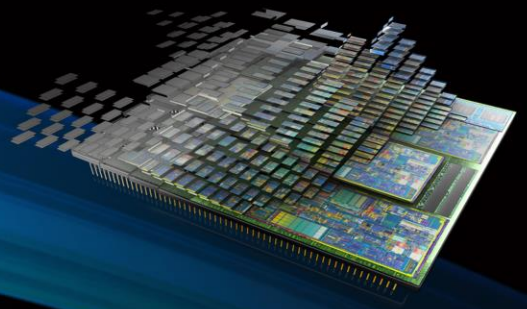
Verification Academy



UVM Basics

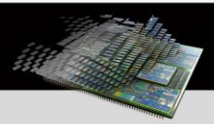
Sequences and Tests

Tom Fitzpatrick
Verification Evangelist

info@verificationacademy.com | *www.verificationacademy.com*

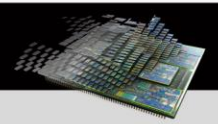
Mentor Graphics

Layered Sequential Stimulus



Drive transactions into DUT

Layered Sequential Stimulus



Constrained random
sequence of transactions

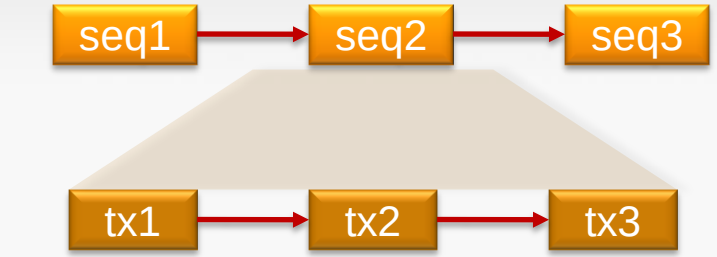
Drive transactions into DUT

Layered Sequential Stimulus

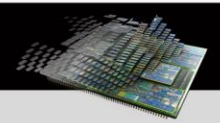
Nested, layered or virtual sequences

Constrained random sequence of transactions

Drive transactions into DUT

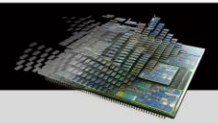


Sequence of Transactions



```
class read_modify_write extends uvm_sequence #(my_transaction);
  `uvm_object_utils(read_modify_write)
  function new (string name = "");
    super.new(name);
  endfunction: new

  task body;
    my_transaction tx;
    int a;
    int d;
    tx = my_transaction::type_id::create("tx");
    start_item(tx);
```

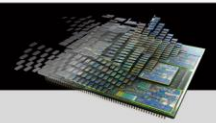
Sequence of Transactions



```
class read_modify_write extends uvm_sequence #(my_transaction);
  `uvm_object_utils(read_modify_write)
  function new (string name = "");
    super.new(name);
  endfunction: new

  task body;
    my_transaction tx;
    int a;
    int d;
    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() );
    tx.cmd = READ;
    finish_item(tx);
```

Read (use enum)

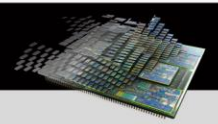
Sequence of Transactions



```
...
a = tx.addr;
d = tx.data;
++d;
```

Modify

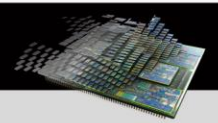
Sequence of Transactions



```
    ...
    a = tx.addr;
    d = tx.data;
    ++d;

    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() );

  endtask: body

endclass: read_modify_write
```

Randomize incidentals

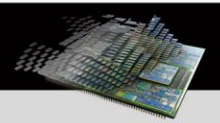
Sequence of Transactions



```
    ...
    a = tx.addr;
    d = tx.data;
    ++d;

    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() );
    tx.cmd =  WRITE; tx.addr =  a; tx.data =  d;
    finish_item(tx);
  endtask: body

endclass: read_modify_write
```

Write

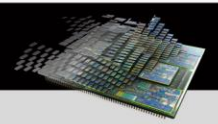
Sequence of Transactions



```
    ...
    a = tx.addr;
    d = tx.data;
    ++d;

    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() with {
       cmd == WRITE;    addr == a;    data == d; });
    finish_item(tx);
  endtask: body

endclass: read_modify_write
```

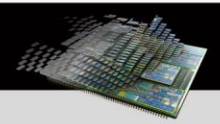
Randomize with constraints

Sequence of Sequences



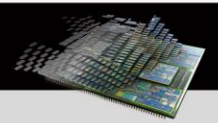
```
class seq_of_commands extends uvm_sequence
                              #(my_transaction);
  `uvm_object_utils(seq_of_commands)
  rand int n;
```

Sequence of Sequences



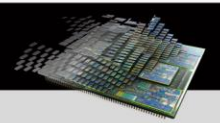
```
class seq_of_commands extends uvm_sequence
                              #(my_transaction);
  `uvm_object_utils(seq_of_commands)
  rand int n;
  constraint how_many { n inside {[2:4]}; }
```

Sequence of Sequences



```
class seq_of_commands extends uvm_sequence
                              #(my_transaction);
  `uvm_object_utils(seq_of_commands)
  rand int n;
  constraint how_many { n inside {[2:4]}; }
  ...

  task body;
    repeat(n)
    begin



    end
  endtask: body
```

Sequence of Sequences

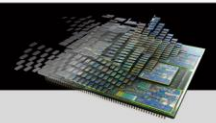


```
class seq_of_commands extends uvm_sequence
                              #(my_transaction);
  `uvm_object_utils(seq_of_commands)
  rand int n;
  constraint how_many { n inside {[2:4]}; }
  ...

  task body;
    repeat(n)
    begin
      read_modify_write seq;
      seq = read_modify_write::type_id::create("seq");
      seq.start( m_sequencer, this);
    end
  endtask: body
```

On sequencer

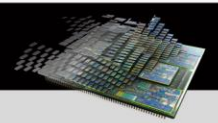
parent

A Set of Sequences



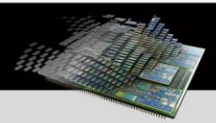
```
package my_sequences;
  import uvm_pkg::*;

  class read_modify_write extends uvm_sequence
                                  #(my_transaction);
    ...

  class seq_of_commands extends uvm_sequence
                                 #(my_transaction);
    ...

endpackage
```

Starting a Sequence



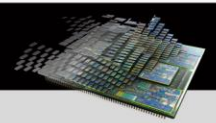
```
class test1 extends uvm_test;
  `uvm_component_utils(test1)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    read_modify_write seq;
    seq = read_modify_write::type_id::create("seq");
```

Starting a Sequence



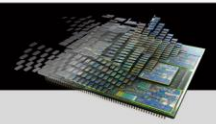
```
class test1 extends uvm_test;
  `uvm_component_utils(test1)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    read_modify_write seq;
    seq = read_modify_write::type_id::create("seq");

    seq.start( ...
```

Starting a Sequence



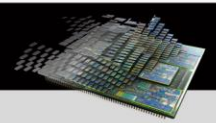
```
class test1 extends uvm_test;
  `uvm_component_utils(test1)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    read_modify_write seq;
    seq = read_modify_write::type_id::create("seq");

    seq.start( my_env_h.my_agent_h.my_sequencer_h );

  endtask
endclass
```

Starting a Sequence



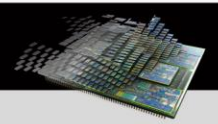
```
class test1 extends uvm_test;
  `uvm_component_utils(test1)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    read_modify_write seq;
    seq = read_modify_write::type_id::create("seq");
    phase.raise_objection(this);
    seq.start( my_env_h.my_agent_h.my_sequencer_h );
    phase.drop_objection(this);
  endtask
endclass
```

Starting a Sequence



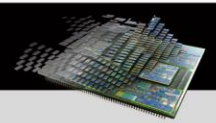
```
class test2 extends uvm_test;
  `uvm_component_utils(test2)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    seq_of_commands seq;
    seq = seq_of_commands::type_id::create("seq");
```

Randomizing a Sequence



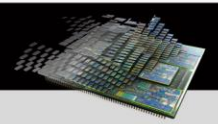
```
class test2 extends uvm_test;
  `uvm_component_utils(test2)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    seq_of_commands seq;
    seq = seq_of_commands::type_id::create("seq");

    assert( seq.randomize() );



  endtask
endclass
```

Randomizing a Sequence



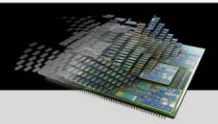
```
class test2 extends uvm_test;
  `uvm_component_utils(test2)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    seq_of_commands seq;
    seq = seq_of_commands::type_id::create("seq");

    assert( seq.randomize() );
    phase.raise_objection(this);

    phase.drop_objection(this);
  endtask
endclass
```

Randomizing a Sequence



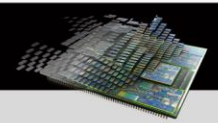
```
class test2 extends uvm_test;
  `uvm_component_utils(test2)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    seq_of_commands seq;
    seq = seq_of_commands::type_id::create("seq");

    assert( seq.randomize() );
    phase.raise_objection(this);
    seq.start( my_env_h.my_agent_h.my_sequencer_h );
    phase.drop_objection(this);
  endtask
endclass
```

Constraining a Sequence



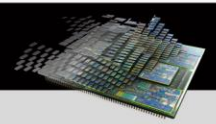
```
class test3 extends uvm_test;
  `uvm_component_utils(test2)

  my_env my_env_h;
  ...

  task run_phase(uvm_phase phase);
    seq_of_commands seq;
    seq = seq_of_commands::type_id::create("seq");
    seq.how_many.constraint_mode(0);
    assert( seq.randomize() with {seq.n > 10 && seq.n < 20;});
    phase.raise_objection(this);
    seq.start( my_env_h.my_agent_h.my_sequencer_h );
    phase.drop_objection(this);
  endtask
endclass
```

Selecting a Test



```
module top;
   ...

   initial
   begin: blk
      ...
      run_test("test3");
   end

endmodule: top
```

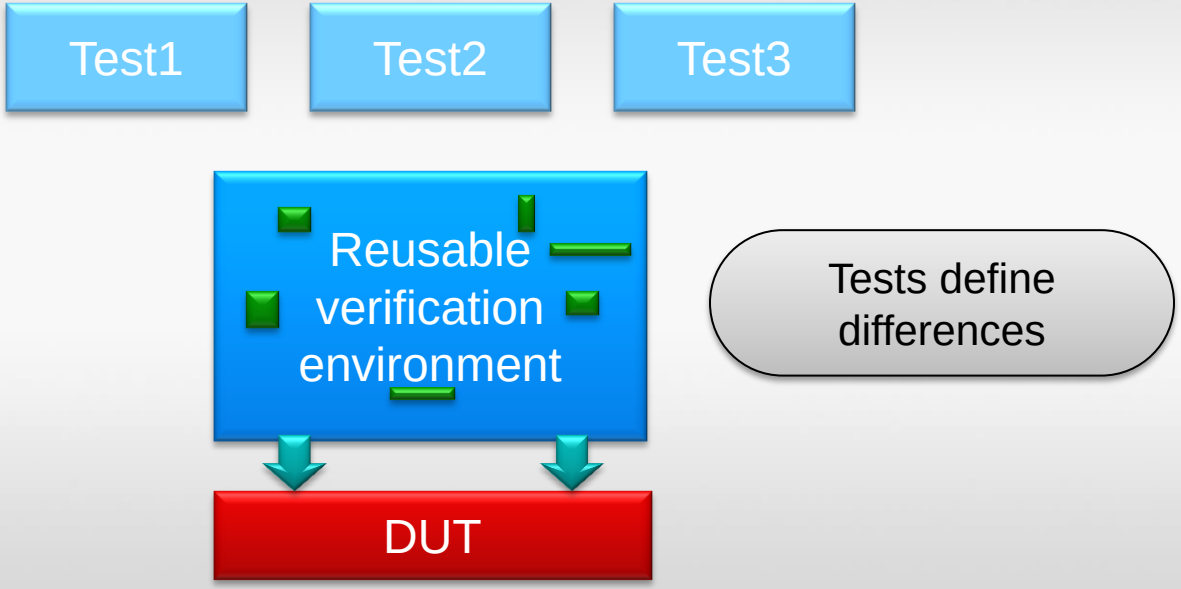
Selecting a Test



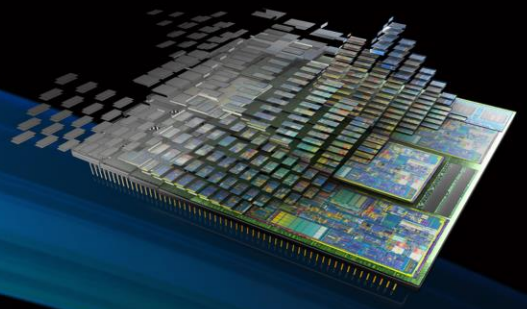
```
module top;
   ...

   initial
   begin: blk
      ...
      run_test();
   end

endmodule: top
```

Command line: `vsim +UVM_TESTNAME=test3`

Summary


Test1
Test2
Test3
Reusable verification environment
DUT
Tests define differences

Verification Academy



UVM Basics

Sequences and Tests

Tom Fitzpatrick
Verification Evangelist

info@verificationacademy.com | *www.verificationacademy.com*

Mentor Graphics