VERIFICATION ACADEMY

UVM Basics

Sequences and Tests

John Aynsley
CTO, Doulos

academy@mentor.com
www.verificationacademy.com

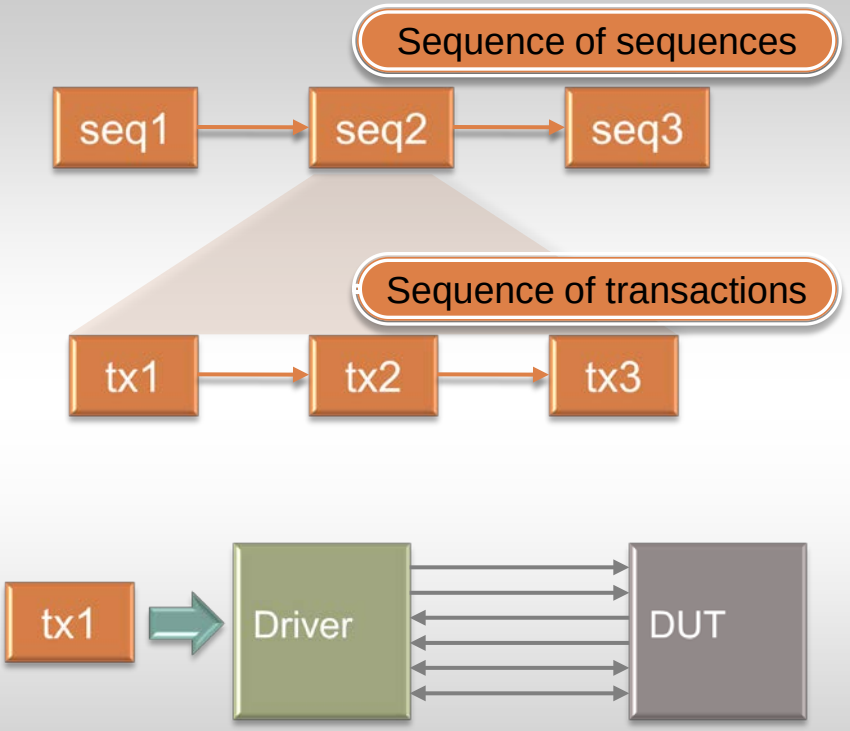
Mentor Graphics®

Layered Sequential Stimulus

Nested, layered or virtual sequences

Constrained random sequence of transactions

Transaction = command to driver



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 has no parent

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 = 0;
    finish_item(tx);
```

Read with random addr and data

Sequence of Transactions

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

Sequence of Transactions

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

    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    tx.cmd = 1;
    tx.addr = a;
    tx.data = d;
    finish_item(tx);
  endtask: body

endclass: read_modify_write
```

Write with same random addr and data

Sequence of Sequences

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

"Knob" – fully random, fully determined, or partially constrained

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");
      start_item(seq);
      finish_item(seq);
    end
  endtask: body
```

End-of-Test

```
task body;
  uvm_test_done.raise_objection(this);

  repeat(n)
  begin
    read_modify_write seq;
    seq = read_modify_write::type_id::create("seq");
    start_item(seq);
    finish_item(seq);
  end

  uvm_test_done.drop_objection(this);
endtask: body
```

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 );
```

A deterministic sequence

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() );

    seq.start( my_env_h.my_agent_h.my_sequencer_h );
```

Constraining a Sequence

```
class test3 extends uvm_test;

  `uvm_component_utils(test3)

  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; } );
    seq.start( my_env_h.my_agent_h.my_sequencer_h );
```

Selecting a Test

```
module top;
  ...

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

endmodule: top
```

Selecting a Test

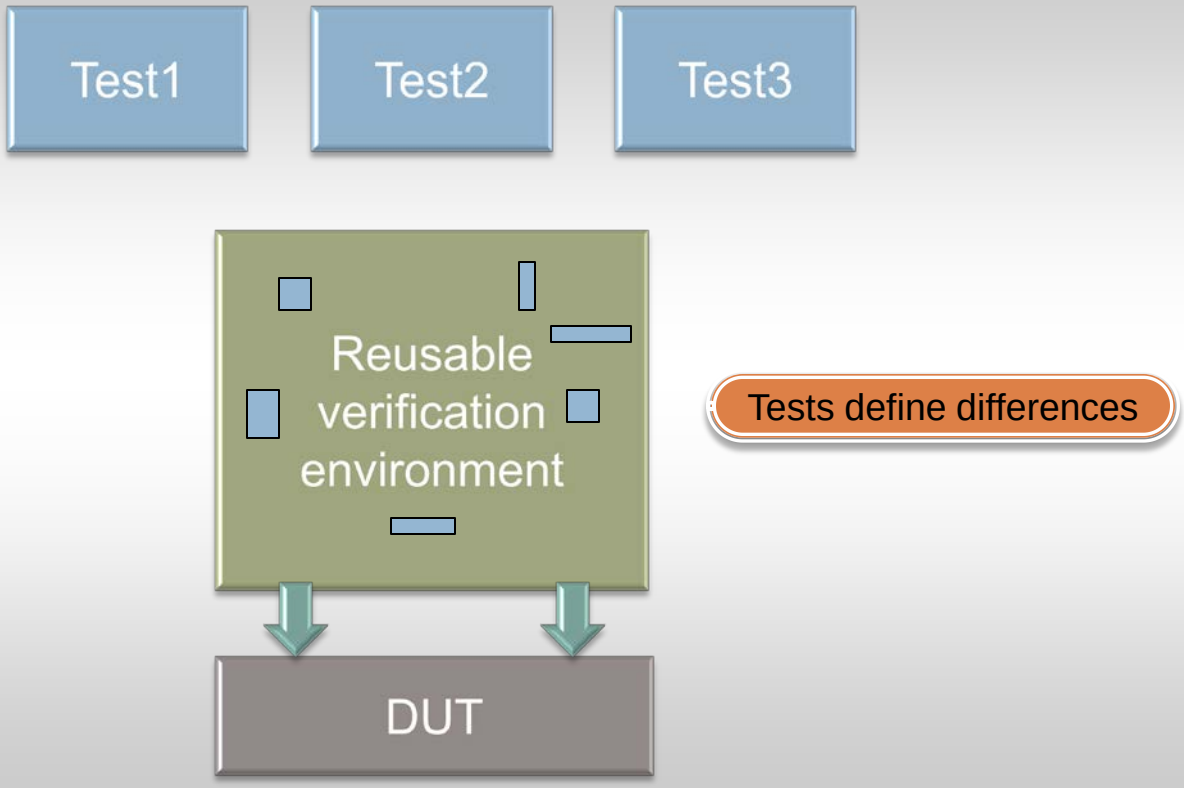
```
module top;
  ...

  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

John Aynsley
CTO, Doulos

academy@mentor.com
www.verificationacademy.com

Mentor Graphics®