transaction using an overloaded `copy` function that is customized for a particular transaction type.
Language	SystemVerilog
Severity	Lint

VMM_XN014

Message: Function compare is not defined for a transaction

Description	This message is displayed when the function compare is not defined for a transaction. You must compare two transactions using an overloaded `compare` function that is customized for a particular transaction type.
Language	SystemVerilog
Severity	Lint

VMM_XT

The rules listed under this category are applicable to classes derived from the base class `vmm_xactor`.

VMM_XT001

Message: No transaction class derived from vmm_xactor is found

Description	This message is displayed when none of the transaction class is derived from `vmm_xactor`. This message distinguishes transactors that are at the command layer and the functional layer. A transactor that is derived from `vmm_xactor` and containing an instance of a virtual modport is classified as a command-layer transactor.
Language	SystemVerilog
Severity	Lint

VMM_XT002

Message: Transactor contains a vmm_log instance

Description	This message is displayed when a transactor containing a `vmm_log` instance is found. A transactor class derived from `vmm_xactor` should not have any reference to a `vmm_log` instance.
Language	SystemVerilog
Severity	Lint

VMM_XT005

Message: rand property found in a transactor

Description	This message is displayed when the `rand` property is found in a transactor. The transactors should be configured using a randomizable configuration descriptor class.
Language	SystemVerilog
Severity	Lint

VMM_XT007

Message: Virtual modport property of a transactor is not public

Description	This message is displayed when the virtual `modport` property of a transactor is not public. This way the test cases can also access physical interface signals by accessing the modport variable in the transactor.
Language	SystemVerilog
Severity	Lint

VMM_XT008

Message: Task main is not defined for a transactor

Description	This message is displayed when the task `main` is not defined for a transactor. All threads of the autonomous behavior of a transactor should be forked off in this task.
Language	SystemVerilog
Severity	Lint

VMM_XT009

Message: Super.main is not the first statement of task main in a transactor

Description	This message is displayed when `super.main` is not the first statement in task `main`. The task `main` must call `super.main()` to assure proper execution.
Language	SystemVerilog
Severity	Lint

VMM_XT010

Message: Function start_xactor is not defined for a transactor

Description	This message is displayed when the function `start_xactor` is not defined for a transactor. This function is needed to start the operation of a transactor.
Language	SystemVerilog
Severity	Lint

VMM_XT011

Message: Super.start_xactor is not the first statement of function start_xactor in a transactor

Description	This message is displayed when `super.start_xactor` is not the first statement of function `start_xactor` in a transactor. The function `start_xactor` must call `super.start_xactor()` to assure proper execution.
Language	SystemVerilog
Severity	Lint

VMM_XT012

Message: Function stop_xactor is not defined for a transactor

Description	This message is displayed when the function `stop_xactor` is not defined for a transactor. This function is needed to stop the operation of a transactor.
Language	SystemVerilog
Severity	Lint

VMM_XT013

Message: Super.stop_xactor is not the first statement of function stop_xactor in a transactor

Description	This message is displayed when `super.stop_xactor` is not the first statement of function `stop_xactor` in a transactor. The function `stop_xactor` must call `super.stop_xactor()` to assure proper execution.
Language	SystemVerilog
Severity	Lint

VMM_XT014

Message: Function reset_xactor is not defined for a transactor

Description	This message is displayed when the function `reset_xactor` is not defined for a transactor. This function is needed to stop the operation of a transactor.
Language	SystemVerilog
Severity	Lint

VMM_XT015

Message: Super.reset_xactor is not the first statement of function reset_xactor in a transactor

Description	This message is displayed when `super.reset_xactor` is not the first statement of function `reset_xactor` in a transactor. The function `reset_xactor` must call `super.reset_xactor()` to assure proper execution.
Language	SystemVerilog
Severity	Lint