UVM Message Display Commands Capabilities, Proper Usage and Guidelines

Clifford E. Cummings

Sunburst Design, Inc.

cliffc@sunburst-design.com

www.sunburst-design.com

World-class Verilog, SystemVerilog & UVM Verification Training

Life is too short for bad or boring training!

Verilog `ifdef DEBUG

- Survey - how many have …
  - added **`ifdef DEBUG** to Verilog designs & testbenches?

Enabled by compiling with +define+DEBUG

- If you did not raise your hand, you …
  - Have never used Verilog
  - Have never done Verilog verification

snug
Synopsys Users Group

Introduction

Sunburst Design

- UVM verbosity settings are ***NOT*** message priority settings!

UVM Verbosity ≠ Message Priority !!

- **`UVM_LOW`** is not a *low* priority message
- **`UVM_LOW`** is *one of the highest priority messages !!*

- Reference sources and public examples … *get it wrong !!*

UVM User Guide
UVM Class Reference
+2 recent UVM books

- This paper offers guidelines on proper usage
- This paper shows useful messaging tricks

Agenda

Synopsys Users Group

Sunburst Design

- Paper — **Detailed descriptions**
  - Introduction
  - Verilog **`$display`** command
  - UVM reporting arguments
  - How to change verbosity settings
  - UVM messages & macros
  - UVM message guidelines
  - Simulation reporting goals
  - Changing simulation verbosities
  - Message catch & throw
  - **`get_type_name()`** command
  - Conditional verbosity printing
  - UVM documentation errors
  - Proposed extensions

- Presentation — **Highlights only** — **Lots of guidelines**
  - Introduction
  - Verilog **`$display`** command
  - UVM reporting arguments
  - How to change verbosity settings
  - UVM messages & macros
  - UVM message guidelines
  - Changing simulation verbosities
  - Message catch & throw — **Cool Trick #2**
  - Conditional verbosity printing — **Cool Trick #1**
  - UVM documentation errors
  - Proposed extensions

UVM Basic Printing Guidelines

- Printing command types
  - Verilog **`$display`** commands

Guideline: quit using `$display`
(quit using `$display` / `$write` / `$strobe`)

  - Messages & messaging macros

Guideline: replace `$display` commands with:
`` `uvm_info("id", "msg", UVM_MEDIUM) ``

  - **`UVM_LOW`**

`UVM_LOW` should almost *NEVER* be used

Widely misused in books and examples

  - **`convert2string`**

User-defined formatting

Guideline: override `convert2string` method in all data/transaction classes

`convert2string` becomes a built-in *"show_my_contents"* method

UVM Message Facilities

- **`$display`** - does not allow easy message filtering

- Built-in UVM message methods

```
uvm_report_* methods include:
  uvm_report_info   (...)
  uvm_report_warning(...)
  uvm_report_error  (...)
  uvm_report_fatal  (...)
```

- Built-in UVM message macros

```
`uvm_* macros include:
  `uvm_info   (...)
  `uvm_warning(...)
  `uvm_error  (...)
  `uvm_fatal  (...)
```

Messages & macros can be filtered many different ways

Message macros recommended by all vendors

`uvm_info/fatal* Macros

- UVM macros are more simulation efficient than messages

- UVM macros take 2-3 arguments, depending on macro type

```
string id
string message
```

Two string values: *"id"* and *"message"*

```
int verbosity
```

Only `uvm_info allows a verbosity setting

Default macro verbosities that cannot be changed:

```
`uvm_warning: UVM_NONE
`uvm_error:   UVM_NONE
`uvm_fatal:   UVM_NONE
```

Macros automatically include *file name* and *line number* *(good for debugging)*

```
task run;
  `uvm_info("run", "env still running", UVM_HIGH)
endtask
```

UVM Messaging Macro Advantages

- UVM message macros:
  - Are more simulation efficient
    - **Wraps `uvm_report_*` calls in an `if`-statement**
    - **More efficient than `uvm_report` methods**
    - **Removes expensive string processing if the verbosity setting would exclude the `uvm_report_*` calls**
  - Include **SystemVerilog-2009** **`__FILE__** and **`__LINE__** arguments
    - **Automatically reports file and line numbers - good for debugging**
    - **To turn off `FILE` and `LINE` info** → ***During Compilation:*** **use command line switch `+define+UVM_REPORT_DISABLE_FILE_LINE`**
  - **`uvm_warning/error/fatal** include pre-defined default **UVM_VERBOSITY** settings
    - **Avoids new-user mistakes** ***(like setting `uvm_report_error` verbosity to `UVM_HIGH`)***

UVM Message Verbosity

- What is verbosity?
  - *Highly verbose* simulations would show lots of messages
  - *Minimally verbose* simulations would only show important messages

- UVM has built-in enumerated type: **`uvm_verbosity`**
  - Defines standard verbosity levels for reports:

`UVM_NONE`	Report is always printed
`UVM_LOW`	Report if selected verbosity is **`UVM_LOW`** or higher
`UVM_MEDIUM`	Report if selected verbosity is **`UVM_MEDIUM`** or higher
`UVM_HIGH`	Report if selected verbosity is **`UVM_HIGH`** or higher
`UVM_FULL`	Report if selected verbosity is **`UVM_FULL`** or higher
`UVM_DEBUG`	Report if selected verbosity is **`UVM_DEBUG`** or higher

Cannot be disabled by verbosity level setting

Lower verbosity = fewer messages

Higher verbosity = more messages

Reporting Command Arguments

Many reporting commands take reporting arguments:

- **uvm_severity**

  UVM_INFO UVM_WARNING UVM_ERROR UVM_FATAL

- **uvm_action**

  UVM_NO_ACTION ← Plus 7 more actions (See UVM Class Reference)

- **uvm_verbosity**
  - Verbosity settings with corresponding integer values:

These print by default

```
UVM_NONE      = 0
UVM_LOW       = 100
UVM_MEDIUM    = 200
UVM_HIGH      = 300
UVM_FULL      = 400
UVM_DEBUG     = 500
```

UVM_NONE = 0 ← Highest priority messages

UVM_MEDIUM = 200 ← UVM_MEDIUM is the default verbosity

UVM_DEBUG = 500 ← Lowest priority messages

UVM Message Guidelines

- Quit using the **`$display`** command!
- Use the message macros, not the message methods
- Use **`` `uvm_info("id", "msg", UVM_NONE) ``** for most important messages
- Use **`` `uvm_info("id", "msg", UVM_LOW) ``** for very important messages
- Use **`` `uvm_info("id", "msg", UVM_MEDIUM) ``** as your new default **`$display`**
- Use **`` `uvm_info("id", "msg", UVM_HIGH) ``** to display semi-useful information
- Use **`` `uvm_info("id", "msg", UVM_FULL) ``** for *testbench* status messages
- Use **`` `uvm_info("id", "msg", UVM_DEBUG) ``** for debug messages
- Use **`` `uvm_fatal("id", "msg") ``**
  `` `uvm_error("id", "msg") ``
  As appropriate
- Use **`` `uvm_warning("id", "msg") ``** ***very sparingly*** ← **They won't shut-up !!**
- Add the **`convert2string()`** method to all of your transaction data classes
- Project and IP providers should implement an intelligent **`"ID"`** scheme
  Helps users modify severities and mask unwanted messages

UVM Message Guidelines

More Usage Details

`uvm_info("id", "msg", *verbosity*) - how to select *verbosity*:

- **UVM_NONE** - NEVER filtered - test-passing messages
- **UVM_LOW** - rarely filtered - block-level test-passing messages.
- **UVM_MEDIUM** - your new default **$display** command
  - messages displayed by default but easily disabled
- **UVM_HIGH** - messages that are shown occasionally
- **UVM_FULL** - design status and UVM status messages
- **UVM_DEBUG** - debug messages added to design or testbench

- **`uvm_warning("id", "msg")** - ***use very RARELY***
  Unfortunately, warnings cannot be disabled using verbosity settings
  `uvm_warning macro uses **UVM_NONE**

Useful Tricks

Testbench & Factory Debugging

Unconditional Printing

- Good technique to view testbench and factory setup

```
class test1 extends uvm_test;
  ...

  function void start_of_simulation_phase(uvm_phase phase);
    super.start_of_simulation_phase(phase);
    this.print();
    factory.print();
  endfunction

  ...
endclass
```

start_of_simulation phase
(after the testbench is built and connected)

Add this code to print out the testbench structure

Add this code to print out the factory entries and overrides

PROBLEM: these printouts are unconditional
(not controlled by verbosity)

Could use `uvm_info(… this.sprint() …)

There is no factory.sprint()

Testbench & Factory Debugging

Verbosity-Controlled Printing

Cool Trick #1

- *Better* technique to view testbench and factory setup

```
class test1 extends uvm_test;
  ...

  function void start_of_simulation_phase(uvm_phase phase);
    super.start_of_simulation_phase(phase);

    if (uvm_report_enabled(UVM_HIGH)) begin
      this.print();
      factory.print();
    end
  endfunction

  ...
endclass
```

start_of_simulation phase
(after the testbench is built and connected)

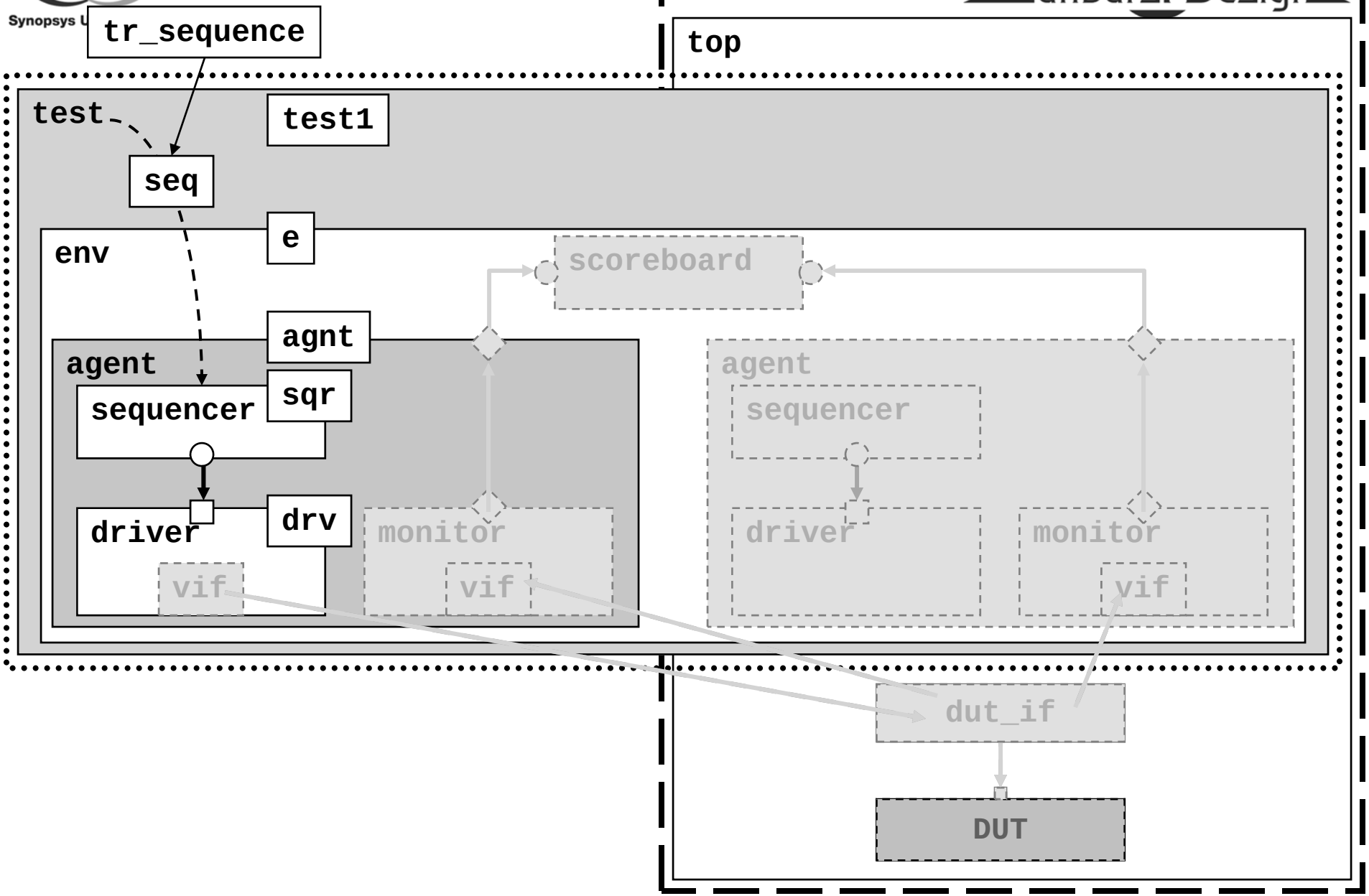
Conditionally **execute *.print() commands when verbosity=UVM_HIGH or higher**

Print testbench structure and factory entries

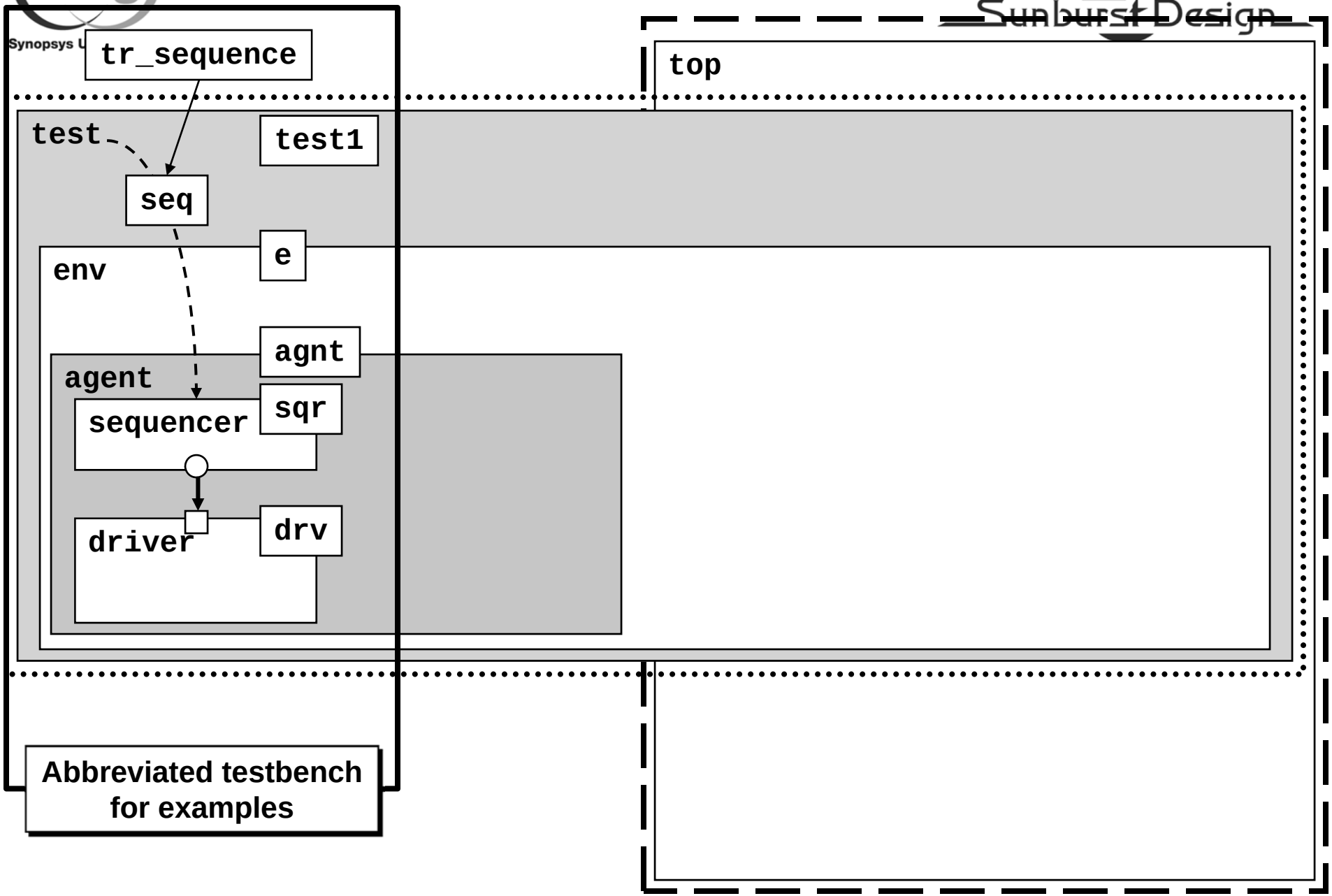
Allows conditional printing based on verbosity

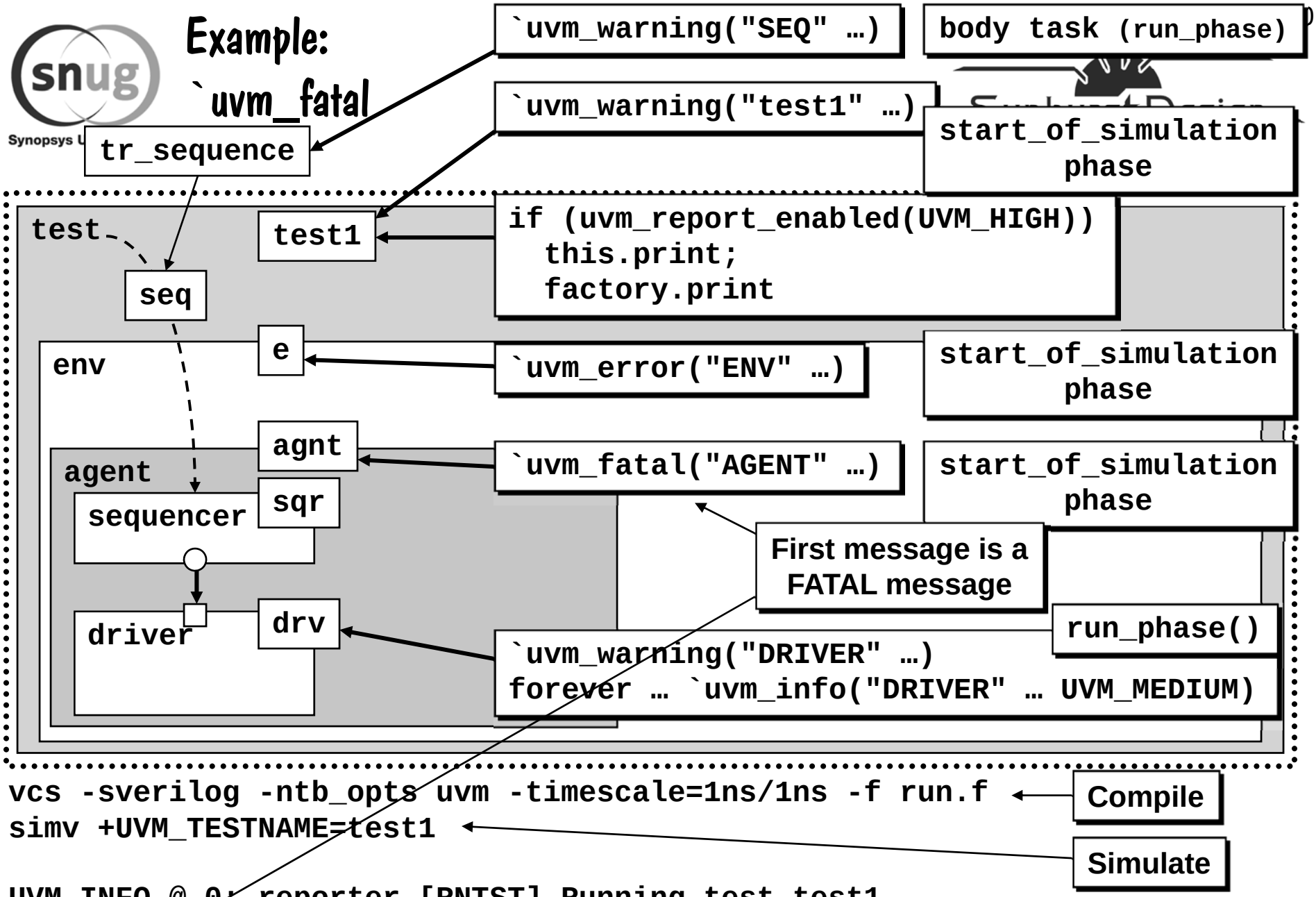
Examples

Typical UVM Testbench



Testbench Used With Examples




Example:
`uvm_fatal
`uvm_warning("SEQ" …)
body task (run_phase)
`uvm_warning("test1" …)
start_of_simulation phase
tr_sequence
test
test1
if (uvm_report_enabled(UVM_HIGH))
this.print;
factory.print
seq
env
e
`uvm_error("ENV" …)
start_of_simulation phase
agent
agnt
`uvm_fatal("AGENT" …)
start_of_simulation phase
sequencer
sqr
First message is a FATAL message
driver
drv
run_phase()
`uvm_warning("DRIVER" …)
forever … `uvm_info("DRIVER" … UVM_MEDIUM)
vcs -sverilog -ntb_opts uvm -timescale=1ns/1ns -f run.f
Compile
simv +UVM_TESTNAME=test1
Simulate
UVM_INFO @ 0: reporter [RNTST] Running test test1...
UVM_FATAL tb_agent.sv(34) @ 0: uvm_test_top.e.agnt [AGENT] agnt fatal msg

Message Catch & Throw

Cool Trick #2

- UVM built-in callback: **`uvm_report_catcher`**

  ← Override `catch()` method

- Get current test state using **`catch()`** method:

  `get_severity`
  `get_verbosity`
  `get_id`
  `get_message`

  → Catch some test state from the message

- **`catch()`** can execute the following:

  `set_severity` → Change severity of the message to severity
  `set_verbosity` → Change verbosity of the message to new verbosity
  `set_message` → Change text of the message to new message
  `set_action` → Change action of the message to new action

UVM_FATAL Demoter Example

catch() & THROW

Step #1: Extend the uvm_report_catcher callback class

Step #1a: Override catch()method

Step #2: Use the uvm_report_cb to add your code to the env from your test *(next slide)*

```
class test1_demoter extends uvm_report_catcher;
  ...

  function action_e catch();
    if(get_severity() == UVM_FATAL) begin
      set_severity(UVM_ERROR);
      `uvm_info("demoter",
                "Caught FATAL / demoted to ERROR",
                UVM_MEDIUM)
    end
    return THROW;
  endfunction
endclass
```

Extend uvm_report_catcher class

Override catch() method

catch() UVM_FATAL

set_severity() UVM_ERROR

THROW the new severity

UVM_FATAL Demoter Example

test1x adds demoter

```
class test1x extends test1;
  ...

  test1_demoter demoter;
  ...

  function void build_phase(uvm_phase phase);
    demoter = test1_demoter::type_id::create("demoter");
    uvm_report_cb::add(e, demoter);
    super.build_phase(phase);
  endfunction
endclass
```

env e inherited from test1 class

Declare test1_demoter handle called demoter

Create demoter

add demoter to environment callback

Step #2:

Test #1 has a fatal message

Test #1x catches fatal messages and throws corresponding error messages

Simulation output on the next slide

UVM_FATAL Demoter Example

test1 & test1x output

test1 output

```
UVM_INFO @ 0: reporter [RNTST] Running test test1...
UVM_FATAL tb_agent.sv(34) ... top.e.agnt [AGENT] agnt fatal msg
```

test1x output

```
UVM_INFO @ 0: reporter [RNTST] Running test test1x...
UVM_INFO test1_demoter.sv(23) ...top.e.agnt [demoter] Caught FATAL / demot
UVM_ERROR tb_agent.sv(34) ...top.e.agnt [AGENT] agnt fatal msg
UVM_ERROR env.sv(15) ...top.e [ENV] env error
UVM_WARNING test1.sv(27) ...top [test1] Test1 warning message
UVM_WARNING tb_driver.sv(21) ...top.e.agnt.drv [DRIVER] Starting tb_driver
UVM_WARNING tr_sequence.sv(26) ...top.e.agnt.sqr@@seq [SEQ] Running sequer
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=4d63  rst
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=b89f  rst
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=c105  rst
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=29ff  rst
UVM_WARNING tr_sequence.sv(26) ... top.e.agnt.sqr@@seq [SEQ] Running seque
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=ed0f  rst
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=749e  rst
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=6345  rst
UVM_INFO tb_driver.sv(24) ...top.e.agnt.drv [DRIVER] trans1: din=552a  rst
```

UVM Documentation Errors

Existing Documentation Problems

- UVM_LOW is pervasive in References, Books & Examples
  - **UVM User Guide**
    - Uses **`$display`** once
    - Uses 3 **`uvm_info`** macros with bugs in the examples
    - Uses 5 **`uvm_info`** macro examples with **UVM_LOW** - wrong verbosity
    - Uses 2 **`uvm_info`** macro examples without **UVM_LOW** - correct!
  - **UVM Class Reference**
    - Uses 1 **`uvm_info`** macro with bugs in the example
    - Uses 3 **`uvm_info`** macro examples with **UVM_LOW** - wrong verbosity
    - Uses 2 **`uvm_info`** macro examples without **UVM_LOW** - correct!
  - Meade/Rosenberg UVM Book
    - More than 20 examples improperly use **UVM_LOW**
  - Cooper UVM Book
    - More than 30 examples improperly use **UVM_LOW**

No wonder the UVM books get it wrong!

For low-priority messages

Annoyances & Enhancements

Request: Modify 2 Existing Macros

- **`uvm_warning** has fixed verbosity of **UVM_NONE**
- Enhancement:
  - Set a default verbosity of **UVM_NONE**
  - Allow user to override default verbosity

Keeps existing UVM_NONE default

Allow users to set the verbosity

- **`uvm_info** *REQUIRES* a verbosity setting
- Enhancement:
  - Set a default verbosity of **UVM_MEDIUM**
  - Prints by default if the user does not want to specify verbosity

Permits users to ignore adding common verbosity setting

- Requests are fully backward compatible

Request: Add 2 New Verbosities

- Add: **`UVM_INFO_PASS`** or **`UVM_INFO_SUCCESS`**:
  - Set at verbosity level 250
  - Off by default but turn on to view successful transaction activity

- Add: **`UVM_LIB_DEBUG`**
  - Set at verbosity level 600
  - To be used by UVM library developers
  - **`UVM_DEBUG`** to be used by users
  - Reason: Too many debug messages still floating around the UVM class library

Summary of Important Guidelines

Sunburst Design Usage Guidelines

Macro Type/Verbosity	Usage Guideline
`uvm_fatal(…)	***fatal*** - test-aborting errors
`uvm_error(…)	***errors*** that do not abort simulation
`uvm_warn (…)	*important warnings*
`uvm_info (… UVM_NONE)	use for final reports
`uvm_info (… UVM_LOW)	high priority messages
`uvm_info (… UVM_MEDIUM)	normal messages - replaces `$display`
`uvm_info (… UVM_HIGH)	use to print passing transactions conditionally print testbench & factory info
`uvm_info (… UVM_FULL)	use to print UVM status messages
`uvm_info (… UVM_DEBUG)	use to add debug messages

Non-maskable*

Use sparingly!

***Almost* always prints**

Almost* always *OFF

Acknowledgements

- Thanks to my reviewers and colleagues

Great reviews of the paper and presentation slides

  - Jeff Montesano and Jonathan Bromley of Verilab
  - Kevin Geiger, Verification AC at Synopsys

Free IEEE SystemVerilog-2012 LRM @
`http://standards.ieee.org/getieee/1800/download/1800-2012.pdf`

UVM Message Display Commands Capabilities, Proper Usage and Guidelines

Clifford E. Cummings

Sunburst Design, Inc.

cliffc@sunburst-design.com

www.sunburst-design.com

World-class Verilog, SystemVerilog & UVM Verification Training

Life is too short for bad or boring training!