UVM Tips and Tricks  
– Compile Time

Douglas L. Perry, Doulos
Constrained Random Verification

Constrained random stimulus

Checker

Does it work?

Design Under Test

Functional Coverage

Are we done?

Increase coverage

Constraints

11001001
01001010
00001001
01110110
01100110
01001001
01001110
000010
010011
000010
100100
001000
110010
000011

000010
010011
000010
100100
001000
110010
000011
Tests Versus Testbench

+UVM_TESTNAME=test2

Test writer

Verification specialist

Verification environment

DUT

From the command line

Tests define differences

Classes in a package

Modules
The Big Picture

Reusable verification environment

Scoreboard
Virtual sequence
Register Layer

Agent
Agent
Agent

DUT

Factory
cfg_db

Config
Sequencer
Monitor
Driver
Simulation Phases

- build
- connect
- end_of_elaboration
- start_of_simulation
- run
- extract
- check
- report
- final

- pre_reset
- reset
- post_reset
- pre_configure
- configure
- post_configure
- pre_main
- main
- post_main
- pre_shutdown
- shutdown
- post_shutdown
`uvm_object_utils Macro

`uvm_object_utils(bus_transaction)

- Used for transaction data
- What happens if missing?

`uvm_object_utils(Bus_transaction)

Wrong class

`uvm_component_utils(bus_transaction)

Wrong macro
class serial_transaction extends uvm_sequence_item;

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

    rand bit [7:0] data;
    rand bit parity_error;
    rand int unsigned idle_delay;

    constraint never_generate_error { parity_error == 0; }

    `uvm_object_utils(serial_transaction)

endclass : serial_transaction

Base transaction class will not generate transactions with bad parity

Register transaction type with factory and use default behavior
\`uvm_component_utils Macro

\`uvm_component_utils(simple_bus_agent)

- Used for verification components
- What happens if missing?

\`uvm_component_utils(my_bus_agent)

Wrong class

\`uvm_object_utils(simple_bus_agent)

Wrong macro
class serial_monitor extends uvm_monitor;

`uvm_component_utils(serial_monitor)

uvm_analysis_port #(serial_transaction) a_port;
virtual serial_if.monitor vif;

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

function void build_phase(uvm_phase phase);
    a_port = new("a_port", this);
endfunction : build_phase

task run_phase(uvm_phase phase);

    serial_transaction tr;
    tr = serial_transaction::type_id::create("tr");
Component Class Constructor

• Part of UVM base classes
• Creates object
• Adds component into component hierarchy
• What happens when missing?
• Wrong parent?

function new(string name, uvm_component parent);
    super.new(name, parent);
endfunction : new
Inconsistent Instance Name

My_driver = simple_bus_driver::type_id::create("m_driver", this);

Best if names match

m_driver = simple_bus_driver::type_id::create("m_driver", this);

SystemVerilog variable name

UVM component name
function void build_phase(uvm_phase phase);
  m_serial_agent = serial_agent::type_id::create
      ("m_serial_agent", this);
  m_arb_bus_agent = arb_bus_agent::type_id::create
      ("m_arb_bus_agent", this);
  m_bus_subscriber = bus_subscriber::type_id::create
      ("m_bus_subscriber", this);
  m_ser_subscriber = serial_subscriber::type_id::create
      ("m_ser_subscriber", this);
endfunction

... 
endclass: my_env

// Test
function void start_of_simulation_phase(uvm_phase phase);
  serial_agent::type_id::set_type_override( bad_parity::get_type() );
endfunction
Not Using Factory

function void build_phase(uvm_phase phase);
    m_serial_agent = new("m_serial_agent", this);
    m_arb_bus_agent = new("m_arb_bus_agent", this);
    m_bus_subscriber = new("m_bus_subscriber", this);
    m_ser_subscriber = new("m_ser_subscriber", this);
endfunction

function void connect_phase(uvm_phase phase);
    m_arb_bus_agent.a_port.connect(m_bus_subscriber.analysis_export);
    m_serial_agent .a_port.connect(m_ser_subscriber.analysis_export);
endfunction

endclass: my_env

// Test
function void start_of_simulation_phase(uvm_phase phase);
    serial_agent::type_id::set_type_override( bad_parity::get_type() );
endfunction

Created using new()
Can't override!
Phase Errors

- Phases separate actions in time
- Build is top down
  - Cannot access lower level components until build_phase is complete
- Connect is bottom up
  - Low level connections cannot access higher connections until connect_phase is complete
Phase Errors Example

class my_test1 extends uvm_test;
...
function void build_phase(uvm_phase phase);
...
uvm_factory factory = uvm_factory::get();
factory.print();  
uvm_top.print_topology();
...

Too soon, lower hierarchy not built yet

class my_test1 extends uvm_test;
...
function void end_of_elaboration_phase(uvm_phase phase);
...
uvm_factory factory = uvm_factory::get();
factory.print();
uvm_top.print_topology();
...

OK, build_phase has completed
Bad Component Hierarchy

\[
m_{\text{driver}} = \text{simple\_bus\_driver::type\_id::create}("m\_driver", \text{null});
\]
Incorrect!

\[
m_{\text{driver}} = \text{simple\_bus\_driver::type\_id::create}("m\_driver", \text{this});
\]
Correct
Field Macros and Transactions

• Scoreboards and other functionality require certain transaction methods
  – Transaction copy
  – Transaction compare
  – Transaction convert2string
  – Transaction pack/unpack, others

• Two ways to do this
  – Manually create methods
  – Field macros
class bus_transaction extends uvm_sequence_item;
  ...
  function void do_copy(uvm_object rhs);
  ...
  function bit do_compare(uvm_object rhs, uvm_comparer comparer);
  bus_transaction my_tr;
  $cast(my_tr, rhs);
  ...
  function string convert2string;
  ...
endclass;

tr2.copy(tr1);
match = tr2.compare(tr1);
mystring = tr.convert2string();
class bus_transaction extends uvm_sequence_item;
...
  `uvm_object_utils_begin(bus_transaction)
  `uvm_field_int(data, UVM_DEFAULT | UVM_HEX | UVM_NOCOMPARE)
  `uvm_field_int(addr, UVM_HEX | UVM_NOCOMPARE)
  `uvm_field_enum(bus_kind_t, kind, UVM_NOCOMPARE)
  `uvm_object_utils_end
...
endclass;

Must be the same

Turn off compare for fields that are manually compared to avoid "anding" results together

UVM_DEFAULT optional here
Transaction Field Macros

```verilog
class instruction extends uvm_sequence_item;
...
  rand logic [3:0] opcode;
  rand logic [3:0] src;
  rand logic [3:0] src2;
  rand logic [3:0] dst;

  `uvm_object_utils_begin(instruction)
    `uvm_field_int(opcode, UVM_DEFAULT)
    `uvm_field_int(src,    UVM_DEFAULT)
    `uvm_field_int(src2,   UVM_DEFAULT)
    `uvm_field_int(dst,    UVM_DEFAULT)
  `uvm_object_utils_end

  ...
endclass : instruction
```

Specify how to treat each field

Specify capabilities and radix

UVM_DEFAULT uses type specified by macro
super.build_phase()

- uvm_component::build_phase()
  - Calls apply_config_settings() to set values of matching fields
  - Values can be set by accident, you may be surprised!

```cpp
function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    m_bus_mon = arb_bus_monitor::type_id::create("m_bus_mon", this);
    m_iss = iss ::type_id::create("m_iss", this);
    m_scbd = cpu_scoreboard ::type_id::create("m_scbd", this);
```
Possible Wrong Value Retrieved

// Test
uvm_config_db#(int)::set(this, "*", "count", 1000);
...

class my_component extends uvm_component;
...
int count;
`uvm_component_utils_begin(my_component)
  `uvm_field_int(count, UVM_DEFAULT)
`uvm_component_utils_end
function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  ...
endfunction

Set "count" into config DB
"count" applied here
Sets count to 1000 automatically
TLM Port Usage

```c
// Monitor
uvm_analysis_port #(bus_transaction) a_port;

// ISS
uvm_analysis_imp #(bus_transaction, iss) bus_monitor_export;
...
function void write(bus_transaction t);

// Environment connect phase
m_bus_mon.a_port.connect(m_iss.bus_monitor_export);
```

- Could be done with events
- You have to debug
// Test
uvm_config_db#(uvm_bitstream_t)::set(null, "/.+arb_bus_agent/", "is_active", UVM_PASSIVE);
...

// Agent
uvm_config_db#(uvm_bitstream_t)::get(null, "/.+arb_bus_agent/", "is_active", is_active);
...
Config DB Setting

- **set**
  - "a.b.c" "foo" = value
  - "a.*" "name" = value1
  - "p.q.r" "foo" = value
  - "a.b.d.f" "name" = value2
  - "*.d.f" "name" = value3

- **get**
  - Searches tables for path & name
  - Take match nearest root of hier

- **(Global table)**
  - Actually its always "uvm_test_top"
  - The winner!

- **During build phase only**
  - Non-existent name

- **uvm_test**
  - "a"
    - uvm_env
      - "b"
        - uvm_agent
          - "c"
            - uvm_sequencer
          - "d"
            - uvm_agent
          - "e"
          - "f"
        - uvm_driver
      - "p.q.r"
Transactions Using Macros

```verilog
`uvm_do(req)

req = tx_type::type_id::create("req");

start_item(req);

if( !req.randomize() ) `uvm_error(...)

finish_item(req);
```
Sequence Macros

Macros

```cpp
`uvm_create(t)
`uvm_do(t)
`uvm_do_pri(t,pri)
`uvm_do_with(t,{con})
`uvm_do_pri_with(t,pri,{con})
```

Methods

```cpp
t = type::type_id::create(...);
start_item(t, pri);
t.randomize() with {con};
finish_item(t, pri);
```

(Transaction)
Sequence Macro Usage

task body;
   `uvm_do_with(req, { constr; })
endtask

task body;
   `uvm_create(req)
   `uvm_rand_send_with(req, { constr; })
endtask

task body;
   `uvm_create(req)
   if(!req.randomize() with { constr; }) ...
   `uvm_send(req)
endtask

equivalent
Sequencer Timing

Generate sequence items based on constraints

Provide sequence items when asked

Ask for sequence items and drive to DUT when it's ready
class serial_sequence extends uvm_sequence #(serial_transaction);
...

    task body;
        if (starting_phase != null) // UVM 1.1
            starting_phase.raise_objection(this);

        repeat (count)
            begin
                @DUT_ready
                `uvm_do( trans )
            end

        if (starting_phase != null)
            starting_phase.drop_objection(this);
    endtask

endclass : serial_sequence
class serial_driver extends uvm_driver #(serial_transaction);
  `uvm_component_utils(serial_driver)
  virtual serial_if.sender vif;
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction : new

  task run_phase(uvm_phase phase);
    serial_transaction tr;
    forever
      begin
        seq_item_port.get(tr);
        @DUT_ready
        drive_trans(tr);
        ...
      end
  endtask: run_phase

endclass: serial_driver
Are modports Required?

```vhdl
interface arb_bus_if();

  logic [ww-1:0] addr, dataw, datar;
  logic we, re, fe;
  logic req, gnt;
  logic clock, reset;

  ...

  modport fabric (  
    output datar, gnt, clock, reset,
    input  dataw, req, addr, we, re, fe
  );

  modport monitor (  
    input  clock, reset, datar, gnt,
    dataw, req, addr, we, re, fe
  );

endinterface : arb_bus_if
```

modport for DUT fabric
modport for testbench monitor
Objections

Verification env

Instruction generator
(uvm_sequencer)

class scoreboard

comparator

class collector

Coverage registers

subscriber

subscriber

class iss
Reference model

bus monitor

I object!

I object!
Objections Around Stimulus

// Test
task run_phase(uvm_phase phase);
    serial_sequence seq;
    uvm_object obj = phase.get_objection();

    obj.set_drain_time( this, 100ns );

    seq = serial_sequence::type_id::create("seq");
    phase.raise_objection(this);
    if (m_env.m_serial_agent.m_sequencer != null)
        seq.start(m_env.m_serial_agent.m_sequencer);
    phase.drop_objection(this);
endtask : run_phase

Does this guarantee that all transactions will run all the way through?
class my_driver extends uvm_driver #(my_transaction);
...

  task run_phase(uvm_phase phase);
  forever
  begin

    seq_item_port.get_next_item(tr);
    phase.raise_objection(this, "Driver busy");
    ... // Driver is busy
    seq_item_port.item_done();
    phase.drop_objection(this, "Driver idle");
  end
  endtask
Don't Propagate Objections

Helps with debug? Faster?

Propagate

Don't propagate

Driver 1 Sequencer 1 Driver 1 Sequencer 1
Turning Off Objection Propagation

```verilog
task run_phase(uvm_phase phase);
    uvm_objective objection = phase.get_objective();
    objection.set_propagate_mode(0);
    phase.raise_objective(this, "Start 1");
    seq.start(sequencer);
    phase.drop_objective(this, "End 1");
endtask
```

Turn off objection propagation

Minimizes simulation overhead
Summary

• UVM can be subtle
  – Create misunderstandings

• Eliminate hard to find errors
  – Follow tips

• Check out Easier UVM for tips as well

www.doulos.com/easier
Thank You!
UVM Tips and Tricks
- Runtime Tips

Presented by Srivatsa Vasudevan - Synopsys, Inc.

Slides by Srinivasan Venkataramanan, VerifWorks
UVM TB Hierarchy
Factory Pattern + Constraint Example

Program

Environment

Monitor components → scoreboard

Transactions

s2p_xactn

s2p_xactn_with_payload_pattern
Simulation

# UVM_INFO @ 0: reporter [RNTST] Running test factory_test...

# UVM_INFO @ 0: reporter [] Doing build

# UVM_INFO @ 0: uvm_test_top [FACTORY] Overriding s2p_xactn with pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>req</td>
<td>s2p_xactn_with_pld+</td>
<td>- exist</td>
<td>req@67</td>
</tr>
<tr>
<td>pkt_pld</td>
<td>integral</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
Multiple Overrides - Replace

```cpp
set_type_override_by_type(
    .original_type
        (s2p_xactn::get_type()),
    .override_type(
        s2p_err_xn::get_type()));

// ... some time later
set_type_override_by_type(
    .original_type
        (s2p_xactn::get_type()),
    .override_type(
        s2p_new_err_xn::get_type())
    .replace(1));
```

1st override

A while later..

Default - replace
Multiple Overrides - Replace

- Sample log
- ID: TPREGR

[TPREGR] Original object type 's2p_xactn' already registered to produce 's2p_err_xactn'. 
**Replacing** with override to produce type 's2p_new_err_xn'.
Multiple Overrides - Ignore

```cpp
set_type_override_by_type(
  .original_type
    (s2p_xactn::get_type()),
  .override_type(
      s2p_err_xn::get_type()));

// ... some time later
set_type_override_by_type(
  .original_type
    (s2p_xactn::get_type()),
   .override_type(
      s2p_new_err_xn::get_type())
  .replace(0));
```

1st override

A while later..

replace = 0
Multiple Overrides - Ignore

• Sample log
• ID: **TPREGRD**

[TPREGD] Original object type 's2p_xactn' already registered to produce 's2p_err_xactn'. Set 'replace' argument to replace the existing entry.
Replace in Instance Specific Override?

- No “replace” feature in instance specific API
- No warning/information from UVM BCL either!
Factory Debug

- UVM has built-in debug features for factory
- `uvm_factory::print()`
  - Not a static method though!

```c
virtual function void end_of_elaboration_phase (uvm_phase phase);
  uvm_factory f;
  f = uvm_factory::get();
  f.print();
  f.print(.all_types(0));
  f.print(.all_types(2));
```

Recommended
Factory Debug - Print

- all_types = 1 (Default)

```
### Factory Configuration (*)

Instance Overrides:

<table>
<thead>
<tr>
<th>Requested Type</th>
<th>Override Path</th>
<th>Override Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2p_xactn</td>
<td>uvm_test_top.env_0.agent0.sequencer.*</td>
<td>s2p_xactn_with_pld_pattern</td>
</tr>
<tr>
<td>s2p_xactn</td>
<td>uvm_test_top.env_0.agent0.sequencer.*</td>
<td>s2p_xactn_extra</td>
</tr>
</tbody>
</table>

No type overrides are registered with this factory

All types registered with the factory: 51 total
(types without type names will not be printed)

<table>
<thead>
<tr>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>factory_test</td>
</tr>
<tr>
<td>s2p_agent</td>
</tr>
<tr>
<td>s2p_base_test</td>
</tr>
<tr>
<td>s2p_driver</td>
</tr>
<tr>
<td>s2p_env</td>
</tr>
<tr>
<td>s2p_fcov</td>
</tr>
<tr>
<td>s2p_par_mon</td>
</tr>
<tr>
<td>s2p_rand_seq</td>
</tr>
<tr>
<td>s2p_scoreboard</td>
</tr>
<tr>
<td>s2p_sequencer</td>
</tr>
<tr>
<td>s2p_ser_mon</td>
</tr>
<tr>
<td>s2p_xactn</td>
</tr>
<tr>
<td>s2p_xactn_extra</td>
</tr>
</tbody>
</table>
```
Factory Debug - Print

- all_types = 0 (Recommended by VerifLabs, CVC)

```plaintext
### Factory Configuration (*)

Instance Overrides:

<table>
<thead>
<tr>
<th>Requested Type</th>
<th>Override Path</th>
<th>Override Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2p_xactn</td>
<td>uvm_test_top.env_0.agent0.sequencer.*</td>
<td>s2p_xactn_with_pld_pattern</td>
</tr>
<tr>
<td>s2p_xactn</td>
<td>uvm_test_top.env_0.agent0.sequencer.*</td>
<td>s2p_xactn_extra</td>
</tr>
</tbody>
</table>

No type overrides are registered with this factory

Overrides alone!
```
Factory Debug - Print

• all_types = 2 (Includes uvm_* too)

### Factory Configuration (*)

Instance Overrides:

- Requested Type       Override Path                        Override Type
  s2p_xactn             uvm_test_top.env_0.agent0.sequencer.*
  s2p_xactn             uvm_test_top.env_0.agent0.sequencer.*

No type overrides are registered with this factory

All types registered with the factory: 51 total types without type names will not be printed

<table>
<thead>
<tr>
<th>Type Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>factory_test</td>
</tr>
<tr>
<td>s2p_agent</td>
</tr>
<tr>
<td>s2p_base_test</td>
</tr>
<tr>
<td>s2p_xactn_extra</td>
</tr>
<tr>
<td>s2p_xactn_with_pld_pattern</td>
</tr>
<tr>
<td>uvm_exhaustive_sequence</td>
</tr>
<tr>
<td>uvm_mem_access_seq</td>
</tr>
<tr>
<td>uvm_mem_shared_access_seq</td>
</tr>
<tr>
<td>uvm_mem_single_access</td>
</tr>
<tr>
<td>uvm_mem_single_walk_seq</td>
</tr>
<tr>
<td>uvm_mem_walk_seq</td>
</tr>
<tr>
<td>uvm_object</td>
</tr>
<tr>
<td>uvm_random_sequence</td>
</tr>
<tr>
<td>uvm_recorder</td>
</tr>
<tr>
<td>uvm_reg_access_seq</td>
</tr>
<tr>
<td>uvm_reg_backdoor</td>
</tr>
<tr>
<td>uvm_reg_bit_bash_seq</td>
</tr>
<tr>
<td>uvm_reg_field</td>
</tr>
<tr>
<td>uvm_reg_hw_reset_seq</td>
</tr>
</tbody>
</table>

Overrides + Registered user types + UVM_*
Factory Undo? Anyone?

- Allowed in UVM 1.2
- Needs a work-around in UVM 1.1d
set_type_override_by_type(
    .original_type
      (s2p_xactn::get_type()),
    .override_type(
        s2p_err_xn::get_type()));

set_type_override_by_type(
    .original_type
      (s2p_xactn::get_type()),
    .override_type(
        s2p_xactn::get_type()));

A while later..
Undo – Error in UVM 1.1d

• Sample log
• ID: **TYPDUP**

[TYPDUP] Original and override type arguments are identical: s2p_xactn
uvm_component::new()

- Consistent, pre-defined prototype for `new()` for all components
  - `name` (string)
  - `parent` (uvm_component)
- All components – driver, sequencer, monitor, etc. **SHOULD** use this

```plaintext
class s2p_driver extends uvm_driver #(s2p_xactn);

function new(string name, uvm_component parent);
    super.new(name, parent);
endfunction : new
```
s2p_agent::build_phase()

- Use factory pattern – type_id::create()

```cpp
s2p_driver_0 = s2p_driver::type_id::create("s2p_driver_0", this);

s2p_sqr_0 = s2p_sequencer::type_id::create("s2p_sqr_0", this);
end
```

Instead of `new()`
Instance Paths - *name*

You’re going to call me WHAT!?
axi_fabric_env – How to Name It?

```cpp
virtual class axi_fabric_base_test extends uvm_test;
    axi_fabric_env axi_fabric_env_0;

    `uvm_component_utils(axi_fabric_base_test)

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

    extern virtual function void build_phase(uvm_phase);
endclass : axi_fabric_base_test

function void axi_fabric_base_test::build_phase(uvm_phase phase);
    super.build_phase(phase);
    axi_fabric_env_0 = axi_fabric_env::type_id::create(.name("CRAZY_ENV"),
                                                              .parent(this));
endfunction : build_phase
```

Crazy??

You’re going to call me WHAT!?
axi_fabric – Factory Overrides

```plaintext
function void axi_fabric_factory_test::build_phase(uvm_phase phase);
  uvm_factory f;
  super.build_phase (phase);
  set_inst_overide_by_type(
    .relative_inst_path("axi_fabric_env_0.*agent*0**.sequencer.*"),
    .original_type(axi_master_xactn::get_type()),
    .override_type(axi_hword_xactn::get_type()));
  set_inst_overide_by_type(
    .relative_inst_path("*.agent*1.axi_master_sequencer.*"),
    .original_type(axi_master_xactn::get_type()),
    .override_type(axi_word_xactn::get_type()));
  set_inst_overide_by_type(
    .relative_inst_path("CRAZY_ENV.*agent*2.axi_master_sequencer.*"),
    .original_type(axi_master_xactn::get_type()),
    .override_type(axi_byte_xactn::get_type()));
  uvm_factory::get();
  f.print(all_types(0));
endfunction : build_phase
```
Objections in UVM

- No news is good news
- By default no one objects!
  - Hence test ends at ZERO time if not taken care of!
Scalable End Of Test via “Objection”

http://www.cvcblr.com/blog/?p=414
virtual task rand_test::main_phase(uvm_phase phase);
    super.main_phase(phase);
    phase.raise_objection(this);
    s2p_seq1_0 = s2p_seq::type_id::create ("s2p_seq1");
    s2p_seq1_0.start(env_0.agent_0.s2p_sequencer_0);
    phase.drop_objection(this);
endtask : main_phase

http://www.cvcblr.com/blog/?p=414
UVM has built-in debug feature for Phases
vw_uvm_sim +UVM_PHASE_TRACE
Look for ID - [PH/TRC/*]

UVM_INFO @ 0: reporter
[PH/TRC/SKIP] Phase
'uvm.uvm_sched.main' (id=284)
No objections raised, skipping phase
Advanced Debug Methodology – Hangs?

```verilog
phase.drop_objection(this);
`uvm_info (get_name, "End of test", UVM_MEDIUM);
endtask:main_phase

task apb_test_1::dbg_eot(uvm_phase phase);
forever begin : fe
  #100;
  phase.phase_done.display_objections();
end : fe
endtask : dbg_eot
```

Built-in display_objections

http://www.cvcblr.com/blog/?p=681
The above indicates there are 3 folks opposing it with clear pointers to who they are.

http://www.cvcblr.com/blog/?p=681
Phase Jumping

- UVM allows phase jumps
  - Forward
  - Backward
- Handy if a test wants to “run” few times
- Multiple runs through:
  - `reset-cfg-main-shutdown`
  - `cfg-main-shutdown`
  - `reset-main`
Subsystem Test

- phase_ready_to_end callback

```cpp
extern virtual function void build_phase(uvm_phase phase);
extern virtual task reset_phase(uvm_phase phase);
extern virtual task configure_phase(uvm_phase phase);
extern virtual task main_phase(uvm_phase phase);
extern virtual task post_shutdown_phase(uvm_phase phase);
extern virtual function void phase_ready_to_end (uvm_phase phase);
endclass: vl_subsys_test
```
Subsystem Test

- Mimics reset, config via #delays
- Uses a virtual sequence in main_phase

```verilog
task vl_subsys_test::reset_phase(uvm_phase phase);
  `uvm_info (vl_id, "Start of Reset Phase ", UVM_MEDIUM)
  phase.raise_objected(this);
  #VL_RST_DEL;
  phase.drop_objected(this);
  `uvm_info (vl_id, "End of Reset Phase ", UVM_MEDIUM)
endtask : reset_phase

task vl_subsys_test::configure_phase(uvm_phase phase);
  phase.raise_objected(this);
  `uvm_info (vl_id, "Start of configure_phase", UVM_MEDIUM)
  #VL_CFG_DEL;
  phase.drop_objected(this);
  `uvm_info (vl_id, "End of configure_phase", UVM_MEDIUM)
endtask : configure_phase

task vl_subsys_test::main_phase(uvm_phase phase);
  phase.raise_objected(this);
  this.vseq_0.start(this.env_0.vseq_0);
  phase.drop_objected(this);
  `uvm_info ("CVC", "End of sample vseq test", UVM_MEDIUM)
endtask : main_phase

task vl_subsys_test::post_shutdown_phase(uvm_phase phase);
  phase.raise_objected(this);
  `uvm_info (vl_id, "Start of post_shutdown_phase", UVM_MEDIUM)
  #VL_SHUT_DEL;
  phase.drop_objected(this);
  `uvm_info (vl_id, "End of post_shutdown_phase", UVM_MEDIUM)
endtask : post_shutdown_phase
```
Innings Break!

```c
function void vl_subsys_test::phase_ready_to_end(uvm_phase phase);
    super.phase_ready_to_end(phase);
    if (phase.get_imp () == uvm_post_shutdown_phase::get()) begin : end_of_one_phase_run
        `uvm_info (vl_id, "Reached post_shutdown_phase, checking for num_jumps", UVM_MEDIUM)
        if (this.num_jumps > 0) begin : more_runs_needed
            `uvm_info (vl_id, "Jumping back to pre_reset_phase as num_jumps is: %d",
                this.num_jumps), UVM_MEDIUM)
            phase.jump (.phase(uvm_pre_reset_phase::get()));
            this.num_jumps--;
        end : more_runs_needed
        else begin : end_of_marathon
            `uvm_info (vl_id, "End of a marathon run with few jumps!", UVM_MEDIUM)
        end : end_of_marathon
    end : end_of_one_phase_run
endfunction : phase_ready_to_end
```
Second Innings

Match not over yet!

function void vl_subsys_test::phase_ready_to_end(uvm_phase phase);
    super.phase_ready_to_end(phase);
    if (phase.get_imp () == 'uvm_post_shutdown_phase::get()) begin : end_of_one_phase_run
        'uvm_info (vl_id, "Reached post_shutdown_phase, checking for num_jumps", UVM_MEDIUM)
        if (this.num_jumps > 0) begin : more_runs_needed
            'uvm_info (vl_id, $sformatf("Jumping back to pre_reset_phase as num_jumps is: %0d", this.num_jumps), UVM_MEDIUM)
            phase.jump (.phase(uvm_pre_reset_phase::get()));
            this.num_jumps--;
        end : more_runs_needed
        else begin : end_of_marathon
            'uvm_info (vl_id, "End of a marathon run with few jumps!", UVM_MEDIUM)
        end : end_of_marathon
    end : end_of_one_phase_run
endfunction : phase_ready_to_end

JUMP!
Phase Synchronization

• By default, all components must allow all other components to complete a phase before all components move to next phase

VIP 1: reset configure main shutdown

VIP 2: reset configure main shutdown

VIP 3: reset configure main shutdown

time
UVM Domain

- Domains are collections of components that must advance phases in unison
  - By default, no inter-domain synchronization

Durations:

1. **VIP 1:**
   - reset
   - configure
   - main
   - [shutdown]

2. **VIP 2:**
   - reset
   - configure
   - main
   - [shutdown]

3. **VIP 3:**
   - reset
   - configure
   - main
   - [shutdown]

**Domain A** and **Domain B** over time.
Phases - Guidelines

• Use correct phasing
• Use objections
• Learn UVM CLP for debug
• Advanced debug – display_objections
• Jumps are cool!
  – And useful 😊
• Can also use domains!
  – Advanced topic, call us again 😊
UVM Config DB
Configuration Database
Field Name vs. Value

- Typically field_name and value are kept the same
- Not a must, set and get must agree on the field_name
Field Name vs. Value

```plaintext
function void s2p_agent::build_phase(uvm_phase phase)
super.build_phase(phase);
if (!uvm_config_db#(uvm_active_passive_enum).get(this,
   "Mста_active",
   is_active)) begin : def_val_for_is_active
   `uvm_warning(get_name,$sformatf("No override for is_active: Using default is_active as:%s", this.is_active.name));
end : def_val_for_is_active

`uvm_info(get_name(),$sformatf("is_active is set to %s",this.is_active.name),UVM_MEDIUM);
```
Field Name vs. Value

function void s2p_env::build_phase(uvm_phase phase);
    super.build_phase(phase);
    agent0 = s2p_agent::type_id::create(.name("agent0"),
        .parent(this));

    uvm_config_db#(uvm_active_passive_enum)::set(.ctx(this),
        .inst_name("*"),
        .field_name("WAS_active"),
        .value(UVM_ACTIVE));

endfunction : build_phase
What if “set”, no “get”?

• Usually a BIG problem!
• Hard to detect bugs (TB bugs)
• uvm_component has handy debug functions for this!
check_config_settings

extern virtual task main_phase(uvm_phase phase);

function void start_of_simulation_phase(uvm_phase phase);
    this.check_config_usage();
endfunction : start_of_simulation_phase

class : rand_test

# --------------------------------------------------------------
# # 0.00 ns UVM_INFO | End Of Elaboration | reporter | Questa UVM | verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv : 272
# 0.00 ns UVM_INFO | :::: The following resources have at least one write and no reads :::: | uvm_test_top | CFGNRD |
# 0.00 ns UVM_INFO | num_masters [*/^uvm_test_top\.env_0\..*/]: (int) 3
# -
#     ----------
# # uvm_test_top.env_0 reads: 0 @ 0.00 ns writes: 1 @ 0.00 ns
# # # 0.00 ns UVM_INFO | Run Phase is Running ..
check_config_settings

```plaintext
num_masters
 Set (write)

num_masters [\^uvm_test_top\ env_0\.*$/] : (int) 3
 NO get (read)
```

```plaintext
# 0.00 ns UVM_INFO | Run Phase is Running ..
# 0.00 ns UVM_INFO | End Of Elaboration | reporter | Questa UVM | verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg_sv : 272
# 0.00 ns UVM_INFO | :::: The following resources have at least one write and no reads :::: | uvm_test_top | CFGNRD |
 : 0
# num_masters [^uvm_test_top\ env_0\.*$/] : (int) 3
# -
# _______
# uvm_test_top.env_0 reads: 0 @ 0.00 ns writes: 1 @ 0.00 ns
```
Config DB Auditing

```verilog

task rand_test::main_phase(uvm_phase phase);
  phase.raise_objectection(this);
  `uvm_info("Rand Test","Test is running...",UVM_LOW)

  uvm_config_db#(int):=set(.cntxt(this),
      .inst_name("*"),
      .field_name("num_jumps"),
      .value(5));

  this.print_config(.recurse(1), .audit(1));

  s2p_seq01 = s2p_rand_seq::type_id::create(.name("s2p_seq"),
                                             .parent(this));
  this.s2p_seq01.start(env_0.agent0.sequencer);
```
Sample Log of Audit

```
# | uvm_test_top.env_0.agent0.m_scoreboard | m_scoreboard | ../tb_src/s2p_scoreboard.sv : 69
# 0.00 ns UVM_INFO | Reset Phase is Running ....
# | uvm_test_top.env_0.agent0.driver | driver | ../tb_src/s2p_driver.sv : 63
# 150.00 ns UVM_INFO | Test is running... | uvm_test_top | Rand Test | ../tests/rand_test.sv : 59
# 150.00 ns UVM_INFO | visible resources: | uvm_test_top | CFGPRT | : 0
# <none>
# 150.00 ns UVM_INFO | visible resources: | uvm_test_top.env_0 | CFGPRT | : 0
# num_jumps [/^uvm_test_top\..*$/] : (int) 5
# -
# _________
# uvm_test_top reads: 0 @ 0.00 ns writes: 1 @ 150.00 ns
# #
# 150.00 ns UVM_INFO | visible resources: | uvm_test_top.env_0.agent0 | CFGPRT | : 0
# s2p_if_0 [/^uvm_test_top\..env_0\.agent0$/] : (virtual s2p_if) /top/s2p_if_0
# -
# _________
# reads: 0 @ 0.00 ns writes: 1 @ 0.00 ns
# uvm_test_top.env_0.agent0 reads: 1 @ 0.00 ns writes: 0 @ 0.00 ns
# #
# num_masters [/^uvm_test_top\..env_0\..*$/] : (int) 5
# -
# _________
# uvm_test_top.env_0 reads: 0 @ 0.00 ns writes: 1 @ 0.00 ns
# #
# num_jumps [/^uvm_test_top\..*$/] : (int) 5
# -
# _________
# uvm_test_top reads: 0 @ 0.00 ns writes: 1 @ 150.00 ns
# #
# is_active [/^uvm_test_top\..env_0\..*$/] : (enum bit uvm_pkg.uvm_resource.uvm_resource__12.convert2string
# m) UVM_ACTIVE
# -
```
Command Line Processor
<table>
<thead>
<tr>
<th>+UVM_MAX.Quit_COUNT=2</th>
<th>Quit after N number of errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default → No limit</td>
<td>Only aware of `uvm_error</td>
</tr>
<tr>
<td>SVA action blocks, DPI calls should use uvm_error</td>
<td></td>
</tr>
</tbody>
</table>
UVM CLP Options for Debug

- Built into UVM library at the major points of execution
- Dump important runtime data into log for debug or post-processing
- Can be activated from command line options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+UVM_PHASE_TRACE</td>
<td>Turns on tracing of phase execution</td>
</tr>
<tr>
<td>+UVM_OBJECTION_TRACE</td>
<td>Turns on tracing of objection activities</td>
</tr>
<tr>
<td>+UVM_RESOURCE_DB_TRACE</td>
<td>Turns on tracing of resource DB access (get &amp; set)</td>
</tr>
<tr>
<td>+UVM_CONFIG_DB_TRACE</td>
<td>Turns on tracing of configuration DB access</td>
</tr>
</tbody>
</table>
+UVM_CONFIG_DB_TRACE

SET-s

GET-s
+UVM_CONFIG_DB_TRACE

- **ID:** CFGDB/GET, CFGDB/SET

```
# UVM_INFO ../tb_src/s2p_fcov.sv(48) @ 5430: uvm_test
#
# ---- UVM Report Summary ----
#
# ** Report counts by severity
# UVM_INFO :   203
# UVM_WARNING :   0
# UVM_ERROR :   0
# UVM_FATAL :   0
# ** Report counts by id
# [CFGDB/GET]    141
# [CFGDB/SET]    2
# [Questa UVM]   3
# [RNTST]        1
# [Rand_TEST]    1
# [Rand_Test]    1
# [SBRD]         21
# [UVMTOP]       1
# [agent0]       1
# [driver]       9
# [get interface]  1
```
+UVM_OBJECTION_TRACE

• Displays “added objections” for raise_objection
+UVM_OBJECT_TRACE

- Displays “dropped objections” for drop_object
- ID: OBJTN_TRC

```c
# UVM_INFO @ 0: reset_object [OBJTN_TRC] Object uvm_test_top added 1 objection(s) to its total (raised from source object uvm_test_top.env_0.agent0.driver): count=0 total=1
# UVM_INFO @ 0: reset_object [OBJTN_TRC] Object uvm_top added 1 objection(s) to its total (raised from source object uvm_test_top.env_0.agent0.driver): count=0 total=1
# UVM_INFO @ 150: reset_object [OBJTN_TRC] Object uvm_test_top.env_0.agent0.driver dropped 1 objection(s): count=0 total=0
# UVM_INFO @ 150: reset_object [OBJTN_TRC] Object uvm_test_top/env_0.agent0.driver all dropped 1 objection(s): count=0 total=0
# UVM_INFO @ 150: reset_object [OBJTN_TRC] Object uvm_test_top/env_0.agent0 subtracted 1 objection(s) from its total (dropped from source object): count=0 total=0
# UVM_INFO @ 150: reset_object [OBJTN_TRC] Object uvm_test_top/env_0.agent0 subtracted 1 objection(s) from its total (all dropped from source object): count=0 total=0
# UVM_INFO @ 150: reset_object [OBJTN_TRC] Object uvm_test_top/env_0 subtracted 1 objection(s) from its total (dropped from source object): count=0 total=0
# UVM_INFO @ 150: reset_object [OBJTN_TRC] Object uvm_test_top/env_0 subtracted 1 objection(s) from its total (all dropped from source object): count=0 total=0
```

Popular, Simple Options

- `+UVM_TESTNAME=my_test`
  - Executes "class my_test extends from uvm_test"
  - Should be registered with factory
    - `\uvm_component_utils (my_test)`
    - Overrides value (if present) in code:
      - `run_test ("default_test")`
  - Only one test is run at a time
UVM_TESTNAME - Multiple

- Multiple values – ignored, warning issued
- `vw_uvm_sim +UVM_TESTNAME=first_test +UVM_TESTNAME=second_test`

Multiple (2) `+UVM_TESTNAME` arguments provided on the command line. 'first_test' will be used.

Provided list: first_test, second_test
Verbosity Setting

- `vw_uvm_sim +uvm_set_verbosity`

**Template**

```
vw_uvm_sim
+uvm_set_verbosity=<comp>,<id>,<verbosity>,<phase|time>,<offset>
```

**Example**

```
vw_uvm_sim
+uvm_set_verbosity=uvm_test_top.env0.agent1.*,_ALL_,UVM_FULL,time,800
```
Severity Override in CMD Line

• `vw_uvm_sim +uvm_set_severity`

**Template**

```plaintext
vw_uvm_sim
+uvm_set_severity=<comp>,<id>,<orig_severity>,<new_severity>
```

**Example**

```plaintext
vw_uvm_sim
+uvm_set_severity=uvm_test_top.env_0.*,SBRD,UVM_INFO,UVM_WARNING
```
Changing “action” on Messaging

- `vw_uvm_sim +uvm_set_action`

```
vw_uvm_sim
+uvm_set_action=<comp>,<id>,<severity>,<action[|action]>
```

**Template**

```
vw_uvm_sim
+uvm_set_action=uvm_test_top.env_0.*,_ALL_,UVM_INFO,UVM_NO_ACTION
```

**Example**
UVM CLP - Guidelines

• Change verbosity, severity, error count on CMD line
• Avoid factory overrides via CMD line
  – Hard to regress
  – Verification becomes spread across UVM, Perl/Scripts
Summary

• Runtime tricks & tips in UVM
• Several built-in hooks in UVM for run-time debug
• Look for our **DVRules** product to flag these @ http://www.verifworks.com
• Use our DVCreate tools to generate quality UVM
Thank You!
Accellera Standards Update
UVM and IEEE-1800.2

On behalf of the working group
Srivatsa Vasudevan
srivats@synopsys.com
What is happening?

- **UVM1.* Library**
- **Standard / LRM**
- **Library**

- **IEEE 1800.2 WG**

- **.. 2014**
- **2016..**
Deliverables

• IEEE 1800.2 WG
  – 1800.2 = SV focused
  – Defines UVM functional API (not an implementation)
  – Produces 1800.2 LRM
  – Allows for multiple implementations

• Accellera WG
  – Delivers UVM Library (SV) Reference Implementation matching 1800.2 LRM
  – Provide bug fixes for UVM library
Contribution options

• IEEE 1800.2 WG
  – For IEEE members every other week @ 9AM PST call
  – Tracking via accellera.mantishub.com → P1800.2

• Accellera WG
  – For Accellera members every other week @ 9AM PST call
  – Tracking via accellera.mantishub.com → UVM
General focus areas 1800.2

- Allow flexibility for future implementations
- Remove implementation artifacts from LRM
- Review API in the light of backward compatibility, consistency, simplification and extensibility
- Fully document and describe API
- (Note: Library can support deprecated API)
1800.2 REG Sub WG

• Identification and closure of register related issues
• Definition of necessary API to support future use models – e.g., SoC use models
• Further align reg sub system with other relevant standards (IP-XACT)
• Currently: Issue collection and prioritization
1800.2 TLM Sub WG

• Align UVM TLM with current IEEE 1666-2011 standardized TLM-1 and TLM-2.0 concepts
• Improve current UVM TLM standard documentation to explicitly explain execution semantics and underlying concepts
• Discuss completeness of current UVM TLM API
Thank You!