ERRATA

The followings errors and omissions have been identified in the first edition of *Verification Methodology Manual for SystemVerilog*. They will be corrected in any future editions.

THROUGHOUT THE BOOK

A few examples and some VMM utilities are shown using the user-defined identifier "*instance*". This identifier is actually reserved by the SystemVerilog language and should not be used. All instances of the identifier "*instance*" should be replaced with "*inst*".

For example, the specification of the *vmm_log::new()* method on page 368 should read:

```
function new( string name,
              string inst,
              vmm_log under = null);
```

PAGE 59

Example 3-10 uses the *$psprintf()* system task provided by Synopsys' VCS® product. Although tools are free to provide additional system tasks, the example does not depend on the *$psprintf()* task.

Example 3-10 should read as follow:

Example 3-10. Using VMM Message Service Interface for Reporting

```
`ifndef SYNTHESIS
  string inst;
  vmm_log log;
  $sformat(inst, "%m");
  log = new("pipeline checks", inst);
....
  check_z : assert property(
     @(negedge reset) data_out === 8'bzzzzzzzz )
  else
    `vmm_error(log, "data_out not Hi-Z when in reset");
`endif
```

PAGE 64

Example 3-16 does not match the description of the following paragraph: the expression *`within x[->2]`* means "within the second occurence of *`x`*".

Example 3-16 should read as follow:

Example 3-16. Bounding Length Using *`within`*

```
a_within: assert property
     ( @(posedge clk)
       y |-> (a ##[1:$] b ##[1:$] c)
                within
              x[->1]
     ) else ...;
```

PAGE 69

The explanatory paragraph after Example 3-24 quotes a portion of the example as *`a |-> #1`*. The example has the correct syntax and should be quoted as *`a |-> ##1 b`*.

The paragraph should read:

The problem is that whenever *`a`* is false, then *`a |-> ##1 b`* succeeds, and hence, the negation (*`not`*) will fail. This is clearly not the intended result. It is required that if *`a`* happens and it is followed by *`b`* then the assertion should fail which is exactly how the property should be written:

PAGE 80

Example 3-42 uses the *$psprintf()* system task provided by Synopsys' VCS® product. Although tools are free to provide additional system tasks, the example does not depend on the *$psprintf()* task.

Example 3-42 should read as follow:

Example 3-42. Instance of the Message Service Interface Class

```
string inst;
vmm_log log;
$sformat(inst, "%m");
log = new("assert_name", inst);
```

PAGE 81

Example 3-45 uses the *$psprintf()* system task provided by Synopsys' VCS® product. Although tools are free to provide additional system tasks, the example does not depend on the *$psprintf()* task.

Example 3-45 should read as follow:

Example 3-45. Assertion Failure Reporting

```
task sva_checker_error;
  input bit [60*8-1:0] err_msg;
  `ifndef SYNTHESIS
  begin
   `ifdef ASSERT_MAX_REPORT_ERROR
   error_count = error_count + 1;
   if (error_count <= `ASSERT_MAX_REPORT_ERROR) begin
   `endif
     `ifndef SVA_CHECKER_NO_MESSAGE
     `ifdef SVA_VMM_LOG_ON
     string msg;
     $sformat(msg,
       "SVA_CHECKER_ERROR:%s:%0s:severity %0d: \
        category %0d",
        msg, err_msg, severity_level, category);
     `vmm_error(log, msg);
     `else
     $display(
       "SVA_CHECKER_ERROR:%s:%s:%0s:severity %0d: \
        category %0d : time %0t : %m",
        assert_name, msg, err_msg,
```

```
            severity_level, category, $time);
      `endif
      `ifdef ASSERT_MAX_REPORT_ERROR
    end
    `endif
    `endif
    if (severity_level == 0) sva_checker_finish;
end
`endif
endtask
...
(* category = category *) assert_one_hot:
  assert property (
    @(sampling_ev) disable iff (!not_resetting)
      ($countones(test_expr) == 1) )
    else
      sva_std_error(.err_msg({"failure ", msg});
```

PAGE 82

Example 3-46 contains a syntax error: the index of the least-significant bit in the bit slicing of *sva_v_deferred_pop_sr* is missing.

Example 3-46 should read as follow:

Example 3-46. Sampling of Design Variables for Auxiliary State Variables

```
clocking sampling_ev @(posedge sva_checker_clk);
   input not_resetting, push, push_data, pop, pop_data;
endclocking : sampling_ev
...
// use sampled value in an assignment
always @(sampling_ev) begin
   sva_v_deferred_pop_sr <=
               { sva_v_deferred_pop_sr[pop_lat-2:1],
                 sampling_ev.pop};
end
```

PAGE 133

Example 4-23 contains a syntax error: the *wait* statement should compare against *run_for_n_rx_frames*.

Example 4-23 should read as follow:

Example 4-23. Configurable Testcase Duration

```
class tb_env extends vmm_env;
   ...
   virtual task wait_for_end();
      super.wait_for_end();
      ...
      wait (this.cfg.run_for_n_tx_frames == 0 &&
            this.cfg.run_for_n_rx_frames == 0);
      ...
   endtask: wait_for_end
   ...
endclass: tb_env
```

PAGE 123

Example 4-14 shows a generator class to be extended from the *vmm_data* class. This is not the appropriate base class for transactors. The generator class must be extended from the *vmm_xactor* class.

Example 4-14 should read as follow:

Example 4-14. Generators are Transactors

```
class eth_frame_gen extends vmm_xactor;
   ...
endclass: eth_frame_gen
```

The following guideline should be added:

Suggestion 4-27.5 —*A* `program` *can be declared* `automatic`.

By default all declarations in a program are static, like in a module or interface. However, a testbench being more software-like than hardware-like, it may be desirable to have all local variable declarations be dynamic, like in C++.

Example 4-14.5. Declaring automatic *program*

```
automatic program test;
   ...
endprogram: test
```

PAGE 160

Recommendation 4-86 recommends using external *constraint* blocks. Unfortunately, the SystemVerilog language does not include the *extern* attribute for *constraint* blocks, even for externally-defined *constraint* blocks. The *extern* attribute must not be used.

Recommendation 4-86 should read as follow:

Recommendation 4-86 —*Undefined constraint blocks named "test_constraintsX" should be declared.*

If *constraint* blocks are left undefined, they are considered empty and do not add any constraints to the *class* instances. These *constraint* blocks can be defined later by individual tests to add constraints to all instances of the *class*. See Alternative 5-21 on page 229.

Example 4-47. Declaring Undefined *constraint* Blocks

```
class eth_frame extends vmm_data;
   ...
   constraint test_constraints1;
   constraint test_constraints2;
   constraint test_constraints3;
   ...
endclass: eth_frame
```

PAGE 166

Example 4-52 shows an extension of the *vmm_xactor::reset_xactor()* method that invokes the *super.start_xactor()* method. This is the wrong method to invoke. It should invoked *super.reset_xactor()* instead.

Example 4-52 should read as follow:

Example 4-52. Extension of a Control Method

```
function void
   mii_mac_layer::reset_xactor(reset_e typ = SOFT_RST);
   super.reset_xactor(typ);
   ...
endfunction: reset_xactor
```

PAGE 174

The following recommendation should be added:

Recommendation 4-113.5 — *The message service interface in all channel instances used by a transactor should be configured as logically below the message service interface in that transactor.*

Message service interfaces can be controlled hierarchically. By configuring the message service interfaces in the channels used by a transactor as hierarchically below the message service interface of the transactor, all message service interfaces used by the transactor can be controlled using a single command. Channels are usually shared by two transactors. Therefore the message service interface in channels will be logically below two different message service interface. The message service interface in the consumer and producer transactors will be logically above the message service interface in the channel.

Example 4-61.5. Hierarchical Message Service Interfaces

```
class mii_mac_layer extends vmm_xactor;
   eth_frame_channel tx_chan;
   eth_frame_channel rx_chan;
   ...
   function new(...
                eth_frame_channel tx_chan = null,
                eth_frame_channel rx_chan = null, ...);
      ...
      this.log.is_above(this.tx_chan.log);
      this.log.is_above(this.rx_chan.log);
   endfunction: new
   ...
endclass: mii_mac_layer
```

Rule 4-115 should be modified as follow:

Rule 4-115 — *Reactive and passive transactors shall allocate a new transaction descriptor instance from a factory instance using the* `vmm_data::allocate()` *or* `vmm_data::copy()` *method.*

PAGE 233

In Example 5-30, the number of generated instances in the `REPEAT_10` scenario is erroneously specified as 100 instead of 10.

In Example 5-30, the type of the *n_insts* argument in the *apply()* task is missing the *unsigned* attribute.

The task *apply()* in Example 5-30 should read as follow:

Example 5-30. User-Defined Scenarios

```
class my_scenarios extends eth_frame_sequence;
   ...
   virtual task apply(eth_frame_channel channel,
                      ref int unsigned  n_insts);
      if (scenario_kind == REPEAT_10) begin
         repeat (10) begin
            channel.put(items[0]);
         end
         n_insts = 10;
         return;
      end
      ...
      super.apply(ch, n_inst);
   endtask: apply
endclass: my_scenarios
```

PAGE 209

In Examples 4-95, there is an extra backslash that causes a syntax error.

Example 4-95 should be as follow:

Example 4-95. Mapping Virtual Tasks to Instance-Specific Module Tasks

```
`define utopia_mgmt(path) \
class \path.utopia_mgmt extends utopia_mgmt; \
   ... \
   virtual task read(input  [11:0] radd, \
                     output [ 7:0] rdat); \
      path.read(radd, rdat); \
   endtask: read \
   ... \
endclass
```

In Examples 4-96, there are two extra semi-colons that cause a syntax error.

Example 4-95 should be as follow:

Example 4-96. Instantiating Instance-Specific VMM-Compliant Interfaces

```
module tb_top;
...
utopia_mgmt_bfm host0(...);
utopia_mgmt_bfm host1(...);
...
endmodule

program test;
   `utopia_mgmt(tb_top.host0)
   `utopia_mgmt(tb_top.host1)

   class tb_env extends vmm_env;
      utopia_host host[2];
      ...
      virtual function void build();
         \tb_top.host0.utopia_mgmt host0 = new(...);
         \tb_top.host1.utopia_mgmt host1 = new(...);
         host[0] = host0;
         host[1] = host;
         ...
      endfunction
      ...
   endclass: tb_env
endprogram
```

PAGE 237

In Example 5-34, the type of the *n_insts* argument in the *apply()* task is missing the *unsigned* attribute.

The task *apply()* in Example 5-34 should be declared as follow:

Example 5-34. Multi-Stream Scenario Descriptor

```
class my_scenarios extends eth_frame_sequence;
   ...
   virtual task apply(eth_frame_channel channel,
                      ref int unsigned  n_insts);
      ...
   endtask: apply
endclass: my_scenarios
```

PAGE 252

In Example 5-52, a *return* statement in the *check()* function is missing the return value expression.

The function *check()* in Example 5-52 should read as follow:

Example 5-30. Using a Hashing Function to Locate Expected Response

```
function bit check(eth_frame actual)
   sb_where_to_find_frame where;
   eth_frame              q[$];
   eth_frame              expect;

   if (!index_tbl[hash(actual)].exists()) return 0;
   where = index_tbl[hash(actual)];
   q = sb.port[where.port_no].queue[where.queue_no];
   expect = q.pop_front();
   if (actual.compare(expect)) check = 1;
endfunction: check
```

PAGE 292

The identifier *I* used in property *S2* in Example 7-10 is not a known state identifier in Figure 7.2. Instead, the identifier *IDLE* should be used.

Property *S2* in Example 7-10 should read:

```
S2: assert property (@(clk)
      (state == IDLE) || (state == RZ) |-> !ack );
```

PAGE 312

In Example 8-5, a *return* statement in the *parse()* function is missing the return value expression.

The function *parse()* in Example 8-5 should read as follow:

Example 8-5. Parsing an Action Command

```
class ahb_from_file extends xvc_action;
   string fname;
   ...
   virtual function xvc_action parse(string argv[]);
```

```
            if (argv.size() != 2) return null;
            if (argv[0] != "read") return null;
            begin
               ahb_from_file act = new;
               act.fname = argv[1];
               parse = act;
            end
         endfunction: parse
         ...
      endclass: ahb_from_file
```

PAGE 365

The following class definition should be added:

VMM_VERSION

The class is used to report the version and vendor of the VMM Standard Library that is currently being used.

`function int major();`

Return the major version number of the implemented VMM Standard Library. Should always return 1.

`function int minor();`

Return the minor version number of the implemented VMM Standard Library. Should always return 0.

`function int patch();`

Return the patch number of the VMM Standard Library implementation. The return value is vendor-dependent.

`function string vendor();`

Return the name of the vendor supplying the VMM Standard Library implementation. The return value is vendor-dependent.

`function void display(string prefix = "");`

Display the version image returned by the *`psdisplay()`* method to the standard output.

`function string psdisplay(string prefix = "");`

Create a well-formated image of the VMM Standard Library implementation version information. The format is "<prefix>VMM Version <major>.<minor>.<patch> (<vendor>)".

PAGE 368

The following methods should be added:

`virtual function void set_name(string name);`
`virtual function void set_instance(string inst);`

Modify the name and instance name of the message service interface.

`virtual function void kill();`

Remove any internal reference to this message service insterface so it may be reclaimed by the garbage collection once all user references are also removed. Once this method has been invoked, it is no longer possible to control this message service interface by name.

PAGE 371

The first paragraph of the specification for the *`vmm_log::text()`* method refers to the *`vmm_log::format()`* method, which does not exist.

The first pagaraph should read:

`virtual function bit text(string msg = “”);`

Adds the specified text to the message being constructed. This method specifies a single line of message text and a newline character is automatically appended when the message issued. Additional lines of messages can be produced by calling this method multiple times, once per line. If an empty string is specified as message text, all previously specified lines of text are flushed to the output, but the message is not

terminated. This method may return FALSE if the message will be filtered out based on the text.

PAGE 372

The *vmm_warning()* macro in the third paragraph is missing a leading back tick (`).

The third paragraph should read:

For single-line messages, the *`vmm_fatal()*, *`vmm_error()*, *`vmm_warning()*, *`vmm_note()*, *`vmm_trace()*, *`vmm_debug()*, *`vmm_verbose()*, *`vmm_report()*, *`vmm_command()*, *`vmm_transaction()*, *`vmm_protocol()* and *`vmm_cycle()* macros can be used as a shorthand notation.

PAGE 374

The reference to the *vmm_enable_types()* method is erroneous.

The first paragraph of the *set_verbosity()* method should read:

Specify the minimum message severity to be displayed when sourced by the specified message service interface. See the documentation for the *enable_types()* method for the interpretation of the *name*, *instance* and *recursive* arguments and how they are used to specify message service interfaces.

PAGE 379

The last argument of the *format_msg()* method should be specified as *ref*.

The *format_msg()* method specification should read:

```
virtual function string format_msg(
                    string     name,
                    string     instance,
                    string     msg_typ,
                    string     severity,
                    ref string lines[$]);
```

PAGE 380

The last argument of the *continue_msg()* method should be specified as *ref*.

The *continue_msg()* method specification should read:

```
virtual function string continue_msg(
                    string     name,
                    string     instance,
                    string     msg_typ,
                    string     severity,
                    ref string lines[$]);
```

This method is called by all message service interfaces to format the continuation of a message on subsequent calls to the *vmm_log::end_msg()* method or empty *vmm_log::text("")* method call. The first call to the *vmm_log::end_msg()* method or empty *vmm_log::text("")* method use the *vmm_log_format::format_msg()* method.

PAGE 381

The *pre_abort()*, *pre_stop()* and *pre_debug()* callback methods should be functions instead of tasks.

The specification for these methods should read:

```
virtual function void pre_abort(vmm_log log);

virtual function void pre_stop(vmm_log log);

virtual function void pre_debug(vmm_log log);
```

PAGE 385

The *$cast_assign()* function in Example A-7 is an OpenVera function. The proper function to use in SystemVerilog is *$cast()*. Also, the polarity of the return value is improperly used.

In Example A-7, a *return* statement in the *copy()* function is missing the return value expression.

Example A-7 should read as follow:

Example A-7. Proper Implementation of the *vmm_data::copy()* Method

```
function vmm_data atm_cell::copy(vmm_data to = null)
   atm_cell cpy;

   if (to != null) begin
      if (!$cast(cpy, to)) begin
         `vmm_fatal(log, "Not a atm_cell instance");
         return null;
      end
   end else cpy = new;

   this.copy_data(cpy);
   cpy.vpi = this.vpi;
   ...
   copy = cpy;
endfunction: copy
```

PAGE 386

The second and third argument of the *byte_pack()* method should be specified as *input*. As specified, they are *ref*.

The *byte_pack()* method specification should read:

```
virtual function int unsigned byte_pack(
                  ref logic [7:0]    bytes[],
                  input int unsigned offset = 0,
                  input int          kind = -1);
```

PAGE 388

The third paragraph incorrectly states that the name of the class defined by the macro is "*<class_name>_chan*". The name of the class is "*<class_name>_channel*". The paragraph should read:

The implementation uses a macro to define a class named "*<class_name>_channel*" derived from the class named "*vmm_channel*" for any user-specified class named "*class_name*".

PAGE 388

The *vmm_channel()* macro cannot be terminated with a semi-colon.

The *vmm_channel()* macro specification should read:

```
`vmm_channel(class_name)
```

PAGE 401

The last argument in the *add_to_output()* method is named "*obj*", not "*dat*".

The last sentence of the first paragraph should read:

If the output channel is configured to use new descriptor instances, the *obj* parameter is a reference to that new instance.

PAGE 402

The last paragraph refers to the non-existent method *super.sched_from_input()*.

The last paragraph should read:

Any user extension of these methods should call *super.sched_on()* and *super.sched_off()*, respectively.

PAGE 403

The input property is missing for the last three arguments of the *get_object()* method. As specified, they are considered outputs.

The specification of the *get_object()* method should read:

```
virtual protected task get_object(
                      output vmm_data     obj,
                      input  vmm_channel  source,
```

```
                    input  int unsigned input_id,
                    input  int          offset);
```

PAGE 405

The *vmm_scheduler_election* class implements a round-robin election process by default. In its current form, turning it into a random election process requires that this class be extended. The following modifications will simplify this process: only the *default_round_robin* constraint blocks needs to be turned off.

The following class properties should read or be added:

int unsigned next_idx;
Value to assign to *source_idx* to implement a round-robin election.

rand int unsigned source_idx;
Index in the *sources* array of the elected source channel. An index of –1 indicates no election. The *vmm_scheduler_election_valid* constraint block constrains this property to be in the *0* to *sources.size()-1* range.

rand int unsigned obj_offset;
Offset, within the source channel indicated by the source_idx property, of the elected transaction descriptor within the elected source channel. This property is constrained to be equal to 0 in the *vmm_scheduler_election_valid* constraint block to preserve ordering of the input streams.

function void post_randomize();

This method is not used.

PAGE 407

The *vmm_notify::ONE_BLAST* configuration does not exist. The correct name is *vmm_notify::BLAST*.

The first paragraph should read:

(...) Otherwise, it returns an integer value corresponding to the current *vmm_notify::ONE_SHOT*, *vmm_notify::BLAST* or *vmm_notify::ON_OFF* configuration.

PAGE 410

The constructor in Example A-10 uses a C-like notation instead of the *function*/*endfunction* notation used in SystemVerilog.

The constructor in Example A-10 should read:

```
function new(vmm_notify notify,
             int        a,
             int        b);
   this.notify = notify;
   this.a      = a;
   this.b      = b;
endfunction: new
```

The *indicate()* task in Example A-10 uses an OpenVera notation to ignore the return value of a function instead of the notation used in SystemVerilog.

The *indicate()* task in Example A-10 should read:

```
virtual task indicate(ref vmm_data status)
   fork
      this.notify.wait_for(a);
      this.notify.wait_for(b);
   join
endtask
```

PAGE 415

The *$cast_assign()* function in the *foreach* loop is an OpenVera function. The proper function to use in SystemVerilog is *$cast()*. Also, the polarity of the return value is improperly used.

The *foreach* loop should read:

```
foreach (this.callbacks[i]) begin
   ahb_master_callbacks cb;
   if (!$cast(cb, this.callbacks[i])) continue;
   cb.ptr_tr(this, tr, drop);
end
```

PAGE 416

The *vmm_atomic_gen()* macro cannot be terminated with a semi-colon.

The *vmm_atomic_gen()* macro specification should read:

```
`vmm_atomic_gen(class_name, "Class Description)
```

PAGE 416

The following macro should be added:

```
`vmm_atomic_gen_using(class_name,
                      channel_type,
                      "Class Description")
```

Defines the same atomic generator class as the macro above, but using the specified output channel type, instead of a *<class_name>_channel* output channel. The generated class must be compatible with the specified channel type and both must exist.

This macro should be used only when generating instances of a derived class that must be applied to a channel of the base class.

PAGE 418

The *vmm_scenario_gen()* macro cannot be terminated with a semi-colon.

The *vmm_scenario_gen()* macro specification should read:

```
`vmm_scenario_gen(class_name, "Class Description)
```

PAGE 418

The following macro should be added:

```
`vmm_scenario_gen_using(class_name,
                        channel_type,
                        "Class Description")
```

Defines the same scenario generator class as the macro above, but using the specified output channel type, instead of a *<class_name>_channel* output channel. The generated class must be compatible with the specified channel type and both must exist.

This macro should be used only when generating instances of a derived class that must be applied to a channel of the base class.

PAGE 419

The following methods should be added:

```
function int unsigned get_n_insts();
function int unsigned get_n_scenarios();
```

The generator will stop after the first limit has been reached and only after entire scenarios have been applied. It can thus generate a few more instances or a few less scenarios than configures. These methods return the actual number of generated and applied instances and scenarios.

PAGE 423

The first paragraph of the specification for the *repeated* class property should read as follow:

```
rand int unsigned repeated;
```

Number of times the items in the scenario are repeated. A value of 0 indicates that the scenario is not repeated, hence is applied only once. The repeated instances in the scenario count toward the total number of instances generated but only one scenario is considered generated, regardless of the number of times it is repeated.

PAGE 442

The *xvc_xactor::wait_if_interrupted()* is specified as *protected*. Unfortunately, this would not allow the method to be called from the

xvc_action::execute() method as documented. Therefore this method is public.

The specification for the *xvc_xactor::wait_if_interrupted()* method should read:

task wait_if_interrupted();
Suspends the execution thread if an interrupt action is waiting to be executed by the XVC. This method must only be called from within an implementation of the *xvc_action::execute()* method.

PAGE 450

The syntax of the *COVFILE* directive should use "*filename*" instead of "*dbname*" to be consistent with the following paragraph.

The *COVFILE* directive specification should read:

```
COVFILE (<filename>|NONE)
```

PAGE 475

The *svSYS_CACHE_BLOCK_START* macro definition is missing and the definition for the *svSYS_CACHE_BLOCK_END* macro is duplicated.

The specification for the *svSYS_CACHE_BLOCK_START* and *svSYS_CACHE_BLOCK_END* macros should read:

svSYS_CACHE_BLOCK_START()
svSYS_CACHE_BLOCK_END()
Macros used to name a block of instructions that can be locked in the instruction cache.

```
#define svSYS_CACHE_BLOCK_START(name) \
   void name##_CacheBlockStart(void) {}

#define svSYS_CACHE_BLOCK_END(name) \
   void name##_CacheBlockEnd(void) {}
```