Making the Most of SystemVerilog and UVM

Hints and Tips for New Users

Dr Christoph Suehnel and Dr David Long

Doulos

September 12, 2013

SNUG Boston

Agenda

Introduction

SystemVerilog Newbie Errors

Avoiding UVM Newbie Errors

Overlooked Features That Enhance Verification

Conclusions

Introduction

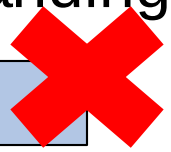
- Universal Verification Methodology (UVM)
  - The ASI Verification Standard since February 2011
  - Lots of companies adopting UVM
  - Lots of engineers need to learn a new methodology
- UVM is SystemVerilog
  - Lots of engineers need to learn a new language too!
- SystemVerilog and UVM are both complex
  - Lots of scope for new users to make mistakes
  - Useful features often get overlooked

SystemVerilog Newbie Errors

SystemVerilog Newbie Issues

- Basic coding errors/misunderstandings

```
assign a <= b;
```



- Working with multiple files
  - packages
  - Correct use `` `include ``, `` `define `` and `` `ifdef ``
- Understand Interfaces
  - modports
  - virtual interfaces

Illegal Package Imports

```
package pkg1;
  typedef struct { int a; int b; } my_struct;
endpackage: pkg1
```

```
package pkg2;
  typedef struct { int a; int b; } my_struct;
endpackage: pkg2
```

```
module top();
  import pkg1::*; //pkg1::my_struct potentially locally visible
  import pkg2::*; //importing name from 2 packages is illegal
  my_struct str; // ???
endmodule: top
```

Compiler directives will not help here!

Interfaces and Modports

```
interface intf(input logic clk, reset);
  import types::*;
  addr_t addr;
  rdata_t rdata;
  wdata_t wdata;
  logic RE;
  logic WE;
  modport mport(input clk, reset, output addr, input rdata,
                output wdata, RE, WE);
endinterface: intf
```

input modport is read by target

output modport is driven by target

SV Class Errors

- Confusion between class and object
- Forgetting to call new before accessing object
- Copying class reference instead of object
- Forgetting to call super.new in derived class
- Inappropriate use of inheritance

Avoiding UVM Newbie Errors

Components and Simulation Phase

- Choose correct base class
- Components should override phase methods
  - understand when to call methods from base class
  - map RTL features onto correct phase
- Differences between class and module
  - SVA `assert property` illegal in classes
  - `covergroup` instantiated in class constructor
  - no `always` or `initial` process in class

UVM Component Potential Errors

```
class my_agent extends uvm_agent;
  ...
  my_driver m_drv;
  my_sequencer r m_seqr;
  ...
  function void build_phase(uvm_phase phase);
    super.build();
    ...
    m_drv = my_driver::type_id::create("m_drv", this);
  endfunction

 function void connect_phase(uvm_phase phase);
    m_drv.seq_item_port.connect(m_sqr.seq_item_expor
  endfunction

  task run_phase(uvm_phase phase);
    ...
  endtask
...
endclass
```

should use this base class for all agents

is this correct? (required for auto-configuration)

Uses factory – do not create components with new()

Called after hierarchy constructed – do not try to connect ports in earlier phases!

Must be task (consumes simulation time)

UVM Macro Issues (1)

- Essential to use:

```
`uvm_object_utils(my_tx)
```

```
`uvm_component_utils(my_comp)
```

```
`uvm_component_param_utils(my_comp#(p))
```

- Recommended:

```
`uvm_info("MON","frame error",UVM_MEDIUM)
```

```
`uvm_info_context("Serial Seq",
                  req.convert2string(),
                  UVM_LOW,uvm_top)
```

UVM Macro Issues (2)

- Unnecessary?

```
`uvm_create(my_tx)
```

```
`uvm_rand_send(my_tx)
```

- Best Avoided?

```
`uvm_component_utils_begin(my_comp)
  `uvm_field_int(m_i,UVM_ALL_ON)
  `uvm_field_enum(OP,m_op,UVM_NOCOPY)
`uvm_component_utils_end
```

Configuration Problems

- Configuration Database
  - wildcards and regular expressions
  - search path starting point

```
class my_test extends uvm_test;
  ...
  uvm_config_db#(int)::set(this, "env.*", "m_bus_state", verif_env::IDLE);

  uvm_config_db#(string)::set(null, "*.m", "m_lookup", str_lookup);

  uvm_resource_db#(string)::set("my_comp", "m_lookup", str_lookup);
```

Use of Objections

- A component or sequence can object to a phase ending
- Two Rules
  1. Must raise at least 1 objection during 1st delta cycle
     - **Or simulation stops unexpectedly at time 0!**
  2. Every objection raised must be dropped by its owner
     - **Or simulation will not finish when expected!**

```
# UVM_ERROR verilog_src/uvm-1.1b/src/base/uvm_phase.svh(1210) @
920000000000ns: reporter [PH_TIMEOUT] Default phase
timeout of 920000000000ns hit. All processes are
waiting, indicating a probable testbench issue. Phase
'run' ready to end
# UVM_WARNING @ 920000000000ns: run [OBJTN_CLEAR] Object
'common.run' cleared objection counts for run
```

Objection Example

```
task run_phase(uvm_phase phase);
  master_sequence seq;  //top-level sequence
  seq = master_sequence::type_id::create();
  phase.raise_objection(this);
  //start top-level sequence and wait until it completes

  seq.start(m_env.m_sequencer);
  phase.drop_objection(this);
endtask: run_phase
```

Objection raised at 0ns

Sequence consumes time

Nothing more to do so drop objection

Synchronisation Differences

RTL Bus Functional Model

- Process triggered by clock edge
- Wait for time, condition or event
- Test calls tasks in BFM
- Multiple concurrent processes?

UVM Driver + Sequencer

- Interface timing sync'd to clocking block
- Also wait for uvm_event, uvm_barrier
- Driver pulls transactions from sequence
- Single task spawning dynamic processes?

Recommended UVM Templates

Synopsys Users Group

Pattern 1

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(...);
    ...
endclass
```

Pattern 2a

```
class my_tx extends uvm_sequence_item;
  `uvm_object_utils(my_tx)

  function new (string name = "");
    super.new(name);
  endfunction

  function string convert2string;
    ...
endclass
```

Pattern 2b

```
class my_seq extends uvm_sequence #(my_tx);
  `uvm_object_utils(my_seq)

  function new(string name = "");
    super.new(name);
  endfunction
  ...
  task body;
    ...
endclass
```

DOULOS

UVM Sequence Body Template

```
class my_seq extends uvm_sequence #(my_tx);
 `uvm_object_utils(my_seq)

 function new (string name = "");
  super.new(name);
 endfunction

 task body;
  repeat (6)
  begin
   req = my_tx::type_id::create("req");

   start_item(req);

   assert( req.randomize() with { data > 127; } );

   finish_item(req);
  end
 endtask

endclass
```

Pattern 2b

Factory-made transaction

Handshake with driver

Late randomization

UVM Test Class Template

```
class my_test extends uvm_test;
  `uvm_component_utils(my_test)
  ...
  my_env env;
  ...
  function void build_phase(uvm_phase phase);
    //
    my_tx::type_id::set_type_override(alt_tx::get_type() );

    my_config config = new;
    assert( config.randomize() );
    uvm_config_db #(my_config)::set(this, "*.*producer*", "config", config);

    env = my_env::type_id::create("env", this);
  endfunction

  task run_phase(uvm_phase phase);
    ...
  endtask
endclass
```

Pattern 1

Factory override replaces my_tx with alt_tx

Configuration object

Overlooked Features That Enhance Verification

String Matching with Regular Expressions

- Strings used to access UVM `config_db`
- `sprint` and `convert2string` return a string
- Global `uvm_is_match` uses regular expression to match a string

```
class cpu_scoreboard extends uvm_scoreboard;
  ...
  task run_phase(uvm_phase phase);
    bus_transaction tx_cpu, tx_iss;
    ...
    if ( uvm_is_match( "/BUS_READ|BUS_FETCH/",
                       tx_iss.convert2string() ) )  ....
```

Regular expression string argument

Rules for Regular Expressions

- String starts and ends with `'/'`
- Dot character `'.'` is a wildcard
- Characters may be grouped together e.g. `"(abc)"`
- Alternatives within a group are separated by `'|'`

`{n}`	`exactly n`
`{n1,n2}`	`between n1 and n2`
`{n, }`	`n or more`
`*`	`0 or more`
`+`	`1 or more`
`?`	`0 or 1`

Quantifiers for chars and groups

Adding Functionality with DPI

- Easy to enhance SystemVerilog functionality with C/C++ functions
- May require "wrapper" to match argument types
- Good match between simple SV types and C (array arguments accessed by DPI functions)
- Many simple cases
  - C source files added to **`vcs`** command line
- Complex examples
  - compile and link shared library using **`-sv_lib`** switch
  - use C/C++ compiler that comes with simulator!

DPI-C Example

```
#include <string>
#include <boost/regex.hpp>

extern "C"
bool regex_search (const char* re, const char* str)
{
        const boost::regex rexp(re);
        const std::string s1 = str;
        return boost::regex_search(s1,rexp);
}
```

C++ wrapper around standard Boost library function

```
import "DPI-C" function bit regex_search (string re, string str);

class cpu_scoreboard extends uvm_scoreboard;
  ...
run_phase(uvm_phase phase);
    ...
    if (regex_search("(\\w{3})_READ",tx_iss.convert2string()))
```

SystemVerilog importing C function via DPI

"Catching" reports

- Can configure actions associated with UVM report – very flexible
- Sometimes need greater control
  - action may depend on more complex conditions
  - might want to do additional/different tasks
- `uvm_report_catcher` is a callback to intercept generated reports
  - Use as base class for custom report catchers
  - Register with specific report handlers (`uvm_component`)

User-Defined Report Catcher

```
class catch_fatals extends uvm_report_catcher;
  function new(string name="catch_fatals");
    super.new(name);
  endfunction

  function action_e catch();
    string filter;
    //This changes all messages that contain search
    //pattern in catch_fatal plusarg to info
    if (uvm_cmdline_proc.get_arg_value("+catch_fatal=",filter) )
    begin
      if(get_severity() == UVM_FATAL &&
                    (uvm_is_match(filter,get_message())) )
        set_severity(UVM_INFO);
    end
    return THROW;
  endfunction: catch
endclass: catch_fatals
```

Override this function to catch report

Pass report to other registered catchers

Conclusions

Conclusions

- Novice errors can be reduced by following a simplified template
- UVM standard includes potentially useful features that are often overlooked
- Where SystemVerilog does not provide a feature you require, consider extending language with C/C++ libraries using DPI

Hardware Design

» VHDL » Verilog » SystemVerilog
» Altera » Microsemi » Xilinx

Embedded Systems and ARM

» C » C++ » UML » RTOS » Linux
» ARM Cortex A/R/M series

ESL & Verification

» SystemC » TLM-2.0 » SystemVerilog
» OVM/VMM/UVM » Perl » Tcl/Tk

Thank You