April 2019

Saturday 13

Sequence items.

class Sequence extend uvm_Sequence_item;

rand bit [3:0] a;

:

rand bit [3:0] b;

function new (string path = "Sequence");

super.new (path);

endfunction

`uvm_object_utils_begin (Transaction)

`uvm_object_int (a, UVM_DEFAULT);

Sunday 14

`uvm_object_int (b, UVM_DEFAULT);

`uvm_object_utils_end

endclass

April 2019

15 Monday

Generator.

class generator extends uvm_Sequence#(transaction)

`uvm_object_utils(generator),

function new(input string Path = "generator")

super.new()

endfunction.

virtual task body()

repeat(15) begin.

tr = transaction::type_id::create("tr"),

start_item(tr);

assert(tr.randomize()).

16 Tuesday

`uvm_info("SEQ", $format("a:%0d b:%0d

c %0d d:%0d sel:%0d y:%0d", tr.a, tr.b,

tr.c,));

end

endtask

class driver extends uvm_driver#(transaction),

`uvm_component_utils(driver)

transaction tr.

virtual mux_if mif;


```
Virtual function void build_phase (uvm_phase phase);
super.build_phase (phase).
  if (!uvm_config_db#(virtual mux_if)::get
          (this, "", "mif", mif)
    `uvm_error ("drv", "unable to access interface");
endfunction.

virtual task run_phase (uvm_phase phase);
    tr = transaction::type_id::create("tr");
forever begin:
      seq_item_port.get_next_item(tr);
       mif.a <= tr.a
       .
       mif.sel <= tr.sel.
```

Thursday 18

```
`uvm_info ("DRV", $sformat("a=%0d
          b=%0d (=%0d, tr.a tr.b ...), UVM_NONE);
  seq_item_port.item_done();
  #20.
 end.
endtask.

endclass.
```

April 2019

19 Friday

Monitor

```
class mon extends uvm_monitor;
    transaction tr;
    virtual mux_if mif;
function new (input string inst = "mon",
                uvm_component Parent = null);
    super.new (inst, Parent);
endfunction

virtual function void build_phase (uvm_phase phase);
    super.build_phase (phase);
    tr = transaction :: type_id :: create ("tr");
    send = new ("send", this);
if (! uvm_config_db #(virtual mux_if) :: get
                (this, "", "mif", mif))
```

20 Saturday

```
    `uvm_error ("drv", "unable to access Interface");
endfunction

virtual task run_phase (uvm_phase phase);
forever begin
    #20;
    tr.a = mif.a
    .
    .
    tr.y = mif.y
`uvm_info ("mon_DUT", $sformat ("a : %0d
    ... y : %0d", tr.a ... tr.y), UVM_NONE)
send.wr (tr);
end
endtask
```

April 2019

Sunday 21

Reference_model

class ref_mod extend uvm_monitor;
// component macro
uvm_analysis_port #(transaction) send_rf;
transaction tr; virtual mux_if mif;
// create Constructor.
Virtual function void build_phase(uvm_phase phase);
super.build_phase(phase);
tr = transaction::type_id::create("tr");
send_rf = new("send rf", this);
if(!uvm_config_db #(virtual mux_if)::get(this,"",
"mif", mif));

Monday 22

`uvm_error("ref_model", "unable to access interface");`
endfunction
function void predict();
case (tr.sel)
2'b00 : tr.y = tr.a;
2'b01 : tr.y = tr.b;
endcase
predict();
`uvm_info("mon_ref", $sformatf("a = %0d
send_rf.write(tr);
end
endtask
endclass

April 2019

2019

23 Tuesday

Scoreboard.

class sco extend scoreboard.

transaction tr, trref.

`uvm_tlm_analysis_fifo # (transaction) sco_data.

`uvm_tlm_analysis_fifo # (transaction) sco_data_ref

// create constructor.

virtual function void build_phase (uvm_phase phase).

super.build_phase (phase).

tr = transaction :: type_id :: create ("tr")

tr_ref = transaction :: type_id :: create ("tr_ref");

sco_data = new ("sco_data", this);

sco_data_ref = new ("sco_data_ref", this)

endfunction

24 Wednesday

virtual task run_phase (uvm_phase phase);

forever begin

sco_data.get(tr);

sco_data_ref.get(trref);

if (tr.compare(trref))

`uvm_info ("sco", "test passed", UVM_NONE)

else

`uvm_info ("sco", "test failed", UVM_NONE)

end

endtask

endclass

MAY 2019

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Agent

class agent extends uvm_agent;
drv d;
uvm_sequencer #(transaction) seqr;
mon m;
ref_model . mref;
virtual function void build_phase (uvm_phase phase);
super. build_phase (phase).
d = drv :: type_id :: create ("d", this);
m = mon :: type_id :: create ("m", this);
mref = ref_model :: type_id : create ("mref", this);
seqr = uvm_sequencer #(transaction) :type id : create ("seqr")
endfunction

virtual function void .connect_phase (uvm_phase phase);

super_connect_phase (phase);
d.seq_item_port. connect (seqr. seq_item_export);
endfunction
endclass

Environment

class env extends uvm_env;
agent a;
sb S;