SYNOPSYS, INC

VMM Standard Library Reference Guide

VMM 1.2

Copyright Notice and Proprietary Information

Right to Copy Documentation

Destination Control Statement

Disclaimer

Registered Trademarks (®)

Trademarks (™)

Service Marks (SM)

VMM 1.2 Standard Library Update

This document is currently a specification for the updates made to the VMM Standard Library with respect to the VMM1.2 revision

The common base class (vmm_object)

vmm_object is a virtual class that is used as the common base class for all VMM related classes. This helps to provide parent/child relationships for class instances. Additionally, it provides local, relative and absolute hierarchical naming.

vmm_object::get_typename()

Return the name of the actual type of this object.

SystemVerilog

```
pure virtual function string vmm_object::get_typename();
```

Description

This function is implemented in ` vmm_typename(string name) macro. It returns the type of this vmm_object extension.

Example

```
class ahb_gen extends vmm_unit;
   `vmm_typename (ahb_gen)
   function new (string name);
      super.new (get_typename(), name);
   endfunction
endclass
```

vmm_object::new();

Construct a new instance of this object,

SystemVerilog

```
function void vmm_object::new(vmm_object parent = null,
                              string name = "[Anonymous]");
```

Description

Construct a new instance of this object, optionally specifying another object as its parent. The specified named cannot contain the ':' character.

Example

```
class A extends vmm_object;
   function new (string name, vmm_object
                 parent=null);
      super.new (parent, name);
   endfunction
endclass
```

vmm_object::create_namespace();

Define a namespace with specified default object inclusion policy.

SystemVerilog

```
static function bit vmm_object::create_namespace(string name,
                                 namespace_type_e = OUT_BY_DEFAULT);
```

Description

Define a namespace with the specified default object inclusion policy. A namespace must be previously created using this method before it can be used or referenced. Returns TRUE of the namespace was successfully created. The empty name space (“”) is reserved and cannot be defined.

Example

```
class A extends vmm_object;
   function new (string name, vmm_object
                 parent=null);
      super.new (parent, name);
      vmm_object::create_namesapce(“NS1”,
                                   vmm_object::IN_BY_DEFAULT);
   endfunction
endclass
```

vmm_object::get_namespaces();

Define a namespace with specified default object inclusion policy.

SystemVerilog

```
static function bit vmm_object::get_namespaces(output string
                                                name[]);
```

Description

Define a namespace with the specified default object inclusion policy.
A namespace must be previously created using this method before it can be used or referenced. Returns TRUE of the namespace was successfully created. The empty name space (“”) is reserved and cannot be defined.

Example

```
initial begin
   string ns_array[];
   . . .
   vmm_object::get_namespaces(ns_array);
   . . .
end
```

vmm_object::set_object_name();

Explicitly set or replace the name of this object in the specified namespace.

SystemVerilog

```
function void vmm_object::set_object_name(string name, string space
= “”);
```

Description

This method is used to set or replace the name of this object in the specified namespace. If no namespace is specified, the name of the object is replaced. If a name has not been specified for a namespace, it defaults to the object name. Names in a named namespaces may contain the ‘:’ character to create additional levels of hierarchy or be empty to skip a level of hierarchy. A name starting with “^” indicates that it is a root in the specified namespace (not applicable to the object name where parent-less objects create roots in the default namespace).

Example

```
class E extends vmm_object;
…
endclass
initial begin
   vmm_object obj;
   E e1 = new (“e1”);
   vmm_object::create_namespace("NS1", vmm_object::IN_BY_DEFAULT);
   …
   obj = e1;
   obj.set_object_name (“new_e1”,”NS1”);
   …
end
```

vmm_object::set_parent_object()

Explicitly set or replace the parent of this object.

SystemVerilog

```
function void vmm_object::set_parent_object(vmm_object parent);
```

Description

Specify a new parent object to this object. Specifying a NULL parent breaksany current parent/child relationship. An object may have only one parent, but the identity of a parent can be changed dynamically.

Example

```
class C extends vmm_object;
   function new(string name, vmm_object
             parent=null);
      super.new (parent,name);
   endfunction
endclass
class D extends vmm_object;
   C c1;
   function new(string name, vmm_object
             parent=null);
     super.new (parent,name);
     c1 = new (”c1”,this);
   endfunction
endclass

initial begin
   D d1 = new (“d1”);
   D d2 = new (“d2”);
   d1.c1.set_parent_object (d2);
end
```

vmm_object::kill_object()

SystemVerilog

```
Virtual function void vmm_object::kill_object();
```

Description

Clear cross-references to this object and all of its children, so the entire object hierarchy rooted at the object can be garbage collected. Killing the root object will enable the garbage collection of the entire object hierarchy underneath it—unless there are other references to an object within that hierarchy. Any external reference to any object in a hierarchy will prevent the garbage collection of that object.

Example

```
class C extends vmm_object;
   function new(string name,
                vmm_object parent=null);
      super.new (parent,name);
   endfunction
endclass
class D extends vmm_object;
   C c1;
   function new(string name,
                vmm_object parent=null);
      super.new (parent,name);
      c1 = new (”c1”,this);
   endfunction
endclass
initial begin
   D d1 = new (“d1”);
   d1.kill_object;
end
```

vmm_object::get_parent_object()

Return the parent of this object.

SystemVerilog

```
function vmm_object vmm_object::get_parent_object();
```

Description

Return the parent object of the specified type, if any. Returns NULL if noparent is found. A root object has no parent.

Example

```
class C extends vmm_object;
   …
endclass
class D extends vmm_object;
   C c1;
   function new(string name, vmm_object
                parent=null);
      c1 = new (”c1”,this);
   endfunction
endclass

initial begin
   vmm_object parent;
   D d1 = new (“d1”);
   parent = d1.c1.get_parent_object;
end
```

vmm_object::is_parent_of()

Return true if the specified object is a parent of this object.

SystemVerilog

```
function bit vmm_object::is_parent_of(vmm_object obj, string space
=””);
```

Description

Return true if the specified object is a parent of this object.

Example

```
class sub extends vmm_subenv;
   . . .
endclass

class tb_env extends vmm_env;
   sub s1 ;
   . . .
   virtual function void build();
       super.build();
       s1 = new (“s1”);
       s1.set_parent(this);
       if (!this.is_parent_of(s1))
          `vmm_error(log, “Unable to set parent for s1”);
       . . .
    endfunction
endclass
```

vmm_object::get_root_object()

Get the root parent of this object.

SystemVerilog

```
function vmm_object vmm_object::get_root_object(string space = "");
```

Description

Get the root parent of this object for the specified namespace.

Example

```
class C extends vmm_object;
   ...
endclass
class D extends vmm_object;
   C c1;
   function new(string name, vmm_object
             parent=null);
      c1 = new ("c1",this);
   endfunction
endclass
class E extends vmm_object;
   D d1;
   function new(string name, vmm_object
             parent=null);
      ...
      d1 = new ("d1",this);
   endfunction
endclass
...
initial begin
   vmm_object root;
   E e1 = new ("e1");
   root = e1.d1.c1.get_root_object;
   ...
end
```

vmm_object::get_num_children()

Get the total number of children for this object.

SystemVerilog

```
function int vmm_object::get_num_children();
```

Description

Get the total number of children object for this object.

Example

```
class C extends vmm_object;
   ...
endclass
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   C c1;
   D d1;
   D d2;
   function new(string name, vmm_object parent=null);
      ...
      c1 = new (“c1”,this);
      d1 = new (”d1”);
      d2 = new (”d2”,this);
   endfunction
endclass
...
int num_children;
                                                    initial
begin
   E e1 = new (“e1”);
   ...
   num_children = e1.get_num_children;
   ...
end
```

vmm_object::get_nth_child()

Return the nth child of this object.

SystemVerilog

```
function vmm_object vmm_object::get_nth_child(int n);
```

Description

Return the nth child of this object. Returns null if there is no child.

Example

```
class C extends vmm_object;
   ...
endclass
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   C c1;
   D d1;
   D d2;
   function new(string name, vmm_object
                parent=null);
      c1 = new (”c1”,this);
      d1 = new (”d1”);
      d2 = new (”d2”,this);
   endfunction
endclass
initial begin
   vmm_object obj;
   string name;
   E e1 = new (“e1”);
   obj = e1.get_nth_child(0);
   name = obj.get_object_name();  //Returns c1
   ...
end
```

vmm_object::get_num_roots()

Get the total number of root objects in the specified namespace.

SystemVerilog

```
static function int vmm_object::get_num_roots(string space = "");
```

Description

Get the total number of root objects in the specified namespace.

Example

```
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   D d1;
   D d2;
   function new(string name, vmm_object
                  parent=null);
      ...
      d1 = new ("d1");
      d2 = new ("d2");
   endfunction
endclass
...
int num_roots;
initial begin
   E e1 = new ("e1");
   ...
   num_roots = E :: get_num_roots(); //Returns 2
   ...
end
```

vmm_object::get_nth_root();

Return the nth root object in the specified namespace.

SystemVerilog

```
static function vmm_object vmm_object::get_nth_root(int n, string
space = "");
```

Description

Return the nth root object in the specified namespace. Returns null if there is no such root.

Example

```
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   D d1;
   D d2;
   function new(string name, vmm_object
                  parent=null);
      ...
      d1 = new ("d1");
      d2 = new ("d2");
   endfunction
endclass
...
int num_roots;
initial begin
   vmm_object root;
   E e1 = new ("e1");
   ...
   root= E :: get_nth_root(0); //Returns d1
   ...
end
```

vmm_object::get_object_name()

Get the local name of this object

SystemVerilog

```
function string vmm_object::get_object_name(string space = "");
```

Description

Get the local name of this object in the specified namespace. If no namespace is specified, return the actual name of the object.

Example

```
class C extends vmm_object;
   function new(string name, vmm_object
                parent=null);
      super.new (parent,name);
   endfunction
endclass
...
initial begin
   string obj_name;
   C c1 = new ("c1");
   ...
   obj_name = c1.get_object_name(); //Returns c1
   ...
end
```

vmm_object::get_object_hiername();

Get the full hierarchical name of this object.

SystemVerilog

```
function string vmm_object::get_object_hiername(vmm_object root =
null, string space = "");
```

Description

Get the full hierarchical name of this object in the specified namespace, relative to the specified root object. If no root object is specified, returns the full hierarchical name of the object. The instance name is composed of the dot-separated instance names of the message service interface of all the parents of the object.

Example

```
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   D d1;
   function new(string name, vmm_object
                  parent=null);
      ...
      d1 = new ("d1",this);
   endfunction
endclass
...
initial begin
   string hier_name;
   E e1 = new ("e1");
   ...
   hier_name = e1.d1.get_object_hiername();
   ...
end
```

vmm_object::find_child_by_name();

Find the named object relative to this object.

SystemVerilog

```
function vmm_object vmm_object::find_child_by_name(string name,
string space = "");
```

Description

Find the named object, interpreting the name as a hierarchical name relative to this object in the specified namespace. If the name is a match pattern or regular expression, the first object matching the name is returned. Returns null if no child was found under the specified name.

Example

```
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   D d1;
   function new(string name, vmm_object
                  parent=null);
      ...
      d1 = new ("d1",this);
   endfunction
endclass
...
initial begin
   vmm_obj obj;
   E e1= new ("e1");
   ...
   obj = e1.find_child_by_name ("d1");
   ...
end
```

vmm_object::find_object_by_name();

Find the named object in the specified namespace.

SystemVerilog

```
static function vmm_object vmm_object::find_object_by_name(string
name, string space = "");
```

Description

Find the named object, interpreting the name as an absolute name in the specified namespace. If the name is a match pattern or regular expression, the first object matching the name is returned.

Returns null if no object was found under the specified name.

Example

```
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   D d1;
   function new(string name, vmm_object
                  parent=null);
      ...
      d1 = new ("d1");
   endfunction
endclass
...
initial begin
   vmm_object obj;
   ...
   obj = E :: find_object_by_name ("d1");
   ...
end
```

vmm_object::psdisplay()

Create a description of the object.

SystemVerilog

```
virtual function string vmm_object::psdisplay(string prefix = "");
```

Description

Create a human-readable description of the content of this object. Each line of the image is prefixed with the specified prefix.

Example

```
class D extends vmm_object;
   ...
   function string psdisplay(string prefix = "");
      ...
   endfuntion
endclass
...
   vmm_log log = new ("Test", "main");
initial begin
   D d1 = new ("d1");
   ...
   `vmm_note (log, d1.psdisplay);
   ...
end
```

vmm_object::display_hierarchy()

Print the object hierarchy

SystemVerilog

```
Static function void vmm_object::display_hierarchy(vmm_object
root = null);
```

Description

Print the object hierarchy rooted at the specified object. Print the hierarchy for all roots if no root is specified.

Example

```
class D extends vmm_object;
   ...
endclass
class E extends vmm_object;
   D d1;
   function new(string name, vmm_object
                  parent=null);
      ...
      d1 = new (”d1”,this);
   endfunction
endclass
...
initial begin
   E e1 = new (“e1”);
   ...
   E :: display_hierarchy();
   ...
end
```

vmm_object::get_log()

Return the vmm_log instance of this object.

SystemVerilog

```
virtual function vmm_log vmm_object::get_log();
```

Description

Return the vmm_log instance of this object or the nearest enclosing object. If no vmm_log instance is available in the object genealogy, a default global vmm_log instance is returned.

Example

```
class ABC extends vmm_object;
   vmm_log log = new("ABC", "class");
   ...
function vmm_log get_log();
    return this.log;
  endfunction
...
endclass

vmm_log test_log;
ABC abc_inst = new("test_abc");
initial begin
test_log = abc_inst.get_log();
...
end
```

vmm_object::implicit_phasing()

If the “is_on” argument is FALSE, inhibit the implicit phasing for this object and all of its children objects.

SystemVerilog

```
virtual function void vmm_object::implicit_phasing(bit is_on);
```

Description

If the “is_on” argument is FALSE, inhibit the implicit phasing for this object and all of its children objects. Used to prevent a large object hierarchy that does not require phasing from being needless walked by the implicit phaser (e.g. RAL model). By default, implicit phasing is enabled.

Example

```
class subsys_env extends vmm_subenv;
   . . .
endclass

class sys_env extends vmm_env;
   subsys_env subenv1;
   . . .
   function void build();
      . . .
      subenv1 = new (“subenv1”, “subenv1”);
      subenv1.set_parent_object(this);
      subenv1.implicitly_phasing(0);
      . . .
   endfunction
   . . .
endclass
```

vmm_object::is_implicitly_phased()

Return TRUE if the implicit phasing is enabled for this object.

SystemVerilog

```
virtual function bit vmm_object::is_implicitly_phased();
```

Description

Return TRUE if the implicit phasing is enabled for this object.

Example

```
class subsys_env extends vmm_subenv;
   . . .
endclass

class sys_env extends vmm_env;
   subsys_env subenv1;
   . . .
   function void build();
      . . .
      subenv1 = new ("subenv1", "subenv1");
      subenv1.set_parent_object(this);
      subenv1.implicitly_phasing(0);
      if(subenv1.is_implicitly_phased)
         `vmm_error(log, "Implict Phasing for subenv1 not
                          disabled");
      . . .
   endfunction
   . . .
endclass
```

class vmm_object_iter

This is the vmm_object hierarchy traversal iterator class.

Example

```
class E extends vmm_object;
   ...
endclass
...
initial begin
   E e11 = new (“e1”);
   vmm_object obj;
   vmm_object_iter  iter = new(“/a1/”, e11 );
   ...
   obj = iter.first();
   while (obj != null)
   begin
      ...
      obj = iter.next;
   end
   ...
 end
```

vmm_object_iter::new();

Instantiates an vmm_object iterator which traverses the hierarchy rooted at the specified root object.

SystemVerilog

```
function vmm_object_iter::new(string name = "/.?/", vmm_object root
= null, string space = "");
```

Description

Traverse the hierarchy rooted at the specified root object, looking for objects whose relative hierarchical name in the specified namespace matches the specified name. The object name is relative to the specified root object. If no object is specified, traverses all of the hierarchies and the hierarchical name is absolute. The specified root (if any) is not included in the iteration.

Example

```
/ Match pattern - /a1/, with root object e11
vmm_object_iter  iter = new ("/a1/", e11 );
```

vmm_object_iter::first()

Reset the state of the iterator to the first object

SystemVerilog

```
function vmm_object vmm_object_iter::first();
```

Description

Reset the state of the iterator to the first object in the vmm_object hierarchy. Returns null if the specified hierarchy has no child objects.

Example

```
                                                                    class
E extends vmm_object;
   ...
endclass
...
initial begin
   E e11 = new (“e1”);
   vmm_object obj;
   vmm_object_iter  iter = new(“/a1/”, e11 );
   ...
   obj = iter.first();
   ...
end
```

vmm_object_iter::next()

SystemVerilog

```
function vmm_object vmm_object_iter::next();
```

Description

Return the next object in the vmm_object hierarchy. Returns null if there are no more child objects. Objects are traversed depth-first.

Example

```
class E extends vmm_object;
   ...
endclass
...
initial begin
   E e11 = new (“e1”);
   vmm_object obj;
   vmm_object_iter  iter = new(“/a1/”, e11 );
   ...
   obj = iter.first();
   while (obj != null)
   begin
      ...
      obj = iter.next;
   end
   ...
 end
```

`foreach_vmm_object()

Shorthand macro to iterate over all objects

SystemVerilog

```
`foreach_vmm_object(classtype, string name, vmm_root root);
```

Description

This is a shorthand macro to iterate over all obejects of a specified type and name under a specified root.

Example

```
class E extends vmm_object;
   ...
endclass
...
initial begin
   E e11 = new (“e11”);
   vmm_object_iter my_iter;
   ...
   `foreach_vmm_object(vmm_object, "@%*", e11)
   begin
      ...
   end
end
```

`foreach_vmm_object_in_namespace()

Shorthand macro to iterate over all objects of a specified type.

SystemVerilog

```
`foreach_vmm_object_in_namespace(classtype, string name, string
space, vmm_root root);
```

Description

Short-hand macro to iterate over all objects of a specified type with the specified name in the specified namespace under a specified root.

Example

```
class C extends vmm_object;
   function new(string name, vmm_object parent=null);
      super.new(parent, name);
      . . .
      vmm_object::create_namespace("NS1",vmm_object::IN_BY_DEFAULT);
      . . .
   endfunction
endclass

C c1 = new(“c1”);
int I;

initial begin
   `foreach_vmm_object_in_namespace(vmm_object, "@%*", "NS1", c1)
begin
       . . .
    end
end
```

Structural component class (vmm_unit)

Class to create structural elements.

SystemVerilog

```
virtual class vmm_unit extends vmm_object;
```

Description

This class is used as the base class for structural elements, such as transactors, transaction-level models and generators. The purpose of this class is to:

- Support structural composition and connectivity.
- Integrate into a simulation timeline.

This can be a leaf or a non-leaf component.

Here are the supported phases, sorted by calling order:

- ***build_ph()***
- ***configure_ph()***
- ***connect_ph()***
- ***start_of_sim_ph()***
- ***reset_ph()***
- ***training_ph()***
- ***config_dut_ph()***
- ***start_ph()***
- ***start_of_test_ph()***
- ***run_ph()***
- ***shutdown_ph()***
- ***cleanup_ph()***
- ***report_ph()***
- ***destruct_ph()***

Example

```
class vip1 extends vmm_unit;
endclass
```

vmm_unit::new()

Constructor for vmm_unit.

SystemVerilog

```
function vmm_unit::new(string name, string inst, vmm_unit parent =
null);
```

Description

Construct an instance of this class with the specified name, instance name and optional parent.

The specified name will be used as the name of the embedded vmm_log. The specified instance name will be used as the name of the underlying vmm_object.

Example

```
class vip1 extends vmm_unit;
   function new (string name, string inst);
      super.new (this,inst);
   endfunction
endclass
```

vmm_unit::consent()

Express this *vmm_unit's* consent to the consensus for the specified reason.

SystemVerilog

```
function void vmm_unit::consent(string why = "No reason specified");
```

Description

Express this *vmm_unit's* consent to the consensus for the specified reason.

Example

```
class  unitExtension extends vmm_unit;
   . . .
   task reset_ph();
      this.oppose("reset phase running");
      fork
      begin
        #50;
        this.consent("reset phase finished");
      end
      join_none
   endtask:reset_ph
   . . .
endclass
```

vmm_unit::oppose()

Express this *vmm_unit's* opposition to the consensus for the specified reason.

SystemVerilog

```
function void vmm_unit::oppose(string why = "No reason specified");
```

Description

Express this *vmm_unit's* opposition to the consensus for the specified reason.

Example

```
class  unitExtension extends vmm_unit;
   . . .
   task reset_ph();
      this.oppose("reset phase running");
      fork
      begin
        #50;
        this.consent("reset phase finished");
      end
      join_none
   endtask:reset_ph
   . . .
endclass
```

vmm_unit::is_unit_enabled()

Returns ‘1’ if unit is enabled.

SystemVerilog

```
function bit vmm_unit::is_unit_enabled();
```

Description

Check if this vmm_unit instance has been disabled or not. By default, all units are enabled. A unit may be disabled by calling its disable_unit() method before the “start_of_sim” phase.

Example

```
class unitExtension extends vmm_unit;
   ...
endclass

class udf_start_def extends vmm_fork_task_phase_def
#(unitExtension);
   ...
   task do_task_phase(unitExtension obj);
      if(obj.is_unit_enabled())
      obj.udf_start_ph();
   endtask:do_task_phase
   ...
endclass
```

vmm_unit::disable_unit()

Method to disable a unit instance

SystemVerilog

```
function void vmm_unit::disable_unit();
```

Description

Disable this instance of the vmm_unit class. This method must be called before the “start_of_sim” phase. A vmm_unit instance can only be re-enabled by resetting its timeline to the “configure” phase or ealier.

Example

```
class unitExtension extends vmm_unit;
   ...
Endclass

unitExtension m1 = new (“unitExtension”,”m1”);
m1.disable_unit();
```

vmm_unit::get_timeline()

Method returns the enclosing timeline

SystemVerilog

```
function vmm_timeline vmm_unit::get_timeline();
```

Description

Returns the run-time timeline this unit is executing under.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph();
      vmm_timeline t = this.get_timeline();
      ...
   endfunction
endclass
```

vmm_unit::build_ph()

Method to build this component.

SystemVerilog

```
virtual function void vmm_unit::build_ph();
```

Description

Build this component. Leaf level or independent root components associated can be created here.

Example

```
class memsys_env extends vmm_unit;
   cpu_subenv extends cpu0;
   vmm_ms_scenario_gen gen;
   memsys_scenario memsys_scn;
   ...
   function void build_ph();
      cpu0 = new("subenv", "CPU0", this);
      cpu1 = new("subenv", "CPU1", this);
      memsys_scn = new();
      gen = new("MS-Generator");
      ...
   endfunction
endclass
```

vmm_unit::configure_ph()

Method for functional configuration.

SystemVerilog

```
virtual function void vmm_unit::configure_ph();
```

Description

Functional configuration of this component.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void unitExtension::configure_ph();
      `vmm_note
    (log,`vmm_sformatf("unitExtension::configure_ph"));
      ...
   endfunction:configure_ph
endclass
```

vmm_unit::connect_ph()

Method for connecting components.

SystemVerilog

```
virtual function void vmm_unit::connect_ph();
```

Description

Connect the interfaces wholly contained within this component.

Example

```
class memsys_env extends vmm_unit;
   cpu_subenv extends cpu0;
   vmm_ms_scenario_gen gen;
   memsys_scenario memsys_scn;
   ...
   function void build_ph();
      cpu0 = new("subenv", "CPU0", this);
      cpu1 = new("subenv", "CPU1", this);
      memsys_scn = new();
      gen = new("MS-Generator");
      ...
   endfunction

   function void memsys_env::connect_ph();
      gen.register_channel("cpu0_chan",
                           cpu0.gen_to_drv_chan);
      gen.register_channel("cpu1_chan",
                          cpu1.gen_to_drv_chan);
      gen.register_ms_scenario( "memsys_scn", memsys_scn);
      ...
   endfunction
endclass
```

vmm_unit::start_of_sim_ph()

Method executes at start of simulation.

SystemVerilog

```
virtual function void vmm_unit::start_of_sim_ph();
```

Description

Method called at start of the simulation.

Example

```
class cpu_driver extends vmm_unit;
   ...
   function void start_of_sim_ph();
      if (iport == null)
       `vmm_fatal(log, "Virtual port not connected to the
        actual interface instance");
   endfunction
   ...
endclass
```

vmm_unit::disabled_ph()

Method executes in lieu of reset_ph() when unit disabled.

SystemVerilog

```
virtual task vmm_unit::disabled_ph();
```

Description

Method executed in lieu of the reset_ph() method if this vmm_unit instance is disabled.

Example

```
class unitExtension extends vmm_unit;
   function void unitExtension::disabled_ph();
      `vmm_note
      (log,`vmm_sformatf("unitExtension::disabled_ph"));
   ...
   endfunction:disabled_ph
endclass
```

vmm_unit::reset_ph()

Method for reset .

SystemVerilog

```
vmm_unit::reset_ph()
```

Description

Reset this unit if it is enabled.

Example

```
class memsys_env extends vmm_unit;
   task reset_ph();
      // Resetting the DUT
     test_top.reset <= 1'b0;
     repeat(1) @(test_top.port0.cb)
     test_top.reset <= 1'b1;
     repeat(10) @(test_top.port0.cb)
     test_top.reset <= 1'b0;
     `vmm_verbose(this.log,"RESET DONE...");
   endtask
endclass
```

vmm_unit::training_ph()

Method for Training.

SystemVerilog

```
virtual task vmm_unit::training_ph();
```

Description

Initialization of this component, such as interface training, if it is enabled.

Example

```
class unitExtension extends vmm_unit;
   function void unitExtension::training_ph();
      `vmm_note
      (log,`vmm_sformatf("unitExtension::training_ph"));
      ...
   endfunction:training_ph
endclass
```

vmm_unit::config_dut_ph()

Method for DUT configuration.

SystemVerilog

```
virtual task vmm_unit::config_dut_ph();
```

Description

Initialization of the DUT attached to this component, if it is enabled.

Example

```
class vdmsys_env extends vmm_unit;
   function void config_dut_ph;
      top.write_reg(N_RD_PORT, 20);
      top.write_reg(N_WR_PORT, 30);
      ...
   endfunction
endclass
```

vmm_unit::start_ph()

Method to start unit components.

SystemVerilog

```
virtual task vmm_unit::start_ph();
```

Description

Method to start processes within this component, if it is enabled.

Example

```
class memsys_env extends vmm_unit;
   ...
   task start_ph();
      this.gen.start_xactor();
   endtask
   ...
endclass
```

vmm_unit::start_of_test_ph()

Method called at start of the test body.

SystemVerilog

```
virtual function void vmm_unit::start_of_test_ph();
```

Description

Method called at start of the test body, if it is enabled.

Example

```
class unitExtension extends vmm_unit;
   function void unitExtension::start_of_test_ph();
      `vmm_note
 (log,`vmm_sformatf("unitExtension::start_of_test_ph"));
   ...
   endfunction:start_of_test_ph
endclass
```

vmm_unit::run_ph()

Body of test, if it is enabled.

SystemVerilog

```
virtual task vmm_unit::run_ph();
```

Description

Body of test, if it is enabled. Can be interrupted by resetting this component. May be stopped.

Example

```
class unitExtension extends vmm_unit;
   function void unitExtension::run_ph();
      `vmm_note
      (log,`vmm_sformatf("unitExtension::run_test_ph"));
      ...
   endfunction:run_ph
endclass
```

vmm_unit::shutdown_ph()

Method to stop all unit components.

SystemVerilog

```
virtual task vmm_unit::shutdown_ph();
```

Description

Method to stop processes within this component, if it is enabled.

Example

```
class cpu_subenv extends vmm_unit;
   ...
   task shutdown_ph();
      if (enable_gen) this.gen.stop_xactor();
   endtask
   ...
endclass
```

vmm_unit::cleanup_ph()

Method for post-execution.

SystemVerilog

```
virtual task vmm_unit::cleanup_ph();
```

Description

Method to perform post-execution verification, if it is enabled.

Example

```
class unitExtension extends vmm_unit;
   function void unitExtension::cleanup_ph();
      `vmm_note
      (log,`vmm_sformatf("unitExtension::cleanup_ph"));
      ...
   endfunction:cleanup_ph
endclass
```

vmm_unit::report_ph()

Method for test reporting.

SystemVerilog

```
virtual function void vmm_unit::report_ph();
```

Description

Method to perform post-test pass/fail reporting, if it is enabled.

Example

```
class memsys_env extends vmm_unit;
   function void report_ph();
      sb.report;
      ...
   endfunction
endclass
```

vmm_unit::destruct_ph()

Method to clear test specific components.

SystemVerilog

```
virtual function void vmm_unit::destruct_ph();
```

Description

Method to remove connections, phases and other testbench extensions introduced by this test and make sure it returns to its original state. Also ensures that memory can be garbage-collected.

Example

```
class unitExtension extends vmm_unit;
   function void unitExtension::destruct_ph();
      `vmm_note
      (log,`vmm_sformatf("unitExtension::destruct_ph"));
      this.un1.kill_object();
               //kill test specific component
   endfunction:destruct_ph
endclass
```

vmm_unit::override_phase();

Method to execute new phase definition in lieu of the existing one.

SystemVerilog

```
virtual function vmm_phase_def vmm_unit::override_phase(string name,
vmm_phase_def def);
```

Description

Override the specified phase with the specified phase definition for this instance. If def is null, the override (if any) is removed. Returns the previous override phase definition (if any).

Example

```
class cust_configure_phase_def #(type T = unitExtension) extends
vmm_topdown_function_phase_def #(T);
 function void do_function_phase( T obj);
  obj.cust_config_ph();
endfunction
endclass

class unitExtension extends vmm_unit;
   function void unitExtension::config_ph();
      `vmm_note
    (log,`vmm_sformatf("unitExtension::configure_ph"));
   endfunction:config_ph

   function void unitExtension::cust_config_ph();
      `vmm_note
(log,`vmm_sformatf("unitExtension::cust_config_ph"));
   endfunction:cust_config_ph
endclass

cust_configure_phase_def  cust_cfg = new();
unitExtension   m1 = new("unitExtension","m1");
`void(m1.override_phase("configure",cust_cfg ));
```

vmm_timeline

The vmm_timeline class coordinates simulation through a user-defined timeline with pre-defined test phases as follows:

```
“build”,
“configure”,
“connect”,
“start_of_sim”,
“reset”,
“training”,
“config_dut”,
“start”,
“start_of_test”,
“run_test”,
“shutdown”,
“cleanup”,
“report”,
“destruct”.
```

Phases may be subsequently added or removed as needed.

vmm_timeline::insert_phase();

Method to insert a phase in timeline.

SystemVerilog

```
function bit vmm_timeline::insert_phase(string phase_name, string
before_name, vmm_phase_def def, string fname = "", int lineno = 0)
```

Description

Create the specified phase before the specified phase in this timeline and issue a Note that a new user-defined phase has been defined. If the phase already exists, add this definition to the existing phase definition. If the before_phase is specified as "^", inserts the phase at the beginning of the timeline. If it is specified as "$", inserts the phase at the end of the timeline. Returns TRUE if the phase insertion was successful.

Example

```
class udf_start_def extends vmm_fork_task_phase_def
#(unitExtension);
   ...
endclass
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      vmm_timeline t = this.get_timeline();
      udf_start_def udfstartph = new;
      ...
      if(t.insert_phase("udf_start", "start_of_sim",
                        udfstartph) == 0)
      `vmm_error (log, " ... ");
    endfunction

endclass
```

vmm_timeline::delete_phase();

Method to delete specified phase from timeline.

SystemVerilog

```
function bit vmm_timeline::delete_phase(string phase_name, string
fname = “”, int lineno = 0)
```

Description

Delete the specified phase in this timeline. Returns FALSE if the phase does not exist.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      vmm_timeline t = this.get_timeline();
      ...
      t.delete_phase ("connect");
      ...
   endfunction
endclass
```

vmm_timeline::rename_phase();

Method to give a new name to the specified phase.

SystemVerilog

```
function bit vmm_timeline::rename_phase(string old_name, string
new_name, string fname = “”, int lineno = 0)
```

Description

Rename the specified phase in this timeline to the new phase name. Returns FALSE if the original named phase does not exist or if a phase already exists with the new name. Issue a warning that a phase had been renamed.Renaming timeline default phases are disallowed.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      vmm_timeline t = this.get_timeline();
      ...
     // Renaming pre-defined phase 'start_of_sim'
     if(t.rename_phase("start_of_sim",
                       "renamed_start_of_sim") == 0)
        `vmm_error(log, " ... ");
        ...
   endfunction
endclass
```

vmm_timeline::get_phase()

Method to get the phase descriptor for a specified phase.

SystemVerilog

```
function vmm_phase vmm_timeline::get_phase(string name)
```

Description

Return the descriptor of the specified phase in this timeline. Returns null if the specified phase is unknown.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      vmm_phase ph;
      vmm_timeline t = this.get_timeline();
      ...
      ph = t.get_phase ("start_of_sim");
      ...
   endfunction
endclass
```

vmm_timeline::task_phase_timeout();

Method to set the timeout value for any task phase.

SystemVerilog

```
function bit vmm_timeline::task_phase_timeout(string name, int
unsigned delta, string fname = "", int lineno = 0)
```

Description

Set the timeout value for the completion of the specified task phase. If the task phase does not complete within the time specified in the timeout value, an error message will be generated.

Returns FALSE if the specified phase does not exist or is not a task phase.

A timeout value of 0 specifies no timeout value. Calling this method while the phase is currently executing causes the timer to be reset to the specified value. By default ,phases do not have timeout.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      vmm_timeline t = this.get_timeline();
      ...
      if(t.task_phase_timeout("reset",4) == 0)
         `vmm_error (log, " ... ");
         ...
   endfunction
endclass
```

vmm_timeline::get_next_phase_name()

Method to get the name of the following phase.

SystemVerilog

```
function string vmm_timeline::get_next_phase_name(string name)
```

Description

Return the name of the phase following the specified phase. Returns “$” if the specified phase is the last one. Returns “?” if the specified phase is unknown.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      string nxt_ph;
      vmm_timeline t = this.get_timeline();
      ...
      nxt_ph = t.get_next_phase_name ("start_of_sim");
                                      //returns "reset"
      ...
   endfunction
endclass
```

vmm_timeline::get_previous_phase_name()

Method to get the name of the preceding phase

SystemVerilog

```
function string vmm_timeline::get_previous_phase_name(string name)
```

Description

Return the name of the phase preceding the specified phase. Returns "^" if the specified phase is the first one. Returns "?" if the specified phase is unknown.

Example

```
class unitExtension extends vmm_unit;
   ...
   function void build_ph ();
      string prv_ph;
      vmm_timeline t = this.get_timeline();
      ...
      prv_ph = t.get_previous_phase_name ("start_of_sim");
                                  //returns "connect"
     ...
   endfunction
endclass
```

task vmm_timeline::run_phase()

Method to run timeline, up toand including specified phase.

SystemVerilog

```
vmm_timeline::run_phase(string name = "$", string fname = "", int
lineno = 0)
```

Description

Executes the phases in this timeline, up to and including, the specified phase. For name "$", run all phases.

Example

```
class test extends vmm_test;
   ...
   vmm_timeline topLevelTimeline;
   ...
endclass
...
initial begin
   test test1 = new ("test1", "test1");
   test1.topLevelTimeline.run_phase ("build");
   ...
   test1.topLevelTimeline.run_phase ();
end
```

vmm_timeline::step_function_phase()

Method to step to next executable phase.

SystemVerilog

```
function void vmm_timeline::step_function_phase(string name, string
fname = “”, int lineno = 0)
```

Description

Executes the specified function phase in this timeline. Must be a function phase and must be the next executable phase.

Example

```
class test extends vmm_test;
   ...
   vmm_timeline topLevelTimeline;
   ...
endclass
...
initial begin
   test test1 = new ("test1", "test1");
   ...
   test1.topLevelTimeline.run_phase ("configure");
   test1.topLevelTimeline.step_function_phase ("connect");
   test1.topLevelTimeline.step_function_phase ("start_of_sim");
   ...
end
```

vmm_timeline::abort_phase()

Method to abort the specified phase if currently executing.

SystemVerilog

```
function void vmm_timeline::abort_phase(string name, string fname =
“”, int lineno = 0)
```

Description

Abort the execution of the specified phase if it is the currently executing phase in the timeline. If another phase is executing, issues a warning message if the specified phase has already been executed to completion and issues an error message if the specified phase has not yet been started.

Example

```
class test extends vmm_test;
    . .
   vmm_timeline topLevelTimeline;
   ...
endclass
...
initial begin
   test test1 = new ("test1", "test1");
   ...
   fork
      test1.topLevelTimeline.run_phase();
      #(reset_cycle) test1.topLevelTimeline.abort_phase ("reset");
      ...
   join
   ...
end
```

vmm_timeline::reset_to_phase()

Method to reset timeline to the specified phase.

SystemVerilog

```
task vmm_timeline::reset_to_phase(string name = "^", string fname =
"", int lineno = 0)
```

Description

Resets this timeline to the specified phase. Any task-based phase still concurrently running is aborted. If the timeline is reset to the "configure" phase or earlier, all of its vmm_unit sub-instances are enabled.

Example

```
class test extends vmm_test;
   ...
   vmm_timeline topLevelTimeline;
   ...
endclass
...
initial begin
   test test1 = new ("test1", "test1");
   ...
   fork
      test1.topLevelTimeline.run_phase();
      #9 test1.topLevelTimeline.reset_to_phase ("build");
      ...
   join
   ...
end
```

vmm_timeline::jump_to_phase()

Abort the execution of the timeline immediately jump to the beginning of the specified phase.

SystemVerilog

```
function void vmm_timeline::jump_to_phase(string name, string fname
= “”, int lineno = 0)
```

Description

Abort the execution of the timeline and immediately jump to the beginning of the specified phase (but does not start executing it). Issues a warning message if the specified phase has already been started or completed.
Executing a phase without the intervening phases may cause severe damage to the state of the executing testcase and verification environment and should be used with care. Should typically be used to abort a testcase or simulation and jump to the report or destruct phase.

Example

```
class timelineExtension #(string jump_phase = "report", int
                          delay_in_jump = 10) extends vmm_timeline;
   . . .
   task reset_ph;
      #delay_in_jump jump_to_phase(jump_phase);
   endtask
   . . .
endclass
```

vmm_timeline::get_current_phase_name()

Method to display current executing phase of the timeline.

SystemVerilog

```
function string vmm_timeline::get_current_phase_name()
```

Description

Return the current phase where the timeline instance is at a given point of time

Example

```
class test extends vmm_test;
   ...
   vmm_timeline topLevelTimeline;
   ...
endclass
...
initial begin
   test test1 = new ("test1", "test1");
   ...
   fork
      begin
         test1.topLevelTimeline.run_phase();
      end
      begin
         #20 `vmm_note (log,psprintf("Current Simulation
                        Phase for test1 is : %s ",
        test1.topLevelTimeline.get_current_phase_name() ));
      end
      ...
   join
   ...
end
```

vmm_timeline::display_phases()

Method to display all phases left to be executed.

SystemVerilog

```
function void vmm_timeline::display_phases()
```

Description

Display all phases left to be executed by this timeline.

Example

```
class test extends vmm_test;
   ...
   vmm_timeline topLevelTimeline;
   ...
endclass
...
initial begin
   test test1 = new ("test1", "test1");
   ...
   fork
      begin
         test1.topLevelTimeline.run_phase();
      end
      begin
         #20 test1.topLevelTimeline.display_phases();
      end
      ...
   join
   ...
end
```

Class vmm_timeline_callbacks

Façade class for callback methods provided by a timeline.

Example

```
class timeline_callbacks extends vmm_timeline_callbacks;

   virtual function void my_f1();

   endfunction

   virtual function void my_f2();

   endfunction

endclass
```

vmm_timeline_callback::break_on_phase()

This method is called if the +break_on_X_phase option has been set for this timeline instance.

SystemVerilog

```
function void vmm_timeline_callabcks::break_on_phase(vmm_timeline
t1,
                                                      string name)
```

Description

This method is called if the +break_on_X_phase option has been set for this timeline instance. The arguments are the instance of the timeline and the name of the phase (“X”). If no callbacks are registered, ***$stop*** is called instead of this method.

Example

```
class timeline_callbacks extends vmm_timeline_callbacks;
    vmm_log log;

    function new(vmm_log log);
      this.log = log;
    endfunction

    function void break_on_phase(vmm_timeline tl, string name);
     if(name==”reset”)
      `vmm_note(log, "user callback executing for reset phase");
    endfunction
  endclass

vmm_timeline tl;

initial begin
    timeline_callbacks  cb1;
    tl = new("my_timeline", "tl");
    cb1 = new(tl.log);
    tl.append_callback(cb1);
    tl.run_phase();
```

vmm_timeline::append_callback()

Method to append the specified callabck.

SystemVerilog

```
function bit vmm_timeline::append_callback(vmm_timeline_callback cb)
```

Description

Append the specified callback extension to the callback registry for this timeline. Returns TRUE if the registration was successful.

Example

```
class timeline_callbacks extends vmm_timeline_callbacks;
   virtual function void my_f1();
   endfunction
endclass

class timelineExtension extends vmm_timeline;
  function new (string name, string inst, vmm_unit parent=null);
     super.new(name,inst,parent);
  endfunction

  function void build_ph();
   `vmm_callback(timeline_callbacks,my_f1());
  endfunction:build_ph
  . . .
endclass

class  timelineExtension_callbacks extends timeline_callbacks;
   int my_f1_counter++;

   virtual function void my_f1();
      my_f1_counter++;
   endfunction
endclass
```

```
initial begin
   timelineExtension tl = new ("my_timeline", "t1");
   timelineExtension_callbacks cb1 = new();

   tl.append_callback(cb1);
      . . .
end
```

vmm_timeline::prepend_callback()

Method to prepend the specified callabck.

SystemVerilog

```
function bit vmm_timeline::prepend_callback(vmm_timeline_callback
cb)
```

Description

Prepend the specified callback extension to the callback registry for this timeline. Returns TRUE if the registration was successful.

Example

```
class timeline_callbacks extends vmm_timeline_callbacks;
   virtual function void my_f1();
   endfunction
endclass

class timelineExtension extends vmm_timeline;
  function new (string name, string inst, vmm_unit parent=null);
     super.new(name,inst,parent);
  endfunction

  function void build_ph();
   `vmm_callback(timeline_callbacks,my_f1());
  endfunction:build_ph
  . . .
endclass

class  timelineExtension_callbacks extends timeline_callbacks;
   int my_f1_counter++;

   virtual function void my_f1();
      my_f1_counter++;
   endfunction
endclass
```

```
initial begin
   timelineExtension tl = new (“my_timeline”, “t1”);
   timelineExtension_callbacks cb1 = new();
   timelineExtension_callbacks cb2 = new();
   tl.append_callback(cb1);
   tl.prepend_callback(cb2);
   . . .
end
```

vmm_timeline::unregister_callback()

Method to unregister a callback.

SystemVerilog

```
function bit vmm_timeline::unregister_callback(vmm_timeline_callback
cb);
```

Description

Remove the specified callback extension from the callback registry for this timeline. Returns TRUE if the unregistration was successful.

Example

```
class timeline_callbacks extends vmm_timeline_callbacks;
   virtual function void my_f1();
   endfunction
endclass

class timelineExtension extends vmm_timeline;
  function new (string name, string inst, vmm_unit parent=null);
     super.new(name,inst,parent);
  endfunction

  function void build_ph();
   `vmm_callback(timeline_callbacks,my_f1());
  endfunction:build_ph
  . . .
endclass

class  timelineExtension_callbacks extends timeline_callbacks;
   int my_f1_counter++;

   virtual function void my_f1();
      my_f1_counter++;
   endfunction
endclass

initial begin
   timelineExtension tl = new ("my_timeline", "t1");
   timelineExtension_callbacks cb1 = new();
```

```
   timelineExtension_callbacks cb2 = new();
   tl.append_callback(cb1);
   tl.append_callback(cb2);
   . . .
   tl.unregister_callback(cb2);
   . . .
end
```

Multi-test Management Base class (vmm_test)

The vmm_test class is an extension of vmm_timeline and handles the test execution timeline with all of the default pre-defined phases. This is used as the base class for all tests.

Instances of this class must be either root objects or children of vmm_test objects.

Example

```
class my_test1 extends vmm_test;
   `vmm_typename(my_test1)
   function new(string name);
      super.new(name);
   endfunction

   function void config_ph;
      cfg cfg1 = new;
      if (cfg1.randomize)
        `vmm_note (log, "CFG randomized successfully" );
      else
         `vmm_error (log, "CFG randomization failed" );
   endfunction
endclass
```

vmm_test::set_config()

SystemVerilog

```
function void vmm_test::set_config()
```

Description

This method may be used to set vmm_unit factory instances and configuration parameters in vmm_unit instances outside of the scope of the test module using the classname::create_by_*() and vmm_opts::set_*() methods.

This method can only be used if tests are executed one per simulation. Using this method makes tests non-concatenatable.

Example

```
class my_ahb_trans extends vmm_object;
   ...
   `vmm_class_factory(my_ahb_trans)
endclass

class my_test1 extends vmm_test;
   `vmm_typename(my_test1)
    function new(string name);
       super.new(name);
    endfunction

   function set_config();
      ahb_trans::create_by_new("@%*",
                               my_ahb_trans::this_type,
                               log, `__FILE__, `__LINE__);
   endfunction
   ...
endclass
```

Simulation timeline management class (vmm_simulation)

The vmm_simulation class extending from vmm_unit is a top-level singleton module that manages the end-to-end simulation timelines. It includes pre-test and post-test timelines with pre-defined pre-test and post-test phases. The pre-defined pre-test phases are “rtl_config” ,“build”, “configure” and “connect”. The pre-defined post-test phase is “report” .

Example

```
program tb_top;
   class my_test extends vmm_test;
      ...
   endclass

   class my_env extends vmm_unit;
      ...
   endclass

   initial begin
      my test test1 = new("test1");
      my_env env = new("env");
      vmm_simulation my_sim;
      ...
      my_sim = vmm_simulation :: get_sim();
      ...
   end
endprogram
```

vmm_simulation::get_sim()

Returns the vmm_simulation singleton.

SystemVerilog

```
static function vmm_simulation vmm_simulation::get_sim()
```

Description

Returns the vmm_simulation singleton.

Example

```
program tb_top;
   class my_test extends vmm_test;
      ...
   endclass

   class my_env extends vmm_unit;
      ...
   endclass

   initial begin
      my test test1 = new("test1");
      my_env env = new("env");
      vmm_simulation my_sim;
      ...
      my_sim = vmm_simulation :: get_sim();
      ...
   end
endprogram
```

vmm_simulation::get_pre_timeline()

Returns the pre-test timeline.

SystemVerilog

```
static function vmm_timeline vmm_simulation::get_pre_timeline()
```

Description

Returns the pre-test timeline.

Example

```
TBD
```

vmm_simulation::get_top_timeline()

Returns the top-level test timeline.

SystemVerilog

```
static function vmm_timeline vmm_simulation::get_top_timeline()
```

Description

Returns the top-level test timeline.

Example

```
program tb_top;
   class my_test extends vmm_test;
      ...
   endclass

   class my_env extends vmm_unit;
      ...
   endclass

   initial begin
      my test test1 = new("test1");
      my_env env = new("env");
      vmm_timeline my_tl;
      ...
      my_tl = vmm_simulation::get_top_timeline();
      ...
   end
endprogram
```

vmm_simulation::get_post_timeline()

Returns the post-test timeline.

SystemVerilog

```
static function vmm_timeline vmm_simulation::get_post_timeline()
```

Description

Returns the post-test timeline.

Example

```
TBD
```

vmm_simulation::allow_new_phases()

Enable the addition of user-defined phases in timelines.

SystemVerilog

```
static function void vmm_simulation::allow_new_phases(bit allow = 1)
```

Description

Enable the addition of user-defined phases in timelines, if *allow* is TRUE. If the insertion of a user-defined phase is attempted when new phases are not allowed, an error message will be issued.

By default, addition of user-defined phases are not allowed

Example

```
program tb_top;
   class my_test extends vmm_test;
      ...
   endclass
   class my_env extends vmm_unit;
      ...
   endclass

   initial begin
      my test test1 = new("test1");
      my_env env = new("env");
      ...
      vmm_simulation::allow_new_phases();
      // insert new phases using
      //  insert_pre/post_test_phase tasks
   end
endprogram
```

vmm_simulation::display_phases()

Display how the various phases in the various timelines will be executed

SystemVerilog

```
static function void vmm_simulation::display_phases()
```

Description

Display how the various phases in the various timelines will be executed (e.g. in sequence or in parallel). Should be invoked after the “build” phase.

Example

```
program tb_top;
   class my_test extends vmm_test;
     virtual function void start_of_sim_ph();
        vmm_simulation::display_phases();
     endfunction
   endclass

   class my_env extends vmm_unit;
   endclass

   initial begin
      my test test1 = new("test1");
      my_env env = new("env");
      ...
      vmm_simulation::run_tests();
   end
endprogram
```

vmm_simulation::run_tests()

Run tests specified at run-time.

SystemVerilog

```
task vmm_simulation::run_tests()
```

Description

Run tests specified at run-time using +vmm_test / +vmm_test_file or run default test

The following is the usage of +vmm_test_file/+vmm_test to specify testcase at run-time:

```
+vmm_test_file+<file name>    - will run list of tests specified in
the file (if concatenation is allowed, otherwise issues a fatal
message)
+vmm_test=<testname>+<testname>+…
Run list of specified tests
+vmm_test=<test name>
- run specific test
+vmm_test=ALL_TESTS
- run all the registered tests (if concatenation is allowed,
otherwise issues a fatal message)
```

Example

```
program tb_top;
   class my_test extends vmm_test;
   endclass

   class my_env extends vmm_unit;
   endclass

   initial begin
      my test test1 = new("test1");
      my_env env = new("env");
      ...
      vmm_simulation::run_tests();
   end
```

```
endprogram
```

The phase description class (vmm_phase)

This class is used as a container for phase descriptors and their associated statistical information.

vmm_phase::get_name()

Method to get the phase descriptor name.

SystemVerilog

```
function string vmm_phase::get_name()
```

Description

Returns the name of the phase.

Example

```
vmm_timeline top;
vmm_phase ph;
string ph_name;

initial begin
   top = new("top", "top");
   ph = top.get_phase("connect");
   ...
   ph_name = ph.get_name();  //returns string "connect"
   ...
end
```

vmm_phase::get_timeline()

Method to get the enclosing timeline.

SystemVerilog

```
function vmm_timeline vmm_phase::get_timeline()
```

Description

Returns the timeline containing this phase.

Example

```
vmm_timeline top;
vmm_phase ph;

initial begin
   vmm_timeline t;
   top = new("top", "top");
   ph = top.get_phase("connect");
   ...
   t = ph.get_timeline;
   ...
end
```

vmm_phase::next_phase()

Method to get the following phase descriptor.

SystemVerilog

```
function vmm_phase vmm_phase::next_phase()
```

Description

Returns the following phase in the timeline containing this phase. Returns null if this is the last phase in the timeline.

Example

```
vmm_timeline top;
vmm_phase ph;

initial begin
   vmm_phase nx_ph;
   top = new("top", "top");
   ph = top.get_phase("connect");
   ...
   nx_ph = ph.next_phase(); //returns phase start_of_sim
   `vmm_note(log,`vmm_sformatf("%s will execute after connect”,
                               nx_ph.get_name());
   ...
end
```

vmm_phase::previous_phase()

Method to get the preceding phase descriptor.

SystemVerilog

```
function vmm_phase vmm_phase::previous_phase()
```

Description

Returns the preceeding phase in the timeline containing this phase. Returns null if this is the first phase in the timeline.

Example

```
vmm_timeline top;
vmm_phase ph;

initial begin
   vmm_phase prv_ph;
   top = new("top", "top");
   ph = top.get_phase("connect");
   ...
   prv_ph = ph.previous_phase(); //returns phase configure
   `vmm_note(log,`vmm_sformatf("connect will execute after %s ”,
                               prv_ph.get_name());
   ...
end
```

vmm_phase::is_running()

Method to get execution status of the phase.

SystemVerilog

```
function bit vmm_phase::is_running()
```

Description

Returns TRUE if the phase is currently being executed. Will always return FALSE for function phases, unless called from within the phase implementation function itself.

Example

```
vmm_timeline top;
vmm_phase ph;

initial begin
   top = new("top", "top");
   ph = top.get_phase("connect");
   ...
   wait(ph.is_running == 0);
   ...
end
```

vmm_phase::is_done()

Method to check completion status of the phase.

SystemVerilog

```
function int vmm_phase::is_done()
```

Description

Returns the number of times the phase has been completed.

Example

```
class myTest  extends vmm_timeline;
    function new(string name, string inst, vmm_object parent =
null);
      super.new(name, inst, parent);
    endfunction

    task run_ph;
      #5;
    endtask

  endclass

  vmm_log           log = new("test", "main");
  myTest            top;

  initial begin
    vmm_phase        ph_start_of_sim;
    vmm_phase        ph_reset;
    top = new("top", "top");
    ph_start_of_sim = top.get_phase("start_of_sim");
    ph_reset = top.get_phase("reset");

    fork
      top.run_phase();
    join_none
      #4 top.reset_to_phase("reset"); //abroting run_test
   #10;
```

```
    if(ph_reset.is_done() != 2)
      `vmm_error(log,`vmm_sformatf(
            $psprintf("Expected reset to be done 2 times, is_done
returns %d",ph_reset.is_done)));

    if(ph_start_of_sim.is_done() != 1)
      `vmm_error(log,
            $psprintf("Expected start_of_sim to be done 1 times,
is_done returns %d",ph_start_of_sim.is_done));
```

vmm_phase::is_aborted()

Method to check aborted status of the phase.

SystemVerilog

```
function int vmm_phase::is_aborted()
```

Description

Returns the number of times the phase has been aborted.

Example

```
  class myTest  extends vmm_timeline;
    function new(string name, string inst, vmm_object parent =
null);
      super.new(name, inst, parent);
    endfunction

    task reset_ph;
      $display("%t:Starting Reset", $time);
      #5;
      $display("%t:Finishing Reset", $time);
    endtask

    task training_ph;
      #5;
    endtask

    task run_ph;
      #5;
    endtask

  endclass

  vmm_log           log = new("test", "main");
  myTest            top;

  initial begin
    vmm_phase        ph_reset;
```

```
    top = new("top", "top");
    ph_reset = top.get_phase("reset");


    fork
      top.run_phase();
    join_none
      #7 top.abort_phase("training"); //aborting training
      #1 top.reset_to_phase("reset"); //aborting run_test
      #1 top.jump_to_phase("run_test");  //aborting reset, skipping
training-start_of_test

   #10;

    if(ph_reset.is_aborted() != 2)
      `vmm_error(log,`vmm_sformatf(
            $psprintf("Expected reset to abort 2 times, is_aborted
returns %d",ph_reset.is_aborted)));
```

vmm_phase::is_skipped()

Returns the number of times the phase has been skipped.

SystemVerilog

```
function int vmm_phase::is_skipped()
```

Description

Returns the number of times the phase has been skipped.

Example

```
  class myTest  extends vmm_timeline;
    function new(string name, string inst, vmm_object parent =
null);
      super.new(name, inst, parent);
    endfunction

    task reset_ph;
      $display("%t:Starting Reset", $time);
      #5;
      $display("%t:Finishing Reset", $time);
    endtask

    task training_ph;
      #5;
    endtask

    task run_ph;
      #5;
    endtask

  endclass

  vmm_log           log = new("test", "main");
  myTest            top;

  initial begin
    vmm_phase        ph_training;
```

```
    top = new("top", "top");
    ph_training = top.get_phase("training");

    fork
      top.run_phase();
    join_none
      #9 top.jump_to_phase("run_test");  //aborting reset, skipping
 training-start_of_test
   // join

   #10;

    if(ph_training.is_skipped() != 1)
      `vmm_error(log,`vmm_sformatf(
           $psprintf("Expected training to abort 1 times,
is_skipped returns %d",ph_training.is_skipped)));
```

event started

Phase execution start event.

Description

This event is triggered when the execution of this phase starts.

Example

```
vmm_timeline top;
vmm_phase ph;

initial begin
   top = new("top", "top");
   ph = top.get_phase("connect");
   ...
   @(ph.started);
   `vmm_note(log,” connect phase execution started”);
   ...
end
```

event completed

Phase execution completion event.

Description

This event is triggered when the execution of this phase completed.

Example

```
vmm_timeline top;
vmm_phase ph;

initial begin
   top = new("top", "top");
   ph = top.get_phase("connect");
   @(ph.completed);
   `vmm_log (log, "Completed execution of phase connect");
   ...
end
```

The phase definition base class (vmm_phase_def)

Description

This virtual class should be extended to create any user defined phase.

vmm_phase_def::is_function_phase()

Method to check type of phase definition (if function).

SystemVerilog

```
virtual function bit vmm_phase_def::is_function_phase()
```

Description

Returns TRUE if this phase is executed by calling the vmm_phase_def::run_function_phase() method. Returns FALSE otherwise.

Example

```
virtual class user_function_phase_def #( user_function_phase_def)
extends vmm_topdown_function_phase_def;
   function bit is_function_phase();
      return 1;
   endfunction:is_function_phase
endclass
```

vmm_phase_def::is_task_phase()

Method to check type of phase definition (if task).

SystemVerilog

```
virtual function bit vmm_phase_def::is_task_phase()
```

Description

Returns TRUE if this phase is executed by calling the vmm_phase_def::run_task_phase() method.Returns FALSE otherwise.

Example

```
virtual class user_task_phase_def #( user_task_phase_def) extends
vmm_fork_task_phase_def;
   function bit is_task_phase();
      return 1;
   endfunction:is_task_phase
endclass
```

vmm_phase_def::run_function_phase();

Method to execute phase definition used by timeline.

SystemVerilog

```
virtual function void vmm_phase_def::run_function_phase(string name,
vmm_object root, vmm_log log)
```

Description

Execute the function phase, under the specified name on the specified root object. This method must be overridden if vmm_phase_def::is_function_phase() returns TRUE.

Example

```
 virtual class user_function_phase_def #( user_function_phase_def)
                extends vmm_topdown_function_phase_def;
   function bit is_function_phase();
      return 1;
   endfunction:is_function_phase

   function run_function_phase(string name, vmm_object root,
                               vmm_log log);
     `vmm_note(log,`vmm_sformatf("Executing phase %s for %s",
                name,root.get_object_name());
   endfuction
endclass
```

vmm_phase_def::run_task_phase()

Method to execute phase definition used by timeline.

SystemVerilog

```
virtual task vmm_phase_def::run_task_phase(string name, vmm_object
root, vmm_log log)
```

Description

Execute the task phase, under the specified name on the specified root object. This method must be overridden if vmm_phase_def::is_task_phase() returns TRUE.

Example

```
virtual class user_task_phase_def #( user_task_phase_def)
              extends vmm_fork_task_phase_def;
   function bit is_task_phase();
      return 1;
   endfunction:is_task_phase
   task run_task_phase(string name, vmm_object root, vmm_log log);
     `vmm_note(log,`vmm_sformatf(" Executing phase %s for %s”,
               name,root.get_object_name());
   endtask

endclass
```

vmm_topdown_function_phase_def

Pre-defined top-down phase definition.

SystemVerilog

```
class vmm_topdown_function_phase_def #(type T) extends vmm_phase_def
```

Description

Implements the vmm_phase_def::run_function_phase(). To call the vmm_topdown_function_phase_def::do_function_phase() method on any object of specified type within the vmm_object hierarchy under the specified root in a top-down order

vmm_topdown_function_phase_def::do_function_phase();

Method to execute on object for particular phase execution.

SystemVerilog

```
virtual function void
vmm_topdown_function_phase_def::do_function_phase(T obj)
```

Description

Implementation of the function phase on an object of the specified type.

The user can choose to execute some non-delay processes of the specified object in this method of a new phase definition class extended from this class.

Example

```
class udf_phase_def extends vmm_topdown_function_phase_def;
   function void do_function_phase(vmm_unit un1);

      un1.my_method();
    endfunction
endclass
```

vmm_bottomup_function_phase_def

Pre-defined bottom-up phase definition

SystemVerilog

```
class vmm_bottomup_function_phase_def #(type T) extends
vmm_phase_def
```

Description

Implements the vmm_phase_def::run_function_phase(). To call the vmm_bottomup_function_phase_def::do_function_phase() method on any object of specified type within the vmm_object hierarchy under the specified root in a bottom-up order.

vmm_bottomup_function_phase_def::do_function_phase();

Method to execute on object for particular phase execution.

SystemVerilog

```
virtual function void
vmm_bottomup_function_phase_def::do_function_phase(T obj)
```

Description

Implementation of the function phase on an object of the specified type. The user can choose to execute some non-delay processes of a specified object in this method of a new phase definition class extended from this class.

Example

```
class udf_phase_def extends vmm_bottomup_function_phase_def;
   function void do_function_phase(vmm_unit un1);
      un1.my_method();
   endfunction
endclass
```

vmm_fork_task_phase_def

Pre-defined task based phase definition

```
class vmm_fork_task_phase_def #(type T) extends vmm_phase_def
```

SystemVerilog

Description

Implements the vmm_phase_def::run_task_phase(). To fork a call to the vmm_fork_task_phase_def::do_task_phase() method on any object of a specified type within the vmm_object hierarchy under the specified root in a top-down order.

vmm_fork_task_phase_def::do_task_phase()

Method to execute on object for particular phase execution.

SystemVerilog

```
virtual task vmm_fork_task_phase_def::do_task_phase(T obj)
```

Description

Implementation of the task phase on an object of the specified type. User can choose to execute time-consuming processesin this method of a new phase definition class extended from this class.

Example

```
class udf_phase_def extends vmm_fork_task _phase_def;
   function void do_task_phase(vmm_unit un1);
      un1.my_method();
   endfunction
endclass
```

vmm_null_phase_def

pre-defined null phase definition.

SystemVerilog

```
class vmm_null_phase_def extends vmm_phase_def
```

Description

Implements empty vmm_phase_def::run_function_phase(). Typically used to override a predefined phase to skip its pre-defined implementation for a specific vmm_unit instance.

Example

```
class myphase_def extends vmm_null_phase_def #(moduleExtension);

endclass : myphase_def

myphase_def  null_ph = new();
unit_extension   m1 = new("unitExtension","m1");
`void(m1.override_phase("configure",null_ph ));
//nothing to de done for this unit in configure phase
```

vmm_xactor_phase_def

Pre-defined vmm_xactor phase definition class.

SystemVerilog

```
class vmm_xactor_phase_def #(type T) extends vmm_phase_def;
```

Description

Implements the vmm_xactor_phase_def::run_function_phase(). To call the vmm_xactor_phase_def::do_function_phase() method on any object of specified type within the vmm_object hierarchy with specified name/instance

vmm_start_xactor_phase_def

Pre-defined vmm_start_xactor phase definition class.

SystemVerilog

```
class vmm_start_xactor_phase_def extends vmm_xactor_phase_def;
```

Description

Implements the vmm_start_xactor_phase_def::do_function_phase().This function calls start_xactor() function on specified object of type vmm_xactor.

Example

```
  class consumer extends vmm_xactor ;
    packet_channel   in_chan;
    function new(string inst, packet_channel in_chan);
      super.new("consumer", inst);
      this.in_chan = in_chan;
    endfunction
    ...
    ...

  class consumer_timeline #(string phase = "start") extends
vmm_timeline;
    `vmm_typename(consumer_timeline)
    consumer         xactor;
    packet_channel   chan;

    function new (string inst, packet_channel  chan, vmm_unit parent
= null);
      super.new(get_typename(),inst, parent);
      this.chan = chan;
    endfunction

    function void build_ph;
      xactor = new("xactor", chan);
      xactor.set_parent_object(this);
    endfunction
```

```
    function void connect_ph;
      vmm_start_xactor_phase_def  start = new("consumer","xactor");
      void'(this.insert_phase(phase, phase, start));
    enfunction
    .
    .
  endclass

consumer_timeline  #("start") ctl = new("ctl", chan);
```

vmm_stop_xactor_phase_def

Pre-defined vmm_stop_xactor phase definition class.

SystemVerilog

```
class vmm_stop_xactor_phase_def extends vmm_xactor_phase_def;
```

Description

Implements the vmm_stop_xactor_phase_def::do_function_phase().This function calls stop_xactor() function on specified object of type vmm_xactor.

Example

```
  class consumer extends vmm_xactor ;
    packet_channel   in_chan;
    function new(string inst, packet_channel in_chan);
      super.new("consumer", inst);
      this.in_chan = in_chan;
    endfunction
    ...
    ...

  class consumer_timeline #(string phase = "stop") extends
vmm_timeline;
    `vmm_typename(consumer_timeline)
    consumer         xactor;
    packet_channel   chan;

    function new (string inst, packet_channel  chan, vmm_unit parent
= null);
      super.new(get_typename(),inst, parent);
      this.chan = chan;
    endfunction

    function void build_ph;
      xactor = new("xactor", chan);
      xactor.set_parent_object(this);
    endfunction
```

```
    function void connect_ph;
      vmm_stop_xactor_phase_def  stop = new("consumer","xactor");
      void'(this.insert_phase(phase,phase, stop));
    enfunction
    .
    .
  endclass

consumer_timeline  #("shutdown") ctl = new("ctl", chan);
```

vmm_reset_xactor_phase_def

Pre-defined vmm_reset_xactor phase definition class.

SystemVerilog

```
class vmm_reset_xactor_phase_def extends vmm_xactor_phase_def;
```

Description

Implements the vmm_reset_xactor_phase_def::do_function_phase().This function calls reset_xactor() function on specified object of type vmm_xactor.

Example

```
  class consumer extends vmm_xactor ;
    packet_channel   in_chan;
    function new(string inst, packet_channel in_chan);
      super.new("consumer", inst);
      this.in_chan = in_chan;
    endfunction
    ...
    ...

  class consumer_timeline #(string phase = "reset") extends
vmm_timeline;
    `vmm_typename(consumer_timeline)
    consumer         xactor;
    packet_channel   chan;

    function new (string inst, packet_channel  chan, vmm_unit parent
= null);
      super.new(get_typename(),inst, parent);
      this.chan = chan;
    endfunction

    function void build_ph;
      xactor = new("xactor", chan);
      xactor.set_parent_object(this);
    endfunction
```

```
    function void connect_ph;
      vmm_reset_xactor_phase_def  reset = new("consumer","xactor");
      void'(this.insert_phase(phase,phase, reset));
    enfunction
    .
    .
  endclass

consumer_timeline  #("reset") ctl = new("ctl", chan);
```

Class Factory

This is the utility class to generate instances of any class through the factory mechanism

factory::this_type()

Returns a dummy instance of class factory

SystemVerilog

```
static function classname classname::this_type();
```

Description

Returns a dummy instance of this class that can be used to call the classname::allocate() method.

Example

```
ahb_trans::create_by_new("@%*", my_ahb_trans::this_type,
                         log, `__FILE__, `__LINE__);
```

factory::create_instance();

Create an instance of the specified class type

SystemVerilog

```
static function classname classname::create_instance(vmm_object
parent, string name, string fname = “”, int lineno = 0);
```

Description

Create an instance of the specified class type for the specified name in the scope created by the specified parent vmm_object . The new instance is created by calling allocate() or copy() on the corresponding factory instance, specified using the create_by_new() or create_by_copy() method, in this class, or any of its parent (base) classes. If no factory instance has been specified, creates an instance of this class. The newly created instance has the specified name and the specified vmm_object as parent if the newly created instance is extended from vmm_object.

Example

```
class ahb_trans extends vmm_object;
   ...
   `vmm_class_factory(ahb_trans)
` endclass

class ahb_gen extends vmm_unit;
   ahb_trans tr;
   virtual function void_build_ph();
      tr = ahb_trans::create_instance(this, "Ahb_Tr0",
                                      `__FILE__, `__LINE__);
    ...
   endfunction
endclass
```

factory::override_with_new();

Set the specified class instance as the create-by-construction factory.

SystemVerilog

```
static function void classname::override_with_new(string name,
classname factory, vmm_log log, string fname = "", int lineno = 0);
```

Description

Set the specified class instance as the create-by-construction factory when creating further instances of that class under the specified instance name. The instance name may be specified as a match pattern or regular expression. An instance name in a specific namespace may be specified by prefixing it with "spacename::". The classname::create_instance() method uses the allocate() method to create a new instance of this class.

This method should always be called using the following pattern:

```
master::override_with_new("@*", extended_master::this_type(),
this.log, `__FILE__, `__LINE__);
```

If the specified name matches the hierarchical name of atomic, single-stream or multi-stream scenario generators of the appropriate type, the matching factory instances they contain are immediately replaced with newly allocated instances of the specified class. If this method is called before the "build" phase, this replacement is delayed until the completion of that phase.

Example

```
class my_ahb_trans extends vmm_object;
   `vmm_class_factory(my_ahb_trans)
endclass

initial begin
   ahb_trans::override_with_new("@%*",
                            my_ahb_trans::this_type, log,
                            `__FILE__, `__LINE__);
end
```

factory::override_with_copy();

Schedules creation of a factory instance by copying the provided instance.

SystemVerilog

```
static function void classname::override_with_copy(string name,
classname factory, vmm_log log, string fname = "", int lineno = 0);
```

Description

Set the specified class instance as the create-by-copy factory when creating further instances of that class under the specified instance name. The instance name may be specified as a match pattern or regular expression. An instance name in a specific namespace may be specified by prefixing it with "spacename::". The classname::create_instance() method uses the copy() method to create new instance of this class.

If the specified name matches the hierarchical name of atomic, single-stream or multi-stream scenario generators of the appropriate type, the matching factory instances they contain are immediately replaced with copies of the specified factory instance. If this method is called before the "build" phase, this replacement is delayed until the completion of that phase.

Example

```
class ahb_trans extends vmm_object;
   rand bit [7:0] addr;
   `vmm_class_factory(ahb_trans)
endclass

initial begin
   ahb_trans tr;
   tr = new("gen_trans");
   tr.addr = 5;
   ahb_trans::override_with_copy("@%*", tr, log,
                                 `__FILE__, `__LINE__);
end
```

`vmm_class_factory(classname)

Macro for factory definition.

Description

Create the class factory methods above for the specified class. Must be specified within the class declaration of a class with a virtual allocate() and copy() methods, that can be called without any arguments.

Example

```
class ahb_trans extends vmm_object;
   rand bit [7:0] addr;
   ...
   `vmm_class_factory(ahb_trans)
endclass
```

Hierachical options (vmm_opts)

Utility class which provides the facility to pass values from the command line during runtime and/or from the source code across hierarchies.

vmm_opts::get_bit();

Returns TRUE if specified option is set via the command line ,else returns FALSE

SystemVerilog

```
static function bit vmm_opts::get_bit(string name, string doc = "",
int verbosity = 0, string fname = "", int lineno = 0);
```

Description

Returns TRUE if the argument name is specified on the command line, else returns FALSE. The option is specified using the command line +vmm_name OR +vmm_opts+name. The user can specify a description of the option through *doc*, the verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
bit b;
b =  vmm_opts::get_bit("FOO",
                       "Value set for 'b' from command line");

Command line: simv +vmm_FOO OR simv +vmm_opts+FOO
```

vmm_opts::get_int();

Returns int value if specified via the command line, else returns default value.

SystemVerilog

```
static function int vmm_opts::get_int(string name, int dflt = 0,
string doc = "", int verbosity = 0, string fname = "", int lineno =
0);
```

Description

Returns int value if the argument name and its int value are specified on the command line, else returns default value specified in the *dflt* argument. The option is specified using the command line +vmm_name=value OR +vmm_opts+name=value. The user can specify a description of the option through *doc*, the verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
int i;
i = vmm_opts::get_int ("FOO", 0,
                       "Value set for 'i' from command line");

Command line: simv +vmm_FOO=100 OR simv +vmm_opts+FOO=100
```

vmm_opts::get_string();

Returns the string value if specified via the command line, else returns default value.

SystemVerilog

```
static function string vmm_opts::get_string(string name, string
dflt, string doc = "", int verbosity = 0, string fname = "", int
lineno = 0);
```

Description

Returns string value if the argument name and its string value are specified on the command line, else returns default value specified in the *dflt* argument. The option is specified using the command line +vmm_name=value OR +vmm_opts+name=value. The user can specify a description of the option through *doc*, a verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
string str;
str = vmm_opts :: get_string ("FOO", "DEF",
                              "str value from command line");

Command line: simv +vmm_FOO=HELLO OR simv +vmm_opts+FOO=HELLO
```

vmm_opts::get_range();

Returns integer range if specified via the command line, else returns the default range.

SystemVerilog

```
static function void vmm_opts::get_range(string name, output int
min,max, input int dflt_min, dflt_max, string doc = "", int
verbosity = 0, string fname = "", int lineno = 0);
```

Description

Return the named integer-range-type option. A range option is specified using the syntax +vmm_name=[min:max] OR +vmm_opts+name=[min:max]. The user can specify a description of the option through *doc*, a verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
int min_val;
int max_val;

vmm_opts::get_range("FOO", min_val, max_val,
                    -1, -1, "SET range", 0);

Command line: simv +vmm_FOO=[5:10] OR simv +vmm_opts+FOO=[5:10]
```

vmm_opts::get_help()

Display the list of available/specified vmm_ runtime options.

SystemVerilog

```
static function vmm_opts::get_help(vmm_object root = null, int
verbosity = 0);
```

Description

Display the known options used by the verification environment with the specified vmm_object hierarchy, with verbosity lower than or equal to the absolute value of the specified verbosity. If no vmm_unit root is specified, the known options used by all object hierarchies are displayed. A verbosity value must be within the range - 10 to 10. If the specified verbosity value is negative, the hierarchical name of each vmm_unit instance that uses an option is also displayed.

Example

```
vmm_opts::get_help(this_object);
```

vmm_opts::get_obj();

Returns vmm_object instance, if specified through vmm_opts::set_object()

SystemVerilog

```
static function vmm_object vmm_opts::get_obj(output bit is_set,
string name, vmm_object dflt = null, string fname = "", int lineno =
0);
```

Description

Return the globally-named object-type option and set the "is_set" argument to TRUE if an explicit value is specified. Object-type options can only be set using the vmm_opts::set_object()

method.

Example

```
class A extends vmm_object;
endclass

initial begin
   A a = new ("a");
   vmm_object obj;
   bit is_set;
   obj = vmm_opts :: get_obj(is_set, "OBJ", a);
end
```

vmm_opts::get_object_bit();

Returns TRUE if named option is set for the hierarchy, else returns FALSE.

SystemVerilog

```
static function bit vmm_opts::get_object_bit(output bit is_set,
input vmm_object obj, string name, string doc = "", int verbosity =
0, string fname = "", int lineno = 0);
```

Description

Return the named boolean-type option for the specified object instance and set the "is_set" argument to TRUE if an explicit value was specified. The user can specify a description of the option through *doc*, a verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10 with 10 being the highest. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
class B extends vmm_object;
   bit foo, is_set;
   function new(string name, vmm_object parent=null);
      foo = vmm_opts::get_object_bit(is_set, this, "FOO",
                                     "SET foo value", 0);
   endfunction
endclass

Command line: simv +vmm_FOO@A:%:b
```

vmm_opts::get_object_int();

Returns int value if named integer option is set for the hierarchy, else returns default value.

SystemVerilog

```
static function int vmm_opts::get_object_int(output bit is_set,
input vmm_object obj, string name, int dflt = 0, string doc = "",
int verbosity = 0, string fname = "", int lineno = 0);
```

Description

Return the named integer-type option for the specified object instance and set the "is_set" argument to TRUE if an explicit value was specified.. The user can specify a description of the option through *doc*, a verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
class B extends vmm_object;
   int foo;
   function new(string name, vmm_object parent=null);
      bit is_set;
      super.new(parent,name);
      foo = vmm_opts::get_object_int(is_set, this, "FOO",
                                     2, "SET foo value", 0);
   endfunction
endclass

Command line: simv +vmm_FOO=25@%:X:b
```

vmm_opts::get_object_string();

Returns string value if named string option is set for the hierarchy, else returns default value.

SystemVerilog

```
static function string vmm_opts::get_object_string(output bit
is_set, input vmm_object obj, string name, string dflt = "", string
doc = "", int verbosity = 0, string fname = "", int lineno = 0);
```

Description

Return the named string-type option for the specified object instance and set the "is_set" argument to TRUE if an explicit value was specified. The user can specify a description of the option through *doc*, a verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
class B extends vmm_object;
   string foo="ZERO";
   function new(string name, vmm_object parent=null);
      bit is_set;
      super.new(parent,name);
      foo = vmm_opts::get_object_string(is_set, this,
                                        "FOO", "DEF_VAL",
                                        "SET foo value", 0);
   endfunction
endclass
Command line: simv +vmm_FOO=HELLO@%:X:b
```

vmm_opts::get_object_range();

Returns the integer range for the specified hierarchy.

SystemVerilog

```
static function void vmm_opts::get_object_range(output bit is_set,
input vmm_object obj, string name, output int min,max, input int
dflt_min, dflt_max, string doc = "", int verbosity = 0, string fname
= "", int lineno = 0);
```

Description

Set the "min" and "max" parameters to the values of the named integer-range-type option for the specified object instance and set the "is_set" argument to TRUE if an explicit range was specified. A range option is specified using the syntax +vmm_name=[min:max]. The user can specify a description of the option through *doc*, a verbosity level of the option through *verbosity*. A verbosity value must be within the range 0 to 10. The arguments *fname* and *lineno* are used to track the file name and the line number where the option is specified. These optional arguments are used for providing information to the user through vmm_opts::get_help().

Example

```
class B extends vmm_object;
   int min_val = -1;
   int max_val = -1;
   function new(string name, vmm_object parent=null);
      bit is_set;
      super.new(parent,name);
      vmm_opts::get_object_range(is_set, this,
                                 "FOO", min_val, max_val,
                                 -1,-1, "SET foo value", 0);
   endfunction
endclass
```

Command line: simv +vmm_FOO=[5:10]@%:X:b

vmm_opts::get_object_obj();

Returns the vmm_object instance for the specified hierarchical name

SystemVerilog

```
static function vmm_object vmm_opts::get_object_obj(output bit
is_set, input vmm_object obj, string name, vmm_object dflt = null,
string fname = "", int lineno = 0);
```

Description

Return the named object-type option for the specified object instance and set the "is_set" argument to TRUE if an explicit value was specified. Object-type options can only be set using the vmm_opts::set_object() method.

Example

```
class A extends vmm_object;
   int foo = 11;
   function new( vmm_object parent=null, string name);
      super.new(parent, name);
   endfunction
endclass

class B extends vmm_object;
   A a1, a2;
   function new(vmm_object parent=null, string name);
      bit is_set;
      super.new(parent, name);
      a1 = new(null, "a1");
      a2 = new(null, "a2");
      a2.foo = 22;
      $cast(a1, vmm_opts::get_object_obj(is_set, this,
                                         "OBJ_F1",a2,"SET OBJ", 0));
   endfunction
endclass
```

vmm_opts::set_bit();

Sets the hierarchically named boolen type option.

SystemVerilog

```
static function void vmm_opts::set_bit(string name, bit val,
vmm_unit root = null, string fname = “”, int lineno = 0, string
fname = “”, int lineno = 0);
```

Description

Sets the hierarchically named boolean type option for the specified vmm_object instances. If no vmm_unit root is specified, the hierarchical option name is assumed to be absolute. The argument *name* can be a pattern. When vmm_opts::get_object_bit is called in any object whose hierarchical name matches the pattern, the option is set for that boolean variable.

Example

```
class B extends vmm_object;
   bit foo;
   function new(string name, vmm_object parent=null);
      bit is_set;
      super.new(parent,name);
      foo = vmm_opts::get_object_bit(is_set, this, "FOO",
      "SET foo value", 0);
   endfunction
endclass

B b2;
initial begin
   vmm_opts::set_bit("b2:FOO",null);
   b2 = new("b2", null);
end
```

vmm_opts::set_int();

Sets the hierarchically named integer type option.

SystemVerilog

```
static function void vmm_opts::set_int(string name, int val,
vmm_unit root = null, string fname = "", int lineno = 0, string
fname = "", int lineno = 0);
```

Description

Sets the hierarchically named integer type option for the specified vmm_object instances. If no vmm_unit root is specified, the hierarchical option name is assumed to be absolute. The argument *name* can be a pattern. When vmm_opts::get_object_int is called in any object whose hierarchical name matches the pattern, 'val' is returned for that integer variable.

Example

```
class A extends vmm_object;
   int a_foo;
   function new(vmm_object parent=null, string name);
      bit is_set;
      super.new(parent, name);
      a_foo = vmm_opts::get_object_int(is_set, this,
                                  "A_FOO", 2 , "SET a_foo value", 0);
   endfunction
endclass

class D extends vmm_object;
   A a1;
   ...
endclass

initial  begin
   D d2;
   vmm_opts::set_int("d2:a1:A_FOO", 99,null);
   d2 = new (null, "d2");
end
```

vmm_opts::set_string();

Sets the hierarchical named string type option.

SystemVerilog

```
static function void vmm_opts::set_string(string name, string val,
vmm_unit root = null, string fname = "", int lineno = 0, string
fname = "", int lineno = 0);
```

Description

Sets the hierarchically named string type option for the specified vmm_object instances. If no vmm_unit root is specified, the hierarchical option name is assumed to be absolute. The argument *name* can be a pattern. When vmm_opts::get_object_string is called in any object whose hierarchical name matches the pattern, val is returned for that string variable.

Example

```
class B extends vmm_object;
string foo="ZERO";
      function new(string name, vmm_object parent=null);
      bit is_set;
      super.new(parent,name);
      foo = vmm_opts::get_object_string(is_set, this,
                                        "FOO","DEF_VAL",
                                        "SET foo value", 0);
   endfunction
endclass

initial begin
    B b2;
    vmm_opts::set_string("b2:FOO", "NEW_VAL", null);
    b2 = new("b2", null);
 end
```

vmm_opts::set_range();

Sets the hierarchically named integer range type option.

SystemVerilog

```
static function void vmm_opts::set_range(string name, int min,max,
vmm_unit root = null, string fname = "", int lineno = 0);
```

Description

Sets the hierarchically named integer range type option for the specified vmm_object instances. If no vmm_unit root is specified, the hierarchical option name is assumed to be absolute. The argument *name* can be a pattern.When vmm_opts::get_object_range is called in any object whose hierarchical name matches the pattern, min and max values are returned.

Example

```
class B extends vmm_object;
   int min_val = -1;
   int max_val = -1;
   function new(string name, vmm_object parent=null);
      bit is_set;
      super.new(parent,name);
      vmm_opts::get_object_range(is_set, this,
                                 "FOO",min_val, max_val,
                                 -1,-1, "SET foo value", 0);
   endfunction
endclass

initial begin
   B b2;
   vmm_opts::set_range("b2:FOO", 1, 99, null);
   b2 = new("b2", null);
end
```

vmm_opts::set_object();

Sets the hierarchically named vmm_object type option.

SystemVerilog

```
static function void vmm_opts::set_object(string name, vmm_object
val, vmm_unit root = null, string fname = "", int lineno = 0, string
fname = "", int lineno = 0);
```

Description

Set the hierarchically named type-specific option for the specified vmm_object instance(s). If no vmm_unit root is specified, the hierarchical option name is assumed to be absolute. When called from the vmm_unit::configure_ph() method, the root unit must always be specified as 'this', because vmm_unit instances can only configure lower-level instances during the "configure" phase.The hierarchical option name is specified by prefixing the option name with a hierarchical vmm_unit name and a colon (:).

The hierarchical option name may be specified using a match pattern or a regular expression, except for the last part of the hierarchical name (i.e. the name of the option itself). The hierarchical option name may specify a namespace. An error will be reported if the option value is not eventually used.

Example

```
class A extends vmm_object;
   int foo = 11;
   function new( vmm_object parent=null, string name);
      bit is_set;
      super.new(parent, name);
   endfunction
endclass

class B extends vmm_object;
   A a1;
   A a2;
```

```
   function new(vmm_object parent=null, string name);
      bit is_set;
      super.new(parent, name);
      a1 = new(null, "a1");
      a2 = new(null, "a2");
      a2.foo = 22;
      $cast(a1, vmm_opts::get_object_obj(is_set, this,
                                         "OBJ_F1",a2,"SET OBJ", 0));
   endfunction
endclass

B b2;
A a3;
initial begin
    a3 = new(null, "a3");
    a3.foo = 99;
    vmm_opts::set_object("b2:OBJ_F1", a3,null,,);
    b2 = new(null, "b2");
end
```

RTL Configuration Class (vmm_rtl_config)

This is the base class for RTL configuration and extends vmm_object. This class is for specifying RTL configuration parameters. A different class to other parameters (that use the vmm_opts class) is used because these parameters must be defined at compilation time and may not be modified at run-time.

Example

```
class ahb_master_config extends vmm_rtl_config;
   rand  int addr_width;
   rand  bit mst_enable;
   string    kind = "MSTR";

   constraint cst_mst {
     addr_width == 64;
     mst_enable == 1;
   }
   `vmm_rtl_config_begin(ahb_master_config)
       `vmm_rtl_config_int(addr_width, mst_width)
       `vmm_rtl_config_boolean(mst_enable, mst_enable)
       `vmm_rtl_config_string(kind, kind)
   `vmm_rtl_config_end(ahb_master_config)

   function new(string name = "", vmm_rtl_config parent = null);
      super.new(name, parent);
   endfunction

endclass
```

vmm_rtl_config::map_to_name()

Maps the specified name to the object name.

SystemVerilog

```
function void vmm_rtl_config::map_to_name(string name)
```

Description

Use the specified name for this instance of the configuration descriptor instead of the object name when looking for relevant vmm_rtl_config instances in the RTL configuration hierarchy. The specified name is used as the object name in the “VMM RTL Config” namespace

Example

```
class ahb_master_config extends vmm_rtl_config;
   function new(string name = "", vmm_rtl_config parent = null);
      super.new(name, parent);
   endfunction
endclass

class env_config extends vmm_rtl_config;
   rand ahb_master_config mst_cfg;
   function void build_config_ph();
      mst_cfg = new("mst_cfg", this);
   endfunction
endclass

initial begin
  env_config env_cfg = new("env_cfg");
  env_cfg.mst_cfg.map_to_name("env:mst");
end
```

vmm_rtl_config_macros

`vmm_rtl_config_begin(classname)

`vmm_rtl_config_boolean(name, fname)

`vmm_rtl_config_int(name, fname)

`vmm_rtl_config_string(name, fname)

`vmm_rtl_config_obj(name)

`vmm_rtl_config_end(classname)

Macros for accessing RTL configuration parameters with default implementation.

Description

Type-specific, shorthand macros providing a default implementation for the setting, randomization and saving of RTL parameter members. The “name” is the name of the member in the class. The “fname” is the name of the RTL configuration parameter in the RTL configuration file.

Example

```
class ahb_master_config extends vmm_rtl_config;
   rand  int addr_width;
   rand  bit mst_enable;
   string    kind = "MSTR";
   `vmm_rtl_config_begin(ahb_master_config)
      `vmm_rtl_config_int(addr_width, mst_width)
      `vmm_rtl_config_boolean(mst_enable, mst_enable)
      `vmm_rtl_config_string(kind, kind)
   `vmm_rtl_config_end(ahb_master_config)
endclass
```

vmm_rtl_config::build_config_ph()

Builds RTL configuration parameters.

SystemVerilog

```
virtual function void vmm_rtl_config::build_config_ph()
```

Description

Build the structure of RTL configuration parameters for hierarchical RTL designs.

Example

```
class env_config extends vmm_rtl_config;
   rand ahb_master_config mst_cfg;
   rand ahb_slave_config  slv_cfg;
   ...
   function void build_config_ph();
      mst_cfg = new("mst_cfg", this);
      slv_cfg = new("slv_cfg", this);
   endfunction
   ...
endclass
```

vmm_rtl_config::get_config_ph()

Sets the RTL configuration parameters

SystemVerilog

```
virtual function void vmm_rtl_config::get_config_ph()
```

Description

Read a configuration file and set the current value of the members to the corresponding RTL configuration parameters. The filename may be computed using the value of the +vmm_rtl_config option using the vmm_opts::get_string("rtl_config") method and the hierarchical name of this vmm_object instance.

A default implementation of this method is created if the `vmm_rtl_config_* () short-hand macros are used.

vmm_rtl_config::save_config_ph()

Saves the RTL configuration parameters in a file.

SystemVerilog

```
virtual function void vmm_rtl_config::save_config_ph()
```

Description

Create a configuration file that specifies the RTL configuration parameters corresponding to the current value of the class members. The filename may be computed using the value of the +vmm_rtl_config option using the vmm_opts::get_string("rtl_config") method and the hierarchical name of this vmm_object instance.

A default implementation of this method is created if the `vmm_rtl_config_* () short-hand macros are used.

vmm_rtl_config::get_ config();

Returns vmm_rtl_config object for the specified vmm_object.

SystemVerilog

```
static function vmm_rtl_config::get_ config(vmm_object obj, string
fname = "", int lineno = 0)
```

Description

Get the instance of the specified class extended from the vmm_rtl_config class whose hierarchical name in the "VMM RTL Config" namespace is identical to the hierarchical name of the specified object. This allows a component to retrieve its instance-configuration without having to know where it is located in the testbench hierarchy.

Example

```
class ahb_master extends vmm_unit;
   ahb_master_config cfg;
   function void configure_ph();
      $cast(cfg, vmm_rtl_config::get_config(this,
                                            `__FILE__, `__LINE__));
   endfunction
endclass
```

vmm_rtl_config::default_file_fmt

Default RTL configuration file format.

SystemVerilog

```
static vmm_rtl_config_file_format vmm_rtl_config::default_file_fmt
```

Description

Default RTL configuration file format writer/parser. Used if vmm_rtl_config::file_fmt is null.

Example

```
class def_rtl_config_file_format extends
   vmm_rtl_config_file_format;
endclass

intial begin
   def_rtl_config_file_format dflt_fmt = new();
   vmm_rtl_config::default_file_fmt = dflt_fmt;
end
```

vmm_rtl_config::file_fmt

RTL configuration file format.

SystemVerilog

```
protected vmm_rtl_config_file_format vmm_rtl_config::file_fmt
```

Description

The RTL configuration file format writer/parser for this instance.

Example

```
//protected vmm_rtl_config_file_format vmm_rtl_config :: file_fmt
class ahb_rtl_config_file_format extends
   vmm_rtl_config_file_format;
endclass

class env_config extends vmm_rtl_config;
   rand ahb_master_config mst_cfg;
   ahb_rtl_config_file_format ahb_file_fmt;

   function void build_config_ph();
      mst_cfg = new("mst_cfg", this);
      ahb_file_fmt = new;
      mst_cfg.file_fmt = ahb_file_format;
   endfunction
endclass
```

RTL Configuration File format class

Base class for RTL configuration file format.

SystemVerilog

```
virtual class vmm_rtl_config_file_format
```

Description

This is the base class for RTL configuration file writer/parser. May be used to simplify the task of implementing the vmm_rtl_config::get_config_ph() and vmm_rtl_config::save_config_ph() methods.

Example

```
class rtl_config_file_format extends    vmm_rtl_config_file_format;
   virtual function bit fopen(vmm_rtl_config cfg,
                              string mode,
                              string fname = "",
                              int lineno = 0);
       string filename = {cfg.prefix, ":",
       cfg.get_object_hiername(), ".rtl_conf"};
       vmm_rtl_config::file_ptr = $fopen(filename, mode);
       if (vmm_rtl_config::file_ptr == 0) return 0;
       else return 1;
   endfunction

   function string get_val(string str);
       if (`vmm_str_match(str, "     : ")) begin
          string fname = `vmm_str_prematch(str);
          string fval = `vmm_str_postmatch(str);
          if (`vmm_str_match(fval, ";")) begin
            fval = `vmm_str_prematch(fval);
          end
          return fval;
       end
   endfunction

   virtual function bit read_int(string name,
                                 output int value);
```

```
      int r;
      string str;
      $display("Calling read_int for %s", name);
      r = $freadstr(str, vmm_rtl_config::file_ptr);
      str = get_val(str);
      value = str.atoi();
      $display("Got %0d for %s", value, name);
      return (r != 0);
   endfunction

   virtual function bit write_int(string name, int value);
      $fwrite(vmm_rtl_config::file_ptr, "%s     : %0d;\n",
              name, value);
      return 1;
   endfunction

   virtual function void fclose();
      $fclose(vmm_rtl_config::file_ptr);
  endfunction

endclass
```

vmm_rtl_config_file_format::fname();

Compute the filename that contains the RTL configuration parameter for the specified instance of the RTL configuration descriptor.

SystemVerilog

```
virtual protected function string vmm_rtl_config_file_format
::fname(vmm_rtl_config cfg)
```

Description

Compute the filename that contains the RTL configuration parameter for the specified instance of the RTL configuration descriptor. By default, concatenates the value of the +vmm_rtl_config option and the hierarchical name of specified RTL configuration descriptor, separating each with a “/” and appending a “.cfg” suffix.

Example

```
TBD
```

vmm_rtl_config_file_format::fopen();

Open RTL config file.

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::fopen(vmm_rtl_config cfg, string mode, string fname = “”, int
lineno = 0)
```

Description

Open the configuration file corresponding to the specified RTL configuration descriptor in the specified mode (“r” or “w”). The filename may be computed using the value of the +vmm_rtl_config option using the vmm_opts::get_string(“rtl_config”) method and the name of specified RTL configuration descriptor. Returns TRUE if the file was successfully opened. If the file is open for read, it may be immediately parsed and its content internally cached.

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit fopen(vmm_rtl_config cfg,
                              string mode,
                              string fname = "",
                              int lineno = 0);
      string filename = {cfg.prefix, ":",
      cfg.get_object_hiername(), ".rtl_conf"};
      vmm_rtl_config::file_ptr = $fopen(filename, mode);
      if (vmm_rtl_config::file_ptr == 0) return 0;
      else return 1;
   endfunction
...
Endclass
```

vmm_rtl_config_file_format::get_fname()

Return the name of the configuration file currently opened. Returns “” if no file is open.

SystemVerilog

```
pure virtual function string vmm_rtl_config_file_format
::get_fname()
```

Description

Return the name of the configuration file currently opened. Returns “” if no file is open.

Example

```
TBD
```

vmm_rtl_config_file_format::read_bit()

Reads a boolean variable from the RTL configuration file

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::read_bit(string name, output bit value)
```

Description

Returns a boolean value with the specified name from the RTL configuration file

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit read_bit(string name, output bit value);
      int r;
      string str;
      r = $freadstr(str, vmm_rtl_config::file_ptr);
      str = get_val(str);
      value = str.atoi();
      $display("Got %b for %s", value, name);
      return (r != 0);
   endfunction
   ...
endclass
```

vmm_rtl_config_file_format::read_int()

Reads an integer variable from the RTL configuration file.

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::read_int(string name, output int value)
```

Description

Returns an integer value with the specified name from the RTL configuration file

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit read_int(string name, output int value);
     int r;
     string str;
     $display("Calling read_int for %s", name);
     r = $freadstr(str, vmm_rtl_config::file_ptr);
     str = get_val(str);
     value = str.atoi();
     $display("Got %0d for %s", value, name);
     return (r != 0);
  endfunction
  ...
endclass
```

vmm_rtl_config_file_format::read_string();

Reads a string value from the RTL config file.

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::read_string(string name, output string value)
```

Set the “value” argument to the value of the named RTL configuration parameter as specified in the file. Returns TRUE if a value for the parameter was found in the file. Returns FALSE otherwise. An implementation may require that the parameters be read in the same order as they are found in the file.

Description

Returns string value with the specified name from the RTL configuration file

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit read_string(string name, output string
value);
      int r;
      string str;
      $display("Calling read_string for %s", name);
      r = $freadstr(str, vmm_rtl_config::file_ptr);
      value = get_val(str);
      $display("Got %s for %s", value, name);
      return (r != 0);
   endfunction
   ...
endclass
```

vmm_rtl_config_file_format::write_bit()

Writes a boolen name and value to the RTL config file.

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::write_bit(string name, bit value)
```

Description

Writes a name and boolean value to the RTL configuration file. Returns TRUE if the parameter was not previously written. Returns FALSE otherwise. An implementation may physically write the parameter values in the file in a different order than if they were written using these methods.

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit write_bit(string name, bit value);
      $fwrite(vmm_rtl_config::file_ptr, "%s     : %b;\n",
              name, value);
      return 1;
   endfunction
   ...
endclass
```

vmm_rtl_config_file_format::write_int()

Writes an integer name and value to the RTL config file.

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::write_int(string name, int value)
```

Description

Writes the name and integer value in the RTL configuration file. Returns TRUE if the parameter was not previously written. Returns FALSE otherwise. An implementation may physically write the parameter values in the file in a different order than if they were written using these methods.

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit write_int(string name, int value);
      $fwrite(vmm_rtl_config::file_ptr, "%s : %0d;\n", name, value);
      return 1;
   endfunction
   ...
endclass
```

vmm_rtl_config_file_format::write_string()

SystemVerilog

```
pure virtual function bit vmm_rtl_config_file_format
::write_string(string name, string value)
```

Description

Write the specified value for the named RTL configuration parameter. Returns TRUE if the parameter was not previously written. Returns FALSE otherwise. An implementation may physically write the parameter values in the file in a different order than if they were written using these methods.

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function bit write_string(string name, string value);
      $fwrite(vmm_rtl_config::file_ptr, "%s     : %s;\n",
              name, value);
      return 1;
   endfunction
   ...
endclass
```

vmm_rtl_config_file_format ::fclose()

Close RTL configuration file

SystemVerilog

```
pure virtual function void vmm_rtl_config_file_format ::fclose()
```

Description

Close the configuration file that was previously opened. An implementation may choose to internally cache the information written to the file using the write_*() methods and physically write the file just before closing it.

Example

```
class rtl_config_file_format extends vmm_rtl_config_file_format;
   ...
   virtual function void fclose();
      $fclose(vmm_rtl_config::file_ptr);
   endfunction
   ...
endclass
```

vmm_rtl_config_DW_format

Pre-defined implementation for an RTL configuration parameter using the DesignWare Implementation IP file format.

SystemVerilog

```
class vmm_rtl_config_DW_format extends vmm_rtl_config_file_format
```

Miscellaneous

Miscellaneous minor additions and improvements to VMM 1.1 classes.

vmm_data additions

vmm_data::filename , vmm_data:lineno

Properties added to the vmm_data class.

SystemVerilog

```
int vmm_data:lineno = 0
string vmm_data::filename = “”
```

Description

These should be automatically set by the create_instance() method and the pre-defined generators. Their content must be copied in the vmm_data::copy_data() method. If set to non-default values, their content should be displayed in the vmm_data::psdisplay() method.

Parameterized Atomic generator class

Parameterized version of the VMM atomic generator.

SystemVerilog

```
class vmm_atomic_gen #(type T) extends vmm_xactor
```

Description

The `vmm_atomic_generator macro will now create a parameterized atomic generator. This generator can now generate non vmm_data transactions as well.

Example

```
class ahb_trans extends vmm_data;
   rand bit [31:0] addr;
   rand bit [31:0] data;
endclass

`vmm_channel(ahb_trans)
`vmm_atomic_gen(ahb_trans, "AHB Atomic Gen")
ahb_trans_channel      chan0 = new("ahb_trans_chan", "chan0");
ahb_trans_atomic_gen   gen0  = new("AhbGen0", 0, chan0);
```

Is same as:

```
vmm_channel_typed #(ahb_trans) chan0 = new("ahbchan", "chan0");
vmm_atomic_gen #(ahb_trans) gen0  = new("AhbGen0", 0, chan0);
```

Parametrized scenario Generator

Parameterized version of the VMM scenario generator.

SystemVerilog

```
class vmm_scenario_gen #(type T) extends vmm_xactor
```

Description

The `vmm_scneario_generator macro will now create a parameterized scenario generator. This generator can now generate non vmm_data transactions as well.

Example

```
class ahb_trans extends vmm_data;
   rand bit [31:0] addr;
   rand bit [31:0] data;
endclass

`vmm_channel(ahb_trans)
`vmm_scenario_gen(ahb_trans, "AHB Scenario Gen")

ahb_trans_channel      chan0 = new("ahb_trans_chan", "chan0");
ahb_trans_scenario_gen gen1  = new("AhbGen1", 0, chan0);
```

Is same as:

```
vmm_channel_typed#(ahb_trans) chan0 = new("ahb_trans_chan",
"chan0");
vmm_scenario_gen #(ahb_trans) gen1  = new("AhbGen1", 0, chan0);
```

Parametrized Single stream scenario class

Parameterized version of the VMM single stream scenario.

SystemVerilog

```
class vmm_ss_scenario #(type T) extends vmm_scenario
```

Description

Parameterized single stream scenario is used by the parameterized scenario generator. It extends vmm_scenario. User can extend this class to create his scenario.

Example

```
class ahb_trans extends vmm_data;
   rand bit [31:0] addr;
   rand bit [31:0] data;
endclass

 `vmm_channel(ahb_trans)
 `vmm_scenario_gen(ahb_trans, "AHB Scenario Gen")

 class user_scenario extends ahb_trans_scenario;
 endclass
```

Is same as:

```
class user_scenario extends vmm_ss_scenario#(ahb_trans);
endclass
```

Parametrized atomic scenario class

Parameterized version of the VMM atomic scenario.

SystemVerilog

```
class vmm_atomic_scenario #(T) extends vmm_ss_scenario#(T)
```

Description

Parameterized atomic scenario is a generic typed scenario extending vmm_ss_scenario. It is used by the parameterized scenario generator as the default scenario.

Example

```
class ahb_trans extends vmm_data;
     rand bit [31:0] addr;
     rand bit [31:0] data;
endclass

`vmm_channel(ahb_trans)
`vmm_scenario_gen(ahb_trans, "AHB Scenario Gen")

class vmm_atomic_scenario#(ahb_trans) extends
vmm_ss_scenario#(ahb_trans);

endclass
```

Parametrized Notify Observer

Class to simplify subscription to a notification callback method.

SystemVerilog

```
class vmm_notify_observer #(type T, type D = vmm_data) extends
vmm_notify_callback
```

Description

Notify observer is a parameterized extension of vmm_notify_callbacks. Any subscriber (like scoreboard, coverage model, etc) can get the transaction status whenever notification event is indicated. `vmm_notify_observer macro is provided to specify the observer and its method name to be called.

Example

```
class scoreboard;
   virtual function void observe_trans(ahb_trans tr);
    ……
   endfunction
endclass
`vmm_notify_observer(scoreboard, observe_trans)
```

Instantiate parameterized vmm_notify_observer passing subscriber handle, vmm_notify handle and the notification id.

```
scoreboard       sb = new();
vmm_notify_observer#(scoreboard, ahb_trans)  observe_start
                        = new(sb, mon.notify, mon.TRANS_START);
```

vmm_notify_observer::new ()

Append a callback method to invoke the *T::observe(D.status)* method in the specified instance,

SystemVerilog

```
function vmm_notify_observer::new(T observer, vmm_notify ntfy, int
notification_id)
```

Description

Append a callback method to invoke the ***T::observe(D)*** method in the specified instance,whenever the specified indication is notified on the specified Notification Service Interface.

Example

```
vmm_notify_observer#(scoreboard, ahb_trans)  observe_start
                        = new(sb, mon.notify, mon.TRANS_START);
```

`vmm_notify_observer

Define a parameterized class in the style of the *vmm_notify_observer* class.

SystemVerilog

```
`define vmm_notify_observer(classname, methodname)
```

Description

Define a parameterized class in the style of the *vmm_notify_observer* class, with the specified name and calling the specified *T::methodname(D.status)* method. Useful for defining a subscription class for an observer with a different observation method.Example

```
class scoreboard;
   virtual function void observe_trans(ahb_trans tr);
    ……
   endfunction
endclass
`vmm_notify_observer(scoreboard, observe_trans)
```

vmm_connect class

Utility class for connecting channels and notifications in the *vmm_unit::connect_ph()* method.

SystemVerilog

```
class vmm_connect #(type T, type D = vmm_data)
```

Description

vmm_connect utility class can be used for connecting channels and notifications in the vmm_unit::connect_ph() method. It does additional check on whether the channels are already connected.

vmm_connect::channel()

Connect the specified channel ports.

SystemVerilog

```
class vmm_connect#(T)::channel(ref T upstream, downstream, string
name = "", vmm_object parent = null);
```

Description

Connect the specified channel ports. If both specified channels are not ***null***, they are connected using the ***upstream.connect(downstream)*** statement. Otherwise, both channels will be connected by referring to the same channel instance. It is an error to attempt to connect two channels that are already connected together or to another channel.

Example

```
class ahb_unit extends vmm_unit;
   ahb_trans_channel gen_chan;
   ahb_trans_channel drv_chan;

   virtual function void build_ph();
      drv_chan = new("ahb_chan", "drv_chan");
      gen_chan = new("ahb_chan", "gen_chan");
   endfunction

   virtual function void connect_ph();
      vmm_connect#(ahb_trans_channel)::connect(gen_chan,
                                               drv_chan,
                                               "gen2drv",
                                               this);
   endfunction

endclass
```

vmm_ connect::notify()

Connect the specified observer to the specification notification.

SystemVerilog

```
class vmm_connect#(T, D)::notify(T observer, vmm_notify ntfy, int
notification_id);
```

Description

Connect the specified observer to the specification notification using an instance of the ***vmm_notify_observer*** class.

Example

```
class scoreboard;
   virtual function void observe_trans(ahb_trans tr);
     ……
   endfunction
endclass
`vmm_notify_observer(scoreboard, observe_trans)

class ahb_unit extends vmm_unit;
   scoreboard sb;

   virtual function void build_ph();
     sb = new();
   endfunction

   virtual function void connect_ph();
    vmm_connect#(scoreboard, ahb_trans)::notify(sb,
                                                mon.notify,
                                                mon.TRANS_STARTED);
   endfunction

endclass
```

TLM Base classes

Base classes aimed at modeling transactors, transport and analysis ports.

Comply with TLM 2.0 OSCI standard.

class vmm_tlm

This class contains the sync_e enum for various phases of the transaction. All TLM port classes use this enum value as the default template for defining the phases of the transaction.

SystemVerilog

```
class vmm_tlm;
   typedef enum { TLM_REFUSED, TLM_ACCEPTED,
      TLM_UPDATED, TLM_COMPLETED } sync_e;
   typedef enum { TLM_BLOCKING_PORT, TLM_BLOCKING_EXPORT,
      TLM_NONBLOCKING_FW_PORT, TLM_NONBLOCKING_FW_EXPORT,
      TLM_NONBLOCKING_PORT,TLM_NONBLOCKING_EXPORT,
      TLM_ANALYSIS_PORT,TLM_ANALYSIS_EXPORT} intf_e;
   sync_e sync;
endclass
```

Description

This class provides enumerated type sync_e which has the various phases of a transaction object. These phases can be updated by different components accessing the same transaction object.

The enumerated type intf_e is used to connect the vmm_channel_typed to TLM transport ports, TLM transport exports and TLM analysis ports and exports.

The vmm_tlm class also provides static methods to print, check and report the bindings of all TLM ports and exports under a specified root.

function vmm_tlm::print_bindings

Static method used to print the bindings of all TLM ports and exports instantiated under a specified root.

SystemVerilog

```
static function print_bindings(vmm_object root= null);
```

Description

Prints the bindings of all TLM ports and exports including transport ports and exports, sockets and analysis ports, and exports instantiated under the vmm_object specified by the root argument. If null is passed, then the bindings are printed for all TLM ports and exports in the environment.

The print_bindings method is also available with all TLM ports and exports and can be invoked for the particular port object.

Example

```
class my_env extends vmm_env;
   function build();
      …
      vmm_tlm::print_bindings(this);
   endfunction
endclass
```

function vmm_tlm::check_bindings

Static method to check if minimum bindings exist for all TLM ports and exports under the specified root.

SystemVerilog

```
static function check_bindings(vmm_object root= null);
```

Description

A warning is issued if a port is unbound or if an export has less than the minimum bindings specified for the export. Analysis port bindings are reported with debug severity. If root is not specified, then the binding checks are done for all TLM ports and exports in the environment.

The check_bindings method is also available with all TLM ports and exports and can be invoked for the particular port object.

Example

```
class my_env extends vmm_env;
   function build();
      …
      vmm_tlm::check_bindings(this);
   endfunction
endclass
```

function vmm_tlm::report_unbound

Static method to report all unbound TLM ports and to export instances available under a specified root.

SystemVerilog

```
static function report_bindings(vmm_object root= null);
```

Description

Reports all unbound TLM ports and exports, including transport ports and exports, sockets and analysis ports, and exports instantiated under the vmm_object specified by the root argument. If null is passed, then the bindings are printed for all TLM ports and exports in the environment.

A warning is issued if any TLM port or export under the specified root is left unbound. For analysis ports a message with debug severity is issued.

The report_unbound method is also available with all TLM ports and exports and can be invoked for the particular port object.

Example

```
class my_env extends vmm_env;
   function build();
      …
      vmm_tlm::report_unbound(this);
   endfunction
endclass
```

class vmm_tlm_port_base#(D,P)

This is an abstract base class for all TLM2.0 transport ports

SystemVerilog

```
virtual class vmm_tlm_port_base#(type D=vmm_data, type P =
vmm_tlm) extends vmm_tlm_base;
```

Description

This is an abstract base class for all TLM2.0 transport ports. This class has the methods that are required by all TLM2.0 transport port implementations. Any user defined port must be extended from this base class.

Argument D is the type of the transaction object the port services. The default type is vmm_data. Argument P is the type of the phasing class. The default value is vmm_tlm.

function vmm_tlm_port_base::new

Constructor of port base class

SystemVerilog

```
function new(vmm_object parent, string name);
```

Description

Set the parent if the base class extends vmm_object. Set the name of the instance.

function vmm_tlm_port_base::tlm_bind

Binds the TLM port to the TLM export passed as an argument.

SystemVerilog

```
function void tlm_bind(vmm_tlm_export_base#(D,P) peer, int id = -1);
```

Description

Binds the TLM port to the TLM export. A port can have only one binding, though multiple bindings are allowed for exports. It is an error to bind a port that already has a binding.

This method adds the current port descriptor to the bindings list of *peer*. Calling port.tlm_bind(export,id) is equivalent to export.tlm_bind(port,id) and the binding can be done either way. It will be an error if both calls are made as the port allows only one binding.

The second argument, *id*, is used to distinguish between multiple ports that bind to the same export. The *id* field is used by the export. If a positive *id* is supplied then it must be unique for that export. It is an error if a positive *id* already used by the export is supplied. If no *id* or a negative *id* is given, then the lowest available unique positive *id* is automatically assigned. This *id* is passed as an argument of the transport method implemented in the exports parent.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_b_transport_port#(producer) b_port =
      new(this,”producer port”);
endclass

class consumer extends vmm_xactor;
    vmm_tlm_b_transport_export#(consumer) b_export =
       new(this,”consumer export”);
endclass

class my_env extends vmm_env;
   producer p[4];
```

```
   consumer c;

   function build();
      consumer c = new();
      foreach(p[i]) begin
          p[i] = new();
          p[i].b_port.tlm_bind(c.b_export, i);
      end
   endfunction
endclass
```

function vmm_tlm_port_base::tlm_unbind

Remove the existing port binding.

SystemVerilog

```
function void tlm_unbind();
```

Description

Set the port binding to null. Also removes the current port descriptors binding from the export that the port is bound to. A warning is issued if a binding does not exist for this port.

This method can be used to dynamically change existing bindings for a port.

Example

```
class my_env extends vmm_env;
   producer p1;
   consumer c1, c2;

   function build();
      p1.b_port.tlm_bind(c1.b_export);
   endfunction
endclass

program test();
   my_env env = new();
   initial begin
      env.build();
      env.run();
      #5000;
      env.p1.b_port.tlm_unbind();
      env.p1.b_port.tlm_bind(c2.b_export);
      env.wait_for_end();
   end
endprogram
```

function vmm_tlm_port_base::get_peer

Get the binding for the port.

SystemVerilog

```
function vmm_tlm_export_base#(D,P) get_peer();
```

Description

Returns the export bound to the current port. Null is returned if the port does not have a binding.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_b_transport_port#(producer) b_port;

   function display_my_id();
      vmm_tlm_export_base peer;
      peer = this.b_port.get_peer();
      $display("My id = %d",peer.get_peer_id(this);
   endfunction
endclass
```

function vmm_tlm_port_base::get_peer_id()

Get the id of this port for its binding

SystemVerilog

```
function int get_peer_id();
```

Description

Gets the id of this port with respect to its export binding. If port is not bound -1 is returned.

Example

```
class my_env extends vmm_env;
   producer p1,p2;
   consumer c1;

   function build();
      p1.b_port.tlm_bind(c1.b_export);
      p2.b_port.tlm_bind(c1.b_export);
      int p1_id = p1.b_port.get_peer_id(); //returns 0
      int p2_id = p2.b_port.get_peer_id(); //returns 1
   endfunction
endclass
```

function vmm_tlm_port_base::tlm_import

Import a port from an inner level in the hierarchy to an outer level.

SystemVerilog

```
function void tlm_import(vmm_tlm_port_base#(D,P) peer);
```

Description

This is a special port to port binding. It simplifies the binding for hierarchical ports and exports by making the inner port visible to the outer hierarchy. The binding finally resolves to a port-export binding.

The method allows only parent-child ports to be imported. An error is issued if the ports do not share a parent-child relationship. It is an error to import a port that has already been imported. It is an error to import a port that has already been bound.

The method can be called for both parent to child binding and child to parent binding. For this the parent xactors must be derivatives of vmm_object. If the parent is a vmm_xactor extension then the vmm_xactor base class should be underpinned. If the vmm_xactor is not underpinned or the parent is not a derivative of vmm_object then only child.port.tlm_import(parent.port) is permitted. The error checks are not executed and the user must ensure legal connections.

Example

```
class initiator_child extends vmm_xactor;
  vmm_tlm_b_transport_port b_port;
endclass

class initiator_parent extends vmm_subenv;
   vmm_tlm_b_transport_port b_port;
   initiator_child initiator;
   function new();
      initiator = new();
      b_port = new();
      initiator.b_port.tlm_import(this.b_port);
```

```
   endfunction
endclass

class target extends vmm_xactor;
   vmm_tlm_b_transport_export b_export;
endclass

class my_env extends vmm_env;
   initiator_parent initiator;
   target           target;
   function build();
      initiator.b_port.tlm_bind(target.b_export);
   endfunction
endclass
```

class vmm_tlm_export_base #(D,P)

This is an abstract base class for all TLM2.0 transport exports. This class has the methods that are required by all TLM2.0 transport export implementations. Any user-defined export must be extended from this base class.

The argument D is the type of the transaction object the export services. The default type is vmm_data. The argument P is the type of the phasing class. The default value is vmm_tlm.

SystemVerilog

```
virtual class vmm_tlm_export_base #(type D = vmm_data, type P =
vmm_tlm) extends vmm_object;
```

Description

Set the parent if it's an extension of vmm_object. Set the name of the instance.

function vmm_tlm_export_base::new

Constructor of export base class.

SystemVerilog

```
function new(vmm_object parent, string name, int max_binds = 1, int
min_binds = 0);
```

Description

Set the parent if it's an extension of vmm_object. Set the name of the instance. Set the maximum and minimum bindings allowed for this export.

function vmm_tlm_export_base::tlm_bind

Bind the TLM export to the TLM port passed as an argument.

SystemVerilog

```
function void tlm_bind(vmm_tlm_port_base#(D,P) peer, int id = -1);
```

Description

Bind the TLM export to the supplied port. Multiple bindings are allowed for exports.

This method adds the supplied port descriptor to the bindings list of the export. An error is issued if the supplied port already has a binding.

The second argument, id, is used to distinguish between multiple ports that bind to the same export. If a positive id is supplied then it must be unique for this export. It is an error if a positive id already used by the export is supplied. If no id or a negative id is given then the lowest available positive id is automatically assigned. This id is passed as an argument of the transport method implemented in the exports parent.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_b_transport_port#(producer) b_port =
      new(this,"producer port");
endclass

class consumer extends vmm_xactor;
   vmm_tlm_b_transport_export#(consumer) b_export =
      new(this,"consumer export");
   function b_transport(int id, vmm_data trans,
                        vmm_tlm phase , int delay);
   if(id == 0 )
      $display("From producer 0");
   else if (id == 1)
   …
   endfunction
endclass
```

```
class my_env extends vmm_env;
   producer p[4];
   consumer c;

   function build();
      consumer c = new();
      foreach(p[i]) begin
         p[i] = new();
         c.b_export.tlm_bind(p[i].b_port, i);
      end
   endfunction
endclass
```

function vmm_tlm_export_base::tlm_unbind

Remove an existing binding of the export.

SystemVerilog

```
function void tlm_unbind(vmm_tlm_port_base#(D,P) peer = null, int id
= -1);
```

Description

Remove the binding supplied as a peer or id from the list of bindings for this export. Also removes the binding of this export with the connected port.

If the supplied peer is not null then the binding of the peer is removed. An error is issued if the supplied peer is not bound to this export.

If the supplied peer is null and the supplied id is a positive number then the binding to the port with the supplied id is removed. An error is issued if there is no binding with the supplied positive id.

If the supplied peer is null and the supplied id negative then all bindings for this export are removed.

On unbinding, the id of the unbound port becomes available for reuse.

Example

```
class my_env extends vmm_env;
   producer p1;
   consumer c1, c2;

   function build();
      p1.b_port.tlm_bind(c1.b_export);
   endfunction

program test();
   my_env env = new();
   initial begin
      env.build();
```

```
      env.run();
      #5000;
      env.c1.b_export.tlm_unbind(p1.b_port);
      env.c2.b_export.tlm_bind(p1.b_port);
      env.wait_for_end();
   end
endprogram
```

function vmm_vmm_tlm_port_base::get_peer

Get the binding for the port.

SystemVerilog

```
function vmm_tlm_port_base#(D,P) get_peer(int id = -1);
```

Description

Returns the port bound to the current export with the specified id. Null is returned if the port does not have a binding with the specified id. If only one binding exists for the export then the handle to be binding is returned without considering the id value passed.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_b_transport_export#(consumer) b_export;
   function display_my_id();
      vmm_tlm_port_base peer;
      peer = this.b_export.get_peer(0);
      $display("My id = %d",peer.get_peer_id();
   endfunction
endclass
```

function vmm_tlm_port_base::get_peer_id()

Get the id of this port for its binding

SystemVerilog

```
function int get_peer_id(vmm_tlm_port_base#(D,P) peer);
```

Description

Gets the binding id of the specified port bound to this export. If the specified port is not bound to this export, -1 is returned.

Example

```
class my_env extends vmm_env;
   producer p1,p2;
   consumer c1;

   function build();
      p1.b_port.tlm_bind(c1.b_export);
      p2.b_port.tlm_bind(c1.b_export);
      int p1_id = c1.b_export.get_peer_id(p1.b_port);
      int p2_id = c1.b_export.get_peer_id(p2.b_port);
   endfunction
endclass
```

function vmm_tlm_export_base::get_n_peers

Return the number of export bindings.

SystemVerilog

```
function int get_n_peers();
```

Description

Returns the number of port export bindings as set with the tlm_bind() method.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_b_transport_export#(consumer) b_export;
   function display_n_connections();
      $display("Export has %d bindings",
               this.b_export.get_n_peers());
   endfunction
endclass
```

function vmm_tlm_export_base::get_peers

Get the list of all bindings of the export.

SystemVerilog

```
function void get_peers(vmm_tlm_port_base#(D,P) peers[$]);
```

Description

Return the queue of bindings of the export in the specified queue.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_b_transport_export#(consumer) b_export;
   function display_connections();
      vmm_tlm_port_base q[$];
      b_export.get_peers(q);
      foreach(q[i])
      $display("Binding[%0d] %s",i, q[i].get_object_name() );
   endfunction
endclass
```

function vmm_tlm_export_base::tlm_import

Import an export from an inner level in the hierarchy to an outer level

SystemVerilog

```
function void tlm_import(vmm_tlm_export_base#(D,P) peer);
```

Description

This is a special way of exporting bindings. It simplifies the binding for hierarchical exports by making the inner export visible to the outer hierarchy. The binding finally resolves to a port-export binding. The method allows only parent-child exports to be imported. An error is issued if the exports do not share a parent-child relationship. It is an error to import an export that has already been imported. It is an error to import an export that has already been bound. The method can be called for both parent to child bindings and child to parent bindings. For this the parent xactors must be derivatives of vmm_object. If the parent is a vmm_xactor extension, then the vmm_xactor base class should be underpinned. If the vmm_xactor is not underpinned, or the parent is not a derivative of vmm_object, then only child.export.tlm_import(parent.export) is permitted. The error checks are not executed and the user must ensure legal connections.

Example

```
class target_child extends vmm_xactor;
  vmm_tlm_b_transport_export b_export;
endclass

class target_parent extends vmm_subenv;
   vmm_tlm_b_transport_export b_export;
   target_child target;
   function new();
      target = new();
      b_export = new();
      target.b_export.tlm_import(this.b_export);
   endfunction
endclass
```

class vmm_tlm_b_transport_port#(T,D,P)

Base class for modeling blocking transport port.

SystemVerilog

```
class vmm_tlm_b_transport_port #(type T = vmm_xactor, type D =
vmm_data, type P = vmm_tlm) extends vmm_tlm_port_base#(D,P);
```

Description

Class providing the blocking transport port. The parameter type T is the class instantiating the transport port. This defaults to vmm_xactor. The parameter D is the data type of the transaction the port services. The default is vmm_data. The parameter type P is the phase class for this port. The default type is vmm_tlm.

The port can be bound to one export. A warning is issued if the port is left unbound.

There is no backward path for the blocking transport.

function vmm_tlm_b_transport_port::new

Constructor for blocking transport port class.

SystemVerilog

```
function new(vmm_object parent, string name);
```

Description

Sets the parent and instance name of the blocking transport port.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_b_transport_port#(producer) b_port;
   function new();
      this.b_port = new(this,”producer port”);
   endfunction
endclass
```

task vmm_tlm_b_transport_port::b_transport

TLM task for blocking transport.

SystemVerilog

```
task b_transport(D trans, P ph = null, int delay = -1);
```

Description

TLM task for blocking transport. Invoke the b_transport method of the bounded export. The 1st argument is a handle of the transaction object, the 2nd argument is a handle to the phase class. Argument delay is the timing annotation.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_b_transport_port#(producer) b_port;
   task main();
      my_data tr;
      while(1) begin
         tr = new();
         this.b_port.b_tranport(tr);
         $display("Transaction Completed");
      end
   endtask
endclass
```

class vmm_tlm_b_transport_export#(T,D,P)

Blocking transport export class.

Any class instanatiating this blocking transport export must provide an implementation of the b_transport task.

SystemVerilog

```
class vmm_tlm_b_transport_export#(type T = vmm_xactor, type D =
vmm_data , type P = vmm_tlm) extends vmm_tlm_export_base#(D,P);
```

Description

Class providing the blocking transport export. The parameter type T is the class instantiating the transport export. This defaults to vmm_xactor. The parameter D is the data type of the transaction the export services. The default is vmm_data. The parameter type P is the phase class for this export. The default type is vmm_tlm.

The export can be bound to multiple ports upto the maximum bindings specified in the constructor of this class.

function vmm_tlm_b_transport_export::new

Constructor of blocking transport export class.

SystemVerilog

```
function new(T parent, string name, int max_binds = 1 , int
min_binds = 0);
```

Description

Set the parent and instance name of the blocking transport export. Sets the maximum and minimum bindings allowed for this export. The default value of maximum bindings is 1 and the minimum binding is 0. An error is issued during tlm_bind if the current binding exceeds the maximum allowed bindings for the export. An error is issued during elaboration if the export does not have the minimum number of specified bindings.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_b_transport_export#(consumer) b_export;
   function new();
      super.new();
      this.b_export = new(this,”consumer export”,5,1);
   endfunction
endclass
```

task vmm_tlm_b_transport_export::b_transport

Blocking transport method of the export.

SystemVerilog

```
task b_transport(int id = -1, D trans, P ph =null , input int delay
= -1);
```

Description

Blocking transport task of the transport export. This task is internally called by the bound transport port. This task calls the b_transport method of parent transactor it is instantiated in.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_b_transport_export b_export;
   task b_transport(int id = -1, vmm_data trans, vmm_tlm
                            ph = null, int delay = -1);
      trans.display("From consumer");
      #10 ph.status = vmm_tlm::TLM_COMPLETED;
   endtask
endclass
```

macro `vmm_tlm_b_transport_export(SUFFIX)

Short hand macro to create unique blocking transport exports. This is required if more than one export is bound in a target transactor.

SystemVerilog

```
`vmm_tlm_b_transport_export(SUFFIX)
```

Description

This macro creates a uniquified vmm_tlm_b_transport_export class with the SUFFIX appended to the class name vmm_tlm_b_transport_export. The class with the name vmm_tlm_b_transport_exportSUFFIX is created in the scope where the macro is called.

This macro is required if there are multiple instances of the vmm_tlm_b_transport_export and each requires a unique implementation of the b_transport task in the parent transactor.

The b_transport methods in the parent transactor must be uniquified using the same SUFFIX to b_transport.

Alternatively, if multiple ports need to service the parent transactor then a single export with multiple bindings using unique ids can be used in place of the macro. The single b_transport method can be programmed to serve the various ports depending on the id.

Example

```
class consumer extends vmm_xactor;
   `vmm_tlm_b_transport_export(_1)
   `vmm_tlm_b_transport_export(_2)
   vmm_tlm_b_transport_export_1#(consumer) b_export1 =
                                    new(this, "export1");

   vmm_tlm_b_transport_export_2#(consumer) b_export2 =
                                   new(this,"export2");
   task b_transport_1(int id = -1, vmm_data trans,
                      vmm_tlm ph=null, int delay = -1);
```

```
      trans.display("From export1");
   endtask

   task b_transport_2(int id = -1, vmm_data trans,
                      vmm_tlm ph=null, int delay = -1);
      trans.display("From export2");
   endtask
endclass

class producer extends vmm_xactor;
   vmm_tlm_b_transport_port#(producer) b_port;
endclass

class my_env extends vmm_env;
   producer p1,p2;
   consumer c1;
   function build();
      c1.b_export1.tlm_bind(p1.b_port);
      c1.b_export2.tlm_bind(p2.b_port);
   endfunction
endclass
```

class vmm_tlm_nb_transport_fw_port#(T,D,P)

Non-blocking transport port for the forward path.

SystemVerilog

```
class vmm_tlm_nb_transport_fw_port #(type T = vmm_xactor, type D =
vmm_data, type P = vmm_tlm) extends vmm_tlm_port_base#(D,P);
```

Description

Class providing the non-blocking forward transport port. Transactions originating from the producer are sent on the forward path using this non-blocking transport port. The parameter type T is the class instantiating the transport port. This defaults to vmm_xactor. The parameter D is the data type of the transaction the port services. The default is vmm_data. The parameter type P is the phase class for this port. The default type is vmm_tlm.

The port can be bound to one export. A warning is issued if the port is left unbound.

function vmm_tlm_nb_transport_fw_port::new

Constructor of non-blocking forward transport port class.

SystemVerilog

```
function new(vmm_object parent, string name);
```

Description

Set the parent and instance name of the non-blocking forward transport port.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_nb_transport_fw_port#(producer) nb_port;
   function new();
      this.nb_port = new(this,”producer port”);
   endfunction
endclass
```

function vmm_tlm_nb_transport_fw_port::nb_transport_fw

Non-blocking forward transport function. The initiator transactor initiating this transport port should call the nb_transport_fw method of the transport port.

SystemVerilog

```
function vmm_tlm::sync_e nb_transport_fw(D trans, P ph = null, int
delay = -1);
```

Description

Call the nb_transport_fw method of the bound export. The argument trans is a handle of the transaction object, ph is a handle of the phase class and delay is the timing annotation.

The user must ensure that delay is provided in the loop where this non-blocking function is being called.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_nb_transport_port#(producer) nb_port;
   task main();
      my_data tr;
      while(1) begin
         tr = new();
         this.nb_port.nb_tranport_fw(tr);
         #5;
      end
   endtask
endclass
```

class vmm_tlm_nb_transport_fw_export#(T,D,P)

Non-blocking forward transport export class.

SystemVerilog

```
class vmm_tlm_nb_transport_fw_export#(type T = vmm_xactor, type D =
vmm_data , type P = vmm_tlm) extends vmm_tlm_export_base#(D,P);
```

Description

Class providing the non-blocking forward transport export. The parameter type T is the class instantiating the transport export. This defaults to vmm_xactor. The parameter D is the data type of the transaction the export services. The default is vmm_data. The parameter type P is the phase class for this export. The default type is vmm_tlm.

The export can be bound to multiple ports upto the max bindings specified in the constructor of this class.

function vmm_tlm_nb_transport_fw_export::new

Constructor of non-blocking forward transport export class. Any class instantiating this non-blocking export must provide an implementation of the nb_transport_fw function.

SystemVerilog

```
function new(T parent, string name, int max_binds = 1 , int
min_binds = 0);
```

Description

Set the parent and instance name of the blocking transport export. Sets the maximum and minimum bindings allowed for this export. The default value of maximum bindings is 1 and minimum binding is 0. An error is issued during tlm_bind if the current binding exceeds the maximum allowed bindings for the export. An error is issued during elaboration if the export does not have the minimum number of specified bindings.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_nb_transport_fw_export#(consumer) nb_export;
   function new();
      super.new();
      this.nb_export = new(this,”consumer export”,5,1);
   endfunction
endclass
```

function vmm_tlm_nb_transport_fw_export::nb_transport_fw

Non-blocking transport method of the export.

SystemVerilog

```
function vmm_tlm::sync_e nb_transport_fw(int id = -1, D trans, P ph
=null , input int delay = -1);
```

Description

Non-blocking transport function of the transport export. This function is internally called by the bound transport port. This function calls the b_transport method of parent transactor it is instantiated in. If the export is bound to multiple ports then the peer can be distinguished using the id field passed to the nb_transport_fw method.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_nb_transport_fw_export nb_export;
   function vmm_tlm::sync_e nb_transport_fw(int id = -1,
       vmm_data trans,vmm_tlm ph = null, int delay = -1);
      trans.display("From consumer");
      return vmm_tlm::TLM_COMPLETED;
   endfunction
endclass
```

macro `vmm_tlm_nb_transport_fw_export(SUFFIX)

Shorthand macro to create unique instances of non-blocking forward transport export. This is useful if multiple exports are required in the same target transactor.

SystemVerilog

```
`vmm_tlm_nb_transport_fw_export(SUFFIX)
```

Description

This macro creates a uniquified vmm_tlm_nb_transport_fw_export class with the SUFFIX appended to the class name vmm_tlm_nb_transport_fw_export. The class with the name vmm_tlm_nb_transport_fw_exportSUFFIX is created in the scope where the macro is called.

This macro is required if there are multiple instances of the vmm_tlm_nb_transport_fw_export and each requires a unique implementation of the nb_transport_fw task in the parent transactor.

The nb_transport_fw methods in the parent transactor must be uniquified using the same SUFFIX to nb_transport_fw.

Alternatively, if multiple ports need to service the parent transactor then a single export with multiple bindings using unique ids can be used in place of the macro. The single nb_transport_fw method can be programmed to serve the various ports depending on the id.

Example

```
class consumer extends vmm_xactor;
   `vmm_tlm_nb_transport_fw_export(_1)
   `vmm_tlm_nb_transport_fw_export(_2)
   vmm_tlm_nb_transport__fw_export_1#(consumer)
                       nb_export1 = new(this, “export1”);
   vmm_tlm_nb_transport_fw_export_2#(consumer)
                       nb_export2 = new(this, “export2”);
   function nb_transport_fw_1(int id = -1, vmm_data
```

```
                trans, vmm_tlm ph=null, int delay = -1);
      trans.display("From export1");
   endfunction
   task nb_transport_fw_2(int id = -1, vmm_data trans,
                       vmm_tlm ph=null, int delay = -1);
      trans.display("From export2");
   endtask
endclass

class producer extends vmm_xactor;
   vmm_tlm_nb_transport_fw_port#(producer) nb_port;
endclass

class my_env extends vmm_env;
   producer p1,p2;
   consumer c1;
   function build();
      c1.nb_export1.tlm_bind(p1.nb_port);
      c1.nb_export2.tlm_bind(p2.nb_port);
   endfunction
endclass
```

class vmm_tlm_nb_transport_bw_port#(T,D,P)

Non-blocking transport port for the backward path.

SystemVerilog

```
class vmm_tlm_nb_transport_bw_port #(type T = vmm_xactor, type D =
vmm_data, type P = vmm_tlm) extends vmm_tlm_port_base#(D,P);
```

Description

Class providing the non-blocking backward transport port. Transactions received from the producer on the forward path are sent back to the producer on the backward path using this non-blocking transport port. The parameter type T is the class instantiating the transport port. This defaults to vmm_xactor. The parameter D is the data type of the transaction the port services. The default is vmm_data. The parameter type P is the phase class for this port. The default type is vmm_tlm.

The port can be bound to one export. A warning is issued if the port is left unbound.

Example

```
TBD
```

function vmm_tlm_nb_transport_bw_port::new

Constructor of non-blocking backward transport port class.

SystemVerilog

```
function new(vmm_object parent, string name);
```

Description

Sets the parent and instance name of the non-blocking backward transport port.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_nb_transport_fw_export#(consumer) nb_export;
   vmm_tlm_nb_transport_bw_port#(producer)nb_port ;
   function new();
      this.nb_port = new(this,”consumer port”);
   endfunction
endclass
```

function vmm_tlm_nb_transport_bw_port::nb_transport_bw

Non-blocking backward transport function. The target transactor instantiating this transport port should call the nb_transport_bw method of the transport port.

SystemVerilog

```
function vmm_tlm::sync_e nb_transport_bw(D trans, P ph = null, int
delay = -1);
```

Description

Non-blocking transport function of the port. Calls the nb_transport_bw method of the bound export. The argument trans is a handle of the transaction object, ph is a handle of the phase class and delay is the timing annotation.

Example

```
class consumer extends vmm_xactor;
   vmm_tlm_nb_transport_bw_port#(consumer) nb_port;
   my_trans current_trans ;
   task main();
      while(1) begin
         this.nb_port.nb_tranport_bw(current_trans);
         #5;
      end
   endtask
endclass
```

class vmm_tlm_nb_transport_bw_export#(T,D,P)

Non-blocking backward transport export class.

SystemVerilog

```
class vmm_tlm_nb_transport_bw_export#(type T = vmm_xactor, type D =
vmm_data , type P = vmm_tlm) extends vmm_tlm_export_base#(D,P);
```

Description

Class providing the non-blocking backward transport export. This class should be instantiated in the initiator transactor which instantiates a non-blocking forward port. The transactions sent from this transactor on the forward path can be received by the transactor on the backward path through this backward export.

The parameter type T is the class instantiating the transport export. This defaults to vmm_xactor. The parameter D is the data type of the transaction the export services. The default is vmm_data. The parameter type P is the phase class for this export. The default type is vmm_tlm.

The export can be bound to multiple ports upto the max bindings specified in the constructor of this class.

Example

```
TBD
```

function vmm_tlm_nb_transport_bw_export::new

Constructor of non-blocking backward transport export class. Any class instantiating this non-blocking export must provide an implementation of the nb_transport_bw function.

SystemVerilog

```
function new(T parent, string name, int max_binds = 1 , int
min_binds = 0);
```

Description

Sets the parent and instance name of the blocking transport export. Sets the maximum and minimum bindings allowed for this export. The default value of maximum bindings is 1 and minimum bindings is 0. An error is issued during tlm_bind if the current binding exceeds the maximum allowed bindings for the export. An error is issued during elaboration if the export does not have the minimum number of specified bindings.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_nb_transport_bw_export#(producer) nb_export;
   vmm_tlm_nb_transport_fw_port#(producer)nb_port;
   function new();
      super.new();
      this.nb_export = new(this,”consumer export”,5,1);
   endfunction
endclass
```

function vmm_tlm_nb_transport_bw_export::nb_transport_fw

Non-blocking transport method of the export.

SystemVerilog

```
function vmm_tlm::sync_e nb_transport_bw(int id = -1, D trans, P ph
=null , input int delay = -1);
```

Description

Non-blocking transport function of the transport export. This function is internally called by the bound transport port. This function calls the nb_transport_bw method of parent transactor it is instantiated in. If the export is bound to multiple ports then the peer can be distinguished using the id field passed to the nb_transport_bw method.

Example

```
class producer extends vmm_xactor;
   vmm_tlm_nb_transport_fw_export nb_export;
   function vmm_tlm::sync_e nb_transport_bw(int id = -1,
       vmm_data trans,vmm_tlm ph = null, int delay = -1);
      trans.display("From producer on backward path.");
   endfunction
endclass
```

macro `vmm_tlm_nb_transport_bw_export(SUFFIX)

Shorthand macro to create unique instances of non-blocking backward transport export. This is useful if multiple exports are required in the same initiator transactor.

SystemVerilog

```
`vmm_tlm_nb_transport_bw_export(SUFFIX)
```

Description

This macro creates a uniquified vmm_tlm_nb_transport_bw_export class with the SUFFIX appended to the class name vmm_tlm_nb_transport_bw_export. The class with the name vmm_tlm_nb_transport_bw_exportSUFFIX is created in the scope where the macro is called.

This macro is required if there are multiple instances of the vmm_tlm_nb_transport_bw_export and each requires a unique implementation of the nb_transport_bw task in the parent transactor.

The nb_transport_bw methods in the parent transactor must be uniquified using the same SUFFIX to nb_transport_bw.

Alternatively, if multiple ports need to service the parent transactor then a single export with multiple bindings using unique ids can be used in place of the macro. The single nb_transport_bw method can be programmed to serve the various ports depending on the id.

Example

```
class producer extends vmm_xactor;
   `vmm_tlm_nb_transport_bw_export(_1)
   `vmm_tlm_nb_transport_bw_export(_2)
   vmm_tlm_nb_transport__bw_export_1#(producer)
                       nb_export1 = new(this, “export1”);
   vmm_tlm_nb_transport_bw_export_2#(producer)
                       nb_export2 = new(this, “export2”);
   function nb_transport_bw_1(int id = -1, vmm_data
```

```
                trans, vmm_tlm ph=null, int delay = -1);
      trans.display("From export1");
   endfunction
   function nb_transport_bw_2(int id = -1,vmm_data
                trans, vmm_tlm ph=null, int delay = -1);
      trans.display("From export2");
   endfunction
endclass

class consumer extends vmm_xactor;
   vmm_tlm_nb_transport_bw_port#(producer) nb_port;
endclass

class my_env extends vmm_env;
   producer p1;
   consumer c1,c2;
   function build();
      p1.nb_export1.tlm_bind(c1.nb_port);
      p1.nb_export2.tlm_bind(c2.nb_port);
   endfunction
endclass
```

class vmm_tlm_nb_transport_port#(T,D,P)

Bidirectional non-blocking port.

SystemVerilog

```
class vmm_tlm_nb_transport_port#(type T = vmm_xactor, type D =
vmm_data, type FW_P = vmm_tlm, type BW_P = FW_P) extends
vmm_tlm_socket_base#(D,FW_P);
```

Description

Bi-directional port providing a non-blocking transport port for the forward path and a non-blocking transport export for the backward path in a single transport port.

Only 1-1 binding is allowed for this bi-directional, non-blocking port. The vmm_tlm_nb_transport_port can only be bound to the vmm_tlm_nb_transport_export.

The vmm_tlm_socket_base provides all the access methods that are provided by the vmm_tlm_port_base class. The methods available with this class are tlm_bind, tlm_unbind, tlm_import and get_peer. For detailed descricption of these methods please refer to the vmm_tlm_port_base class.

This class provides non-blocking transport methods for both, the forward path, nb_transport_fw and the backward path nb_transport_bw.

Any transactor class instantiating this bi-directional port must provide an implementation of the nb_transport_bw method and should call the nb_transport_fw method of this port.

Example

```
 class producer extends vmm_xactor;
vmm_tlm_nb_transport_port#(producer) nb_port =
                          new(this,"producer_bi");

function vmm_tlm::sync_e nb_transport_bw(int id=-1,
```

```
                  my_trans trans, vmm_tlm ph, int delay);
   endfunction
   virtual task main();
     my_trans tr;
     while(1) begin
        tr = new();
        tr.randomize();
        this.nb_port.nb_transport_fw(tr);
        #5;
     end
   endtask
endclass
```

macro `vmm_tlm_nb_transport_port(SUFFIX)

Shorthand macro to create unique classes of the bi-directional port. This is useful if multiple vmm_tlm_nb_transport_port instances are required in the same initiator transactor each having its own implementation of the nb_transport_bw method.

SystemVerilog

```
`vmm_tlm_nb_transport_port(SUFFIX)
```

Description

The use model is similar to the shorthand macros provided for the unidirectional non-blocking ports. For details, please refer to the macro description of `vmm_tlm_nb_transport_fw_export.

Example

```
class producer extends vmm_xactor;
   `vmm_tlm_nb_transport_port(_1)
   `vmm_tlm_nb_transport_port(_2)
    vmm_tlm_nb_transport_port_1#(producer) nb_port1;
    vmm_tlm_nb_transport_port_2#(producer) nb_port2;
    function vmm_tlm::sync_e nb_transport_bw_1
      (int id=-1,my_trans trans,vmm_tlm ph,int delay);
    endfunction
    function vmm_tlm::sync_e nb_transport_bw_2
      (int id=-1,my_trans trans,vmm_tlm ph,int delay);
    endfunction
endclass
```

class vmm_tlm_nb_transport_export#(T,D,P)

Bidirectional non-blocking export.

SystemVerilog

```
class vmm_tlm_nb_transport_export#(type T = vmm_xactor, type D =
vmm_data, type FW_P = vmm_tlm, type BW_P = FW_P) extends
vmm_tlm_socket_base#(D,FW_P);
```

Description

Bi-directional export providing non-blocking transport export for the forward path and non-blocking transport port for the backward path in a single transport export.

Only 1-1 binding is allowed for this bi-directional non-blocking export. The vmm_tlm_nb_transport_export can only be bound to the vmm_tlm_nb_transport_port.

The vmm_tlm_socket_base provides all the access methods that are provided by the vmm_tlm_export_base class. The methods available with this class are tlm_bind, tlm_unbind, tlm_import and get_peer. For detailed descricption of these methods please refer to the vmm_tlm_port_exbase class.

This class provides non-blocking transport methods for both, the forward path, nb_transport_fw and the backward path nb_transport_bw.

Any transactor class instantiating this bi-directional export must provide an implementation of the nb_transport_fw method and should call the nb_transport_bw method of this export.

Example

```
    class consumer extends vmm_xactor;

  vmm_tlm_nb_transport_export#(consumer) nb_export =
                              new(this,"consumer_bi");
```

```
   function vmm_tlm::sync_e nb_transport_fw(int id=-1,
                  my_trans trans, vmm_tlm ph, int delay);
   endfunction
   virtual task main();
     my_trans tr;
     while(1) begin
        this.tr.notify.wait_for(vmm_data::ENDED);
        this.nb_port.nb_transport_bw(tr);
        #5;
     end
   endtask
endclass
```

macro `vmm_tlm_nb_transport_export(SUFFIX)

Shorthand macro to create unique classes of the bi-directional export. This is useful if multiple vmm_tlm_nb_transport_export instances are required in the same initiator transactor, each having its own implementation of the nb_transport_fw method.

SystemVerilog

```
`vmm_tlm_nb_transport_export(SUFFIX)
```

Description

The use model is similar to the shorthand macros provided for the unidirectional non-blocking exports. For details, please refer to the macro description of `vmm_tlm_nb_transport_fw_export.

Example

```
class consumer extends vmm_xactor;
   `vmm_tlm_nb_transport_export(_1)
   `vmm_tlm_nb_transport_export(_2)
    vmm_tlm_nb_transport_export_1#(producer) nb_exp1;
    vmm_tlm_nb_transport_export_2#(producer) nb_exp2;
    function vmm_tlm::sync_e nb_transport_fw_1(int id=-1,
                          my_trans trans,vmm_tlm ph,int delay);
    endfunction
    function vmm_tlm::sync_e nb_transport_fw_2(int id=-1,
                          my_trans trans,vmm_tlm ph,int delay);
    endfunction
endclass
```

class vmm_tlm_simple_initiator_socket#(T,D,P)

Bidirectional socket port providing both blocking and non-blocking paths.

SystemVerilog

```
class vmm_tlm_simple_initiator_socket#(type T =
       vmm_xactor, type D = vmm_data, type P = vmm_tlm)
                   extends vmm_tlm_socket_base#(D,P);
```

Description

Bi-directional socket port providing blocking transport port, non-blocking transport port for the forward path and non-blocking transport export for the backward path in a single transport socket.

Only 1-1 binding is allowed for this bi-directional socket. The vmm_tlm_simple_initiator_socket can only be bound to the vmm_tlm_simple_target_socket.

The vmm_tlm_socket_base provides all the access methods that are provided by the vmm_tlm_port_base class. The methods available with this class are tlm_bind, tlm_unbind, tlm_import and get_peer. For detailed descricption of these methods please refer to the vmm_tlm_port_base class.

This class provides blocking transport method b_transport, and non-blocking transport methods for both, the forward path, nb_transport_fw and the backward path nb_transport_bw.

Any transactor class instantiating this bi-directional socket must provide an implementation of the nb_transport_bw method and should call either or both the b_transport and nb_transport_fw methods of this socket.

Example

```
    class producer extends vmm_xactor;
  vmm_tlm_simple_initiator_socket#(producer) socket =
                             new(this,"producer_socket");
```

```
   function vmm_tlm::sync_e nb_transport_bw(int id=-1,
                  my_trans trans, vmm_tlm ph, int delay);
   endfunction
   virtual task main();
     my_trans tr;
     while(1) begin
        tr = new();
        tr.randomize();
        this.socket.nb_transport_fw(tr);
        #5;
     end
   endtask
endclass
```

macro `vmm_tlm_simple_initiator_socket(SUFFIX)

Shorthand macro to create unique classes of the bi-directional socket. This is useful if multiple vmm_tlm_simple_initiator_socket instances are required in the same initiator transactor, each having its own implementation of the nb_transport_bw method.

SystemVerilog

```
`vmm_tlm_simple_initiator_socket(SUFFIX)
```

Description

The use model is similar to the shorthand macros provided for the unidirectional non-blocking ports. For details, please refer to the macro description of `vmm_tlm_nb_transport_fw_export.

Example

```
class producer extends vmm_xactor;
   `vmm_tlm_simple_initiator_socket(_1)
   `vmm_tlm_simple_initiator_socket(_2)
    vmm_tlm_simple_initiator_socket_1#(producer) s1;
    vmm_tlm_simple_initiator_socket_2#(producer) s2;
    function vmm_tlm::sync_e nb_transport_bw_1
      (int id=-1,my_trans trans,vmm_tlm ph,int delay);
    endfunction
    function vmm_tlm::sync_e nb_transport_bw_2
      (int id=-1,my_trans trans,vmm_tlm ph,int delay);
    endfunction
endclass
```

class vmm_tlm_simple_target_socket#(T,D,P)

Bidirectional socket export providing both blocking and non-blocking paths.

SystemVerilog

```
class vmm_tlm_simple_target_socket#(type T =
       vmm_xactor, type D = vmm_data, type P = vmm_tlm)
                   extends vmm_tlm_socket_base#(D,P);
```

Description

Bi-directional socket export providing blocking transport export, non-blocking transport export for the forward path and non-blocking transport port for the backward path in a single transport socket.

Only 1-1 binding is allowed for this bi-directional socket. The vmm_tlm_simple_target_socket can only be bound to the vmm_tlm_simple_initiator_socket.

The vmm_tlm_socket_base provides all the access methods that are provided by the vmm_tlm_port_base class. The methods available with this class are tlm_bind, tlm_unbind, tlm_import and get_peer. For detailed descricption of these methods please refer to the vmm_tlm_port_base class.

This class provides blocking transport method b_transport, and non-blocking transport methods for both, nb_transport_fw for the forward path, and nb_transport_bw for the backward path.

Any transactor class instantiating this bi-directional socket must provide an implementation of the nb_transport_fw and b_transport methods and should call nb_transport_bw method of this socket for the backward path.

Example

```
    class consumer extends vmm_xactor;

  vmm_tlm_simple_target_socket#(consumer) nb_export =
```

```
                              new(this,"consumer_socket");

   function vmm_tlm::sync_e nb_transport_fw(int id=-1,
                  my_trans trans, vmm_tlm ph, int delay);
   endfunction
   task b_transport(int id=-1, my_trans trans, vmm_tlm ph,
                                                int delay);
   virtual task main();
     my_trans tr;
     while(1) begin
        this.tr.notify.wait_for(vmm_data::ENDED);
        this.nb_port.nb_transport_bw(tr);
        #5;
     end
   endtask
endclass
```

macro `vmm_tlm_simple_target_socket(SUFFIX)

Shorthand macro to create unique classes of the bi-directional socket. This is used if multiple vmm_tlm_simple_target_socket instances are required in the same target transactor, each having its own implementation of the nb_transport_bw method.

SystemVerilog

```
`vmm_tlm_nb_simple_target_socket(SUFFIX)
```

Description

The use model is similar to the short hand macros provided for the unidirectional exports. For details, please refer to the macro description of `vmm_tlm_nb_transport_fw_export.

Example

```
class consumer extends vmm_xactor;
   `vmm_tlm_simple_target_socket(_1)
   `vmm_tlm_simple_target_socket(_2)
    vmm_tlm_simple_target_socket_1#(producer) soc1;
    vmm_tlm_simple_target_socket_2#(producer) soc2;
    function vmm_tlm::sync_e nb_transport_fw_1
      (int id=-1,my_trans trans,vmm_tlm ph,int delay);
    endfunction
    function vmm_tlm::sync_e nb_transport_fw_2
      (int id=-1,my_trans trans,vmm_tlm ph,int delay);
    endfunction
    task b_transport_1(int id=-1, my_trans trans,
                           vmm_tlm ph, int delay);
    endfunction

    task b_transport_2(int id=-1, my_trans trans,
                           vmm_tlm ph, int delay);
```

```
    endfunction
endclass
```

class vmm_tlm_analysis_port#(T,D,P)

Analysis ports are useful to broadcast transactions to observers like scoreboards and functional coverage models. Analysis ports can be bound to any number of observers through the observers analysis export.

The analysis port calls the write method of all the observers bound to it.

SystemVerilog

```
class vmm_tlm_analysis_port#(type T = vmm_xactor, type D =
vmm_data,) extends vmm_tlm_analysis_port_base#(D);
```

Description

The analysis port can be instantiated in any transactor class that wishes to broadcast the transaction object to the connected observers.

Any number of bindings are allowed for the analysis port. The analysis port calls the write methods of the connected analysis exports, which in turn execute the write methods of their respective parent components.

The vmm_tlm_analysis_port_base provides all the access methods that are provided by the vmm_tlm_port_base class. The methods provided by the vmm_tlm_analysis_port_base are get_peers, get_n_peers, get_peer_id, get_peer, tlm_bind, tlm_unbind, tlm_import. For details on these access methods, please refer to the description provided in the vmm_tlm_port_base class.

Example

```
    class consumer extends vmm_xactor;
   vmm_tlm_analysis_port#(consumer) analysis_port =
                          new(this,"consumer_analysis");

   function b_transport(int id=-1,my_trans trans,
                        vmm_tlm ph, int delay);
      this.analysis_port.write(trans);
   endfunction
endclass
```

class vmm_tlm_analysis_export#(T,D,P)

Analysis exports are used by observer components that implement a write method to receive broadcast transactions from other components that instantiate the vmm_tlm_analysis_port. Analysis exports can be bound to any number of analysis ports as specified in the constructor of the analysis export. The different analysis ports connected to this export can be distinguished using the peer id of the analysis port.

The analysis export implements the write method which is called by the analysis port(s) bound to this export.

SystemVerilog

```
class vmm_tlm_analysis_export#(type T = vmm_xactor, type D =
vmm_data,) extends vmm_tlm_analysis_export_base#(D);
```

Description

The analysis export can be instantiated in a component class that wishes to receive broadcast transaction objects from other components.

The vmm_tlm_analysis_export_base provides all the access methods that are provided by the vmm_tlm_export_base class. The methods provided by the vmm_tlm_analysis_port_base are get_peers, get_n_peers, get_peer_id, get_peer, tlm_bind, tlm_unbind, tlm_import. For details on these access methods, please refer to the description provided in the vmm_tlm_export_base class. Methods to get and set the minimum and maximum bindings for the port are also provided.

Available methods are:

```
function vmm_tlm_analysis_export::set_max_bindings(
                                      int unsigned max);
function vmm_tlm_analysis_export::set_min_bindings(
                                      int unsigned min);
function int unsigned vmm_tlm_analysis_export::get_max_bindings();
function int unsigned vmm_tlm_analysis_export::get_min_bindings();
```

Example

```
 class scoreboard extends vmm_unit;
vmm_tlm_analysis_export#(scoreboard) analysis_export =
                        new(this,”scb_analysis”);
function write(int id=-1, my_trans trans);
endfunction
endclass
```

macro `vmm_tlm_analysis_export(SUFFIX)

Shorthand macro to create unique class names of the analysis export. This is used if multiple vmm_tlm_analysis_export instances are required in the same observer class each having its own implementation of the write method.

SystemVerilog

```
`vmm_tlm_analysis_export(SUFFIX)
```

Description

The use model is similar to the shorthand macros provided for the unidirectional exports. For details, please refer to the macro description of `vmm_tlm_nb_transport_fw_export.

Example

```
class scoreboard extends vmm_unit;
   `vmm_tlm_analysis_export(_1)
   `vmm_tlm_analysis_export(_2)
    vmm_tlm_analysis_export_1#(scoreboard) scb1;
    vmm_tlm_analysis_export_2#(scoreboard) scb2;
    function write_1 (int id=-1,my_trans trans);
       `vmm_note(log, $psprintf("Received %s from %0d",
                      Trans.psdisplay(""), id);
    endfunction
    function write_2 (int id=-1,my_trans trans);
       `vmm_note(log, $psprintf("Received %s from %0d",
                      Trans.psdisplay(""), id);
    endfunction
 endclass
```

RAL updates

All *vmm_ral* classes will be based on vmm_object as they do not need to be phased. The top-level ***vmm_ral_block_or_sys*** object instances are specified as the roots of the RAL namespace.

The specified arguments `...(..., string fname = "", int lineno = 0)` are following methods and use in any error message issued by these methods.

```
vmm_mam::reserve_region();
vmm_mam::request_region();
vmm_mam_region::read();
vmm_mam_region::write();
vmm_mam_region::burst_read();
vmm_mam_region::burst_write();
vmm_mam_region::peek();
vmm_mam_region::poke();
vmm_ral_access::read();
vmm_ral_access::write();
vmm_ral_access::burst_read();
vmm_ral_access::burst_write();
vmm_ral_access::read_by_name();
vmm_ral_access::write_by_name();
vmm_ral_access::read_mem_by_name();
vmm_ral_access::write_mem_by_name();
vmm_ral_block_or_sys::update();
vmm_ral_block_or_sys::mirror();
vmm_ral_field/mem/reg/vfield/vreg::read();
vmm_ral_field/mem/reg/vfield/vreg::write();
vmm_ral_field/mem/reg/vfield/vreg::peek();
vmm_ral_field/mem/reg/vfield/vreg::poke();
vmm_ral_field/mem/reg::mirror();
vmm_ral_field/mem/reg::append_callback();
vmm_ral_field/mem/reg::prepend_callback();
vmm_ral_mem::burst_read();
vmm_ral_mem::burst_write();
vmm_ral_mem/reg::set_frontdoor();
vmm_ral_mem/reg::set_backdoor();
vmm_ral_field/reg::predict();
```

```
vmm_ral_field/reg::set();
vmm_ral_field/reg::get();
vmm_ral_mem/reg_frontdoor::read();
vmm_ral_mem/reg_frontdoor::write();
vmm_ral_mem_frontdoor::burst_read();
vmm_ral_mem_frontdoor::burst_write();
```

Example

```
ral_sys_dut ral_model = new;
vmm_log log = new("prog_blk","test");
vmm_ral_access acc = new();
vmm_ral_reg regs[];
vmm_ral_block_or_sys my_ral;
vmm_rw::status_e status;
initial begin
   acc.set_model(ral_model);
   ral_model.b[1].update(status, vmm_ral::DEFAULT, `__FILE__,
`__LINE__);
```

vmm_ral_block_or_sys::get_n_tops();

Return the number of top-level block or system RAL models.

SystemVerilog

```
static function int vmm_ral_block_or_sys::get_n_tops();
```

Description

The number of tops depends upon the way it is instantiated in the env.If a ral block has no parent then it is considered as top.

Example

```
ral_sys_dut ral_model = new;
vmm_log log = new("prog_blk","test");
vmm_ral_access acc = new();
vmm_ral_reg regs[];

initial begin
acc.set_model(ral_model);

$display("No. of tops are %0d",ral_model.get_n_tops());

if (ral_model.get_n_tops() != 4)
     `vmm_error(log,$psprintf("expected total num of tops is 4 but
got %0d",ral_model.get_n_tops()));

log.report();
```

vmm_ral_block_or_sys::get_top

Return the specified top-level block or system RAL model.

SystemVerilog

```
static function vmm_ral_block_or_sys
vmm_ral_block_or_sys::get_top(int
n = 0);
```

Description

This returns the specified top-level block or system RAL model.

Example

```
ral_sys_dut ral_model = new;
vmm_log log = new("prog_blk","test");
vmm_ral_access acc = new();
vmm_ral_reg regs[];
vmm_ral_block_or_sys my_ral;
initial begin
   acc.set_model(ral_model);

   my_ral = ral_model.get_top(2);
   $display("2nd top is %s",my_ral.get_name());
```

vmm_rw::HAS_X

Add to the ***vmm_rw::status_e*** type.

SystemVerilog

```
vmm_rw::status_e;
```

Scoreboard Updates

The following modification has been made to the datastream scoreboard to take advantage of the new VMM 1.2 features. The interface to the datastream scoreboard compile-time type checked. Additional methods are added to handle parameterized types.

class vmm_sb_ds

Datastream scoreboard class with parameters for input and expect data types.

SystemVerilog

```
class vmm_sb_ds#(type INP = vmm_data, type EXP = INP);
```

Description

This class implements a generic data stream scoreboard which accepts parameters for the input and output packet types. A single instance of this class is used to check the proper transformation, multiplexing and ordering of multiple data streams. The Packet types INP and EXP can be vmm_data derivatives or non vmm_data extensions.

Example

```
class my_scb #(apb_mst_trans,apb_slv_trans) extends
               vmm_sb_ds#(apb_mst_trans,abp_slv_trans);
endclass
```

function vmm_sb_ds::inp_stream_id

Return the stream identifier for an INP packet.

SystemVerilog

```
virtual function int vmm_sb_ds::inp_stream_id(INP pkt);
```

Description

This method returns a non-negative stream identifier corresponding to the specified packet which is of INP kind. This method can be used to determine the stream a packet belongs to based on the packet's content, such as source or destination address. This method is used by the insert and expect methods of the scoreboard if these methods are called with negative inp_stream_id.

Example

```
virtual function int inp_stream_id(apb_mst_trans pkt);
   apb_mst_trans tr;
   tr = pkt;
   return tr.master_id;
endfunction
```

function vmm_sb_ds::exp_stream_id

Return the stream identifier for an EXP packet.

SystemVerilog

```
virtual function int vmm_sb_ds::exp_stream_id(EXP pkt);
```

Description

This method returns a non-negative stream identifier corresponding to the specified packet which is of EXP kind. This method can be used to determine the stream a packet belongs to based on the packet's content, such as source or destination address. This method is used by the insert and expect methods of the scoreboard if these methods are called with negative exp_stream_id.

Example

```
virtual function int exp_stream_id(apb_slv_trans pkt);
   if(pkt.addr[0] == 0)
      return 0;
   else
     return 1;
endfunction
```

function vmm_sb_ds::inp_insert

Insert a packet of INP type into the scoreboard.

SystemVerilog

```
virtual function int vmm_sb_ds::inp_insert(INP pkt,
                                   int inp_stream_id =-1,
                                   int exp_stream_id=-1);
```

Description

This method inserts the specified packet of INPUT type into the scoreboard and returns TRUE if the insertion was successful. The packet pkt is considered to be a stimulus packet and will be transformed by calling the vmm_sb_ds::transform() method and stored in the scoreboard to compare with an EXP packet when the expect method is called.

Example

```
class apb_master_sb_callbacks extends apb_callbacks;
   mst_to_slv_sb#(apb_mst_trans,apb_slv_trans) m2s;
   virtual task pre_cycle(apb_master xactor,
                          apb_mst_trans cycle,
                          ref bit    drop);
        void'(this.m2s.inp_insert(cycle,.exp_stream_id(1)));
   endtask: pre_cycle
endclass
```

function vmm_sb_ds::exp_insert

Insert a packet of EXP type into the scoreboard.

SystemVerilog

```
virtual function int vmm_sb_ds::exp_insert(INP pkt,
                                           int inp_stream_id =-1,
                                           int exp_stream_id=-1);
```

Description

This method inserts the specified packet which is of EXP kind into the scoreboard and returns TRUE if the insertion was successful. The packet pkt is considered a stimulus packet and will stored in the scoreboard to compare with an EXP packet when the expect method is called.

Example

```
class apb_master_sb_callbacks extends apb_callbacks;
   mst_to_slv_sb#(apb_mst_trans,apb_slv_trans) m2s;
   virtual task pre_cycle(apb_master xactor,
                          apb_slv_trans cycle,
                          ref bit    drop);
      void'(this.m2s.exp_insert(cycle,.exp_stream_id(1)));
   endtask: pre_cycle
endclass
```

function vmm_sb_ds::inp_remove

Remove a packet of INP type from the scoreboard.

SystemVerilog

```
virtual function bit vmm_sb_ds::inp_remove(INP pkt,
                                           int inp_stream_id =-1,
                                           int exp_stream_id=-1);
```

Description

This method removes the specified packet of INP type from the scoreboard and returns one if the corresponding packet was successfully found in the scoreboard and removed. Zero is returned if the packet is not found in the scoreboard for the specified stream_id.

Example

```
class apb_master_sb_callbacks extends apb_callbacks;
   mst_to_slv_sb#(apb_mst_trans,apb_slv_trans) m2s;
   virtual task post_cycle(apb_master xactor,
                           apb_mst_trans cycle,
                           ref bit    drop);
      if(cycle.corrupt == 1)
         void'(this.m2s.inp_remove(cycle,.exp_stream_id(1)));
   endtask: post_cycle
endclass
```

function vmm_sb_ds::exp_remove

Remove a packet of EXP type from the scoreboard.

SystemVerilog

```
virtual function bit vmm_sb_ds::exp_remove(EXP pkt,
                                   int inp_stream_id =-1,
                                   int exp_stream_id=-1);
```

Description

This method removes the specified packet of EXP type from the scoreboard and returns one if the corresponding packet was successfully found in the scoreboard and removed. Zero is returned if the packet is not found in the scoreboard for the specified stream_id.

Example

```
class apb_master_sb_callbacks extends apb_callbacks;
   mst_to_slv_sb#(apb_mst_trans,apb_slv_trans) m2s;
   virtual task post_cycle(apb_master xactor,
                           apb_slv_trans cycle,
                           ref bit    drop);
      if(cycle.corrupt == 1)
         void'(this.m2s.exp_remove(cycle,.exp_stream_id(1)));
   endtask: post_cycle
endclass
```

vmm_sb_ds::inp_ap

VMM TLM analysis export for INP transactions.

SystemVerilog

```
`vmm_tlm_analysis_export(_inp)
 vmm_tlm_analysis_export_inp#(vmm_sb_ds#(INP,EXP),INP) inp_ap;
```

Description

TLM analysis export for data stream scoreboard. This analysis export services the INP data type. To use this export the vmm_sb_ds extension should implement the write_inp method. This analysis export can be bound to an analysis port. The write_inp method of the vmm_sb_ds will get executed when the write method of the bound analysis port is called.

Example

```
class mst_to_slv_sb #(apb_mst_trans,apb_slv_trans)
                 extends vmm_sb_ds(apb_mst_trans, apb_slv_trans);

   virtual function void write_inp(apb_mst_trans tr);
      this.inp_insert(tr,.exp_stream_id(1));
   endfunction
endclass
```

vmm_sb_ds::exp_ap

VMM TLM analysis export for EXP transactions.

SystemVerilog

```
`vmm_tlm_analysis_export(_exp)
 vmm_tlm_analysis_export_exp#(vmm_sb_ds#(INP,EXP),EXP) exp_ap;
```

Description

TLM analysis export for data stream scoreboard. This analysis export services the EXP data type. To use this export the vmm_sb_ds extension should implement the write_exp method. This analysis export can be bound to an analysis port. The write_exp method of the vmm_sb_ds will get executed when the write method of the bound analysis port is called.

Example

```
class mst_to_slv_sb #(apb_mst_trans,apb_slv_trans)
                  extends vmm_sb_ds(apb_mst_trans, apb_slv_trans);

   virtual function void write_exp(apb_slv_trans tr);
      this.expect_in_order(tr,.exp_stream_id(1));
   endfunction
endclass
```