UVM Tips and Tricks
– Compile Time

Douglas L. Perry, Doulos
Constrained Random Verification

Constrained random stimulus

Checker

Does it work?

Design Under Test

Constraints

Increase coverage

Functional Coverage

Are we done?

Constrained random stimulus

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

000010
010011
000010
100100
001000
110010
000011
Tests Versus Testbench

Test writer
- test1
- test2
- test3

Verification specialist

Verification environment

DUT

From the command line
+UVM_TESTNAME=test2

Tests define differences

Classes in a package

Modules
The Big Picture

Reusable verification environment

Scoreboard
Virtual sequence
Register Layer
Agent
Agent
Agent
DUT

Factory
config_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**

- **Virtual seq**
  - **Subsc'r**
  - **Sequ'r**
  - **Seq**
  - **Monitor**
  - **Driver**
`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
serial_transaction

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");
  endtask : run_phase

Register component type with factory so that you can override later
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
Factory Usage

```cpp
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

```cpp
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
```

Can't override!

```cpp
// 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()
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_driver = simple_bus_driver::type_id::create("m_driver", null);

Incorrect!

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

mystring = tr.convert2string();

function string convert2string;
...  
endclass;
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;

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

Must be the same

UVM_DEFAULT optional here
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
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!

```verbatim
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);

Calls build_phase of uvm_component
```
// 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

• 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

Actually its always "uvm_test_top"

(set)

"a.b.c"  "foo" = value

The winner!

(set)

"a.*"  "name" = value1
"p.q.r"  "foo" = value

Non-existent name

(set)

"a.b.d.f"  "name" = value2

(set)

"*.d.f"  "name" = value3

get

Searches tables for path & name
Take match nearest root of hier
Transactions Using Macros

\`uvm_do(req)

```cpp
req = tx_type::type_id::create("req");
start_item(req);
if(!req.randomize()) `uvm_error(...)
finish_item(req);
```
Sequence Macros

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

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

`uvm_send(t)
`uvm_send_pri(t,pri)
`uvm_rand_send(t)
`uvm_rand_send_pri(t,pri)
`uvm_rand_send_with(t,{con})
`uvm_rand_send_pri_with(t,pri,{con})

(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

- Sequencer
  - Generate sequence items based on constraints
  - Provide sequence items when asked
  - Ask for sequence items and drive to DUT when it's ready

- Sequence

- Driver

- DUT
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

DUT timing belongs here in driver
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
Objections

![Diagram of verification environment with objects labeled as follows:
- Instruction generator (uvm_sequencer)
- class iss
- Reference model
- class scoreboard
- comparator
- class collector
- Coverage registers
- subscriber
- bus monitor

I object!

DIAGRAM SOURCE: 2016 Design and Verification Conference (DVCon) United States

2/29/2016 Douglas L. Perry, Doulos
Objections Around Stimulus

```vhdl
// Test
task run_phase(uvm_phase phase);
    serial_sequence seq;
    uvm_object obj = phase.get_object();

    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
```

Drain time after all objections dropped

Raise objection before stimulus start

Run sequence on sequencer

Drop objection after stimulus completed

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 Propogate Objections

Helps with debug?

Faster?

Propagate

Don't propagate
Turning Off Objection Propagation

task run_phase(uvm_phase phase);

    uvm_objection objection = phase.get_objection();

    objection.set_propagate_mode(0);

    phase.raise_objection(this, "Start 1");

    seq.start(sequencer);

    phase.drop_objection(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!